Task and Communication Allocation for Real-time Tasks to Networks-on-Chip Multiprocessors

Chawki Benchehida, Mohammed Kamel Benhaoua, Houssam-Eddine Zahaf, Giuseppe Lipari

Task and Communication Allocation for Real-time Tasks to Networks-on-Chip Multiprocessors

Chawki Benchehida[1,2], Mohammed Kamel Benhaoua[1,3], Houssam-Eddine Zahaf[2], and Giuseppe Lipari[2]

[1]Univ. Oran1 - LAPECI Laboratory. Oran, Algeria
[2]Univ. Lille - CRIStAL, Centrale Lille, UMR 9189. Lille, France
[3]Univ. Mustapha Stambouli. Mascara, Algeria
Email: {firstname.lastname}@univ-lille.fr, k.benhaoua@univ-mascara.dz

***Abstract*—In this paper, we address the problem of analyzing the behavior of a set of real-time tasks on a Network-on-chip-based (NoC) architecture. Our approach is to transform the allocation of tasks and communications within a NoC into a classical real-time allocation problem. It provides an extension of classical bin-packing heuristics to allocate a set of real-time applications modeled using a directed acyclic graphs (DAGs) to a set of processors interconnected through a NoC.**

The paper describes the schedulability analysis, including allocation and communication. It provides also a comparative study of different allocation and communication algorithms and presents accordingly a set of promising research insights.

***Index Terms*—real-time, analysis, Network-on-chip, allocation, communication, latency**

I. Introduction

Autonomous driving, video monitoring, gaming are examples of recent soft and hard real-time applications. These applications process a large amount of data and require high computational power, that can be satisfied by having numerous computing elements on the same chip, i.e. massively parallel architectures. Traditional parallel hardware have been a direct descendant of single core architectures, that is a single bus links memory to computing elements. When increasing the number of processors, concurrency on memory access can lead to bus contention, therefore downgrades performances. Recent tends replace the classical interconnections using bus, by a complex embedded networking infrastructure. That is, computing elements are directly connected to routers, which themselves are connected to other routers, leading to the well-known network-on-chip paradigm.

Real time systems are subjected to timing constraints, i.e., system's correctness depends on the result delivery time. A real-time workload must complete within a specified time window, called deadline. A real-time system is defined as schedulable, if according to certain scheduling policy, all tasks meet their deadlines. The respect of real-time constraints has to be ensured prior to runtime. To achieve precise schedulability analysis, the real-time tasks implementation can be abstracted through different models. Directed acyclic graphs (DAGs) are one of the most expressive models. In such model, tasks are represented by nodes and edges between nodes denote the data transfer that has to be achieved between tasks.

When executing real-time systems onto a NoC-based architecture, the schedulability depends on (i) the task allocation, and (ii) inter-task communication. Each has received a particular attention in the real-time community. The task allocation has been widely addressed for bus-based architecture, and effective algorithms have been proposed to optimize several goals, such as energy [21], resource utilization, etc. Regarding the communication, several research works have proposed NoC architectural modifications to reduce worst case inter-task communication time. Mainly they can be classified to: (i) priority-based and TDMA-based. We have reported an experimental and analytical comparison between communication protocols for real-time systems onto NoCs in [4].

Few works only have focused on both allocation and communication issues at the same time for real-time systems. This problem is NP-hard in the strong sense. It is extremely time consuming to compute optimal solution as the design space is very large. Therefore, it is more convenient to design efficient heuristics to achieve fast and efficient design-space exploration. One way, to design allocation heuristics, is to assign intermediate artificial deadlines for every task, therefore allowing tasks to be analysed independently from each other.

1) Contributions: The artificial offset-and-deadline-assignment techniques have not been proposed for NoC-based architectures. In this paper, we propose an new approach to allocate real-time tasks into NoC architecture by extending classical deadline assignment heuristics, as well as classical bin-packing heuristics for real-time DAG tasks. We first, propose an extension of bin-packing heuristics to the NoC-based multiprocessors. Further, we propose a new deadline assignment techniques for NoC based architecture, and provide analysis for allocating both communications and tasks. The paper also provides promising research insights when considering allocation and communication issues at the same time.

