Remote Control: A Simple Deadlock Avoidance Scheme for Modular Systems-on-Chip

Pritam Majumder, Sungkeun Kim, Jiayi Huang, *Student Member, IEEE,* Ki Hwan Yum, *Member, IEEE,* Eun Jung Kim, *Member, IEEE*

Abstract— Ever increasing performance demand and shrinking in the transistor size together result in complex and dense packing in large chips. That motivates designers to opt for many small specialized hardware modules in a chip to extract maximum performance benefits with relatively lower complexity and cost. These altogether opens up new directions for heterogeneous modular System-on-Chip (SoC) research, where a large system is built by assembling small independently designed chiplets (small chips). We focus on the communication aspect of such SoCs, especially newly observed deadlock among chiplets. Even though deadlock is a classic problem in networks and many solutions are available, the modular SoC design demands customized solutions that preserves the design flexibility for chiplet designers. We propose *Remote Control (RC)*, a simple routing oblivious deadlock avoidance scheme based on selective injection-control mechanism. Along with guarantee on deadlock freedom, *RC* aims to provide a methodology to make each independently designed chiplet seamlessly integrate in any modular SoCs. We achieve up to 56.34% throughput and 15.49% zero load latency improvements on synthetic traffic and up to 20% on real workloads taken from vast range of benchmark suites, over the state-of-the-art turn restriction based technique applied in modular SoC domain.

Index Terms—network deadlock, modular SoC, boundary router, inbound and outbound traffic, injection control.

1 Introduction

With the advancements in silicon technology, Systems-on-Chip (SoCs) are becoming more complex and expensive, which motivates the designers to break the whole SoC into multiple small independent *chiplets* for reducing design cost and achieve better scalability. The modular design of SoCs using 2.5D integration technology is a total paradigm shift from the monolithic SoC design to the hierarchical SoC design [1]–[3]. It allows to design smaller independent chiplets such as CPU, GPU, and accelerators with low cost and complexity, and integrate them together on an interposer, creating heterogeneous chiplet-based architectures. The chiplet-based design also increases the usability of chiplets in different SoCs and provides flexibility for vendors to manufacture using any desired process technology. In this paper, we use modular SoCs and chiplet-based systems interchangeably.

One of the major concerns in any network-based system is deadlock due to cyclic hold-and-wait among virtual channels (VCs) [4]. Since chiplets are designed independently, their integration on an interposer brings new challenges to provide correctness validation. Connecting several deadlock-free NoCs together in a modular SoC may introduce a new kind of deadlock formed among different chiplets, as they are oblivious to each other's routing algorithm [5].

There have been many studies that address deadlock issues in conventional interconnection networks [4], [6]–[10]. Conventional deadlock avoidance techniques cannot be applied directly to modular SoCs, as they consider the whole SoC as a single network, which violates the fundamental modularity principle of the chiplet-based system design. Keeping the design modularity in mind, recently Yin et al. [5] propose Modular Turn Restriction (MTR), which imposes extra turn restrictions on the boundary routers of chiplets to avoid deadlocks in modular SoCs. MTR is easy to implement in hardware but needs changes in both chiplet routing and SoC routing. In addition, because of the skewed turn restrictions, this approach can lead to load imbalance and create several hotspots, which are detrimental for network throughput. From this work we notice that the boundary routers are playing a major role in inter-chiplet deadlocks.

We exploit two key insights regarding deadlocks in modular SoCs for providing a solution that works with any chiplet routing. First, outbound packets (going out of the chiplet) may block inbound (going inside a chiplet from outside) and intra-chiplet (source and destination in the same chiplet) packets to reach destinations. The other key observation is that packets involved in a deadlock cross the boundary of the chiplets through a set of specific boundary routers[1]. Since the chiplets and interposer have independent deadlock free routing techniques, a deadlock is not possible in each of them separately, meaning it must involve both. Based on MTR and our observations, we pin-point the reason for this newly evolved deadlock among the chiplets in the modular SoC design. So we propose *Remote Control (RC)*, which is specific and highly optimized to solve this deadlock issue with minimum cost. It is also generalized enough to be applicable to chiplets with any kind of network, and SoC with any kind of chiplets connected in any

• *All the authors are associated with the Department of Computer Science and Engineering, Texas A&M University, Texas, TX-77843. E-mail: (pritam2309, ksungkeun84, jyhuang, yum, ejkim)@tamu.edu*

Digital Object Identifier no. 10.1109/TC.2020.3029682

1. Boundary Router: Chiplet routers that are connected with interposer router.

topology with corresponding routing.

In Figure 1, we draw a real-life analogy with a road situation, where the highway is considered as part of the interposer network and city roads are considered as part of the chiplet network. Cars moving from city to highway are considered as outbound cars. The ramp in between the highway and the city roads is analogous to the existing output buffer (we named it as *rc_buffer*) in the boundary router. In Figure 1(a) it is clear that if all the outbound cars cannot be accommodated in the ramp, it may result in a deadlock. Therefore, we must get reservation for ramp (*rc_buffer*) before getting down to city roads as shown in Figure 1(b) to avoid a deadlock. That means at any point there will be only those many outbound cars on the city road, which can be accommodated in the ramp between the city road and highway. We assume that the highway and city roads do not fall in deadlock if they operate in isolation. Hence we propose *RC* to control the injection of outbound packets to ensure isolation of these two types of traffic. The main contributions of our paper are as follows.

- We tackle an emerging problem of deadlock in chiplet-based systems using *Remote Control*. We aim to provide better design flexibility to both the chiplet and SoC designers, so that chiplet designers can exercise their expertise to fully optimize the chiplet performance. To the best of our knowledge, this is the first work that uses injection control based technique to guarantee deadlock freedom in the modular SoC design domain.
- We apply conventional deadlock avoidance techniques successfully in Modular SoC and analyze comprehensively to provide consolidated comparison. We observe that even after our best effort to make them efficient in the modular SoC, they are not able to exhibit desired performance. Hence, to overcome their limitations, considering several aspects together, we show that our solution is the most efficient in this context.
- We formally prove that *RC* guarantees deadlock freedom in modular SoC using an illustrative generalized example.
- Along with the theoretical proof, we also validate the design in terms of energy, area, and timing constraints through RTL. We evaluate the SoC network performance using a network simulator and provide system performance results using full system simulations for both homogeneous (only CPUs) and heterogeneous (CPU and GPU) system configurations. We achieve up to 56.34% throughput and 15.49% zero load latency improvements, respectively, over state-of-the-art MTR.

The rest of the paper is organized as follows. In Section 2, we briefly describe the background, and motivations for *RC* are explained in Section 3. Then we introduce a theory and its formal proof in Section 4, followed by the discussion on implementation challenges. In Section 5, the *RC* implementation is presented. We discuss the experimentation methodology in Section 6 and show the effectiveness of *RC* compared to MTR in Section 7. We discuss previous work on network deadlock applied in different types of networks in Section 8. Finally, we state our conclusions and discuss future work in Section 9.

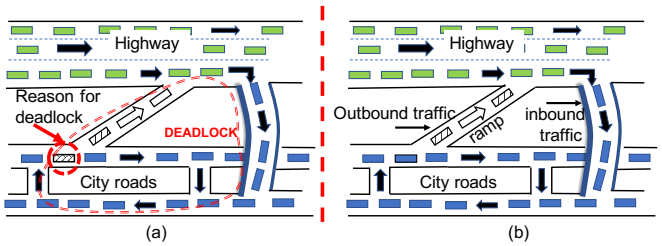


Fig. 1: Analogy with a Road Situation. (a) Traffic is forming deadlock involving both highway and city-road traffic because of mutual blocking. (b) Once we make sure that all the highway-bound cars can stay in the ramp, until they get a space in highway, deadlock freedom is guaranteed. Note that in this case (also in out inter-chiplet deadlock) traffic isolation is enough to avoid deadlock, as highway and city-roads are deadlock free.

2 Background

In this section, we first introduce the modular SoC design concept and 2.5D silicon interposer technology, as an important and elegant way of system scalability for performance boost. However, integrating independently designed chiplets introduces network deadlock, involving multiple chiplets. We study MTR and two possible conventional ways that can be applied for tackling that issue, followed by a discussion on the limitations of existing mechanisms as motivation for our work.

2.1 Modular 2.5D SoC Integration

Modularity has been advocated as a new design principle to reduce the complexity and cost of the SoC design. An SoC is called modular if all the chiplets on that SoC are designed independently. Contemporary multi-chiplet SoC integration uses a passive silicon interposer [11], where the only way to make connections between chiplets is to make fixed wire connections.

In a passive interposer, dedicated wire connections are required from a chiplet to connect with different chiplets [12]. This may lead to long wire usage with multiple repeaters and a huge number of dedicated communication channels, making it hard to scale in terms of area and energy. In addition, the channels in a passive interposer should be standardized for the modular SoC design. Hence, there has been an increase in research of active interposers [2], [13], [14] both in industry and academia. We also consider an active interposer substrate for designing the interposer network.

Active interposer facilitates interconnection between the chiplets [12], [15], [16] by adopting the router design in the silicon substrate, which is more area-and energy-efficient. The integration process is generally known as 2.5D integration, featuring a silicon interposer. It is placed between the System-in-Package (SiP) substrate and the dice, where this silicon interposer has Through-Silicon-Vias (TSVs) connecting the metalization layers on its upper and lower surfaces [16].

