Memory-processor co-scheduling for real-time tasks on network-on-chip manycore architectures

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

▶ **To cite this version:**

Chawki Benchehida, Mohammed Kamel Benhaoua, Houssam Zahaf, Giuseppe Lipari. Memory-processor co-scheduling for real-time tasks on network-on-chip manycore architectures. International Journal of High Performance Systems Architecture (IJHPSA), 2022, 11 (1), pp.1-11. 10.1504/IJHPSA.2022.121877 . hal-03595577

HAL Id: hal-03595577
https://cnrs.hal.science/hal-03595577v1

Submitted on 3 Mar 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

MEMORY-PROCESSOR CO-SCHEDULING FOR REAL-TIME TASKS ON NETWORK-ON-CHIP MANYCORE ARCHITECTURES

Chawki Benchehida
Univ. Oran1 - LAPECI Laboratory, Oran, Algeria
Univ. Lille, CNRS, Inria, Centrale Lille,
UMR 9189 CRIStAL, F-59000 Lille, France
chawki.benchehida@univ-lille.fr

Mohammed Kamel Benhaoua
Univ. Oran1 - LAPECI Laboratory, Oran, Algeria
Univ. Mustapha Stambouli, Mascara, Algeria
k.benhaoua@univ-mascara.dz

Houssam Eddine Zahaf
Univ. Nantes - LS2N UMR 6004
Nantes, France
houssameddine.zahaf@univ-nantes.fr

Giuseppe Lipari
Univ. Lille, CNRS, Inria, Centrale Lille,
UMR 9189 CRIStAL, F-59000 Lille, France
giuseppe.lipari@univ-lille.fr

ABSTRACT

The Network-on-Chip (NoC) provides a viable solution to bus-contention problems in classical Multi/Many core architectures. However, NoC complex design requires particular attention to support the execution of real-time workloads. In fact, it is necessary to take into account task-to-core allocation and inter-task communication, so that all timing constraints are respected. The problem is more complex when considering task-to-main-memory communication, as the main memory is off-chip and usually connected to the network edges, within the 2D-Mesh topology, which generates a particular additional pattern of traffic.

In this paper, we tackle these problems by considering the allocation of tasks and inter-task-communications, and memory-to-task communications (modeled using Directed Acyclic Graphs *DAGs*) at the same time, rather than separating them, as it has been addressed in the literature of real-time systems. This problem is highly combinatorial, therefore our approach transforms it at each step, to a simpler problem until reaching the classical single-core scheduling problem. The goal is to find a trade-off between the problem combinatorial explosion and the loss of generality when simplifying the problem. We study the effectiveness of the proposed approaches using a large set of synthetic experiments.

1 Introduction

The Network-on-Chip (NoC) provides an alternative to classical bus-based Multi/Many-core architecture interconnection, which experiences high contention when a significant number of cores are integrated on the same SoC. A typical NoC architecture can connect, through an embedded network, more than hundreds of cores. In addition, the NoC itself communicates with the main memory using a bus, which is connected to one or multiple routers on the edges of the network, as disclosed in Figure 1. The complexity of this design makes it difficult to support real-time critical systems, as it requires to incorporate properly the worst-case communication traversal time within the schedulability analysis.

Usually, a real-time system is a compound of several communicating tasks, that might run in parallel on several cores. In traditional real-time systems, the communication between the different tasks is estimated and included in the task execution time. Such design leads to a pessimistic estimation and therefore pessimistic analysis of the system behavior. Such analysis is acceptable for bus-based systems, as the difference between best and worst-case memory access latency is acceptable. When considering NoC based architectures, the communication overheads depend drastically on the task to core allocation. For example, the latency when two communicating tasks are allocated on the opposite network edges is very large compared to the latency when they are allocated to adjacent cores. Hence, it is preferable to separate the communication overheads from the task worst-case execution time estimation.

In the literature of real-time systems, the task-to-resources allocation, and inter-task communication within NoC-based architectures have been studied separately. Several works have proposed different techniques to allocate real-time workload on many-core architecture, while a lot of others have focused on estimating the worst-case communication time (latency) under different network topology and NoC-design assumptions. Although the proposed solutions are efficient and very interesting, they provide poor performances when combined [1]. In this work, we tackle both task-to-core allocation and inter-task communication at the same time, to avoid increasing the pessimism that can be accounted in the two steps. In addition, we consider task-to-main-memory communication, as they generate additional and particular traffic patterns. We, therefore, distinguish off-chip traffic (due to main memory operations), and on-chip traffic (due to inter-task data sharing)

Off-chip communications are costly, as they require to access to main memory, passing therefore by different I/O components, memory controllers and a shared bus. To avoid costly memory access after a job starts its execution, and achieve high memory access predictability, real-time systems community has proposed to split the real-time task execution into two memory phases and one execution phase. During the first memory phase, the data are copied from the main memory to the core local memory where the task is allocated. Further, the task is executed without any access to the main-memory and finally the results are copied back to the main memory in the second memory phase. This execution model is commonly known as The Acquisition Execution Restitution (AER) task model. In this work, we extend this model to DAG-tasks.

