SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

June 2017, Vol. 60 062402:1–062402:12
doi: 10.1007/s11432-016-0306-y

HyBar: high efficient barrier synchronization based on a hybrid packet-circuit switching Network-on-Chip

Zhenqi WEI* , Peilin LIU & Rongdi SUN

School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240*, China*

Received August 11, 2016; accepted November 6, 2016; published online February 9, 2017

Abstract Realizing barrier synchronization in multi-/many-core processors with high efficiency becomes more and more challenging as the number of cores integrated in a single chip keeps growing. Quite a few barrier solutions have been proposed, while they provide limited improvements for synchronizing large amounts of cores or incur unfavorable restrictions on performing concurrent barriers. This paper presents HyBar, a hardware barrier based on a hybrid switching NoC which adopts packet switching and circuit switching methods in two sub-networks respectively. Dedicated channels in the circuit-switching sub-network are dynamically built and removed when barrier requests traverse the packet-switching sub-network according to a modified dimension-order routing algorithm. The efficiency of inter-core communication for concurrent barriers is improved by merging barrier arrival requests and broadcasting release requests along the circuit channels. The execution time of synthetic cases, benchmark kernels and parallel applications using various barrier solutions are evaluated in an RTL-based simulation platform. Experimental results show that our proposal provides about 15%–50% performance improvement compared to previous solutions, while the hardware overhead is marginal under SMIC 40 nm technology. Moreover, HyBar introduces a minor efficiency loss for concurrent barriers with no limitation on their layouts of participating cores in the on-chip network.

Keywords NoC, barrier synchronization, packet-circuit switching, concurrent barriers, routing algorithm

Citation Wei Z Q, Liu P L, Sun R D. HyBar: high efficient barrier synchronization based on a hybrid packet-circuit switching Network-on-Chip. Sci China Inf Sci, 2017, 60(6): 062402, doi: 10.1007/s11432-016-0306-y

1 Introduction

As one of the mostly used synchronization methods in parallel programming on chip multi-processors (CMP), barrier [1] ensures the correctness of multithreaded applications by making fast threads wait for slow ones until all threads reach the same barrier point. According to the evaluation given by [2], the overall performance of simultaneous multithreading seriously degrades due to the overhead of barrier processes as the granularity of parallelism decreases. Hence realizing barrier synchronization efficiently on Network-on-Chip (NoC) based array processors [3] which integrate tens of hundreds of processing cores becomes more and more challenging.

* Corresponding author (email: awaylovemusic@sjtu.edu.cn)

Software-based barriers are widely used because of their simplicity in implementation, while both traffic and latency for inter-core communication scale with the number of participants [4], which results in inefficiency of synchronizing tens of hundreds of on-chip cores. Previously proposed solutions apply dedicated hardware to improve barrier performance, while they have some drawbacks such as lack of scalability for barrier participants or limited support for concurrent barriers. In this paper we propose HyBar, a hardware barrier running in a hybrid packet-circuit switching on-chip network. We use a circuit-switching (CS) sub-network (Cnet) on top of a packet-switching (PS) sub-network (Pnet) to improve the execution efficiency of barrier synchronization. Circuit channels between participating cores are dynamically built and removed in the Cnet, where barrier requests are merged and broadcasted during concurrent barriers to reduce traffic and latency of inter-core communication. With manual configuration on circuit connections, the Cnet can also be used for normal data transmission. The high efficiency of HyBar is gained as a result of threefold contributions.

(1) A low-latency Cnet which supports merge and multicast of barrier requests among participating cores.

(2) A dynamic setup and removal scheme of Cnet channels with no interlock when multiple barriers are concurrently performed.

(3) A modified routing algorithm used in the Pnet to build circuit channels which can be fully leveraged by barrier requests.

The rest of the paper is organized as follows. Section 2 describes background and related works. Section 3 gives the design details of HyBar. Section 4 presents HyBar's hardware cost and performance evaluation. Finally, our conclusion is drawn in Section 5.

2 Background & related work

According to [4], barriers can be categorized into four models, which are depicted in Figure 1. The master-slave (MS) model shown in Figure 1(a) applies a centralized approach where a barrier core (master) is responsible to suspend other cores (slaves) in arrival phase and then to release them in departure phase after they have all arrived at the barrier. The centralized communication may lead to severe congestion around the master core. The tree (Tr) model shown in Figure 1(b) collects and duplicates barrier requests based on a tree construction during the two phases, which reduces part of on-chip traffic within the MS model. The butterfly (Bf) model shown in Figure 1(c) synchronizes pairs of participants successively, while the all-to-all (A2A) model shown in Figure 1(d) makes each participant send its arrival messages to all the rest ones. Both models apply distributed methods to resolve the congestion issue brought by centralized communication, while their on-chip traffic increases sharply with the number of participants.

Various hardware barrier solutions based on different models are presented in former literatures. Monchiero et al. [5] proposed a synchronization-operation buffer (SB) in the barrier core of MS barrier to reduce remote polling on shared resources. Xiao et al. [6] implemented MS barriers on a bus-based multiprocessor by exchanging messages on a full crossbar network with explicit initialization. Wei et al. [7] designed a hybrid Tr-A2A barrier (TAB) supporting exchange of merged arrival requests among at most eight cores, which can reduce execution time for synchronizing small amounts of cores. Chen et al. [8] applied Cooperative Communication (CC) technique to merge barrier requests in a packet-switching network. In [9], they proposed an A2A barrier which broadcasts barrier acquire packets between all on-chip cores whether they participate in a synchronization process or not. However, their solutions do not support concurrent barriers. Abellan et al. [10] built a cluster-based network to allow one barrier within each cluster, and TLSync [11] removes this constraint by allocating various frequencies for each barrier groups in a transmission-line network. Sartori et al. [2] used express virtual channel (EVC) [12] to make synchronization requests bypass intermediate routers. Krishna et al. [13] proposed a single-cycle reconfigurable NoC with all crossbar select lines preset before applications run on the NoC. By placing the sorted addresses of multiple destinations in the header of a message, Daneshtalab et al. [14] presented a path-based routing algorithm for multicast communication including barrier synchronization in 2D mesh