The rest of the paper is organized as follows: In the next section, we report the related work. System and architecture model are presented in Section III. Further, we present independently real-time approaches for (i) bounding communication latency within NoCs, and (ii) deadline assignment for multiprocessor based architectures. Section IV-C reports how real-time tasks can be executed within NoC-based

multiprocessors using TDMA for communication and classical bin-packing heuristics for task allocation. Results of the both simulation and analysis and their discussions are presented in Section V. We draw our conclusions in Section VI.

II. Related Work

Network-on-chip interconnection paradigm has many advantages compared to classical bus-based interconnections, such as its scaling capacity. However, as hundreds of computing units are embedded on the same chip, the task allocation is a complex issues compared to classical bus-based architecture compound of few processors. Finding an optimal task allocation is an NP-complete problem and it has been the subject of several works. For non-real-time tasks, authors in [5], [9], [16], [18] have proposed offline (static), as well as on run-time mapping strategies (on-the-fly) for both task and communications. However, None of these proposals considers critical real-time system requirement. Authors in [14], [17], [21], [20], [8] have considered the task allocation for real-time tasks within classical bus based architectures. A good survey on real-time task allocation can be found in [15].

In bus-based real-time systems, communication latency is analyzed and included in the task worst case execution time as it do not much depend on the task allocation itself. However, such techniques can not be used in the context of NoC-based architecture, communication depends drastically on the task allocation, therefore it must be considered independently from the task worst-case execution time. Several techniques have been proposed that can be classified into two categories. The first uses *TDMA* (Time-division multiple access) to regulate the medium communication access, while the second assign a priority to each message, and a scheduling policy is applied. A comparative study of these techniques are reported in [4] using simulation and analysis. [10], [13] proposed techniques for TDMA quantum assignment to minimize communication latency, i.e find a near-optimal VC slot assignment using heuristics. An exhaustive survey can be found in [11].

However, None of these works above combine task and communication assignment at the same time. In this paper, we propose an allocation approach for both tasks and communication using TDMA for communication and classical bin-packing heuristics for allocations.

III. System Model

In this section, we present our architecture and our task models. First, we overview NoC architectures basic concepts. Further, we model real-time tasks using DAGs.

A. Architecture Model

The NoC is an interconnection paradigm has been introduced in [6] to overcome the bus-based interconnection limitations. It links the processors through an embedded network. Each processor is connected to a router via a network interface. Each couple of a router and processor is called a tile. As in classical networks, tiles can be arranged according to different topology. 2D-Mesh is the most used topology in NoCs. It aligns tiles in a square matrix of $\mathsf{m} \times \mathsf{m}$. Let $\mathcal{P}$ denote a set processor, $\mathcal{P} = \{p_1, \cdots, p_{m \times m}\}$. Each tile p_i is connected to typically 4 neighbors, except those at the network edge. For example in Figure 1, we show NoC-based multiprocessor arranged in 2D mesh 3×3 NoC.

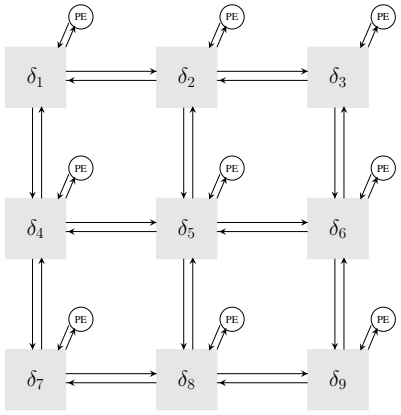

Fig. 1: 2D-Mesh NoC Architecture

When exchanged between tiles, a message M_i is first split into several *Packets* ($M_i = \{P_{i1}, \cdots, P_{ij}\}$). In Wormhole switching, each packet is broken into small data units, called *FLIT* (FLow control unIT). Within a tile, incoming FLITs are stored in local buffers, called virtual channels (VCs). Further, They are moved forward to their next destination according to a routing algorithm. In this paper, we consider XY routing. Flits are first moved to the next router in the X-axis (horizontal way) until reaching the destination router column, and then in the Y-axis (vertically) until reaching the target router. XY routing is simple, deadlock-free and deterministic. When two or more VCs are routed to the same output, they must be arbitrated, as the physical links support only one communication at same time. In this paper, we use a TDMA-based arbitration mechanism. TDMA assigns for each VC a given number of slots, where it is served in a round robin fashion. That is, each VC has a guaranteed service time. TDMA ensures isolation between communications as well as a predictable contention free service time.

