Received October 5, 2020, accepted October 20, 2020, date of publication October 26, 2020, date of current version November 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3033853

hp-DCFNoC: High Performance Distributed Dynamic TDM Scheduler Based on DCFNoC Theory

TOMÁS PICORNELL, JOSE FLICH, (Senior Member, IEEE), DUATO JOSE, (Senior Member, IEEE), AND CARLES HERNÁNDEZ
Department of Computer Architecture, Universitat Politècnica de València, 46022 Valencia, Spain

Corresponding author: Tomás Picornell (tompic@gap.upv.es)

This work was supported in part by the Secretara de Estado de Investigacin Desarrollo e Innovacin (MINECO) under Grant BES-2016-076885, in part by the European Regional Development Fund (ERDF) under Grant TIN2015-66972-C05-1-R and Grant RTI2018-098156-B-C51, and in part by the EC H2020 European Institute of Innovation and Technology (SELENE) Project under Grant 871467.

ABSTRACT The need for increasing the performance of critical real-time embedded systems pushes the industry to adopt complex multi-core processor designs with embedded networks-on-chip. In this paper we present hp-DCFNoC, a distributed dynamic scheduler design that by relying on the key properties of a delayed conflict-free NoC (DCFNoC) is able to achieve peak performance numbers very close to a wormhole-based NoC design without compromising its real-time guarantees. In particular, our results show that the proposed scheduler achieves an overall throughput improvement of 6.9× and 14.4× over a baseline DCFNoC for 16 and 64-node meshes, respectively. When compared against a standard wormhole router 95% of its network throughput is preserved while strict timing predictability as property is kept. This achievement opens the door to new high performance time predictable NoC designs.

INDEX TERMS Dynamic scheduler, multicore, real-time, tdma, time predictable network.

I. INTRODUCTION

The continuous quest for performance of critical real-time embedded systems pushed the safety-critical systems industry to move from platforms including relatively simple electronic computing units to complex multicore processor designs. The interconnection architecture included in these processor designs has evolved from bus-centric architectures interconnecting a relatively small number of cores to more scalable distributed approaches implementing networks-on-chip (NoCs) [1].

In a context in which NoCs are becoming ubiquitous in safety-critical real-time systems [2]–[4] it becomes mandatory finding NoC designs that provide high quality real-time guarantees. In this paper we aim at achieving this goal and propose a new real-time specific NoC design that provides top peak performance while preserving strict real-time guarantees. Our NoC design uses a dynamic scheduler that builds on top of the delayed conflict-free NoC (DCFNoC) [5]. DCFNoC is a TDM-based NoC with unique features that are exploited by our dynamic scheduler design. The combination of the dynamic scheduler and DCFNoC is termed hp-DCFNoC (high-performance DCFNoC).

As the main contribution in this paper, we propose a novel distributed dynamic scheduler that relies on DCFNoC properties and improves the throughput of the baseline NoC by taking advantage of unused TDM slots. We implement the scheduler design in synthesizable verilog RTL showing the feasibility of this approach and compare its performance against the one provided by DCFNoC and a regular wormhole NoC.

Our results show that hp-DCFNoC maximizes performance when using DCFNoC, achieving an overall throughput improvement of 6.9× and 14.4× over a baseline DCFNoC for 16 and 64-node meshes, respectively. Experimental results also confirm that hp-DCFNoC is able to achieve the same performance guarantees that DCFNoC. Additionally, we show how hp-DCFNoC achieves a peak throughput (0.45 flits/cycle/node), close to maximum throughput in a 4 × 4 wormhole NoC. In general terms, 95% of achievable throughput by a wormhole router is guaranteed by hp-DCFNoC.

The associate editor coordinating the review of this manuscript and approving it for publication was Rongni Yang.

The rest of this paper is organized as follows. Section II briefly describes DCFNoC theory. Section III presents the distributed dynamic scheduler design. Section IV provides a performance evaluation of the proposed scheduler. Section V discusses related work and, finally, conclusions are given in Section VI.

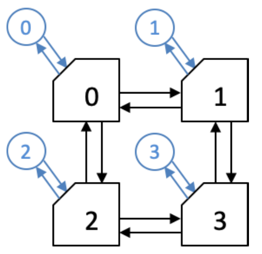
II. DELAYED CONFLICT-FREE NETWORK

In this section we briefly describe the methodology for designing TDM-based networks relying on the DCFNoC approach. DCFNoC builds a network in which conflict-free transmission is guaranteed by using a time-division multiplexing (TDM) window. DCFNoC relies on the utilization of channel dependency graphs (CDGs) to identify the existing message dependencies (contention) and eliminate message conflicts by the simple introduction of delays at the router output ports. The CDG helps to identify where conflicts occur in the NoC and how these conflicts are always the consequence of dependencies between messages reaching the same destination with a variable number of hops. In this context, the introduction of delays at the output ports of specific routers located at particular positions in the NoC, ensures transmissions are naturally serialized and conflicts are avoided by design.

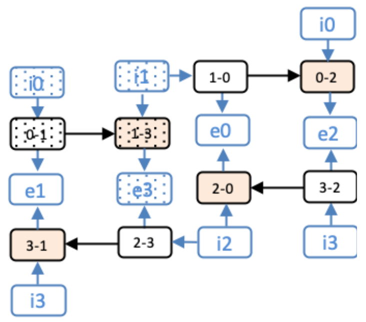
In order to describe DCFNoC we assume a 2×2 mesh using the Dimension Order Routing DOR [6] algorithm (see Figure 1a). From this topology and based on the routing algorithm we build its channel dependency graph (*CDG*). The *CDG* can be used to derive the paths for every source-destination pair of end nodes (see Figure 1b). For instance, path $0 \rightarrow 3$ will cross links $\{0-1\}$, $\{1-3\}$ whereas path $1 \rightarrow 3$ will use link $\{1-3\}$. Notice that these two paths, with dots, may create conflicts since they share $\{1-3\}$ link and ejection link (dotted $e3$) at router 3 (ejection links are represented by ($e\#$).

From the *CDG*, the DCFNoC approach follows two steps. In the first step we layer the *CDG* to obtain an equivalent layered *CDG* (CDG_l). To do so, we assign a layer ($L\#$) to each link (nodes in the *CDG*) following a partial order as determined by the routing algorithm (see Figure 1c). If we assume links introduce one cycle delay, each layer then represents one cycle delay. Notice that building a layered *CDG* is always possible when the *CDG* is acyclic [6] and thus, DCFNoC is compatible with any deterministic or partly adaptive deadlock-free routing algorithm. Note that a mechanism to avoid deadlocks is a limitation implemented in the majority of existing NoC. The CDG_l shows arranged (downwards in the figure) the paths for every source-destination pair of end nodes following the routing algorithm. Assuming only one node injects a message at a time, potential conflicts will only occur due to channels connecting non-consecutive layers (represented by red dashed arrows in Figure 1c).

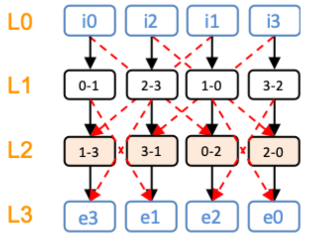
To avoid those potential conflicts, in the second step, the CDG_l is modified by inserting delays to enforce all paths cross all layers, thus, to have the same length. Figure 1e shows how to remove every potential conflict (red arrow)



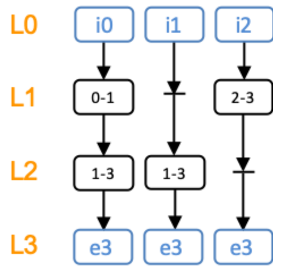
(a) 2×2 mesh topology. End nodes shown as circles.



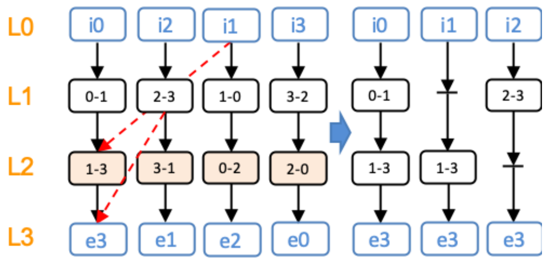
(b) Channel dependency graph of the 2×2 mesh, using DOR routing algorithm. Squares represent links and its label indicates the routers attached to the link.



(c) Layered Channel dependency graph obtained from CDG. Red arrows represent links with potential conflicts between messages.



(d) CDG_{dl} by destination node (paths shown only from all to 3).



(e) From CDG_{l} to CDG_{dl}. In order to remove potential conflicts (red arrows) we need to add delays (horizontal lines).

FIGURE 1. Building a DCFNoC from a 2D mesh with *XY* routing algorithm.

of Figure 1c. In this particular example potential conflicts with end node 3 are shown. As a result, we add one cycle delay (horizontal lines) to remove every potential conflict. Note that we add one delay because in this case arrow indicates channels connecting non-consecutive layers with a jump of one layer. The longer the jump, the more delays we need. The main idea behind DCFNoC is that conflicts can be avoided if transmissions are serialized by forcing all messages to traverse the same number of layers in the same order (from $L0$ to $L3$).