Task-to-core allocation problem is an NP-complete. The design space widens when considering inter-task communication, and task to main-memory communication, hence the problem is more complex. The exact algorithm's complexity is very high and very likely untraceable. In this work, we apply a series of conversion transforming the allocation problem, of different tasks and inter-tasks communications to NoC resources, to a single core scheduling problem. This allows achieving a fast, and efficient design space exploration, with an acceptable loss of generality. Therefore, we first extend the directed acyclic task model to include off-chip communication patterns. Further, we propose an efficient memory-free schedulability analysis by eliminating the cost of communication from the task laxity, to finally apply single-core schedulability analysis.

The remainder of this article is organized as follows. In Section 2, we review the related work. Task and architecture models are described in Section 3. In Section 4, we detail the proposed approach. Section 5 extends the proposition by including the off-chip communications. Synthetic experiments are reported in Section 6. Finally, We draw our conclusions and future work in Section 7.

2 Related Work

Network-on-chip interconnection paradigm has been introduced by the seminal paper of Benini et al. [2]. As the task allocation problem is a high conbinatorial problem, many studies are reported on the literature providing techniques to tackle it efficiently with a less computational complexity. Thus, concerning the non-real time applications, the authors in [3], [4], [5] have proposed offline (static) as well as on-line (on-the-fly) mapping strategies for both tasks and communications.

On studies related to bus-based multiprocessors addressed to the real-time systems, the on-chip communication latency is analyzed and included as a part of the worst case execution time of a task and it doesn't much depend on the allocation schema. Obviously, such analysis is not tailored to NoC-based architecture since the communication depends drastically on the task allocation, and therefore, it must be considered independently from the task worst case execution time. Several techniques have been proposed allocate real-time tasks onto multicore architecture [6][7][8][9], as well as communications on NoC. The NoCs can be classified following two categories according to their strategy on network traffic handling. Indeed, congestion might occurs when two communication flows would like reserve a path through the network. The first category concerns the use of *TDMA* (Time-division multiple access) to share the communication medium among communication flows by time quantum reservation, whereas the second category consist of assigning priority to flows, and therefore, the arbitration follows the priority map [10]. A comparative study of both strategies is reported in [1] through the simulation and analysis of several scenarios. Moreover, [11], [12] proposed heuristics to find the optimal TDMA quantum assignment for a minimum communication latency. An exhaustive survey adressed to real-time support on NoC is reported in [13].

Regarding the task-to-main-memory communications, many studies have included an off-chip memory, such as DRAM with the NoC model. In such way, [14] proposed a design of a DRAM-sensitive NoC router consisting of a strategy to handle the communications to the main memory since the packets issued at this purpose have a high priority and are routed first. On the other hand, the authors in [15] proposed a mechanism to handle multiple DRAM coupled to a NoC. Thus, they proposed a solution to sort an out-of-order arrival requests from the NoC to different main memories by an algorithm that re-orders those requests and routes them to the specified DRAM. However, all those previous

work consider a NoC with 2D-Mesh topology. Likewise, Jin et al. [16] proposed a mapping technique in a hierarchical tree-based NoC where bridges are deployed to connect directly the routers to the DRAM controller. Thus, they proposed a task allocation algorithm to avoid the congestion since only one task has the access to the off-chip memory through one bridge and therefore, the tasks are sequentially re-ordered.

Nonetheless, the previous studies do not consider applications with real-time constraint. Hence, in the real-time off-chip memory-aware field, Giannopoulou et al. [17] have proposed a Simulated Annealing-based mapping technique for mixed-criticality tasks on Karlay MPPA NoC with a DRAM memory. Moreover, Gomony et al. [18] proposed a middleware to adapt any TDMA-NoC with a main-memory to support real-time systems. In fact, they proposed an adaptation of the main-memory controller by computing a network interface (NI) bandwidth and cores operational frequency to optimize the NoC energy consumption. Besides that, Perret et al. [19] proposed a mapping technique using constraint programming while the budgeting of real-time applications is calculated *a priori*, e.i they evaluate the computing power need as well as the number of memory access of a application.

To the best of our knowledge, there is no study addressing the task mapping problem while the latency of both off-chip and on-chip communications are considered onto a non-clustered NoC platform using TDMA arbitration. The non-clusted mode is when the whole NoC is considered as a unique region, by opposition of the COTS (Commercial-Off-The-Shelf) NoCs where they are mainly sub-divided onto several regions.

3 System Model

3.1 Hardware platform

We consider a NoC-based platform composed of N tiles. Each tile A is composed of a computing element PE, a router R and the interconnection between them. Routers of the different tiles are connected by a set of unidirectional links. The router interconnection scheme defines the network *topology*. In this work we consider 2D-mesh topology, i.e. tiles are structured as a square matrix of m columns and m rows, i.e. $\mathsf{N} = \mathsf{m}^2$. Therefore, we additionally denote a tile as $\mathsf{A}(x, y)$, where x is its position on x-axis, and y its position on the y-axis. For sake of simplicity we denote $\mathsf{PE}(x, y)$ (resp. $\mathsf{R}(x, y)$) to denote the processing element (resp. the router) of tile $\mathsf{A}(x, y)$. Each router $\mathsf{R}(x, y)$ is connected to its *north, south, east* and *west* neighbor, except those on the edges, which have less than 4 neighbors.