B. The DAG task model

A *DAG task* is a Directed Acyclic Graph, characterized by a tuple $\tau = \{\mathsf{T}, \mathsf{D}, \mathcal{V}, \mathcal{E}\}$, where: T is the task period, it represents the minimum inter-arrival time between two consecutive activation of task τ ; D is the relative deadline, all sub-tasks of τ must complete not later than D time units from its arrives; $\mathcal{V}$ is a set of nodes that represent *sub-tasks*. The set $\mathcal{E}$ is the set of edges of the graph $\mathcal{E} : \mathcal{V} \times \mathcal{V}$.

A sub-task $v \in \mathcal{V}$ is the basic computation unit. It represents the elementary chunk of the task code that be implemented by a thread and executed by any computing elements within $\mathcal{P}$ in our architecture. Every sub-task v is characterized by its worst-case execution time $C(v)$. An edge $e(n_i, n_j) \in \mathcal{E}$ models a precedence constraint (and related communication) between node n_i and node n_j and it is characterized by the maximum

amount of the data (in FLIT number) exchanged by its source node n_i and destination node n_j, denoted by $M(n_i, n_j)$.

The set of *immediate predecessors* of a node n_j, denoted by $\mathsf{pred}(n_j)$, is the set of all nodes n_i such that there exists an edge (n_i, n_j). The set of *predecessors* of a node n_j is the set of all nodes for which there exist a path toward n_j. If a node has no predecessor, it is a *source node* of the graph. In our model we allow a graph to have several source nodes. In the same way we define the set of *immediate successors* of node n_j, denoted by $\mathsf{succ}(n_j)$, as the set of all nodes n_k such that there exists an edge (n_j, n_k), and the set of *successors* of n_j as the set of nodes for which there is a path from n_j. If a node has no successors, it is a *sink node* of the graph, and we allow a graph to have several sink nodes.

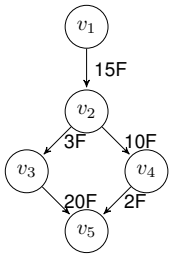

Fig. 2: An example of a dag task

Example 1. *Let consider* $\tau = \{\mathsf{T} = 100, \mathsf{D} = 80, \mathcal{V}, \mathcal{E}\}$ *be a task. The dag,* $\mathcal{V}, \mathcal{E}$*, are reported in Figure 2.*

The task has only one source node, therefore when the task arrives, sub-task v_1 *is activated. Sub-task* v_2 *can not start its executions before* v_1 *finishes its execution and that* v_2 *has received completely* 15 *Flits sent by* v_1*.* v_1 *can not be activated again before at least* 100 *time units have elapsed. When* v_2 *finishes it sends* 10 *flits to* v_4 *(respectively 3 Flits to* v_3*.* v_5 *must finish its execution not later than* 80 *time units from the task arrival.*

IV. Real-time allocation and schedulability

Meeting timing constraints for a set of DAG real-time tasks requires allocating properly their sub-tasks and communications to different tiles. As these sub-tasks communicate, they are forced to respect an execution order dictated by the precedence constraints imposed by the graph structure. Therefore, every sub-task must wait for the completion of its immediate predecessors and their communications before it can start. Analyzing this behavior is complex, because of the large number of tiles to consider. The number of solutions is equal to $m \cdot m \cdot |VC| \cdot \sum_{\tau \in \mathcal{T}} |\mathcal{V}(\tau)|$ where $|VC|$ is the number of VCs per port, thus the design space is extremely large and cannot be completely explored to find optimal solutions in a reasonable time. In this paper, we propose an efficient methodology to allocate sub-tasks to PEs and communications to VCs to achieve fast design space exploration while meeting real-time constraints.

Our algorithm starts by allocating sub-tasks using classical bin-packing heuristics and by doing fast schedulability checking, as described in the next section. Later, it assigns to every communication a virtual channel (VC) where it will be served. Finally, to reduce the complexity of dealing with precedence constraints directly, we impose intermediate offsets and deadlines on each sub-task, i.e. for each sub-task, we compute an artificial activation time and an artificial deadline, and we ensure by analysis that if every sub-tasks is executed within its artificial activation time and artificial deadline, the task where it belongs will respect its deadline. In this way, precedence constraints are automatically respected. Further, the schedulability for every tile can be checked independently using classical single-core analysis, which can be found in the literature of real-time systems, for both fixed priority and EDF. We detail every step in our allocation algorithm in the following sections.