2.2 Deadlock Freedom in Modular SoC

2.2.1 Modular Turn Restriction (MTR)

Based on the principle of turn restrictions [17], recently, Yin et al. [5] propose a deadlock-free routing algorithm for modular SoC. As the best of our knowledge, this is the only work on modular SoC deadlock freedom so far. At design time, MTR finds the optimal placement and turn restrictions for boundary routers of each chiplet independently with the help of Channel Dependency Graph (CDG) analysis. The turn restrictions are applicable to both the packets that go out from the chiplet (*outbound packets*) as well as to the packets that reach from other chiplets (*inbound packets*). Once the list of turn restrictions is obtained, MTR applies the turn restrictions in the chiplet routing, which are applicable only for the outbound packets. For imposing turn restriction on the inbound packets, interposer routing also needs to be modified, which imposes constraint on the SoC designers and increases design complexity.

2.2.2 VC Separation (VC-SEP)

The idea of VC separation is widely applied to avoid protocol deadlock as well as routing deadlock based on Duato's theory [18]. We showcase it as a potential solution for SoC deadlock since it is a natural fit for this particular problem. The traffic in the Modular SoC can be categorized into two. (1) *Inter-chiplet*: traffic of packets which do not have destinations in the source chiplet. (2) *Intra-chiplet*: traffic of packets having both sources and destinations in the same chiplet. Without any constraint on the area, cost and energy, VC-SEP naturally segregates two different traffic by providing two different virtual networks throughout the system. For outbound packets, we allocate first half of the total set of VCs, and other set of VCs are being allocated for inbound packets and all the intra-chiplet packets.

2.2.3 In-Transit Buffer (ITB)

The idea of ITB is originally used by Flich et al. to avoid deadlocks in irregular networks and later extended for off-chip networks in the cluster of workstations [10]. Here, we adopt the idea of ITB and apply in the Modular SoC to avoid deadlocks. ITB uses the Network Interface Card (NIC) memory as an in-transit buffer in some pre-decided nodes. Those special nodes are being selected after CDG analysis as deadlock breaking points. Any packet that reaches to those nodes are forced to eject in that node. Using DMA, the whole packet is stored in the NIC memory. In case the NIC memory gets exhausted, the packet is dropped and a NACK packet is being generated and sent to the source node for re-transmission. This process continues till the packet is ejected successfully in that special node. If the packet is successfully stored in NIC memory, the NIC sends an ACK message to the source node. To port this idea in the Modular SoC, we consider the boundary routers as special nodes and use the Network Interface (NI) connected to boundary routers to place ITB, a small buffer to store packets (no DMA) in the similar way described above. The ejection and reinjection in some special nodes (boundary routers) break the circular channel dependency chain and hence avoid deadlocks. By using domain specific knowledge we further optimize the performance of this implementation by doing ejection-injection only when a packet is outbound. Please note that we made necessary changes to the original implementation for accommodating the idea of ITB into the modular SoC context.

	Modularity	Design Efficiency	Energy Efficiency	Performance
MTR	$+++$	$++$	$+++$	$+$
VC-SEP	$--$	$++$	$---$	$-$
ITB	$+++$	$--$	$--$	$++$
RC	$+++$	$+++$	$+++$	$+++$

TABLE 1: Qualitative Comparison with Different Deadlock Avoidance Techniques for Modular SoC. $(+)$ means high and $(-)$ means low. We project the degree of high and low efficiency with number of $(+)$ or $(-)$, respectively.

3 Motivations for *RC*

Limitations in the existing techniques discussed in Section 2 motivates us to find a better solution. State-of-the-art MTR identifies an important emerging problem and provides a solution. However, MTR has a few limitations. The major constrain in MTR is that it forces the chiplet designers to implement turn restrictive routing to guarantee deadlock freedom in the modular SoC, which is the main motivation of *RC*. In addition, extra turn restrictions can lead to non-minimal path for intra- and inter-chiplet traffic. Also, the turn restrictions obtained by MTR does not balance the turn restrictions among the boundaries well. Hence, a few boundary routers get huge inter-chiplet traffic load while others do not, causing several hotspots in the system. MTR also constrains the routing design and incurs design overhead. The complexity of the CDG analysis, which is the core of this technique, grows exponentially with the increase in the number of chiplet routers and boundary routers, which unnecessarily elongates the design cycle. Moreover, MTR imposes restrictions on interposer routing to restrict the inbound packets route, which increases system network design complexity and traffic contention further.

VC-SEP is a well known technique for avoiding deadlock. The main drawback of this approach is that it is very expensive in terms of energy and area consumption. In addition, the buffer utilization is low, which leads to sub-optimal performance. Hence, even though this solution is fairly simple, it is not attractive for designing cost-effective and high throughput modular SoC. Hence, we use the idea of traffic isolation and come up with more efficient implementation by adopting remote injection control mechanism with small buffer.

ITB could be a promising solution for deadlock problems in Modular SoC. However, it has two major drawbacks in this context. (1) Dropping a packet in on-chip reliable network introduces unnecessary complexity and overhead. (2) Ejection and reinjection in multiple nodes increases overall hop counts as well as average packet latency. Furthermore, due to packet dropping/reinjection and use of ACK/NACK packets, the overall system throughput suffers. In principle *RC* is different than this solution as *RC* does not rely on ejection and reinjection for deadlock freedom. In fact, *RC* ensures deadlock freedom just by isolating two types of traffics in the system, which is inspired by the VC-SEP idea.

In Table 1, we summarize the comparisons of these techniques and project the expectation of *Remote Control*,

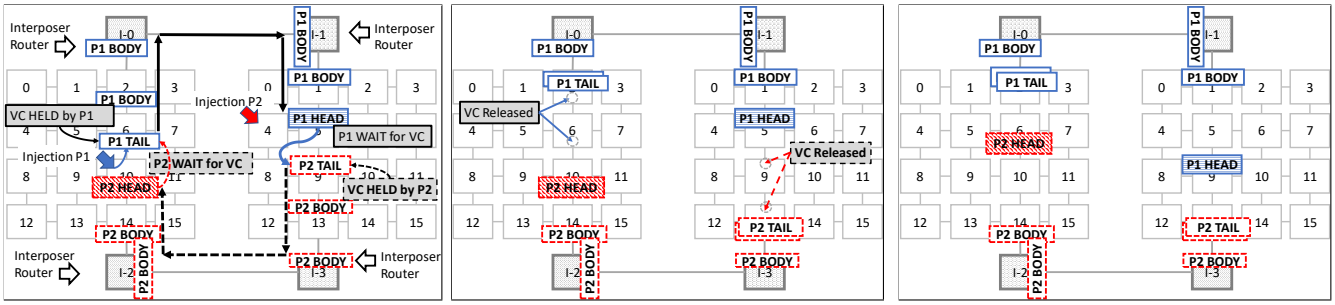


Fig. 2: An example of deadlock, formed in between two (4×4) mesh chiplets, and how *RC* can avoid the deadlock. (a) Packets P1 (blue solid line) and P2 (red dashed line) forming deadlock. (b) `rc_buffer` in the boundary routers store outbound packets. (c) *RC* avoids the deadlock by allowing all the outbound packets to be stored in the boundary routers until they get credit from downstream interposer routers.

which aims for improving the limiting aspects of existing solutions. In a nutshell, the goal of *RC* is to provide routing design flexibility and eliminate unnecessary packet dropping incurring packet re-transmission by introducing a flow control based technique. Additionally, *RC* targets to save energy and area by segregating traffic only in chiplet boundary routers.

4 Remote Control

RC is a deadlock avoidance solution for modular SoC, implemented using injection control imposed on outbound packets from nodes connected to non-boundary chiplet-routers. Since outbound packets get consumed in other chiplets, to avoid cross-chiplet deadlocks, we provide intermediate sink (*rc_buffer*) for outbound packets in the boundary routers. Therefore, outbound packets are drained to *rc_buffer* so that they release the chiplet VC buffers to be used by intra-chiplet and inbound packets. If all the chiplets follow the same, intra-chiplet and inbound packets are never indefinitely blocked by outbound packets, and in turn outbound packets can also make progress, as an outbound packet for one chiplet is an inbound for other chiplet. In this section, we first walk through a simple practical example to show a case of deadlock formation between two chiplets and how *RC* can solve it. Then we generalize it for any chiplet-based systems and theoretically prove that *RC* guarantees deadlock freedom in Modular SoCs.

4.1 Deadlock in Modular SoC

Figure 2a shows a deadlock case in a modular SoC, where two (4×4) 2D mesh chiplets are connected through an interposer. We denote router i on chiplet j as R-i/C-j for simplicity, where chiplet-0 is on the left and chiplet-1 is on the right. In this system, R-2/C-0, R-14/C-0, R-1/C-1 and R-13/C-1 are boundary routers connected to the interposer network. Packets P1 and P2 are two outbound packets in a circular hold-and-wait situation, forming a deadlock. The P2-head flit in R-10/C-0 requests for the south VC of R-6/C-0, which is held by packet P1. On the other hand, P1-head flit in R-5/C-1 requests for the north VC of R-9/C-1, which is taken by P2. Such a case creates a circular hold-and-wait situation and forms a deadlock, where neither P1 nor P2 can make forward progress[2]. To avoid deadlock in this scenario MTR may impose turn restriction from *R-6/C-0* to *I-0* through *R-2/C-0*, increasing pressure on the other boundary of C-0 for outbound traffic. Note that MTR needs to impose more turn restrictions to avoid all other possible circular hold-and-wait scenarios.

2. Progress/forward-progress means moving near to the destination.

4.2 Deadlock Avoidance using *RC*

As shown in Figure 2a, an outbound packet P1 in C-0 is blocking P2 packet to reach its destination while P2 is blocking P1 in C-1. *RC* separates outbound packets from others in the boundary router, and allows them to be stored completely in the *rc_buffer* until they get credits from the downstream interposer router. This makes sure that the chiplet VCs will get free in a bounded time, after the header flit reaches at *rc_buffer*. Hence, P1 will release all the VCs currently blocked in C-0 and so does P2 in C-1. That is why with *RC* the circular channel dependency among chiplets will never result in deadlock.

