Supporting Efficient Synchronization in Multi-core NoCs Using Dynamic Buffer Allocation Technique

Xiaowen Chen[†,‡], Zhonghai Lu[‡], Axel Jantsch[‡] and Shuming Chen[†]
[†]National University of Defense Technology, 410073, Changsha, China
[‡]KTH-Royal Institute of Technology, 16440 Kista, Stockholm, Sweden
[†]{xwchen,smchen}@nudt.edu.cn [‡]{xiaowenc,zhonghai,axel}@kth.se

***Abstract*—This paper explores a dynamic buffer allocation technique to guide a distributed synchronization architecture to support efficient synchronization on multi-core Network-on-Chips (NoCs). The synchronization architecture features two physical buffers to be able to concurrently queue and handle synchronization requests issued by the local processor and remote processors via the on-chip network. Using the dynamic buffer allocation technique, the two physical buffers are dynamically allocated to form multiple virtual buffers in order to improve buffers' utilization. Experiments are carried on to evaluate buffers' utilization.**

I. Introduction and Related Work

Exploiting efficient parallelism and hence achieving high performance of parallelized applications running on multi-core Network-on-Chips (NoCs) require high bandwidth memory subsystem, as well as efficient synchronization mechanisms. In this paper, we explore a dynamic buffer allocation technique to guide a distributed synchronization architecture, named *Synchronization Handler (SH)*, for efficient synchronization mechanism, targeting multi-core NoCs. Traditional synchronization mechanisms such as *spin lock*, which are based on the *busy-wait* techniques to ensure mutual exclusion. The SH features two physical buffers to store and manage, locally, synchronization requests issued by the local processor and remote processors via the on-chip network, avoiding network traffic needed by polling a shared synchronization variable. In order to improve the two physical buffers' utilization, the dynamic allocation technique is proposed and the two physical buffers are dynamically allocated and logically organized as multiple virtual buffers. Regarding efficient synchronization mechanism, in [1], Sampson presented a novel mechanism for barrier synchronization on chip multiprocessors (CMPs). His contribution is on barrier synchronization rather than lock/semaphore synchronization. In [2], Monchier explored an optimization technique of synchronization mechanisms for shared memory MPSoCs based on NoC. A synchronization hardware module was designed to augment the memory controller to support polling a lock locally in the shared memory. However, since all synchronization requests issued by all nodes flow through the network into the synchronization module, contention latency induced by heavy traffic near the module brings negative performance. In [3], Yu proposed a synchronization architecture for embedded multiprocessors to effectively implement the queued-lock semantics in a completely distributed way. In their multiprocessor platform, each processing node hosts a synchronization controller. However, 1) their multiprocessor platform uses the bus rather than complex interconnect (e.g. packet-switched network), and 2) the announcement of remote lock acquire request in the bus induces much traffic and is lack of scalability.

II. Multi-core Network-on-Chips with Distributed Synchronization Architectures

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 2D-mesh. As shown in Fig. 1 b), each PM node contains a processor, a local memory, a Network Interface (NI) and a Synchronization Handler (SH). The SH contains a set of synchronization variables and two physical buffers to provide efficient synchronization support for mutual exclusion. The SH is connected to the processor and the Network Interface (NI) and serves synchronization requests from the local processor and remote processors via the on-chip network concurrently.

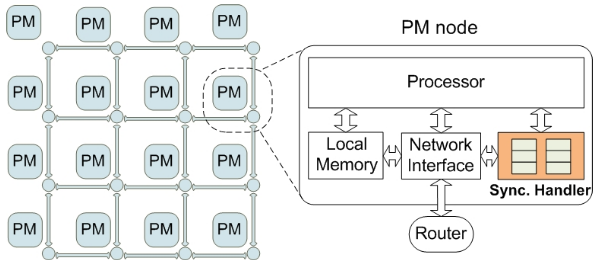

Fig. 1. a) A 16-node mesh multi-core NoC, b) Processor-Memory node

Two physical buffers in the SH permit responding to two simultaneous synchronization requests from both the local processor and remote processors via the on-chip network. Each physical buffer owns a buffer queue with the depth of **Q** and can receive synchronization requests from either the local processor or the network. The physical buffers store incoming requests and respond to the request which meets the synchronization requirement. To improve the two physical buffers' utilization, the two physical buffers are dynamically allocated according to coming synchronization requests and logically organized as multiple virtual buffers (equals the number of locks). Each virtual buffer is coupled with a lock. Under this organization, synchronization requests accessing different locks go through their independent virtual buffers and do not interfere with one another.