A. Task allocation

This paper uses classical bin-packing heuristics Best-Fit (BF) and Worst-Fit (WF) as disclosed in Algorithm 1.

It starts by sorting tasks according to order, that is either by deadline, or utilization (Line 3). Later, it selects the task on the top of the ordered task list, let it be τ. For every sub-task v in τ, the algorithm selects a sub-set of tiles where v is allowed for allocation (Line 6), (according to Definition 2 and Theorem 1). Further, the eligible tile list is sorted according to the bin-packing allocation heuristics (Line 7), BF for increasing utilization order and WF for a decreasing utilization order. A fast schedulability test is achieved to find the first tile allowing a schedulable allocation. If all eligible tiles have been investigated without finding an allocation that satisfies the schedulabilty test, the system aborts on fail. Otherwise, our algorithm moves to the next sub-task. When all sub-tasks have been allocated, our algorithm moves to the next task. When all tasks have been allocated, Algorithm 1 achieves deadline assignment for every task (Lines 20-22), by subtracting properly the communication latencies from the available slack time, as described in Section IV-C.

This procedure allows our algorithm to convert a complex allocation problem to multiple single-processor schedulability problem, for which well-known techniques can be found in the literature of real-time systems (Lines 23-27). If schedulability fails in a tile, the algorithm aborts on fail, otherwise the sub-task and communication assignment allows respecting timing constraints.

The function select_eligible_tiles (Line 6), returns a list of couples processor-VC where the sub-task and communication can be feasibility allocated according to the necessary schedulability test in Theorem 1.

B. Communication latency

We adopt for this paper TDMA-based communication. Therefore, synchronously on all routers, one VC is served for its quantum. It provides isolation of FLIT forwarding, therefore prevents *miss-behaving* communications from monopolizing the network. However it requires synchronization

Algorithm 1 Bin-packing allocation

input: $\mathcal{T}$: set of tasks, alloc : BF or WF, order : D or U
output: A list of tuples (sub-task, Processor, VC)
sort_tasks_by(order)
for $(\tau \in \mathcal{T})$ **do**
  for $(v \in \tau)$ **do**
    eligible_list = select_eligible_tiles(v)
    sort_tiles(alloc, eligible_list)
    allocated = false
    for $(p \in$ eligible_list$)$ **do**
      if $((u(v) + U(\mathcal{T}_p) \leq 1)$ **then**
        add_sub − task_to_taskset$(v, \mathcal{T}_p)$
        allocated = true
      end if
    end for
    if (allocated == false) **then**
      return FAIL
    end if
  end for
end for
for $(\tau \in \mathcal{T})$ **do**
  assign_deadlines_and_offsets(τ)
end for
for $(\mathsf{p} \in \mathcal{P})$ **do**
  if (check_schedulability(p) == FAIL) **then**
    return FAIL
  end if
end for
return success

mechanisms in the routers. Under TDMA, each communication latency between vertex v and v' can be computed as shown in Equation (1).

$$\mathsf{lat}(VC, v, v') = \frac{L_i}{n_{slot}} \cdot \frac{\Delta}{\eta_i} + H_i \tag{1}$$

Where :

- L_i : number of flits in the message.
- n_{slot} : The amount of data sent in one slot (1 Flit by default) in Virtual channel VC.
- Δ / η_i : The total number of slots in a *TDMA cycle* / the assigned slot number.
- H_i : Hop number between δ_{source} and $\delta_{destination}$.

Once the allocation of every sub-task is achieved, our algorithm computes all communications costs according to Equation (1) [12]. Further, schedulability can be checked. We describe how to isolate the schedulability is checked for each sub-task by the mean of deadline and offset assignment.

C. Deadlines and offsets assignment

Many authors have proposed techniques to assign intermediate deadlines and offsets to task graphs. In this paper we report the two of the most used techniques, *proportional share* and *fair share*, reported in [22].

Most of the deadline assignment techniques are based on the computation of the execution time of the critical path. A path $\pi_x = \{v_1, v_2, \cdots, v_l\}$ is a sequence of sub-tasks of task τ such that:

$$\forall v_l, v_{l+1} \in \pi_x, \exists e(v_l, v_{l+1}) \in \mathcal{E}.$$

Let $\Pi(\tau)$ denote the set of all possible paths of task τ. The critical path $\pi_{crit}(\tau) \in \Pi(\tau)$ is defined as the path with the largest cumulative execution time of the sub-tasks.

