Partial Order Based Non-Preemptive Communication Scheduling Towards Real-Time Networks-on-Chip

Peng Chen[1,2], Hui Chen[1], Jun Zhou[1], Di Liu[1],
Shiqing Li[1], Weichen Liu[1], Wanli Chang[3], Nan Guan[4]

[1]School of Computer Science and Engineering, Nanyang Technological University, Singapore.
[2]College of Computer Science, Chongqing University, Chongqing, China.
[3]Department of Computer Science, University of York, UK.
[4]Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR, China.

ABSTRACT

Due to the increasing performance requirement of cyber-physical systems, many-core processors with high parallelism are gaining wide utilization, where network-on-chip (NoC) is a prevalent choice for inter-core communication. Unfortunately, the contention on NoCs introduces large timing uncertainties, which complicates the response time estimation. To address this problem, for real-time applications modeled as a directed acyclic graph (DAG), we introduce DAG-Order, a partial order based time-predictable scheduling paradigm, resulting in real-time NoCs. Specifically, DAG-Order is built upon an existing single-cycle long-range traversal (SLT) NoC that is to simplify the process of validation and verification. Then, DAG-Order is proposed based on a dynamic scheduling approach, which jointly considers communication as well as computation workloads, and matches SLT NoC. DAG-Order achieves provably bound *safety* by enforcing certain partial order constraints among edges/vertices that eliminate the execution-timing anomaly during the runtime phase. Finally, an effective algorithm exploring for a proper schedule order is deployed to *tighten* the upper bound. Experimental results demonstrate that DAG-Order performs better than state-of-the-art scheduling approaches.

1 INTRODUCTION

Many-core platforms are increasingly attractive for cyber-physical systems, e.g., transportation, environmental monitoring, to meet the high parallel performance demand. Due to the properties of scalability and parallelism, network-on-chip (NoC) [13, 25] is a prevalent interconnection network for many-core platforms. One of the main challenges on real-time NoCs is to estimate the response time upper bounds in the design-time schedulability tests, which is related to both the underlying NoC architecture and the scheduling policy. In particular, the widely-used parallel pipeline wormhole NoCs [19, 40] are generally designed for average-case performance, thus demonstrating poor quality (i.e., "hard-to-get-right" analysis) on bounds estimation [38]. Till the present, the problem of designing a real-time NoC architecture and the corresponding scheduling policy with accurate bounds estimation (i.e., safe and tight) is still open.

To address such an open problem, rather than build the analysis to fit the wormhole NoCs, we employ another kind of NoC with single-cycle long-range traversal (e.g., [16, 20, 26, 29]), called SLT NoC, in which a higher degree of timing predictability can be provided. Specifically, on the priority-preemptive wormhole NoCs, flit-level preemption and pipeline traversal of different packets on a source-destination link will induce the multi-point progressive blocking (MPB) [40], which inevitably increases the timing unpredictability. By contrast, on SLT NoC, a flit is able to reach the destination in a single cycle via a pre-built SLT path, which cannot be shared with other flits, eliminating MPB latency of wormhole NoCs naturally. Off-the-shelf NoCs, such as SMART [26] and optical NoC [16], demonstrate single-cycle long-range traversal and can be employed as SLT NoC.

In this paper, to achieve timing predictability on NoCs, we propose a partial order based non-preemptive communication scheduling paradigm based on a kind of SLT NoC. To our knowledge, this is the first work to guarantee response time bound safety for non-preemptive, dynamic schedulers on SLT NoCs. The contributions are summarized as follows:

- We adopt off-the-shelf SLT NoC as the interconnection network for many-core platforms, with which the real-time analysis is simplified (e.g., unpredictable MPB latency of wormhole NoCs is eliminated naturally).
- We formally prove that the bound *safety* is guaranteed for any non-preemptive dynamic scheduling approach, as long as certain partial order constraints are enforced among DAG edges/vertices at runtime.
- We develop an effective algorithm to generate a proper partial schedule order, with which *tight* upper bounds are derived by "simulating" the runtime execution of the analyzed DAG task on SLT NoC.
- We conduct the experiments based on synthetic and realistic benchmarks, demonstrating the effectiveness of DAG-Order compared with the state of the art scheduling approaches in multiple dimensions.

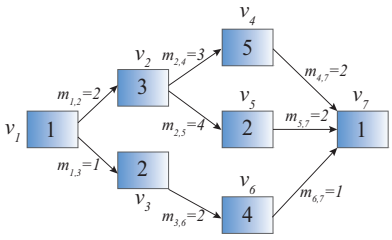


Figure 1: A parallel real-time task represented by a computation-communication DAG $\mathcal{G}$.

2 SYSTEM MODEL

2.1 NoC Platform

We consider the mesh-based NoC as the interconnection network of many-core platforms. An NoC consists of multiple components with computation/storage requirements, e.g., processor, memory, namely *processing element* (PE), which are interconnected by on-chip routers. To guarantee the parallel performance of applications, the cooperating computation-intensive subtasks are usually mapped to different PEs. Real-time messages are thus required among these PEs.

2.2 Real-Time Task Model

This paper considers a real-time parallel application represented by a directed acyclic graph (DAG) [5]. More generally, a parallel task can be represented by a computation-communication graph $\mathcal{G} = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is a set of computation vertices denoted by rectangles, and $E \subseteq V \times V$ is a set of directed communication edges, shown in Fig. 1. Each vertex v_i of $\mathcal{G}$ represents a segment of program (called *job*), and the number inside the vertex denotes the worst-case execution time c_i of the corresponding job. Each job is statically mapped to a PE on NoCs. As for each directed communication edge $m_{i,j} \in E$, it represents the data/control dependence between jobs v_i and v_j, and it has an associated worst-case message size also denoted by $m_{i,j}$. v_i is a predecessor job of v_j, and v_j is a successor job of v_i. That is, job v_j only starts execution after the transmission completion of all the messages from its predecessor jobs v_i.

The job that has no incoming edges and the job that has no outgoing edges is called *source* and *sink* job, respectively. Without loss of generality, we assume $\mathcal{G}$ has only one *source* job denoted by v_{src} and one *sink* job denoted by v_{sink} respectively, since a DAG task with multiple source/sink jobs can be trivially transformed to a DAG task exactly with only one source and one sink job by adding dummy jobs with zero execution time. Every task has a deadline, and we here consider constrained-deadline sporadic tasks (i.e., deadline of each task is *no greater than* its minimum inter-arrival time) with no conditional structures (e.g., *if-then-else* statement).

2.3 Real-Time Scheduling

The DAG task $\mathcal{G}$ is allocated a cluster of dedicated NoC region by using federated scheduling [4] that is an effective approach to provide empirically and theoretically real-time guarantee. The proper NoC region is chosen from the whole platform until the deadline can be guaranteed. During the job execution and message transmission, a job/message is assumed to be scheduled in a non-preemptive manner. Because of the pessimistic estimation of c_i and $m_{i,j}$, the actual execution time c'_i of each job v_i and the actual message size $m'_{i,j}$ of each message $m_{i,j}$ can be potentially less than their estimated values (formally $c'_i \leq c_i$, $m'_{i,j} \leq m_{i,j}$). To validate the schedulability of $\mathcal{G}$, let R denote the *estimated* response time upper bound which is the maximum duration time between the start time of the source job and the finish time of the sink job, and R' be the *actual* response time at runtime. $\mathcal{G}$ is *schedulable* if the bound R is no more than its deadline. By using a bound with high-quality timing predictability, the system safety can be guaranteed and resource allocation can be also well controlled to avoid resource over-provisioning.

Therefore, to guarantee the timing predictability, we aim to derive a *safe* and *tight* bound for a DAG task $\mathcal{G}$. Here, "*safe*" means the actual response time under any execution scenario is no more than the bound (i.e., $R' \leq R$), while "*tight*" means the bound R should be minimized as possible without the violation of the "*safe*" condition.

3 MOTIVATION

On real-time NoCs, the edges/vertices of an analyzed DAG task experience high contention in computation and communication resources due to concurrent execution. Thus, the response time of the DAG task also shows high timing variation, which is determined by the underlying NoC architecture and the corresponding scheduling policy. To make the response time of the DAG task on NoCs predictable, we introduce the off-the-shelf SLT NoC (e.g., SMART [26] / optical [16] NoCs) and illustrate the scheduling challenges.