A tile on the edge of the network can be connected to a memory sub-controller, denoted as $\mathsf{A}^m(x, y)$. $\mathsf{A}^m(x, y)$ is connected to an edge router $\mathsf{R}(x, y)$ and is in charge of handling memory access requests. Different sub-controllers are connected to the main controller (referred further by DRAM controller), which arbitrates access to the main memory.

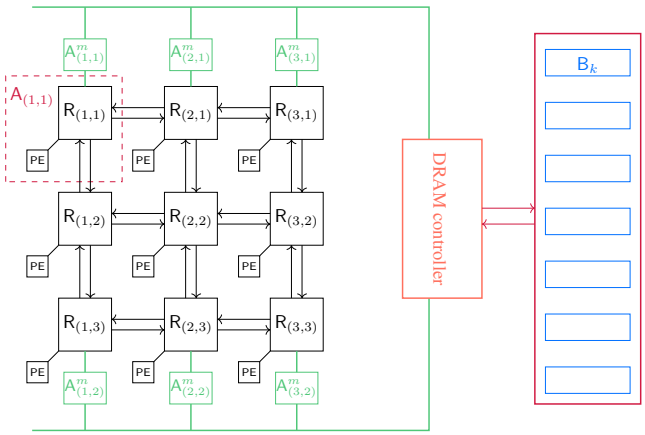

Figure 1: The 3×3 NoC architecture connected with the off-chip DRAM memory by 6 memory sub-controllers

In Figure 1, we define an architecture of 9 tiles, arranged in 3×3 mesh. We define six memory sub-controllers (in green) on northern and southern edge routers, which are connected to the DRAM controller (in orange).

We use wormhole switching flow-control protocol to handle the network traffic flow, within routers. In this protocol, a message $\mathcal{M}_i$ is divided to one or several packets $\mathcal{P}_j$, and each packet is divided on a fixed number of small data unit called FLits (FLow control units). Additionally, Virtual Channels (VC) are deployed onto routers to store flits and serve as buffers when congestion occurs. In fact, Virtual Channels create a multiplexing over link to allow several communications to share the medium. The first flit, i.e. *header* (resp. last flit *tail*) allocates (resp. release) the virtual

channel for exclusive traversal to the packet where it belongs, i.e. a packet has an exclusive access on the VC until its tail flit moves out from the VC of a given router. Further information about wormhole switching can be found in [20].

We consider the TDMA (Time Division Multiple Access) protocol to arbitrate concurrent communication flows at router level. The TDMA allows to each VC to be served on the output link for n_{slot} time slots per TDMA cycle, denoted by Δ.

A message is routed from its source router to its destination router, according to the DOR (Dimension Order Routing) XY protocol. In DOR-XY a message moves between direct neighbor on the x-axis until reaching the y position of its destination router. Afterward, it moves on Y-axis until it reaches its final destination. This routing policy is deterministic and predictable.

3.2 DRAM background

The DRAM is a 3D memory structure, as depicted in Figure 2. It is composed of several independent banks B_k. Each bank is composed of a 2D structure in which a matrix of cells is deployed. By default, a cell holds 1 bit represented by a specific <bank, row, column> triple, denoted also as the memory coordinate. As usually a given data is encoded by several bits, it is indexed by an array of memory coordinate accordingly.

The basic read and write operations are triggered by a tile toward the main memory.

Read operation When $\mathsf{PE}(x, y)$ requests a *read* operation, a sequence of operations are triggered:

1. $\mathsf{A}(x, y)$ initiates the read operation parameters, and sends the request to its network interface NI.
2. $\mathsf{R}(x, y)$ defines according to a lookup table (see Algorithm 2), which memory sub-controller is responsible to achieve this memory operation, let it be $A^m(x', y')$.
3. The request is routed between $\mathsf{R}(x, y)$ and $\mathsf{A}^m(x', y')$ using the mechanisms described above.
4. The memory sub-controller sends the request to main controller through a single bus.
5. DRAM controller computes the data location and sends back data to $\mathsf{A}^m(x', y')$.
6. Data are routed back from $\mathsf{A}^m(x', y')$ to $\mathsf{R}(x, y)$ using the routing protocol.

Write operation The write operation is very similar to the read operation. Instead of sending request and wait for the reply, the request and the data are sent sequentially between $\mathsf{R}(x, y)$ and $\mathsf{A}^m(x', y')$.

3.3 DRAM access latency & bank commands

The latency of memory RW requests between the memory sub-controller and the main memory are difficult to estimate. Perret et al. [21] have proposed a predictable model that simplify the calculation of the data copy latency from a DRAM to a NoC inner-tile and *vice versa*. Their main results can be resumed in Table 1, which discloses different memory operations costs, expressed by clock cycle.

Bank command	Cycles
ACT	68
RD	15
WR	35
PRE	50

Table 1: Bank commands expressed by the number of cycles

Row Activate (ACT) Is the first operation applied on a row, which basically move the row on a dedicated region called *row buffer* in order to fetch it. After the operation, the row is denoted as *opened*.

Read (RD) Read a column from an opened row and retrieve the bit stored in the cell.

Write (WR) Write on a column that belong to an opened row.

Precharge (PRE) Close an opened row. This operation move the row from the row buffer to the bank. It involves the end of the read/write operation on the row.

