ASCENT: Communication Scheduling for SDF on Bufferless Software-defined NoC

Vanchinathan Venkataramani, Bruno Bodin, Aditi Kulkarni Mohite, Tulika Mitra and Li-Shiuan Peh

***Abstract*—Bufferless software-defined Network-on-Chip (NoC) is a promising alternative to conventional dynamic routing as it offers predictable data movement with real-time guarantees. Existing Time-Division Multiplexing (TDM)-based mechanisms for predictability assume the worst-case communication pattern (e.g., all-to-all) and compute a fixed schedule wherein the cores can only communicate during the allocated time-slots. These approaches lead to low application throughput as they cannot adapt to application characteristics. In this paper, we present an application-specific, non-TDM based communication scheduling mechanism for bufferless software-defined NoCs. We choose Synchronous Dataflow (SDF) model-of-computation to represent the input streaming applications. We propose *ASCENT*, a novel offline approach that takes the SDF-specified streaming application and the NoC architecture as input, exploits the task interactions and the timing information in the SDF, and generates the task-to-core mapping and communication schedule that is represented compactly in hardware. *ASCENT* achieves 5.8x better performance on average than existing TDM-based NoCs and manages to achieve the performance of an ideal dynamically routed NoC, yet ensuring predictability.**

***Index Terms*—Time-predictability, Many-core architecture, Synchronous dataflow.**

I. Introduction

Conventional Networks-on-Chip (NoCs) are usually packet-switched, with routers that dynamically multiplex on-chip traffic at run-time to deliver high bandwidth. The dynamic router consists of multiple pipeline stages, multiple buffer queues per port or virtual channels (VC), making it heavyweight. The run-time routing and flow control also imply that the NoC is unpredictable, making it challenging to provide real-time guarantees. Yet, the last decade has seen an increase in the demand for architectures that provide real-time guarantees, due to domains like robotics, autonomous vehicles, etc. [1]. To ensure real-time guarantees, just like how scratchpads replace caches [2], [3], [4], bufferless software-defined NoCs can take the place of dynamically-routed NoCs to meet real-time application constraints.

In bufferless software-defined NoCs [5], a static scheduler completely orchestrates the NoC traffic. First, the scheduler needs to ensure that the packets reach the correct destinations. Second, as there is no logic to perform routing or flow control in hardware and no buffer to store the packets, the scheduler must guarantee contention-free routing and flow control. Finally, the scheduler also needs to make sure that the application throughput requirements are maximized or at least satisfied.

Several architectures containing general-purpose processors connected using a bufferless software-defined NoC [6] with zero or more additional dynamically routed NoCs have been proposed to support diverse applications and meet the timing guarantees, e.g., T-Crest [7], Æthereal [8], SPECTRUM [5], etc. Most of these works utilize Time Division Multiplexing (TDM) in which the cores can only communicate periodically in the allocated time-slots. The scheduler takes the worst-case communication pattern as input (e.g., all-to-all), determines the packet routes, and stores them in slot tables within the router [9], [10]. Computing the TDM slot table in bufferless NoCs is challenging as consecutive time slots need to be allocated in adjacent routers along the path. Additionally, a time-slot is allocated to only one flow conservatively even when contention may not happen at run-time. The NoC topology also restricts the number of concurrent flows that can be supported. Theoretically, for n cores, with one ejection port to the NoC, only $n-1$ time-slots should be required for communicating with the remaining $n-1$ cores (one time-slot per destination core), assuming that there is no network contention. However, in [10], the optimum minimum slot table could be tractably calculated for only up to 5x5 mesh size with 34 slot table entries (instead of the theoretical lower bound of 24). For larger mesh size, the slot table size has to be determined by greedy heuristics [9]. Moreover, as the cores can only communicate in the time-slots allocated to them, TDM-based approaches lead to poor network and application throughput especially when the total number of time-slots is large.

In this paper, we move away from the application-agnostic worst-case behavior based TDM approaches to application-aware communication scheduling for bufferless NoCs starting with the commonly used Synchronous Dataflow (SDF) [11] model-of-computation for representing streaming applications. The execution time of the tasks and inter-task communication is represented accurately in the SDFs. Thus, static analysis of the SDF can generate a deterministic schedule of the tasks and the packets, making them a preferred model especially in real-time systems [12]. The scheduling information gives us a notion of time corresponding to each data packet that is sent in the application. In this paper, we argue that the determinism and timing information of data flows available in the SDFs can be leveraged to perform packet transfers at the precise time compared to core-based TDM mechanisms that rely on time-slots. We resolve conflicts statically because the

V.Venkataramani is with Advanced Micro Devices, India. E-mail: (vanchinathan.venkataramani@amd.com). A.K. Mohite, B. Bodin, T. Mitra, and L.S. Peh are with the Department of Computer Science, School of Computing, National University of Singapore (NUS), SG. Email: ((aditi, bruno, tulika, peh)@comp.nus.edu.sg). V.Venkataramani was with National University of Singapore when this research was conducted. (Corresponding author: Tulika Mitra). This research is supported by the National Research Foundation, Singapore under its Competitive Research Programme Award NRF-CRP23-2019-0003 and NRF-RSSS2016-005.

SDFs expose the periodic run-time behavior of the application. This improves the utilization of the NoC links by supporting cycle-by-cycle scheduling of the NoC instead of relying on time-slots. We are also able to store the schedule compactly within routers by performing periodic scheduling of the SDF application.

We propose *ASCENT*, a task-to-core mapping and communication scheduling mechanism that is entirely done offline. *ASCENT* determines the packet flows through offline task scheduling and ensures that contention never happens on the NoC links (router is bufferless) and packets reach appropriate destinations. We achieve this using static analysis of the SDFs that contain precise timing information of task execution and data flows. This communication schedule will then be loaded into the software-defined NoC routers compactly before applications execute. These routers are called software-defined as they can be programmed at run-time. An underlying control logic uses this schedule and re-configures switch connections to move packets appropriately over the NoC.

Our concrete contributions are the following:

- We propose *ASCENT*, an offline approach which takes in an SDF-specified application and a bufferless software-defined NoC topology as input and performs application-specific task-to-core mapping and communication scheduling that exploits the precise timing information.
- The generated schedule can be succinctly encoded and stored within routers with limited memory. At run-time, the routers are reconfigured to realize the switch connections based on this schedule with the aid of Finite State Machines. We describe the changes required in the router micro-architecture to efficiently store and perform this schedule.
- *ASCENT* achieves better performance than TDM-based mechanisms, approaching the throughput of ideal dynamically-routed NoCs, yet ensuring predictability.

II. Background on Real-Time NoCs

NoCs such as the asynchronous MANGO NoC [13], SoCBus [14] and the synchronous ÆThereal [8], dAElite [15], S4NoC [10] that provide real-time guarantees have been proposed in the past [6]. Most of these approaches perform communication scheduling on bufferless software-defined NoCs using TDM, where each core can only send data in its allocated time slots.

