Parallel Multipath Transmission for Burst Traffic Optimization in Point-to-Point NoCs

Hui Chen[1], Zihao Zhang[1], Peng Chen[1,2], Shien Zhu[1], Weichen Liu[1]
{hui.chen,liu}@ntu.edu.sg,{zzhang049,shien001}@e.ntu.edu.sg,chnp616@gmail.com
[1]School of Computer Science and Engineering, Nanyang Technological University, Singapore.
[2]College of Computer Science, Chongqing University, Chongqing, China.

ABSTRACT

Network-on-chip (NoC) is a promising solution to connect more than hundreds of processing elements (PEs). As the number of PEs increases, the high communication latency caused by the burst traffic hampers the speedup gained by computation acceleration. Although parallel multipath transmission is an effective method to reduce transmission latency, its advantages have not been fully exploited in previous works, especially for emerging point-to-point NoCs since: (i) Previous static message splitting strategy increases contentions when traffic loads are heavy, degrading NoC performance. (ii) Only limited shortest paths are chosen, ignoring other possible paths without contentions. (iii) The optimization of hardware that supports parallel multipath transmission is missing, resulting in additional overhead. Thus, we propose a software and hardware collaborated design to reduce latency in point-to-point NoCs through parallel multipath transmission. Specifically, we revise hardware design to support parallel multipath transmission efficiently. Moreover, we propose a reinforcement learning-based algorithm to decide when and how to split messages, and which path should be used according to traffic loads. Experiments show that our algorithm achieves a remarkable performance improvement (+12.1% ∼ +21.0%) when compared with the state-of-the-art dual-path algorithm. Also, our hardware decreases power and area consumption by 23.2% and 10.3% over the dual-path hardware.

1 INTRODUCTION

Network-on-Chip (NoC), as a promising solution for connecting more than hundreds of processing elements (PEs), provides high bandwidth by allowing transmitting messages in parallel. As the number of PEs increases, the high communication latency caused by the burst traffic erodes the speedup gained by parallelism and computation acceleration. Researchers have proposed their solutions to decrease NoC latency. From the hardware perspective, point-to-point NoCs [1, 3, 4, 13] transmits unconflicted flits to distant PEs using one cycle, targeting the end-to-end latency reduction. In [4], authors proposed a point-to-point NoC which supports arbitrary routing and eliminates contentions at intermediate routers by adding delay at the source. From the software perspective, mapping [15, 20] and routing [5, 6, 12] are two common ways to improve NoC performance. Although NoC resources, e.g., routers and links, are managed properly through mapping and routing, the parallelism of transmission is not fully utilized due to the unique path selection for a specific source-destination communication pair. In network communication, one communication pair can take more than one path, i.e., multipath, to transmit data to improve the transmission efficiency, inspiring researchers to apply multipath designs on NoCs.

In NoCs, initially, the multipath transmission has been proposed to reduce network congestion and traffic hotspots. Since most approaches just select one of the alternatives and send data sequentially, the high transmission latency is not fully alleviated. To further reduce latency, the simultaneous dual-path approach [21] is proposed. This approach sends data through XY and YX paths concurrently if source and destination nodes are not in the same row or column. However, statically splitting the data into two fixed paths cannot fully exploit the advantages of parallel multipath transmission due to the following reasons. (i) Static message splitting strategy without considering the NoC utilization state increases contentions when traffic loads are heavy, degrading NoC performance. Specifically, the dual-path algorithm indiscriminately injects up to twice of sub-messages into the NoC so that performance is dropped when encountering too much contentions. (ii) In dual-path design, only limited shortest paths, i.e., XY and YX, are chosen with the consideration that the end-to-end latency depends on the length of the transmission path, i.e., the number of hops, in traditional NoCs. Applying XY and YX paths limits the split degree to two and ignores other possible paths that can transmit data without contentions. However, in point-to-point NoCs, the end-to-end latency dependency on distance is reduced or eliminated, enlarging the routing design space of multipath. (iii) The optimization of hardware that supports parallel multipath transmission is missing in dual-path design, bringing large area and power overhead.

To address these problems, in this paper, we propose a software and hardware collaborated design to fully exploit the advantages of parallel multipath transmission in point-to-point NoCs. The main contributions of this paper are as follows:

1) We revised NoC router and network interface (NI) designs to support the parallel multipath transmission competently and re-order the packets from different ports with minimal overhead.
2) We proposed a parallel multipath algorithm, with which, the complex problems: when to split data transmission, how to split it, and which path should be taken to transmit data, are efficiently answered through the reinforcement learning-based approach to improve the NoC performance.