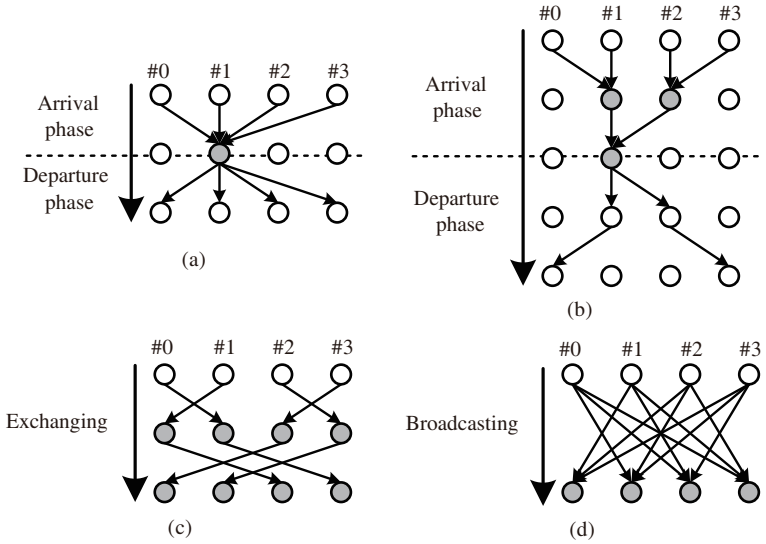

Figure 1 The four models of barrier synchronization. (a) MS barrier; (b) Tr barrier; (c); Bf barrier; (d) A2A barrier.

NoCs.

NoCs based on either PS or CS were widely discussed in previous works, while recently quite a few attempts have been made to combine PS and CS within a single on-chip network. Modarressi et al. [15] and Lin et al. [16] divided NoC links and switch allocators between PS and CS networks using spatial and time division multiplexing respectively to improve network efficiency. Abousamra et al. [17] split the original network into a fast plane for critical requests and a power-efficient plane for non-critical requests. Ou et al. [18] proposed a double-layer NoC to transmit scattered data in PS layer and bulk data in CS layer, where routing is determined by manual configurations in advance. Jerger et al. [19] proposed a hybrid network where data can be piggy-backed immediately behind a circuit setup request. When the setup request finds no available circuits, the least recently used circuit will be reconfigured as PS for the following data flits, which may stop or alter other CS transfers. In [20], circuit channels which are built by reservation packets cannot be used until acknowledgement signals of complete channels are returned to source cores.

Based on the two-phase model of MS barrier with simple configuration on synchronization control registers, HyBar applies a modified dimension-order routing (DOR) algorithm in the Pnet to build complete or incomplete circuit channels for concurrent barriers in the low-latency Cnet, where barrier requests can be merged or broadcasted between participating cores like Tr barrier. Thus HyBar has advantages on performance improvement of barrier synchronization and on support of concurrent barriers over previous solutions.

3 Design of Hybar

We consider an NoC-based many-core processor with 2D mesh-grid topology as our baseline processor, which is shown in Figure 2(a). As Figure 2(b) depicts, a synchronization controller (SC) integrated in each core is used to handle barrier synchronization requests. Pnet flits and Cnet messages are transmitted respectively in the two separated sub-networks through pairs of unidirectional links, which is presented in Figure 2(c). As shown in Figure 2(d), five-ported crossbars (C in Figure 2(a)) forward Cnet messages according to port connections which are dynamically configured during the transmission of Pnet flits through routers (R in Figure 2(a)).

3.1 Cnet crossbar & barrier requests

As shown in Figure 3, along with Cnet messages, flits inputted to and outputted from a Pnet router are

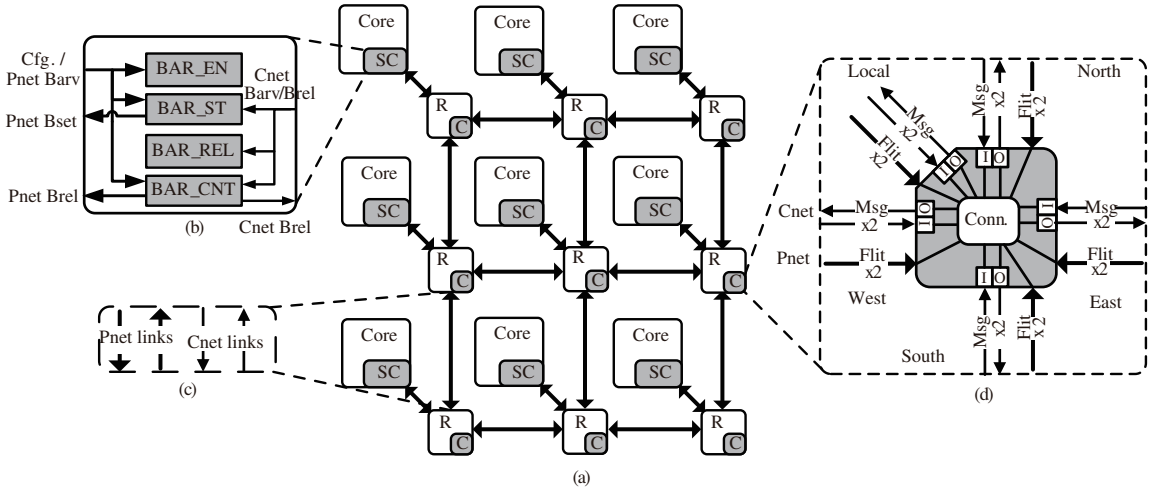

Figure 2 Architecture of an NoC processor adopting HyBar. (a) 2D mesh NoC; (b) synchronization controller; (c) inter-core links; (d) Cnet crossbar.