4.3 Deadlock Freedom

We propose the theoretical support for *RC*. The *rc_buffer* reservation is atomic for each of the requesters and allows only one outstanding *rc-request* per requester at any point of time. The request for a slot in the *rc_buffer* for any outbound packet is granted only when there is space for the whole packet, and the slot is reserved. The slot is released once the tail flit leaves the *rc_buffer* and then only a new request is granted. We define one rule and two definitions to keep our theorem statement concise. We layout the proof of the theorem by contradiction, starting by assuming that there is a deadlock.

Injection Rule: An outbound packet can be injected from NI to its attached router **if and only if** the *rc-request* is granted.

Definition-1: *Inflight outbound packet* is the packet whose source and destination are in different chiplets, and holds at least one VC in one chiplet-router. Note that once the outbound packet leaves its source chiplet, it is considered as *inbound* packet for its destination chiplet.

Definition-2: *Destination boundary router* is a chiplet-router used by a set of nodes/routers in the chiplet as a gateway to communicate between the chiplet and interposer. Any communication between these nodes/routers in the chiplet and interposer happen only through their *destination boundary router*.

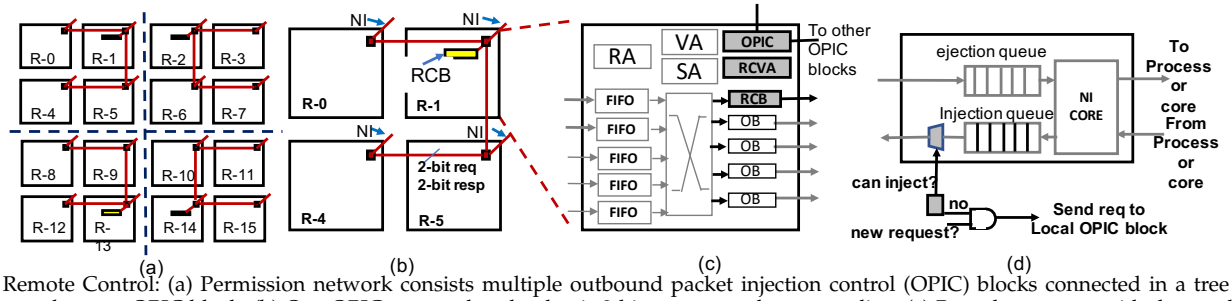


Fig. 3: Remote Control: (a) Permission network consists multiple outbound packet injection control (OPIC) blocks connected in a tree fashion. Each router has one OPIC block. (b) One OPIC tree, and each edge is 2-bit request and response line. (c) Boundary router with the newly added components marked with gray. Note that the router attached to a non-boundary node does not have RCVA and RCB. (d) The changes in NI of the non-boundary router marked in gray. There is no change in NI attached to boundary routers.

Theorem: *The SoC network is guaranteed to be deadlock free as long as all the outbound packets reserve slots in rc_buffer in the destination boundary router before injection (Injection Rule).*

Proof: We define the network system (S) at any instant T as $S_T = \{Q \mid Q = \{\lambda, \beta\}, \lambda \subseteq P_T, \beta$ is the set of all the buffers reserved by packets in λ and $\beta \subseteq B\}$, B is the set of total buffers in the system (independent of T) and P_T is the set of total packets in the system at an instant T. Let us assume there exists an i such that $Q_i = \{\lambda_i, \beta_i\} \in Q$ forms a deadlock. Let $\mathfrak{B}$ denote a set of buffers in all the boundary routers in the SoC system and ρ denote the set of inflight outbound packets. We categorize all possible scenarios into four cases as follows and prove that *RC* avoids deadlock for any modular SoC network by contradicting our initial assumption that Q_i is in deadlock.

```
if βi ⋂ 𝔅 == ∅ :
   - contradiction; //no deadlock ..............(1)
else:
   if λi ⋂ ρ == ∅ :
     - contradiction; //no deadlock ..............(2)
   else:
     if ∀ ρx ∈ (λi ⋂ ρ) ∃ slot in rc_buffer:
       - contradiction; //no deadlock ..............(3)
     else:
       - violation (Injection Rule);//no deadlock.(4)
```

(1) If there is no boundary router buffer involved, that means the deadlock is formed only inside a chiplet or in the interposer, but not involving both. Since a chiplet and the interposer use their own deadlock free routing logic, it is impossible to form deadlock without involving both, which contradicts our initial assumption of Q_i being in deadlock.
(2) If there is no inflight outbound packet involved in Q_i, the circular deadlock chain is incomplete as the packets in λ_i are either intra (source and destination in the same chiplet and chiplet routing is deadlock free) or inbound (outbound packet that left its source chiplet as in Definition-1, so cannot connect with its source in the deadlock chain). So there cannot be any deadlock in Q_i.
(3) If the Injection Rule is followed by each chiplet, then *rc_buffer* will let all the inflight outbound packets to sink in there, allowing intra and inbound packets to reach their destinations following deadlock-free chiplet/interposer routing. This in turn allows outbound packets to leave their source chiplets and become inbound for their destination chiplets (Definition-1). In addition, deadlock-free chiplet routing guarantees that all the outbound packets reach their destination boundary routers in the source chiplets. There cannot be any deadlock as long as all the packets (intra and inbound) reach their destinations. Therefore, Q_i cannot be in deadlock.
(4) If any inflight outbound packet ρ_x in λ_i that has no *rc_buffer* slot reserved in the boundary router, it violates *RC*'s Injection Rule that all the outbound packets MUST reserve a slot in *rc_buffer* before injection (Injection Rule). Therefore, this situation cannot happen. Hence *RC* provides guarantee in deadlock freedom for modular SoCs.

4.4 Challenges

There are several implementation challenges *RC* may face. (1) Remote injection control implementation needs to establish an extra channel of communication, which may look structurally similar to the credit channel. The challenging part is to keep the communication overhead as minimum as possible. (2) The *rc_buffer*, situated in the boundary router needs to allow all the inflight outbound packets to be drained from the router VCs. In conventional system, a packet leaves the VC from the upstream router only when the VC in the downstream router has been reserved and the router switch is allocated. Whereas *RC* requires the packets to be drained irrespective of the success in the conventional VC allocation in the boundary router. (3) The *rc_buffer* is common for all types of packets, and hence it might suffer from fragmentation if not taken care properly for different packet sizes.

4.5 Routing Oblivious Design

RC is oblivious to both the chiplet routing and the interposer routing. The boundary routers are considered as local destinations for outbound packets in a chiplet. Hence, any routing technique can be used to reach to the boundary routers. Through a boundary router an outbound packet reaches a downstream interposer router, and it follows the interposer routing to reach another local destination in the interposer. Since the interposer takes care of the communication between two chiplets, one inter-chiplet packet only traverses through its source and destination chiplets. Once the packet enters its destination chiplet, it follows the chiplet routing to reach its destination node.

5 Implementation

Is this section, we first discuss one possible implementation of *rc_buffer* (RCB) along with a supporting protocol called RC virtual channel allocation (RCVA). Then, we build a

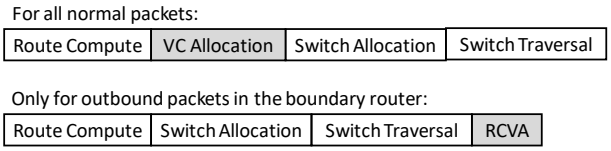


Fig. 4: Router pipeline stages in case of normal packets followed by modified router pipeline in case of handling outbound packets in the boundary router.

permission network connecting outbound packet injection control (OPIC) blocks. Our goal is to achieve deadlock freedom with complete routing flexibility both in the chiplet and in the system backbone network.

5.1 Boundary Routers

Each boundary router has three new components, RCVA, RCB, and OPIC as shown in Figure 3 (c). In *RC*, RCVA protocol helps to operate the *rc_buffer*, while the permission network built by connecting OPIC blocks transports permission requests and responses to and from the *rc_buffer*. Altogether, they establish a fine control over the outbound packet injection and its safe transmission to *rc_buffer* to achieve deadlock freedom by alleviating mutual blocking among different traffic types.

5.1.1 RCVA

RCVA implements two functionalities. First, it makes sure the outbound packets injected from non-boundary routers do not participate in VC allocation (VA) in the boundary router but directly do switch allocation as shown in Figure 4. We bypass the VA stage of the router to save its latency, and when switch allocation is successful, the packet reaches the output port through the crossbar. Then, we push the packet in the reserved slot in RCB.

Second, in each cycle RCVA checks if there is any candidate waiting in RCB for VC allocation. The VC allocation logic in RCVA is much simpler and straightforward than that in the VA, as RCVA deals with only one port. Moreover, since RCB collects all the outbound packets from all the input VCs, VA does not deal with the outport that connects the downstream interposer router. Hence, RCVA does not increase the number of stages in the router for any packets. For outbound packets we consider a different router pipeline, which has same number of router stages as in the normal router as shown in Figure 4.

5.1.2 RCB

RCB is a very small buffer located between the crossbar switch and link to the downstream router. It has one or more slots for packets. If the head flit of a packet is written in the first half of a cycle, it may be considered for VC allocation in the other half of the cycle by RCVA. RCB reserves a space when a request arrives from the OPIC block, and allocates one of the empty reserved slots to a packet when its head flit arrives. Flits leave RCB when credit is available for downstream VC buffer. The slot is freed once the tail flit leaves from RCB. To handle different packet sizes efficiently, maximum packet size is being reserved in RCB after receiving request, and once the head flit arrives, we allocate exact number of slots as needed.

