Design-Time Scheduling of Periodic, Hard Real-Time Flows for NoC-based Systems

Anderson R. P. Domingues*, Sergio J. Filho*, Alexandre de M. Amory†, Luciano Ost‡, Fernando G. Moraes*
*School of Technology, Pontifical Catholic University of Rio Grande do Sul – PUCRS – Porto Alegre, Brazil
†RETIS Lab., Sant'anna School of Advanced Studies – Pisa, Italy
‡ Loughborough University – England
anderson.domingues@edu.pucrs.br, alexandre.amory@santannapisa.it, l.ost@lboro.ac.uk, {sergio.filho, fernando.moraes}@pucrs.br

2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS) | 978-1-6654-8823-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICECS202256217.2022.9970868

***Abstract*—Real-time Networks-on-Chips (RT-NoCs) provide timing guarantees for communication in many-cores. However, RT-NoCs customized routers may conflict with other non-functional requirements such as low-energy consumption, safety, and security. To alleviate the effects of non-functional requirements on the NoC design, we proposed a framework to deal with hard real-time flows without modifying the NoC architecture. One of the drawbacks of the previous framework was its scalability due to the employed integer-linear programming (ILP) backend. In this work, we propose a breadth-first depth backend with parameterized search to accelerate the scheduling to polynomial-time. Due to the large solution spaces, ILP solvers struggle with performance, even for medium-sized applications. Results show that our framework computes a feasible hard real-time flow scheduling with acceptable performance.**

***Index Terms*—Hard real-time, scheduling, NoC, optimization**

I. Introduction and Related Work

Real-time Networks-on-Chips (RT-NoCs) provide a foundation for guaranteeing timing requirements in NoC-based systems. In our previous work [1], we proposed an analytical framework to statically schedule hard real-time packets in NoCs, using the same concept of hyper-period often employed in task scheduling [2]. Instead of presenting another RT-NoC design, we designed a custom module called "time-controlled network interface" (TCNI). When attached to the local ports of NoC routers, the TCNI delays the injection of the packets, forcing them to enter the network at an exact cycle. The configuration of the TCNI is generated by an *scheduling algorithm*, which searches for a table of injection times that could make all packets be delivered before their deadline.

Compared to RT-NoCs, our approach requires no modification of the NoC architecture, relying solely on the auxiliary TCNI module and the network zero-load latency model. Consequently, our approach can be used with NoCs without native support for real-time traffic. As a drawback, our approach assumes some waste of the network bandwidth and adds some area and energy overhead to the design (the TCNI).

A. Related Work

Offline scheduling has been applied to allocate time slots in Time-division multiplexing (TDM) NoCs [3, 4]. However, as Picornell et al. [5] point out, offline scheduling of TDM slots often relies on non-optimal solutions to be computationally affordable. In our framework, time is depicted in *time units*, which in its finest grain is represented in *cycles*. On the one hand, our framework has more possibilities for packet allocation than TDM-based techniques, as packets traversal will not be interrupted once it enters the network. However, a fine grain time unit dramatically increases the search space, often resulting in an intractable problem. Picornell et al. [5] eliminate offline scheduling by configuring delays on each input port, thus relying on the architecture to solve scheduling.

Our past solution to the offline scheduling problem [1] adopted integer linear programming (ILP) and was described using Minizinc language. Other NoC-related studies previously adopted ILP to solve NP-complete problems in a similar context, also having difficulties with solving time and problem representation due to language expressiveness [6, 7]. In this scenario, heuristic algorithms are alternatives to search the solution space in polynomial-time at the cost of sub-optimal solutions, e.g., admitting certain packets to miss their deadlines [7]. However, deadline violations are not acceptable in the hard real-time domain.

B. Goals and Contribution

This paper presents a new scheduling framework, executed at design time, for hard real-time flows. In contrast to our previous ILP framework, the new framework implements a custom backend for a breadth-first search (BFS) algorithm with polynomial-time performance. Among the new features, the new framework can either schedule for a given NoC frequency and, in case of an unfeasible schedule, find the NoC frequency to make the schedule feasible. We dedicate the remainder of this paper to introducing the new framework and discussing the scheduling algorithms, focusing on the implemented pruning strategies and heuristics.

II. The Scheduling Framework

Figure 1 shows the building blocks of our framework. User inputs comprise the application and NoC models and task mapping. The *instantiator* creates an application instance from the inputs targeting a candidate frequency. Then, the *scheduler* — our original contribution — is responsible for scheduling the application packets by generating a table of injection times. The outputs include the configuration of the TCNI modules (injection times) and the minimum frequency required for the schedule to be feasible. Finally, the outputs consist of graphical views of the scheduling and a testbench stub to be used with RTL tools, e.g., in Modelsim.