3.1 SLT NoC Adoption

The widely-used pipeline wormhole NoCs [19, 40] are generally designed for best-effort traffics. Specifically, they demonstrate high performance in throughput and average latency, but often show significantly poor estimation quality of worst-case latency for real-time traffics. For example, the worst-case timing behavior is difficult to identify due to the unpredictable MPB phenomenon [40], and thus the derived latency bounds are usually either unsafe or overestimated.

To make the response time of DAG tasks predictable, rather than fix the erroneous patch of the existing analysis for wormhole NoCs, we employ another existing kind of NoCs [16, 20, 26, 29] featured with single-cycle long-range traversal. For simplicity, we call such NoCs as SLT NoC henceforth. To be specific, among the SLT NoCs, in the rest of this paper, we employ some terminology and techniques of the representative one, SMART NoC [26], which has been proved as a promising NoC technique for many-core systems.

SMART (single-cycle multi-hop asynchronous repeated traversal) [26] enables single-cycle long-range (up to HPC_{max}

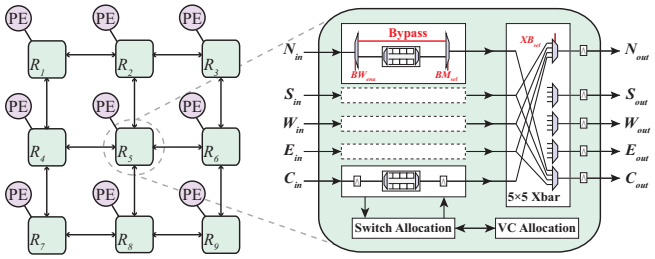
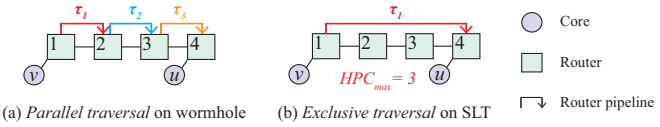


Figure 2: An illustrative example of SMART NoC [26] and SMART router micro-architecture.

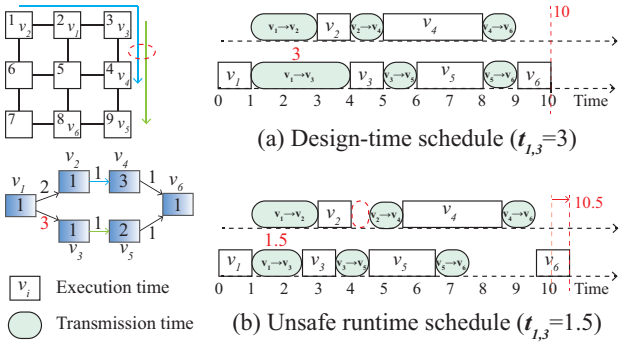
hops, where *HPC* denotes hops per cycle) traversal by dynamically building a direct bypass from the source/start to the destination. To achieve the long-range bypass traversal, low-swing clockless repeated link and bypass control are integrated into the traditional on-chip routers. Before packet transmission, the SLT path is dynamically pre-built by setting the control signals. In Fig. 2, BW_{ena} determines whether the incoming flits are buffered or bypassed; BM_{sel} sets the bypass path; and XB_{sel} connects the corresponding crossbar. As long as the SLT path is pre-built, SMART enables the exclusive transmission of the SLT path by avoiding buffering at intermediate routers. Recently, many research works (e.g., [3, 8, 10, 30, 41]) improve SMART performance and employ SMART techniques to build new works (e.g., [11, 21, 24, 27]).



(a) *Parallel traversal* on wormhole (b) *Exclusive traversal* on SLT

Figure 3: Traversal of τ_1 on wormhole and SLT NoCs.

Distinct from wormhole NoC of Fig. 3(a) with *parallel* traversal, a flit of τ_1 can *exclusively* traverse the Link $1 \rightarrow 4$ on SLT NoC of Fig. 3(b). Specifically, when $HPC_{max} = 3$, a flit of τ_1 can reach the destination (Router 4) in a single cycle. Then, the source-destination Link $1 \rightarrow 4$ seems to be considered as a physically single resource without being shared and logically "1-hop" traversal as long as the number of the hops of the SLT path is within HPC_{max}. The observations about the non-preemptive SLT NoC are twofold. First, compared with distributed priority-preemptive wormhole NoCs (e.g., [19, 40]), MPB is avoided since the flit-level preemption and pipeline traversal on a source-destination link are naturally eliminated. Second, in centralized priority-preemptive NoCs (e.g., [38]), the source-destination link is considered as a *logically* single resource without being shared, which leads to large resource waste at runtime. By contrast, the SLT paths within HPC_{max} [26] on SLT NoC are *naturally* and *physically* mutually-exclusive resources. Therefore, we motivate the adoption of SLT NoC for real-time many-core systems.



(a) Design-time schedule ($t_{1,3}$=3) (b) Unsafe runtime schedule ($t_{1,3}$=1.5)

Figure 4: Timing-anomaly execution under the dynamic scheduling with reduced transmission time.

3.2 Timing Anomaly

Dynamic scheduling [2] is widely utilized on multi-/many-core systems, where the scheduling decisions are made at runtime. In this paper, we consider the non-preemptive dynamic scheduling due to simple WCET analysis compared with the preemptive one. However, the non-preemption behaviors may be problematic for real-time systems, as the estimated bound can no longer be guaranteed due to timing anomalies. This phenomenon is observed by Graham et al. [15] when employing dynamic scheduling on multi-core systems. Specifically, the actual response time of a real-time task can probably be greater than the estimated bound under the same dynamic scheduling approach, when at least one of the following conditions occurs: (1) some inter-job precedence constraints are removed; (2) the number of cores is increased; (3) the execution time of some jobs is reduced. Thus, these conditions may make real-time tasks that are originally regarded as schedulable unschedulable, leading to the false-positive bound estimation.

We observe that the execution-timing anomaly phenomenon can also occur during dynamically scheduling the real-time messages in a non-preemptive manner on SLT NoC. We use an example to illustrate such a timing anomaly. As shown in Fig. 4, the DAG task is scheduled upon a 3×3 SLT NoC. Assume the assigned NoC region and the computation-communication DAG are given and fixed while only condition (3) occurs. Here, also assume the actual transmission time of the message $m_{1,3}$ is reduced to 1.5 at runtime, instead of WCET 3. The actual response time of the DAG task is counter-intuitively increased from the design-time bound 10 to actual response time 10.5 under the same dynamic scheduling approach (e.g., breadth-first scheduling). In this case, the originally *schedulable* real-time task can be potentially *unschedulable* if the deadline is set as 10.

The recent work [9] has addressed the timing anomalies for multi-cores, but their approach cannot be trivially extended from multi-core to NoC scheduling due to direct and indirect contention in communication resources, showing pessimistic runtime performance. Besides, when considering the joint scheduling of computation and communication tasks, all of the possible runtime execution scenarios can be exponential.

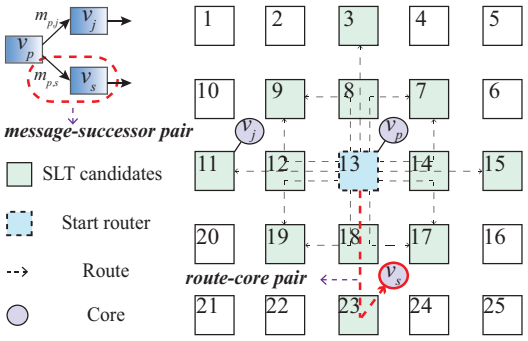


Figure 5: An illustrative example of task and resource pair on 2D-mesh SLT NoC ($HPC_{max} = 2$).

Deriving the optimal bound by enumerating all of the runtime schedules can be computationally infeasible.

4 BOUND SAFETY GUARANTEE

In this section, we guarantee the bound *safety* by adopting SLT NoC to eliminate the unpredictable MPB, and proposing certain order constraints among DAG edges/vertices at runtime to eliminate timing anomalies.