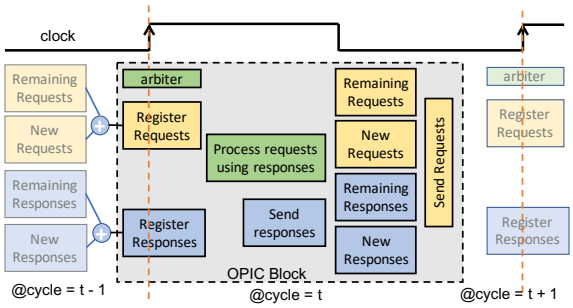


Fig. 5: Components in an OPIC block along the timeline (not to scale).

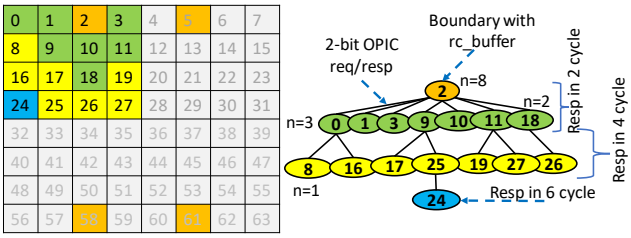


Fig. 6: Example of permission network in 8×8 mesh with 4 boundary routers with rc_buffer connected. n denotes the radix of the OPIC block.

5.1.3 Permission Network

In this section the basic building block (OPIC) of the permission network is explained, followed by the procedure for building the permission network, maintaining transparency with the chiplet network.

OPIC block: As the name suggests, this block is responsible for regulating the injection of outbound packets in the chiplet through its customized permission network. Each of the blocks supports send-and-receive functionality for both permission requests and responses. In Figure 5 we divide the OPIC into smaller blocks, and place them in the timeline with respect to the clock. In the beginning of the clock all the requests and responses reached in the last cycle are registered in separate registers for each connecting OPIC blocks. Those registers are read asynchronously and based on the available permissions, responses are sent to requesters in a Round Robin (RR) fashion after processing their requests using combination circuits; after that, remaining requests and responses are also calculated. Again in the next clock, total requests and responses are being registered and the same process continues.

Network: We explain the permission network building process in Algorithm 1 with an example. The chiplet network topology and set of boundary routers being the inputs, this permission network building process can be demonstrated in general for any chiplet topology and any number or positions of boundary routers. In the example, the input topology is provided for an 8×8 mesh in the form of adjacency matrix, along with boundary router set that contains node numbers 2, 5, 58 and 61. Depending on the technology size, chiplet designers can decide the wire-length for the OPIC block connections (it is a hardware deployment concern, so not included in the algorithm), which may result into different tree depths. The number of permission trees is equal to the number of boundary

```
   Input  : Adjacency matrix ADJ of size N × N
   Input  : Boundary router list BDRL of size K
   Input  : Node list NODELIST of size N
   Output : list of OPIC trees (TREES)
1  // initialize the variables;
   Vector  : TREES; // list of OPIC trees;
   Variable: num_trees = # boundary routers;
   Variable: all_visited = false;
2  // initialize one tree for each boundary router;
3  for i ← 0 to num_trees do
       Variable: root = BDRL[i];
4      TREES.push_back(root); // pushed the root of tree;
5  end
6  // keep looping until the node_list becomes empty;
7  while all_visited == false do
8      // check for each tree;
9      for t ← 0 to num_trees do
           Variable: cur_tree_size = TREES[t].size();
10         // try to add neighbors for each tree node;
11         for node ← 0 to cur_tree_size do
               Variable: target_node = TREES[t][node];
12             // check for all neighbors of the target_node;
13             for a ← 0 to ADJ[target_node].size() do
                   Variable: nbr = ADJ[target_node][a];
14                 // checking if neighbor of the target_node is there
                     in the node_list;
                   Variable: inList, index, boundary =
                                 FindInNodeList (nbr);
15                 if inList == true then
16                     if boundary==false then
17                         // adding node to the OPIC tree;
18                         TREES.push_back(nbr);
19                     end
20                     // removing the node from node list;
21                     NODELIST.erase(index);
22                     if NODELIST.empty() then
23                         all_visited = true;
24                     end
25                 end
26             end
27         end
28     end
29 end
```

Algorithm 1: Build Permission Network

routers in the chiplet. OPIC block in node-2 connects with OPIC block in nodes-0, 1, 3, 9, 10, 11, 18 using request and response lines of width 2-bit each in a n-ary tree. Similarly, node-0 is connected with nodes-8, 16 with the request and response lines, and so on.

For instance, at any cycle t, node-8 and node-27 want to inject outbound packets, the requests will be registered in node-0 and node-11 at the beginning of $t+1$, respectively. At $t+2$ the requests from node-0 and node-11 will be registered in node-2. Let us suppose the *rc_buffer* does not have a space, in that case the requests will be standby in these nodes. Now suppose at cycle T one packet space gets free in *rc_buffer* and depending on the arbitration one of these two will get the response at $T+1$. Suppose node-0 gets the response, then at cycle $T+2$ node-8 will get the permission to inject an outbound packet.

5.2 Non-boundary Routers

In a non-boundary router, we append a control on injection process, which allows all the intra-chiplet packets to go without any check. Only for inter-chiplet traffic, the modified injection system checks for injection permission from a local OPIC block. In Figure 3 (d), we show that if the

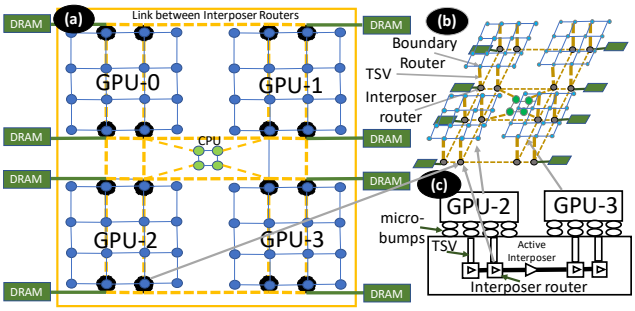


Fig. 7: SoC viewed from different angles. (a) Shows the top view of the SoC. There are four GPU chiplet (4×4 mesh) at the four corners, and a CPU chiplet (2×2 mesh) in the center. DRAM memory is connected with edge interposer routers. (b) 3D view of the same SoC, highlighting the interposer router, boundary router, and TSV that connects them. It also show the active interposer, and the mesh network in the interposer. (c) Microscopic cross-section view of SoC highlighting the micro-architectural details of 2.5D SoC integration on active interposer.

permission is not there, the outbound packet is not injected and a request for permission is sent by the local OPIC block to the remote OPIC block in the boundary router through the permission network. Once the response reaches, the outbound packet is injected. Hence, it may happen that if the outbound packet does not get permission to inject, it can block intra-chiplet packets. A separate injection queue for outbound packets in the non-boundary routers may solve the issue. The pros and cons of adding an extra injection queue are discussed as follows.

5.2.1 Separate Injection Queue

We consider the separation of injection queues in the non-boundary routers as a design choice for the following reasons. The advantage of having a separate queue can be exploited only if (1) the VC is abundantly available for injection in the router, (2) the workload is very unevenly distributed among chiplets. For instance, in a hypothetical situation with two chiplets, where one chiplet has huge intra-chiplet traffic and huge congestion inside the chiplet, and the other chiplet has a few outbound packets. Those outbound packets may block the intra-chiplet packets in the injection queue for a long time.

In case the number of VC buffers is low/minimal, unavailability of the VCs becomes the bottleneck and injection queue separation turns to be almost irrelevant. So in our general design, we do not consider an extra injection queue for the outbound packets as it increases the NI design complexity significantly.

5.3 Case Study: Modular CPU-GPU Integration Using Silicon Interposer

To check the feasibility of our design, we conduct a thorough case study on simple heterogeneous system comprised of four GPU chiplets (16-PEs/GPU, 32-SIMD/PE) and one CPU chiplet (4-cores) [5]. The first challenge we face is to equip the network of each chiplet with *RC* independently. In non-boundary routers the area overhead of OPIC is less than 0.2%. However, in boundary routers area overhead is almost 1.6% over the router area, including *rc_buffer* of depth 8 packets, while the overhead on the NI is negligible. In

the established permission network, we ensure the setup and hold time has sufficient slack for both requests and responses between the OPIC blocks. The permission network works independent of the chiplet network, operating with the same 2GHz frequency as that of the chiplet network.

Once the chiplets are equipped independently with *RC*, they are ready to be integrated in an SoC using 2.5D active interposer as shown in Figure 7. In this SoC, the interposer provides a (4×4) mesh network for inter-chiplet communication. Since in the interposer network, edge routers are connected with DRAM memory, memory controllers are connected with those routers. The edge routers in the interposer also contain the coherence directory. Chiplets are connected with the interposer using micro-bumps. TSV connects the micro-bumps with the interposer routers. Interposer routers use internal link to connect with each other. For instance, in Figure 7 (c), if GPU-2 wants to send a request packet to GPU-3, then that packet will reach to the boundary router of GPU-2 first. In the boundary router, the packet will make an entry in *rc_buffer*. From the boundary router, the packet will reach to the interposer router through the TSV. Once the packet reaches to the interposer router, it will be routed to the interposer router that is connected to GPU-3. Again, through the TSV the packet will reach from the interposer router to the boundary router of GPU-3.

6 METHODOLOGY

We verify and evaluate feasibility of the design in terms of both functional correctness and design efficiency by synthesizing permission network, along with state-of-the-art 4 stage routers RTL [19], using *TSMC 45nm* library. Functional correctness is verified using extensive test-cases. Average area, power, and delay experienced by permission network are analyzed by simulating RTL model.