The key idea is to spread the delays among multiple routers for every path between source and destination nodes. The resulting graph after the addition of delays is a delayed layered channel dependency graph CDG_{dl}, see Figure 1d. Note that in this plot only channels and dependencies for paths with destination end node 3 are shown. In CDG_{dl} all paths

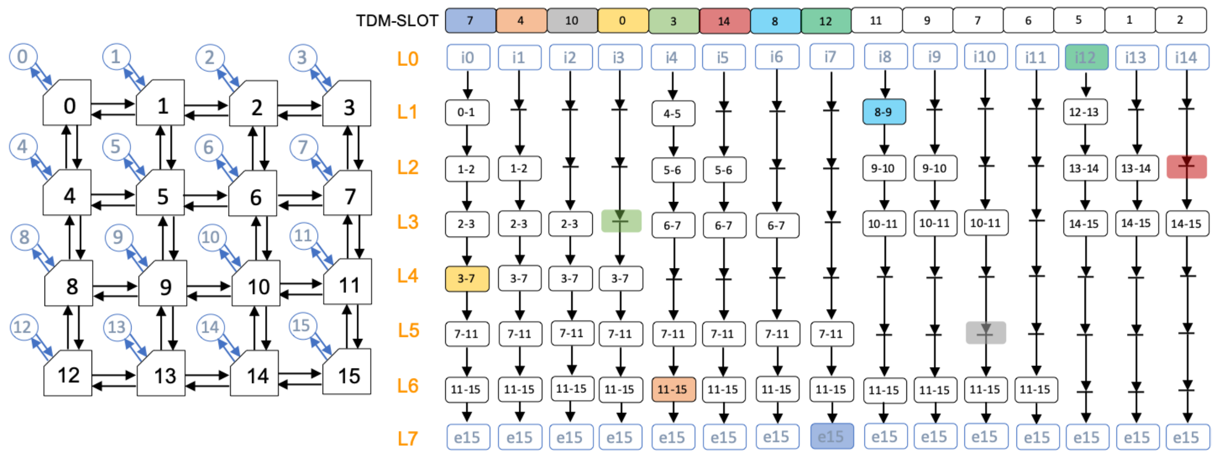


FIGURE 2. *CDG_{dl}* **subgraph by destination for the 4 × 4 mesh topology and the XY routing algorithm (paths shown only from every node to node 15). Colours represent the TDM-slot of a potential arrangement of messages in the network.**

experience the same latency as injection channels are located at $L0$ and ejection channels are located at $L3$. Additional delays are represented by a horizontal line after the arrow. The delay for every possible path is equal to the network diameter plus two (injection and ejection layers/channels).

TDM slots at the end nodes are used to guarantee every node can only inject messages in a given slot and that only one node injects in a particular cycle. Combined with additional delays we enforce the serialization (one-message-per-layer rule). Destination end nodes do not affect slot assignment since conflicts are avoided completely. Figure 2 shows a subgraph of the CDG_{dl} derived from a 4 × 4 mesh topology and the DOR routing algorithm. Only links and dependencies for paths with destination end node 15 are shown in the plot. In the figure every TDM injection slot is associated to one injector ID. As shown in the plot, TDM slots can be assigned arbitrarily with the guarantee that conflicts will not be created within the network. We use colours to represent the location of packets according to the TDM slot assignment. Note that every row represents a network layer in the CDG_{dl}. To avoid conflicts, DCFNoC enforces the one packet per layer rule allowing only one injector per slot. Up to eight messages will be within the network, one from each end node but each message will be on a different layer. Messages from end node 0 will take six cycles as they cross six links. On the other hand, messages from end node 11 will take six cycles but only one network link will be crossed ($\{11 - 15\}$). Five delay cycles are added to the path to globally avoid conflicts.

The way in which delays are implemented is design specific. So, each router will have a predefined output port delay configuration depending of its location within the *CDG*. Notice that each delay (layer) will need a latch to store the message for one cycle. In DCFNoC, extra delays accounted in CDG_{dl} are added to the output port of the router.

DCFNoC methodology is topology agnostic and it only requires that the CDG resulting from a particular topology and routing implementation is acyclic. Figure 3 illustrates

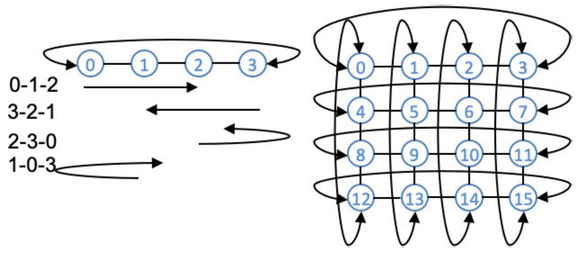


FIGURE 3. **Breaking cycles in bi-torus.**

how DCFNoC can be applied in a bi-torus topology. Routing loops via wrap-around links can be avoided by applying a couple of routing restrictions. For instance, we can force paths through the wraparound links take one direction or another depending whether source end nodes are odd or even.

After breaking cycles (i.e. making the CDG acyclic) the longest path in the topology (network diameter) in a 4 × 4 Bi-torus is four. Thus, the resulting CDG_{dl} will contain 6 layers (from $L0$ to $L5$) as the longest paths in this NoC traverse four links (e.g. from node 0 to node 10). Shorter paths will have additional delay cycles to serialize packets and avoid conflicts. It is important to recall that the delay for every possible path with DCFNoC is equal to the network diameter plus two (injection and ejection layers/channels).

III. A DISTRIBUTED DYNAMIC SCHEDULER DESIGN

DCFNoC limits flit injection to only one node at a time in a given slot. Therefore, although timing guarantees are preserved, network throughput is severally limited to one flit per cycle regardless of network size. hp-DCFNoC overcomes this limitation by introducing a dynamic scheduler design that is able to inject more than one flit per cycle by exploiting the existing conflict-free paths (paths that do not share any network resource between them). In particular, we exploit two conflict-free situations: (1) packets from two nodes injected in the same slot that do not share any resource along their path and (2) messages injected at different cycles. In both cases they will never conflict in DCFNoC. Our dynamic scheduler exploits these two properties to maximize the number of packets that can be injected in the NoC at a given cycle while preserving the real-time properties provided by DCFNoC.

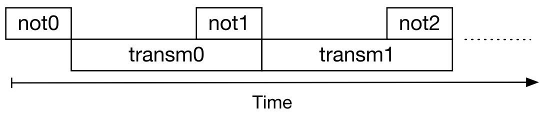


FIGURE 4. Scheduler phases. Notification phase configures next transmission window.

hp-DCFNoC implements a distributed TDM scheduler at network interfaces of every end node. The scheduler uses two DCFNoC networks, one for notification and one for data transmission as shown at the upper part of Figure 4. Two phases are identified. In the notification phase the scheduler determines which routes will be used by each node in a given time slot. If routes are compatible (i.e.. do not generate conflicts) they can be scheduled in the same time slot. In the second phase data is actually transmitted. During the transmission phase all the slots are run. Data transmission and notification phases are overlapped to maximize both throughput and average message latency.

A. SCHEDULER ARCHITECTURE

The proposed scheduler consists of several modules interconnected as shown in Figure 5. This picture shows a detailed architectural diagram of a 4-way distributed TDM scheduler for a 16-node system in a 4×4 mesh network. Each way_i module has a queue with pending messages the node wants to inject (4 messages in this particular configuration). In the scheduler a route scheduling table (RST with as many entries as ways) keeps the following information related to pending messages: valid bit (*valid*), way id (way_{id}), destination node (*dst*), the set of available slots (*slot x*), a selected slot vector (*selected*) and the corresponding id of the selected slot (*slot sel*).

During the scheduling phase, each node receives notification messages from other nodes via the notification DCFNoC. Every time a notification arrives to the scheduler the route checker modules compare the routes information included in the received notification with the one for the pending local messages. A notification message can include several routes. In our implementation we use slots of two cycles for each node to send notifications. Each cycle $ways/2$ routes are notified. By using two cycles the width of the notification network is reduced. The route checker modules determine which routes are disjoint with the pending messages and which ones are incompatible. The selected slot is updated in the RST module using a chain of Fixed Priority Arbiters (FPA). Once notification phase ends, RST information is stored in the data window module. This module injects pending messages in the assigned slot by using an enable signal during the data transmission phase.

In order to maximize performance, the scheduler is pipelined in two stages. In the first stage notifications are received (or injected) and checked. In the second stage the compatible slots are found.

B. NOTIFICATION PHASE

At each network interface newly generated messages arrive in order and are stored in a two slot buffer (way). Each message generates a pending route to be scheduled (the suitable slot must be computed). Routes are determined by a source and destination pair (src-dst). Message destination and way ID are stored in the route scheduling table, which is used to find the most suitable slot. Each RST row contains one control bit per each possible slot in the next transmission window. A control bit set to one means that this route can use that slot ID for transmission. The scheduler must guarantee two conflicting paths do not end up using the same slot. We define as many slots as end nodes and statically assign one slot per node. Each slot will be prioritized by the scheduler to the assigned node. Thus, the scheduler guarantees that at least one node will be able to use its prioritized slot and is irrevocable. This is the most valuable guarantee, no one can use this slot unless it uses a disjoint route. This is key to ensure that DCFNoC timing guarantees are preserved.