Apart from handling general-purpose all-to-all traffic, prior works have also explored tailoring TDM schedules to the specific application's communication demands. For instance, [16] uses the packet flow information between tasks to allocate multiple slots to match application bandwidth requirements using constraint programming. However, the constraint programming-based scheduling, due to its complexity, may not generate a feasible TDM schedule or may produce an extremely large slot table in order to match the exact bandwidth requirements. A recent real-time TDM NoC proposal DCFNoC [17] routes each flow such that the latency is always that of the worst-case route, thereby simplifying the TDM scheduling and allowing it to approach the ideal TDM throughput.

Inherently, any task in a TDM NoC needs to wait for the slot allocated to the core on which this task is executing, to inject a packet. Hence, TDM-based scheduling leads to under-utilization of resources, higher latency, and lower throughput than dynamic packet-switched NoCs, which can deftly interleave packets on links on a cycle-by-cycle basis.

Non-TDM statically scheduled NoCs. In Tilera TILEPro64 many-core (successor of MIT RAW [18]), dynamically routed networks are used as a backup to support the software-defined NoC when the system faces unknown traffic patterns. The work in [5] eliminates the need for TDM slots, but it targets a specific class of application, baseband processing, and cannot generalize to diverse flows.

ASCENT eliminates time slots by configuring routers to match the periodicity of the application's communication patterns and the cycle-by-cycle scheduling allows it to achieve better throughput than existing TDM-based methods, approaching that of the ideal dynamic NoCs.

III. Synchronous Data Flow Graphs

Streaming applications with throughput requirements typically contain a set of cooperating tasks that execute periodically. The Synchronous Data Flow (SDF) [11] model-of-computation comprehensively represents these applications with concurrent tasks and dependencies among the tasks. Static analysis of the SDF can generate a deterministic schedule of the tasks, making SDF a preferred model especially in real-time systems. Several DSP and multimedia applications have been represented as SDF [12].

A. Models and notations

An SDF defines an application as a set of tasks $\mathcal{T}$ communicating through a set of buffers $\mathcal{B}$. Whenever a task $t \in \mathcal{T}$ is executed, it consumes a fixed number of data packets (known as tokens in SDF parlance) in its input buffers, has an execution time (E_t), and produces a fixed number of tokens in output buffers. Each buffer $b = (t_i, t_j) \in \mathcal{B}$ has one input task $t_i \in \mathcal{T}$ and one output task $t_j \in \mathcal{T}$. The production rate of t_i in $b \in \mathcal{B}$ is in_b: the number of tokens produced in b when t_i is executed. The consumption rate of t_j in b is out_b: the tokens consumed when t_j is executed. For example, in Figure 1, when task A fires it consumes 8 tokens, executes for $E_A = 1$ time units and produces 2 tokens that in turn will be consumed by task B. The buffer b_{AB} between the two tasks A and B has $in_{b_{AB}}$ and $out_{b_{AB}}$ as 2 and 2, respectively.

B. SDF Scheduling

An important aspect of SDFs is that they can be scheduled at compile-time taking system specifications and application objectives into account. In the context of SDF, meeting the application throughput requirements is a commonly targeted objective. Scheduling algorithms can generate the maximal throughput schedule of an application by choosing the start times of the tasks appropriately.

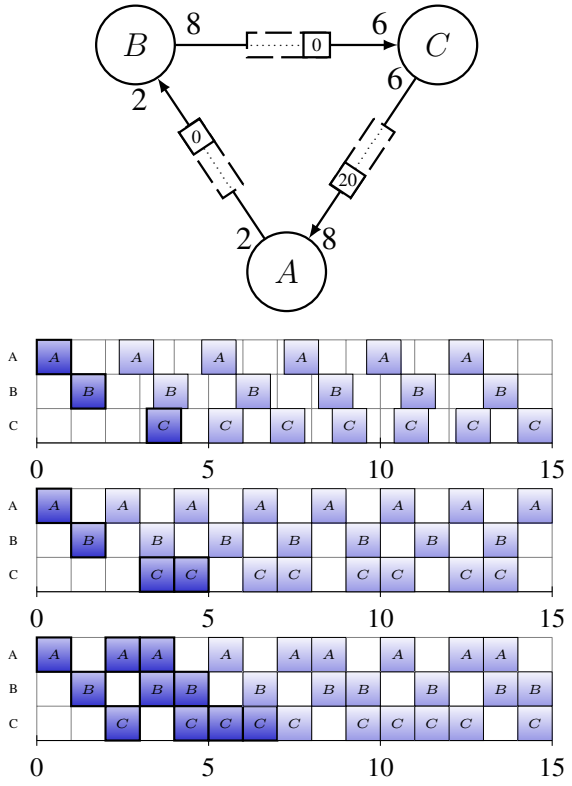


Fig. 1: SDF containing three tasks and different associated K-Periodic schedules. The periodicity vector used for the top, middle and bottom K-Periodic schedules are $K = [1, 1, 1]$, $[1, 1, 2]$, and $[3, 3, 4]$ respectively. For example, in the middle figure, we can see the first execution of the task A starts at 0 with a period of 2 units of time, while the task C has two starting times (at 3 and 4) with a period of 3.

A common approach to SDF scheduling is As Soon As Possible (ASAP) scheduling of the tasks wherein a task is scheduled as soon as its input tokens are available. This scheduling representation is composed of a transitory phase followed by a steady-state phase. The transitory phase is known to be of finite size, and the steady-state phase is modeled by an execution pattern that is infinitely repeated. ASAP schedule is optimal in the absence of resource constraints [19]. However, the length of an ASAP schedule can be exponential in the number of SDF tasks in the application.

An alternative is to build a K-Periodic schedule [20] with a periodicity vector K as well as a set of periods $\mathcal{P}$. For any task t, the value K_t is its periodicity factor and P_t is its period. The schedule of any task t ($t \in \mathcal{T}$) is modeled by a sequence of K_t start times and a period $P_t \in \mathcal{P}$.

The pattern representing the start times of initial instances: S_1^t, S_2^t, $\cdots$, $S_{K_t}^t$ is repeated every P_t time units. Figure 1 shows three different K-Periodic schedules for the SDF presented on the top. For the schedule in the middle subfigure, we observe that task C with $k_C = 2$ has two continuous execution instances ($S_1^C = 3$, $S_2^C = 4$) repeating every three time units ($P_C = 3$).

Importantly, it has been shown that there always exists a periodicity vector K for which the schedule reaches maximal throughput [21]. We consider the scheduling methodology proposed in [21] as it obtains optimal throughput while keeping the values of K as low as possible, leading to a compact schedule which is critical in software-defined NoCs as the schedule has to be stored in hardware routers.