We performed extensive experiments on various real applications, which demonstrate that our algorithm achieves a remarkable performance improvement (+12.1% ∼ +21.0%) when compared with the state-of-the-art dual-path algorithm. Also, our router and NI designs decrease power and area consumption by 23.2% and 10.3% over the dual-path hardware.

2 BACKGROUND AND MOTIVATION

Point-to-point NoCs. In traditional NoCs, flits are forwarded hop by hop and the end-to-end latency is defined as:

$$L = (L_r + L_w) \times |\gamma| + (|p| - 1)\,/b + L_c$$

Where L_r and L_w are router-stage and link delay; $|p|$ and b are the number of flits and bandwidth; $|\gamma|$ is the length of path. Thus, the latency of a packet depends on $|\gamma|$. However, point-to-point NoCs [1, 3, 4, 13] are proposed to tailor a high performance NoC at run-time. By applying efficient pre-arbitration, links for one message are arbitrated and allocated before transmission, so that intermediate per-hop computation and arbitration are eliminated, enabling to forward data straight to the destination or HPC_{max} (maximum hops per cycle caused by propagation delay) router in on cycle. Thus, the latency for point-to-point NoCs is modified as:

$$L_s = L_{pre} + (L_r + L_w) \times \lceil |\gamma| / HPC_{max} \rceil + (|p| - 1)\,/b + L_c$$

Where L_{pre} is the pre-arbitration delay which depends on the NoC design. Thus the latency dependency on $|\gamma|$ is reduced or eliminated (if $|\gamma|$ is less than HPC_{max}). However, L_c is not reduced by the improved NoC infrastructure, thus, in point-to-point NoCs, the routing algorithm should focus on contention minimization.

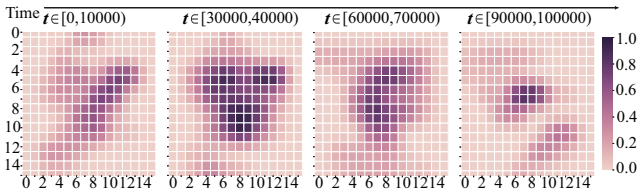

Figure 1: Router utilization for H264 using XY routing.

NoC Traffics. For real applications, when it runs on NoC-based many-core systems, the corresponding NoC traffics varies in time and space. In Figure 1, we show the 16×16 NoC router utilization state changes over time when executing the application H.264 using XY routing. We observe that there is a huge utilization variation for different routers over different time intervals. The traffic bursts in the center routers, in terms of space, and in the middle of the execution, in terms of time.

Multipath Transmission. To reduce network congestion and traffic hotspots, the multipath transmission strategy has been proposed. In [7–10], load balance is maintained by choosing an alternative contention-free path. In [10], the in-order problem is addressed using additional buffer and stall cycles, increasing energy and area consumption as well as end-to-end latency. Moreover, multipath transmission enhances the fault-tolerance guarantees, as discussed in [11, 14, 16]. However, these approaches just select one of the alternative paths and send data sequentially, resulting in that transmission latency is not fully alleviated. The simultaneous dual-path algorithm [21] is proposed targeting the shorter latency and higher bandwidth. This dual-path algorithm splits data statically and sends data through XY and YX paths.

Previous works have not exploited the benefits of parallel multipath transmission. Three so-far-unseen but urgent-to-solve problems still exist. In the following, we use simple examples to illustrate these problems. A given application is profiled and represented as a directed acyclic graph (DAG) as shown in Figure 2(a). The number inside the vertex is the execution time, and the number on the edge is the message size $|m|$, i.e., the number of packets. We apply the point-to-point NoC design supporting arbitrary routing proposed in [4]. Using NoC in [4], when contention occurs, the message with high priority is transmitted at first and the others are blocked at the source. In these examples, transmission from router 1 has a higher priority, the packet size is 1 flit, and the link bandwidth is 1 flit per cycle. The black nodes are taken by other tasks. Message splitting generates i ($2 \le i \le 4$) sub-messages sub_m from the message m. For unification, we use $i = 1$ to represent non-splitting strategy, in which $|sub_m|$ equals $|m|$. These sub-messages are sent simultaneously and the revised task graphs which include sub-messages are presented in Figure 2(b-d).