4.1 Preliminaries

To facilitate analysis of computation and communication workloads simultaneously, we develop task and resource abstractions, i.e., *route-core/resource pair* from the SLT NoC, *message-successor/task pair* from the DAG structure. It means that each *task pair* has to simultaneously use a subset of the given NoC links and a core as a *resource pair*. Assume the assigned SLT NoC with identical cores is denoted by π.

Definition 1 (Route-Core Pair). *On SLT NoC π, a route-core pair $\mathcal{R}_{p,\lambda} = \langle r_{p\to s}, P_s\rangle$ is a pair of a given route $r_{p\to s}$ from core P_p to core P_s, and the core P_s, where $\mathcal{R}_{p,\lambda}$ is the λ^{th} route-core pair candidate of the predecessor job v_p.*

In Fig. 5, for simplicity, only a part of cores are shown, and SLT candidates refer to the candidates that can be reached via a single-cycle long-range traversal path from the router R_{13}. Assuming v_p is mapped to P_{13}, $\mathcal{R}_{p,\lambda} = \langle R_{13} \to R_{18} \to R_{23}, P_{23}\rangle$ is the λ^{th} *route-core pair* of v_p. We abstractly call such a *route-core pair* as resource pair. For an analyzed resource pair $\mathcal{R}_{p,\lambda} = \langle r_{p\to s}, P_s\rangle$, the number of intermediate routers of the route $r_{p\to s}$ is denoted by $|\mathcal{R}_{p,\lambda}|$.

Definition 2 (Message-Successor Pair). *For $\mathcal{G} = (V, E)$, a message-successor pair $\langle m_{p,s}, v_s\rangle$ is a pair of inter-job message $m_{p,s}$ and successor job v_s, satisfying $m_{p,s} \in E$.*

For instance, in Fig. 5, $\langle m_{p,s}, v_s\rangle$ is a *message-successor pair* denoted by τ_i, which is also abstractly called task pair. In Fig. 1, the task graph $\mathcal{G}$ can be divided into task pairs $\{\langle null, v_1\rangle, \langle m_{1,2}, v_2\rangle, ..., \langle m_{6,7}, null\rangle, \langle m_{4,7}, v_7\rangle\}$. Note that, the task $\mathcal{G} = (V, E)$ can generate $|E| + 1$ task pairs, and the synchronization job (e.g., v_7 in Fig. 1) is only attached to the latest incoming message. For task pair $\tau_i = \langle m_{p,s}, v_s\rangle$ and resource pair $\mathcal{R}_{p,\lambda} = \langle r_{p\to s}, P_s\rangle$, v_s can be allocated to P_s and $m_{p,s}$ can be transmitted to P_s via the route $r_{p\to s}$. $\mathcal{R}_{p,\lambda}$, which is assigned to τ_i, can be simply called $\mathcal{R}_i$. For τ_i, let s'_i and s_i be the execution start time points, f'_i and f_i be the finish time points, w'_i and w_i ($w'_i \le w_i$) be the effective execution time at runtime and design time, respectively. The effective execution time denotes the total time of the message transmission and job execution for a task pair.

Table 1: Notations used throughout the paper.

Notation	Definition
$\mathcal{G}$	An analyzed real-time DAG task
$v_i/v_{src}/v_{sink}$	i^{th}/*source*/*sink* job of $\mathcal{G}$
$m_{i,j}$	A message from v_i to v_j
π	An SLT NoC
R'	Actual response time of $\mathcal{G}$
R	Estimated response time bound of $\mathcal{G}$
$\tau_i/\tau_1/\tau_{\lvert E\rvert+1}$	i^{th}/*first*/*last* message-successor/task pair
$\mathcal{R}_i/\mathcal{R}_{p,\lambda}$	Route-core/resource pair assigned to τ_i
Conf	Task-to-resource pair configuration of $\mathcal{G}$
$\mathcal{M}_{\tau_i\to\mathcal{R}_i}$	Allocation of τ_i to $\mathcal{R}_i$
s'_i/s_i	Start time of τ_i at runtime/design time
w'_i/w_i	Execution time of task pair τ_i
f'_i/f_i	Finish time of τ_i at runtime/design time
Sch'	Runtime schedule order of task pairs
Sch	Design-time schedule order of task pairs
$\mathsf{dc}(\tau_i)$	Direct-contention task set of τ_i
$\mathsf{ic}(\tau_i)$	Indirect-contention task set of τ_i
$\mathsf{cf}(\tau_i)$	Contention-free task set of τ_i
$\mathcal{L}(\tau_i)$	The length of remaining critical path of τ_i
HPC_{max}	The maximum bypass hops on SLT NoC

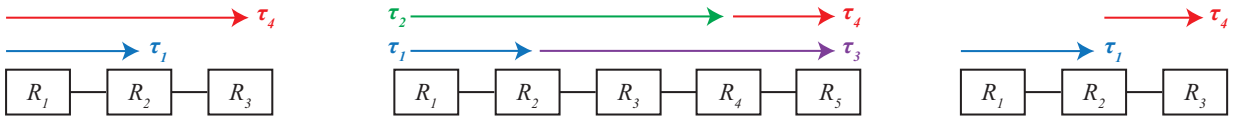
Before a DAG task $\mathcal{G}$ starts execution on SLT NoC π, the configuration information, such as the allocation of task pair to resource pair, should be given. In the following, we define the configuration information.

Definition 3 (Task-to-Resource Configuration). *For the real-time DAG task $\mathcal{G} = (V, E)$ running on SLT NoC π, the task-to-resource configuration is defined as*

$$Conf = \{\mathcal{M}_{\tau_1\to\mathcal{R}_1}, ..., \mathcal{M}_{\tau_i\to\mathcal{R}_i}, ..., \mathcal{M}_{\tau_{|E|+1}\to\mathcal{R}_{|E|+1}}\}.$$

where $\mathcal{M}_{\tau_i\to\mathcal{R}_i}$ refers to the allocation of the task pair τ_i to the resource pair $\mathcal{R}_i$.

To validate the schedulability of real-time tasks on SLT NoC, the upper bound R must be compared with the specified deadline. In this paper, assume a DAG task $\mathcal{G}$ is scheduled by a certain non-preemptive dynamic scheduling and configured by *Conf*, the bound R is derived by "simulating" such dynamic scheduling, assuming the jobs/inter-job messages are executed/transmitted for their WCETs. Specifically, in Fig. 4(a), the simulated bound of $\mathcal{G}$ is $R = 10$. Intuitively, under the same scheduling policy, the actual response time R' at runtime is within the scope of the bound



(a) Direct contention relationship (b) Indirect contention relationship (c) Contention-free relationship

Figure 6: The contention relationship between τ_1 and the analyzed task pair τ_4.

R, formally $R' \leq R$, since some jobs/messages may be executed/transmitted for less than their WCETs. The frequent notations are summarized in Table 1.

4.2 Safety Guarantee

In the previous subsection, the estimated bound R is derived by simulating the runtime execution. However, this bound can be potentially unsafe due to the variable execution scenario. To uniquely identify a particular runtime execution scenario of the real-time DAG task $\mathcal{G}$, we introduce the following definition of *schedule order* of task pairs.