The notification phase uses a TDM network in order to let every node send their notifications in turns. To avoid wasting a complete TDM window to notify all nodes we use DCFNoC native support for broadcast. Thus, every node will broadcast its notifications on a single slot to the rest of nodes. Also, DCFNoC guarantees all notifications arrive at the same time to all nodes which simplifies the scheduler design. The notification reception time ($time_i$) identifies the sender node and therefore is equal to *Notification ID*. On every notification window, the first node sending notifications is assigned in a round-robin fashion.

On every notification reception the end nodes process the notifications. The RST is updated taking into account the conflicts that may arise between received notifications and the current assignments of paths to slots. Notice that all nodes update the RST at the same time and via the same manner. Also, each end node checks if there is a pending route (i.e. a route that is waiting to be served) that requests the same network resources to determine if these routes are compatible. The rules used by the scheduler to determine priorities in case of incompatible paths are the following:

- The notified route uses the priority slot of the receiver. (For the examples provided below we consider TDM slot i is assigned to node i by default). Given that these routes are compatible by default no action is required at the receiver side. Figure 6 illustrates this case. Notification node 4 uses priority slot of receiver (slot 6). Receiver has one route in the RST with slot 6. In this case, receiver has the priority to use this slot. When the notification turn arrives, receiver node will send the notification and other nodes using this slot with incompatible routes must disable this selected time slot in the RST.

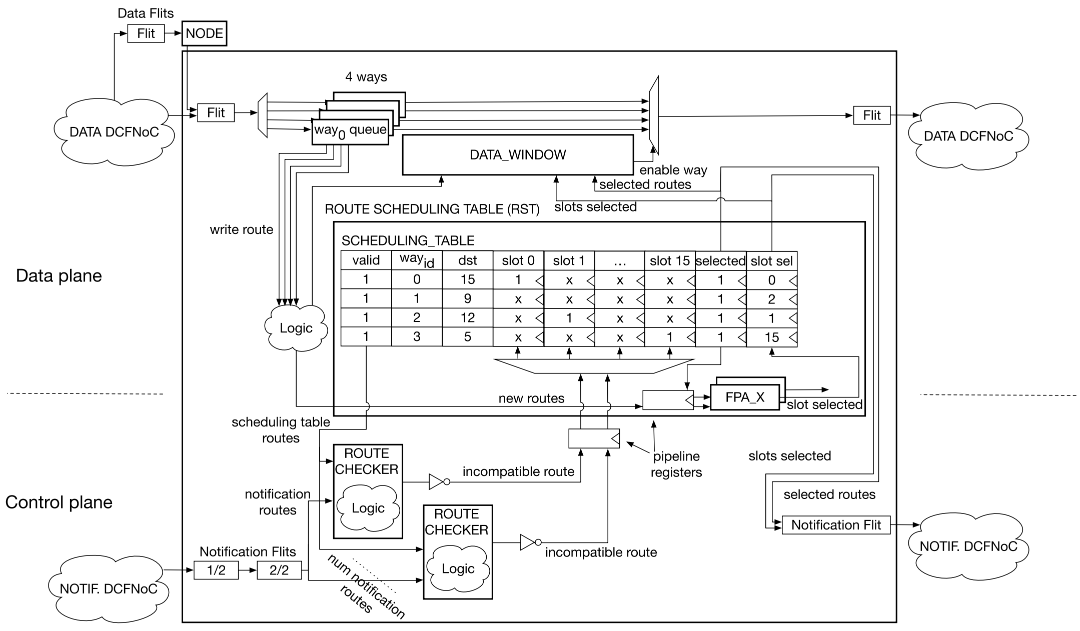


FIGURE 5. Scheduler to optimize DCFNoC performance and resource utilization.

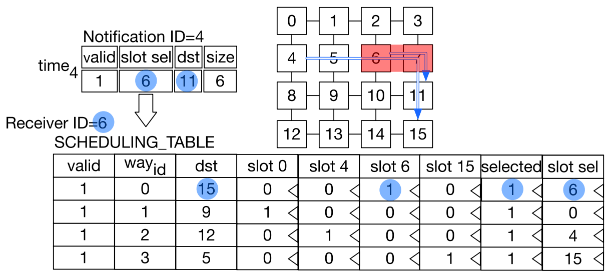


FIGURE 6. First priority rule in case of incompatible paths: notified route uses the priority slot of the receiver.

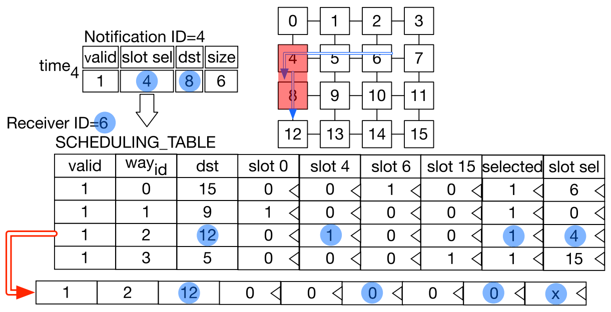


FIGURE 7. Second priority rule in case of incompatible paths: notified route uses the priority slot of the sender.

- The notified route uses the priority slot of the sender. The receiver must disable the requested time slot in the RST. Figure 7 shows an example where notification node 4 uses its own priority slot (slot 4). Receiver (node 6) has one route in the RST with slot 4 and both routes have incompatible paths. In this case, sender has the priority to use this slot and therefore, receiver node must disable this selected time slot in the RST row.

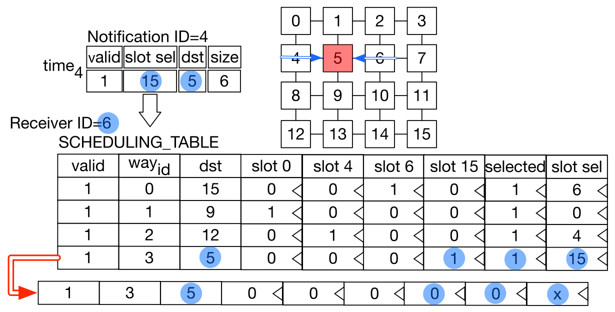


FIGURE 8. Third priority rule in case of incompatible paths: notified route does not use either the priority slot of the sender nor the receiver's one.

- The notified route does not use either the priority slot of the sender or the receiver's one. Figure 8 illustrates this case. The sender is node 4 and uses slot 15. Receiver is node 6 and has one route in the RST with slot 15. Both routes have incompatible paths due to destination node sharing. In this case, the first node that notified the routed (node 4 in our example), has the priority to use this slot, therefore receiver node must disable this selected time slot in the RST row.

Once notification turn arrives, the slot manager on every node selects one data slot for each pending route through a chain of FPA. Slot selection process is based on the updated control bits of the RST. First, by means of Round-Robin arbiter, one pending route is selected and uses the first FPA to select a time slot. The FPA selects the first active bit of each SRT row starting from this node's priority slot. The remaining pending routes use the next entry of the one provided by FPA. It is important to take into account that the slot selection process is exclusive. That is, no pending route can select the

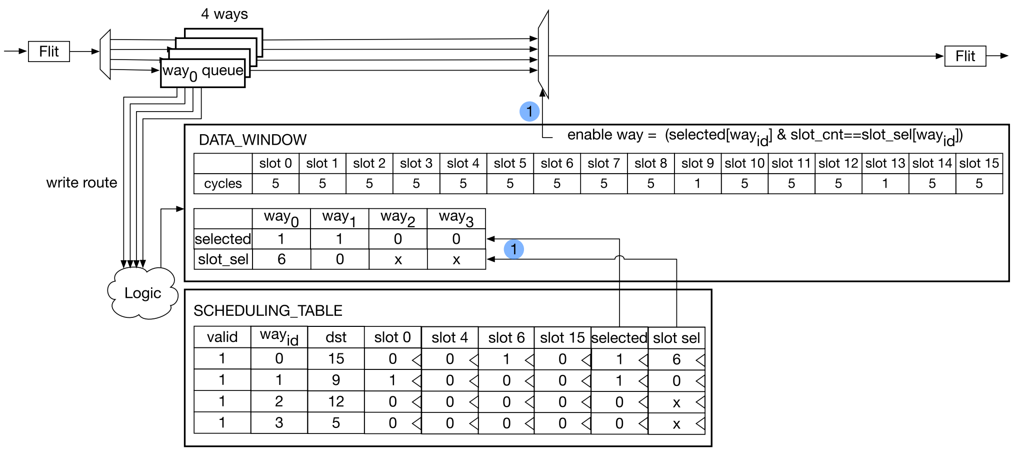

4 ways
Flit
way0 queue
Flit
write route
Logic
1
enable way = (selected[wayid] & slot_cnt==slot_sel[wayid])
DATA_WINDOW
slot 0 slot 1 slot 2 slot 3 slot 4 slot 5 slot 6 slot 7 slot 8 slot 9 slot 10 slot 11 slot 12 slot 13 slot 14 slot 15
cycles 5 5 5 5 5 5 5 5 5 1 5 5 5 1 5 5
way0 way1 way2 way3
selected 1 1 0 0
slot_sel 6 0 x x
SCHEDULING_TABLE
valid wayid dst slot 0 slot 4 slot 6 slot 15 selected slot sel
1 0 15 0 0 1 0 1 6
1 1 9 1 0 0 0 1 0
1 2 12 0 0 0 0 0 x
1 3 5 0 0 0 0 0 x

FIGURE 9. Data phase. At the end of notification phase the slot manager sends information about the slots assigned to each message.

same slot. Once slot selection ends the selected control bit is enabled and the selected slot is stored at the row.

