A Novel Partitioning Approach for Real-Time Scheduling of Mixed-Criticality Systems

Hayfa Ben Abdallah[1], Hamza Gharsellaoui[1] and Sadok Bouamama[1,2]
[1]*National School of Computer Science (ENSI), University of Manouba, Tunisia*
[2]*Higher Colleges of Technology (HCT), Dubai, U.A.E.*

Keywords: Real-Time System RTSys, Homogeneous Multi-Core Architecture, Tasks Allocation, Clustering, Communication Cost.

Abstract: In real-time system (RTSys), a program is split into small tasks and distributed among several computing elements to minimize the overall system cost. Intrinsically, tasks allocation problem is NP- hard. To overcome this issue, it is necessary to introduce heuristics for generating near optimal solution to the given problem. This paper deals with the problem of dependent and periodic tasks to be assigned to different cores interconnected by a network-on-chip (NoC) in such a way that the load on each Core is almost acceptable. Further, the development of an effective algorithm for allocating 'N' tasks to 'P' cores. The system using task clustering to reduce the Communication Cost on the NoC. Experiment results and simulations demonstrate the efficiency of the proposed approach.

1 INTRODUCTION

The design of embedded real-time systems is developing more and more with the increasing integration of critical functionalities for surveillance applications, particularly in the biomedical, environmental, aeronautics, mobile communication systems and so on (Houssein and Hadi, 2016), (S. Meskina and Z. Li, 2017). The development of these systems must meet various challenges in terms of minimizing energy consumption. Managing such fully autonomous embedded devices requires however to solve various problems related to the quantity of energy available in the battery, to the real-time scheduling of the tasks that must be executed before their deadlines, reconfiguration scenarios, particularly in the case of adding tasks and the communication constraint to be able to ensure the exchange of messages between the processors, while maintaining an acceptable level of service quality to the system treatment.

To deal with this problem, we propose in this work a strategy for placement and scheduling of tasks allowing the execution of real-time applications on an architecture containing homogeneous cores (Gammoudi and Chillet, 2015), a periodic tasks τ_i is characterized by two parameters (Liu and Layland, 2016), its WCET W_i and its period T_i, which are assumed as constants for all task instances (homogeneous multi-core architecture). So to optimize one or more measure(s) of effectiveness: acceptable load on the cores of processors, minimization of cost communication, maximization of system reliability, etc.

A real-time system, denoted RTSys, can be implemented by N dependent and periodic tasks to be assigned by the allocation method to P cores connected by a network on chip (NoC) (Hu and Marculescu, 2005).

According to the requested quality of service (QoS), the tasks may increase or decrease their execution rate to fit the requirements of other concurrent activities, but this will not engender the systems failure.

The heuristic-based scheduling techniques are the most common approaches for task scheduling. These are usually classified into three classes,

* Priority-based scheduling
* Duplication-based scheduling
* Cluster-based scheduling

In priority-based scheduling, priorities are calculated and assigned to the tasks which are then scheduled on the processors according to their priorities.
In duplication based scheduling, while tasks are allocated to a processor, its parent (and predecessor) tasks are duplicated to occupy the idle times of the processor to eliminate the communication delay that occurs

Ben Abdallah, H., Gharsellaoui, H. and Bouamama, S.
A Novel Partitioning Approach for Real-Time Scheduling of Mixed-Criticality Systems.
DOI: 10.5220/0012411200003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 3*, pages 882-889
ISBN: 978-989-758-680-4; ISSN: 2184-433X

when message is passed from the parent tasks to the allotted task.

In cluster-based scheduling, some tasks, that need to communicate among themselves, are grouped together to form a cluster. Clusters are then scheduled on to an available processor. The main problem arises when the number of clusters is greater than the number of processors. This leads to programming the communicating clusters on the same processor and which remains in the nearest processor.

In this paper, a clustering based task allocation algorithm which finds a near optimal solution to the problem is adopted.

The remainder of this article is structured as follows: Section 2 deals with real-time systems and more specifically with types and properties of tasks. In the Section 3, the multicore architecture and its scheduling approaches are presented. In Section 4, we briefly presented our proposed method of tasks allocation. Finally, Section 5, concluded the paper.

2 THE REAL-TIME MIXED-CRITICALITY SYSTEM

Many embedded systems are real-time, i.e. they perform tasks under time constraints. Many definitions have been proposed for the real-time systems notion. The most widely used definition is that of Stankovic et al (J.Stankovic, 1988). Thus, in this section, we present many concepts related to the real-time systems from the literature.