- **Question 1. When to split messages into sub-messages?**

Generally, when the router utilization is low, the multipath algorithm should split messages into as many parts as possible (up to 4) to utilize available links. By contrast, if the router utilization is high, we do not need to split messages. The dual-path transmission is shown in Figure 2(b). Since v_1 and v_3 are mapped to the same column, message from v_1 is not split. Finally, the schedule length is 85. However, without splitting, as shown in Figure 2(a), the total schedule length is 80, showing the importance of choosing the correct time to split messages.

- **Question 2. How many sub-messages should be split into? And how to decide the size of each sub-message?**

Splitting messages indiscriminately would increase the contention for NoC resources and reduce the performance gain. We use the term "**split degree**" to notate the number of sub-messages the original message is split into. We observe that the high split degree cannot guarantee the decreased schedule length through the previous example. Also, the size of the sub-message affects the schedule length. In Figure 2(c), we apply the same split degree as the dual-path method but the size of sub-messages are changed to 10 and 40, respectively. The final schedule length is decreased to 70 since less data is transmitted through the contention path.

- **Question 3. Which path should be chosen?**

By using the state-of-the-art point-to-point NoC infrastructure, the length of the path should not be limited since the non-minimal path can transmit data with the same latency as the minimal path. Thus, the non-shortest path selection lets more available links can be used to improve the NoC performance. In Figure 2(d), we split the messages using the same split degree as the dual-path algorithm, but the path of one sub-message from v_2 is changed. Since there is no link contention, the total schedule length is reduced to 60.

3 PARALLEL MULTIPATH HARDWARE

To implement parallel multipath transmission, two components need to be revised: **router** and **NI**. The 5-port crossbar router design which has a single port connected to the processor is limited to serial data transmission. We reviews router designs for point-to-point NoCs and find that in [4] the proposed router can send and receive data from different ports simultaneously. We adopt such design, as shown in Figure 3(b), to support our parallel multipath transmission. Using point-to-point NoCs, distance dependency is

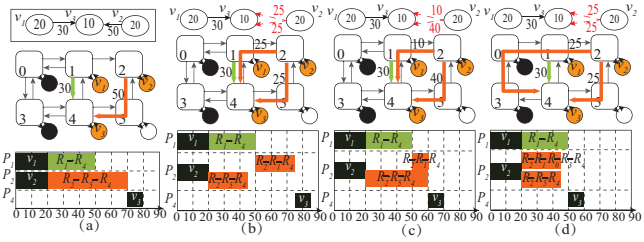

Figure 2: Motivation examples. (a) Non-splitting strategy; (b) Dual-path transmission mentioned in [21]; (c) Multi-path transmission with different sub-message size; (d) Multi-path transmission with non-shortest paths.

reduced or removed. However, data transmission through disparate paths may encounter different contentions, resulting in that packets are received out of the order. In dual-path design [21], the data is split in packet-level. One packet is sent through the odd-even pattern, which indicates the odd flits are sent through the XY path while the even flits are sent through the YX path. Based on the order information indicated in the header, at the destination, the re-ordering is conducted with the time complexity of $O(|m|)$ as well as the space complexity of $O(|m|)$. Such re-ordering leads to an additional delay to data transmission and undermines the benefits of parallel multipath transmission. Thus, we present a novel message splitting strategy and a new design of NI, as shown in Figure 3(c).

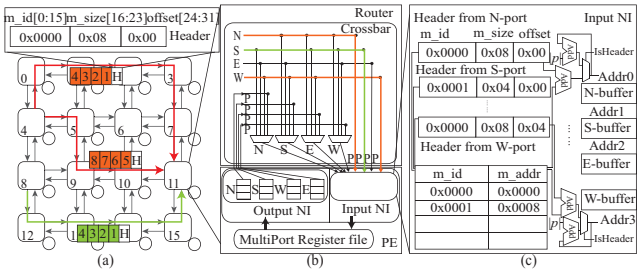

Figure 3: Illustration of hardware design. (a). Overview of data transmission; (b). Router design; (c) Input NI design.