Definition 4 (Schedule Order). *For the real-time DAG task $\mathcal{G} = (V, E)$ running on SLT NoC π, the schedule order Sch of $\mathcal{G}$ is the task pair sequence by scheduling points, where the task pairs start execution, in ascending order.*

$$Sch = \{\tau_1 \rightsquigarrow \tau_2 \rightsquigarrow \tau_3 \rightsquigarrow \ldots \rightsquigarrow \tau_i \rightsquigarrow \ldots \rightsquigarrow \tau_{|E|+1}\}.$$

As an example, in Fig. 4, the real-time DAG task $\mathcal{G}$ can be divided into task pairs $\{\langle null, v_1\rangle, \langle m_{1,2}, v_2\rangle, \langle m_{1,3}, v_3\rangle, \langle m_{2,4}, v_4\rangle, \langle m_{3,5}, v_5\rangle, \langle m_{4,6}, null\rangle, \langle m_{5,6}, v_6\rangle\}$, denoted as τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_7, respectively. In Fig. 4(a), the design-time schedule order Sch is as follows.

$$Sch = \{\tau_1 \rightsquigarrow \tau_2 \rightsquigarrow \tau_3 \rightsquigarrow \tau_4 \rightsquigarrow \tau_5 \rightsquigarrow \tau_6 \rightsquigarrow \tau_7\}.$$

Specifically, "$\tau_2 \rightsquigarrow \tau_3$" means that τ_2 starts execution *before or not later than* τ_3. The corresponding estimated bound R of the task $\mathcal{G}$ in Fig. 4(a) is 10. While in Fig. 4(b), the runtime schedule order is as follows.

$$Sch' = \{\tau_1 \rightsquigarrow \tau_2 \rightsquigarrow \tau_3 \rightsquigarrow \tau_5 \rightsquigarrow \tau_4 \rightsquigarrow \tau_7 \rightsquigarrow \tau_6\}.$$

The actual response time at runtime is $R' = 10.5$ in Fig. 4(b), which is more than the estimated bound $R = 10$. Thus, the bound R is unsafe, and the DAG task $\mathcal{G}$ is unschedulable if the deadline is set as 10, leading to false positive estimation. The observations are twofold:

- When the DAG task $\mathcal{G}$ is re-scheduled at runtime, some task pairs may not execute for their WCETs. The earlier execution behavior changes the design-time schedule order that is used to obtain the estimated bound. Specifically, in Fig. 4, when the message transmission time of $\tau_3 = \langle m_{1,3}, v_3\rangle$ is reduced from the designed WCET 3 to actual transmission time 1.5, the sub-schedule order is changed from $(\tau_4 \rightsquigarrow \tau_5 \rightsquigarrow \tau_6 \rightsquigarrow \tau_7) \subseteq Sch$ to $(\tau_5 \rightsquigarrow \tau_4 \rightsquigarrow \tau_7 \rightsquigarrow \tau_6) \subseteq Sch'$.
- Besides, dynamic scheduling may behave differently in the same situation, even if the task pairs are executed for their WCETs. For example, assuming a DAG task is scheduled by the breadth-first scheduling, the decision at a scheduling point to choose which vertex from the ready queue (including the DAG vertices in the same layer) to be scheduled first may be different.

Therefore, the fundamental reason for the timing-anomaly execution is the change of schedule order. Given the task-to-resource configuration *Conf* and any non-preemptive dynamic scheduling approach, we present a schedule order constraint *Sch* among task pairs to guarantee safe scheduling. Specifically, a feasible solution is to consistently keep the same order of task pairs at design time and runtime. Before that, inspired by the work [35] on priority-preemptive wormhole NoCs, we also introduce the following definitions related to the spatial relationship between task pairs, including direct, indirect contention, and contention-free relationships.

Definition 5 (Direct Contention). *For the analyzed τ_i and a task pair $\tau_j (\forall j \in \{1, 2, ..., |E|+1\} \wedge (i \neq j))$, if*

$$(\mathcal{M}_{\tau_i \to \mathcal{R}_i}) \wedge (\mathcal{M}_{\tau_j \to \mathcal{R}_j}) \wedge (\mathcal{R}_i \cap \mathcal{R}_j \neq \emptyset) \wedge (\tau_j \rightsquigarrow \tau_i)$$

then, τ_i suffers the direct contention from τ_j. Such direct contention is denoted by $\tau_j \in \mathsf{dc}(\tau_i)$, where $\mathsf{dc}(\tau_i)$ denotes the set of task pairs that have direct contention with τ_i.

Example 1. *In Fig. 6(a), assume $\mathcal{R}_1 \cap \mathcal{R}_4 \neq \emptyset$ and $\tau_1 \rightsquigarrow \tau_4$, thus τ_4 suffers the direct contention (formally $\tau_1 \in \mathsf{dc}(\tau_4)$) and blocking from τ_1 in the worst case.*

Definition 6 (Indirect Contention). *For the analyzed τ_i and a task pair $\tau_j (\forall j \in \{1, 2, ..., |E|+1\} \wedge (i \neq j))$, if*

$$(\mathcal{M}_{\tau_i \to \mathcal{R}_i}) \wedge (\mathcal{M}_{\tau_j \to \mathcal{R}_j}) \wedge$$
$$(\tau_j \notin \mathsf{dc}(\tau_i)) \wedge (\exists \tau_k \in \mathsf{dc}(\tau_i), \tau_j \in \mathsf{dc}(\tau_k) \cup \mathsf{ic}(\tau_k))$$

then, τ_i suffers the indirect contention from τ_j. Such indirect contention is denoted by $\tau_j \in \mathsf{ic}(\tau_i)$, where $\mathsf{ic}(\tau_i)$ denotes the set of task pairs that have indirect contention with τ_i via an(multiple) intermediate task pair(s) τ_k.

Example 2. *In Fig. 6(b), τ_1, $\tau_2 \notin \mathsf{dc}(\tau_4)$. Assume $\exists \tau_3 \in \mathsf{dc}(\tau_4)$, $\mathsf{dc}(\tau_3) = \{\tau_2\}$, then $\tau_2 \in \mathsf{ic}(\tau_4)$. Similarly, $\tau_1 \in \mathsf{ic}(\tau_3)$. By definition, $\mathsf{ic}(\tau_4) = \mathsf{dc}(\tau_3) \cup \mathsf{ic}(\tau_3) = \{\tau_1, \tau_2\}$. τ_4 suffers the indirect contention and thus blocking from τ_1 and τ_2 via intermediate τ_3 in the worst case.*

Definition 7 (Contention Free). *For the analyzed τ_i and a task pair $\tau_j (\forall j \in \{1, 2, ..., |E|+1\} \wedge (i \neq j))$, if*

$$(\mathcal{M}_{\tau_i \to \mathcal{R}_i}) \wedge (\mathcal{M}_{\tau_j \to \mathcal{R}_j}) \wedge$$
$$(\tau_j \notin \mathsf{dc}(\tau_i)) \wedge (\tau_j \notin \mathsf{ic}(\tau_i))$$

then, τ_i suffers no contention from τ_j. Such contention free relationship is denoted by $\tau_j \in \mathsf{cf}(\tau_i)$, where $\mathsf{cf}(\tau_i)$ denotes the set of task pairs that are free of contention with τ_i.

EXAMPLE 3. *In Fig. 6(c), assume* $\mathsf{dc}(\tau_4) = \emptyset$ *and* $\mathsf{ic}(\tau_4) = \emptyset$. *Thus,* $\tau_1 \notin \mathsf{dc}(\tau_4)$ *and* $\tau_1 \notin \mathsf{ic}(\tau_4)$. τ_4 *suffers no contention and blocking from* τ_1.

To granularly analyze the resource contention, we divide all of the task pairs into multiple groups according to the spatial relationships defined above. Without loss of generality, τ_i and τ_j $(\forall i, j \in \{1, 2, ..., |E|+1\}(i \neq j)$ are mapped to the same group if $\tau_j \in \mathsf{dc}(\tau_i) \cup \mathsf{ic}(\tau_i)$ or $\tau_i \in \mathsf{dc}(\tau_j) \cup \mathsf{ic}(\tau_j)$. For example in Fig. 6(b), τ_1, τ_2, τ_3 and τ_4 should belong to the same group due to direct or indirect contention relationship; while in Fig. 6(c), τ_1 and τ_4 belong to different groups due to absence of direct and indirect contention. Assume all of the task pairs are divided into χ groups, i.e., $group_1$, $group_2$, ..., $group_\chi$. And the schedule order constraint in $group_\chi$ is denoted by $sch(group_\chi)$. Therefore, the total schedule order consists of several independent partial orders, i.e., $Sch' = sch(group_1) \cup sch(group_2) \cup ... \cup sch(group_\chi)$. Let R' denote the actual response time of the DAG task $\mathcal{G}$ enforced with the schedule order constraint Sch'.

LEMMA 1. *If* $(\tau_j \rightsquigarrow \tau_i) \subseteq sch(group_\mu)$ *and* $(\tau_k \rightsquigarrow \tau_j) \subseteq sch(group_\mu)$, $(\tau_k \rightsquigarrow \tau_j \rightsquigarrow \tau_i) \subseteq sch(group_\mu)$.