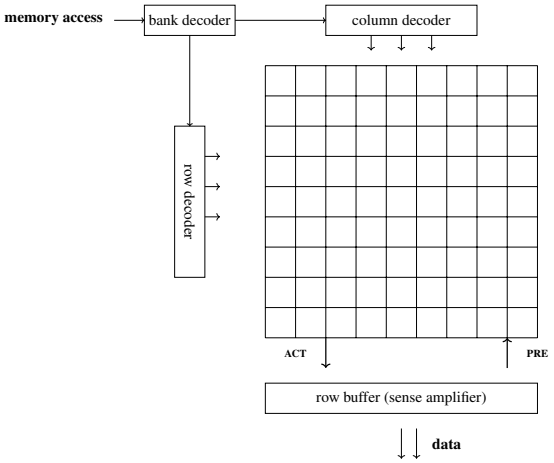

Figure 2: A bank structure of the 3D SDRAM

Similarly [21], we calculate the latency of the main memory access lat_mem as a function of the volume of requested data (vol(req)) by Equation (1).

$$\begin{aligned} \mathsf{lat\_mem}(\mathsf{req}, x) =& \left(\left\lceil \frac{\mathsf{vol}(\mathsf{req})}{\mathsf{vol}^{\mathsf{one}}} \right\rceil \times t_{trans} \right) \times t^{x}_{req} \\ t^{r}_{req} =& t_{ACT} + t_{RD} + t_{PRE} \\ t^{w}_{req} =& t_{ACT} + t_{WR} + t_{PRE} \end{aligned} \tag{1}$$

Where :

- x: can be either r or w for read or write operation respectively.
- $\mathsf{vol}(\mathsf{req})$: is the size of the requested data
- $\mathsf{vol}^{\mathsf{one}}$: is a constant that represents the size of the maximum copied data in one transaction
- t_{trans} the cost of in cycles of a single transaction

t^{r}_{req} and t^{w}_{req} are computed using the parameters disclosed in Table 1.

3.4 AER DAG task model

In this paper, we consider the allocation of a set of tasks $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$ to a NoC-based architecture. Each task τ_i is a Directed Acyclic Graph (DAG), defined by tuple: $\tau_i = \{\mathcal{V}_i, \mathcal{E}_i, \mathsf{D}_i, \mathsf{T}_i\}$.

$\mathcal{V}_i$ denotes the set of sub-tasks of task τ_i. A sub-task of $\mathcal{V}_i$ can be one of two types : *compute* or *memory*. A compute sub-task, denoted as $v_{i,j}$, represents an atomic sequential chunk of code, it is characterized by $C(v)$ denoting its execution time. Different sub-tasks are allowed to run in parallel or concurrently. Similarly to AER model, we have a memory read phase, modeled by sub-task v^{r}_{i}, i.e. loads data from the main memory, while the memory write-back operation is denoted by v^{w}_{i}. When it is not necessary, we unload the notation of sub-task $v_{i,j}$ as only v.

$\mathcal{E}_i$ denotes the set of edges. Each edge $e(v, v')$ is a communication between source sub-task v and destination sub-task v'. An edge $e(v, v') \in \mathcal{E}$ expresses also the precedence constraint between v and v', i.e. v' can not start its execution before the completion of v. Each edge is weighted by the maximum amount of data that can be sent from v to v'. Thus, it is expressed by $\mathcal{M}(v_i, v_j)$, the number of exchanged packets between them.

T_i represents the period of a task, e.i the minimum inter-arrival time between two consecutive instances of τ_i. D_i is the relative deadline of task τ_i. Every sub-task of $\mathcal{V}_i$ must finish its execution no later than D_i time units from the task arrival time. We consider constrained deadline tasks, that is $\mathsf{D}_i \leq \mathsf{T}_i$

Sub-task v' is an *immediate predecessor* of sub-task v, if it exists an edge $e(v', v) \in \mathcal{E}$. We denote by $pred(v)$ the set of all immediate predecessors of sub-task v. The set of all *predecessors* of a sub-task v is the set of all sub-tasks for

which there exists the a path toward v_i. The sub-task having no predecessors is the *source* sub-task of the DAG. In our task model, the source sub-task is always the memory sub-task v_i^r. Similarly v' is an *immediate successor* of v, if v is an immediate predecessor of sub-task v. We denote by $succ(v)$ as the set of all immediate successors of v. The set *successors* of v is the set of sub-tasks for which there is a path from v. Sink node v^w is a sub-task having no successor. In our model, it is always the memory sub-task v_i^w.

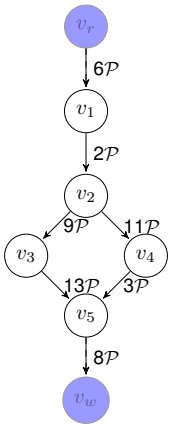

Figure 3: An AER DAG task example.