2.1 Real-Time Systems Definition

The notion of real-time system RTSys is attached to reactive computer system which must respect time constraints. It depends not only on results of processing carried out but also on temporal aspect "the correctness of system depends not only on logical results of computation, but also on time at which results are produced" (J.Stankovic, 1988). The real-time system is typically composed of several processes (sequential) provided with temporal constraints. We call this processes tasks. In real-time system to ensure proper functioning of system, it is necessary to provide guarantees on respect time deadlines of tasks that are scheduled. There are real-time applications in field of aeronautics, automotive, telecommunications, robotics, etc.

2.2 Definition of the Real-Time Systems with Mixed Criticality

Nowadays, execution media are more efficient and provide greater computing capacities than before. Indeed, current platforms include several calculation units. Subsequently, material possibilities have greatly increased and energy consumption has become key problem (Burns and Davis, 2017). However, execution of application on medium is no longer profitable because calculation unit is under-used.

It is therefore essential to take advantage of these computing capacities and to minimize waste of energy. Then, some applications with different criticality levels on same execution medium will be integrated and some real-time systems with mixed criticality have been proposed (Alan and Robert, 2017).

2.3 Real-Time Tasks

A real-time system is composed by a set of real-time tasks subject to real-time constraints. Generally, a real-time task is described by temporal properties and three distinguished types.

2.3.1 Definition

A tasks τ_i is a generic entity which has temporal constraints and communication relations. It has time properties that indicate arrival time, execute time and critical time allowed for execution of the task τ_i. Often, precedence relationships can be defined to indicate order in which tasks are executed.

2.3.2 Real-Time Task Temporal Properties

The guarantee of real-time systems is ensured by respecting time constraints when executing application.

Generally, a real-time task is described by temporal properties defined as follows:

1. Ready time or release time "R_i":
   Date on which a task τ_i can start its execution;
2. Worst Case Execution Time "W_i":
   This parameter is considered in majority of jobs as worst case execution time of task τ_i (Computing Time "C_i") on processor which is assigned;
3. Deadline "D_i": Represents the instant at which the execution of the task must be terminated;
4. Latence "L_i": Represents the remaining time before occurrence of start or the Deadline;
5. Period "P_i": Fixed Time interval between successive arrivals of task.

* Wake up date: ↑,
* Date of Deadline: ↓,

The figure 1 below represents temporal properties of tasks.

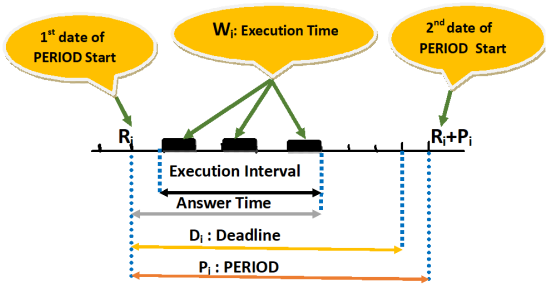

Figure 1: Temporal properties of real-time task.

2.3.3 Tasks Types

The representation of real-time applications in form of tasks models operation of these applications. In the latter, we distinguish tasks characterized by law of arrival which defines their distribution on time and dates creation of their instances. Tasks executing on processor platform are types of recurring tasks, generally periodic, and non-recurring tasks, aperiodic tasks or sporadic tasks.

Periodic Task. The periodic tasks τ_i represent recurring tasks whose successive activations are separated by constant period. They are generally characterized by strict deadlines and constitute majority of tasks composing real-time application.

According to (M.Alabau and Dechaize, 1991), a periodic task τ_i is modeled by the four temporal parameters: (R_i, W_i, D_i, P_i).

Aperiodic Task. In this case, activation date of aperiodic tasks τ_{a_i} cannot be anticipated, since its execution is determined by occurrence of an internal event (for example the arrival of message) or external (for example operator requests). Such task is often characterized by relative deadlines.

Then τ_{a_i} is modeled by single time parameter: W_i its worst case execution time.

Sporadic Task. Sporadic τ_{s_i} tasks are a combination of periodic and aperiodic task, where execution time is aperiodic but execution rate is periodic in nature, these tasks are generally associated with deadline and represent minimum separation between two consecutive instances.

Then τ_{s_i} is modeled in general case by: $(W_i, D_i, T_{i_{min}})$ where $T_{i_{min}}$ corresponds to a minimum interval of time separating two successive activations of sporadic tasks and D_i to critical delay.

3 REAL-TIME SCHEDULING ON MULTICORE ARCHITECTURE

This part addresses tasks scheduling on multi-core architectures. We will first start by defining multicore architectures. Then, we present the different scheduling approaches. Placing real-time task on cores means executing it on cores. So this task and all the other tasks that are already allocated on this same core are schedulable of homogeneous multicore platforms according to schedulability condition (see equation (3)) which is specified. In this part, we study some existing approaches in multicore architecture allowing the resolution of the problems related to allocation and scheduling to improve the performance of the system and its quality of service.

