PREPRINT: "H. Kashif and H. Patel, "Buffer Space Allocation for Real-Time Priority-Aware Networks," in proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016, pp. 1-12."

Buffer Space Allocation for Real-Time Priority-Aware Networks

Hany Kashif
Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada.
Email: hkashif@uwaterloo.ca

Hiren Patel
Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada.
Email: hiren.patel@uwaterloo.ca

***Abstract*—In this work, we address the challenge of incorporating buffer space constraints in worst-case latency analysis for priority-aware networks. A priority-aware network is a wormhole-switched network-on-chip with distinct virtual channels per priority. Prior worst-case latency analyses assume that the routers have infinite buffer space allocated to the virtual channels. This assumption renders these analyses impractical when considering actual deployments. This is because an implementation of the priority-aware network imposes buffer constraints on the application. These constraints can result in back pressure on the communication, which the analyses must incorporate. Consequently, we extend a worst-case latency analysis for priority-aware networks to include buffer space constraints. We provide the theory for these extensions and prove their correctness. We experiment on a large set of synthetic benchmarks, and show that we can deploy applications on priority-aware networks with virtual channels of sizes as small as two flits. In addition, we propose a polynomial time buffer space allocation algorithm. This algorithm minimizes the buffer space required at the virtual channels while scheduling the application sets on the target priority-aware network. Our empirical evaluation shows that the proposed algorithm reduces buffer space requirements in the virtual channels by approximately 85% on average.**

I. Introduction

Chip-multiprocessors (CMPs) provide a solution to the increasing computational requirements of software applications. Network-on-chips (NoCs) provide an efficient and scalable interconnect for the components of the CMP [1]. CMPs are generally designed to optimize the average case performance of applications. Real-time applications require additional guarantees to meet their timing constraints. Various NoC implementation schemes have been proposed to accommodate real-time applications [2], [3]. Recent research also focuses on worst-case latency (WCL) analysis techniques that provide real-time guarantees in NoCs [3], [4], [5].

An example of an implementation that provides timing guarantees for real-time applications, is run-time arbitration. This scheme allows contention for the network resources unlike other resource reservation schemes such as time-division multiplexing (TDM). Router arbiters deterministically resolve the contention at run-time. WCL analysis techniques consider contention when computing worst-case bounds for the network communication. Priority-aware networks are an example of run-time arbitration schemes [3], [6], [7].

Recent WCL analysis techniques, including flow-level analysis (FLA) [3] and stage-level analysis (SLA) [4] have been developed for priority-aware networks with flit-level preemption. Priority-aware networks employ wormhole switching [8] and virtual channel (VC) resource allocation [9]. These techniques reduce the required buffer space by handling packets at the flit level, and allowing multiple flit buffers (virtual channels) to access the same physical channel. However, priority-aware networks are susceptible to chain-blocking (blocked flits spanning multiple routers). Chain-blocking creates back-pressure in the priority-aware network, which eventually leads to blocking of the computation and communication tasks. Although we know that the blocking of communication tasks due to back-pressure affects its WCL, recent analyses do not account for blocking due to back-pressure. This is because they assume infinite buffer space in the VCs. This assumption makes it difficult to deploy applications on realized implementations. Although a recent work [10] introduces buffer space analysis it computes the necessary buffer space requirements to not suffer chain-blocking. In practice, however, the platform dictates the buffer spaces available per VC, and deploying an application on such a platform may result in chain-blocking. This chain-blocking is not accounted for in prior analyses.

Consequently, in this work, we address the problem of constructing a WCL analysis that incorporates buffer space restrictions. This analysis incorporates the chain-blocking effect in the analysis. We extend the most recent WCL analysis, SLA [4], with buffer space restrictions, which we call SLA+. We experiment with a large set of synthetic benchmarks with 400000 different configurations. This allows us to stress-test the proposed analysis. We show that SLA+ can schedule task sets on priority-aware networks with buffer sizes as small as two flits, while still improving schedulability over prior analyses [3]. It is important to limit design costs through buffer space reduction [11], [12]. Therefore, given a buffer space constraint for each router in the NoC, we propose a polynomial time algorithm for allocating the buffer space between the router's VCs. The algorithm reduces buffer space requirements, on average, by 85% and 89% compared to prior buffer analyses on priority-aware networks.

The rest of the paper is organized as follows. Section II presents the related work, and Section III presents the necessary background to appreciate the proposed work. In Section IV, we present SLA with buffer constraints followed by a buffer space allocation algorithm in Section V. We present experimental evaluation in Section VI, and make concluding statements in Section VII.

II. Related Work

Recent research work focuses on buffer management issues and on optimizing buffer space usage. Some work also devises buffer management techniques to improve system performance. We provide an overview of these works.

Multiple optimization techniques have been proposed to reduce buffer space utilization. Lan et al. [13] replace unidirectional channels in the NoC by bidirectional channels to increase bandwidth utilization. This enables the transmission of flits in both directions to efficiently utilize the hardware resources. Hu and Marculescu [14] propose a buffer space allocation algorithm. The main objective of the algorithm is minimizing the end-to-end packet latencies by efficiently distributing the router's buffer space between channel buffers. Kumar et al. [15] propose a two-phase buffer-sizing algorithm that addresses the non-uniform network utilization. The second phase of the algorithm simulates the NoC and injects data that models the application's communication requirements. Al Faruque and Henkel [16] present a two-step approach for minimizing virtual channel buffers in NoCs. This approach first uses a multi-objective mapping algorithm, followed by further optimizations for buffer space reduction. All these approaches, however, do not provide real-time guarantees for applications.

Some approaches attempt to minimize buffer sizes while meeting performance constraints. Coenen et al. [17] present a buffer sizing algorithm using TDM and credit-based flow control. They attempt to minimize buffer sizes while meeting performance constraints of applications. However, they do not present an approach for computing latencies in the presence of fixed buffer sizes. In this work, we first extend the WCL analysis to enable taking buffer space into account. Then, we attempt to minimize buffer space while meeting latency constraints for priority-aware pipelined communication resource models. Manolache et al. [18] present an approach for buffer space optimization while meeting timing constraints of the applications. The proposed approach uses offline packet routing and communication traffic shaping to reduce contention in the network. The core of their buffer demand computation uses the WCL analysis by Palencia and Gonzalez [19]. This results in pessimistic buffer space bounds [10]. On the other hand, the techniques used to reduce contention between tasks are orthogonal to this work.

Dataflow models can also be used to compute end-to-end timing behavior of jobs. Hansson et al. [20], [21] explain how to conservatively model NoC connections using dataflow models. They then use these models to compute buffer sizes at network interfaces to guarantee system performance. The proposed techniques, however, are limited to applications that can be modeled using dataflow models, and dataflow models need to be constructed for each application. In this work, we directly incorporate buffer sizes in the WCL analysis of priority-aware networks.

Kashif and Patel [10] introduce techniques for computing the buffer space requirements for FLA [3] and SLA [4]. Shi and Burns [3] introduce FLA to compute WCL bounds using response-time analysis techniques. SLA borrows the same priority-aware router and preemption model introduced by the authors by Shi and Burns [3]. FLA views a communication task that spans multiple network nodes as an indivisible entity. Hence, FLA does not incorporate pipelined and parallel transmission of data in the NoC. This results in pessimistic WCL bounds of the communication tasks. On the other hand, SLA [4] performs the WCL analysis on the stage level of the network, which enables SLA to consider the pipelined and parallel transmission of data in the NoC. This results in tighter WCL bounds compared to FLA. In particular, in [10], Kashif and Patel present FLBA (flow-level buffer analysis) and SLBA (stage-level buffer analysis) to compute the necessary buffer space bounds that prevent network backpressure using FLA and SLA, respectively. Both SLA and FLA do not take buffer space limits into account when computing WCL bounds; hence, assume that chain-blocking never occurs. The authors in [10] attempt to address this by finding the minimum buffer space required at the routers to ensure that chain-blocking never happens. In this work, we include the effect of buffer space constraints in the VCs on the WCL computation using SLA, i.e., we consider chain-blocking in the WCL analysis itself. We compare the assigned buffer space using our allocation algorithm against both FLBA and SLBA.

Other analysis techniques such as network calculus and real-time calculus can also be used to perform worst-case latency analysis as well as discover buffer requirements. For example, Bakhouya et al. [22] present an approach to compute delay and buffer bounds on NoCs using network calculus. Note that network calculus relies on convolution of arrival curves with the service curve at each component; hence, not incorporating the pipelined transmission of data across the network. The advantage of network calculus is that it is general, but it often renders pessimistic bounds. SLA [4], on the other hand, uses response-time analysis techniques and is specific to the priority-aware network, which can provide tighter bounds.