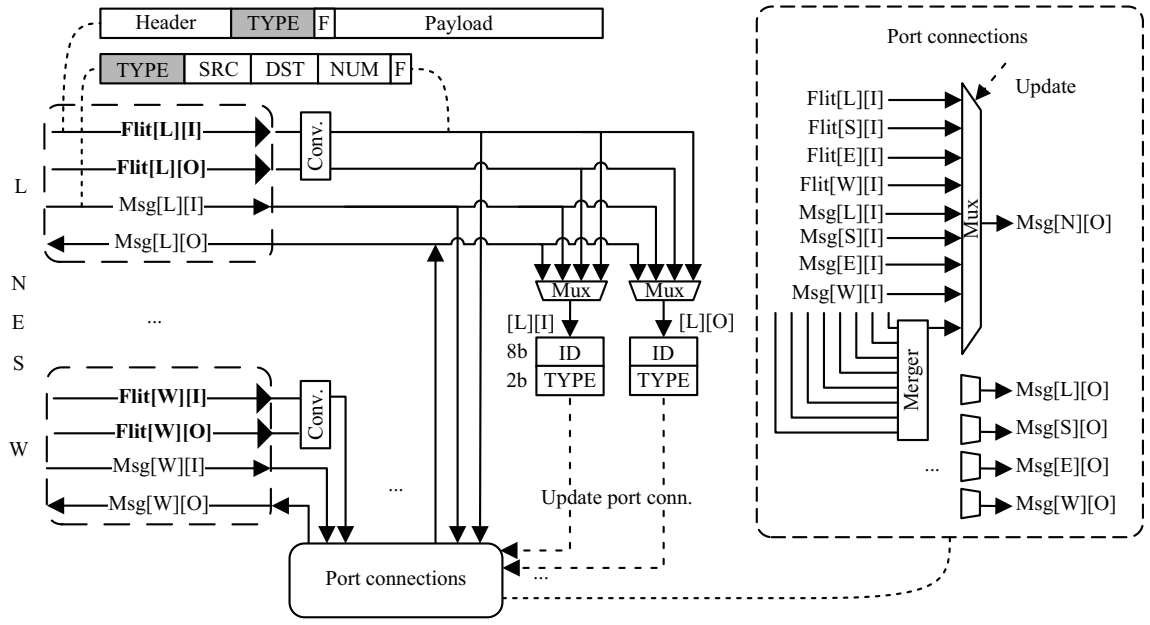

Figure 3 Cnet crossbar.

both fed to the coupled crossbar and are then converted to message format to build or remove circuit channels by configuring 8-bit ID and 2-bit TYPE control registers for each crossbar port. Besides a TYPE field telling the type of barrier requests listed in Table 1, SRC and DST fields reveal the source and destination core IDs of a Cnet message respectively. With various core IDs in DST fields, barrier setup requests (Bset) sent by the barrier core are used to build Cnet channels for participants from the farthest to the nearest. Once finding an available circuit channel, a barrier arrival request (Barv) which initially travels in the Pnet will enter the Cnet to continue its transmission with a delay of one cycle per hop. When passing through a crossbar simultaneously, multiple Barvs with same DST can merge into a single Cnet message whose NUM field stores the total number of merged participants. However only the ID of the farthest participant is stored in the SRC field of the merged message. After receiving all Barvs, the barrier core sends in the Cnet a release message (Brel) with its core ID being saved in both SRC and DST fields.

Figure 4 shows the configuration of control registers for crossbar's input port [X][I], where X can be any one of the five sides including L (local), N (north), E (east), S (south) and W (west). While a similar

Table 1 Types of Pnet flits and Cnet messages

Type	Pnet flits	Cnet messages
Bset	Build Cnet channels for Barvs	N/A
Barv	Inform the barrier core of participants' arrivals, and build Cnet channels for Brels	
Brel	Release participants and remove Cnet channels	
Mcfg	Manually configure crossbar ports	N/A

```
1: if ((Flit[X][O][Type] == Bset) && (TYPE[X][I] invalid)) then          // Bset flit builds channels for Barv msg
2:       ID[X][I] <= Flit[X][O][SRC];
3:       TYPE[X][I] <= Barv;
4: else if ((Flit[X][O][Type] == Barv) && (TYPE[X][I] invalid)) then   // Barv flit builds channels for Brel msg
5:       ID[X][I] <= Flit[X][O][DST];
6:       TYPE[X][I] <= Brel;
7: else if ((Msg[X][O][Type] == Barv) && (TYPE[X][I] invalid)) then  // Barv msg builds channels for Brel msg
8:       ID[X][I] <= Msg[X][O][DST];
9:       TYPE[X][I] <= Brel;
10:else if (((Flit[X][I][TYPE] == Brel) && (Flit[X][I][SRC] == ID[X][I])
         && (TYPE[X][I] == Barv || Brel))                                // Brel flit releases allocated port
   || (Msg[X][I][TYPE] == Brel)) then                                    // Brel msg releases allocated port
11:      Invalidate ID[X][I] & TYPE[X][I];
13:else if (Flit[X][I][TYPE] == Mcfg) then                             // Manually configure port [X][I]
```

Figure 4 Configuration for crossbar port [X][I] (X can be L, N, E, S, or W).

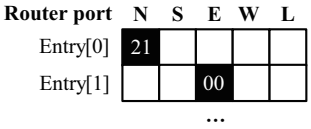

Figure 5 Entries of multicast control register MC_REG.

configuration method is applied to the corresponding output port [X][O]. Within a barrier process, Bset flits outputted from a router port are used to configure the coupled crossbar's input port on the same side to forward Barvs in the Cnet. Pnet flits and Cnet messages with type Barv allocate crossbar ports for Brels in the same way. When receiving Brel flits or messages, the allocated crossbar ports are released and thus circuit channels are removed from the Cnet. Besides, normal requests can also transmit in the Cnet after crossbar ports have been configured manually by Pnet flits with type Mcfg in advance.

Once barrier requests fail to configure crossbar ports already allocated by other cores, the 1-bit flags within the requests (see F field in Figure 3) will be turned on to invalidate their configuration on other crossbar ports later. Due to the competition for Cnet links between concurrent barriers or normal communications, circuit channels ending at destination cores for Barvs or Brels may start from crossbars of intermediate cores traversed by Pnet flits, which makes our proposal avoid interlocking when multiple channels are simultaneously being built for various inter-core communications. Consequently, in order to support duplication of Brels in the Pnet before they can transmit in the Cnet, a multicast control register MC_REG is added to each router to record input ports of Barvs which fail to build circuit channels for Brels. The number of concurrent barriers allowed in HyBar is limited by the number of entries within MC_REG, which is depicted in Figure 5.