PROOF. By Definition 4, "$(\tau_j \rightsquigarrow \tau_i) \subseteq sch(group_\mu)$" means that τ_j starts execution *before or not later than* τ_i in a certain schedule order $sch(group_\mu)$ (e.g., $s_j \leq s_i$). According to $\tau_j \rightsquigarrow \tau_i$ and $\tau_k \rightsquigarrow \tau_j$, the start times can be sorted in ascending order with $s_k \leq s_j \leq s_i$. Therefore, $(\tau_k \rightsquigarrow \tau_j \rightsquigarrow \tau_i) \subseteq sch(group_\mu)$ and the lemma is true. □

THEOREM 1. *For a DAG task* $\mathcal{G}$, *scheduled by any non-preemptive dynamic scheduling and configured by Conf , if*

$$Sch' = sch(group_1) \cup sch(group_2) \cup ... \cup sch(group_\chi)$$

where $sch(group_\mu) \subseteq Sch(\forall \mu \in \{1, 2, ..., \chi\})$.
then

$$R' \leq R$$

PROOF. Let s_i and s'_i be the start time points, f_i and f'_i be the finish time points, and w_i and w'_i $(w'_i \leq w_i)$ be the effective execution time of task pair τ_i at design time and runtime, respectively. To prove the design-time estimated bound is safe, it is sufficient to prove the runtime finish time $f'_{|E|+1}$ of the $(|E|+1)^{th}/last$ task pair is no more than that $(f_{|E|+1})$ of the design-time bound estimation, since $R' = f'_{|E|+1} + \mathcal{C}'_{ctrl}$ and $R = f_{|E|+1} + \mathcal{C}_{ctrl}$:

$$f'_{|E|+1} + \mathcal{C}'_{ctrl} \leq f_{|E|+1} + \mathcal{C}_{ctrl} \Rightarrow f'_{|E|+1} \leq f_{|E|+1}.$$

As described, $\mathcal{C}_{ctrl}$ and $\mathcal{C}'_{ctrl}$ are the worst-case and actual time offset to conduct task pair scheduling and control operations $(\mathcal{C}'_{ctrl} \leq \mathcal{C}_{ctrl})$, respectively. We use the mathematical induction to prove $f'_i \leq f_i$ in the following.

Base Case: When $i = 1$, since the first task pair τ_1 starts execution at time 0 at the design-time and runtime phases, $s'_1 = s_1 = 0$. Then $s'_1 + w'_1 \leq s_1 + w_1$, namely $f'_1 \leq f_1$.

Inductive Step: Assume $f'_j \leq f_j (j = 1, 2, ..., i-1)$, we will prove $f'_i \leq f_i$ in the following.

The analyzed task pair τ_i will be dispatched if the following conditions are met simultaneously: (i) precedence constraint in the DAG structure, i.e., all of the predecessor task pairs finish execution; (ii) the assigned resource pair $\mathcal{R}_i$ is available; (iii) proposed schedule order constraint, i.e., Sch'. We prove this theorem from the following steps:

For the condition (i), let τ_p $(p \leq i-1)$ be the last predecessor of τ_i in *Sch*, meaning τ_i can be executed only after the execution completion of τ_p, formally $f_p \leq s_i$. At the scheduling point s_i at runtime, all of the predecessors must have finished execution, since $f'_p \leq f_p \leq s_i$ by assumption.

For the condition (ii), let τ_k $(k \leq i-1)$ be the last task pair that fully/partly shares the resource pair $\mathcal{R}_i$ in *Sch*. $\mathcal{R}_i$ is free at s_i at runtime since $f'_k \leq f_k \leq s_i$.

For the condition (iii), given any task pair τ_h $(h \leq i-1)$ and assume $\tau_i \in group_g$, there are two cases to be considered according to the spatial relationship between τ_h and τ_i.

(1) $\tau_h \in group_g$. This means τ_h contends with τ_i directly or indirectly, meeting $\tau_h \rightsquigarrow \tau_i \subseteq sch(group_g)$. τ_h has started/finished execution at s_i when enforced with the design-time schedule order $sch(group_g)$.
(2) $\tau_h \notin group_g$. This means τ_h is free of contention with τ_i. No order is enforced between τ_h and τ_i.

Therefore, the conditions (i-iii) are met simultaneously at s_i at runtime. According to the greed of dynamic scheduling, τ_i is dispatched and starts execution *before or not later than* s_i at runtime, namely $s'_i \leq s_i$; we get $f'_i \leq f_i$ since $f'_i = s'_i + w'_i$, $f_i = s_i + w_i$ and $w'_i \leq w_i$. Hence, $f'_i \leq f_i$ is met under the schedule order constraint for any dynamic scheduling policy. When $i = |E|+1$, the theorem is proved.

□

Following Theorem 1, by using the consistent schedule order at design time and runtime, i.e., Sch', the timing-anomaly-free execution can be guaranteed, and thus the estimated response time bound is safe for any non-preemptive dynamic scheduling approach (formally $R' \leq R$).

5 TIGHT BOUND EXPLORATION

In Sec. 4, when an analyzed DAG task $\mathcal{G}$ is scheduled by any non-preemptive dynamic scheduling approach with the task-to-resource configuration *Conf*, the timing-anomaly-free execution is eliminated as long as the schedule order constraints *Sch* are enforced among task pairs at runtime. Thus, a *safe* bound is guaranteed for any *Conf* and *Sch*. To further derive a *tight* bound, we propose an effective algorithm to explore for proper configurations *Conf* and *Sch*.

Since the design space can be exponentially increased with the increasing of DAG size, deriving the optimal *Conf* (i.e., job-to-core mapping, message routing) is an NP-Hard problem [23]. Besides, the DAG task $\mathcal{G}$ may generate an exponential number of orders (i.e., topological order) when scheduled by a non-preemptive dynamic scheduling approach. Thus, deriving the tightest upper bound by finding the optimal *Conf* and *Sch* is computationally infeasible. We instead develop appropriate *Conf* and *Sch* with heuristic algorithms to make

the bound *tight* as possible based on SLT NoC. Before introducing the algorithm, we provide the following definitions.

Definition 8 (Remaining Path). *For the task pair τ_i, its remaining path Λ is defined as the task pair sequence from τ_i to $\tau_{|E|+1}$. The schedule length of Λ, including the time of job execution and message transmission, is denoted by $\mathcal{L}(\Lambda)$.*

Definition 9 (Remaining Critical Path). *For a task pair τ_i of the real-time DAG task $\mathcal{G}$, its remaining critical path Λ_c is a path that satisfies the following property:*

$$\forall \Lambda_{path} \in S_{paths}(\tau_i), \mathcal{L}(\Lambda_c) \geq \mathcal{L}(\Lambda_{path})$$

where $S_{paths}(\tau_i)$ is the set of paths from τ_i to $\tau_{|E|+1}$.

By Def. 9, the schedule length of remaining critical path Λ_c for τ_i is $\mathcal{L}(\tau_i) = \mathcal{L}(\Lambda_c)$. To reduce the runtime uncertainties induced by parallel transmission on a single resource pair, we assume all of the task pairs are allocated to the resource pairs that are within the HPC_{max} (i.e., $|\mathcal{R}_{p,\lambda}| \leq HPC_{max}$). For a message $m_{i,j}$, the worst-case latency is $m_{i,j} + t_r$ (e.g., t_r represents the time to set up SLT path [26]). The remaining worst-case computation and communication time can be calculated together since the worst-case latency of a message remains the same under any job-to-core mapping and any dimension-order route (DOR) [13] within HPC_{max}. To illustrate the remaining critical path Λ_c, for v_2 in Fig. 1, assuming $t_r = 1$, $\Lambda_1 = \{\langle m_{1,2}, v_2\rangle, \langle m_{2,4}, v_4\rangle, \langle m_{4,7}, v_7\rangle\}$, $\mathcal{L}(\Lambda_1) = 19$; $\Lambda_2 = \{\langle m_{1,2}, v_2\rangle, \langle m_{2,5}, v_5\rangle, \langle m_{5,7}, v_7\rangle\}$, $\mathcal{L}(\Lambda_2) = 17$. Thus, Λ_c is Λ_1 where $\mathcal{L}(\tau_2) = \mathcal{L}(\Lambda_c) = 19$.