A. User Inputs

The **application model** is a directed graph $G_A = V_A \times E_A$, where V_A is the set of vertices and E_A is the set of edges.

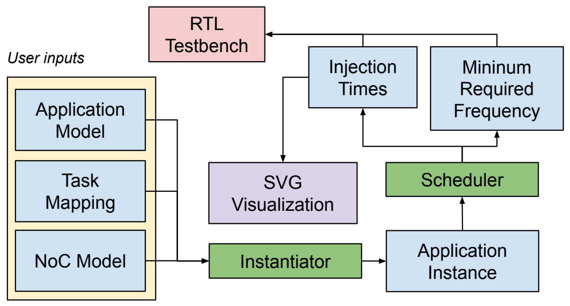

Fig. 1. The building blocks of the proposed framework.

Vertices denote the source of the communication flows in the system, e.g., tasks and sensors. Edges represent the communication flows, containing: (i) period; (ii) data size; (iii) deadline.

Figure 2 illustrate an example application. For the sake of simplicity, their tasks periods and deadlines are equal so that each flow generates only a single packet during the hyperperiod. Tasks A, B, C, and D are mapped into processing elements 0, 2, 1, and 3, respectively. Formally, **task mapping** is a relation $M : V_A \times V_T \to \mathbb{B}$, where $\mathbb{B}$ is the binary domain.

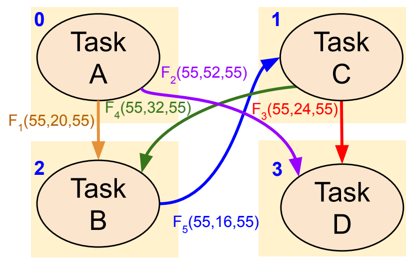

Fig. 2. Synthetic application. Edge labels represent flows period, data size, and deadline (in parenthesis). Periods and deadlines are measured in seconds, data is measured in bytes.

The **NoC model** corresponds to (i) the NoC topology, (ii) flit width, (iii) routing algorithm, the (iv) zero-load latency model, and (v) target frequency. We describe the NoC topology using directed graphs $G_T = E_T \times V_T$, where the edges E_T corresponds to links, and vertices V_T correspond to routers. The routing algorithm is a complete function $R : V_T \times V_T \to (E_T)^*$, denoting the path (sequence of links) that a packet takes when traversing from one router to another. The zero-load latency model represents the cost of transmitting packets when no contention exists, formally described as $Z : V_T \times V_T \to \mathbb{D}$, where $\mathbb{D}$ is the discrete-time domain. Finally, the framework will try to schedule packets considering the target frequency whenever possible. However, may the schedule be unfeasible for the target frequency, the framework will suggest a higher frequency.

B. Instantiator

The instantiator module uses the NoC model to create a candidate version of the target application assuming the selected frequency. The application instance is described as $I : E_T \times P \to \mathbb{D}^3$, representing a mapping of links (network resources) and packets onto three fields: (i) occupancy, the amount of time that a packet requires from a single resource during its transmission; (ii) the minimum release time, which prevent packets from being injected too soon in the network (lower bound injection time); and (iii) deadline (upper bound injection time is given by $deadline - occupancy$). Table I shows an example of an application instance created by our framework using the *flow unwrapping* method [1], which extents the timing analysis to the hyperperiod H, given by $H = lcm(F_p)$, the least common multiplier of all flow periods.

Equation 1 computes the occupancy of packets, where r is the routing time (first flit only, wormhole), $hops$ is the number of routers in the path of the packet, $data\_size$ is as in the application model, and $flit\_width$ is as defined in the NoC model. For instance, packet $P2$ (from flow $F2$) uses links L-0, 0-1, 1-3, 3-L, with an occupancy equals to 32 cycles (2 hops, 52 bytes in 32-bit flits, r equal to 6 cycles).

$$Occupancy_i = r \times (hops + 1) + \frac{data\_size_i}{flit\_width} + 1 \quad (1)$$

TABLE I
AN INSTANCE OF THE SYNTHETIC APPLICATION.