3.2 Barrier synchronization using HyBar

Generally, the middlemost one of all participating cores is chosen to be the barrier core, and the choice is sent to other participants before a barrier starts. Figure 6 depicts the update of SC's control registers shown in Table 2 during barrier synchronization based on HyBar's two-phase model. The arrival phase starts after enable register BAR_EN is set, and then the barrier core sends Bsets in the Pnet to participants according to corresponding bits of barrier status register BAR_ST. Once reaching the barrier, participants issue Barvs to the barrier core along previously built Cnet channels and then locally poll on release flags BAR_REL. Being initialized with the number of cores to be synchronized, barrier counter BAR_CNT

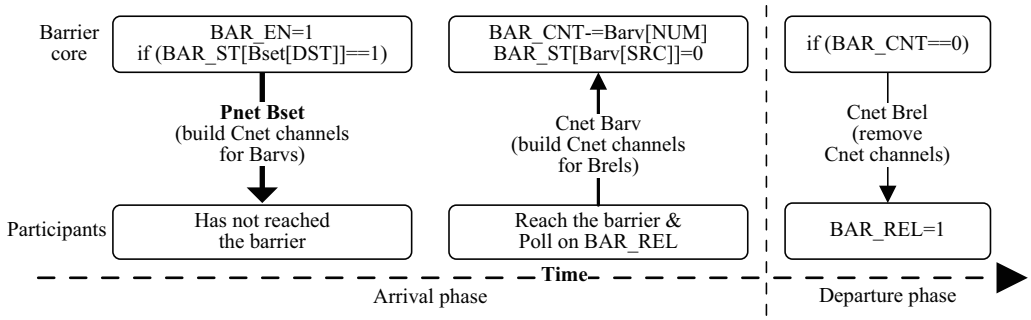

Figure 6 Two-phase model of HyBar.

Table 2 Barrier control registers in SC

Register	Description
BAR_EN	1-bit enable register to start a barrier process
BAR_ST	Barrier status register with 1 in bit [i] meaning that participating core i has not been sent a Bset to nor has arrived at the barrier
BAR_CNT	Barrier counter recording the number of unreached participants
BAR_REL	1-bit flag to indicate the release of a participant

Figure 7 Allowed and prohibited turns of the XY-YX routing.

in the barrier core is decreased by NUM fields of received Barvs. When the counter becomes zero, the barrier process enters the departure phase, and a Brel is broadcasted in the Cnet to all participants to turn on their BAR_RELs and to remove Cnet channels.

Based on widely used DOR algorithms such as XY routing, routes between two on-chip cores starting from either one are usually different. Therefore, participants' Barvs may not find circuit channels built by Bsets issued from the barrier core, nor Brels find channels built by Barvs. In the Pnet, we apply XY-YX routing, a simple variant of the XY routing, for Barvs and Brels to fully leverage the low-latency Cnet. The XY-YX routing is like the XY routing when a destination lies to the west of current core, and behaves as the YX routing if the destination is to the east. Thus the routing paths between the barrier core and a participant in forward and backward directions are identical, which helps barrier requests to transmit in the Cnet as soon as possible. Like the deadlock-free XY routing [21], the XY-YX routing also allows four turns to prevent deadlock, which is shown in Figure 7.

In Figure 8, we demonstrate two concurrent barriers running in a 3×3 NoC, where each core is numbered with a unique ID for clarity. Crossbar's port connections along with the configuration of MC_REG for each core are also revealed in the figure. Among the nine on-chip cores, core 00 is the only participant of one barrier core 11, while another barrier core 21 synchronizes the rest six cores.

As shown in Figure 8(a), during the transmission of a Pnet Bset (see ⓐ) issued from the barrier core 11, its ID is saved in the crossbar ports of traversed cores to build a Cnet channel between cores 00 and 11 to transmit Barvs with core ID 11 in their DST fields. Core 11's barrier counter BAR_CNT is decreased by one since the barrier core receives a local Barv. In Figure 8(b), core 00 issues a Cnet Barv (see ⓑ) in the previously built channel heading to core 11, while a backward channel for core 11's Brel is also set up. When the BAR_CNT becomes zero, a Cnet Brel (see ⓒ) is sent to core 00 through the backward channel, which is shown in Figure 8(c). All crossbar ports allocated to core 11 are released and thus both the forward and backward channels between cores 00 and 11 are removed from the Cnet.

For another barrier, the barrier core 21 receives Pnet Barvs (see ① and ②) from cores 20, 22 and itself, and sends out a Pnet Bset (see ③) to core 02, which is shown in Figure 8(a). Since the 01-11