In the following, we propose an algorithm to derive configurations *Conf* and *Sch*, with a goal to shorten the estimated bound R as much as possible. The bound and configurations are derived by "simulating" the runtime execution at design time, assuming the jobs/messages are executed/transmitted for their WCETs, Here, for illustrative simplicity, we combine *Conf* and *Sch* as *CS*. This configuration label for each generated task pair τ_i is defined as a tuple $\Upsilon_i = (\mathcal{M}_{\tau_i \to \mathcal{R}_i}, s_i)$, where $\mathcal{M}_{\tau_i \to \mathcal{R}_i}$ refers to the allocation of task pair τ_i to resource pair $\mathcal{R}_i$; s_i is the effective design-time start time of τ_i. Then, the configuration vector can be derived by $CS = \{\Upsilon_1, \Upsilon_2, ..., \Upsilon_{|E|+1}\}$ that is used at runtime.

In Alg. 1, we define the scheduling point t_{sp} starting from 0, and put the first task pair $\tau_1 = \langle null, v_1\rangle$ into the ready queue $\mathcal{Q}_{ready}$. $\mathcal{Q}_{tp}$ contains the task pairs that have been dispatched, while $\mathcal{Q}_{na}$ contains the task pairs that cannot find available computation and communication resources at the current scheduling point. Also define the task pair τ_i ($\in \mathcal{Q}_{ready}$) with the longest remaining critical path as the *critical task pair* $\tau_c = \langle m_{p,s}, v_s\rangle$. To shorten the estimated upper bound as much as possible, we always firstly schedule τ_c at each t_{sp}, shown in Line 5. When there are multiple available resource pairs (collected in $\mathcal{Q}_{rp}(v_p)$) of v_p, τ_c will be scheduled on the idle resource pair $\mathcal{R}_{p,\lambda} = \langle r_{p\to s}, P_s\rangle$ that has minimum manhattan distance so that it would cause less potential contention to unexecuted task pairs. If a task pair points to a synchronization job v_{syn} and the manhattan distance between its predecessor and v_{syn} is more than

Algorithm 1: Response Time Bound Exploration

Input: $\mathcal{G} = (V, E)$; State Ψ of SLT NoC π;
Output: Upper bound R and configuration *CS*;

```
1  Scheduling point t_sp ← 0; Q_ready ← {τ_1};
2  while (Q_tp.size() ≠ |E| + 1) do
3      Q_na ← ∅; //not available task pairs;
4      while (Q_ready ≠ ∅) do
5          τ_c ← max_{τ_i ∈ Q_ready} L(τ_i); //critical task pair;
6          v_p ← Pred(τ_c); //the predecessor job of τ_c;
7          for (R_{p,λ} ∈ Q_rp(v_p)) do
8              if (isIdle(R_{p,λ}) ∧ isShortest(R_{p,λ})) then
9                  R_{p,min} ← R_{p,λ};
10         if (R_{p,min} ≠ null) then
11             Map(τ_c → R_{p,min});
12             s_c ← t_sp; updateState(Ψ, π);
               Q_tp ← Q_tp ∪ {τ_c};
13         else
14             Q_na ← Q_na ∪ {τ_c};
15         Q_ready ← Q_ready \ τ_c;
16     Derive the next scheduling point t_sp;
17     Q_ready ← Q_na ∪ {new ready task pairs};
18 Return (R, CS);
```

HPC_{max}, an available DOR route [13] can be employed via an/multiple intermediate router(s). v_{syn} is logically attached to the latest incoming message as a task pair. According to the state of the returned $\mathcal{R}_{p,min}$, there are two cases.

(1) $\mathcal{R}_{p,min} \neq null$ in Line 10-12. Each critical task pair $\langle m_{p,s}, v_s\rangle$ in $\mathcal{Q}_{ready}$ will be mapped to the returned resource pair $\langle r_{p\to s}, P_s\rangle$. The running state is "updated". τ_c is inserted into $\mathcal{Q}_{tp}$ and removed from $\mathcal{Q}_{ready}$.
(2) $\mathcal{R}_{p,min} = null$ in Line 13-14. It means none of the resource pairs around v_p are available. τ_c is put into $\mathcal{Q}_{na}$ and will be scheduled at the next scheduling point.

When all the remaining ready task pairs are traversed, the next scheduling point is triggered by the completion of a task pair. Combined with $\mathcal{Q}_{na}$, newly generated ready task pairs that have no unfinished predecessors are created and put into $\mathcal{Q}_{ready}$ in Line 16-17. The last scheduling point t_{sp} triggered by the completion of the *last* task pair is the estimated bound R of the DAG task $\mathcal{G}$. Then, the bound R and the configuration vector $CS = \{\Upsilon_1, \Upsilon_2, ..., \Upsilon_{|E|+1}\}$ are returned in Line 18. Since $\Upsilon_i = (\mathcal{M}_{\tau_i \to \mathcal{R}_i}, s_i)$, $\mathcal{M}_{\tau_i \to \mathcal{R}_i}$ and s_i constitute *Conf* and *Sch*, respectively. The total time complexity of Alg. 1 is $\mathcal{O}(HPC_{max}^2 \times |E|^2)$, where HPC_{max} refers to the maximum number of hops can be reached per cycle. Since HPC_{max} is a constant (e.g., $HPC_{max} = 6$) when SLT NoC is designed, the actual time complexity is $\mathcal{O}(|E|^2)$, which can be solved within polynomial time.

6 IMPLEMENTATION AND OVERHEAD ANALYSIS

The previous sections have derived safe and tight bounds for DAG tasks. To facilitate analysis in Sec. 4, $\langle message, successor \rangle$ and $\langle route, core \rangle$ are logically attached as task and resource pairs, respectively. At runtime, jobs and messages can be scheduled separately since they do not have resource contentions. Therefore, the partial order constraints are only enforced among jobs or messages with a direct and indirect relationship. According to *Sch*, the runtime order constraints could have different possible implementations. By Theorem 1, one possibility is to use the start time points as the priorities, i.e., $s_k \leq s_i$ denotes $\tau_k \rightsquigarrow \tau_i$, meaning that τ_k actually starts execution *not later than* τ_i at runtime.

The proposed scheduling approach is presented based on off-the-shelf SLT NoCs, which are well developed such as SMART [26] / optical [16] NoCs. To support the global decision making of our scheduler, an SLT NoC is split into multiple cluster networks, in which the cluster size is designed according to actual requirements. The global control wires are limited to the cluster level, instead of the whole on-chip network, and thus SLT NoC can be scalable for the large-size network. Due to the need for job-to-core allocation, job and message scheduling, and chip thermal distribution, each cluster in NoCs is necessarily equipped with the resource manager [1, 7] to guarantee timing predictability and chip thermal reliability in the dark silicon era. The proposed scheduling strategy can be embedded in the resource manager.

Before execution, a suitable cluster is selected from SLT NoC until the specified deadline is satisfied. $\mathcal{G}$ is executed within the assigned cluster. Similar to circuit switching, data transmission relies on the established "1-cycle" SLT path. By the resource occupation state, which can be stored in the resource manager, the SLT path is established by sending a "*path-setup*" control packet along the available assigned contention-free path from the source to the destination. Once the resources (e.g., core, links) are released by the finished job/message, the resource occupation state is updated immediately. The overhead of the control layer (e.g., control links) on SLT NoC is a common issue in centralized management (e.g., [22, 34]), in which the existing hardware resources can be reused. Since resource allocation in our paper is conducted in a centralized manner, the router buffer of SLT NoC (e.g., SMART NoC) can be removed to mitigate the overhead for real-time systems. In this paper, we mainly focus on the strategic design of the scheduling approach. Additional researches such as exploring more efficient distributed cluster-based SLT NoC will be studied in future work.

7 EXPERIMENTAL EVALUATION

In this section, we conduct experiments to systematically evaluate the effectiveness (i.e., bound tightness, runtime performance) of the proposed scheduling approach DAG-Order with synthetic and realistic benchmarks.