To evaluate viability of *RC* in the target system, we build software prototype in gem5 [20]. We experience that *RC* can be seamlessly integrated in cycle-accurate network model of BookSim [21]. Hence for full system setup, we integrate BookSim with gem5. We configure gem5 for both heterogeneous (CPU-GPU) and homogeneous (CPU-CPU) systems. We want to prove that *RC* is easy to integrate in full systems and its functionality does not introduce any new issue and report the observed full system performance.

Finally, we thoroughly study the deadlock issue in hierarchical network system using extensively modified BookSim (reliably implement hierarchical network topology and routing) for several synthetic traffic patterns across large range of injection rates (even beyond saturation points), different VC and `rc_buffer` sizes, different number of boundary routers, different network dimensions, and different routing techniques. We want to prove that in modular SoC, routing flexibility is not an option, it is necessity.

6.1 Experimental Setup

In full system setup, we integrate BookSim with Gem5 to simulate network of Compute Units (CUs) in the GPU chiplets (GCN-3 [22]), CPUs in CPU chiplet, and also different chiplets on active interposer as summarized in Table 2. We use 4 stage routers having 1-flit buffers per control VC and 4-flit buffers per data VC. Flit size and link channel width is 64 bit. The control packets are 1 flit and data packets are 5 flits. In homogeneous setup we configure SoC using multiple (4×4 mesh) CPU chiplets only and use MOESI hammer as the coherence protocol. Heterogeneous setup uses the multi-chiplet APU configuration [5], consisting of four GPU chiplets (4×4 mesh, 16 CUs), one CPU chiplet (2×2), and an active interposer (4×4 mesh) as shown in Figure 7. We use the in-built memory model in gem5 equipped with eight memory channels and 8 banks per channel. We run heterogeneous system-level simulation on APU applications taken from AMD ROCm Developer Tools [24] and Rodinia [25] suites. We also evaluate *RC* in homogeneous full system setup using PARSEC [26] and SPEC CPU2017 [27]. For running the Machine Learning applications, we attach one accelerator [28] on each node and experimented for training and ring all-reduce operations using underlying BookSim network simulator. Unless otherwise mentioned, for synthetic experiments packet size is 8 flits; we use four 4×4 chiplet and one 2×2 chiplet connected using 4×4 interposer network, having 2-VC-4-stage routers with 4-flit buffer depth and 4 packet space in the *rc_buffer*.

Parameter	Value
CPU	2 GHz frequency, TimingSimple
CPU Cache	L1I and L1D - 32KB 4-way L2 - 64KB 8-way
GPU	1 GHz frequency [22]
GPU Cache	SQC (shared L1I) - 32KB, 8 way TCP (private L1D) - 16KB , 16 way TCC (Texture Cache per Channel) - 256 KB, 16 way
Memory	Build-in memory model in Gem5 [23]
Network	Booksim integraded with Gem5, 4-stage routers 1-flit buffers per control VC, 4-flit buffers per data VC 64 bit flit size and channel width
Permission N/W	2 cycles/ OPIC hop (round trip including OPIC block latency)

TABLE 2: Parameters of simulated architecture.

6.2 System Speedup

We evaluate our design using both latency sensitive workloads as well as throughput sensitive workloads as shown in Figure 8. We evaluate our design for the latency sensitive PARSEC benchmarks and GPU benchmarks and observe performance difference with *canneal* in which the average packet injection rate is moderate to high and almost similar performance for other benchmarks with MTR as *RC* performs also similar to MTR for low and moderate network load. On the other hand for throughput-sensitive programs, we expose both the techniques to several Machine Learning applications [28] as well as parallel execution of multiple instances (32 copies of same application on 32 different cores) of SPEC CPU2017 benchmarks [27]. As expected, we observe 14% to 20% system speedup, only attributed to an efficient deadlock avoidance scheme (*RC*). Among the throughput hungry benchmarks, *AlexNet* is a slight outlier as the amount of time spent on communication for this benchmark is significantly low as compared to other benchmarks. In addition, full system performance can be boosted by prioritizing delinquent packets [29]. We can partially achieve that by simply modifying the arbitration policy in RCVA. Since, that is orthogonal to our current work, we leave the performance optimization for the full system setup as our future work.

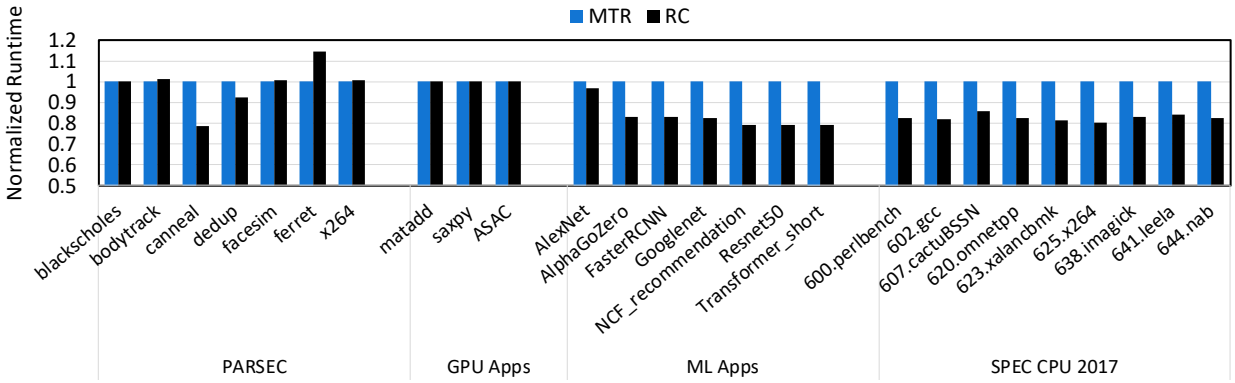


Fig. 8: Normalized execution time for real applications **(lower is better)**.

7 Performance Evaluation

Keeping the gravity of the inter-chiplet deadlock problem in mind, we quantitatively compare *RC* with MTR, ITB and VC-SEP. We extensively modify BookSim to reflect the hierarchy of networks. We introduce the concept of chiplet and interposer in BookSim to reflect their independent topology and routing. Different link latencies are also reflected depending on their length in the interposer. Initial system we mimic in BookSim, is similar to our prototype with CPU-GPU in terms of their corresponding network. Then we expand our design space to evaluate different sensitivity aspects for complete study. To thoroughly study the deadlock formation we use several synthetic traffic patterns across huge range of injection rates. We observe that since the chiplets and interposer routing techniques are deadlock free, inter-chiplet deadlock forms during high congestion, near to the saturation points.

7.1 Throughput Analysis

Figure 9 shows that *RC* outperforms MTR, ITB, and VC-SEP in terms of network throughput in all the synthetic traffic patterns. We explain the throughput for uniform random (UR) traffic as a representative of synthetic traffic patterns. In *UR*, the source and destinations are generated randomly, where most traffics result into inter-chiplet communication. For example, 3/4 of the generated traffics consist of outbound packets in the simulated configuration, which poses more stress on the boundary routers and interposer network. Across all the techniques, VC-SEP has least throughput, due to under utilization of buffer resources. While in MTR, turn restrictions on boundary routers create load imbalance, leading to throughput degradation. In ITB, ACK/NACKs packets are used for re-transmit request, data and control packets, which leads to higher network load and saturates the network earlier. In addition, ejection and reinjection of packets add latency in the critical path. In contrast, *RC* regulates outbound packet injections facilitated by *rc_buffer* in boundary routers and frees VC usage constraints for better resource utilization. Additionally, *RC* provides routing flexibility so that traffics can be distributed evenly to all boundary routers. With these benefits, *RC* improves network throughput upto 1.7×.

To analyze the traffic distributions and communication bottlenecks of different designs, we depict hotspots as heatmap for UR for MTR, ITB, VC-SEP, and *RC* at their near saturation load[3] as shown in Figure 10. Hotspot is defined as the average packet residency time in the router. Darker color represents higher packet residency time due to congestion. MTR imposes multiple extra turn-restrictions, resulting into hotspots due to imbalanced traffic distribution inside chiplets as shown in Figure 10a, which leads to low network throughput as shown in Figure 9. Heatmap for VC-SEP, as depicted in Figure 10c, shows the severe congestion throughout the SoC network, which is due to the intensive usage of limited outbound VCs, making network saturation early. Interestingly, for ITB the contention inside the chiplets is very low, which can be attributed to packet drop [10] that yields the buffer resources in the network. However, extra packet transmissions cause high energy and power consumption in the chiplets. Since *RC* has uniform flow of packets as shown in Figure 10d, it exhibits a better throughput than MTR and VC-SEP. *RC* alleviates the long waiting of outbound packets from the chiplet routers. However, the contention in the interposer network partially offsets the throughput benefit, observed using *RC*. Note that we have plotted the heatmap with different injection rate (throughput injection rate) for each technique to show their distinct saturation behaviors, and point out the key reason for saturation. We notice that across all the techniques the interposer network is heavily used (outbound packets from multiple chiplet nodes go through one interposer network), which could be a bottleneck for achieving throughout improvements. To alleviate the contention from the interposer, we plan to extend our work to investigate innovative SoC topologies as our future work.