Selection and notification process take two cycles at each node. The process selects and notifies half of the pending routes in the first cycle and the other half in the second cycle.

C. DATA PHASE

After the notification phase each node knows the exact slot in which the different messages have to be scheduled. The duration of the data phase that guarantees all nodes have one slot is a complete TDM window. The size of each slot is the maximum size among all the routes using the slot and depends on the size of messages scheduled for that slot.

To maximize performance transmission phase is overlapped with notification phase. As Figure 9 shows, when a notification phase ends the slot manager module sends information about the slots assigned to each message. Once the new data window is ready the system starts sending stored messages from ways corresponding to an assigned slot.

The upper part of Figure 11 shows how the data window module activates the notification phase of the next data transmission phase before the current transmission window is completed. Note also that shifting notification phase to the end of the current data window transmission maximizes the chances to find compatible routes since more messages are potentially available.

D. ASSIGNMENT OF TDM SLOT PRIORITIES

The success rate of slot assignments for pending routes is significantly influenced by how priority slots are designated to the different nodes. Achieving a higher rate of selected slots requires finding the maximum number of disjoint paths (i.e. without conflict) and for this a policy for slot assignation has to be designed. This assignment is computed by software before the application is deployed.

Let us illustrate this with an example. Figure 10a shows a 4 × 4 network in which packets are routed using XY.

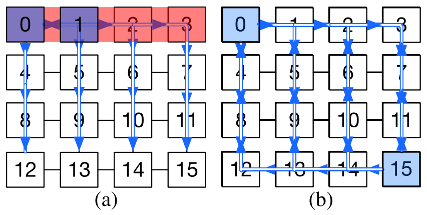


FIGURE 10. Disjoint paths between node 0 and node 1 in (a). In (b) node 0 and node 15. Paths sharing red resources or destination node are incompatible.

As shown in the plot, nodes 0 and 1 have many potential conflicting paths since they share many links to the potential target destinations. On the contrary, as shown in Figure 10b, node 0 and node 15 do not have conflicts except those that target one of these two nodes as destination. For this setup, the best slot assignment is the one that assigns priority slots to nodes with no or few number of sharing resources for all potential target destinations.

In this section, we show how to find the optimal slot assignment for uniform traffic with XY routing in a 2D mesh network. To do so, we have done an exhaustive search with an offline program to get the configuration with less number of conflicting paths. The pseudo-code of this search is shown in Algorithm 1. The algorithm explores for all potential paths (i.e. source and destination pair) in the network which is the priority assignments that maximizes the number of disjoint routes. Note that the algorithm also takes into account the amount of ways in the scheduler.

E. RESCHEDULING TECHNIQUE

As explained in Section III-A the proposed scheduler overlaps notification and transmission phases (see the upper part of Figure 11). However, as notification and data transmission phases have large timing differences, performance may be compromised. To solve this drawback, we propose a

Algorithm 1: Algorithm to Get Best Priority Slot Mapping

```
function build_pslot_map(#ways,#tiles,#slots)

  tiles temp_map
  tiles best_mapping
  tile tx
  tile ty
  way wx
  int temp_disjoint = 0;
 int best_disjoint = 0;
  destination dx

  for every tile in tiles (tx)
   temp_map(tx).prio_slot=get_pslot(slots);
    for every tile in tiles (ty)
      temp_map(ty).prio_slot = get_pslot
      (slots);
    endfor

    for every way in ways (wx)
      temp_map(tx).way(wx).sel_slot =
      (temp_map(tx).prio_slot + wx) % slots;
      for every tile in tiles (dx)
        temp_map(tx).way(wx).dest = dx;
      endfor
    endfor

    temp_disjoint=check_disjoint(temp_map);
    if(temp_disjoint>best_disjoint)
      best_mapping = temp_map;
      best_disjoint = temp_disjoint;
    endif
  endfor

  return best_mapping;

end function
```

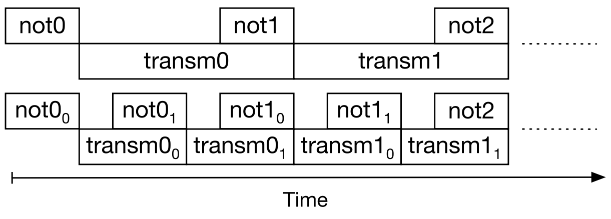


FIGURE 11. Comparison between common scheduler phases (up) and using rescheduling technique (down). Notification phase configures next transmission window. Data transmission phase is now broken down in $transmX_0$ and $transmX_1$.

rescheduling mechanism that finds the best way to seize notification and timing of each phase.

Let us illustrate the proposed mechanism with an example. The timing of the notification phase depends on the number of nodes (#*nodes*), the number of cycles required for notification (#$noti_{cy}$) for each node and the network latency (net_L).

$$Notification\ delay = (\#nodes \times \#noti_{cy}) + net_L \tag{1}$$

Data delay depends on the number of nodes (#*nodes*) and the message length in cycles ($message_L$). Both Notification and Data delay refers to the maximum delay may last these phases.

$$Data\ delay = \#nodes \times \#message_L \tag{2}$$

A 4×4 DCFNoC mesh has a latency of 7 cycles and needs 2 cycles for node notification. Using messages of 5 flits:

$$\begin{aligned} Notification\ delay &= (16 \times 2) + 7 = 39\ cycles \\ Data\ delay &= 16 \times 5 = 80\ cycles \end{aligned} \tag{3}$$

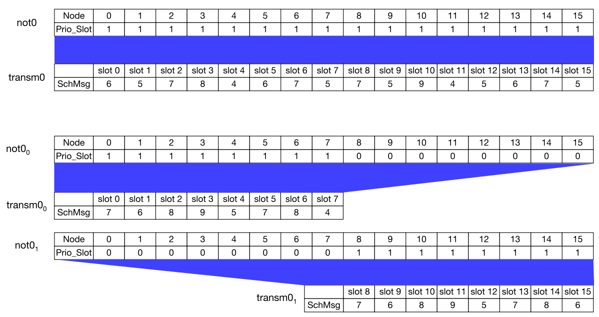
As we can see there is a huge difference between the 39 cycles of the notification phase and the 80 cycles required for transmitting data. Thus, we propose a rescheduling approach in which data phase is split. The first half of window slots in a first round and the other half of the slots in a second one. After this modification notification and data delay are as follows:

$$\begin{aligned} Notification\ delay &= (16 \times 2) + 7 = 39\ cycles \\ Data\ delay &= \frac{16}{2} \times 5 = 40\ cycles \end{aligned} \tag{4}$$

Notification delay remains in 39 cycles since we notify routes from all network nodes but data delay changes from 80 to 40 which results in an almost perfect overlapping.

At the lower part of Figure 11 we show the new procedure for notification and transmission using this rescheduling technique. Data transmission phase is broken down in $transmX_0$ and $transmX_1$. The first notification $notX_0$ deals with the first half of the data window slots and the second one $notX_1$ schedules the remaining slots.

To maximize efficiency of the rescheduling technique it is important to match the notification phase of all nodes $notX_0$ with the same or less slots for data transmission in order to maximize the matching at the end of whole data window (see the lower part of Figure 11).



Node	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Prio_Slot	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	slot 0	slot 1	slot 2	slot 3	slot 4	slot 5	slot 6	slot 7	slot 8	slot 9	slot 10	slot 11	slot 12	slot 13	slot 14	slot 15
SchMsg	6	5	7	8	4	6	7	5	7	5	9	4	5	6	7	5

Node	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Prio_Slot	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

	slot 0	slot 1	slot 2	slot 3	slot 4	slot 5	slot 6	slot 7
SchMsg	7	6	8	9	5	7	8	4

Node	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Prio_Slot	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

	slot 8	slot 9	slot 10	slot 11	slot 12	slot 13	slot 14	slot 15
SchMsg	7	6	8	9	5	7	8	6

FIGURE 12. Analysis of notification nodes targeting scheduling slots in common scheduler phases (up) and using rescheduling technique (down). Note that resulting transmission phase when rescheduling is the addition of $transmX_0$ and $transmX_1$.

Figure 12 shows how the scheduling window is organized in the regular case (top) and when the rescheduling technique is applied (bottom). In both cases each node is assigned the priority in a given slot but routes are notified differently. In the regular scheduling the notification phase occurs once every N slots while with the rescheduling technique the notification phase occurs several times per window (two times in this example). Notifying routes more frequently increases the chances to schedule packets. Note that nodes can use any of the slots. However, packet transmission for non priority nodes only occur if the node with priority is not using the assigned slot.

IV. EVALUATION

In this section we compare the performance achieved by hp-DCFNoC with DCFNoC and a regular wormhole NoC.