Instead of splitting data in packet-level, in our design, the data is split in message-level, i.e., the message is split into multiple sub-messages and within each sub-message, packets are ordered. That is, if we split the message with the size of $|m|$ into two sub-messages, the first sub-message consists of packets from 0 to $\lceil |m|/2 \rceil$, and the other packets are included in the second sub-message. For example, the message from router 4 is split into two parts as shown in Figure 3(a). Such splitting can be easily implemented using a multiport register file. By computing the addresses of different sub-messages, we read the data from different ports of the register file and send them into the corresponding ports in parallel. Many multiport registers have been proposed in recent years, we apply the design mentioned in [22] which supports at least 12 read and 6 write ports (we only need 4 read and 4 write ports). Since within the sub-message, the packets are in order, only the order of the sub-message is needed during the re-ordering. Before the first packet of each sub-message, we add the header including the message-id *m_id*, message size *m_size*, and the offset of this sub-message, i.e., the start address of the sub-message in the whole message. Once receiving the header packet from any port, the input NI check whether the *m_id* is existed. If no *m_id* is matched, the PE allocates the memory for the whole message and stores the *m_id* and corresponding start address in the table as shown in Figure 3(c), otherwise, the input NI finds the start address based on the *m_id*. The address of memory is the summation of the start address and the sub-message offset. For example, in Figure 3(c), the initial address of the message received from the south port is its message address (0x0008) plus sub-message offset (0x00). If the received packet is not the header packet, the address is added by the packet size $|p|$. By using such design, once any port receives the packet, after address computation, the data can be written to the local register file, avoiding the high re-ordering delay.

4 PARALLEL MULTIPATH ALGORITHM

4.1 Algorithm overview

As mentioned before, to improve the NoC performance, our parallel multipath algorithm should answer three questions, i.e., when, how, and which. These three questions indicate different design dimensions including time and space, constructing a very large exploration space. Thus, although based on the task graph generated by profiling, our algorithm can be conducted offline, we need to provide an efficient algorithm to process complex applications.

Given a specific task graph, for every message, at first, we should decide if it needs to be split and how many parts it should be split into. This is a complex problem since the network state changes at almost every time point. Fortunately, in the mesh topology, the number of ports connected to the processor is maximally limited to four, i.e., north, south, west, and east. Thus, as a greedy strategy, we can find the best splitting degree by varying the splitting degree from 1 to 4. Moreover, for routers at the boundary, the number of available ports is further reduced. At most, up to $|\mathcal{E}|$ to the power of four different task graphs can be generated through splitting, where $|\mathcal{E}|$ is the number of edges in the task graph. We need to calculate the route for each sub-message efficiently since splitting increases the number of objects we need to process. Note that, although as mentioned before, in the optimal solution, the size of each sub-message may be unequal, in this work, we evenly split each message to get an approximate solution.

The backbone of our parallel multipath algorithm is described as: For each message, before splitting it, we check the ports of its source and destination routers. For a message, the maximum degree *md* we can benefit from the splitting is the less one between the number of available ports of source and destination routers. Then we try the splitting degree from 1 to *md*. For each splitting case, we compute paths for all sub-messages. After that, we evaluate the performance of such splitting using "**splitting indicator**" (we will introduce it in Sec. 4.3). We choose the splitting case with the best splitting indicator for this message and continue to process the next message. The order of message processing is breadth-first.

4.2 Path Computation

Finding paths for sub-messages, as an important part of the parallel multipath algorithm, affects the efficiency of our parallel multipath algorithm. As mentioned before, routing algorithm for point-to-point NoCs should focuses on contention minimization. The state-of-the-art routing algorithm proposed for point-to-point NoCs is shown in [5]. Its time complexity is $O(|\mathcal{E}|^2 \cdot |\mathcal{R}|^2)$, where $|\mathcal{R}|$ is the number of routers. Such time complexity is not acceptable to be integrated in our algorithm to process complex task graphs. Thus, reinforcement learning, which shows its high efficiency in solving various problems, is adopted as our path computation backbone. The basic reinforcement learning is modeled as:

- The set of environment and agent states S.
- An action function $A(s_t)$ includes all possible actions the agent can take in state s_t;
- The set of possible rewards R. r_t is the reward at t;
- A transition model $P_{a_t}(s_t, s_{t+1})$.

• **Environment.** In our design, the environment is the NoC system. Specifically, the link usage state $\mathcal{U}$ describes the links occupied by all sub-messages. $\mathcal{U}$ changes over time since the links are occupied and released in different time slots.

• **Agent.** For each sub-message, although it can be sent from the source to the destination within one cycle in point-to-point NoCs, during the route computation, we separate the route by routers. Specifically, in each router, the sub-message chooses one direction and latches the next router.

• **State and action.** In our design, the state is the router the sub-message has reached and the action is to select one direction to forward this sub-message.