III. Dynamic Buffer Allocation Technique

The SH can respond to synchronization requests from the local processor and remote processors via the on-chip network concurrently. The two physical buffers are dynamically allocated and logically organized as multiple virtual buffers. As shown in Fig. 2 a), synchronization requests come from the local processor and the network, go through the corresponding virtual buffers and finally enter the Synchronization Variable Pool, which provides a set of synchronization variables. The requests are differentiated into two types: *Lock Acquire* and *Lock Release*. The "*Lock Release*" request always goes along the bypass path, while the "*Lock Acquire*" request also can go along the bypass path only when the related virtual buffer is empty. That is to say, the virtual buffers are only used for the "*Lock Acquire*" requests.

Fig. 2b) shows an example of how the two physical buffers are

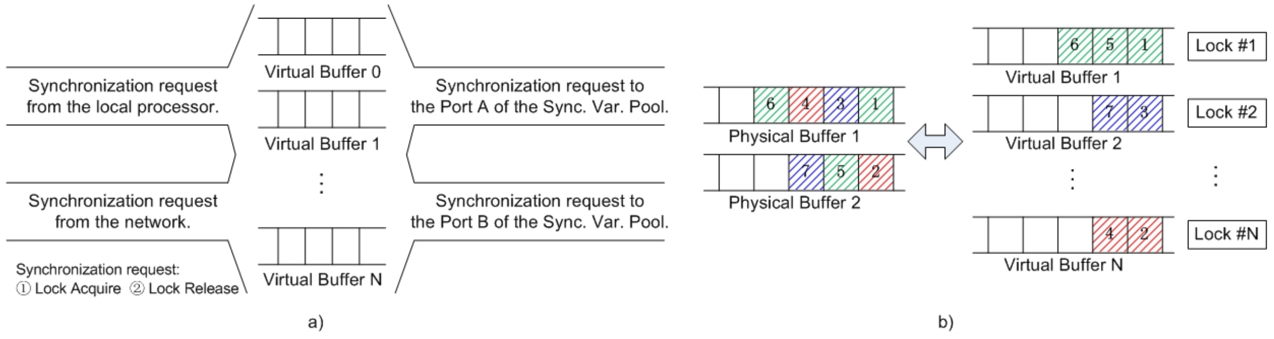
Synchronization request from the local processor.
Synchronization request from the network.
Synchronization request: ① Lock Acquire ② Lock Release
Virtual Buffer 0
Virtual Buffer 1
Virtual Buffer N
Synchronization request to the Port A of the Sync. Var. Pool.
Synchronization request to the Port B of the Sync. Var. Pool.
a)
Physical Buffer 1
Physical Buffer 2
Virtual Buffer 1
Virtual Buffer 2
Virtual Buffer N
Lock #1
Lock #2
Lock #N
b)

Fig. 2. a) Buffer's dynamic allocation and "virtual buffers" organization, and b) an example describing how the two physical buffers are allocated dynamically and organized as multiple virtual buffers

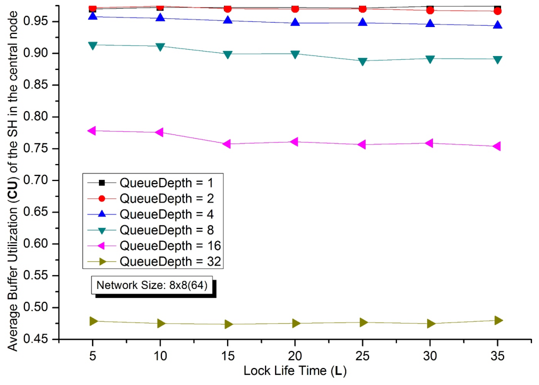

Fig. 3. average buffer utilization (**CU**) of the SH in the central node

allocated dynamically and organized as multiple virtual buffers. Assuming that the Lock #1, #2 and #N are used, there are seven "*Lock Acquire*" requests coming from the local processor or remote processors. Request 1, 5 and 6 acquire Lock #1, Request 3 and 7 acquires Lock #2, and Request 2 and 4 acquires Lock #N. Because Lock #1, #2 and #N are used, the seven requests are buffered in the physical buffers. Their locations in the physical buffers reveal the order of their arrivals. In functionality, these requests are re-ordered according to the locks they acquire. Hence, Request 1, 5 and 6 logically form Virtual Buffer 1, Request 3 and 7 logically form Virtual Buffer 2, and Request 2 and 4 logically form Virtual Buffer N.