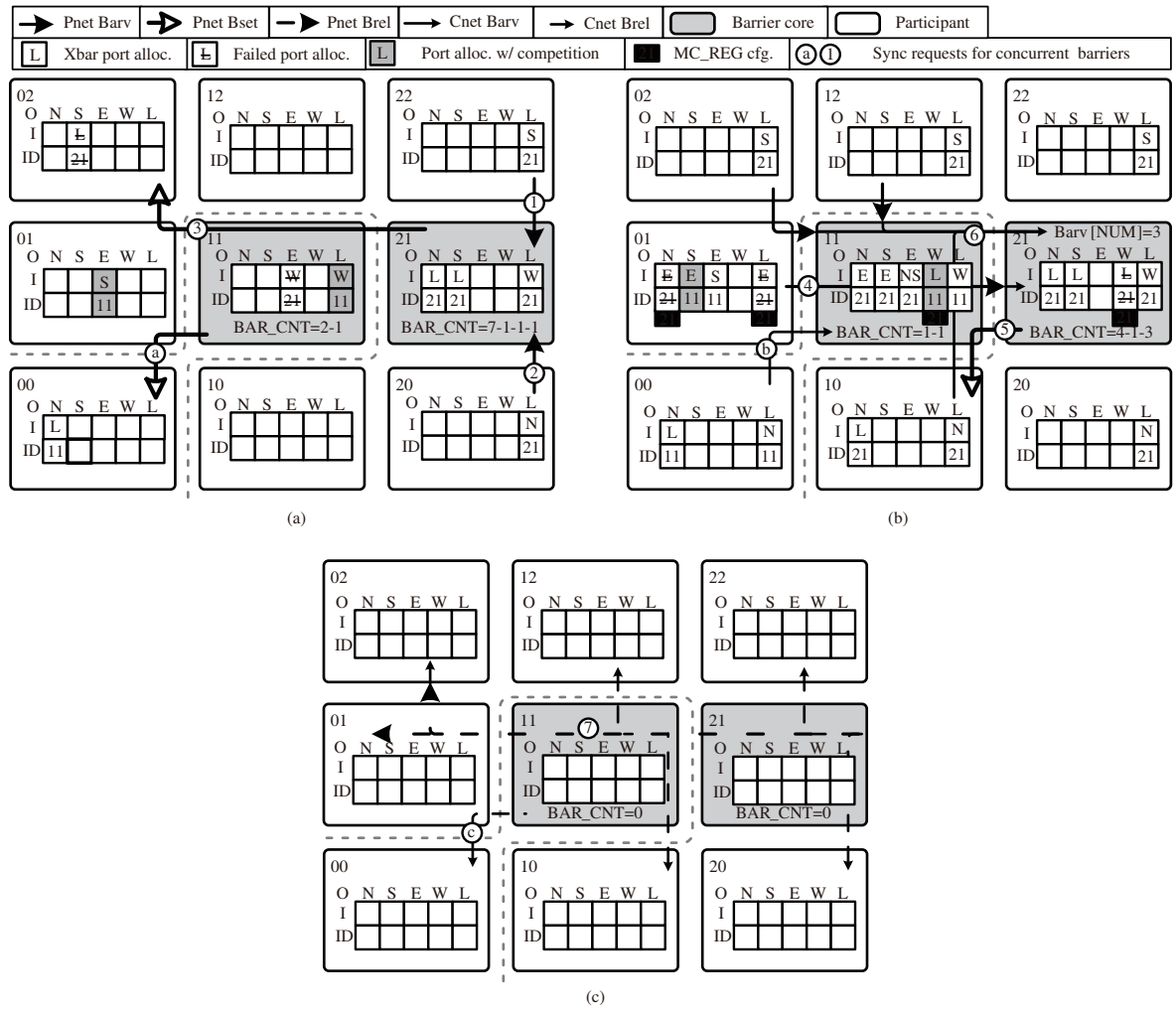

Figure 8 Concurrent barriers performed in a 3×3 NoC using HyBar. (a) Two barrier cores 11 & 21 issue Pnet Bsets, and barrier core 21 receives Pnet Barvs from cores 20 & 22; (b) barrier core 11 receives core 00's Cnet Barv, and barrier core 21 collects Barvs from Cnet; (c) barrier core 11 sends a Cnet Brel to core 00, and barrier core 21 broadcasts Brels in Pnet & Cnet.

links in the Cnet have already been occupied by a different barrier, crossbar ports of cores 11, 01 and 02 cannot be allocated to core 21. In Figure 8(b), a Pnet Barv (see ④) issued by core 01 fails to build a Cnet channel for Brels due to the competition of crossbar ports between concurrent barriers. As a result, MC_REG registers of cores 01, 11 and 21 are configured to perform multicast of Brels in the Pnet. The barrier core 21 sends another Bset (see ⑤) to core 10, which makes a Barv issued from the latter merge with other Barvs (see ⑥) at core 11. Once the barrier core finishes collecting Barvs, a Brel (see ⑦) is duplicated and transmitted in both the Pnet and the Cnet , which is depicted in Figure 8(c). Meanwhile all allocated crossbar ports and configured MC_REGs are reset.

4 Experimental results

HyBar is integrated into a 2D-mesh NoC processor, which is composed of 32-bit RISC cores with five-stage pipeline and two-stage routers [22] with minor modification to support the XY-YX routing. After synthesizing the NoC processor in SMIC 40 nm technology at a working frequency of 400 MHz for the both sub-networks, we get circuit area and average runtime power consumption of HyBar during barrier synchronization. The SC takes 0.6% area (4024 μm^2) and 1.1% power (0.09 mW) of a single on-chip node consisting of a core and a router with an eight-entry MC_REG, while the Cnet crossbar consumes

Table 3 Comparison of barrier solutions

Barrier solution	CC barrier [8]	TAB barrier [7]	HW-A2A barrier [4]	SB barrier [5]	Dedicated barrier	HyBar
Barrier model	MS	Tr-A2A	A2A	MS	MS	MS
NoC switching	PS	PS	PS	PS	CS	PS-CS
Routing algorithm	XY	XY	Static	XY	N/A	XY-YX

2.7% area (18950 μm^2) and 3.2% power (0.27 mW) per node. We also implement the SB barrier [5] with a 16-entry buffer and the CC barrier [8] for comparison, and the two barrier solutions based on PS NoCs cost additional area/power of 8818 μm^2/0.20 mW and 10514 μm^2/0.24 mW per node respectively. While the hardware overhead of HyBar is mainly introduced by the Cnet, which is used for not only synchronization requests but also normal data transmissions.

We build an RTL-based NoC processor simulator to evaluate the performance of HyBar and other barrier solutions in synthetic experiments with inter-core communications of barrier requests only and in real applications with both synchronization and normal traffics.

4.1 Synthetic experiments

In synthetic experiments, various numbers of cores (ranging from 4 to 256) participate in barrier processes with a random delay of maximal 0, 50 and 500 cycles per core before issuing arrival requests to simulate zero-, small-, and medium-delay cases respectively, which reveal different extents of workload imbalance between participants in practical situations. After multiple runs with different random seeds, we acquire average duration time of barriers in cycles, which is counted from the first arrival request being issued until the last participant receiving release signal. The performance of HyBar is compared to other PS-based barriers, such as the CC barrier [8], the TAB barrier [7], the HW-A2A barrier [4], and the SB barrier [5]. We also evaluate the efficiency of a dedicated MS barrier running in a CS network with manual configurations on crossbar connections via a PS network, which is similar to [15] and [18]. Besides, HyBar adopting 1-cycle routers instead of pipelined routers is also added to the comparison to evaluate the effect of router's latency on barrier performance. Table 3 shows comparison of barrier models, NoC switching methods and routing algorithms adopted by various barrier solutions.

In zero-delay cases, barrier arrival requests issued from participants with similar distances to the barrier core may reach same intermediate routers simultaneously and thus result in NoC congestions. As Figure 9(a) shows, with merge and multicast of barrier requests, HyBar outperforms the dedicated barrier by 50.1% on average. Compared to the CC barrier and the TAB barrier which also adopt the merge/multicast technique in the Pnet with delay of flit processing in routers, HyBar still saves over 28% time by transmitting barrier requests in the Cnet. HyBar with no merging consumes 30.1% more cycles to collect Barvs. Adopting 1-cycle routers instead of pipelined routers further helps HyBar to save about two-third cycles. The HW-A2A barrier and the SB barrier, which has sufficient entries in the buffer to avoid its full condition, have similar performance to other barriers when synchronizing a small number of cores. However, their efficiencies are seriously degraded for cases with more than 64 participants, which generate large amounts of on-chip traffic and thus result in severe traffic congestions in NoCs.