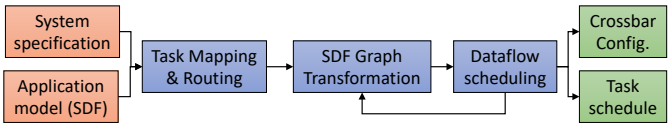


Fig. 2: *ASCENT* communication scheduling Overview

C. Related works on SDF Scheduling.

A number of frameworks have been developed to compile dataflow models and perform task mapping on many-cores and heterogeneous architectures [22], [23]. However, most of these tools optimize task scheduling while communications are offloaded to the targeted hardware (i.e., dynamic NoC, shared memory). Existing works on compile-time communication scheduling using Constraint Programming [24] or ILP (Integer Linear Programming) [25], [26] not only have long compute run-time but also create schedules that are too large to fit in the router memory. While [27] uses periodic scheduling to reduce the schedule size, the authors only consider dynamic NoCs and do not take communication patterns into account. Although communications are factored in [28], [29] using Time Division Multiple Access (TDMA), they are limited by the maximal throughput achievable with a fixed and small number of time slots.

Applying existing SDF scheduling algorithms to communication scheduling requires a complete rethink of how the resources are modeled. The cycle-by-cycle interleaving supported by non-TDM software-defined NoCs leads to an explosion in the search space and can result in a bulky schedule that cannot be realized in hardware. To the best of our knowledge, this is the first work that models NoC resources using appropriate SDF graph transformations and exploits periodic scheduling of tasks and communications to configure a software-defined NoC. This approach leads to better throughput, while still ensuring a statically generated schedule that easily fits in hardware.

IV. *ASCENT*: Communication Scheduling Mechanism

We present *ASCENT*, a scheduling strategy that takes in as input an application specified as SDF, a specification of the bufferless NoC architecture, and generates a task mapping and a communication schedule (Figure 2). We first rely on a generic task-to-core mapping with associated communication routing (Section V-A). Next, a series of graph transformation operations are applied on the SDF to model resources constraints and hardware features (Sections V-B and V-C). The novelty of our approach is that the application schedule is used and refined all along to direct these transformations, making sure that the final schedule remains K-Periodic and contention-free (Section IV-B). Importantly, the periodicity in our approach enables a compact representation of schedules

that can easily be stored in hardware. This schedule is finally used to configure the NoC to support communications between cores.

Algorithm 1 ASCENT Algorithm

Input: SDFG G and NoC Architecture A
Output: Configuration of NoC Routers C

$R \leftarrow MapAndRoute(G, A)$ // (Section V-A)
$G' \leftarrow ModelNetwork(G, R)$ // (Section V-B)
$X \leftarrow FindConflict(G')$ // (Section V-C)
while $\exists\ x \in X$ **do**
  $G' \leftarrow MergeTasks(G', x)$ // (Section V-C)
  $X \leftarrow FindConflict(G')$ // (Section V-C)
end while
$S \leftarrow KPeriodicSched(G')$ // (Section IV-B)
$C \leftarrow RouterConfigs(S)$ // (Section V-D)
return C

A. ASCENT Router Micro-architecture

As previously discussed, a key goal of the *ASCENT*'s methodology is to produce periodic and very compact schedules that can easily fit in NoC routers' memory.

In Figure 3, we propose a micro-architecture for such a router. Since the software scheduler takes care of routing and ensures contention freedom (no two incoming ports will simultaneously use the same outgoing port), the router hardware does not need any routing, flow control logic, or buffering. However, it needs efficient hardware mechanisms to store and realize the software schedule every cycle. If we use slot tables like in TDM, the table size can be large, as ASCENT supports cycle-by-cycle configuration by the software scheduler. Instead, we support software-defined scheduling through finite state machines that set the crossbar muxes.

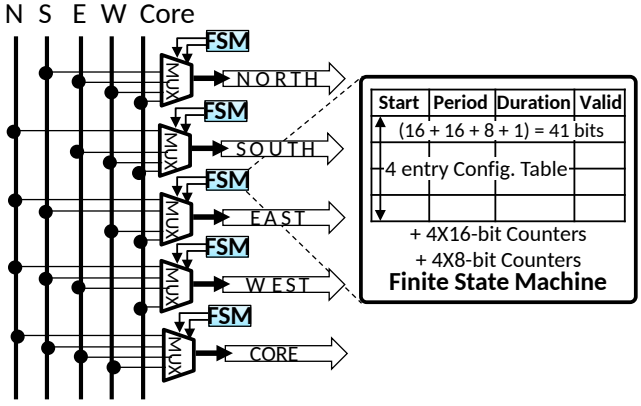


Fig. 3: ASCENT router micro-architecture

Each router comprises of five 4:1 multiplexers. A finite state machine (FSM) controls the select bits of the muxes. Each FSM is defined in hardware as four registers of size 41 bits. Data is written to the FSM only once at the beginning of the execution of the schedule. At any given time, only one of the four incoming links is connected to the output. For example, for an outgoing link in the North direction, we can only have packets from one of the four incoming links (no U-turns): South, East, West and Core. There are thus four corresponding entries per port and a total of 20 entries across 5 ports (N, S, E, W, Core) of a mesh buffer.

The software scheduler sets up 4 fields for each entry: Start, Period, Duration, Valid. For example, if an entry corresponding to the North $\rightarrow$ South connection is configured with the value Start=20, Period=10, Duration=5 and the East $\rightarrow$ South connection is configured with value Start=35, Period=20, Duration=2, then the North $\rightarrow$ South connection is established in cycles (20-24), (30-34), (40-44) and the East $\rightarrow$ South connection is established in cycles (35-36), (55-56), (75-76) and so on.

A 16-bit Period counter is associated with each of the 20 entries, initialized to 0. In each cycle, the counter is increased, and the specific input-output port is connected (mux select signal is set) when the counter reaches the Period value. The counter is then reset to 0 once the Period value is reached. Another 8-bit counter is set to Duration and decremented each cycle, disconnecting the mux when it hits 0.

B. Scheduling

To limit the size of the schedule so that it can fit in the router's memory, our strategy relies on K-Periodic scheduling. K-Periodic scheduling assigns to each task a set of initial starting times and a period that will repeat these executions. In most SDF graph use-cases, only one starting time is needed per task. This makes this class of scheduling extremely compact, while still reaching the maximal throughput performance of an application [21].

The main principle of the K-Periodic scheduling technique from [21] is to solve a Linear Program made of multiple constraints corresponding to execution dependencies of tasks connected by any channel in the graph. More specifically, a valid schedule is found when for every channel b in the SDF graph G, Equation 1 is satisfied. We recall that S_i^u is the starting time of the i^{th} execution of the task u, E_u the execution time of u, P_S the global period of the graph, and $delay_b(i,j)$ is a constant value that models the precedence constraints between the i^{th} execution of the task u and the j^{th} execution of the task v where channel $b = (u, v)$.

$$S_j^v - S_i^u \geq E_u + P_S \times \text{delay}_b(i,j) \tag{1}$$

Then the objective of the linear program solver is to minimize P_S, the global period of the system, while satisfying all the precedence constraints induced by the channels as modelled in Equation 1.

However, this scheduling technique does not support resource constraints. In ASCENT NoCs, where the hardware does not handle contention, the software scheduler has to ensure resource constraints are met. In other words, the software schedule has to make sure that at any point in time, a communication link is only used by one flow. Thus, to enable this scheduling technique to work over ASCENT's software-defined bufferless NoC, we have made two major extensions to the original work: bufferless channels and initialization phases.