	Pck.	**Links**											
		0-1	1-3	2-3	2-0	3-1	3-2	0-L	L-0	1-L	L-2	3-L	L-3
occupancy	P_1	18	–	–	–	–	–	–	18	18	–	–	–
	P_2	32	32	–	–	–	–	–	32	–	–	32	–
	P_3	–	–	19	–	–	–	–	–	–	19	19	–
	P_4	–	–	27	–	27	–	–	–	27	27	–	–
	P_5	–	–	–	23	–	23	23	–	–	–	–	23
min. start	P_1	0	–	–	–	–	–	–	0	0	–	–	–
	P_2	0	0	–	–	–	–	–	0	–	–	0	–
	P_3	–	–	0	–	–	–	–	–	–	0	0	–
	P_4	–	–	0	–	0	–	–	–	0	0	–	–
	P_5	–	–	–	0	–	0	0	–	–	–	–	0
deadline	P_1	55	–	–	–	–	–	–	55	55	–	–	–
	P_2	55	55	–	–	–	–	–	55	–	–	55	–
	P_3	–	–	55	–	–	–	–	–	–	55	55	–
	P_4	–	–	55	–	55	–	–	–	55	55	–	–
	P_5	–	–	–	55	–	55	55	–	–	–	–	55

Note: unused links were omitted

After generating the first application instance, it is submitted to the *scheduler*, which will try to determine the injection time of packets. If the scheduler fails to find a *feasible schedule*, the instantiator increases the frequency and re-executes the scheduler until it finds a feasible schedule. After finding the first feasible schedule, the instantiator searches the minimum frequency using a binary search heuristic. As the frequency increases, the possibilities for allocating packets increases as well, although occupancy is constant.

III. SCHEDULER

The scheduler goal is to find a feasible schedule, that is, to search the *solution space* for a schedule in which the following constraints hold: (I) no packet deadline is violated, (II) packet will not simultaneously share resources, (III) no packet will be injected into the network before their minimum release time. Constraints I and III are guaranteed by construction, as the application instance carries the candidate release times interval. Constraint II is a range overlapping checking that performs $O(n)$ on the number of links.

The solution space is given by the intervals in which packets can be injected into the network, bound by their minimum release time and maximum release time ($deadline - occupancy$). Table II shows the solution space for the application in Figure 2, whose the number of possible schedules is given by $(37 \times 23 \times 36 \times 28 \times 32) = 2,7449,856$. Such a solution space can be searched in a couple of seconds. However, the size of the application instance and NoC (2×2 2D mesh, single channel, 4-byte channel width, XY routing) are very small and do not coincide with real-world applications size. For instance, our framework unwraps the application introduced by Shi et. al. [8] into 659 packets (same NoC but 4×4 dimensions) with $\simeq 7 \times 10^{206}$ possible schedules. The scheduling problem is *intractable* and cannot be solved through exhaustive search except for very small instances.

TABLE II
SEARCH SPACE FOR SYNTHETIC APPLICATION

	Pck.	Links											
		L-0	0-1	1-L	1-3	3-L	L-2	2-3	3-1	L-3	3-2	2-0	0-L
search space	P_1	[0;37]	–	–	–	–	–	–	[0;37]	[0;37]	–	–	–
	P_2	[0;23]	[0;23]	–	–	–	–	–	[0;23]	–	–	[0;23]	–
	P_3	–	–	[0;36]	–	–	–	–	–	–	[0;36]	[0;36]	–
	P_4	–	–	[0;28]	–	[0;28]	–	–	–	[0;28]	[0;28]	–	–
	P_5	–	–	–	[0;38]	–	[0;38]	[0;38]	–	–	–	–	[0;38]

We prune the search space by inserting a *pruning factor* (PF) parameter to the algorithm. Instead of increment the release time by one cycle after each allocation attempt, we increment it by the prune factor. The prune factor "aligns" the allocation to a multiple factor by skipping neighbor solutions. For instance, if an allocation attempt just failed, we can skip ahead a couple of values as neighbor values are likely to fail as well. For any $n \in \mathbb{N}^+$, a PF of n reduces the size of the search space to $\frac{a}{n}$ where a is the number of alternatives for the exhaustive algorithm.

A. Guided Search

Pruning the search space considerably reduce the number of alternatives for medium-to-small sized problems. However, it is not of much use in larger problems. For instance, it reduces the solution space of Shi et al. [8] problem from $\simeq 7 \times 10^{206}$ to only 7×10^{204} if using a PF of 100 (which means skipping 99 alternatives ahead).

For larger problems, we selectively choose which packets to try to allocate first, using one of three *sorting criteria*: (i) Least Slack-Time First (LSTF), allocates packets with fewer scheduling alternatives first; (ii) Most Bandwidth Consuming First (MBCF), allocates packets that consume more bandwidth first as they are likely to collide with other packets; and (iii) Most Critical Path First (MCPF), a version of MBCF weighted on the use of individual network links. The guided search algorithm has two steps: (i) executing LSTF, MCPF, or MBCF to generate an ordered list of packets, and (ii) allocate nodes according to the generated list. Regardless the adopted criteria, the guided search algorithm performs $O(pm)$, where p is the number of packets to schedule and $m = a/n$ is the number of alternatives per packet.