III. Background

In this section, we overview the resource and task models used in this work. We also discuss priority-aware routers, stage-level analysis, and stage-level buffer space analysis.

A. System Model

We use the resource model proposed in [4], [5], and used in [10], [23]. The resource model is a pipelined communication resource model consisting of processing and communication resources. Computation tasks execute on the processing resources, and communicate using communication tasks which execute on the communication resources. The communication resources between any two processing resources are a number of pipelined stages. The communication data can execute simultaneously on the pipelined stages. One communication datum transmits in one time slot. A datum executing on one stage at time t, will be ready to transmit on the next stage at time $t + 1$. Another datum can execute on the former stage

at time $t+1$. Communication tasks support fixed priority preemption at the datum level. Hence, a datum, that executed on one stage at time t, will only execute on the next stage at time $t+1$, if no higher priority data is waiting to access the latter stage at $t+1$.

Priority-aware networks with wormhole switching and flit-level preemption is an example of this resource model. The network's processing elements and routers map to the model's processing resources. The network's links between the processing elements map to the model's communication resources. These links support wormhole switching which enables the pipelined transmission of data. Data packets in the network are divided into smaller units called flits. In what follows, we will use the term *flits* to refer to the smaller data units of the resource model. Fixed priority preemption at the flit level is achieved by priority-aware routers which we discuss in more detail in Section III-B.

We use the same task model as in [4], [10], [23]. A set of n communication tasks, $\Gamma := \tau_i : \forall\ i \in [1, n]$, is deployed on the resource model. A communication task, τ_i, is a 5-tuple $\langle P_i, T_i, D_i, J_i^R, L_i \rangle$. The task τ_i has a fixed priority P_i, a period (periodic task) or a minimum interarrival time (sporadic task) T_i, a deadline D_i, and a release jitter J_i^R. A task's basic stage latency L_i is the WCL of the task, without any interference, on one communication stage. A task is schedulable if its WCL R_i (including interferences) is less than or equal to its deadline D_i. The route δ_i taken by the communication task τ_i is a sequence of communication stages $(s_1, \ldots, s_{|\delta_i|})$ where $|\delta_i|$ is length of the route δ_i. To simplify the mathematical formulation, we denote a stage that precedes a stage s on a task's path using s^-. We also denote a stage that directly follows a stage s on a task's path using s^+. Figure 1 shows a simple network with three communication tasks: τ_0, τ_1, and τ_2. Consider stage $s = (v_2, v_3)$ on the path δ_1 of task τ_1. Then, stages s^- and s^+ becomes (v_1, v_2) and (v_3, v_4), respectively. Please note that we provide a symbol table in Table I in the Appendix as a source for reference for all the symbols and their semantics.

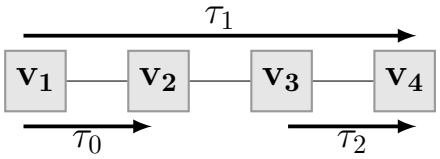


Fig. 1: A simple illustrative example

We have a few assumptions similar to [4], [5], [10]. We assume a given mapping of computation tasks to processing resources. We also assume given paths for the set of communication tasks Γ based on a deterministic offline routing algorithm. Communication tasks have distinct priorities. While priority sharing can be supported, it is not part of this work. We assume as well that the buffer sizes at the network interfaces are large enough such that they do not cause blocking. This simplifies the analysis to focus on the buffer sizes of the routers. However, we believe the analysis can be further extended to support these network interface buffers and their effect on blocking as well. For clarity, we assume that a flit executes on a stage in one time unit. This simplifies the mapping between buffer space and latency computations. As mentioned earlier, a flit incurs a delay of one time unit as it moves from one stage to the other.

B. Priority-Aware Router

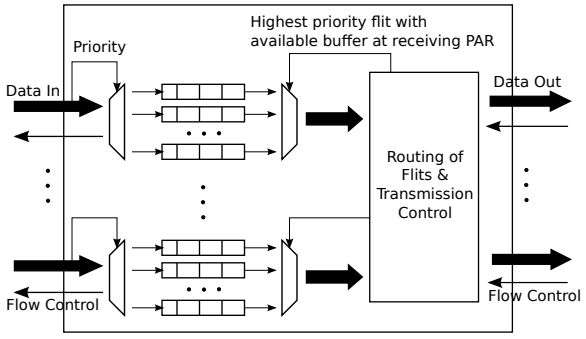


Fig. 2: Priority-aware router

We use the same priority-aware router (Figure 2) proposed in [3], and used in [23], [10]. Each input port consists of multiple VCs; one for each distinct priority level. Each VC buffers flits of the corresponding priority level in a FIFO order. Flits are forwarded to output ports in the same order in which they were buffered. The priority-aware router selects the output port of a flit based on the path of the flit's task. If multiple flits require access to the same output port, based on our resource model, the router will forwards the flit with the highest priority.

The priority-aware router implements a flow-control mechanism. Flow-control prevents overflowing of downstream buffers. The router will allow a lower priority flit to access an output port, if a buffered higher priority flit, that is waiting for the same port, is blocked by flow-control. In particular, we assume a credit flow-control mechanism such as in [2]. Each VC buffer, in a router, has a corresponding credit counter in the upstream router. The number of credits corresponds to the number of available buffer slots in the downstream VC. The upstream router decrements the credit counter every time it sends a flit downstream, and cannot send flits when the counter reaches zero. Whenever the downstream VC sends a flit out, a credit is sent simultaneously upstream to increment the upstream credit counter. The credit feedback delay, *CF*, is the time that one credit takes to reach the upstream router including any processing overhead.

For a stage s, the upstream router is symbolized by $\underset{s}{UR}$, and the buffer space available for task τ_i in $\underset{s}{UR}$ is $\underset{s}{VC}_i$. The downstream router is denoted by $\underset{s}{DR}$ (which is also the upstream router of stage s^+, $\underset{s^+}{UR}$), and the buffer space available at $\underset{s}{DR}$ for task τ_i is $\underset{s^+}{VC}_i$. In Figure 1, consider stage $s = (v_2, v_3)$ on path δ_1 of task τ_1. The upstream router $\underset{s}{UR}$

of stage s is at node v_2, and the downstream router $\underset{s}{DR}$ is at node v_3.

C. Direct and Indirect Interference

Direct interference occurs when a higher priority task τ_j preempts a lower priority task τ_i on a shared stage. The set of tasks directly interfering with task τ_i on a stage s is represented by $\mathbb{S}_{si}^{D}$. In Figure 1, assume that the priorities of tasks are such that task τ_1 has higher priority than τ_2 and less priority than τ_0. In that case, task τ_0 directly interferes with τ_1 on its first stage such that $\mathbb{S}_{s_1}^{D} = \{\tau_0\}$. Also, task τ_1 directly interferes with τ_2 on its only stage such that $\mathbb{S}_{s_1}^{D} = \{\tau_1\}$.

A task τ_k indirectly interferes with a lower priority task τ_i when task τ_i has direct interference with an intermediate task τ_j, and τ_j has direct interference with task τ_k. Tasks τ_i and τ_k do not directly interfere, however. Also, the interference between tasks τ_j and τ_k occurs before the interference between tasks τ_j and τ_i.

Task τ_k can delay the release of task τ_j. Hence, when computing the interference suffered by task τ_i, in addition to the release jitter of task τ_j, the delay caused by τ_k has to also be considered. This extra delay suffered by τ_j is called indirect interference jitter, and is represented by the term $\underset{s}{J}_{j}^{I}$. Indirect interference jitter adds to the jitter of directly interfering tasks. The set of tasks indirectly interfering with task τ_i on stage s is denoted by $\mathbb{S}_{si}^{I}$. In Figure 1, and still assuming the priorities: $P_0 > P_1 > P_2$. Task τ_2 suffers indirect interference from task τ_0 through the intermediate task τ_1 such that $\mathbb{S}_{s_1}^{I} = \{\tau_0\}$. For more details and formal definitions, we direct the reader to [4].