A. EXPERIMENTAL SETUP

We implement all designs: hp-DCFNoC, DCFNoC, and a wormhole NoC in verilog RTL. The resulting implementation can be synthetized in FPGAs and ASIC. To obtain performance numbers we simulate the system using the Xilinx Vivado [7] RTL simulator.

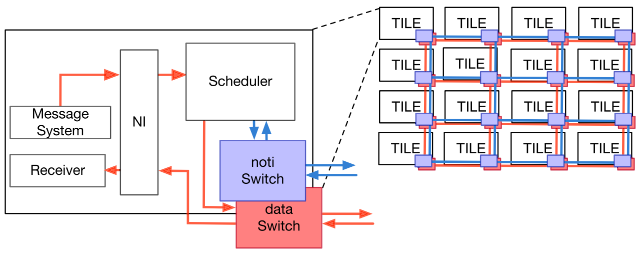


FIGURE 13. **4 × 4 mesh system with schedulers using two DCFNoC networks.**

Figure 13 shows a schematic of the NoC platform for the 4 × 4 mesh system. Note that our approach includes the schedulers and two DCFNoC networks, one for notification and one for data transmission.

For traffic generation we create uniform traffic patterns using a message system generator that is attached to each network interface. In order to create uniform traffic pattern, we use a pseudo random number generator with Linear Feedback Shift Registers (LFSR) to generate a random destination label for every message, thus all nodes have the same probability to be targets. The use of uniform traffic allows us to simulate an unpredictable network load as well as unpredictable used paths. One thousand warm-up messages are generated at the beginning of each test.

We implement NoCs of 16 (4×4 mesh) and 64 nodes (8×8 mesh) using this platform.

B. THEORETICAL WORST-CASE PERFORMANCE

hp-DCFNoC preserves the performance guarantees provided by DCFNoC network. hp-DCFNoC performance is tightly coupled with the scheduling period. While traditional TDM approaches have difficulties to find schedules for large networks hp-DCFNoC is able to find conflict-free scenarios in arbitrarily large NoC sizes. hp-DCFNoC achieves this without degrading the quality of the achieved guarantees by simply ensuring that no more than one node is injecting packets in the same time slot unless it uses a disjoint route. Since only one node is injecting in a particular cycle the resulting NoC guaranteed productivity is equal to $1/N$ flits/cycles/node being N the number of network nodes. Hence, the network injects $1/t_{\text{slot}}$ (one message per TDM slot), at a minimum, resulting in N/P_{TDM} (N messages per TDM period), or in other words N messages per window. Regarding message latency, all communication flows experience a latency that is equal to the time required to traverse the NoC diameter (H) plus the ejection and injection links (2).

hp-DCFNoC provides TDM periods significantly better than other proposals. For instance, for a 25-node mesh NoC the approach in [8] requires a period of 34 while hp-DCFNoC requires only 25 cycles. Additionally, approaches using ILP to find optimal scheduling periods suffer from limited scalability not being able to find schedules for meshes beyond 25 nodes [8]. Other approaches based on heuristics, although being able to find schedules for larger NoCs, result in TDM periods that are significantly worse than the ones achieved by hp-DCFNoC. For instance, in [9] a 64-node mesh requires a period of 481 cycles while hp-DCFNoC requires only 64.

In terms of latency, for very small injection rates and the smallest NoC sizes DCFNoC latency is slightly worse than the one achieved in ILP [8] for the shortest paths but better for the longest ones. For higher NoC sizes and/or higher injection rates DCFNoC achieves always better results [4]. The reason for this is the smaller period of DCFNoC that decreases the average time each message is waiting until it is aligned with the assigned slot. The smaller period also enables DCFNoC achieve small latency values for higher injection rates. Other similar approaches like PhaseNoC [10] achieve the same schedule period but at the cost of higher latency. In summary, hp-DCFNoC baseline performance is in general better than the rest of state-of-the-art TDM approaches.

C. TESTING WORST-CASE PERFORMANCE

Figure 14 shows hp-DCFNoC throughput guarantees for a 4×4 mesh system. Horizontal line represents the throughput guaranteed for each node ($1/16 = 0.0625$ flits/cycle/node). For this experiment we configure node 5 to inject traffic at 10% injection rate while the rest of nodes are injecting at 50%. Note that while the guaranteed throughput is lower than the one injected by node 5 the scheduler is able to meet latency guarantees also when the other messages are injecting at a high rate. For node 5 throughput reaches 0.10 flits per cycle, above the minimum guaranteed throughput, and average message latency does not exceed the horizontal line.

Figure 14b analyzes for an 8 × 8 NoC a different traffic scenario. In this case, six nodes (10, 20, 30, 40, 50 and 60) inject at 50% injection rate while others injects at 5%. For a 64-node configuration hp-DCFNoC guarantees a throughput of 1/64 flits/cycle/node. However, with the dynamic scheduler we have that nodes injecting at 5% are able to sustain this throughput that is above the one actually guaranteed in spite of having several nodes with a very high injection rate. Note however that for nodes injecting at 50% throughput is not preserved.

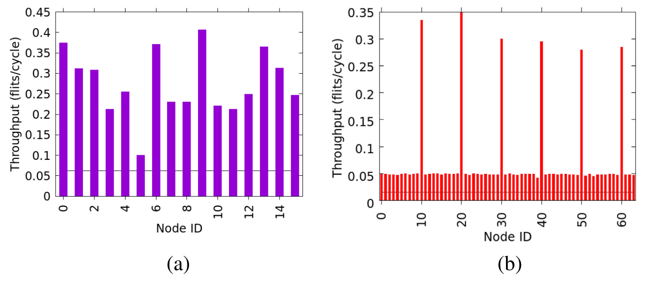


FIGURE 14. Throughput guarantees for a 4 × 4 and 8 × 8 mesh system. In (a) node 5 is injecting traffic at 10% injection rate while others are injecting at 50%. In (b) nodes (10, 20, 30, 40, 50 and 60) inject at 50% injection rate while the rest inject at 5%.

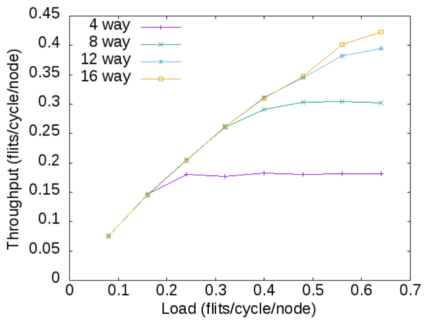


FIGURE 15. Network throughput using baseline scheduler (***BaseSched***) for different ways in a 4 × 4 mesh system.

1) IMPACT OF THE NUMBER OF WAYS

Figure 15 shows the network throughput achieved for a 4 × 4 mesh system with the baseline scheduler (*BaseSched*) without the re-scheduling technique. As shown, throughput achieved with the scheduler is significantly better than the one achieved by DCFNoC (0.00625 flits/cycle/node) even with the a small number of ways (4). Recall that the number of ways represents the maximum amount of messages that are pending to be scheduled in each notification phase. Thus, increasing the number of ways increases the potential throughput of the network although this also implies higher hardware costs. For a 16-node NoC the throughput improves from 0.18 to 0.42 flits/cycles/node when moving from 4 to 16 ways. An interesting observation is that improvements are not so relevant for more than 8 ways.

2) IMPACT OF RESCHEDULING

By implementing the rescheduling technique, the scheduler is able to maximize DCFNoC utilization reaching a significantly higher performance. Figure 16 and 17 show results for 4×4 and 8×8 mesh systems using DCFNoC without dynamic scheduler (*DCFNoC*), the baseline scheduler (*BaseSched*) and the improved scheduler that implements the rescheduling technique (*ImpSched*). For the 4 × 4 mesh scheduler we use 8 ways. Data window contains 16 slots (one per node). Figure 16a shows how network throughput has been improved from 0.062 (1/16) to 0.30 when using the dynamic scheduler reaching 0.43 flits/cycle/node with the *ImpSched*. In other words, the number of messages that can be transmitted per slot goes, on average, from 1 in DCFNoC to 5.8 in the *BaseSched* to 6.9 in *ImpSched*. For the *ImpSched* this results in nearly 110 messages per window. With respect to latency values Figure 16b shows how average message latency is kept low until 0.48 flits/cycle/node.

For a 8 × 8 mesh, the scheduler implements 16 ways at each node and data window contains 64 slots (one per node). Before analyzing the results, it is important to remark that in a 8 × 8 mesh the throughput achieved per node is roughly divided by 2 with respect to a 4 × 4 mesh. Hence, network performance per node is expected to be reduced in a similar manner. Figure 17a shows network throughput for the 64-node mesh. DCFNoC obtains 0.015 (1/64), *BaseSched* gets 0.12 and the improved scheduler reaches 0.23 flits/cycle/node. These throughput numbers show how a network with 64 nodes is able to improve network capacity up to 14 messages per slot which doubles the capacity achieved in the 4 × 4 mesh. This means that the network is able to send nearly 900 messages every TDM window. Figure 17b shows how the network keeps very low message contention until 0.20 flits/cycle/node keeping latency values low.