1) Bufferless channels: First, we integrate the notion of bufferless channels as the original SDF models defined buffers as boundless FIFOs. We require the notion of bufferless channels as the data produced needs to be immediately consumed

as they cannot be stored. To introduce this feature, we include additional constraints between execution times of every pair of tasks connected by buffers.

2) Initialization phases: To model resource constraints, we propose to merge multiple SDF tasks into one (Section V-C). Such transformation can induce extra constraints to the graph and pose a risk in introducing deadlocks. For the merged tasks to perfectly match the original graph, we will produce Cyclo-Static Dataflow Graphs with initialization phases. Cyclo-Static DataFlow Graphs or CSDFG [30] is a commonly used model wherein a single task can have multiple phases. Each phase has a particular consumption and execution pattern. However, initialization phases are much less common, and model phases are executed only once. We can also find an example of initialization phases with the Lucy-n language by Mandel and Plateau [31].

From a periodic schedule, we can naturally obtain the 'Start' and 'Period' information required by the ASCENT routers. In Section V, we describe the step-by-step workflow of ASCENT that obtains the router's configurations from a given SDF graph.

V. ASCENT Workflow

In this section, we describe the *ASCENT* framework using Algorithm 1.

A. Task-to-core Mapping and Routing

In this first step indicated as the $MapAndRoute$ function in Algorithm 1, we map each SDF task to a core and establish communication routes between them, similar to the approach proposed in [32]. We obtain a path with the shortest NoC latency using XY routing and minimal NoC link contention by considering all the tasks mapped before this step.

First, the set of tasks $\mathcal{T}$ is sorted topologically. In case of cycles within the graph, we randomly remove one feedback edges to break the cycle. Next, we iterate through the tasks $t \in \mathcal{T}$ and perform the task-to-core mapping. In each iteration, we consider placing the task t in each of the unallocated cores and finally map the task on the core that has the least routing cost. Section V-A1 below describes the procedure for finding the NoC path between communicating tasks while Section V-A2 defines the routing cost when t is mapped to an unallocated core c.

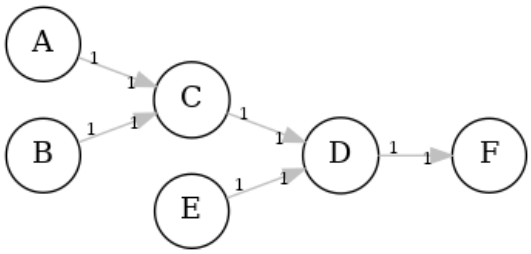


Fig. 4: A simple SDF graph with six tasks.

1) NoC Path: Let $T' \subseteq T$ denote the set of previously mapped tasks that receive/send data from/to t. Let t' denote a task in T' that is mapped to core c'. Using XY routing with shortest NoC latency, multiple paths may exist between c and c' . Let $s_0, s_1, \ldots, s_n$ denote the number of flows that use links $l_0, l_1, \ldots, l_n$ in a given path between c and c'. Let $Sharing\_Degree(t, t') = max(s_0, s_1, \ldots, s_n)$ denote the sharing degree of this path. We iteratively consider each task $t' \in T'$ and choose the path between t and t' that has the least sharing degree using Dijkstra's algorithm [33]. In case of a tie, we randomly pick one among all the paths that have the least sharing degree.

2) Routing Cost: Let t, t' denote two tasks in T that are mapped to cores c and c' respectively. $NoCLat(t, t')$ denotes the shortest latency using XY routing between c and c'. The routing cost is defined as the sum of the cost due to the NoC latency ($Cost_{lat}$) and the cost due to the NoC link contention ($Cost_{cont}$). We define $Cost_{lat}$ as the sum of $NoCLat$ between t and $t', \forall t' \in T'$ divided by the total number of communicating tasks ($|T'|$) as stated in Equation 2.

$$Cost_{lat} = \frac{\sum_{t' \in T'} NoCLat(t, t')}{|T'|} \tag{2}$$

We define $Cost_{cont}$ as the maximum sharing degree between t and $t', \forall t' \in T'$ as stated in Equation 3.

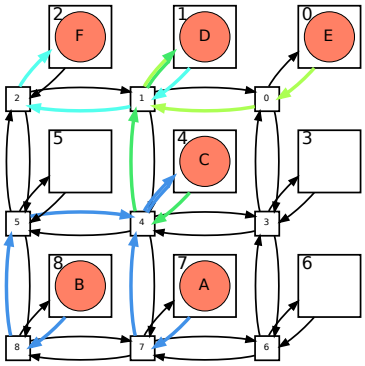
$$Cost_{cont} = max_{t' \in T'}(Sharing\_Degree(t, t')) \tag{3}$$

We map t on the unallocated core with the least routing cost and store the chosen path between t and t', $\forall t' \in T'$ (obtained using Dijkstra's algorithm as described before). Figure 4 shows an SDF graph containing six tasks and Figure 5a is its corresponding task-to-core mapping and NoC routing on 3x3 2D-Mesh.

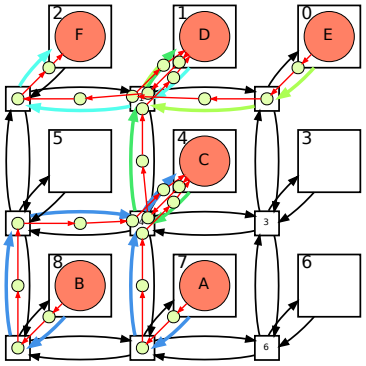
In this example, the tasks are first ordered topologically, E, B, A, C, D, F. The first three tasks can be mapped to any core as they do not share any communication (the cost is null). In this example, E is mapped to 0, B is mapped to 8, A is mapped to 7. No routing is required yet. Next, tasks C and D are mapped one after another, while limiting contention and NoC Latency, i.e. the number of router hops. The algorithm goes through all available cores, considers only paths with shortest NoC Latency and selects the path with lowest sharing degree. Task C is mapped in such a way that it is close to tasks A and B, while task D on the other hand is mapped by taking the mapping of tasks C and E into consideration. When the task F has to be mapped contention becomes larger. Thus, by placing F closer to the task D in the core 2, we are able to achieve low contention and NoC latency.

B. Graph Transformations: Modelling Network Links

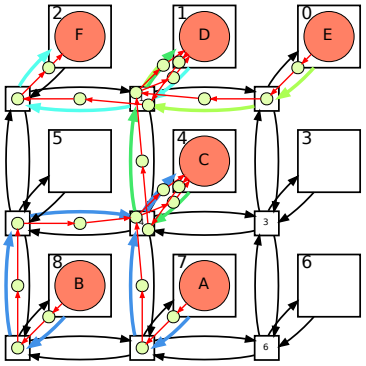
The second stage of ASCENT is the $ModelNetwork$ function that models network communications. While network modelling is common [34], we diverge from previous works as we consider a particular kind of channel where token lifetime



(a) MAPPING & ROUTING: This is the result of a combined mapping and routing strategy applied to the motivation example from Figure 4. The graph is mapped on a 3x3 NoC. Each route is marked with a distinct color.