D. Stage-Level Analysis

SLA performs the WCL analysis on the stage-level of the pipelined resource model [4]. It accounts for both direct and indirect interferences. Starting with the first stage on a task's path, SLA adds new interferences that the task under analysis experiences on each stage. A higher priority task, interfering with the task under analysis on a contiguous set of stages, cannot cause more interference on latter stages than what it caused on the first stage of interference. The WCL of the task under analysis is monotonically non-decreasing with respect to the sequence of stages forming the task's path.

A level-i busy interval, $\underset{s}{B}_i$, is the time interval during which flits, with priority higher than or equal to P_i, continuously execute on stage s before the stage is idle. The level-i busy interval of task τ_i on stage $s \in \delta_i$ is given by:

$$\underset{s}{B}_i = \underset{s^-}{B}_i + \sum_{\forall \tau_j \in \mathbb{S}_{si}^{D}} \left\lceil \frac{\underset{s}{B}_i + J_j^R + \underset{s}{J}_j^I}{T_j} \right\rceil * L_j + \left\lceil \frac{\underset{s}{B}_i + J_i^R}{T_i} \right\rceil * L_i - \sum_{\forall \tau_j \in \mathbb{S}_{si}^{D} \cap \mathbb{S}_{s^-i}^{D}} \left\lceil \frac{\underset{s^-}{B}_i + J_j^R + \underset{s}{J}_j^I}{T_j} \right\rceil * L_j - \left\lceil \frac{\underset{s^-}{B}_i + J_i^R}{T_i} \right\rceil * L_i \quad (1)$$

Equation 1 shows how SLA computes the busy interval of task τ_i on stage s, by adding new interferences to the busy interval $\underset{s^-}{B}_i$ of the preceding stage s^-. The second term accounts for interference by higher priority tasks on stage s in the set $\mathbb{S}_{si}^{D}$, and the third term accounts for instances (jobs) of the task under analysis executing on stage s. The fourth and fifth terms subtract interference and latency of jobs of τ_i that are common on stages s and s^- and, hence, have already been accounted for in the busy interval $\underset{s^-}{B}_i$. Note that for the first stage of δ_i, the first, fourth, and fifth terms are all equal to zero.

Using SLA, the WCL R_i of a task τ_i on its path δ_i is given by:

$$R_i = \max_{p=1 \ldots \underset{s}{p}_{B,i}} (\underset{s}{w}_i(p) - (p-1) * T_i + J_i^R) + |\delta_i| - 1 \quad (2)$$

where

$$\underset{s}{w}_i(p) = \underset{s}{I}_i(p) + p * L_i$$

$$\underset{s}{I}_i(p) = \underset{s^-}{I}_i(p^-) + \sum_{\forall \tau_j \in \mathbb{S}_{si}^{D}} \left\lceil \frac{\underset{s}{w}_i(p) + J_j^R + \underset{s}{J}_j^I}{T_j} \right\rceil * L_j - \sum_{\forall \tau_j \in \mathbb{S}_{si}^{D} \cap \mathbb{S}_{s^-i}^{D}} \left\lceil \frac{\underset{s^-}{w}_i(p^-) + J_j^R + \underset{s}{J}_j^I}{T_j} \right\rceil * L_j$$

$$p^- = \begin{cases} p & \text{if } p \leq \underset{s^-}{p}_{B,i} \\ \underset{s^-}{p}_{B,i} & \text{otherwise} \end{cases}$$

Equation 2 computes the WCL of τ_i as the maximum WCL across all jobs of τ_i that execute in τ_i's busy interval on the last stage $s_{|\delta_i|}$. The largest-numbered job of τ_i in the busy interval $\underset{s}{B}_i$ is $\underset{s}{p}_{B,i}$. The term $\underset{s}{w}_i(p)$ represents the worst-case completion time of job p on stage s measured from the start of the busy period $\underset{s}{B}_i$. The term $\underset{s}{I}_i(p)$ represents the interference suffered by job p on stage s from higher priority tasks. To find the worst-case completion time of job p, similar to Equation 1, Equation 2 progressively adds new interferences on each stage of δ_i. For more details and proofs of the WCL analysis, we direct the reader to [4].

E. Stage-Level Buffer Space Analysis

SLBA computes the buffer space at each VC of each router on the path of the task under analysis such that there is no back-pressure in the NoC [10]. SLBA utilizes the interference computations from SLA to bound the buffer space required at the VCs. The buffer space required at the virtual channel $\underset{s}{VC}_i$ in the upstream router $\underset{s}{UR}$ of stage s, is bounded by $\underset{s}{I}_i + 1$. Hence, the worst-case interference $\underset{s}{I}_i$ suffered on stage s is used to compute an upper bound on the buffer space needed at $\underset{s}{VC}_i$. Note that this buffer space is also bounded by the maximum number of flits of τ_i that can exist in the busy interval $\underset{s}{B}_i$. Therefore, the buffer space $\underset{s}{VC}_i$ can be given by:

$$\underset{s}{VC}_i = \min(\underset{s}{p}_{B,i} * L_i, \sum_{\forall \tau_j \in \mathbb{S}_{si}^{D}} \left\lceil \frac{\underset{s}{B}_i + J_j^R + J_j^I}{T_j} \right\rceil * L_j + 1)$$

where the first term accounts for the maximum number of flits of τ_i in $\underset{s}{B_i}$, and the second term accounts for $\underset{s}{I_i} + 1$. For more details and proofs, we direct the reader to [10].

IV. SLA with Buffer Space Constraints

First, we need to establish a separation between two types of interferences: (1) the interference that a task τ_i suffers from higher priority tasks on a stage s, and (2) the interference/blockage that τ_i suffers on stage s due to limited buffer space at the downstream router of the same stage. The first type of interference, $\underset{s}{I_i}$, results from the contention on stage s from higher priority tasks. The second type of interference results from back-pressure. It is the amount of time for which flits of τ_i are blocked from accessing stage s because the corresponding buffer in the downstream router is full. For clarity, we refer to the second type of interference as *blockage*.

We will first compute the worst-case blockage suffered by task τ_i while ignoring credit feedback delay. Then, we extend it to include credit feedback delay. Finally, we present theorems to incorporate the buffer space restrictions into SLA.

A. Worst-Case Blockage without Credit Feedback Delay

A task under analysis τ_i will suffer a worst-case blockage $\underset{s}{IB_i}$ on stage s of its route due to limited buffer space at the downstream router of stage s. The amount of blockage on stage s due to back-pressure depends on the interference suffered by τ_i from higher priority tasks on the next stage s^+.

Lemma 1. *Given a task under analysis τ_i, the worst-case blockage that τ_i suffers due to back-pressure on the last stage $s_{|\delta_i|}$ of its route δ_i is $\underset{s}{\mathrm{IB}_i} = 0$.*

Proof: The buffer space $\underset{s}{VC_i}$ required by task τ_i at $\underset{s}{UR}$ depends on the interference suffered by τ_i on stage s [10]. The buffer space required when there is no interference from higher priority tasks is only one flit, i.e., $\underset{s}{VC_i} = 1$ [10]. Since no stage follows the last stage $s_{|\delta_i|}$, task τ_i does not suffer interference on any stages following $s_{|\delta_i|}$. Hence, the buffer space needed at the destination router of task τ_i is only one flit. A flit of task τ_i, at the destination router, will not be blocked because it will not traverse a stage on which it can suffer interference (we consider only interference on the network's stages). Since there is no blockage at the downstream router of stage $s_{|\delta_i|}$, then no blockage can occur on that stage due to back-pressure. Therefore, the worst-case blockage that τ_i suffers due to back-pressure on the last stage $s_{|\delta_i|}$ is $\underset{s}{IB_i} = 0$. ∎

Next, we compute $\underset{s}{IB_i}$ for any stage s on the route of task τ_i. To compute $\underset{s}{IB_i}$, we consider the interference that happens on stages following stage s on the route δ_i. Using this interference, we can find the amount of blockage that τ_i suffers on stage s because of the limited buffer space at $\underset{s}{DR}$.