Example 1. *Let consider* $\tau = \{\mathcal{V}_i, \mathcal{E}_i, \mathsf{D} = 120, \mathsf{T} = 120\}$ *a DAG task, depicted in Figure 3.*

When the task is activated, it starts by the read memory sub-task v^r *to perform a data copy from the main-memory.* v_1 *begin its execution directly after the data has been copied entirely (6 packets) to the local memory of its PE. After* v_1 *ends its execution, it sends data to* v_2 *through the network since* v_2 *must wait the end of communication.* v_3 *and* v_4 *are allowed to run in parallel as soon as they receive the packets (9 and 11 packets respectively). As* v_5 *has two predecessors, it must waits until both of its predecessors terminate their execution. Finally, the sub-task* v^w *achieves the write-back operation. The latter must finish no later than the task deadline* 120 *time units from the task activation.*

4 Task and Communication Allocation to PEs and VCs

Meeting timing constraints for a set of DAG real-time tasks requires allocating properly their sub-tasks and communications to different PEs and VCs. 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 starts. Analyzing this behavior is complex, due to the large number of combinations to consider. The number of mapping combination 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 an optimal solution in a reasonable time.

In this work, we apply a set of conversion on the original problem to reduce the complexity of exploring the design space.

Our algorithm is greedy and iterative. At each iteration it starts by defining an arbitrary allocation to sub-task, according to one of a set of bin-packing heuristics. Further, it eliminates the communication allocation problem by extracting the worst-case latency from the task laxity, i.e. the available processor time to execute a given task without missing deadlines. Further, it defines a list of eligible processor on which the next sub-task will be allocated. Finally, we apply single core schedulability analysis. Indeed, we use artificial-intermediate deadline and offset assignment techniques in order to isolate the analysis of the sub-tasks independently of their dependencies. In this section, we describe the allocation algorithm as well as the schedulabilty analysis to ensure a timing respect.

4.1 Task allocation & communication latency

In order to execute the DAG tasks onto the NoC, we allocate the tasks on the platform at the design time following a heurstic. Thus, the mapping schema implies to allocate the computing sub-tasks onto processing engines PE_i while we reserve the VCs routers to the communications between sub-tasks. We use in this paper the classical bin-packing heuristics Best-Fit (BF) and Worst-Fit (WF) for the allocation as disclosed in Algorithm 1.

It starts by sorting tasks according to order, that is either by deadline, or utilization increasingly (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 4.2 and Theorem 2). 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 4.2.

This procedure allows our algorithm to convert a complex allocation problem to multiple single-processor schedulability problem, for which well-known techniques are available in the literature of real-time systems (Lines 23-27). If schedulability fails in a tile, the algorithm aborts on fail, otherwise, it returns the indication of schedulability success in which every sub-task meets its deadline. We simply resume the task allocation worklflow by Figure 4 as we can distinguish along the flowchat each step to make the schedulibility analysis much simpler with less complexity.

Algorithm 1 Bin-packing allocation

```
1: input: Γ: set of tasks, alloc : BF or WF, order : DL or U
2: output: schedulability test
3: sort_tasks_by(order)
4: for (τ ∈ Γ) do
5:     for (v ∈ τ) do
6:         eligible_list = select_eligible_tiles(v)
7:         sort_tiles(alloc, eligible_list)
8:         allocated = false
9:         for (p ∈ eligible_list) do
10:            if ((u(v) + U(Γ_p) ≤ 1) then
11:                add_sub − task_to_taskset(v, Γ_p)
12:                allocated = true
13:            end if
14:        end for
15:        if (allocated == false) then
16:            return FAIL
17:        end if
18:    end for
19: end for
20: for (τ ∈ Γ) do
21:    assign_deadlines_and_offsets(τ)
22: end for
23: for (p ∈ P) do
24:    if (check_schedulability(p) == FAIL) then
25:        return FAIL
26:    end if
27: end for
28: return Success
```

We use in this paper the TDMA arbitration onto NoC routers, as described in Section 3.1. This aims at providing a predictable communication time, in which the latency is known and calculated following Equation (2) [22] regardless the impact of concurrent communication flows. It also provides isolation of Flit forwarding and prevents *miss-behaving* communications to monopolize the network which might lead to resource starvation. However, it requires synchronization mechanisms in routers, and thus, we assume that all routers are synchronous.

$$\mathsf{lat}(VC, v, v') = \frac{L_i}{n_{slot}} \cdot \frac{\Delta}{\eta_i} + H_i \tag{2}$$

Where :

- L_i : number of Flits in the message.

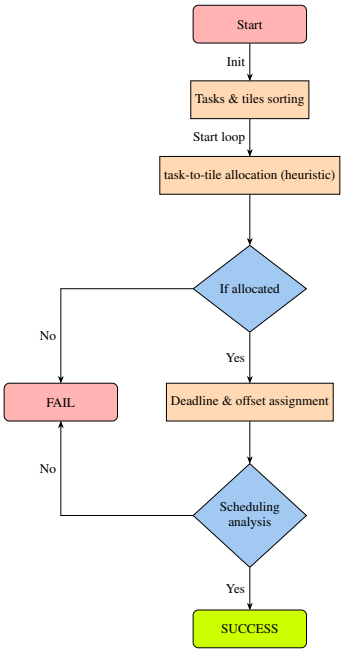

Figure 4: Tasks allocation workflow

- n_{slot} : The amount of data sent in one slot (1 Flit by default) throught Virtual channel (VC).
- Δ / η_i : The total number of slots in a *TDMA cycle* / the assigned slot number.
- H_i : Hop number between v (source sub-task) and v' (destination sub-task).