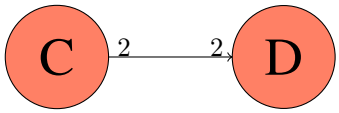


(b) NETWORK MODELLING: Each network device used by SDF buffers is modeled. Note that multiple tasks can represent the same device. For example, three tasks correspond to Routers 0 and 4 as three communication channels pass through them.

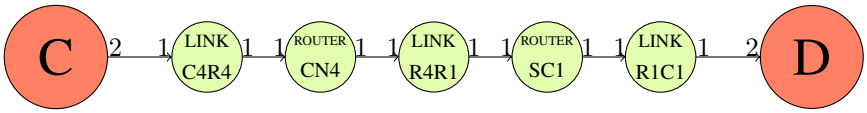


(c) CONSTRAINTS MODELLING: Conflicting tasks are merged. For example, Routers 1 and 4 only have two tasks associated with them. When tasks are not merged, it means that they did not overlap in the final schedule as depicted in Figure 8.

Fig. 5: This figure represents the three major steps of the ASCENT framework. Figure 5a represents the mapping and routing of the SDF. Figure 5b shows how the network devices are modeled. Figure 5b presents how conflicting tasks are merged after multiple iterations of scheduling.



(a) Before



(b) After

Fig. 6: For C to communicate with D as per the specification in Figure 5a, data has to transit through three different links (C4R4, R4R1, and R1C1) and two different routers (CN4, SC1). The result of this transformation is shown in Figure 5b.

must be equal to zero. These channels are used to model *bufferless* NoC communication.

Each buffer involved in network communications can be replaced by a series of additional tasks representing the different network devices used and the cost of communication along the routes previously determined. Precisely, we perform the following transformations:

- Any link in the network traversed by the communication between two tasks is modeled by a "link" task t that consumes and produces $PktSize_t$ tokens (the packet size) with an execution time of E_t (the hop latency). To match the NoC architecture we target, we set a packet size of $PktSize_t = 1$ and a hop latency of $E_t = 1$ for every link t.
- Any router traversed by communication is modeled as a "router" task t that also consumes and produces $PktSize_t$ tokens with an execution time of E_t. To match the NoC architecture we target, we set $PktSize_t = 1$ and $E_t = 0$ for every link t.
- The execution time of routers is zero as within a single clock cycle a packet can traverse a router and the following link. Additionally, the data cannot be stored between two network tasks; when data is produced by a "link" task, it must be consumed by the "router" task. For this reason, all buffers between network devices are considered *bufferless*.

These artificial tasks are used to collect the number of times a router needs to be activated. Ultimately this information will be used to configure the NoC routers.

For example, in the SDF presented in Figure 5a, task C is mapped to core 4, while task D is mapped on core 1 in a 3x3 mesh. The buffer between the tasks C and D will thus be replaced by the proposition in Figure 6. As the route from C to D goes through routers 4 and 1, we replace the buffer b_{CD} by a series composed of two "router" tasks and three "link" tasks. First, the "link" task C4R4 connects the Core 4 (C4) to Router 4 (R4). Second, the "router" task CN4 connects the Core (C) to the North (N) of Router 4. It is followed by a "link" R4R1 from Router 4 to Router 1, a "router" task SC1 (South to Core 1), and finally a "link" R1C1 from Router 1 to Core 1.

The result of this transformation is depicted in Figure 5b. We denote an important number of "router" tasks associated with the same routers (i.e., routers 1 and 4). This happens when multiple communications go through the same router and thus instantiate different "router" tasks. It is also important to note these "router" tasks might or might not be compatible in the sense of scheduling at the same time. For example, NW1 and CS1 can be scheduled simultaneously, while CS1 and CW1 cannot. To ensure the absence of this contention, we will transform the graph and serialize communications that would conflict otherwise: this is the modelling of resource

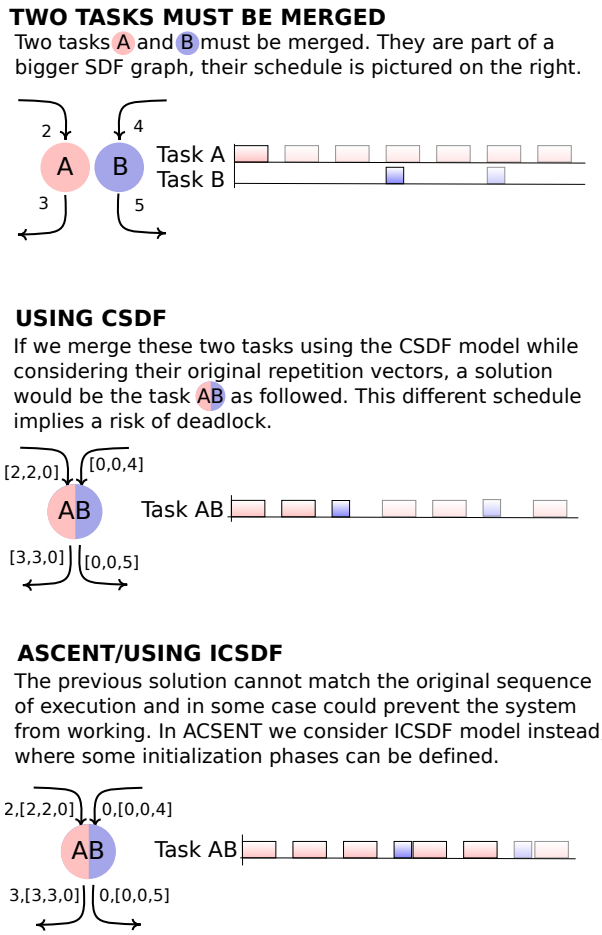


Fig. 7: Merging SDF tasks into a single node can decrease performance or even generate deadlock situations. Simply turning SDF tasks into CSDF tasks can be problematic. Our solution creates ICSDF where CSDF tasks can have initialization phases.

constraints.

C. Graph Transformations: Modelling Resource Constraints

This third stage will iteratively look for contention (with $FindConflict$) and sequentialize the "router" tasks that cannot be executed at the same time (with $MergeTasks$).

1) $FindConflict$: A conflict is defined by two "router" tasks (for example CS1 and CW1) that are scheduled at the same time while they should not be (CS1 and CW1 both need to use the C1R1 entry link). To identify conflicts, we schedule the system using the K-Periodic scheduling strategy (previously defined Section IV-B) and we note the incompatible "router" tasks scheduled at the same time. Once a conflict is identified we perform the merge operation described below.

2) $MergeTasks$: Given two tasks, the `MergeTasks` function merges them into a single SDF task. To obtain the required granularity, the merged tasks will be modeled by Cyclo-Static tasks as defined in the Cyclo-Static Dataflow (CSDF) model [30]. The CSDF tasks can have multiple phases, such that each of these phases can have different production and consumption rates. In our case, each phase of the CSDF task would correspond to one of the tasks that will be merged.