• **Transmission model.** In A2C, actor is a policy-based function, and critic is a value-based function. Specifically, the actor generates the action a_t based on the state s_t while the critic evaluates the value of taking that action at that state.

Next, we demonstrate how the agent interacts with the environment and vice versa. At time step t, the agent observes state s_t and actor outputs an action a_t and generate a new state s_{t+1} and reward r_{t+1}. The critic evaluates the value of taking action a_t at state s_t. Then, the actor updates its parameters using a specific value function $Q(s, a)$ and generate the next action a_{t+1}. Based on these two steps, critic updates its parameters based on the gradient of value function $\nabla Q(s, a)$. However, $Q(s, a)$ based methods are hard to converge due the high variability. To decrease variability, advantage function $Ad(s, a)$ is defined by: $Ad(s, a) = Q(s, a) - V(s)$. Where the $V(s)$ is the average state value. The classic methods of advantage value estimation include the Monte Carlo method and temporal difference learning (TD learning). Since Monte Carlo methods need a large number of samples and enough learning time to make the model converge, we apply TD learning in this work. Using the TD estimation, advantage value $Ad(s, a)$ is estimated as: $Ad(s, a) = r + \gamma V(s') - V(s)$. Where γ is the discount-rate parameter. Advantage actor-critic (A2C) and asynchronous advantage actor-critic (A3C) are two state-of-the-art advantage-based algorithms. It was empirically found that A2C shows comparable performance to A3C and is more efficient [19].

Specifically, in state s_t, the sub-message has reached a router but has not reached the destination, it waits for the actor to tell which direction it should transmit next and then the agent executes this action to move to the next router. The state changes from s_t to s_{t+1}. After that, the critic evaluates the action according to the advantage value which depends on reward r, and then use advantage value to update the network parameters of actor and critic.

Algorithm 1: Reward computation

Input: Link usage state $\mathcal{U}$, sub-message sub_m, action a
Output: Reward R

```
1  if sub_m.start = INF then
2      if sub_m.src.endTime ≠ INF then
3          sub_m.start = sub_m.src.endTime ;
4          sub_m.end = sub_m.start + sub_m.L'_t ;
5      else
6          Return -INF;
7  initStart = sub_m.start;
8  R = 0;
9  while (1) do
10     U' = loadLinkState(U,[sub_m.start, sub_m.end]);
11     m' = checkContention(a, U');
12     if m' ≠ Φ then
13         newStart = Max(m'.end);
14         R = initStart - newStart;
15         sub_m.start = newStart ;
16         sub_m.end = sub_m.start + sub_m.L'_t ;
17     else
18         Return R;
```

4.3 Reward Computation

In our case, the reward is the opposite number of transmission time along the specific direction, i.e., if the sub-message takes the path with less transmission latency, a higher reward would be given. With the help of NoC simulator, such transmission latency can be simulated. However, this method is time-consuming and is hard to be integrated into our algorithm. Thus, we propose a simple method to estimate the transmission latency of a sub-message through the specific links. The reward computation is discussed in Algo. 1.

Given a sub-message sub_m and an action a, at first, we check if it is the first action. If yes, we initialize its time interval as shown in Line 1-6. Initially, the start time of sub_m is the end time of its source task. However, since for each task v, its start time is the maximum end time of all input messages and sub-messages, we cannot compute the end time of v if there is any message or sub-message with the unknown end time. The algorithm returns "-INF" as shown in Line 6. This unsettled sub-message will be inserted into the message list again so that we can process it in the future. If the start time is initialized, the initial end time can be estimated as $sub_m.start + sub_m.L'_t$, where $sub_m.L'_t = (L_r + L_w) \times |sub_m| \times \lceil |\gamma'| / HPC_{max} \rceil + |sub_m| \times (|p| - 1) / b$, $|\gamma'|$ refers to the existing path length. After that, we load the link usage state $\mathcal{U}$ of time interval $[sub_m.start, sub_m.end]$. In one state, one specific link corresponds to the action a. We check if this link is contented with any sub-messages. Without any contentions, the algorithm is terminated and the reward is returned. Otherwise, the