Lemma 2. *Given a task under analysis τ_i with a busy interval $\underset{s}{B_i}$ on stage s of its route δ_i, and given that $\underset{s^+}{\mathrm{VC}_i} \geq \underset{s}{p_{B,i}} * L_i$, the worst-case blockage that τ_i suffers due to back-pressure on stage s is $\underset{s}{\mathrm{IB}_i} = 0$.*

Proof: We mentioned in Section III-D that $\underset{s}{p_{B,i}}$ is the largest-numbered job (starting from $p = 1$) of τ_i that executes in the busy interval $\underset{s}{B_i}$. Hence, the maximum latency of flits of task τ_i that can execute in $\underset{s}{B_i}$ is $\underset{s}{p_{B,i}} * L_i$, which is also the maximum number of flits of τ_i in $\underset{s}{B_i}$. Consider, the buffer space $\underset{s^+}{VC_i}$ available for task τ_i at $\underset{s}{DR}$. If $\underset{s^+}{VC_i} \geq \underset{s}{p_{B,i}} * L_i$, this means that the buffer space will be enough to hold all flits of τ_i that can execute in $\underset{s}{B_i}$. Hence, in that case, flits of τ_i cannot be blocked on stage s, and $\underset{s}{IB_i} = 0$. ∎

Lemma 3. *Given a task under analysis τ_i with a busy interval $\underset{s}{B_i}$ on stage s of its route δ_i, and given that $\underset{s^+}{\mathrm{VC}_i} < \underset{s}{p_{B,i}} * L_i$, the worst-case blockage, $\underset{s}{\mathrm{IB}_i}(\underset{s}{B_i})$, that τ_i suffers due to back-pressure on stage s is given by:*

$$\max\left(0, \underset{s^+}{\mathrm{IB}_i}(\underset{s}{B_i}) + \sum_{\forall \tau_j \in \underset{s^+}{\mathbb{S}_i^D} \setminus \underset{s}{\mathbb{S}_i^D}} \left\lceil \frac{\underset{s}{B_i} + \underset{s}{J_j^R} + \underset{s}{J_j^I}}{T_j} \right\rceil * L_j - \underset{s^+}{\mathrm{VC}_i}\right)$$

Proof: For the last stage $s_{|\delta_i|}$, the succeeding stage s^+ does not exist. Hence, the term $\underset{s^+}{IB_i}(\underset{s}{B_i}) = 0$. Also, the set $\underset{s^+}{\mathbb{S}_i^D} \setminus \underset{s}{\mathbb{S}_i^D}$ does not contain any tasks. Hence, for the last stage $s_{|\delta_i|}$, $\underset{s}{IB_i}(\underset{s}{B_i}) = \max(0, -\underset{s^+}{VC_i}) = 0$. This is equivalent to what we derived in Lemma 1, and forms the base case of our proof. Our proof proceeds by induction. Assume that the lemma holds for stage s^+, we prove the lemma for stage s.

Consider the interference that τ_i suffers from higher priority tasks on stage s^+. These higher priority tasks can be divided into two groups: (1) tasks that also interfere with τ_i on stage s, and (2) tasks that only interfere with τ_i on stage s^+. Consider the first group of tasks. From [4], we know that tasks of this group cannot cause more interference on stage s^+ than the interference they caused on stage s. We also established that, given interference only from this group of tasks, if a flit of τ_i transmits at time t on stage s, then it will transmit at time $t + 1$ on stage s^+. Since, in that case, flits of τ_i cannot suffer more interference from this group of tasks on s^+, this means that no blockage can occur from the first group of tasks on s^+. Now, consider the second group of tasks. These tasks cause *new* interferences on stage s^+ that did not exist on stage s. These new interferences will block the flits of τ_i. The number of blocked flits corresponds to the amount of new interference suffered on stage s^+. The set $\underset{s^+}{\mathbb{S}_i^D} \setminus \underset{s}{\mathbb{S}_i^D}$ contains the tasks causing new interferences. Hence, the term $\sum_{\forall \tau_j \in \underset{s^+}{\mathbb{S}_i^D} \setminus \underset{s}{\mathbb{S}_i^D}} \lceil \frac{\underset{s}{R_i} + \underset{s}{J_j^R} + \underset{s}{J_j^I}}{T_j} \rceil * L_j$ computes the new interference suffered on stage s^+.

Now, consider the blockage $\underset{s^+}{IB_i}(\underset{s}{B_i})$ that τ_i suffers on stage s^+. This means that $\underset{s^+}{IB_i}(\underset{s}{B_i})$ flits of τ_i will be blocked from accessing stage s^+. Due to back-pressure, this blockage will

be propagated to stage s.

The total blockage that task τ_i can suffer on stage s, therefore, equals the blockage suffered on stage s^+ in addition to the new interference caused on stage s^+. The buffer at the downstream router of stage s will hold $\underset{s^+}{VC_i}$ flits of these blocked flits. Hence, the number of blocked flits of τ_i on stage s are only the ones that cannot be buffered at $\underset{s}{DR}$. Therefore, the blockage suffered on stage s now becomes:

$$\underset{s^+}{IB_i}(\underset{s}{R_i}) + \sum_{\forall \tau_j \in \underset{s^+}{\mathbb{S}}_i^D \setminus \mathbb{S}_{si}^D} \left\lceil \frac{\underset{s}{R_i} + J_j^R + \underset{s^+}{J_j^I}}{T_j} \right\rceil * L_j - \underset{s^+}{VC_i}$$

Clearly, this blockage will only occur if the buffer at the downstream router of stage s cannot hold all the blocked flits. This means that if the blockage on stage s is less than or equal to the number of flits $\underset{s^+}{VC_i}$, no blockage will occur. Hence, the $\max$ operator is used to set the blockage to zero in that case. ■

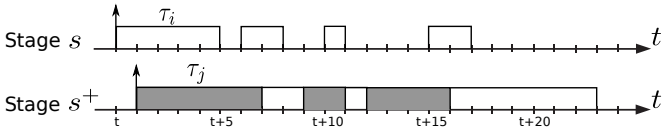


Fig. 3: Illustrative example for Lemma 3

We illustrate Lemma 3 using Figure 3. Task τ_i sends flits on stage s, and suffers blockage from task τ_j that causes interference on stage s^+. The buffer space available for τ_i at $\underset{s}{DR}$, $\underset{s^+}{VC_i} = 5$. Hence, τ_i, while being blocked, can send five flits before $\underset{s}{DR}$ is full. Task τ_j blocks τ_i from $t+1$ to $t+7$, i.e., a blockage of six flits. Task τ_i can send five flits between t and $t+5$, after which any blockage creates a gap during the transmission of τ_i on stage s. Task τ_j sends 12 flits on stage s^+. The blockage suffered by τ_i is equal to $12 - \underset{s^+}{VC_i} = 12 - 5 = 7$ time units.

B. Credit Feedback Delay

So far, we have computed the blockage resulting from back-pressure in the network on any stage of the task's path. We, however, did not consider the credit feedback delay associated with the credit flow-control mechanism. Apart from the blockage that we computed in Lemmas 3, there is a credit feedback delay that is incurred every time the upstream buffer of any stage runs out of credit.

Lemma 4. *A task τ_i, without suffering any interference or blockage, can continuously send flits on stage s, only if* $\underset{s^+}{\mathrm{VC}_i} \geq \mathrm{CF} + 1$.

Proof: Assume that the first flit of τ_i is sent at time t on stage s from the upstream buffer and reaches the downstream buffer at time $t+1$. Before sending the first flit, the upstream buffer has $\underset{s^+}{VC_i}$ credits equivalent to the size of the downstream buffer. The upstream router decrements a credit at time t as it sends a flit out. Since there is no interference or blockage, then the downstream router can send out the received flit at time $t+1$ on stage s^+. The credit corresponding to the first flit is sent upstream at time $t+1$, and is received by the upstream buffer after CF time units, i.e., at time $t+1+CF$. The upstream buffer will lose the last of its initial $\underset{s^+}{VC_i}$ credits at time $t + \underset{s^+}{VC_i} - 1$ while sending out the flit number $\underset{s^+}{VC_i}$. To be able to send a flit at time $t + \underset{s^+}{VC_i}$, the upstream buffer must receive the credit corresponding to the first flit that it sent out by time $t + \underset{s^+}{VC_i}$ at the latest. Therefore, for τ_i to continuously send flits on stage s, the following condition must be satisfied: $t+1+CF \leq t + \underset{s^+}{VC_i}$, i.e., $1 + CF \leq \underset{s^+}{VC_i}$. ■