IV. Analysis of Buffer Utilization

Since the SH's main feature is the two physical buffers and their "virtual buffer" organization, the buffer utilization is analyzed in the experiment.

- **cu(t):** Buffer utilization at cycle #t. **cu(t)** = the number of used items in the two physical buffers at cycle #t / the total item numbers in the two physical buffers (2***Q**).
- **CU:** Average buffer utilization during a simulation. **CU** = the sum of **cu(t)** / the total cycles in a simulation.

Fig. 3 shows the average buffer utilization (**CU**) of the SH in the central node versus the lock life time and the queue depth. The network size is fixed to be 8x8(64). As we can see, (i) the **CU** for larger queue depth is lower than that for smaller queue depth. For instance, the **CU** for QueueDepth=1, 2, 4, 8, 16 and 32 are more or less 97%, 96%, 94%, 90%, 76% and 47%, and (ii) the **CU** is almost a constant for each value of queue depth, no matter what the lock life time is. This is because larger lock life time leads to not only longer simulation time but also the larger sum of **cu(t)** .

Fig. 4 shows the buffer utilization (**cu(t)**) of the SH in the central node for different queue depth, versus the clock cycle. For larger queue depth, the width of the bottom of the curve is smaller and the

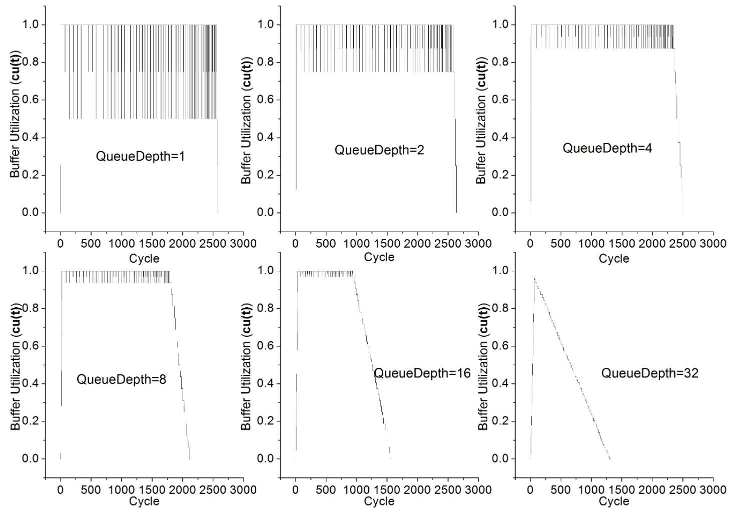

Fig. 4. buffer utilization (**cu(t)**) in the central node for different queue depth

top of the curve is narrower. The width of the bottom reflects how long the simulation runs, so the smaller the width of the bottom is, the better the performance is. The narrower top of the curve leads to the lower **CU**. For instance, for QueueDepth **Q**=1, the width of the top is almost equal to the width of the bottom. It means that the two physical buffers are always close to be full. For QueueDepth=32, the top is very sharp, meaning that the two physical buffers are only close to be full during a very small part of the entire simulation. This is why the (**CU**) for larger queue depth in Fig. 3 is lower. We can obtain that higher queue depth can cause better performance, but brings more area cost and lower (**CU**).

Acknowledgment

The research is partially supported by the FP7 EU project MOSART (No. IST-215244), the National 863 Program of China (No. 2009AA011704), the Innovative Team of High-performance Microprocessor Technology, and the National Natural Science Foundation of China (No. 60676010).

References

[1] 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.

[2] M. Monchiero, G. Palermo, C. Silvano, and O. Villa, "Power/performance hardware optimization for synchronization intensive applications in mpsocs," in *Proc. of the Conf. on Design, automation and test in Europe (DATE'06)*, 2006, pp. 606–611.

[3] C. Yu and P. Petrov, "Distributed and low-power synchronization architecture for embedded multiprocessors," in *Proc. of the 6th IEEE/ACM/IFIP Int'l Conf. on Hardware/Software codesign and system synthesis (CODES+ISSS'08)*, 2008, pp. 73–78.