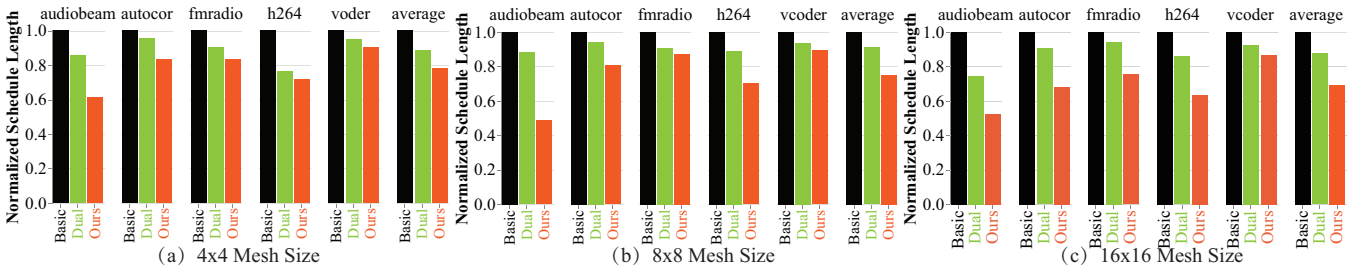

Figure 4: Normalized schedule Length comparison.

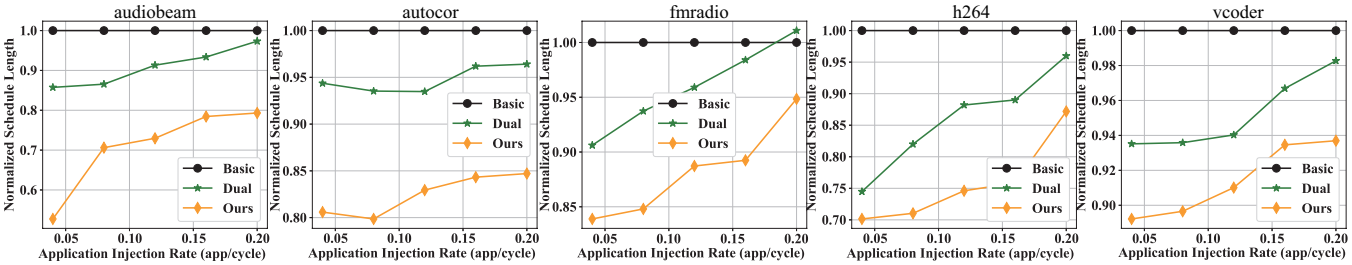

Figure 5: The schedule length comparison in terms of AIR.

sub-message should wait until this link is available as shown in Line 9-17. In this case, the reward is the difference between the original start time and the new start time, i.e., the inverse contention estimation. According to the reward generator algorithm, estimated time intervals for all sub-messages, i.e., $[sub_m.start, sub_m.end]$, are recorded. For each splitting case, we set the maximum $sub_m.end$ as its **splitting indicator**.

5 EXPERIMENT

Table 1: Benchmark description

Benchmark	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	Max In/Out	Introduction
Audiobeam	22	35	15/15	Audio beamformer
Autocor	12	18	8/8	Auto-correlation series
Fmradio	31	37	4/4	FM radio with equalizer
H264	51	71	11/12	Video compression standard
Vcoder	55	70	17/17	Channel voice coder

Table 2: Default Configurations of NoC

Parameter	Specification	Parameter	Specification
Topology	2D Mesh	NoC Size	8×8
HPC_{max}	8	Flit Width	128-bit
Technology	22 nm	Buffer Size	4 flits
Packet Size	4 flits	L1 Cache	Private,32KB
L2 Cache	Shared,512KB/bank	Frequency	1 GHz

In this section, we conduct a series of experiments to compare the performance of our algorithm with the basic XY and the state-of-the-art dual-path algorithm [21]. We notate these three algorithms as "ours", "basic", and "dual". As we mentioned before, "dual" needs to re-order packets at the destination with the additional latency overhead. We ignore such overhead in the algorithm comparison so that algorithm efficiency is fairly compared. However, area and power overhead are included in the hardware comparison. Since our algorithm are conducted offline based on the profiled task graphs, we generate task graphs according to streaming applications from StreamIt benchmark [18]. The description of the benchmark is listed in Table 1. Tasks are mapped using the algorithm proposed in [20]. We implement the point-to-point NoC design proposed in [4] in gem5 [2] with the revised NI design as mentioned before and its default configuration is shown in Table 2. In the overhead analysis, we compare the power and area for the basic hardware (router and NI), the hardware supporting dual-path, and our proposed hardware denoted as "basic", "dual", and "ours". The hardware simulation tool we used is DSENT [17] with the technology class of 22 nm.