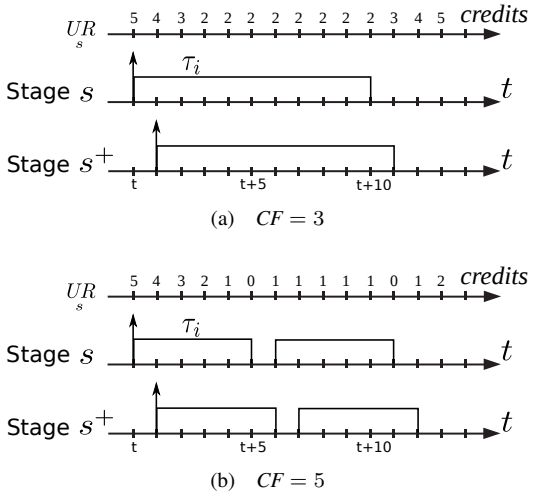


Fig. 4: Illustrative example for Lemma 4

We illustrate Lemma 4 using Figure 4. Task τ_i suffers no interference on stages s and s^+, and the buffer space available for τ_i at $\underset{s}{DR}$ is $\underset{s^+}{VC_i} = 5$. Hence, the upstream router $\underset{s}{UR}$ of stage s starts with five credits at time t. In Figure 4a, the credit feedback delay is $CF = 3$, hence, the relation $\underset{s^+}{VC_i} \geq CF + 1$ is satisfied. When the downstream router of stage s, sends out the first flit at time $t+1$, it sends a credit upstream that is received at time $t+4$. Therefore, the credit being used by $\underset{s}{UR}$ at time $t+4$ gets replaced, and the number of available credits stays at two. This continues until task τ_i sends all of its flits on stage s. In Figure 4b, the credit feedback delay is $CF = 5$, hence, violating the relation $\underset{s^+}{VC_i} \geq CF + 1$. The first credit sent upstream by $\underset{s}{DR}$ at time $t+1$ will be received at time $t+6$. Hence, $\underset{s}{UR}$ will run out of credits at time $t+5$, and the transmission of the flits of τ_5 will be interrupted.

Lemma 5. *When considering credit feedback delay, and under the condition* $\underset{s^+}{\mathrm{VC}_i} \geq \mathrm{CF}+1$*, the worst-case blockage,* $\underset{s}{\mathrm{IB}_i}(\underset{s}{B_i})$*, suffered by task τ_i due to back-pressure on stage s increases by* $\mathrm{CF}+1$ *time units.*

Proof: Assume the first flit of τ_i is sent at time t from the upstream buffer $\underset{s}{UR}$ of stage s. The earliest time at which

the credit for the first flit will be replaced is $t + 1 + CF$. At time $t + CF$, the number of credits will drop to $VC_i - CF$. From Lemma 4, we know that if there is no interference or blockage, flits will be continuously sent. In that case, at time $t + 1 + CF$, a credit will be sent and another will be received, fixing the number of credits at $VC_i - CF$. This will continue as long as no interference or blockage is suffered on stage s.

Consider the time instant t_c which is the first time instant at which the number of credits available in UR becomes one. The least amount of blocking needed to reach the time instant t_c can be obtained by computing the drop required in the number of credits to reach one credit. This drop is equal to $VC_i - CF - 1$ credits. Since a flit is sent in one time unit, then a blockage of $VC_i - CF - 1$ flits is required to reach the time instant t_c. Note that if $VC_i = CF + 1$, the number of credits will drop to one after CF time units without any blockage ($VC_i - CF - 1 = 0$).

Any blockage suffered, beyond the $VC_i - CF - 1$ flits (required to reach t_c), will cause the number of credits to drop to zero. This creates a gap (empty time slot) on stage s. Hence, any blockage suffered, beyond a blockage of $VC_i - CF - 1$ flits, creates a gap on stage s. The size of this gap is equivalent to the number of flits causing the blockage.

Consider the time $t_f > t_c$ at which the last flit sent by task τ_i on stage s is received by DR. Between t and t_f, task τ_i is either sending flits on stage s or suffering blockage (ignoring interferences on stage s). The blockage suffered in this time interval is only the blockage in excess of $VC_i - CF - 1$ flits. We showed in Lemma 3 that task τ_i suffers blockage in excess of VC_i. Therefore, the extra blockage suffered due to credit feedback delay is equal to $VC_i - (VC_i - CF - 1) = CF + 1$.

So far, we have ignored interference suffered on stage s. Any interference suffered on stage s will only stop UR from sending flits. This means that UR will stop using credits and can only gain credits. In that case, only more blockage might be needed until the number of credits drops to one and the blockage starts creating gaps on stage s. This blockage will always be less than or equal to the interference suffered. Hence, the number of flits $VC_i - CF - 1$, beyond which blockage causes gaps in the execution of τ_i on stage s, can only increase in the presence of interference on stage s. Therefore, $CF+1$ is an upper bound to the increase in blockage suffered by task τ_i on stage s due to credit feedback delay. ■

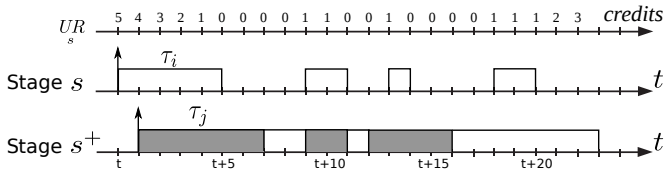


Fig. 5: Illustrative example for Lemma 5

We illustrate Lemma 5 using Figure 5, which is a rework of the example in Figure 3 while taking into account a credit feedback delay $CF = 2$. Task τ_i suffers blockage from τ_j and runs out of credit at UR at time $t + 5$. After a blockage of six flits, the first credit is sent upstream at time $t + 7$ and is received at time $t + 9$. Between $t + 5$ and $t + 9$, τ_i suffers a blockage of four flits which is equivalent to $6 - VC_i + CF + 1 = 6 - 5 + 2 + 1 = 4$. Any further blockage by τ_j after time $t+7$ creates an equivalent gap on stage s. The figure shows how any blockage beyond $VC_i - CF - 1 = 5 - 2 - 1 = 2$ flits, creates gaps in the transmission of the flits of τ_i on stage s. The total blockage suffered by τ_i is, therefore, $12 - VC_i + CF + 1 = 12 - 5 + 2 + 1 = 10$ flits.

C. The Worst-Case Latency Analysis

We can now extend SLA to account for buffer space restrictions. Theorem 1 computes the modified WCL.

Theorem 1. *The WCL R_i of a task τ_i along its path δ_i and under the condition $\mathrm{VC}_i \geq \mathrm{CF} + 1$, is given by:*

$$R_i = \max_{p=1\ldots p_{B,i}} (w_i(p) - (p-1) * T_i + J_i^R) + |\delta_i| - 1$$

where

$$w_i(p) = I_i(p) + \mathrm{IB}_i(w_i(p)) + p * L_i$$

$$I_i(p) = I_{i^-}(p^-) + \sum_{\forall \tau_j \in \mathbb{S}_i^D} \left\lceil \frac{w_i(p) + J_j^R + J_j^I}{T_j} \right\rceil * L_j - \sum_{\forall \tau_j \in \mathbb{S}_i^D \cap \mathbb{S}_{i^-}^D} \left\lceil \frac{w_{i^-}(p^-) + J_j^R + J_j^I}{T_j} \right\rceil * L_j$$

$$\mathrm{IB}_i(w_i(p)) = \begin{cases} 0 & \text{if } \mathrm{VC}_{i^+} \geq p_{B,i} * L_i \\ \max\left(0, \sum_{\forall \tau_j \in \mathbb{S}_{i^+}^D \setminus \mathbb{S}_i^D} \left\lceil \frac{w_i(p) + J_j^R + J_j^I}{T_j} \right\rceil * L_j + \mathrm{IB}_{i^+}(w_i(p)) - \mathrm{VC}_{i^+} + \mathrm{CF} + 1\right) & \text{otherwise} \end{cases}$$

$$p^- = \begin{cases} p & \text{if } p \leq p_{B,i^-} \\ p_{B,i^-} & \text{otherwise} \end{cases}$$

Proof: Equation 2 computes the WCL of τ_i using SLA without taking blockage into account. To compute the worst-case completion time $w_i(p)$ of each job p in the busy interval of τ_i, we compute the blockage suffered by p only during $w_i(p)$. We add the term $IB_i(w_i(p))$ to account for the blockage suffered by job p of task τ_i due to buffer space restrictions. In Lemma 2, we showed that the blockage $IB_i = 0$ when $VC_i \geq p_{B,i} * L_i$. Using Lemma 5, the blockage computed in Lemma 3 increases by $CF + 1$ time units. Hence, the formulation of $IB_i(w_i(p))$ as shown above. ■

Theorem 2. *The WCL computed using FLA is an upper bound to that computed using SLA while considering buffer space restrictions.*

Proof: The blockage computed using Theorem 1 is maximized when $-VC_{i}^{s^+} + CF + 1 = 0$. In that case, computing the blockage on one stage becomes equivalent to accounting for interferences that occur on subsequent stages. FLA [3] treats the task's route as a single communication resource, and, hence, assumes that all interferences occur on this single resource. Therefore, in the worst-case, accounting for buffer space restrictions using SLA results in accounting for interferences in a manner equivalent to that of FLA. ■

V. Buffer Space Allocation Algorithm

Normally, there is a limit on the buffer space available for each router in the pipelined interconnect. This buffer space available to a router is distributed between the VCs of the tasks whose routes include this particular router. Increasing the buffer space for a task can help reduce the blockage that this task suffers. A buffer space allocation algorithm should achieve the following objectives:

Objective 1. *Schedule all tasks in the task set to be deployed on the interconnect.*

Objective 2. *Minimize buffer space usage in the routers.*

Optimally, the algorithm would check every possible buffer configuration to find the least buffer space usage while scheduling all tasks of the task set. Such algorithm, however, would be highly exponential. Therefore, we propose a polynomial time algorithm that attempts to achieve these objectives using observations from the WCL analysis.