7.2 Latency Analysis

As shown in Figure 9, we observe a similar low load latency among *RC*, MTR and VC-SEP across various traffic patterns. Figure 11a presents the detailed comparison of low-load latency for *UR* as an example. It shows ITB increases a few more cycles as compared to other techniques. Extra ejection and re-injection at low-load incurs two extra hops that causes high latency overhead. On the other hand, MTR fails to follow minimum path to destination because of the extra turn restrictions. In case of *RC*, because of the

3. Different techniques have different saturation load.

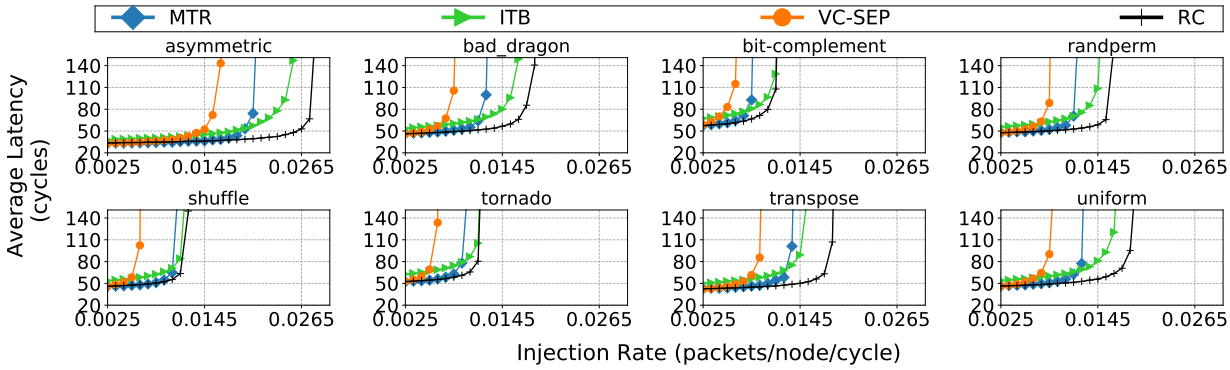
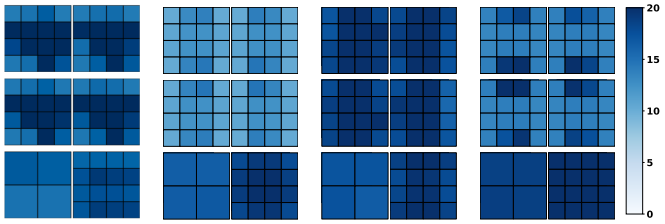


Fig. 9: Throughput graph for synthetic traffic pattern study. (#VC = 2, VC buffer size = 4, packet size = 8 flits, *rc_buff* = 4 packets).



(a) MTR. (b) ITB. (c) VC-SEP. (d) RC.

Fig. 10: Heat map of average packet residency latency (cycles) per router (smallest cube) in UR plotted for **near saturation point** for each technique. Top four cubes (big cubes) represent GPU chiplets, having 16 routers (small cubes) each. The bottom-left cube is the CPU chiplet having 4 routers. Beside the CPU chiplet we show silicon interposer with 16 interposer routers.

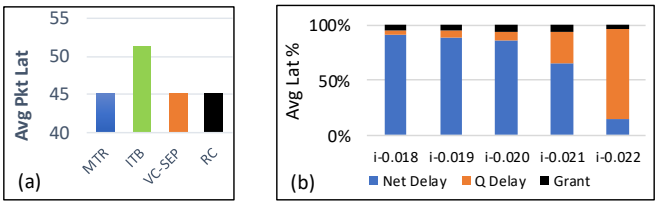


Fig. 11: Analysis on throughput graph in Figure 9. (a) Zero load latency of UR, representing the trend for others as well. (b) The network breakdown for *RC* across different injection rates for UR. "Net Delay" is the network delay including retry latencies. "Q Delay" is the injection queue latency. "Grant" response latency for getting the permission from OPIC including waiting for *rc_buffer* full condition.

modular design, route is optimized in each of the independently designed modules, which may not result into shortest path from source node to destination node. We expect to achieve better low-load latency for *RC* if we incorporate more chiplet information by relaxing modularity constraints while designing interposer routing.

In Figure 11b, we show the average packet latency breakdown for *RC* for UR to understand the overhead incurred by injection control. It shows that the portion of granting delay for *rc_buffer* reservation over packet latency increases with the increase in network load at the beginning, and decreases while moving from medium load to high load. This is because at low load, *rc_buffer* reservation causes constant delay without contention. Whereas at medium load, contention on *rc_buffers* increases the granting delay. As injection rate increases to high load, the exponential injection queuing delay dominates the packet latency, which reduces the impact of *rc_buffer* reservation significantly. To alleviate round-trip delay of the permission network, *rc_buffers* can be operated in a more proactive way, similar to token circulation rather than on-demand requesting to reduce the constant delay at low to medium load. That may improve the permission request-response delay, if enough tokens flow throughout the network [30].

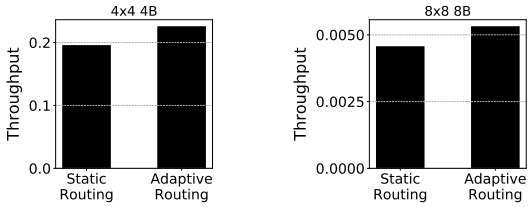


Fig. 12: Throughput improvement by using adaptive routing for different size of modular SoCs for the following systems. The left two bars are for 4 boundaries in 4×4 GPU chiplet and 4 boundaries in 2×2 CPU chiplet. The right two bars 8 boundaries in 8×8 GPU chiplet and 4 boundaries in 4×4 CPU chiplet.

7.3 Routing Obliviousness

In this section, we show *RC* is routing oblivious by implementing Dynamic Credit-based Routing, where each router adaptively selects either XY, or YX routing depending on the credit availability in the downstream router. To demonstrate the benefits of routing oblivious *RC*, we alleviate the bottleneck in interposer as discussed in Section 7.1 by providing 2 extra VCs only for interposer routers. In Figure 12, we show that when dynamic routing is applied, the throughput improves in both smaller system (68 node, 84 routers) and bigger system (272 nodes, 304 routers) by 15.3% and 21%, respectively. The main advantage of *RC* is that it gives complete freedom to the chiplet designers to implement the best routing for the chiplet, using their domain expertise, without being worried about system-level deadlock issue.

7.4 Starvation and Fairness

The system ensures that starvation never happens by serving each of the nodes in a Round Robin (RR) fashion. In terms of the OPIC delay, the first time request for outbound packet injection is logically decoupled with the consecutive retries, by registering the OPIC response in the local node's

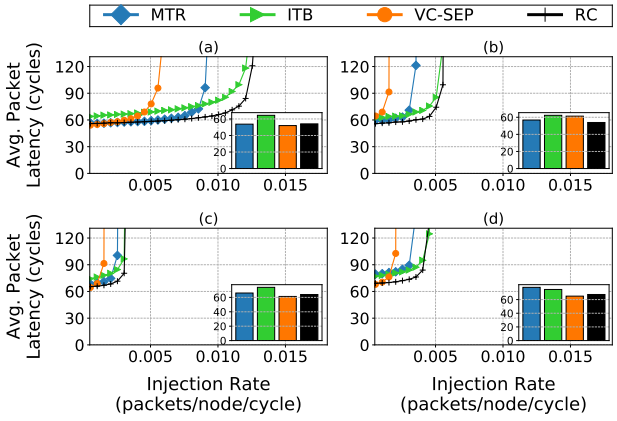


Fig. 13: Sensitivity study: Scaling the chiplet size keeping the number of boundaries same as 4/chiplet. (a) Eight GPU chiplets of size 4x4 and one CPU chiplet of size 2x2 mesh. (b) Two GPU chiplets of size 8x8 and one CPU chiplet of size 2x2 mesh. (c) Four 8x8 GPU and one 4x4 CPU. (d) Same as (c) except 8 boundaries/GPU chiplet. The small bar chart in each of the graphs represents zero load latency for that particular configuration.

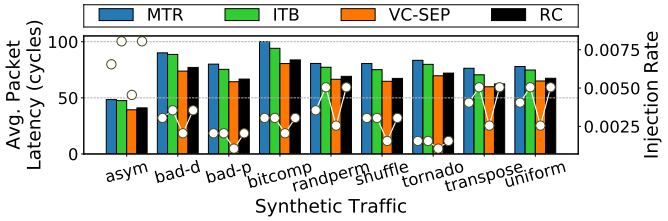


Fig. 14: Doubled the number of boundaries 8 boundaries/ 8×8 GPU chiplet and 4 boundaries in 4×4 CPU chiplet. The major Y-axis corresponds to zero load latency shown in bar graphs, and minor Y-axis corresponds to the throughput as shown in white dots.

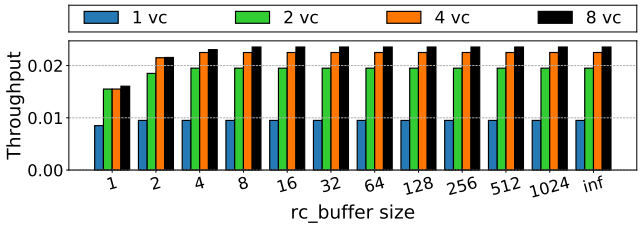


Fig. 15: Throughput sensitivity and interplay between virtual channel and *rc_buffer* size for 4×4 chiplets (68 nodes setup) with 4 boundaries/chiplet.

NI. This system works since *RC* proactively responds to the requesters whenever their turn comes in RR and the slot for entire packet is available.

The fairness issue can be broken down into two distinct situations, (1) when the *rc_buffer* is full, and (2) when the *rc_buffer* is available. When the *rc_buffer* is full, none of the requesters get served, regardless of their location with respect to the *rc_buffer*. In that case, all the requests wait in the OPIC block of the boundary router, so the Round Robin policy can serve them fairly. When the *rc_buffer* is available, a nearer node stands a higher chance to get served than a farther node, if they generate requests at the same time. However, it is not always true. In the OPIC block, when a requester's turn comes, the request gets served following Round Robin policy. This process continues until all the responses present in that OPIC block get exhausted. The serving starts again from the point, where it stopped last time. That is why in some scenarios, even if the request from the near node reaches first, the responses may get exhausted before its turn comes. By the time new response arrives, the request from the far node may get registered. Then depending on the last serving location, either the near or the far requester may get served. In summary, we guarantee that there is no starvation in the system. However, fairness is always not preserved, as the nodes that are near to the boundary router may consume higher OPIC bandwidth than the nodes situated farther, which is an inherent nature of any multi-hop network.