In state-of-the-art (SOTA) works (e.g., [19, 28]) on real-time NoCs, the task model focuses on the independent packet flows and priority-preemptive scheduling approach. In this work, we consider a different task model (i.e., including computation and communication workloads) and the non preemptive scheduling approach on SLT NoCs. While the classical bound formula [15] (i.e., use DAG volume and critical path length) of DAG tasks on multi-cores is not applicable due to the assumption of no communication cost.

Considering that there are no proper works of literature on the bound derivation of joint consideration of computation and communication of DAG tasks on real-time NoCs, to facilitate quantitative comparison, our proposed bound exploration algorithm (i.e., Alg. 1) is divided into two slightly different schemes, using the performance metric *Normalized Time*. Let R_b be the estimated baseline bound, while R_h be our improved bound. The difference is that when there are multiple ready jobs and messages (i.e., Line 5 in Alg. 1) in the algorithm of bound exploration, R_b randomly chooses a job/message, while R_h selects the job/message that has the longest remaining critical path. Regarding the actual response time, [9] is regarded as the SOTA work with an order-based scheduling approach. Let R'_b and R'_h denote the actual response time with [9] and our proposed order at runtime, respectively. The actual response time is obtained by simulating the runtime execution and varying the execution time within the WCET of each job/message.

7.1 Synthetic Benchmarks

To validate the effectiveness, the synthetic DAG tasks are generated by the widely-used Erdos-Renyi method $G(|V|, p)$ [12]. Specifically, the parameters of the DAG tasks and SLT NoC are generated as follows:

- *Connectivity probability threshold* p_{con}. If a randomly generated probability from (0, 1) is less than the predefined probability threshold p_{con}, a directed edge between two vertices is generated.
- *NoC mesh size* $m \times m$. The synthetic tasks are "executed" on $m \times m$ 2D-mesh SLT NoC with $HPC_{max} = 6$.
- *WCET*. The WCET of each job/message complies with the realistic measurement from OpenMP programs [39].
- *WCET varying probability threshold* p_{vary}. A probability p'_{vary} is randomly generated in (0.1, p_{vary}), and actual execution time is set as $p'_{vary} \times WCET$.
- *Maximum number of jobs* N_{max}. The number of jobs of the DAG task is to choose in (4, N_{max}).

In Fig. 7, all the results are averaged by a large set of experiments, and normalized to the STOA actual response time R'_b. Our derived bound R_h is consistently less than the baseline bound R_b under different parameter settings, since we always select the critical job/message that has the longest remaining critical path, when there are multiple ready jobs and messages at each scheduling point. It means that, by employing our proposed bound R_h, less computation and communication resources are needed under the same deadline, in which the resource assignment is well controlled and

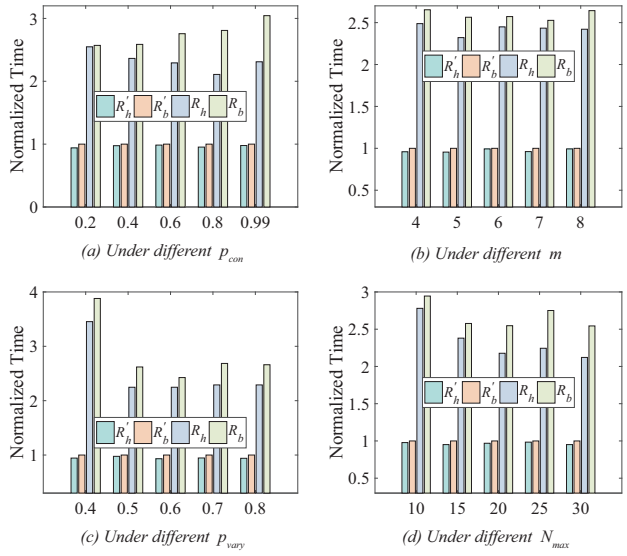


Figure 7: Normalized time for synthetic benchmarks.

the resource over-provisioning is avoided. Note that, the difference (formally $R_h - R_b$) between R_h and R_b is lower for the smaller p_{con} values. This is because, the DAG structure is less complex with the smaller p_{con}, and the baseline bound R_b with random schedule performs closer to R_h. And, there is a small number of cases where R_h is more than R_b since the non-preemptive critical scheduling is not optimal.

Moreover, the runtime performance (i.e., actual response time) of R'_h performs better than R'_b. This is because, R'_h granularly considers the direct and indirect contention, and more idle resources are utilized. It means that the scheduling flexibility of R'_h is better than [9], since our proposed order constraints did not sacrifice too much flexibility when guaranteeing the timing anomaly free execution. Since the analyzed DAG task is finished earlier than that of [9], the assigned resources can also be available and assigned to other tasks earlier. Note that, R'_h is strictly not more than R'_b in all the runtime executions, since R'_b is under a global but partial order for R'_h on the shared resources. Besides, all of runtime executions are free of timing anomaly in the experiments (i.e., $R'_b, R'_h \leq R_b, R_h$), also proved in Theorem 1.

7.2 Realistic Benchmarks

We collect 4 realistic streamit benchmarks [36], including *Autocor* (atr), *Audiobeam* (adm), *FMRadio* (fmo) and *H264* (h264). The employed benchmarks are broadly representative of the realistic benchmarks, as they have different orders of magnitude in workloads, and cover audio processing, scientific processing, etc. Also, the proposed scheme can be extended to other benchmarks modeled as DAG. As shown in Fig. 8, the degree of performance improvement is not very obvious due to bias to the limited range of benchmarks, but the results are consistent with that of synthetic benchmarks. R_b is less than the baseline R_h, and the actual response time R'_h also less than the SOTA actual response time R'_b.

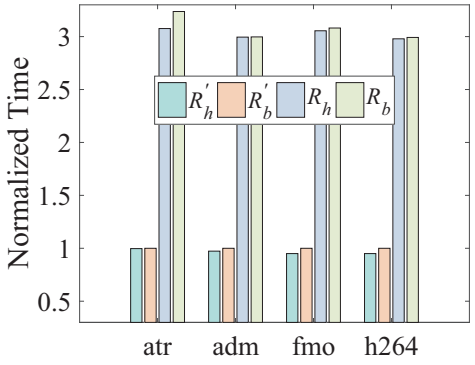


Figure 8: Normalized time for realistic benchmarks.

8 RELATED WORK

The literature on real-time NoCs is broadly extensive. The authors in [31, 32] propose a time-division-multiplexing (TDM)-based time-triggered routing scheme on NoCs, which can isolate conflicting packets and be suitable for time-critical systems. But it is not easy to build the TDM schedule table and construct the global clock synchronization.

More researchers in [11, 19, 28, 35, 40] conduct real-time communication analysis on the distributed priority-preemptive wormhole NoCs, borrowing the uniprocessor scheduling theory. But the communication resources (e.g., router, link) between the source and the destination are not mutually-exclusive (i.e., single resource) and can be shared concurrently by multiple flows. The actual worst-case timing behavior is difficult to identify due to the multi-point progressive blocking (MPB), and thus a series of new analyzes [19, 28, 40] tried to address the safety problem of the worst-case latency bound by considering more runtime cases (i.e., counterexamples) since the first attempts to solve the schedulability analysis in [17]. The state-of-the-art works [19, 28] on real-time wormhole NoCs are proposed by accurately considering the MPB latency [40], but overestimated. Another research employs network calculus (e.g., [33]) or compositional performance analysis (e.g., [37]). The worst-case latency is derived by summarizing the latency of the routers from the source to the destination, which can be pessimistic except the one [14] applying "pay multiplexing only once" principle.

To guarantee the tightness of the worst-case latency, there are two recent works. The one [6] is proposed by Burns et al. with a new distributed flow control mechanism. This communication scheduling approach can improve the latency bound tightness by eliminating the MPB latency [19, 40] (i.e., backpressure), but challenges exist on tile memory management. The other one [38] is proposed by Ueter et al. with a centralized priority-preemptive scheduling approach with *all-or-nothing* property. The essential idea is that a source-destination link, including multiple router-to-router links, is modeled as a *logically* single link, thus also eliminating the unpredictable MPB latency. The logically "single" link is mutually exclusively used by a packet flow without being shared

at a time, at the cost of the link resource waste. Both the distributed (e.g., [19, 40]) and centralized (e.g., [38]) priority-preemptive NoC scheduling methods did not pay attention to the computation tasks and inter-packet dependence.