The new interferences suffered by the task under analysis on a stage s^+ contributes to the blockage suffered on the preceding stage s. The blockage contributing to the WCL is that in excess of $VC_{i}^{s^+} - CF - 1$. Hence, increasing the size of the buffer $VC_{i}^{s^+}$ in the router leading to the stage s^+ reduces the suffered blockage. Therefore, if a task is unschedulable, the proposed algorithm will primarily attempt to gradually increase the buffer space available in a VC preceding a stage on which interference occurs.

Algorithm 1 shows the proposed buffer space allocation algorithm. The inputs to the algorithm are the set of tasks Γ, the limit B_{lim} on the buffer space per router, and the step B_{step} by which the algorithm increments buffer space in VCs. The efficiency of the buffer space allocation can be increased by decreasing B_{step} at the expense of the computation time of the algorithm. The output from the algorithm is a buffer space assignment to the VCs along the route of each task such that the tasks are schedulable. The algorithm uses the helper function, *buffer*, which returns the total buffer space usage of VCs in a specific router.

The algorithm starts by assigning the size of all VCs to $CF + 1$ flits. This is the minimum size of a VC for which Theorem 1 holds (line 1). The algorithm loops on all tasks in order of decreasing priority (line 2), and computes the WCL of each task using Theorem 1 (line 3). The algorithm checks whether each task meets its deadline (line 4). If a task does not meet its deadline ($R_i > D_i$), the algorithm will start increasing the buffer space usage for this task. The algorithm modifies the buffer size of one VC at a time. The VC that the algorithm chooses to modify must have room for increasing its buffer space, i.e., $buffer(UR_{s}) < B_{lim}$. Amongst all VCs of a task, the algorithm chooses to modify the VC preceding the stage suffering most interference to reduce blockage. If multiple stages suffer the same interference, the algorithm will choose to modify the VC belonging to the router with the least buffer space usage (line 8). After selecting the VC to modify, the algorithm will increase its buffer size by the smaller of B_{step} and the buffer space available in the router before reaching B_{lim} (line 13). Note that if the algorithm cannot find a VC to modify, this means that all VCs of the task have allocated their maximum possible buffer space. In such case, the task and the task set are unschedulable, and the algorithm exits (line 15). After each VC modification, the algorithm re-computes R_i to check whether the task meets its deadline.

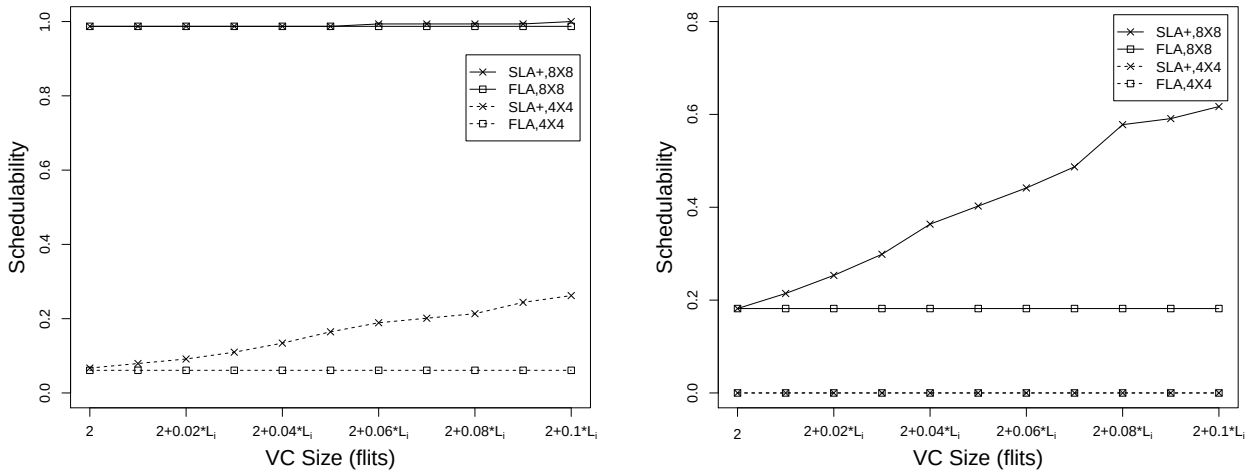
Algorithm 1 BUFFER SPACE ALLOCATION

Input: $\Gamma, B_{lim}, B_{step}$
Output: $VC_{i}^{s},\ \forall s \in \delta_i,\ \forall \tau_i \in \Gamma$

1: Set $VC_{i}^{s} = CF + 1,\ \forall s \in \delta_i,\ \forall \tau_i \in \Gamma$
2: **for all** $\tau_i \in \Gamma$ **do**
3: Compute R_i
4: **while** $R_i > D_i$ **do**
5: $s_{mod} = NULL$
6: $I_{s_{mod}}^{\ i} = -1$
7: **for all** $s \in \delta_i$ **do**
8: **if** $buffer(UR_{s}) < B_{lim}$ **and** $(I_{s}^{i} > I_{s_{mod}}^{\ i}$ **or** $(I_{s}^{i} = I_{s_{mod}}^{\ i}$ **and** $buffer(UR_{s}) < buffer(UR_{s_{mod}}))$ **then**
9: $s_{mod} = s$
10: **end if**
11: **end for**
12: **if** $s_{mod} \neq NULL$ **then**
13: $VC_{s_{mod}}^{\ i} = VC_{s_{mod}}^{\ i} + \min(B_{step}, B_{lim} - buffer(UR_{s}))$
14: **else**
15: Set Γ unschedulable and exit
16: **end if**
17: Compute R_i
18: **end while**
19: **end for**
20: **return** $VC_{i}^{s},\ \forall s \in \delta_i,\ \forall \tau_i \in \Gamma$

VI. Experimentation

We quantitatively evaluate our proposed extensions to SLA (referred to in this section as SLA+ for clarity). We also evaluate the proposed buffer space allocation algorithm. We perform the evaluation on a set of synthetic benchmarks as in [18]. We perform our experiments on 4×4 and 8×8 instances of the priority-aware NoC. Our goals from these experiments are to:



(a) U=1210%, 100 tasks (b) U=2410%, 100 tasks

Fig. 6: Schedulability results for SLA+ and FLA against VC size

1) Demonstrate the feasibility of considering buffer space restrictions in the WCL analysis
2) Evaluate the performance of SLA+ compared to FLA when taking buffer space restrictions into account
3) Compare the buffer space assigned using the proposed allocation algorithm against SLBA and FLBA [10]
4) Compare the computation times of the proposed algorithm against SLBA and FLBA