7.5 Sensitivity Analysis

We scrutinize the system using various size and number of chiplets to obtain better understanding about system scalability with *RC*. Difference of throughput is also observed with different VC sizes and increasing size of *rc_buffer*. We intend to provide enough insight for estimating the best combination of these parameters for the system designers.

7.5.1 System Scalability

We extensively study the system scalability as shown in Figure 13 by increasing number of chiplets in Figure 13b (132 nodes) as compared to Figure 9 (68 nodes). To compare the scalability with different size of chiplets, we also keep the total number of nodes same between Figure 13a (132 nodes) and Figure 13b (132 nodes) and contrast their zero load latency and throughput. Figure 13d shows a large system with large number of nodes per chiplet (total 272 nodes) with doubled number of boundaries in each chiplet. In all the configurations *RC* outperforms MTR, ITB, and VC-SEP in terms of throughput. Also in terms of zero load latency *RC* exhibits same or better than MTR and much better than ITB. This is because the detour caused by turn restrictions in MTR surpluses *rc_buffer* request delay in *RC*. For example, in Figure 13c, MTR has 2 extra hops than *RC*, which accounts for 17% more in average hops. We observe that the throughput difference reduces with the increase in the system size, as more nodes saturate the bisection bandwidth earlier.

We quantitatively show that with the increase in the chiplet size, overhead of OPIC does not hamper performance. In a 4×4 mesh with four boundaries, each boundary gets three requesters, and all of them get response from the boundary OPIC block in 2 cycle. However, in a 8×8 mesh network, maximum seven requester nodes can be connected as they are in one, or two hop distance from the boundary. In that case the furthest node from the boundary gets the response in total 6 cycles. Other nodes get response in much lesser time. Since the requests get registered in the next OPIC block, requester needs to send request only once. When we scale the number of nodes further we do not need to reconsider the setup time and hold time, as the amount of work needed to be done in one cycle will still be same. Only the number of cycles of getting response will increase with the increase in distance from the boundary

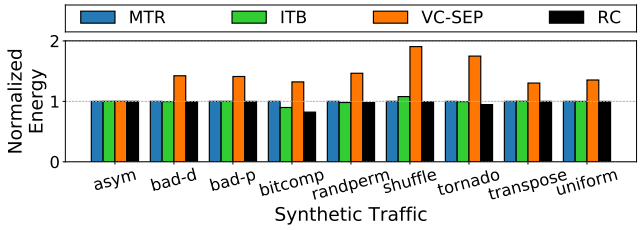


Fig. 16: Normalized energy for all the techniques, across all the synthetic traffic patterns.

router. However, it is worth noting that we opt for modular SoC design as we do not want to make large chips, rather want to put multiple small chiplets together to scale the system size. In Figure 13a and in Figure 13b, we show that system with multiple smaller chiplets in Figure 13a has better throughput than system with fewer large chiplets. One reason for the difference in lower throughput with large number of smaller chiplets is the number of boundary routers are same for each chiplet both the systems [5], resulting in more boundaries in total, in case of systems with smaller chiplets.

Going one step further we doubled the number of boundary routers for 8×8 chiplets keeping every other parameter values same. In Figure 14, the result shows that the average packet latency in case of *RC* improve significantly) over MTR by up to 17%. Figure 13(c) and 13(d) show results for SoCs with 4 boundaries and 8 boundaries per GPU chiplet, respectively. Average packet latency increased from 68 cycles to 80 cycles in case of MTR, and there is almost no change for our techniques. Our experimental results show that MTR travels more than 19.5% extra hops as compared to *RC*. This can be attributed to extra turn restrictions imposed by MTR. In addition, since the complexity of CDG analysis increased exponentially, we run the MTR algorithm for 7 days to explore the design space and pick the optimal result, which may not be the optimum turn restrictions for 8 boundary router setup. Interestingly, VC-SEP shows the zero load latency in this setup. However, the throughput suffer a lot because of low VC buffer utilization. In contrast, CDG analysis in 4 boundary setup takes only less than 2 hours to finish in one *intel core-i7* processor.

7.5.2 Sensitivity to rc_buffer Size and VC Size

In Figure 15, we show impact of *rc_buffer* size on network throughput, which is the saturation injection rate for SoC network with four 8×8 GPU chiplets and one 4×4 CPU chiplet (272 nodes), which shows similar trend for the smaller baseline setup with 4×4 GPU chiplets. We observe that increase of both *rc_buffer* size and the number of VCs have impact on the system throughput. With *rc_buffer* size of 1, we see hardly any throughput improvement with increasing VC sizes. The throughput improvement from single packet slots to two packet slots in *rc_buffer* is almost 2×. Also with 1-VC, increase in the *RC* size improves throughput marginally. Result shows that for all the VC sizes, *rc_buffer* size of 4 is good enough to provide achievable throughput, which is the case in infinite *rc_buffer*, where the OPIC delay is zero. In addition, in terms of throughput, the difference between 4-VC and 8-VC result is also not very significant. Even 2-VC result also shows a good trade-off between throughput and energy consumption.

7.6 Area and Energy Analysis

The hardware complexity and area overhead of *RC* is very minimal. As per our detailed synthesis report, in each router of size 49667.53 μm^2, OPIC logic consumes only $785.68 \mu m^2$ area, which is 1.6% of the router area. There are four *rc_buffer* in each chiplet, and each has 4 packet buffers, consuming $6.0424 \mu m^2$ in total. Area overhead and hardware complexity incurred is negligible as compared to the total chiplet area and complexity.

Since we focus on the network deadlock aspect in this work, we estimate only the network energy to compare between MTR, ITB, VC-SEP and *RC* using DSENT [31] and *rc_buffer* access energy from RTL simulation ($0.10425 pJ/flit/access$). The energy consumed by the wire connections in the OPIC tree are not significant. Figure 16 shows energy consumption of different techniques normalized to MTR under 0.013 packets/node/cycle injection rate for 100000 packets. It shows *RC*, MTR and ITB consume similar energy across all the synthetic traffic patterns. In contrast, VC-SEP consumes more energy due to low utilization of VCs, leading to longer simulation time that consumes more static energy. We expect *RC* to save more energy by reducing the static energy in high load since it sustains higher throughput.

8 Related Works

Deadlock avoidance mechanisms fan out in two distinct branches, namely VC and turn model based, and flow control based techniques. The first type either rely on turn restrictions, or on dedicated/ordered VC buffer for different traffic types/directions. On the other hand, flow control techniques either control the injection of packets, or ensures bubble in the buffer to avoid deadlocks. The state-of-the-art solves the new SoC deadlock issue using routing based turn restrictive technique while *RC* follows flow control based deadlock avoidance.

8.1 VC and Turn Model Based

Duato proposed escape-VC [6], a theory for deadlock freedom for routing with cyclic channel dependency. Duato's theory can be applied for both deadlock avoidance [32], [33] and deadlock prevention [18], [34] techniques. Idea of escape channel cannot be applied directly in modular SoC as the packets in the escape-VC must be propagated using a deterministic deadlock free algorithm, which cannot be guaranteed in a modular SoC. Recently Ebrahimi et al. [35] propose *EbDa* that provides exclusive sets of VCs to isolate traffics (say, intra-chiplet traffic, and inter-chiplet, or outbound traffic) to avoid deadlocks. However, VC separation leads to lower utilization and is shown less attractive in MTR [5], and we also find the same way.

Dally et al. [4] propose to use two or more VCs in order to avoid the cyclic channel dependencies. It ensures deadlock freedom by using total ordering of VCs. Even though this condition is sufficient to avoid the deadlock, it is not necessary [7]. Extra VCs result in increase in the router area

and energy consumption. Based on Dally's theory, a few other techniques have been proposed that use additional VCs [36], [37] to avoid deadlock. Another way to achieve strict order of reservation for the shared VCs is by imposing turn restrictions [38], [39] on the packet traversal.

8.2 Flow Control Based

For providing deadlock freedom, flow control techniques either regulate the injection [40] of the packets or allow a packet to go forward depending on the buffer occupancy [41] in the ring. The second concept is coined as bubble flow control by Puente et al. [42] and applied in torus network for the flow control in escape channel. This concept is being used in in-transit buffer for avoiding deadlock in k-ary n-cube torus network [43], and extended later for irregular off-chip network [10], worm-whole switching [44], torus cache-coherent NoCs [45].

Recently Ramrakhani et al. [7] propose *SPIN*, a synchronized flow control technique for deadlock prevention in flat network. It is very challenging to apply synchronized flow control in modular SoC, where the chiplets are designed independently, and connected through the interposer routers. Moreover, synchronization of packet movement among chiplets make the design very complicated.

9 Conclusions

Chiplet-based system integration on an active interposer is a scalable and economic solution for improving system performance. As deadlock freedom is one of the main concerns, we propose *RC*, a simple routing oblivious technique for modular SoCs. It completely protects the idea of modular design by providing total independence to the chiplet vendors, in terms of routing logic, topology, dimension, etc. The low load latency improvements of *RC* over MTR, ITB and VC-SEP are up to 15.49%, 19.17%, and 13.76% across different configurations for all the synthetic workloads, respectively. The throughput improvements achieved by *RC* over MTR, ITB, and VC-SEP are up to 56.34%, 12.12%, and 2.5×, respectively. In full system simulations for real workloads, we improve performance upto 20% as compared to state-of-the-art MTR. As part of future work, we want to investigate application-aware *OPIC* system, where critical packets can be prioritized in the *rc_buffer* for better system performance.