• **Schedule Length.** We show the schedule length comparison on 4×4, 8×8, 16×16 system in Figure 4. What stands out in the figure is the high efficiency of our proposed solution. At least 22% time saving is observed compared with the basic algorithm. For the small mesh size, the performance of the three routing algorithms is similar due to its limited resources and inevitable contentions. With the mesh size increases, the improvement of our algorithm is more obvious, indicating that our algorithm utilizes the available communication resources properly. On average, our algorithm decreases 22.1%, 24.8%, and 30.9% total schedule length compared with the basic algorithm.

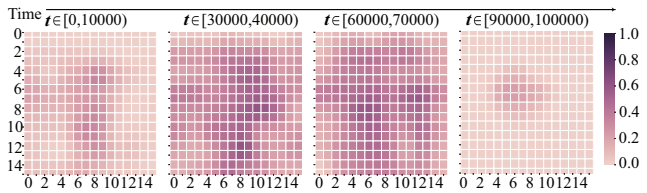

Figure 6: Router utilization for H264 using our algorithm.

• **Application Injection Rate.** The influence of application injection rate (AIR) is shown in Figure 5. It is apparent from this figure that our algorithm consistently performs better than the basic and dual-path solution. The result for fmradio is counterintuitive since the dual-path spends more time than the basic algorithm. The explanation for this phenomenon is that the dual-path algorithm split the message statically and causes frequent contentions, resulting in lower performance. As the figure shows, with the AIR increases, the advantage of "dual" over "basic" vanishes quickly. On the contrary, our approach splits messages reasonably, resulting in better performance even on high traffic loads.

• **Utilization.** To demonstrate the advantage of our approach, we present the router utilization of H264 in the same time interval of Figure 1, as shown in Figure 6. As we mentioned before, the method used in Figure 1 is the basic method while this figure is our proposed algorithm. Since the total schedule length of these two methods is different, the traffic loads of these two cases are different even in the same time interval. However, our approach shows a more balanced link usage than the basic method.

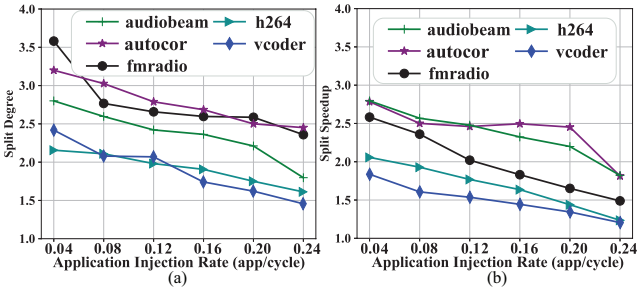

Figure 7: Illustration of results. (a) Split degree comparison; (b) Split speedup comparison.

• **Split Degree.** As we mentioned before, our approach splits messages according to the specific NoC link usage state. The correlation between the split degree and AIR is shown in Figure 7(a). In this figure, there is a clear trend of decreasing on the split degree with the increment of AIR, showing our approach adjusts to the changing traffic loads. We also observe that the split degree of our approach on autocor and fmradio is more than two, i.e., the maximum split degree of the dual-path algorithm. Such observation is consistent with the complexity of the application. For simple and serial transmission patterns, our algorithm tends to split it into more parts to take the advantage of available links, while for complex and parallel transmission patterns, our algorithm limits the split degree to decrease contentions.

• **Split Speedup.** To demonstrate the efficiency of splitting for a message m, we design a indicator named split speedup $ss(m)$ which is defined as: $ss(m) = L_m/Max(\boldsymbol{L}_{sub_m})$. The result of the average split speedup for all messages is shown in Figure 7(b). Since the contention is implied in the split speedup, it is a better indicator than the split degree and it inspires us to design the online parallel multipath algorithm in future works. The correlation between split degree and split speedup is interesting because although these two indicators have a similar trend in terms of AIR, the high split degree does not indicate the high split speedup. The explanation for this phenomenon is that the more sub-messages are split, the higher possibility of link contention occurs.

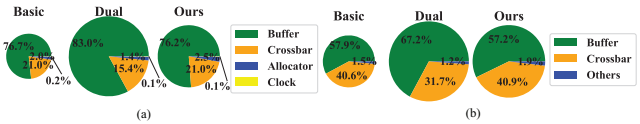

Figure 8: Overhead analysis. (a) Power Comparison; (b) Area Comparison.