For small-delay cases, the occurrence of merging Barvs in HyBar becomes less possible than it in the zero-delay cases, and the merging technique saves 22.8% time for HyBar, which is shown in Figure 9(b). Many Cnet channels have been built by Bsets before participants issue Barvs. Because of the reduction of Pnet traffic and thus less network congestions, HyBars using pipelined and 1-cycle routers both exhibit stable performances with a much less significant efficiency gap of 5.4%. Therefore, HyBar even needs fewer cycles to synchronize 128 or 256 participants with a delay of up to 50 cycles than it for the zero-delay cases. Moreover, HyBar cuts down 20%–45% cycles of the dedicated barrier, the CC barrier and the TAB barrier. As shown in Figure 9(c), performances of various barriers for medium-delay cases are quite close, because the duration of barrier synchronization is dominated by the delay time. However, HyBar still achieves an over 5% improvement compared to other barriers when synchronizing 256 cores. Above all, HyBar requires less time to synchronize all on-chip cores of an NoC processor within a single

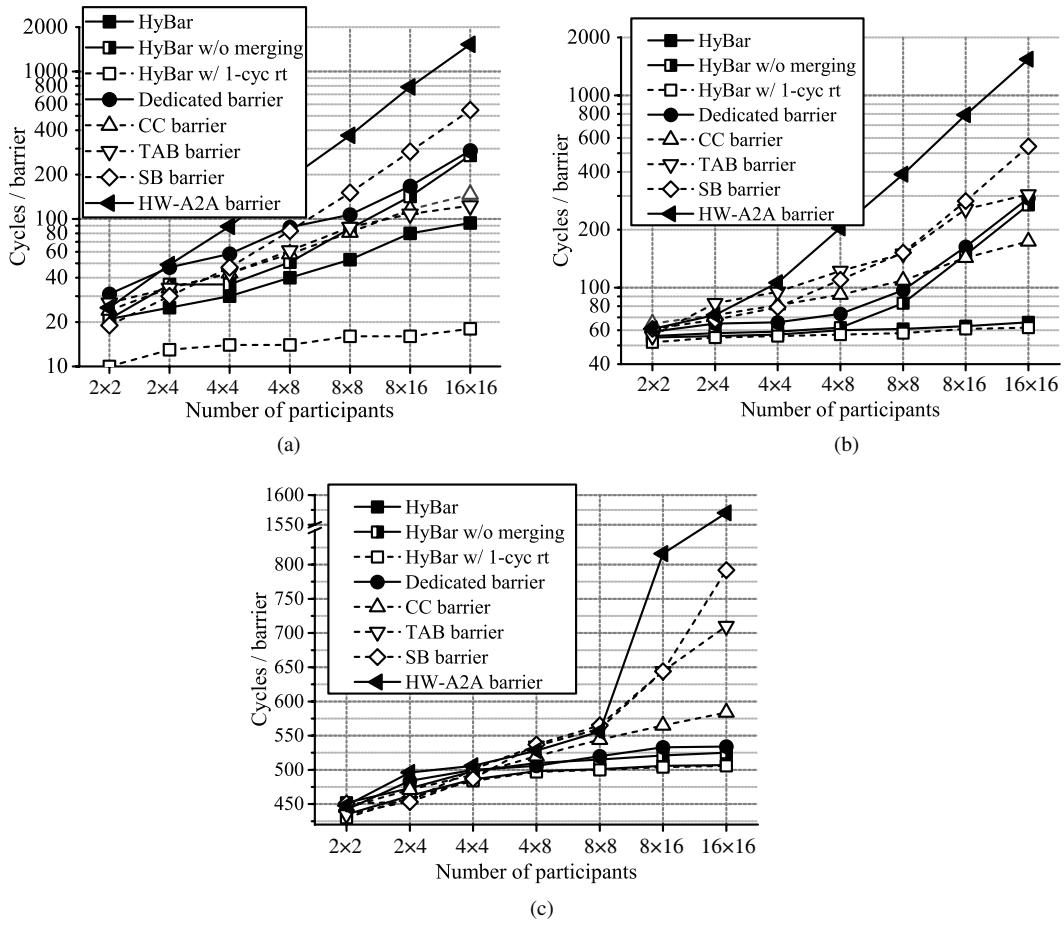

Figure 9 Average duration of barriers (b/w the 1st issued barrier request & the last received release signal) in synthetic cases. (a) Max. delay = 0 cycle; (b) Max. delay = 50 cycles; (c) Max. delay = 500 cycles.

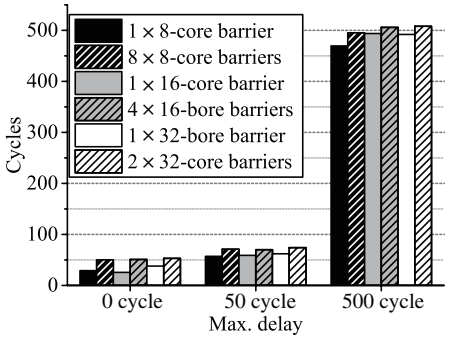

Figure 10 Performance of concurrent barriers in an 8×8 NoC.

barrier than previous solutions, and the performance improvement gained from HyBar increases with the number of participants.