Acknowledgments

The authors would like to thank Swathi Gowda and Divya Siravara (during the work they were masters students in Texas A&M University, and currently both of them working in Apple Inc.) for helping in area and energy analysis by synthesizing the proposed modules.

References

[1] D. Green, "Common heterogeneous integration and ip reuse strategies (chips)," *DARPA DARPA-BAA-16-62*.

[2] A. Kannan, N. E. Jerger, and G. H. Loh, "Enabling interposer-based disintegration of multi-core processors," in *Microarchitecture (MICRO), 2015 48th Annual IEEE/ACM International Symposium on*, pp. 546–558, IEEE, 2015.

[3] S. S. Iyer, "Heterogeneous integration for performance and scaling," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 6, no. 7, pp. 973–982, 2016.

[4] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," 1988.

[5] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. Shoaib, N. Jerger, and G. H. Loh, "Modular routing design for chiplet-based systems," in *International Symposium on Computer Architecture*, IEEE, 2018.

[6] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE transactions on parallel and distributed systems*, vol. 4, no. 12, pp. 1320–1331, 1993.

[7] A. Ramrakhyani, P. Gratz, and T. Krishna, "Synchronized progress in interconnection network (SPIN): A new theory for deadlock freedom," in *International Symposium on Computer Architecture*, IEEE, 2018.

[8] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE micro*, vol. 27, no. 5, pp. 15–31, 2007.

[9] M. Garcia, E. Vallejo, R. Beivide, M. Odriozola, C. Camarero, M. Valero, J. Labarta, and C. Minkenberg, "On-the-fly adaptive routing in high-radix hierarchical networks," in *Parallel Processing (ICPP), 2012 41st International Conference on*, pp. 279–288, IEEE, 2012.

[10] J. Flich, P. Lopez, M. P. Malumbres, J. Duato, and T. Rokicki, "Applying in-transit buffers to boost the performance of networks with source routing," *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1134–1153, 2003.

[11] B. Black, "Die stacking is happening," in *Intl. Symp. on Microarchitecture, Davis, CA*, 2013.

[12] D. Stow, Y. Xie, T. Siddiqua, and G. H. Loh, "Cost-effective design of scalable high-performance systems using active and passive interposers," in *Proceedings of the 36th International Conference on Computer-Aided Design*, pp. 728–735, IEEE Press, 2017.

[13] N. E. Jerger, A. Kannan, Z. Li, and G. H. Loh, "Noc architectures for silicon interposer systems: Why pay for more wires when you can get them (from your interposer) for free?," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 458–470, IEEE, 2014.

[14] E. Beyne and A. Manna, "High-bandwidth chip-to-chip interfaces: 3d stacking, interposers and optical i/o," in *IMEC technology forum, Taiwan*, 2013.

[15] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-chip-module gpus for continued performance scalability," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 320–332, ACM, 2017.

[16] "3DIC." www.3dic.org/2.5D_integration.

[17] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," in *Proceedings of the 19th Annual International Symposium on Computer Architecture*, ISCA '92, pp. 278–287, 1992.

[18] J. Duato and T. M. Pinkston, "A general theory for deadlock-free adaptive routing using a mixed set of resources," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 12, pp. 1219–1235, 2001.

[19] D. Becker, D. William, K. Christoforos, and A. O. Oyekunle, "Efficient microarchitecture for network-on-chip routers." http://purl.stanford.edu/wr368td5072, 2012.

[20] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[21] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pp. 86–96, IEEE, 2013.

[22] AMD Research, "The AMD gem5 apu simulator: Modeling heterogeneous systems in gem5," in *In gem5 User Workshop"*, 2015.

[23] A. Hansson, N. Agarwal, A. Kolli, T. Wenisch, and A. N. Udipi, "Simulating dram controllers for future system architecture exploration," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pp. 201–210, IEEE, 2014.

[24] AMD, "HCC sample applications, github repository, 2016." https://github.com/ROCm-Developer-Tools/HCC-Example-Application. Accessed: November 27, 2018.

[25] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pp. 44–54, Ieee, 2009.

[26] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pp. 72–81, ACM, 2008.

[27] "SPEC CPU2017." https://www.spec.org/cpu2017.

[28] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[29] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das, "Application-aware prioritization mechanisms for on-chip networks," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 280–291, ACM, 2009.

[30] A. Kumar, L.-S. Peh, and N. K. Jha, "Token flow control," in *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, pp. 342–353, IEEE Computer Society, 2008.

[31] C. Sun, C.-H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "Dsent - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *Proceedings of the 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, NOCS '12, (Washington, DC, USA), pp. 201–210, IEEE Computer Society, 2012.

[32] W. J. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," *IEEE transactions on Parallel and Distributed Systems*, vol. 4, no. 4, pp. 466–475, 1993.

[33] P. Gratz, B. Grot, and S. W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pp. 203–214, IEEE, 2008.

[34] S. Ma, N. Enright Jerger, and Z. Wang, "Dbar: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 3, pp. 413–424, 2011.

[35] M. Ebrahimi and M. Daneshtalab, "Ebda: A new theory on design and verification of deadlock-free interconnection networks," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 703–715, 2017.

[36] G.-M. Chiu, "The odd-even turn model for adaptive routing," *IEEE Transactions on parallel and distributed systems*, vol. 11, no. 7, pp. 729–738, 2000.

[37] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*, pp. 77–88, IEEE, 2008.

[38] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacco, "Ariadne: Agnostic reconfiguration in a disconnected network environment," in *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pp. 298–309, IEEE, 2011.

[39] B. Fu, Y. Han, J. Ma, H. Li, and X. Li, "An abacus turn model for time/space-efficient reconfigurable routing," in *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 259–270, ACM, 2011.

[40] C. Carrion, R. Beivide, J. Gregorio, and F. Vallejo, "A flow control mechanism to avoid message deadlock in k-ary n-cube networks," in *High-Performance Computing, 1997. Proceedings. Fourth International Conference on*, pp. 322–329, IEEE, 1997.

[41] B. Roscoe, "Routing messages through networks: an exercise in deadlock avoidance," 1987.

[42] V. Puente, R. Beivide, J. A. Gregorio, J. M. Prellezo, J. Duato, and C. Izu, "Adaptive bubble router: a design to improve performance in torus networks," in *Proceedings of the 1999 International Conference on Parallel Processing*, pp. 58–67, Sep. 1999.

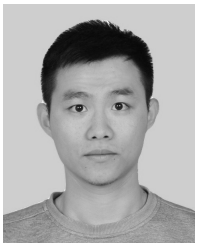
[43] V. Puente, C. Izu, R. Beivide, J. A. Gregorio, F. Vallejo, and J. Prellezo, "The adaptive bubble router," *Journal of Parallel and Distributed Computing*, vol. 61, no. 9, pp. 1180–1208, 2001.

[44] L. Chen and T. M. Pinkston, "Worm-bubble flow control," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pp. 366–377, IEEE, 2013.

[45] S. Ma, Z. Wang, Z. Liu, and N. E. Jerger, "Leaving one slot empty: Flit bubble flow control for torus cache-coherent nocs," *IEEE Transactions on Computers*, vol. 64, pp. 763–777, March 2015.

Pritam Majumder received the B.Tech degree in Computer Science and Engineering from WBUT, India, in 2011. He received his MS degree in Computer Science and Engineering from Indian Institute of Technology, Madras, in 2015. He is a Ph.D. student in the Department of Computer Science and Engineering, Texas A&M University. His research interests lie in the fields of computer architecture and systems, and machine learning. He is a student member of the ACM.

Sungkeun Kim received the B.Eng degree in Computer Science and Engineering from Kyungpook National University, Republic of Korea, in 2011. He is a Ph.D. student in the Department of Computer Science and Engineering, Texas A&M University. His research interests are the fields of computer architecture and systems, especially on networks-on-chip, memory systems and near-data processing, and hardware security. Before starting a Ph.D., he worked as a software engineer at Samsung Electronics, Suwon, Republic Korea.



Jiayi Huang received the B.Eng degree in information and communication engineering from Zhejiang University, China, in 2014. He is a Ph.D. student in the Department of Computer Science and Engineering, Texas A&M University. His research interests lie in the fields of computer architecture and systems, including networks-on-chip, approximate computing, memory system and near-data processing. He is a student member of the ACM and the IEEE Computer Society.

Ki Hwan Yum received the BS degree in mathematics from Seoul National University, Korea, in 1989, the MS degree in computer science and engineering from Pohang University of Science and Technology, Korea, in 1994, and the PhD degree in computer science and engineering from the Pennsylvania State University in 2002. From 1994 to 1997, he was a member of Technical Staff in Korea Telecom Research and Development Group. He is currently an research assistant professor in the Department of Computer Science and Engineering, Texas A&M University. His research interests include Computer Architecture, Parallel/Distributed Systems, Cluster Computing, and Performance Evaluation. He is a member of the IEEE, the IEEE Computer Society, and the ACM.

Eung Jung Kim received the BS degree in computer science from KAIST, Korea, the MS degree in computer science from Pohang University of Science and Technology, Korea, and the PhD degree from the Department of Computer Science and Engineering, Pennsylvania State University. She is an associate professor in the Department of Computer Science and Engineering, Texas A&M University. Her research interests include computer architecture, power efficient systems, parallel/distributed systems, cluster computing, security and sensor network. She worked as a member of technical staff in Korea Telecom for three years. She is a member of the IEEE Computer Society. More information about her research is available at http://faculty.cse.tamu.edu/ejkim.