3) hp-DCFNoC VERSUS WORMHOLE

The goal of hp-DCFNoC is improving the performance achieved with DCFNoC while keeping its valuable QoS properties. In this section we show how hp-DCFNoC is able to achieve that goal but also how its peak performance numbers are very close to the one of wormhole NoCs. Figure 18 compares the network throughput achieved by hp-DCFNoC with the one of a high-performance wormhole NoC. For the wormhole NoC we use a conventional NoC implementation with canonical and pipelined routers, one single virtual channel, round-robin arbitration, and XY routing. As shown in the plots, in both cases, the throughput of hp-DCFNoC is very close to the one that can be achieved with wormhole being close to 0.45 flits/cycle/node for a 16-node mesh and 0.22 for 64-node mesh. Interestingly, while wormhole NoCs have been able to provide performance guarantees [11] the latency bounds provided by these NoCs are much worse than the ones provided by hp-DCFNoC. Figure 19 shows how average message latency is kept low until 0.48 flits/cycle/node for a 16-node mesh and 0.22 for 64-node mesh.

D. ANALYZING THE IMPACT ON APPLICATIONS BEHAVIOUR

In order to evaluate the benefits of using hp-DCFNoC on applications execution time we have designed several synthetic kernels. The synthetic kernels generated produce a variable amount of instructions and the corresponding network messages targeting different destinations based on nature of the message. Instructions labeled as L1hit do not produce any network message. Instructions labeled as L1miss are injected into the network to a random destination and no further instruction is processed until the origin node receives a response. Messages originated from the L1miss instruction

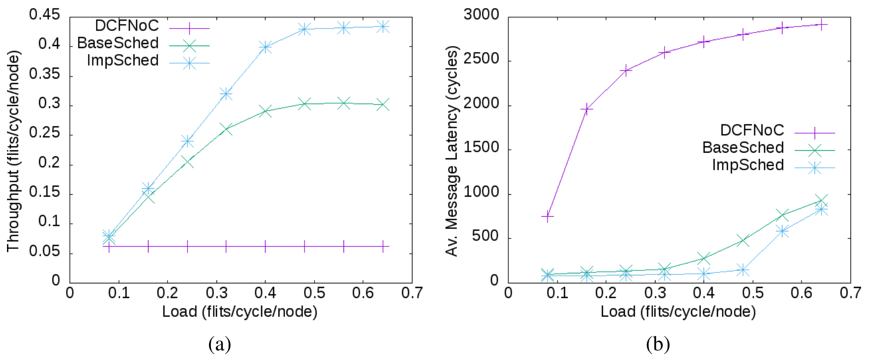


FIGURE 16. Network throughput (a) and message latency (b) in a 4 × 4 mesh.

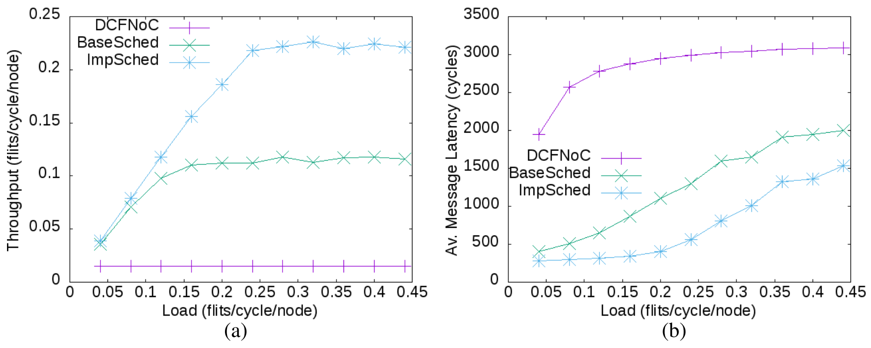


FIGURE 17. Network throughput (a) and message latency (b) in a 8 × 8 mesh.

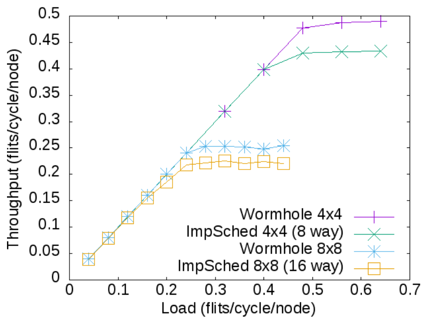


FIGURE 18. Throughput comparison versus standard wormhole in a 4 × 4 and 8 × 8 mesh.

that are labeled as L2hit do not produce additional traffic. However, messages labeled as L2miss produce an additional request to the memory controller (MC) and the node originating the request cannot progress until the response is received. To model requests processing time L2 requests and MC requests are also delayed at the destination by 10 and 40 cycles, respectively. L1hit requests are processed in one cycle. To evaluate the behaviour of this kernel under different levels of contention this traffic model is only executed at one node. The rest of the nodes inject random traffic at a specified load. All modeled scenarios are shown in Table 1.

We executed 5000 instructions in the considered scenarios when using hp-DCFNoC and wormhole NoCs. For these experiments we model a 4 × 4 NoC. The application traffic

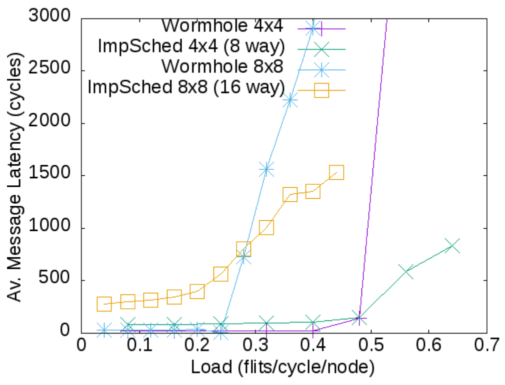


FIGURE 19. Average message latency comparison versus standard wormhole in a 4 × 4 and 8 × 8 mesh.

is executed in node 0 and the MC is located at node 15 (see Fig 2). Figure 20 shows the results of this experiment. As shown in the plot wormhole always provides higher throughput values. This is explained by the fact that average latency values of hp-DCFNoC are generally higher since a notification phase is required before transmitting the data. However, the performance differences are lower in the context of highly saturated scenarios. For these scenarios the network throughput provided by hp-DCFNoC is very close to the one provided by wormhole and zero load latency is less important. In particular, for scenario 1 hp-DCFNoC throughput is 20% lower than the one achieved by wormhole while in

TABLE 1. Different scenarios evaluated. We have modeled three main different scenarios for application traffic with four different levels of network congestion.

Scenario	L1hit	L1miss-L2hit	L1/L2miss-MC	Inj.Rate
1	95%	4%	1%	5%
2	95%	4%	1%	20%
3	95%	4%	1%	40%
4	95%	4%	1%	60%
5	80%	10%	10%	5%
6	80%	10%	10%	20%
7	80%	10%	10%	40%
8	80%	10%	10%	60%
9	70%	20%	10%	5%
10	70%	20%	10%	20%
11	70%	20%	10%	40%
12	70%	20%	10%	60%

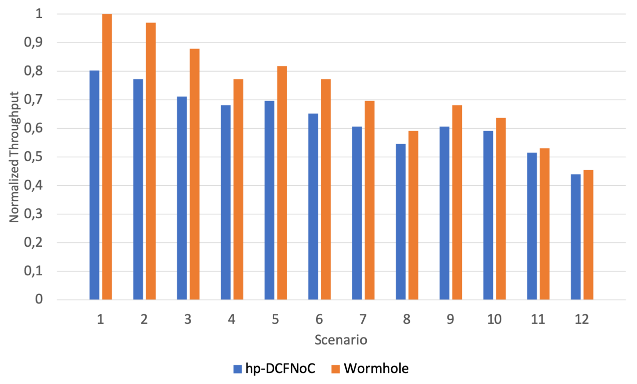


FIGURE 20. Normalized throughput comparison for different scenarios in a 4 × 4 mesh system when using hp-DCFNoC and wormhole NoCs. Application is originated traffic in the farthest node w.r.t to the MC.

scenario 12 the difference is only 12%. It is also important to mention that these differences do actually represent a corner case since processor architectures usually include mechanisms to hide memory latency like write-buffers, out-of-order execution, data and instruction prefetchers and the like. In our experiments for each instruction labeled as L1miss the injector is stalled until a response is received.

The obtained results also indicate hp-DCFNoC is very well suited to applications with high-bandwidth requirements like the ones found in autonomous driving systems [12]. Note that performance guarantees of hp-DCFNoC are identical to the ones provided by DCFNoC while the performance that can be guaranteed in a 4 × 4 mesh wormhole NoC is much lower [4], [11].

E. AREA OVERHEAD AND FREQUENCY

Maximum operating frequency and area utilization are obtained using Cadence RC Compiler and the 45-nm Nangate library [13]. The scheduler implemented is *ImpSched* for a 4 × 4 and 8 × 8 mesh. The scheduler implements 8 and 16 ways at each node, respectively and data window contains 64 slots (one per node). The wormhole NoC implemented is 64-bit width and uses 8 slot input buffers with Stop&Go flow control. For the implementation results we consider a hp-DCFNoC notification (*Noti_sw*) and data (*Data_sw*) routers for a 4×4 and 8×8 mesh. Data network configuration used includes 96-bit width links to implement a mesh network topology. Notification network implements 38-bit width links in this particular case. The wormhole router (*WH_sw*) used for comparison purposes implements (1, 2, and 4) virtual channels.