B. Adaptive Guided Search

The guided search (LSTF, MBCF, and MCPF) may eventually fail to allocate one or more packets, approximating the performance of the exhaustive search algorithm. Critical packets are those that congest links due to their period-size ratio. Allocating such packets early in the algorithm skips the exploration of dead nodes (local minima) in the solution space.

Figure 3 shows the pseudo-code for the *adaptive guided search* algorithm. This algorithm assumes the same two steps as the guided search algorithm. However, we add a *threshold* to limit the number of tries when scheduling packets. Once a packet fails to be scheduled, it will be selected to be scheduled first at the next algorithm reset. The algorithm works as follows. First, it generates a ordered list R of packets using either LSTF, MBCF, or MCPF (line 3). The goal is to remove all packets from R and either schedule them (line 15) or flag them as critical (line 21). Every time a packet could not be scheduled, it increments the number of tries for that packet (line 16). Finally, the algorithm returns the generated schedule and the set of packets that could not be scheduled (line 22).

```
Algorithm Adaptive Guided Search
 1. Inputs: the set of packets to be scheduled, P.
 2. Begin
 3.     Let R ← lstf(P) or mbcf(P) or mcpf(P)
 4.     Let Q ← an empty ordered set
 5.     Let S ← { } // the skipped packets
 6.     Let F ← the prunning factor,
 7.     Let G ← threshould (max. tries per packet)
 8.     Let T ← [ zeroes ] // tries per packet (critical pkts.)
 9.     While |R| ≠ 0:
10.         Let schedule ← { }, temp_schedule ← { }
11.         For q in Q:    // partial schedule
12.             schedule ← allocate(schedule, q, F)
13.         temp_schedule ← schedule
14.         For r in R:    // adaptive permutation
15.             If not allocate(temp_schedule, r, F) Then
16.                 T[r] ← T[r] + 1
17.                 If T[r] < G Then
18.                     R.push_front(r) // next to be tried
19.                     Goto Line 7
20.                 Else // packet could not be scheduled
21.                     R.remove(r), S.push(r)
22.         Return temp_schedule, S
23. End
```

Fig. 3. Adaptive guided search algorithm. Packets that could not be scheduled will be selected to be tried first at the next algorithm reset. The output is the set of scheduled packets and a list of packets that could not be scheduled.

C. The Minimum Frequency Required for a Feasible Schedule

Using the adaptive guided search algorithm, we detect which packets could not be scheduled for the given application instance. Although not searching the entire search space, we could find a feasible schedule in an acceptable computing time. In cases where we could not find a schedule, we detect which packets could not be scheduled. By employing a binary search algorithm, we tune the NoC frequency in such a way to find the minimal frequency allowing to schedule all packets.

TABLE III
EXECUTION PERFORMANCE (*visited solutions/skippped solutions*). COLUMNS: CRITERIA AND PRUNING FACTOR.

	LSTF			MBCF			MCPF		
Applications	10	20	40	10	20	40	10	20	40
Synthetic-Application-D	40/96	16/12	6/2	40/96	16/12	6/2	6/4	6/2	10/3
DCT-Verify [9]	8/158	8/314	8/80	8/416	8/209	8/105	214/8970	111/2322	60/626
Shi et al. [8]*	660/13629	660/7267	660/3683	660/13641	660/7282	660/3684	660/5073	660/3002	660/1762

* : Prunning Factor ×100

TABLE IV
MINIMUM SCHEDULING FREQUENCY (MHZ)

		Criteria and Pruning Factor		
Applications	**PF**	LSTF	MBCF	MCPF
Synthetic-Application-D	10	1,928.1	1,928.1	1,928.1
	20	1,928.1	1,928.1	1,928.1
	40	2,182.7	2.182.7	2,182.7
DCT-Verify [9]	10	792.0	1700.6	791.9
	20	794.8	1706.3	794.7
	40	800.0	1706.3	800.0
Shi et al. [8]	10	2,812.5	2,812.5	4,218.7
	20	2,812.5	2,437.5	4,218.7
	40	2,812.5	2,437.5	4,687.5

PF: Prunning Factor

IV. RESULTS

We evaluate our framework by comparing the three proposed criteria LSTF, MBCF, and MCPF, using three applications: (i) Synthetic-Application-D (Section II); (ii) *verify* task from discrete-cosine transform (DCT-verify) application [9]; (iii) Shi et al. [8] application. The Synthetic-Application-D has multiple scheduling alternatives; it has 4 tasks mapped into 4 nodes with 5 flows and 5 packets. The DCT-verify application has 7 tasks mapped to 6 nodes, each task producing one flow. Finally, the Shi et al. [8] application has 33 tasks producing 38 flows, mapped to 16 nodes, resulting in 660 packets. We use PFs 10, 20, and 40, for the first two applications. We increase the PF to 1000, 2000, and 4000 for the third application so that the pruning would have a significant effect on the number of skipped nodes.