To evaluate the performance loss of concurrent barriers in HyBar due to the incompletion of Cnet channels, we simulate at most eight barriers with 8, 16 or 32 participants each in an 8×8 network. The participants of concurrent barriers are randomly placed in the network. A maximal delay time of 0, 50 or 500 cycles is also considered in the simulation. The performance loss given in Figure 10 drops from 17.9% per barrier for zero-delay cases to 5.8% and 1.0% for small- and medium-delay cases respectively, which indicates the incompleteness of Cnet channels has less impact on barrier performance as the delay time increases. Different from [10] which allows only one barrier per cluster, HyBar has no limitation on the layout of concurrent barriers, and can achieve zero efficiency loss just like [11] if participants of

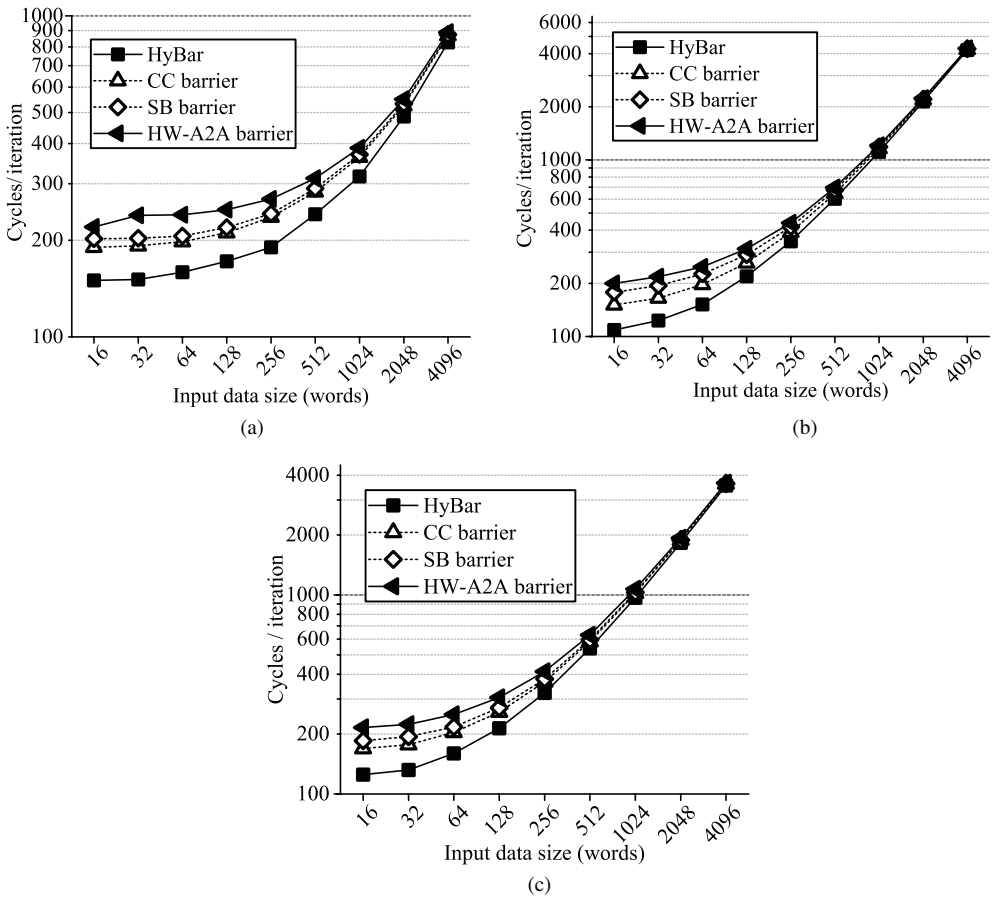

Figure 11 Performance of Livermore kernels using various barrier solutions. (a) Kernel 2; (b) Kernel 3; (c) Kernel 6.

different barriers are placed with no overlap of their routing paths.

4.2 Livermore kernels & parallel applications

Livermore Kernels [23] have long been used to evaluate the performance of compilers and architectures for parallel systems, and Kernels 2, 3 and 6 of the Livermore suite are often chosen to test the efficiency and scalability of barrier synchronization in previous literatures [2, 6, 11]. Kernel 2 is an excerpt from an incomplete Cholesky conjugate gradient code, Kernel 3 is a simple inner product, and Kernel 6 is a general linear recurrence equation. The three kernels with input data size ranging from 16 to 4096 words are run on 16 cores using HyBar and three hardware barriers. As shown in Figure 11, HyBar outperforms the competitors for all three kernels with various input data sizes. As the input size doubles, the performance improvement of HyBar over other hardware barriers becomes less significant since synchronization cost takes a smaller part of total execution time. HyBar saves on average 12.8%, 14.4% and 13.6% time of the CC barrier for Kernels 2, 3 and 6 respectively, while the speedups over the SB barrier and the HW-A2A barrier turn out to be more prominent (15%–24%) for the three kernels.

In order to compare the performance improvement of parallel applications adopting various barrier solutions, we distribute the workloads of 512-point 1D-FFT and 64×64-sized matrix multiplication (MMul) on 4–64 cores respectively. The speedups of multi-core parallelization over single-core execution are shown in Figure 12 with total consumed cycle presented on the bars. As shown in Figure 12(a), the 64-core FFT application which performs one barrier process per stage achieves an over 40x speedup using HyBar compared to the single-core version, while the CC barrier provides a 37x improvement. For parallelized MMul with one barrier in all, HyBar and the CC barrier both show good performance scalability with the number of participants and realize speedups of 52x and 49x respectively in 64-core cases, which is shown in Figure 12(b). Parallelization using different barrier solutions presents similar performances when the

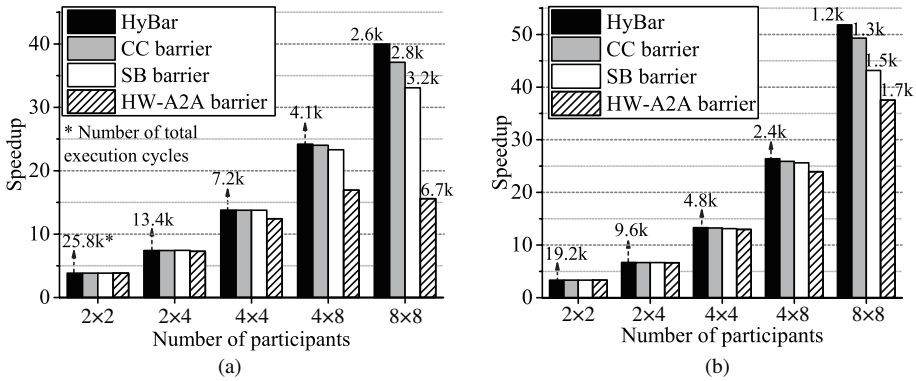

Figure 12 Performance speedups of parallel applications. (a) 512-point 1D-FFT; (b) 64×64 MMul.

number of participating cores is relatively small, since most time costs is consumed by computation tasks. More participants with less workload per core make synchronization processes occupy a larger fraction of total time. Due to the excessive traffic overhead of barrier requests in the SB and the HW-A2A barriers, the efficiency difference between them and other barriers becomes increasingly obvious in 64-core cases of both applications. In spite of non-barrier traffic in NoC and computation tasks of participants, HyBar still has notable performance advantages over previous solutions for NoC processors to perform real applications.