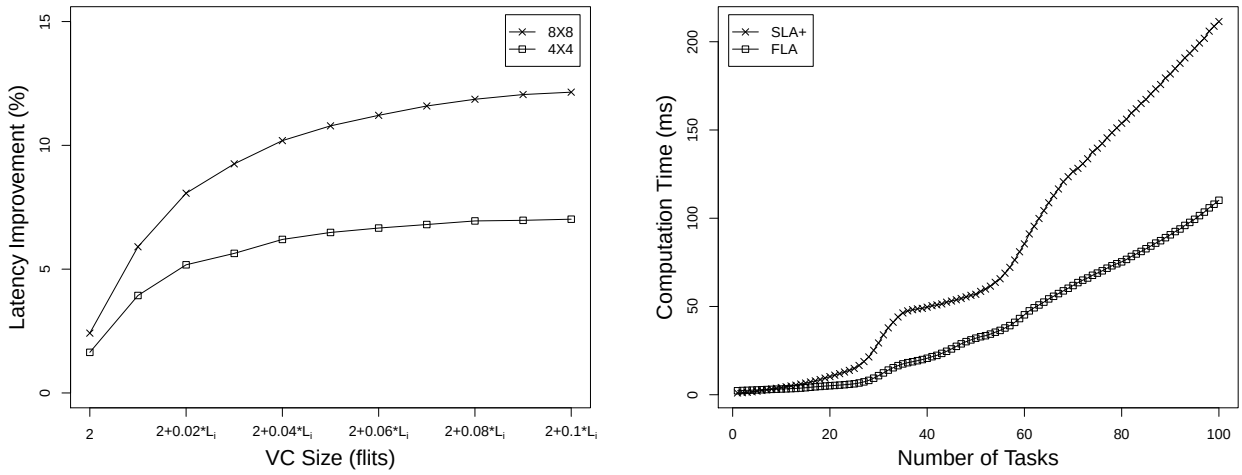
We vary the following factors in our experiments:

1) The number of communications tasks varies from 1 task to 100 tasks (in steps of 1).
2) The source and destination pairs of the tasks are randomly mapped to the NoC.
3) The routes for the tasks are computed using a shortest-path algorithm.
4) A uniform random distribution is used to assign periods (or minimum interarrival time for sporadic tasks) T_i to communication tasks in the range (1000,1000000).
5) The task's deadline D_i is an integer multiple of its period. This includes the case when $D_i = T_i$ (no self-blocking). Note that a larger deadline is more general because a job of task τ_i can suffer interference from jobs of higher priority tasks as well as jobs of the same task τ_i (self-blocking).
6) An arbitrary priority assignment scheme is used for selecting task priorities.
7) The utilization of the NoC's communication resources varies from 10% to 6000% (in steps of 60%). The full utilization of a single communication resource is represented by 100% utilization. Thus, the full utilization of a 4×4 mesh, for instance, is represented by 2400%.
8) The credit feedback delay *CF* is equal to the time taken to send one flit.
9) VC sizes are increased uniformly in 10 steps starting from two flits ($CF + 1$) with a step equal to $\frac{1}{100}$ from the size of a packet.
10) We have 400000 possible configurations, and we generate 100 different test cases for each configuration.

We use the following evaluation metrics:

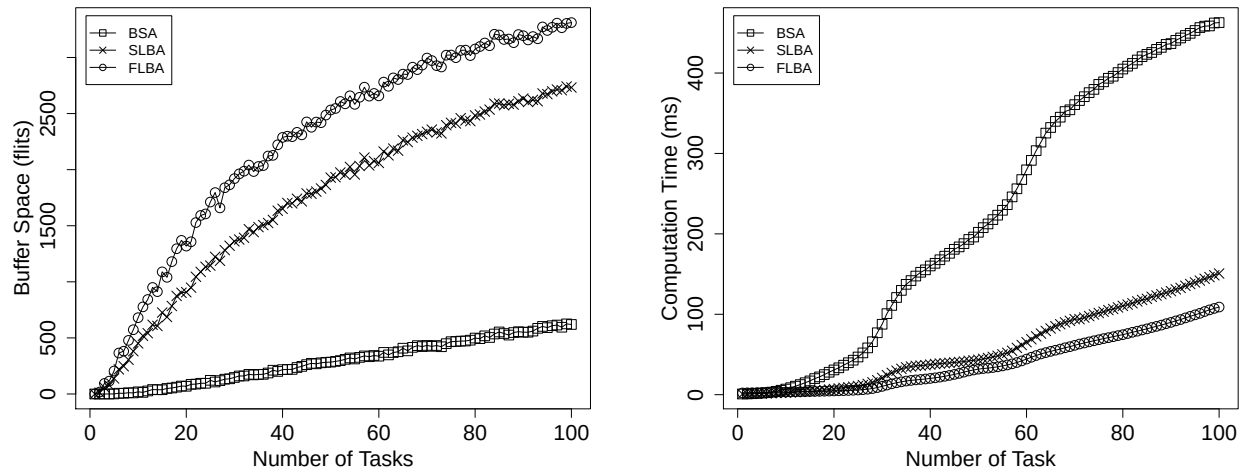
- **Schedulability:** A test case will be unschedulable if one of the tasks in its task set is unschedulable. The schedulability metric is a measure of the percentage of schedulable test cases for a particular configuration.
- **Improvement in WCLs:** For each test case, we compute the WCL of each task using both SLA+ and FLA. Remember that FLA is still an upper bound to SLA even when buffer space restrictions are considered, i.e., FLA is an upper bound to SLA+. We report the average improvement for a test configuration. This metric is only valid for schedulable tasks.
- **Analysis time:** This is the time taken to compute the latency bounds for all tasks in a test case using both SLA+ and FLA. For any given configuration, we report the average analysis time over all test cases.
- **Buffer space requirements:** The buffer space requirement for a particular task is the sum of the buffer space needed at all the virtual channels along the task's route. For each configuration, we report the average buffer space requirement across all test cases.
- **Algorithm Computation time:** This is the time taken to allocate buffer space using the proposed algorithm for all tasks in a test case. For any given configuration, we report the average computation time over all test cases.

Schedulability: Figures 6a and 6b show the schedulability against the buffer size of the VCs for task sets with 100 tasks and utilizations 1210% and 2410%, respectively. We use two different utilizations because for 4×4 NoCs the schedulability improvement can be highlighted at lower utilizations. A higher utilization highlights the schedulability improvement for 8×8 NoCs. Note that FLA does not consider buffer sizes, and, hence, the schedulability result does not change for different buffer sizes. Also note that L_i is computed using the utilization



(a) U=3610%, 100 tasks (b) Computation time

Fig. 7: Latency and computation time results for SLA+ and FLA



(a) U=910%, 100 tasks (b) Computation time

Fig. 8: Results for the buffer space allocation algorithm

and periods of the tasks. At a communication utilization of 1210%, SLA+ improves schedulability over FLA only slightly for 8×8 NoC instances as buffer space increases. For 4×4 NoC instances, the schedulability of SLA+ increases from 1.1 times that of FLA for a buffer size of two flits, up to 4.3 times for a buffer size equal to $2 + 0.1 * L_i$. The reason is that as more buffer space becomes available, blockage decreases and SLA+ computes tighter WCL bounds which increases schedulability. For a communication utilization of 2410%, both SLA+ and FLA cannot schedule any task sets with 100 tasks for 4×4 NoC instances due to high interferences. In 8×8 NoC instances, the schedulability of SLA+ is equal to that of FLA for a buffer size of two flits, and increases to 3.4 times that of FLA for a buffer size equal to $2 + 0.1 * L_i$. The schedulability of SLA+ increases over FLA as more buffer space is available for the tasks.

Latency Improvement: Figure 7a shows the latency improvement against buffer sizes for task sets with 100 tasks and a randomly chosen communication utilization of 3610%. As more buffer space is available, the improvement in latencies computed by SLA+ over FLA increases from 1.6% to about 7.0% for 4×4 NoC instances. For 8×8 NoC instances, this improvement increases from 2.4% to 12.1% as more buffer space is available for the tasks. The latency improvement increases because when more buffer space is available, SLA+ computes tighter WCL bounds.

Analysis Time: Figure 7b compares the average computation times of both SLA+ and FLA. The analysis time for SLA+ is approximately double that of FLA, which is acceptable given the quality of the results of SLA+.

Buffer Space Requirements: Figure 8a shows the average buffer spaces computed using the buffer space allocation algorithm, SLBA, and FLBA against the number of tasks at a random network utilization of 910%. As the number of

tasks increases, the required buffer space increases due to the increase of virtual channels and increase of interference in the network. The proposed buffer space allocation algorithm shows a large reduction in the buffer space requirements. On average, the buffer space allocation algorithm reduces the buffer space requirements by 85.1% and 88.9% compared to SLBA and FLBA, respectively.

Algorithm Computation Time: Figure 8b compares the average computation times of the buffer space allocation algorithm to SLBA and FLBA. The algorithm's computation time is about 3.8 times that of SLBA. This is a reasonable increase in computation time given the large reductions in the computed buffer space requirements.