Once the allocation of every sub-task is achieved, our algorithm computes all the communications costs related to the task set. However, the schedulability analysis involves the use of complex methods since the sub-tasks remain correlated and tightly-coupled. In the following, We describe how to provide isolation between sub-tasks and how the schedulability is checked for each sub-task by the mean of deadline and offset assignment.

4.2 Deadline & offset assignment

Many authors have proposed techniques to assign intermediate deadlines and offsets to DAG tasks. In this paper we report the two of the most used techniques, *proportional share* and *fair share*, reported in [23].

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.

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

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

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:

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

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

$$\mathsf{calculate\_share}(v, \pi) = \frac{\mathsf{C}(v)}{\mathsf{C}(\pi)} \cdot \mathsf{Sl}(\pi, \mathsf{D}) \tag{4}$$

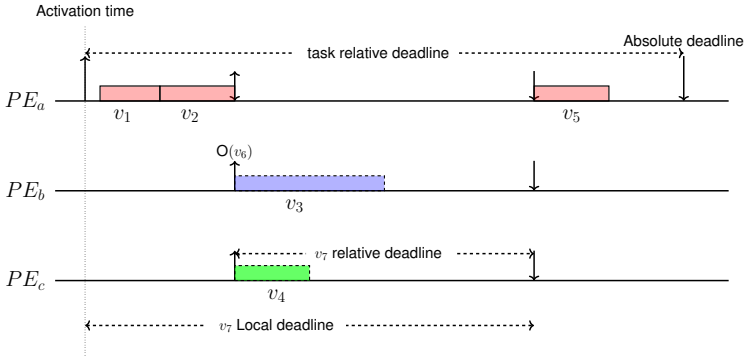

Figure 5: Example of offset and local deadline

Figure 5 illustrates the relationship between the activation times, the intermediate offsets, relative deadlines and local deadlines of the sub-tasks of the task depicted in Figure 3. 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 4.1. *Sub-task $v_j \in \mathcal{V}$ is feasible if and only if for a given task τ arrived at a_i, the sub-task v_j it's executed in the interval bounded by its arrival time $a_j = a_i + \mathsf{O_i}$ and its absolute deadline $a_j + \mathsf{D_j}$.*

Lemma 1. *A task is feasible if all its sub-tasks are feasible.*
Proof. *By the definition, the local deadline d_l of the virtual sink sub-task is equal to the deadline D of the task. Moreover, the sub-task offset cannot be shorter than the local deadline of the previous sub-task. Therefore (i) the precedence constraints are respected (ii) if the sink sub-task is feasible then the task is feasible.*

Definition 4.2. *Let p a processing engine, v a sub-task of a task τ and $\Pi(\tau)$ a set of processors allocated to τ.*

p is an illegible processor for v if:

$\forall_\pi \in \Pi(\tau)$ such that $v \in \pi \Rightarrow \exists \mathsf{VC} \in p$ that can be allocated to v and satisfies the condition $\mathsf{Sl}(\pi, \mathsf{D}) \geqslant 0$

Theorem 2. *Let p a processor and v a sub-task.*
if p is not an illegible processor to v, then v can not be feasible on p.
Proof. *we perform the proof by counter example. Let assume that p is not illegible to v and the system is schedulable. By negating the Definition 2, it exists at least one path where $\mathsf{Sl}(\pi, \mathsf{D}) < 0$. Therefore, one or more sub-tasks will continue their execution over their deadline, thus missing their deadline and Lemma 1 cannot be satisfied.*

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

4.3 Single core schedulability analysis

We use the *Earliest Deadline First* scheduling policy to execute the task on the processing engine. It has been demonstrated that EDF is an optimal policy for task scheduling on single core. EDF schedulability can be checked using workload requirement using the schedulability test proposed in [24]. This test has been extended for tasks experiencing offsets, as follows:

$$dbf(\tau, t) = \max_{v \in \tau} \sum_{v' \in \tau} \lfloor \frac{t - \Theta(v') - D(v') + T(\tau)}{T(\tau)} \rfloor \tag{5}$$

Memory sub-controllers	Routers
$\mathsf{A}^m(1,1)$	$R(1,1)$, $R(1,2)$
$\mathsf{A}^m(2,1)$	$R(2,1)$, $R(2,2)$
$\mathsf{A}^m(3,1)$	$R(3,1)$
$\mathsf{A}^m(1,2)$	$R(1,3)$
$\mathsf{A}^m(2,2)$	$R(2,3)$
$\mathsf{A}^m(3,2)$	$R(3,2)$, $R(3,3)$

Table 2: Routing table to define the corresponding memory sub-controllers for requests coming from or aiming a router

where:

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

Thus, through this approach, we converted the task and communication allocation problem to a single core analysis issue.

5 Off-chip Memory-aware Mapping

We consider memory sub-tasks as routines that interact with the main memory without any compute capacities. Therefore, we propose a static allocation on which a memory sub-task triggered by tile $\mathsf{A}(x, y)$ is allowed exclusively to perform its memory access through a unique dedicated memory sub-controller defined offline.

We use the classical bin-packing heuristics as mentioned on Section 4. In the following, we propose a new algorithm that wraps Algorithm 1 and includes also, the memory sub-tasks allocation. Algorithm 2 starts by browsing every sub-tasks on the task set and check sub-task type (Line 5). Afterward, following the sub-task type (either computing or memory), it allocates the memory sub-task on a router according to a pre-defined allocation schema(Line 6), otherwise, the computing sub-task is allocated on a computing engine according to Algorithm 1.