In this work, we consider a general task model including computation and communication on NoCs, which is missing in the state-of-the-art works (e.g., [18, 19, 28, 40]).

9 CONCLUSION

For computation-intensive cyber-physical systems, NoC-based many-core processors are increasingly adopted, but the challenges also are introduced on timing predictability. In this paper, to guarantee timing predictability, we propose a novel scheduling paradigm on a kind of SLT NoC, called DAG-Order, taking into account task computation and inter-core communication. DAG-Order is based on non-preemptive scheduling enforced with certain order constraints, with which the DAG tasks can be safely and tightly scheduled on NoCs. Especially, this work opens new research directions in future work, e.g., the exploration of more efficient SLT NoC.

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] Mohammad Abdullah Al Faruque and et al. 2008. ADAM: run-time agent-based distributed application mapping for on-chip communication. In *DAC*. IEEE, 760–765.
[2] Hamid Arabnejad and Jorge G Barbosa. 2013. List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table. *TPDS* 25, 3 (2013), 682–694.
[3] Yashar Asgarieh and et al. 2019. Smart-Hop Arbitration Request Propagation: Avoiding Quadratic Arbitration Complexity and False Negatives in SMART NoCs. *TODAES* 24, 6 (2019), 1–25.
[4] Sanjoy Baruah. 2015. The Federated Scheduling of Systems of Conditional Sporadic DAG Tasks. In *EMSOFT*. IEEE, 1–10.
[5] Sanjoy Baruah and et al. 2012. A Generalized Parallel Task Model for Recurrent Real-time Processes. In *RTSS*. IEEE, 63–72.
[6] Alan Burns and et al. 2020. A Novel Flow Control Mechanism to Avoid Multi-Point Progressive Blocking in Hard Real-Time Priority-Preemptive NoCs. In *RTAS*. IEEE.
[7] Guilherme Castilhos and et al. 2013. Distributed resource management in NoC-based MPSoCs with dynamic cluster sizes. In *ISVLSI*. IEEE, 153–158.
[8] Hui Chen and et al. 2020. ArSMART: An Improved SMART NoC Design Supporting Arbitrary-Turn Transmission. *arXiv preprint arXiv:2011.09261* (2020).
[9] Peng Chen and et al. 2019. Timing-Anomaly Free Dynamic Scheduling of Conditional DAG Tasks on Multi-Core Systems. *TECS* 18, 5s (2019), 1–19.
[10] Peng Chen and et al. 2020. Contention Minimized Bypassing in SMART NoC. In *ASP-DAC*. IEEE, 205–210.
[11] Peng Chen and et al. 2020. Reduced Worst-Case Communication Latency Using Single-Cycle Multi-Hop Traversal Network-on-Chip. *TCAD* (2020).
[12] Daniel Cordeiro and et al. 2010. Random Graph Generation for Scheduling Simulations. In *ICST*.
[13] William James Dally and Brian Patrick Towles. 2004. *Principles and Practices of Interconnection Networks*. Elsevier.
[14] Frédéric Giroudot and et al. 2018. Buffer-Aware Worst-Case Timing Analysis of Wormhole NoCs using Network Calculus. In *RTAS*. IEEE, 37–48.
[15] Ronald L. Graham. 1969. Bounds on Multiprocessing Timing Anomalies. *SIAM J. Appl. Math.* (1969).
[16] Huaxi Gu and et al. 2008. A Novel Optical Mesh Network-on-Chip for Gigascale Systems-on-Chip. In *APCCAS*. IEEE, 1728–1731.
[17] SL Hary and F Özgüner. 1997. Feasibility Test for Real-time Communication Using Wormhole Routing. *IEE Proceedings-Computers and Digital Techniques* 144, 5 (1997), 273–278.
[18] Qingqiang He and et al. 2019. Intra-Task Priority Assignment in Real-Time Scheduling of DAG Tasks on Multi-cores. *IEEE Transactions on Parallel and Distributed Systems* (2019).
[19] Leandro Soares Indrusiak and et al. 2018. Buffer-aware Bounds to Multi-point Progressive Blocking in Priority-preemptive NoCs. In *DATE*. IEEE, 219–224.
[20] Chris Jackson and Simon J Hollis. 2010. Skip-links: A Dynamically Reconfiguring Topology for Energy-efficient NoCs. In *2010 International Symposium on System on Chip*. IEEE, 49–54.
[21] Biresh Kumar Joardar and et al. 2018. High Performance Collective Communication-Aware 3D Network-on-Chip Architectures. In *DATE*. IEEE, 1351–1356.
[22] Thawra Kadeed and et al. 2019. Integrated Energy Control for Hard Real-Time Networks-on-Chip. In *RTSS*. IEEE, 4–16.
[23] Muhammad Kafil and Ishfaq Ahmad. 1998. Optimal Task Assignment in Heterogeneous Distributed Computing Systems. *IEEE concurrency* (1998).
[24] Manupa Karunaratne and et al. 2017. Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect. In *DAC*. 1–6.
[25] Nikolay Krasimirov Kavaldjiev and et al. 2003. *A Survey of Efficient On-Chip Communications for SoC*. Centre for Telematics and Information Technology, University of Twente.
[26] Tushar Krishna and et al. 2013. Breaking the On-chip Latency Barrier using SMART. In *HPCA*. IEEE, 378–389.
[27] Weichen Liu and et al. 2017. Work-in-Progress: Fixed Priority Scheduling of Real-time Flows with Arbitrary Deadlines on SMART NoCs. In *EMSOFT*. IEEE, 1–2.
[28] Borislav Nikolić and et al. 2019. Real-time Analysis of Priority-preemptive NoCs with Arbitrary Buffer Sizes and Router Delays. *Real-Time Systems* (2019).
[29] Umit Y Ogras and Radu Marculescu. 2006. It's a Small World After All: NoC Performance Optimization Via Long-Range Link Insertion. *TVLSI* 14, 7 (2006), 693–706.
[30] Iván Pérez and et al. 2019. SMART++: Reducing Cost and Improving Efficiency of Multi-hop Bypass in NoC Routers. In *NOCS*. 1–8.
[31] Toms Picornell Picornell and et al. 2020. Enforcing Predictability of Many-cores with DCFNoC. *IEEE Trans. Comput.* (2020).
[32] Anastasios Psarras and et al. 2015. PhaseNoC: TDM scheduling at the virtual-channel level for efficient network traffic isolation. In *DATE*. IEEE, 1090–1095.
[33] Yue Qian and et al. 2009. Analysis of Worst-Case Delay Bounds for Best-Effort Communication in Wormhole Networks on Chip. In *NOCS*. IEEE, 44–53.
[34] Marcelo Ruaro and et al. 2019. Distributed SDN Architecture for NoC-Based Many-core SoCs. In *NOCS*. 1–8.
[35] Zheng Shi and Alan Burns. 2008. Real-time Communication Analysis for On-chip Networks with Wormhole Switching. In *NOCS*. IEEE, 161–170.
[36] William Thies and Saman Amarasinghe. 2010. An empirical characterization of stream programs and its implications for language and compiler design. In *PACT*. IEEE, 365–376.
[37] Sebastian Tobuschat and Rolf Ernst. 2017. Real-Time Communication Analysis for Networks-on-Chip with Backpressure. In *DATE*. IEEE, 590–595.
[38] Niklas Ueter and et al. 2019. Simultaneous Progressing Switching Protocols for Timing Predictable Real-Time Network-on-Chips. *arXiv preprint arXiv:1909.09457* (2019).
[39] Yang Wang and et al. 2017. Benchmarking OpenMP Programs for Real-time Scheduling. In *RTCSA*. IEEE, 1–10.
[40] Qin Xiong and et al. 2017. Extending Real-time Analysis for Wormhole NoCs. *IEEE Trans. Comput.* (2017).
[41] Lei Yang and et al. 2017. Task Mapping on SMART NoC: Contention Matters, Not the Distance. In *DAC*. 1–6.