5 Conclusion

In this paper we propose HyBar, a hardware barrier based on a hybrid packet-circuit switching network for NoC processors. During the arrival phase of two-phase HyBar model, barrier requests initially transmit in the Pnet according to the deadlock-free XY-YX routing algorithm, which makes forward and backward routing paths between the barrier core and any one of the participating cores identical with each other. By configuring control registers of crossbar ports, setup requests and arrival requests within a barrier process dynamically build circuit channels dedicated for merge of arrival requests and multicast of release requests respectively in the Cnet with low transmission latency. In the departure phase, allocated crossbar ports are released when barrier release requests are broadcasted to all participants, and thus the circuit channels for the barrier process are removed from the Cnet. Having been synthesized in a 40 nm technology, HyBar brings an NoC processor low hardware overhead generated mainly by the Cnet, which can be used for not only synchronization requests but also normal inter-core communication. Evaluation results gained from an RTL-based NoC simulator show that HyBar outperforms previously proposed barrier solutions in both synthetic cases and parallel applications. Besides, HyBar introduces a minor efficiency loss when performing concurrent barriers with no limitation on the layout of participating cores.

Acknowledgements This work was partially supported by Equipment Pre-Research Foundation of China (Grant No. 9140A08010414JW03025).

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Wilkinson B, Allen M. Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers. Upper Saddle River: Prentice Hall, 2004
2 Sartori J, Kumar R. Low-overhead, high-speed multi-core barrier synchronization. In: Proceedings of the 5th International Conference on High Performance Embedded Architectures and Compilers (HiPEAC'10), Pisa, 2010. 18–34
3 Shen X B. Evolution of MPP SoC architecture techniques. Sci China Ser F-Inf Sci, 2008, 51: 756–764
4 Villa O, Palermo G, Silvano C. Efficiency and scalability of barrier synchronization on NoC based many-core architectures. In: Proceedings of International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES'08), New York, 2008. 81–90

5 Monchiero M, Palermo G, Silvano C, et al. Efficient synchronization for embedded on-chip multiprocessors. IEEE Trans Very Large Scale Integration Syst, 2006, 14: 1049–1062
6 Xiao H, Wu N, Ge F, et al. Efficient synchronization for distributed embedded multiprocessors. IEEE Trans Very Large Scale Integration Syst, 2016, 24: 779–783
7 Wei Z Q, Liu P L, Sun R D, et al. TAB barrier: hybrid barrier synchronization for NoC-based processors. In: Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS'15), Lisbon, 2015. 409–412
8 Chen X, Lu Z, Jantsch A, et al. Cooperative communication based barrier synchronization in on-chip mesh architectures. IEICE Electron Expr, 2011, 8: 1856–1862
9 Chen X W, Lu Z, Jantsch A, et al. Cooperative communication for efficient and scalable all-to-all barrier synchronization on mesh-based many-core NoCs. IEICE Electron Expr, 2014, 11: 20140542
10 Abellan J L, Fernandez J, Acacio M E, et al. Design of a collective communication infrastructure for barrier synchronization in cluster-based nanoscale MPSoCs. In: Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE'12), Dresden, 2012. 491–496
11 Oh J, Prvulovic M, Zajic A. TLSync: support for multiple fast barriers using on-chip transmission lines. In: Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA'11), San Jose, 2011. 105–115
12 Kumar A, Peh L S, Kundu P, et al. Express virtual channels: towards the ideal interconnection fabric. In: Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07), San Diego, 2007. 150–161
13 Krishna T, Peh L S. Single-cycle collective communication over a shared network fabric. In: Proceedings of the 8th IEEE/ACM International Symposium on Networks-on-Chip (NoCS'14), Ferrara, 2014. 1–8
14 Daneshtalab M, Ebrahimi M, Mohammadi S, et al. Low-distance path-based multicast routing algorithm for network-on-chips. IET Comput Digit Tech, 2009, 3: 430–442
15 Modarressi M, Sarbazi-Azad H, Arjomand M. A hybrid packet-circuit switched on-chip network based on SDM. In: Proceedings of Conference on Design, Automation and Test in Europe (DATE'09), Nice, 2009. 566–569
16 Lin J, Zhou W, Yu Z, et al. A hybrid router combining circuit switching and packet switching with virtual channels for on-chip networks. In: Proceedings of the 10th IEEE International Conference on ASIC (ASICON'13), Shenzhen, 2013. 1–4
17 Abousamra A K, Melhem R G, Jones A K. Déjà Vu switching for multiplane NoCs. In: Proeedings of the 6th IEEE/ACM International Symposium on Networks on Chip (NoCS'12), Copenhagen, 2012. 11–18
18 Ou P, Zhang J, Quan H, et al. A 65nm 39 GOPS/W 24-core processor with 11 Tb/s/W packet-controlled circuit-switched doublelayer network-on-chip and heterogeneous execution array. In: Proceedings of IEEE International Solid-State Circuits Conference (ISSCC'13), San Francisco, 2013. 56–57
19 Jerger N D E, Peh L S, Lipasti M H. Circuit-switched coherence. In: Proceedings of the 2nd IEEE/ACM International Symposium on Networks-on-Chip (NoCS'08), Newcastle upon Tyne, 2008. 193–202
20 Chen G, Anders M A, Kaul H, et al. A 340 mV-to-0.9V 20.2 Tb/s source-synchronous hybrid packet/circuit-switched 16×16 network-on-chip in 22nm tri-gate CMOS. In: Proceedings of IEEE International Solid-State Circuits Conference (ISSCC'14), San Francisco, 2014. 276–277
21 Glass C J, Ni L M. The turn model for adaptive routing. In: Proceedings of the 19th Annual International Symposium on Computer Architecture (ISCA'92). New York: ACM, 1992. 278–287
22 Becker D U. Efficient microarchitecture for network-on-chip routers. Dissertation for Ph.D. Degree. Palo Alto: Stanford University, 2012
23 McMahon F H. Livermore Fortran Kernels: a Computer Test of Numerical Performance Range. Technical Report UCRL-53745. 1986