In contrast to classical deadline assignment techniques, We define the slack $\mathsf{Sl}(\pi, \mathsf{D})$ along path π as a function of the execution time of its sub-tasks and also of the communications latency that must be achieved between the sub-tasks of path π.

$$\mathsf{Sl}(\pi, \mathsf{D}) = \mathsf{D} - \sum_{v_l \in P} C(v_l) - \sum_{\substack{v_l \in \pi \\ v_{l+1} \in \pi}} \mathsf{lat}(\mathsf{VC}, v_l, v_{l+1})$$

where VC is the allocated virtual channel for communication between v_l and v_{l+1}.

The assignment algorithm starts by assigning an intermediate relative deadline to every sub-task along a path by distributing the path's slack as follows:

$$\mathsf{D}(v) = \mathsf{C}(v) + \mathsf{calculate\_share}(v, \pi)$$

The calculate_share function computes the slack for sub-task v along the path. This slack can be shared according to two alternative heuristics:

- **Fair distribution:** assigns slack as the ratio of the original slack by the number of sub-tasks in the path:

$$\mathsf{calculate\_share}(v, \pi) = \frac{\mathsf{Sl}(\pi, \mathsf{D})}{|\pi|} \tag{2}$$

- **Proportional distribution:** assigns slack according to the contribution of the sub-task WCET in the path:

$$\mathsf{calculate\_share}(v, \pi) = \frac{\mathsf{C}(v)}{\mathsf{C}(\pi)} \cdot \mathsf{Sl}(\pi, \mathsf{D}) \tag{3}$$

Once the relative deadlines of the sub-tasks along the critical path have been assigned, we select the next path in order of decreasing cumulative execution time, and assign the deadlines to the remaining sub-task by appropriately subtracting the already assigned deadlines. The complete procedure has been described in [19], and is not reported here for space constraints.

Let $\mathsf{O}(v)$ be the offset of a sub-task with respect of the arrival time of the task's instance. The sum of the offset and of the intermediate relative deadline of a sub-task is called *local deadline* $\mathsf{O}(v) + \mathsf{D}(v)$, and it is the deadline relative to the arrival of the task's instance.

The offset of a sub-task is set equal to 0 if the sub-task has no predecessors; otherwise, it can be computed recursively as the maximum among the local deadlines of the predecessor sub-tasks.

Figure 3 illustrates the relationship between the activation times, the intermediate offsets, relative deadlines and local

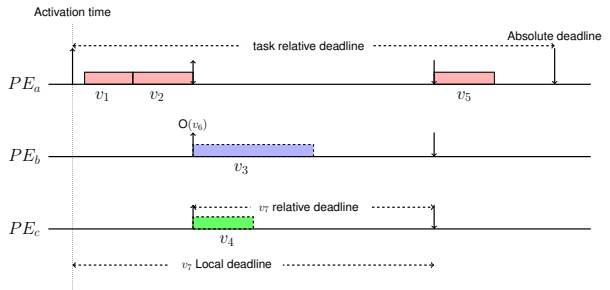

Fig. 3: Example of offset and local deadline

deadlines of the sub-tasks of the task depicted in Figure 2. We assume that v_1, v_2, v_5 have been allocated on the same PE whereas v_3 and v_4 each on a different engine. The activation time is the absolute time of the arrival of the sub-task instance. The activation time of a source sub-task corresponds to the activation time of the task graph. The offset is the interval between the activation of the task graph and the activation of the sub-task. The local deadline is the interval between the task graph activation and the sub-task absolute deadline.

Definition 1. *Sub-task $v \in \mathcal{V}_\tau$ is feasible if for each task instance arrived at a_j, sub-task v executes within the interval bounded by its arrival time $a(v) = a_j + \mathsf{O}(v)$ and its* absolute *deadline* $a(v) + \mathsf{D}(v)$.