Algorithm 2 compute and memory sub-tasks allocation

```
 1: input: Γ: set of tasks, r_tab: routing table, alloc : BF or WF, order : DL or U
 2: output: schedulability test
 3: for (τ ∈ Γ) do
 4:     for (v ∈ τ) do
 5:         if is_memory(v) then
 6:             allocate_on_mem_sub_ctrl(v, r_tab)
 7:         else
 8:             allocation(v)                                   ▷ Algorithm 1
 9:         end if
10:     end for
11: end for
12: for (τ ∈ Γ) do
13:     assign_deadlines_and_offsets(τ)
14: end for
15: for (p ∈ P) do
16:     if (check_schedulability(p) == FAIL) then
17:         return FAIL
18:     end if
19: end for
20: return Success
```

6 Experiments & Results

In this section, we describe our experimental protocol and discuss the results issued from our synthetic experiments.

The code has been executed on a regular laptop with Intel Core i5-7200U processor (2×2.5 GHz) and 8 GB of ram. All simulations are carried out by the same hardware platform description: a 3×3 2D-Mesh NoC with synchronous routers that contain 6 VC at each *input port*. Also, we assign the TDMA configuration by the following quantum slot

assignment array $[4, 2, 3, 5, 3, 3]$. Each simulation scenario includes a task set of DAGs generated by TGFF tool [25]. Afterward, we analyze the task set schedulability by our own analysis tool developed in *Python*. We disclose in Table 2 which memory sub-controllers is responsible to handle memory requests coming from the inner-NoC routers.

We use UUnifast algorithm to generate a set of utilization factors U_i that suits with the number of DAG tasks, and afterward, each U_i assgnined to a DAG task is shared among sub-tasks, and we produce several scenarios by varying the utilization factor. To avoid untractable hyper-periods, the period of every task is randomly generated from a list of values between the interval of 1000 and 100000 by step of 1000. Communications workload are flit-based quantified, i.e; for each communication, we assign a random number of flit in the range of 10 and 40. Then, We perform separately two simulations with the task set following the two models: (i) task allocation while considering only the on-chip communications (ii) task allocation when both the on-chip and the off-chip communications matter.

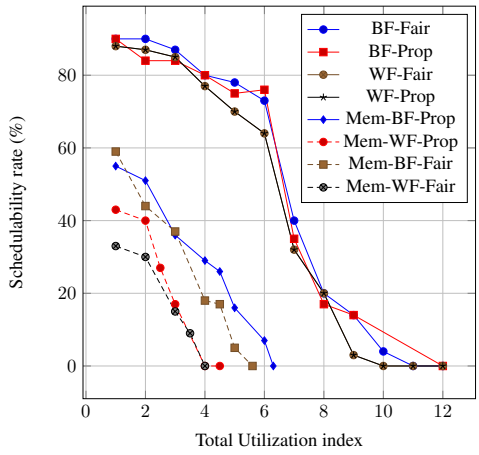

Figure 6: Schedulability success of task allocation strategies with and main memory access footprint

We report in Figure 6 the efficiency of each bin-packing mapping strategy coupled with a deadline-assignment technique. We measure the effectiveness of each method by the average of schedulable task set rate issued from 60 experiences, and applied for a range of utilization factor. Thus, the calculations use the latency formula and schedulability analysis presented in Section 4 as well as additional parameters detailed in Table 3 to calculate the communications cost in seconds with the DRAM.

Network-on-chip		External memory	
size($\mathcal{P}$)	32 Flits	N_{mem}^{req}	1 cycle
n_{slot}	1 Flit	t_{trans}	1 cycle
Freq	600MHz	Freq	200/800 MHz

Table 3: Hardware parameters setting

The simulation results are presented as follows. We denote the results with memory and computing sub-tasks by the prefix Mem-XX as they include DRAM accesses, otherwise, the experiences concern only the allocation of computing tasks and on-chip communications. The Best-fit and Worst-fit heuristics are denoted by (BF) and (WF) as well as Proportional and Fair distribution by (Prop) and (Fair) respectively. We produce similar experiences for both models – considering the same task and hardware model and varying the utilization factor as well. The results are shown in Figure 6 and we observe that the experiences with BF heuristic dominate those performed by WF, which is explained by the fact that BF tries to pack the maximum of tasks on a core. Thus, the inter-sub-task communications are produced inside the core without using the network, and therefore, the latency of communication is drastically reduced. On the contrary, WF dispatch equally the tasks among cores which may create multi-hop communications with congestions while providing high latencies.

We notice also that the experiences with off-chip communications provide a low schedulability success and are outperformed by the former experiences. This is mainly due to the latency of data copy from DRAM to local memory, as DRAM is very slow memory compared to on-chip memory. Moreover, Figure 7 reports the difference between DRAM technologies, the DDDR-SDRAM and DDDR3-SDRAM clocked by 200 MHz and 800 MHz respectively. We notice that the more the main memory is highly clocked the more data are served fast to the communication bus.