Figure 21 shows area overheads for every component when targeting high frequency. The 8 × 8 mesh hp-DCFNoC data router uses 30.42% less area than the WH-based one for a 8 × 8 mesh, when using one virtual channel (1vc), with a total area of $28,935\ mm^2$ and $41,273\ mm^2$, respectively. Wormhole router requires implementing input buffers, flow control logic, routing units, output port arbiters and crossbar interconnect. On the other hand, the hp-DCFNoC router implements very simple routing logic in order to compute the output port, a crossbar interconnect as well as output delay registers. Due to small bit-width hp-DCFNoC notification router is the lightest router with a total area of $9,007\ mm^2$ and $17,715\ mm^2$ for a 4×4 and 8×8 mesh implementations, respectively.

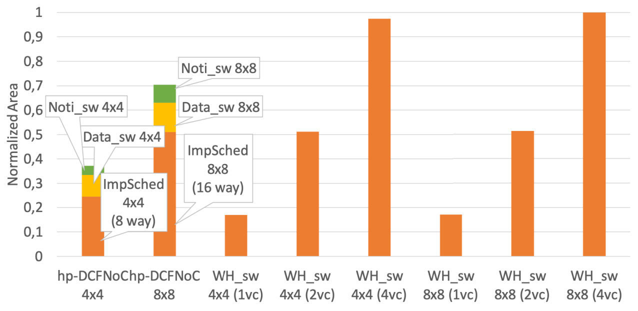


FIGURE 21. Area overhead at each node.

However, the 8 × 8 mesh scheduler uses more area than a wormhole for a 8 × 8 mesh with single virtual channel, although uses roughly the same area compared to WH-based using 2 virtual channels with a total area of $123,551\ mm^2$ and $124,143\ mm^2$, respectively. It is important to remark that in general NoCs found in commercial processor require at least 2 vcs to avoid request and response messages deadlock and even more virtual channels when using cache coherence protocols to prevent protocol-induced deadlocks.

Figure 21 also shows total area of full hp-DCFNoC implementation including the *ImpSched*, notification and data routers. The 8 × 8 mesh hp-DCFNoC implementation uses 27% more area than the *WH_sw* for a 8 × 8 mesh with 2 vcs, however, is 42% lighter than a wormhole router when using 4 virtual channels.

Figure 22 illustrates how area overhead of hp-DCFNoC is affected as the number of ways increases. As we can see the area overhead grows linearly as the *ImpSched* module implements more ways.

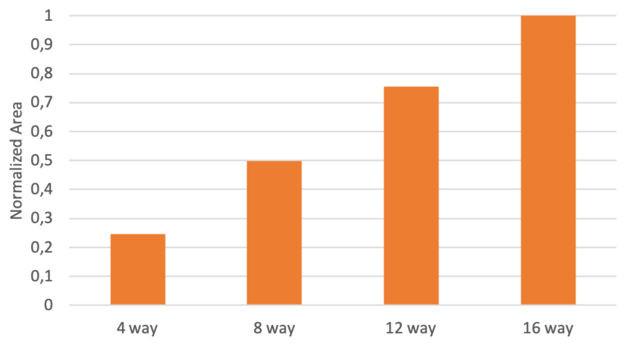


FIGURE 22. Area overhead for *ImpSched* implementations with different number of ways in a 4 × 4 mesh system.

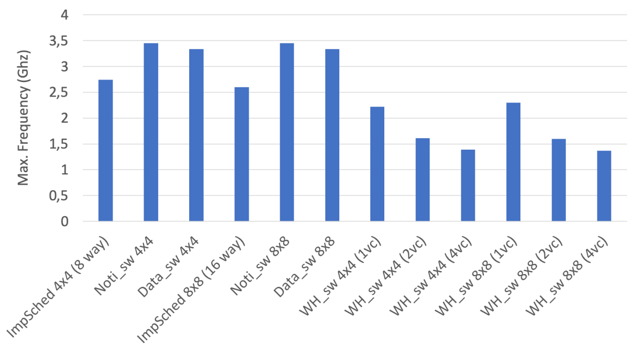


FIGURE 23. Maximum attainable clock frequency for all modules of hp-DCFNoC. Wormhole is also shown for comparison purposes.

We have also analyzed the maximum attainable clock frequency of the difference routers. Figure 23 show that the simpler hp-DCFNoC routers design gets a significant boost in clock frequency by improving wormhole router's one by 55% and 50% for notification and data router, respectively. The critical path of the wormhole router limits clock frequency to 2.22 GHz. However, the hp-DCFNoC routers exhibits a critical path of 290 ps and 300 ps leading to a maximum clock frequency of 3.45 GHz and 3.33 GHz for notification and data router, respectively. Although wormhole router is more complex, it is pipelined in 4-stages which allows achieving high frequencies.

For hp-DCFNoC, in addition to the fast hp-DCFNoC routers we have a relatively complex scheduler that includes arbiters and route checkers. For the scheduling process we use only one cycle simplifying route notifications and control logic. Even with such critical path, hp-DCFNoC clock frequency reaches up to 2.74 GHz and 2.60 GHz for a 4 × 4 and 8 × 8 mesh implementations, respectively.

V. RELATED WORK

Our proposal relies on the utilization of TDM to provide real-time guarantees. In particular, we build on top of DCFNoC [5] a distributed TDM implementation to solve the difficulties that previous TDM approaches have to find optimal schedules for medium/large NoC sizes. Traditionally, TDM schedules are statically [8], [9], [14]–[19], or dynamically computed [20]. In [8], [18] perform offline scheduling (with a perfect a-priori knowledge of the applications to be running on the system). On the contrary, DCFNoC [5] removes the need for using a computationally expensive off-line process to find optimal scheduling periods.

Other works [21], [11] analyse the real-time properties of safety-critical real-time systems NoCs (e.g wormhole NoCs). These works show that performance guarantees can be achieved with certain design configuration, however when considering time composability aspects, the achieved guarantees are in general very poor [11]. Many NoC proposals rely on the utilization of virtual channels to ensure non-interfering operations across communication flows [10], [22], [23]. These proposals use virtual channels to build domains that isolate traffic flows and result in an improvement in the performance guarantees with respect to conventional wormhole NoC designs. Other approaches use virtual channel prioritization with flit-level preemption [24], [25]. These approaches achieve tight latency bounds for the highest priority flows and fit well with response time analysis methodologies. In general, approaches based on using VCs suffer from limited scalability since achieving the desired levels of isolation or performance guarantees for a high number of traffic flows requires implementing a non negligible amount of virtual channels. For on-chip networks the amount of virtual channels that can be included is very limited and the available ones are usually required to avoid protocol deadlocks in coherence protocols.

hp-DCFNoC significantly improves peak performance capabilities of DCFNoC using a dynamic scheduler. Dynamic TDM slot management is also implemented in [26] and [27]. However, unlike these approaches our proposal relies on a distributed slot management mechanism which makes our solution more scalable. AEthereal NoC also proposes a distributed and dynamic time-slot reservation procedure [14], [28]. However, for this NoC slot reservation is done at source node and the procedure fails in case one of the requested slots are not available.

A dynamic and centralized scheduler was proposed in [20]. As in our approach the work in [20] employs two independent and parallel networks. It uses credit-based end-to-end flow control to avoid data network overflow via best-effort network. The main differences with our work are the NoC architecture and the slot reservation procedure. In [20] credits travel with no deterministic latency and therefore, performance cannot be completely guaranteed. In hp-DCFNoC performance is guaranteed and a simpler negotiation mechanism leads to more efficient NoC implementation.

VI. CONCLUSION

Future safety-critical real-time systems will need processor designs able to provide performance guarantees without renouncing to peak throughput numbers. In this paper we propose hp-DCFNoC, a new NoC design that satisfies this requirement by providing throughput numbers that are very close to the ones that can be achieved with standard best-effort wormhole NoCs. At the same performance level hp-DCFNoC is able to guarantee performance to applications with much less resources (area and power) than an standard wormhole implementation. To achieve the aforementioned demanding features hp-DCFNoC relies on a distributed scheduler built on top of DCFNoC that maximizes the number of packets that can be scheduled.

REFERENCES

[1] J. Flich and D. Bertozzi, *Designing Network On-Chip Architectures in the Nanoscale Era*. Boca Raton, FL, USA: CRC Press, 2010.

[2] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 82:1–82:37, Nov. 2017. [Online]. Available: http://doi.acm.org/10.1145/3131347

[3] S. Hesham, J. Rettkowski, D. Goehringer, and M. A. A. El Ghany, "Survey on real-time networks-on-chip," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 5, pp. 1500–1517, May 2017, doi: 10.1109/TPDS.2016.2623619.

[4] T. Picornell, J. Flich, C. Hernandez, and J. Duato, "Enforcing predictability of many-cores with DCFNoC," *IEEE Trans. Comput.*, early access, Apr. 15, 2020, doi: 10.1109/TC.2020.2987797.