Lemma 1. *A task is feasible if all its sub-tasks are feasible.*

Proof. By the definition, the local deadline of the sink sub-tasks is equal to the deadline of the task D. Moreover, the offset of a sub-task is never before the local deadline of a preceding sub-task. Therefore 1) the precedence constraints are respected, 2) if sink sub-tasks are feasible, then the task is feasible. □

Definition 2. *Let p be a processor and v be a sub-task of task τ.*

p is an illegible processor for v if :

$\forall \pi \in \Pi(\tau)$ such that $v \in \pi \implies \exists \mathsf{VC} \in p$ that can be allocated to v and verifying condition $\mathsf{SI}(\pi, \mathsf{D}) \geq 0$

Theorem 1. *Let p be a processor and v be a sub-task.*

if p is not an illegible processor to v, than v can not be feasibility allocated to p.

Proof. The proof is done by counter example. Let assume that p is not illegible to v and that the system is schedulable.

By negating Definition 2, it exis ts at least one path where $\mathsf{SI}(\pi, \mathsf{D}) < 0$. Therefore, one or more sub-tasks will have an execution time greater to their deadlines, thus missing their deadlines and Lemma 1 can not be satisfied. □

The slack computation allows us to ensure that all communications will be achieved will not push a sub-task miss its deadline as they have their own reserved time that is not included in the distributed slack.

D. Single core schedulability analysis

At this step, we assume that deadlines have been assigned to every sub-task, and that every sub-task is allocated to a tile, and communication to VC. The goal of this step is to check if the real-time tasks will respect their timing constraints. Schedulability for FP can be checked using the test proposed in [1]. The latter has a high complexity, therefore we allow also using the test proposed in [3] which is less compless but pessimistic. EDF schedulability can be checked using workload requirement using the schedulability test proposed in [2]. This test has been extended for tasks presenting offsets, as follows :

$$dbf(\tau, t) = \max_{v \in \tau} \sum_{v' \in \tau} \lfloor \frac{t - \Theta(v') - D(v') + T(\tau)}{T(\tau)} \rfloor \quad (4)$$

where:

$$\Theta(v') = (O(v') - O(v)) \ mod \ T(\tau)$$

Thus, our approach has converted the task and communication allocation problem to a single core analysis problem.

V. Results and Discussions

In this section, we evaluate the performances of our proposals on a large set of synthetic experiments. The taskset generation starts by invoking UUnifast algorithm to generate the utilization rate of n sub-tasks. Hence, DAG and inter-sub-tasks communications are generated by TGFF [7]. In the architecture side, we map these tasks on a 3×3 2D-Mesh NoC with routers, containing for each of them 6 VC with the following TDMA slot configuration : $[4, 2, 3, 5, 3, 3]$. To avoid untractable hyper- periods, the period of every task are randomly generated from a list of values between the interval of 1000 and 10000. Communications workload are flit-based quantified, i.e; for each communication, we assign a random number of flit in the range of 3 and 40.

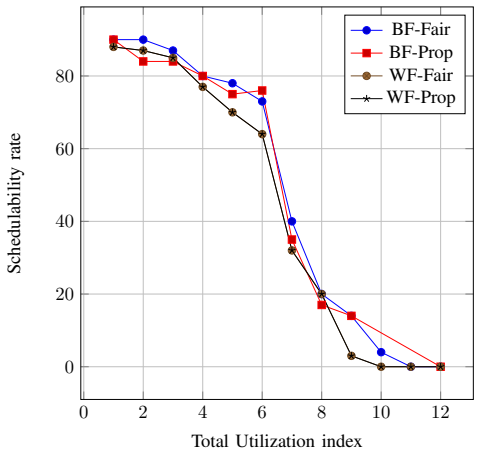

Fig. 4: Heuristics schedulability rate by tasks utilization rate

The code has been executed on a regular laptop with Intel Core i5-7200U processor (2×2.5 GHz) and 8 GB of

ram. The results are shown in Figure 4 and are presented as follows. Each experience takes a heuristic allocation and artificial-deadline assignment method as detailed in IV-C. The results are reported by a function measuring the evolution of the schedulability rate by varying the task utilization rate. Thus, we notice BF heuristic combinations dominate the WF heuristic. This can be explained by observing that BF tries to pack the maximum number of sub-tasks into the minimum of engines, and this allows more flexibility to schedule heavy tasks on other engines.

VI. Conclusion

In this paper, we provide real-time tasks support into NoC-based multiprocessor. Our approach converts a extremely complex task and communication allocation problems to a set of classical scheduling problem, for which efficient algorithms exist, while preseving allocation problem timing properties. We used bin-packing heuristics to allocate tasks on cores and we provided also promising methods for offset and deadline assignment. As future work, we would like to investigate exact solutions for budgeting VCs and more sophisticated heuristics for task allocation.