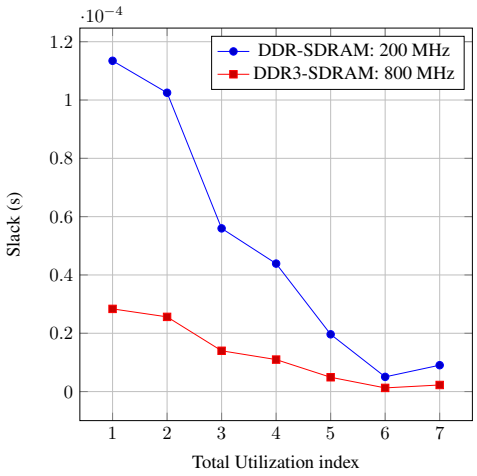

Figure 7: Slack time considering different DRAM technologies

7 Conclusion

In this paper, we provided a support for DAG tasks onto NoC-based manycores architecure. Our apporach converts a complex task and communication allocation problem to a set of classical single core scheduling problem, for which efficient algorithms exist, while preseving timing properties. We used bin-packing heuristics to allocate tasks on cores as as well as on-chip communications and communications toward the main memory from the NoC.

As future work, we are considering extending the model to include the main memory as an important part. We would like also to investigate exact solutions for budgeting VCs and using more sophisticated heuristics for task and communication allocation.

Acknowledgement

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] Chawki Benchehida, Mohammed Kamel Benhaoua, Houssam-Eddine Zahaf, and Giuseppe Lipari. An analysis and simulation tool of real-time communications in on-chip networks: A comparative study. *SIGBED Rev.*, 17(1):5–11, July 2020.

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

[3] Andreas Hansson, Kees Goossens, and Andrei Rădulescu. 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.

[4] Pradip Kumar Sahu, Tapan Shah, Kanchan Manna, and Santanu 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.

[5] Krishnan Srinivasan and Karam 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.

[6] 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, 2019.

[7] Houssam-Eddine ZAHAF, Giuseppe Lipari, Smail Niar, et al. Preemption-aware allocation, deadline assignment for conditional dags on partitioned edf. In *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–10. IEEE, 2020.

[8] José Carlos Fonseca, Vincent Nélis, Gurulingesh Raravi, and Luís Miguel Pinho. A multi-dag model for real-time parallel applications with conditional execution. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1925–1932, 2015.

[9] Ashikahmed Bhuiyan, Zhishan Guo, Abusayeed Saifullah, Nan Guan, and Haoyi Xiong. Energy-efficient real-time scheduling of dag tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(5):1–25, 2018.

[10] Zheng Shi and Alan Burns. Schedulability analysis and task mapping for real-time on-chip communication. *Real-Time Systems*, 46(3):360–385, December 2010.

[11] Tim Harde, Matthias Freier, Georg von der Brüggen, and Jian-Jia Chen. Configurations and optimizations of tdma schedules for periodic packet communication on networks on chip. In *RTNS*, pages 202–212, 2018.

[12] Borislav Nikolic, Robin Hofmann, and Rolf 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.

[13] Salma Hesham, Jens Rettkowski, Diana Goehringer, and Mohamed A. Abd El Ghany. Survey on Real-Time Networks-on-Chip. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1500–1517, May 2017.

[14] Wooyoung Jang and David Z. Pan. Application-Aware NoC Design for Efficient SDRAM Access. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(10):1521–1533, October 2011.

[15] Masoud Daneshtalab, Masoumeh Ebrahimi, Pasi Liljeberg, Juha Plosila, and Hannu Tenhunen. A Low-Latency and Memory-Efficient On-chip Network. In *2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, pages 99–106, Grenoble, France, 2010. IEEE.

[16] Xi Jin, Nan Guan, Qingxu Deng, and Wang Yi. Memory Access Aware Mapping for Networks-on-Chip. In *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 339–348, Toyama, Japan, August 2011. IEEE.

[17] Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, Lothar Thiele, and Benoît Dupont de Dinechin. Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources. *Real-Time Systems*, 52(4):399–449, July 2016.

[18] Manil Dev Gomony, Benny Akesson, and Kees Goossens. Coupling TDM NoC and DRAM controller for cost and performance optimization of real-time systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014*, pages 1–6, Dresden, Germany, 2014. IEEE Conference Publications.

[19] Quentin Perret, Pascal Maurère, Éric Noulard, Claire Pagetti, Pascal Sainrat, and Benoît Triquet. Mapping hard real-time applications on many-core processors. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 235–244, 2016.

[20] Zheng Shi and Alan Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*, pages 161–170. IEEE, 2008.

[21] Quentin Perret, Pascal Maurère, Éric Noulard, Claire Pagetti, Pascal Sainrat, and Benoît Triquet. Predictable composition of memory accesses on many-core processors. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, TOULOUSE, France, January 2016.

[22] Zhonghai Lu and Axel 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 18–25. IEEE Press, 2007.

[23] Zahaf Houssam-Eddine, Nicola Capodieci, Roberto Cavicchioli, Giuseppe Lipari, and Marko Bertogna. The hpc-dag task model for heterogeneous real-time systems. *IEEE Transactions on Computers*, 2020.

[24] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *[1990] Proceedings 11th Real-Time Systems Symposium*, pages 182–190, Dec 1990.

[25] 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.