We show how two tasks in an SDF graph will be merged in Figure 7. Task A consumes 2 tokens and produces 3 tokens in each of its execution, and the task B consumes 4 tokens and produces 5 tokens every time it executes. When scheduled, these tasks requires two executions of A for one execution of B. To merge these tasks, the most common strategy would be to produce a CSDF task, i.e., a task with multiple phases where each phase can either model the execution of A or B. In this example, the task AB mimics the execution of A two times followed by B. This pattern will repeat forever. The CSDF notation $[v_1, v_2, \cdots]$ in Figure 7, denotes the consumption rate and production rate for each phase. For example, $[2, 2, 0]$ means that for two phases, task AB will consume 2 tokens from the top left buffer, while the third phase will consume none.

However, the execution order of the tasks diverges from the original schedule. By simply specifying such phases when merging two tasks, we risk degrading the performance of an application or even creating a deadlock. This is demonstrated with the schedules in Figure 7.

In our solution, we define initialization phases that are executed only once at the start of the scheduling. This allows us to accurately match the original schedule and thus avoid deadlock. We note initialisation phases before the periodic phases of a CSDF task; $i_1, i_2, \cdots, [v_1, v_2, \cdots]$ denotes that the initial phases $i_1, i_2, \cdots$ will be executed first, then the periodic pattern $v_1, v_2, \cdots$ will repeat forever. Precisely, from the Initialized CSDF (ICSDF) in Figure 7, task AB has one initial phase where it consumes and produces data only from the right buffer, next followed by a standard periodic pattern. From experimental evaluations, we observe that this methodology does not degrade performance in most cases.

D. Generating NoC switch configuration

We have obtained a conflict-free version of our application that can lead to a valid schedule (see Figure 8). The application was simple enough that the obtained schedule is 1-Periodic, there is only one starting time defined per task. The starting times are $S_1^A = S_1^B = S_1^E = 0$, $S_1^C = 5$, $S_1^D = 9$, and $S_1^F = 13$. The periods for every task are the same, $P = 2$.

The final step is to reuse the final schedule to populate the routers' configurations. Indeed, the generated schedule consists of multiple starting times and a period per task. It is then used to generate the task schedule and crossbar switch configurations of the NoC. In addition, given the K-Periodic nature of the scheduler we used, the size of schedules can be controlled to fit hardware limitations, while maximizing throughput.

For example, two tasks are responsible for Router 1's schedule. To program Router 1, we need to enumerate each periodic activation of these tasks. The result of this configuration is shown in Figure 9. From these tasks, we found four different configurations that start at cycles 3, 5, 7, and 8 with an execution period of 2 cycles.

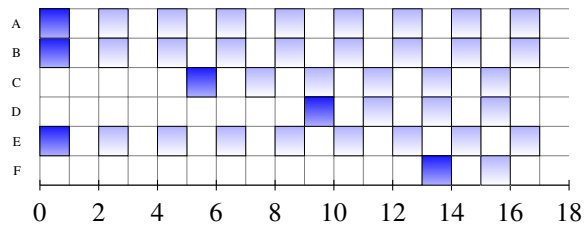


Fig. 8: Cycle-by-cycle application schedule after modelling resource constraints depicted in Figure 5c. Highlighted executions are the task starting times defined by the schedule.

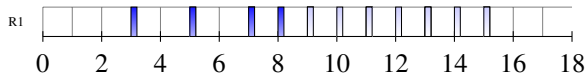


Fig. 9: Router 1 configuration after modelling resources constraints depicted in Figure 5c.

VI. Experimental Evaluation

In this section, we compare the throughput attained in *ASCENT* with respect to the state-of-the-art TDM-based NoCs and an ideal dynamically-routed NoC.

Benchmarks. We utilize five Digital Signal Processing (DSP) benchmarks: *BufferCycle*, *H263Decoder*, *Modem*, *MP3*, *SampleRate* from the SDF3 benchmark suite [12]. These benchmarks are executed on 4x4 cores connected using a 2D mesh network, with one-to-one task-to-core mapping. We chose 4x4, the lowest number of cores that can fit all these applications.

Methodology. We run *ASCENT* offline scheduler for each application to obtain the router configurations and exact scheduling. At run-time this information is loaded into the router. We compare this result with different techniques in terms of performance and hardware resources requirement. We assume that the NoC link hop latency is one cycle across all mechanisms. We choose *TDM OPT* as it provides the best performance with the lowest slot-table entries amongst other TDM-based approaches. In *TDM OPT* [10], the minimum number of time-slots and their NoC configurations are determined for a worst-case scenario in which a core communicates with every other core in the system, i.e., all-to-all communication. We also simulate an ideal dynamically-routed NoC where packets take the shortest paths and are never blocked within the router due to unlimited buffering and virtual channels. The task-to-core mapping in each technique is determined using the approach explained earlier in Section V-A, i.e., the same task-to-core mapping is used for different communication scheduling mechanisms. Thus communication scheduling can be fairly compared across different mechanisms.

Simulation. The SDF graph contains precise timing information on task execution time and data flows between tasks. The offline mapping and scheduling algorithm is able to compute a deterministic task and communication schedule. We are also able to find the set of task executions and packet transfers that repeats itself (called hyper-period in SDF parlance). We are able to determine the application throughput from this schedule. We also generate a packet trace with timing information for this entire period. We faithfully model software-defined ASCENT routers connected along a 2D-Mesh that takes packet flows, task mapping and router configurations as input and performs packet routing on the NoC based on the communication schedule. We use this simulator to verify that the data packets reach the destination without NoC link contention.

TABLE I: Throughput in ASCENT with respect to TDM OPT [10] and Ideal dynamic routing

Benchmark	Throughput in ASCENT w.r.t.	
	Ideal Dyn. Routing	TDM OPT [10]
BufferCycle	1	10.80
H263Decoder	1	4.28
Modem	1	6.70
MP3	1	1.00
SampleRate	1	6.00

Performance. *ASCENT*, the offline communication scheduler proposed in this work consumes less than 1 minute to compute the communication schedule for all the benchmarks evaluated in this work.

We compare the throughput of the different benchmarks while using different communication scheduling algorithms in Table I. For *TDM OPT* and ideal dynamically-routed NoCs, tasks are executed using As Soon As Possible execution strategy (ASAP) where tasks can execute immediately after the required data are available in their buffers. *ASCENT* follows the mechanism explained earlier in Section IV. From this table, we observe that *ASCENT* achieves better performance with an average of $5.8\times$ improvement over the TDM-based approach for the benchmarks. This is because the packets no longer need to wait till their allocated time-slot. Besides, *ASCENT* also takes packet timing information into account for better communication scheduling. In *MP3*, *TDM OPT* attains the same throughput as *ASCENT* because it is not a communication-intensive benchmark. *ASCENT* is able to obtain the same throughput as that of an ideal dynamically-routed NoC as our communication scheduler interleaves packets cycle-by-cycle on the links just like dynamic NoCs.