Summary: The main conclusion from our experiments is that SLA+ is able to schedule task sets for buffers with a size of only two flits. The schedulability of SLA+ improves over FLA by only 0.1% for a buffer size of two flits and increases up to 11.9% for a buffer size equal to $2 + 0.1 * L_i$. Compared to FLA, SLA+ reduces WCLs by 1.1% for two flit buffers, and by 4.7% for buffer sizes equal to $2 + 0.1 * L_i$. Over all test cases, the average analysis times using SLA+ and FLA are 81.3 mS and 40.5 mS, respectively. The buffer space allocation algorithm reduces the required buffer space to schedule task sets by 85.1% and 88.9% compared to SLBA and FLBA, respectively. Over all test cases, the average computation time of the buffer space allocation algorithm is 222.7 mS.

VII. Conclusion

Recent research develops techniques for computing WCL bounds for communication latencies in priority-aware NoCs. This facilitates adopting priority-aware NoCs as a platform for deploying real-time applications. These WCL analysis techniques, however, assume the existence of enough buffer space in the network VCs to prevent back-pressure. Recently, we computed these buffer space bounds in the virtual channels such that back-pressure does not happen in the network, and, hence, providing guarantees for the validity of the WCL analyses [10]. In practice, however, the network buffer space constraints might be smaller than these computed buffer space bounds. Therefore, in this work, we present theorems that incorporate buffer space limitations into the WCL bounds computed using SLA. SLA+ was able to schedule task sets with buffer spaces as small as two flits per virtual channel. We also present a polynomial time buffer space allocation algorithm that uses the extensions made to SLA. The proposed algorithm reduces the required buffer space to schedule task sets by 85.1% and 88.9% compared to SLBA and FLBA, respectively.

Appendix
Summary of Symbols

This section summarizes the symbols used in this paper. Table I shows these symbols.

TABLE I: Symbol table

Symbol	Definition
Γ	A set of communication tasks deployed on the pipelined resource model.
τ_i	A communication task i.
P_i	Priority of communication task τ_i.
T_i	Period or minimum interarrival time between jobs of communication task τ_i.
D_i	Deadline of communication task τ_i.
J_i^R	Release jitter of communication task τ_i.
L_i	Worst-case latency of communication task τ_i on a single communication resource.
δ_i	A path formed of a series of contiguous communication resources of communication task τ_i from a source stage s_1 to a destination stage $s_{\lvert\delta_i\rvert}$.
s	A stage on the path of a communication task.
s^-	A stage preceding stage s on the path of a communication task.
s^+	A stage succeeding stage s on the path of a communication task.
$\underset{s}{\mathbb{S}_i^D}$	Set of directly interfering tasks with task τ_i on stage s of its path δ_i.
$\underset{s}{\mathbb{S}_i^I}$	Set of indirectly interfering tasks with task τ_i on stage s of its path δ_i.
$\underset{s}{J_i^I}$	Indirect interference jitter for a task under analysis on stage s through a directly interfering task τ_i.
R_i	Worst-case latency of communication task τ_i along its path δ_i.
$\underset{s}{R_i}$	Worst-case latency of communication task τ_i on stage s of its path δ_i.
$\underset{s}{I_i}$	The worst-case interference suffered by communication task τ_i from higher priority tasks on stage s of its path δ_i.
$\underset{s}{B_i}$	Busy period on stage s on the path of communication task τ_i.
$\underset{s}{w_i(p)}$	The worst-case completion time of each job p of communication task τ_i on stage s from the start of the busy period $\underset{s}{B_i}$.
$\underset{s}{I_i(p)}$	The worst-case interference suffered by job p of communication task τ_i from higher priority tasks on stage s of its path δ_i.
$\underset{s}{p_{B,i}}$	Maximum value of p for task τ_i on stage s.
$\underset{s}{VC_i}$	The buffer space in the virtual channel of task τ_i in the priority-aware router sending flits on stage s of the task's path δ_i.
$\underset{s}{IB_i}$	The blockage suffered by task τ_i on stage s of its route δ_i due to limited buffer space in the downstream router of stage s.
CF	The credit feedback delay of the credit-based task control mechanism in the priority-aware network.
$\underset{s}{UR}$	The upstream router of stage s.
$\underset{s}{DR}$	The downstream router of stage s.

References

[1] D. Sanchez, G. Michelogiannakis, and C. Kozyrakis, "An analysis of on-chip interconnection networks for large-scale chip multiprocessors," *ACM Trans. Archit. Code Optim.*, vol. 7, no. 1, pp. 4:1–4:28, May 2010.

[2] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal Network on Chip: Concepts, Architectures, and Implementations," *IEEE Design and Test*, vol. 22(5), 2005.

[3] Z. Shi and A. Burns, "Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching," in *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, 2008.

[4] H. Kashif, S. Gholamian, and H. Patel, "SLA: A Stage-level Latency Analysis for Real-time Communication in a Pipelined Resource Model," *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 1177–1190, April 2014.

[5] H. Kashif, S. Gholamian, R. Pellizzoni, H. D. Patel, and S. Fischmeister, "ORTAP:An Offset-based Response Time Analysis for a Pipelined Communication Resource Model," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2013.

[6] Z. Shi and A. Burns, "Priority assignment for real-time wormhole communication in on-chip networks," in *Proceedings of the 2008 Real-Time Systems Symposium*, ser. RTSS '08. IEEE Computer Society, 2008.

[7] ——, "Real-time communication analysis with a priority share policy in on-chip networks," in *Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems*, ser. ECRTS '09. IEEE Computer Society, 2009.

[8] L. Ni and P. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, 1993.

[9] W. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, 1992.

[10] H. Kashif and H. Patel, "Bounding buffer space requirements for real-time priority-aware networks," in *Proceedings of the Asia South Pacific Design Automation Conference*, SunTec, Singapore, January 2014.

[11] L. Benini and G. De Micheli, "Networks on chip: a new paradigm for systems on chip design," in *Proceedings of the conference on Design, Automation and Test in Europe*, 2002.

[12] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *JOURNAL OF SYSTEMS ARCHITECTURE*, vol. 50, 2004.

[13] Y.-C. Lan, S.-H. Lo, Y.-C. Lin, Y.-H. Hu, and S.-J. Chen, "Binoc: A bidirectional noc architecture with dynamic self-reconfigurable channel," in *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, May 2009, pp. 266–275.

[14] J. Hu and R. Marculescu, "Application-specific buffer space allocation for networks-on-chip router design," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2004.

[15] A. Kumar, M. Kumar, S. Murali, V. Kamakoti, L. Benini, and G. De Micheli, "A simulation based buffer sizing algorithm for network on chips," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2011.

[16] M. A. Al Faruque and J. Henkel, "Minimizing virtual channel buffer for routers in on-chip communication architectures," in *Proceedings of the conference on Design, automation and test in Europe*, ser. DATE '08. ACM, 2008.

[17] M. Coenen, S. Murali, A. Ruadulescu, K. Goossens, and G. De Micheli, "A buffer-sizing algorithm for networks on chip using TDMA and credit-based end-to-end flow control," in *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '06. ACM, 2006.

[18] S. Manolache, P. Eles, and Z. Peng, "Buffer space optimisation with communication synthesis and traffic shaping for nocs," in *Proceedings of the conference on Design, automation and test in Europe*, ser. DATE '06. European Design and Automation Association, 2006.

[19] J. C. Palencia and M. González Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Proceedings of the IEEE Real-Time Systems Symposium*, ser. RTSS '98. IEEE Computer Society, 1998.

[20] A. Hansson, M. Wiggers, A. Moonen, K. Goossens, and M. Bekooij, "Applying dataflow analysis to dimension buffers for guaranteed performance in networks on chip," in *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, April 2008.

[21] A. Hansson, M. Wiggers, A. Moonen, K. Goossens, and M. Bekooij, "Enabling application-level performance guarantees in network-based systems on chip by applying dataflow analysis," *IET Computers & Digital Techniques*, vol. 3, no. 5, 2009.

[22] M. Bakhouya, S. Suboh, J. Gaber, and T. El-Ghazawi, "Analytical modeling and evaluation of on-chip interconnects using network calculus," in *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, May 2009, pp. 74–79.

[23] H. Kashif, H. D. Patel, and S. Fischmeister, "Using link-level latency analysis for path selection for real-time communication on nocs," in *Proceedings of the Asia South Pacific Design Automation Conference*, February 2012, pp. 499–504.