[5] T. Picornell, J. Flich, C. Hernández, and J. Duato, "DCFNoC: A delayed conflict-free time division multiplexing network on chip," in *Proc. 56th Annu. Design Autom. Conf.* New York, NY, USA: ACM, Jun. 2019, pp. 95:1–95:6, doi: 10.1145/3316781.3317794.

[6] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*, 1st ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 1997.

[7] (2016). *Vivado Design Suite 2016.2*. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html

[8] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki, "A statically scheduled time-division-multiplexed network-on-chip for real-time systems," in *Proc. IEEE/ACM 6th Int. Symp. Netw.-Chip*, May 2012, pp. 152–160.

[9] F. Brandner and M. Schoeberl, "Static routing in symmetric real-time network-on-chips," in *Proc. 20th Int. Conf. Real-Time Netw. Syst. (RTNS)*. New York, NY, USA: ACM, 2012, pp. 61–70, doi: 10.1145/2392987.2392995.

[10] A. Psarras, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos, "PhaseNoC: TDM scheduling at the virtual-channel level for efficient network traffic isolation," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 1090–1095.

[11] M. Panic, C. Hernandez, E. Quinones, J. Abella, and F. J. Cazorla, "Modeling high-performance wormhole NoCs for critical real-time embedded systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Vienna, Austria, Apr. 2016, pp. 267–278, doi: 10.1109/RTAS.2016.7461342.

[12] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 751–766, Nov. 2018, doi: 10.1145/3296957.3173191.

[13] (2010). *The Nangate Open Cell Library, 45 nm Freepdk*. [Online]. Available: https://projects.si2.org/openeda.si2.org/projects/nangatelib/

[14] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip:Concepts, architectures, and implementations," *IEEE Design Test Comput.*, vol. 22, no. 5, pp. 414–421, May 2005.

[15] A. Hansson, M. Subburaman, and K. Goossens, "Aelite: A flit-synchronous network on chip with composable and predictable services," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Apr. 2009, pp. 250–255.

[16] R. Andrei Stefan, A. Molnos, and K. Goossens, "DAElite: A TDM NoC supporting QoS, multicast, and fast connection set-up," *IEEE Trans. Comput.*, vol. 63, no. 3, pp. 583–594, Mar. 2014.

[17] Z. Lu and A. Jantsch, "TDM virtual-circuit configuration for network-on-chip," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 8, pp. 1021–1034, Aug. 2008.

[18] E. Kasapaki, M. Schoeberl, R. B. Sorensen, C. Müller, K. Goossens, and J. Sparso, "Argo: A real-time network-on-chip architecture with an efficient GALS implementation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 2, pp. 479–492, Feb. 2016.

[19] R. B. Sorensen, J. Sparso, M. R. Pedersen, and J. Hojgaard, "A metaheuristic scheduler for time division multiplexed networks-on-chip," in *Proc. IEEE 17th Int. Symp. Object/Component/Service-Oriented Real-Time Distrib. Comput.*, Jun. 2014, pp. 309–316.

[20] N. Concer, A. Vesco, R. Scopigno, and L. P. Carloni, "A dynamic and distributed TDM slot-scheduling protocol for QoS-oriented networks-on-chip," in *Proc. IEEE 29th Int. Conf. Comput. Design (ICCD)*, Oct. 2011, pp. 31–38.

[21] D. Rahmati, S. Murali, L. Benini, F. Angiolini, G. De Micheli, and H. Sarbazi-Azad, "Computing accurate performance bounds for best effort networks-on-chip," *IEEE Trans. Comput.*, vol. 62, no. 3, pp. 452–467, Mar. 2013. [Online]. Available: http://infoscience.epfl.ch/record/170499

[22] H. M. G. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood, "SurfNoC: A low latency and provably non-interfering approach to secure networks-on-chip," in *Proc. 40th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2013, pp. 583–594, doi: 10.1145/2485922.2485972.

[23] A. Psarras, J. Lee, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos, "PhaseNoC: Versatile network traffic isolation through TDM-scheduled virtual channels," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 5, pp. 844–857, May 2016.

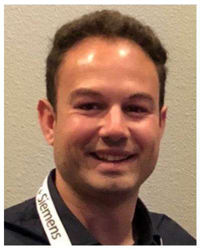
[24] B. Nikolić, S. Tobuschat, L. S. Indrusiak, R. Ernst, and A. Burns, "Real-time analysis of priority-preemptive NoCs with arbitrary buffer sizes and router delays," *Real-Time Syst.*, vol. 55, no. 1, pp. 63–105, Jan. 2019, doi: 10.1007/s11241-018-9312-0.

[25] L. S. Indrusiak, A. Burns, and B. Nikolic, "Buffer-aware bounds to multi-point progressive blocking in priority-preemptive NoCs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Dresden, Germany, Mar. 2018, pp. 219–224, doi: 10.23919/DATE.2018.8342006.

[26] O. Moreira, J. J.-D. Mol, and M. Bekooij, "Online resource management in a multiprocessor with a network-on-chip," in *Proc. ACM Symp. Appl. Comput. (SAC)*. New York, NY, USA: ACM, 2007, pp. 1557–1564, doi: 10.1145/1244002.1244335.

[27] T. Marescaux, B. Bricke, P. Debacker, V. Nollet, and H. Corporaal, "Dynamic time-slot allocation for QoS enabled networks on chip," in *Proc. 3rd Workshop Embedded Syst. Real-Time Multimedia*, Sep. 2005, pp. 47–52.

[28] B. Gebremichael, F. Vaandrager, M. Zhang, K. Goossens, E. Rijpkema, and A. Rădulescu, "Deadlock prevention in the æthereal protocol," in *Correct Hardware Design and Verification Methods*, D. Borrione and W. Paul, Eds. Berlin, Germany: Springer, 2005, pp. 345–348.



TOMÁS PICORNELL received the M.S. degree in computer and network engineering from the Technical University of Valencia [Universitat Politècnica de València (UPV)], Spain, in 2016. He is currently pursuing the Ph.D. degree with the UPV. His research interests include network-on-chip architectures with support for time predictability and performance isolation as well as system level solutions to minimize the effects of variability on NoC performance.

JOSE FLICH (Senior Member, IEEE) received the Ph.D. degree in computer engineering in 2001. He is a Full Professor with the Universitat Politècnica de València (UPV), where he leads the research activities related to NoCs. He has published over 150 conference papers and journal articles, and has served on different conference program committees, such as ISCA, PACT, HPCA, NOCS, ICPP, IPDPS, HiPC, CAC, CASS, ICPADS, and ISCC, as the Program Chair (INA-OCMC and CAC) and Track Co-Chair (EUROPAR). He has collaborated with different institutions, such as Ferrara, Naples, Catania, Jonkoping, and the USC, and companies such as AMD, Intel, and Sun. He has co-invented different routing strategies, and reconfiguration and congestion control mechanisms, some of them with high recognition (RECN and LBDR for on-chip networks). His current research interests include routing, coherency protocols, and congestion management within NoCs. He is a member of the Hipeac-2 NoE. He is the Coeditor of the book *Designing Network-on-Chip Architectures in the Nanoscale Era*, has Coordinated the FP7 NaNoC Project, and leads the H2020 MANGO Project.

DUATO JOSE (Senior Member, IEEE) is a Professor with the Department of Computer Engineering (DISCA), Technical University of Valencia (Universitat Politècnica de València). He has published over 500 refereed articles. According to Google Scholar, his publications received more than 15 000 citations. He proposed the theory of deadlock-free adaptive routing that has been used in the design of the routing algorithms for the Cray T3E supercomputer, the on-chip router of the Alpha 21364 microprocessor, and the IBM BlueGene/L supercomputer. He also developed RECN, a scalable congestion management technique, and a very efficient routing algorithm for fat trees that has been incorporated into Sun Microsystem's 3456-port InfiniBand Magnum switch. He led the Advanced Technology Group, HyperTransport Consortium, and was the main contributor of the High Node Count HyperTransport Specification 1.0. He also led the development of rCUDA, which enables remote virtualized access to GP-GPU accelerators using a CUDA interface. He is the first author of the book *Interconnection Networks: An Engineering Approach*. His current research interests include interconnection networks, multicore and multiprocessor architectures, and accelerators for deep learning.

Prof. Jose was awarded the National Research Prize in 2009 and the ''Rey Jaime I'' Prize in 2006. He is a member of the Spanish Royal Academy of Sciences. He also served as a member of the editorial boards of the IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, and IEEE Computer Architecture Letters.

CARLES HERNÁNDEZ was a Senior Researcher at the Barcelona Supercomputing Center, CAOS Group, from 2012 to 2018. In 2012, he worked as an Intern at the IP Verification Group, Intel Mobile Communications Munich. He is currently co-advising five Ph.D. degree students. He is a Senior Researcher with the Universitat Politècnica de València. In 2015, he was granted a Young Researcher Grant by the Spanish ministry to conduct research on high-performance and reliable processor design. He is the Project Coordinator of the H2020 SELENE Project on high-performance computing for safety-related applications and participates in RECIPE, DEEPHEALTH H2020, and FRACTAL ECSEL Projects. His research interests include on-chip interconnects, processor design, and real-time aware hardware design and reliability.