Acknowledgment

This work was supported in part by MESRS, Algeria and by PHC Tassili project 19MDU213 and by the PRIMA WATERMED 4.0 project.

References

[1] N. C. Audsley. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times*. Citeseer, 1991.

[2] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-time systems*, 2(4):301–324, 1990.

[3] I. Bate and A. Burns. Schedulability analysis of fixed priority real-time systems with offsets. In *Proceedings Ninth Euromicro Workshop on Real Time Systems*, pages 153–160. IEEE, 1997.

[4] C. Benchehida, M. K. Benhaoua, H. E. Zahaf, and G. Lipari. An analysis and simulation tool of real-time communications in on-chip networks: A comparative study. In *EWILI'19*, 2019.

[5] M. K. Benhaoua, A. Singh, A. Benyamina, A. Kumar, and P. Boulet. Heuristic for accelerating run-time task mapping in noc-based heterogeneous mpsocs. *Journal of Digital Information Management*, 12(5):293, 2014.

[6] L. Benini and G. De Micheli. Networks on chips: A new soc paradigm. *computer*, 35(1):70–78, 2002.

[7] R. P. Dick, D. L. Rhodes, and W. Wolf. Tgff: task graphs for free. In *Proceedings of the Sixth International Workshop on Hardware/Software Codesign. (CODES/CASHE'98)*, pages 97–101, March 1998.

[8] P. Dziurzanski, A. K. Singh, and L. S. Indrusiak. Multi-criteria resource allocation in modal hard real-time systems. *EURASIP Journal on Embedded Systems*, 2017(1):30, 2017.

[9] A. Hansson, K. Goossens, and A. Rdulescu. A unified approach to constrained mapping and routing on network-on-chip architectures. In *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 75–80. ACM, 2005.

[10] 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 *RTNS*, pages 202–212, 2018.

[11] S. Hesham, J. Rettkowski, D. Goehringer, and M. A. Abd El Ghany. Survey on Real-Time Networks-on-Chip. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1500–1517, May 2017.

[12] Z. Lu and A. Jantsch. Slot allocation using logical networks for tdm virtual-circuit configuration for network-on-chip. In *Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD 07, page 1825. IEEE Press, 2007.

[13] B. Nikolic, R. Hofmann, and R. Ernst. Slot-based transmission protocol for real-time nocs-sbt-noc. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[14] Q. Perret, P. Maurere, E. Noulard, C. Pagetti, P. Sainrat, and B. Triquet. Mapping hard real-time applications on many-core processors. pages 235–244. ACM Press, 2016.

[15] P. K. Sahu and S. Chattopadhyay. A survey on application mapping strategies for network-on-chip design. *Journal of Systems Architecture*, 59(1):60–76, 2013.

[16] P. K. Sahu, T. Shah, K. Manna, and S. Chattopadhyay. Application mapping onto mesh-based network-on-chip using discrete particle swarm optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(2):300–312, 2013.

[17] M. N. S. M. Sayuti and L. S. Indrusiak. Real-time low-power task mapping in networks-on-chip. In *2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 14–19. IEEE, 2013.

[18] K. Srinivasan and K. S. Chatha. A technique for low energy mapping and routing in network-on-chip architectures. In *ISLPED'05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005.*, pages 387–392. IEEE, 2005.

[19] Y. Wu, Z. Gao, and G. Dai. Deadline and activation time assignment for partitioned real-time application on multiprocessor reservations. *Journal of Systems Architecture*, 60(3):247 – 257, 2014. Real-Time Embedded Software for Multi-Core Platforms.

[20] H. Zahaf, G. Lipari, M. Bertogna, and P. Boulet. The parallel multi-mode digraph task model for energy-aware real-time heterogeneous multi-core systems. *IEEE Transactions on Computers*, 68(10):1511–1524, Oct 2019.

[21] H.-E. Zahaf, A. E. H. Benyamina, R. Olejnik, and G. Lipari. Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms. *Journal of Systems Architecture*, 74:46 – 60, 2017.

[22] H.-E. Zahaf, N. Capodieci, R. Cavicchioli, M. Bertogna, and G. Lipari. A c-dag task model for scheduling complex real-time tasks on heterogeneous platforms: preemption matters. *arXiv preprint arXiv:1901.02450*, 2019.