Table III shows the performance of the LSTF, MBCF, and MCPF algorithms for each application using different PFs. We observed that the adaptive guided search algorithm could find valid schedules for the Shi et al. [8] application inspecting the same number of nodes in the solution space. The PF is inversely proportional to the number of skipped solutions in all cases. Besides, the number of skipped nodes were similar in LSTF and MBCF criteria. For that application, the MCPF algorithm showed the best visited solution nodes per skipped nodes ration, reaching a solution faster than the other criteria.

Except for Synthetic-Application-D, the number of visited nodes was linear to the number of packets in the application. In the case of the first application, the search space was too small for the PF, leading to a late discovery of a valid solutions for LSTF and MBCF. Contrarily, the MCPF could mitigate the effect of a higher PF.

Table IV shows the achieved frequencies. We initialized our framework targeting 2MHz for all the applications. All criteria reach the same minimum frequency for Synthetic-Application-D when using a PF of 10 and 20. However, a PF of 40 dramatically decreased the number of potential solutions, thus the algorithm picked a “bad” solution, requiring a higher frequency for that schedule to be feasible. A higher PF increase necessary frequency in most cases, although it seemed to have a lower impact for the Shi et al. [8] application.

The Shi et al. [8] application took on average 5 seconds to execute one iteration of the adaptive guided search algorithm (Intel I9-7940X, 3.10GHz, 64 GB RAM). Our previous framework (ILP backend) executed 1 week, and failed to find a solution for the same application.

V. CONCLUSION

This paper presented a framework for scheduling hard real-time traffic in NoC-based systems. In comparison to our previous work, the new framework can handle larger problems due to the herein discussed adaptive guided search. In comparison to the strict behavior of ILP, our framework can tune the target frequency and always find a feasible schedule. Future works include extending the framework to handle design space exploration for energy consumption and area overhead.

ACKNOWLEDGMENT

This work was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES (Finance Code 001), CNPq (grant 309605/2020-2), and FAPERGS (grant 21/2551-0002047-4).

REFERENCES

[1] A. R. P. Domingues, S. J. Filho, A. de M. Amory, and F. G. Moraes, “Design-time analysis of real-time traffic for networks-on-chip using constraint models,” in *SBCCI*, 2022, to appear in.
[2] I. Ripoll and R. Ballester-Ripoll, “Period selection for minimal hyperperiod in periodic task systems,” *IEEE Transactions on Computers*, vol. 62, no. 09, pp. 1813–1822, sep 2013.
[3] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki, “A statically scheduled time-division-multiplexed network-on-chip for real-time systems,” in *NOCS*, 2012, pp. 152–160.
[4] F. Brandner and M. Schoeberl, “Static routing in symmetric real-time network-on-chips,” in *RTNS*, 2012, p. 61–70. [Online]. Available: https://doi.org/10.1145/2392987.2392995
[5] T. Picornell, J. Flich, C. Hernández, and J. Duato, “DCFNoC: A Delayed Conflict-Free Time Division Multiplexing Network on Chip,” in *DAC*, 2019, pp. 1–6.
[6] D. Lee, B. Lin, and C.-K. Cheng, “Smt-based contention-free task mapping and scheduling on smart noc,” *IEEE Embedded Systems Letters*, vol. 13, no. 4, pp. 158–161, 2021.
[7] W. Liu, P. Chen, L. Yang, M. Li, and N. Guan, “Work-in-progress: fixed priority scheduling of real-time flows with arbitrary deadlines on smart nocs,” in *2017 International Conference on Embedded Software (EMSOFT)*, 2017, pp. 1–2.
[8] Z. Shi, A. Burns, and L. Indrusiak, “Schedulability analysis for real time on-chip communication with wormhole switching,” *IJERTCS*, vol. 1, pp. 1–22, 04 2010.
[9] B. Rouxel and I. Puaut, “STR2RTS: Refactored StreamIT benchmarks into statically analyzable parallel benchmarks for WCET estimation & real-time scheduling,” in *WCET*, 2017.