• **Power and area.** Finally, we compare the power and area overhead of our hardware which enables the parallel message transmission. Figure 8 (a) and (b) demonstrate the power and area comparison of hardware (router and NI) for the basic, dual-path, and our proposed design. Since both the dual-path and our method need to send data to more than one port, the crossbar overhead on power and area are increased. Thanks to the efficient NI proposed in Sec. 3, our hardware decreases 30.8% of the buffer power consumption and 31.1% of buffer area compared with the dual-path method. Totally, our hardware decreases power and area consumption by 23.2% and 10.3% over the dual-path hardware.

6 CONCLUSION

In this work, besides mapping and routing, we proposes a software and hardware collaborated design to reduce transmission latency in point-to-point NoCs through the parallel multipath transmission. We present an algorithm to decide when and how to split messages and which path each transmission should use according to the NoC traffic loads. Moreover, we revised the NoC router design to inject and eject data from multiple ports simultaneously. Experiments show that our algorithm achieves a remarkable performance improvement (+12.1% ∼ +21.0%) when compared with the state-of-the-art dual-path algorithm. Also, our hardware decreases 23.2% power and 10.3% area consumption over the dual-path hardware.

ACKNOWLEDGMENTS

This work is partially supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (MoE2019-T2-1-071) and Tier 1 (MoE2019-T1-001-072), and Nanyang Technological University, Singapore, under its NAP (M4082282) and SUG (M4082087).

REFERENCES

[1] Asgarieh, Y. et al. 2019. Smart-hop arbitration request propagation: Avoiding quadratic arbitration complexity and false negatives in SMART NoCs. *TODAES* (2019), 1–25.
[2] Binkert, N. et al. 2011. The gem5 simulator. *SIGARCH* (2011), 1–7.
[3] Chen, C.H.O. et al. 2013. SMART: A single-cycle reconfigurable NoC for SoC applications. In *DATE*. 338–343.
[4] Chen, H. et al. 2020. ArSMART: An Improved SMART NoC Design Supporting Arbitrary-Turn Transmission. *arXiv preprint arXiv:2011.09261* (2020).
[5] Chen, P. et al. 2020. Contention Minimized Bypassing in SMART NoC. In *ASP-DAC*. 205–210.
[6] Chou, C.L. et al. 2008. Contention-aware application mapping for network-on-chip communication architectures. In *ICCD*. 164–169.
[7] Ding, X. et al. 2015. Bandwidth-based Application-Aware Multipath Routing for NoCs. In *CISIA*.
[8] Du, G. et al. 2015. MPCC: Multi-path routing packet connect circuit for network-on-chip. In *ASID*. 86–91.
[9] Morvarid, M. 2011. Dual Path Odd-Even Routing Algorithm for Network on Chips. Iccms (2011), 57–61.

[10] Murali, S. et al. 2006. A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. *DAC* (2006), 845–848.

[11] Murali, S. et al. 2007. A method for routing packets across multiple paths in NoCs with in-order delivery and fault-tolerance gaurantees. *VLSi DeSign* (2007).

[12] Palesi, M. et al. 2006. A methodology for design of application specific deadlock-free routing algorithms for NoC systems. *CODES+ISSS* (2006), 142–147.

[13] Pérez, I. et al. 2019. SMART++ reducing cost and improving efficiency of multi-hop bypass in NoC routers. In *NOCS*. 1–8.

[14] Shafiei, F. et al. 2018. Development of an adaptive multipath routing algorithm by examining the congestion and channel fault of one-hop nodes in network-on-chip. *ICCKE* (2018), 231–236.

[15] Singh, A.K. et al. 2010. Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms. *JSA* (2010).

[16] Stefan, R. et al. 2011. Enhancing the security of time-division-multiplexing networks-on-chip through the use of multipath routing. *ICPS* (2011), 57–62.

[17] Sun, C. et al. 2012. DSENT-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *NOCS*. 201–210.

[18] Thies, W. et al. 2010. An empirical characterization of stream programs and its implications for language and compiler design. In *PACT*. 365–376.

[19] Wu, Y. et al. 2017. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *arXiv preprint arXiv:1708.05144* (2017).

[20] Yang, L. et al. 2017. Task Mapping on SMART NoC: Contention Matters, Not the Distance. *DAC* Part 12828 (2017).

[21] Yang, Y.S. et al. 2018. SDPR: Improving Latency and Bandwidth in On-Chip Interconnect Through Simultaneous Dual-Path Routing. *TCAD* (2018), 545–558.

[22] Yantir, H.E. et al. 2013. Efficient implementations of multi-pumped multi-port register files in FPGAs. In *Euromicro DSD*. IEEE, 185–192.