We define run-time as the end-to-end execution time of an application that includes SDF initialization. When K-Periodic scheduling is utilized, application throughput requirements can be achieved. However, it could lead to poor application runtime due to initialization overhead. Hence, we analyse the runtime speedup of *ASCENT* with respect to *TDM OPT* [10] in Figure 10. From this figure, we observe that with smaller input sizes *ASCENT* may achieve lower performance improvements. For instance, in *SampleRate* with small input size, the runtime is dominated by the initiation phase in *ASCENT* leading to poor runtime compared to *TDM OPT*. However, as the input size increases, the runtime improvement approaches the throughput improvement.

Scalability. *ASCENT*'s micro-architectural structures per core for supporting cycle-by-cycle software scheduling (FSMs, counters) scale with the router specification, i.e., the number of possible flows between incoming and outgoing links, rather than with the number of cores. In *ASCENT*, each of the five ports (North, South, East, West, Core) are connected to four

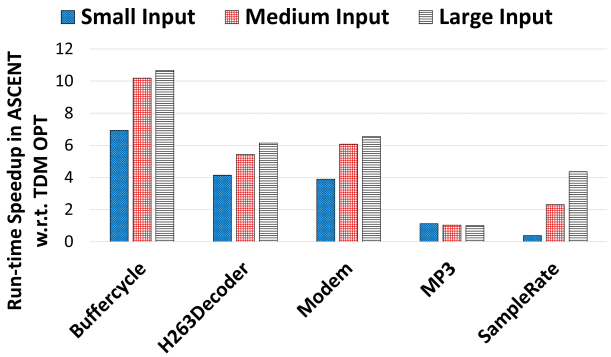


Fig. 10: Application run-time speedup in *ASCENT* with respect to TDM OPT [10].

TABLE II: Slot table size for different mechanisms.

Mesh Config.	TDM Lower Bound [10]	TDM OPT [10]	TDM Heur. [9]	ASCENT
4x4	15	18	59	20
5x5	24	34	112	20
8x8	63	N.A.	481	20

links (excluding self). Hence, the number of slot table entries remains unchanged at 20 even for a large 100x100 mesh. This is unlike the slot table overhead of TDM-based approaches.

Table II shows the slot table size of several state-of-the-art TDM techniques. In the TDM techniques, the slot table entries are determined assuming all-to-all communication, i.e. every core communicates with all the remaining cores in the system. In *TDM Lower Bound* [10], we present the theoretical least number of slots required to realize an all-to-all communication pattern. A n core system will require $n-1$ slots per router as each core communicates with $n-1$ cores. This is a theoretical slot allocation mechanism that may not be feasible. *TDM OPT* [10] represents the optimal number of slot size with a feasible schedule. Note that though *TDM Lower Bound* requires 15 slot entries for 4x4 mesh, we need a minimum of 18 entries for achieving a feasible schedule. *TDM OPT* determines the schedule and slot table size using Integer Linear Programming (ILP). Thus, it is not possible to obtain the optimal number of time slots for meshes larger than 5x5 due to an exponential increase in the problem-solving run-time. Thus, we compare with *TDM Heur.* [9], a heuristics-based approach for TDM NoCs. Although this mechanism provides a feasible schedule, there is a huge increase in the slot table sizes between *TDM OPT* and *TDM Heur* [9] as mesh size increases. *ASCENT* is not only scalable in terms of the number of slot table entries but also provides at least 5.8X better average throughput when compared to the existing TDM based approaches like *TDM OPT* and *TDM Heur*.

RTL Design and Synthesis. The ASCENT NoC router described in Section IV-A has been implemented in RTL and synthesized using Synopsys Design Compiler. A NoC router consumes 4.9 mW at 1.1 GHz on a commercial 40 nm process at 0.0142 mm^2 area. In particular, the FSMs form the bulk of the router (nearly 90% of the total area) as there is no buffering and they only contain wires. This trend is similar to the TDM NoC where significant area is taken up by the slot tables. Compared to a 4x4 mesh TDM NoC with 18 slot table entries, ASCENT's counters and FSMs lead to only 19.5% area overhead while consuming 2.3X less power. Though there is slight area overhead, we achieve significant power savings. The reduction in power arises as the ASCENT router only needs to change the connections periodically, leading to significantly reduced dynamic power. However, TDM NoC needs to change the router connections every cycle and hence consumes more dynamic power. ASCENT router consumes more area as counters are required to manage the periodic connections. As 4.9 mW power is rather low (it is only 4.3% of core power in [5]), we leave further optimization of the ASCENT NoC router for future work. For instance, the counters and FSMs can be obviated by embedding the schedule within the packet header instead, similar to source-routed NoCs. At each NoC router, the hardware looks at the header bits to determine which output port a flit will be connected to and sets the mux select signals appropriately. Hardware-wise, the tradeoff is header overhead vs. router overhead. Embedding the schedule in the header will require more wires to transmit the header, but this is amortized over multiple flits of a packet anyway, just as in dynamic NoCs. Software-wise, this requires the scheduler to handle multi-flit packets when reserving network resources.

VII. Conclusion and Future Work

We propose *ASCENT*, a scheduling mechanism that exploits the inherent periodicity in SDF applications to perform task-to-core mapping and task-/communication- scheduling on buffer-less software-defined NoCs. *ASCENT* provides 5.8x better throughput on average for the applications in the SDF3 benchmark suite compared to existing TDM-based mechanisms and matches the performance of an ideal dynamically-routed NoC while ensuring predictability.

In this work, we evaluated the proposed mechanism only on 2D-mesh as it is commonly used in many-core architectures. This approach could be used to schedule communication on other regular and irregular NoC topologies like torus, hybrid mesh-star, etc. as long as the NoC network can be expressed using a directed graph. Sophisticated mechanisms and evaluation would be required to attain the best schedules for various NoC topologies. We would like to explore this as a future work.

The proposed work is focused on the scheduling and assumes a specific microarchitecture and topology of the NoC hardware. A future research direction would be to co-design the NoC architecture in conjunction with the scheduling method. For example, the proposed work considers a small and homogeneous NoC design. But we could also explore heterogeneous designs where router sizes could gradually increase depending on their location. This direction could potentially lead to even smaller design, while still maintaining guarantees for the type of application it supports.

References

[1] T. Mitra, J. Teich, and L. Thiele, "Time-Critical Systems Design: A Survey," *IEEE Design and Test*, 2018.

[2] R. Banakar, S. Steinke, B. S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: A design alternative for cache on-chip memory in embedded systems," in *Hardware/Software Codesign - Proceedings of the International Workshop*, 2002.

[3] V. Venkataramani, A. Pathania, and T. Mitra, "Unified thread-and data-mapping for multi-threaded multi-phase applications on spm many-cores," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1496–1501.

[4] V. Venkataramani, M. C. Chan, and T. Mitra, "Scratchpad-memory management for multi-threaded applications on many-core architectures," *ACM Trans. Embed. Comput. Syst.*, vol. 18, no. 1, Feb. 2019. [Online]. Available: https://doi.org/10.1145/3301308

[5] V. Venkataramani, A. Kulkarni, T. Mitra, and L.-S. Peh, "Spectrum: A software-defined predictable many-core architecture for lte/5g baseband processing," *ACM Trans. Embed. Comput. Syst.*, vol. 19, no. 5, Sep. 2020. [Online]. Available: https://doi.org/10.1145/3400032

[6] S. Hesham, J. Rettkowski, D. Goehringer, and M. A. El Ghany, "Survey on real-time networks-on-chip," 2017.

[7] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, A. Rocha, C. Silva, J. Sparsø, and A. Tocchi, "T-CREST: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, 2015.

[8] K. Goossens, J. Dielissen, and A. Rădulescu, "Æthereal network on chip: Concepts, architectures, and implementations," *IEEE Design and Test of Computers*, 2005.

[9] F. Brandner and M. Schoeberl, "Static routing in symmetric real-time network-on-chips," in *ACM International Conference Proceeding Series*, 2012.

[10] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki, "A statically scheduled time-division-multiplexed network-on-chip for real-time systems," in *Proceedings of the 2012 6th IEEE/ACM International Symposium on Networks-on-Chip, NoCS 2012*, 2012.

[11] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, 1987.

[12] S. Stuijk, M. Geilen, and T. Basten, "SDF3: SDF for free," in *Proceedings - International Conference on Application of Concurrency to System Design, ACSD*, 2006.

[13] T. Bjerregaard, "The MANGO clockless network-on-chip: Concepts and implementation," Ph.D. dissertation, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2005, supervised by Assoc. Prof. Jens Sparsø, IMM. [Online]. Available: http://www2.compute.dtu.dk/pubdb/pubs/4025-full.html

[14] D. Wiklund and D. Liu, "SoCBUS: Switched network on chip for hard real time embedded systems," in *Proceedings - International Parallel and Distributed Processing Symposium, IPDPS 2003*, 2003.

[15] R. A. Stefan, A. Molnos, and K. Goossens, "DAElite: A TDM NoC supporting QoS, multicast, and fast connection Set-Up," *IEEE Transactions on Computers*, 2014.

[16] U. M. Mirza, F. Gruian, and K. Kuchcinski, "Mapping streaming applications on multiprocessors with time-division-multiplexed network-on-chip," *Computers and Electrical Engineering*, 2014.

[17] T. Picornell, J. Flich, C. Hernández, and J. Duato, "DCFNoC: A delayed conflict-free time division multiplexing network on chip," in *Proceedings - Design Automation Conference*, 2019.

[18] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J. W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The raw microprocessor: A computational fabric for software circuits and general-purpose programs," *IEEE Micro*, 2002.

[19] S. Sriram and S. S. Bhattacharyya, *Embedded multiprocessors: Scheduling and synchronization*. CRC press, 2018.

[20] B. Bodin, A. Munier-Kordon, and B. D. de Dinechin, "K-periodic schedules for evaluating the maximum throughput of a synchronous dataflow graph," in *2012 International Conference on Embedded Computer Systems (SAMOS)*. IEEE, 2012, pp. 152–159.

[21] B. Bodin, A. Munier-Kordon, and B. D. De Dinechin, "Optimal and fast throughput evaluation of CSDF," in *Proceedings - Design Automation Conference*, 2016.

[22] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J. F. Nezan, and S. Aridhi, "Preesm: A dataflow-based rapid prototyping framework for simplifying multicore DSP programming," in *EDERC 2014 - Proceedings of the 6th European Embedded Design in Education and Research Conference*, 2014.

[23] R. Leupers, M. A. Aguilar, J. F. Eusse, J. Castrillon, and W. Sheng, "MAPS: A software development environment for embeddedmulticore applications," in *Handbook of Hardware/Software Codesign*, 2017.

[24] A. Bonfietti, M. Lombardi, M. Milano, and L. Benini, "Maximum-throughput mapping of SDFGs on multi-core SoC platforms," *Journal of Parallel and Distributed Computing*, 2013.

[25] H. N. Tran, A. Honorat, J. P. Talpin, T. Gautier, and L. Besnard, "Efficient contention-aware scheduling of SDF graphs on shared multi-bank memory," in *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, 2019.

[26] V. Venkataramani, B. Bodin, A. Kulkarni, T. Mitra, and L.-S. Peh, "Time-predictable software-defined architecture with sdf-based compiler flow for 5g baseband processing," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 1553–1557.

[27] Y. Lesparre, A. Munier-Kordon, and J. M. Delosme, "Evaluation of Synchronous Dataflow Graph Mappings onto Distributed Memory Architectures," in *Proceedings - 19th Euromicro Conference on Digital System Design, DSD 2016*, 2016.

[28] T. Harde, M. Freier, G. von der Brüggen, and J. J. Chen, "Configurations and optimizations of TDMA schedules for periodic packet communication on networks on chip," in *ACM International Conference Proceeding Series*, 2018.

[29] K. Rosvall and I. Sander, "Flexible and tradeoff-aware constraint-based design space exploration for streaming applications on heterogeneous platforms," *ACM Transactions on Design Automation of Electronic Systems*, 2017.

[30] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete, "Cyclo-static data flow," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 1995.

[31] L. Mandel and F. Plateau, "Scheduling and buffer sizing of n-synchronous systems typing of ultimately periodic clocks in lucy-n," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012.

[32] S. Friedman, A. Carroll, B. Van Essen, B. Ylvisaker, C. Ebeling, and S. Hauck, "SPR: An architecture-adaptive CGRA mapping tool," in *Proceedings of the 7th ACM SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA'09*, 2009.

[33] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[34] S. Sriram and S. S. Bhattacharyya, *Embedded multiprocessors: Scheduling and synchronization, second edition*. CRC Press Inc, 2009.

Vanchinathan Venkataramani is a Senior Design Engineer at Advanced Micro Devices (AMD), Bengaluru. He received his Ph.D. in Computer Science from School of Computing, National University of Singapore (NUS) in 2020. His research focuses on low-power real-time many-core architecture design and power-management for heterogeneous multi-core architectures.

Bruno Bodin is an Assistant Professor at Yale-NUS College, Singapore with a joint appointment at the School of Computing, National University of Singapore. He received his PhD degree in Computer Science at the University Pierre and Marie Curie, France, in 2013. His research interests include performance analysis and optimisation; embedded systems and robotic applications.

Aditi Kulkarni Mohite received her MS degree from University of Southern California, USA, in Electrical Engineering in 2011. She is currently working as a Research Assistant at National University of Singapore.

Peh Li Shiuan is Provost's Chair Professor of Computer Science at National University of Singapore (NUS). Previously, she was on the faculty of MIT and Princeton. She graduated with a Ph.D. in Computer Science from Stanford University in 2001, and a B.S. in Computer Science from the National University of Singapore in 1995. Her research focuses on networked computing, in many-core chips as well as mobile wireless systems. She is an IEEE Fellow.

Tulika Mitra is Provost's Chair Professor of Computer Science at National University of Singapore (NUS). Her research interests span various aspects of the design automation of embedded real-time systems, cyber-physical systems, and Internet of Things. Prof. Mitra currently serves as the Editor-in-Chief of the ACM Transactions on Embedded Computing Systems.