In the literature, some works (Zhang, 2012) consider that tasks scheduling in multi-core systems is strongly dependent on tasks allocation. Jia and al (Huang and Raabe, 2011) propose tasks allocation heuristics based on the list by assigning each of them a priority. This list (there is only one list for all the tasks: global approach) determines the order of tasks allocation on different cores of processors. At each step of the heuristic, a task is selected from the list of candidate tasks to be scheduled on core, then the list is updated, this is repeated until the list is empty. This single scheduling strategy is applied to all cores. In other words if dependent tasks are placed on different cores then, the communication cost and the energy consumption are so high. So, the system performance decreases and we obtain a poor quality of service. Indeed; it is necessary to find the best optimal allocation of tasks for scheduling which minimizes the cost of the communication. In our work and based on (Bhardwajn and Kumar, 2013)) works, we propose an heuristic approach to frequently place each Cluster (contains communicating tasks) on the same core (each C_k core is associated with a subset of tasks: hybrid approach) or on the nearest core to reduce the cost of communication and adding the verification constraints on core loads (If more than one Core, the least loaded one be chosen). The load of the existing cores is an additional parameter L proposed to improve the system's performance, feasibility and quality of services.

3.1 Architecture of Multicore Processor

A multi-core processor contains multiple computing units close to each other, they can communicate with each other at high speed through bus.

Generally, increase the number of cores in processor has advantage to execute several tasks at same time, but doing with less energy than other architectures. So multi-core allows more performance with several clocked cores at lower frequency, thus offering more flexibility and energy savings.

3.2 Multicore Scheduling

Scheduling is a mechanism by which cores have access to place or run in order a set of tasks for application defines. The main goal is properly balance use of cores so that tasks are execute out most efficiently. Then we will talk about task scheduler (Joel Goossens, 2016).

3.3 Multicore Scheduling Approach

Multi-core scheduling is a 2 dimensional problem. First, allocation problem (spatial organization) determines for each tasks which core should run on. Second, scheduling problem (temporal organization) defines date and order execution of tasks.

There are two basic types of multi-core strategies: global approaches and partitioning approaches.

3.3.1 Global Approach

The objective is to allocate at each instant tasks to cores P. For this, there is a single queue for all tasks and single scheduling strategy is used which applies the set of cores (see Figure 2). In this case, an essential property is that tasks migration is allowed.

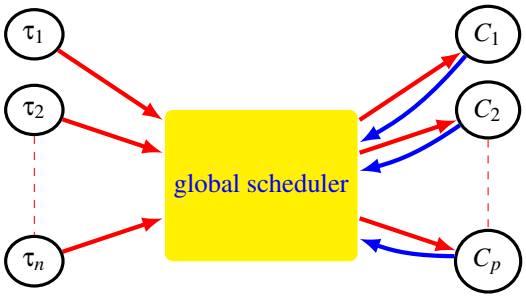

Figure 2: Global approach.

3.3.2 Partitioning Approach

The objective is to definitively assign each task to one core or to define subset of tasks each assigned to core.

It is thus possible to apply single-core strategies on each of the architecture's processors, in which each processor has their own execution queue (see Figure 3).

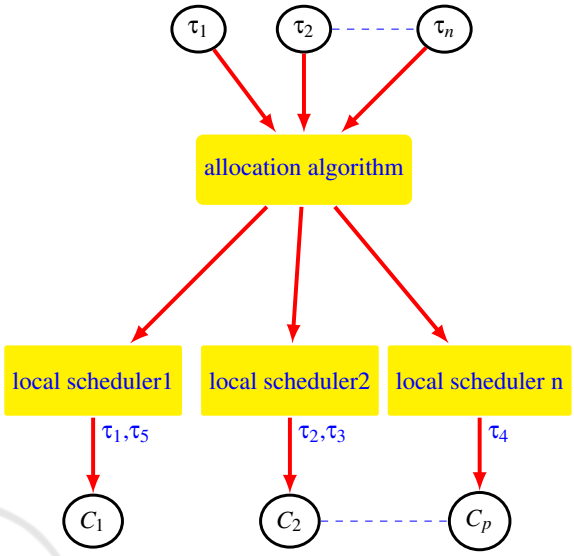

Figure 3: Partitioning Approach.

3.3.3 Hybrid Approach

The hybrid or semi-partitioned approach is derived from the partitioned approach. In this approach, certain occurrences of tasks can be executed on different cores (migration).

Each core receives certain tasks after the partitioning heuristic and can perform an exchange of tasks with other cores, this is called task migration.

It should be noted that for all these approaches, there is a necessary condition of schedulability, (see equation (3)).

In our work, we propose a development strategy based on an hybrid approach because the latter reduces problems of communication cost and energy consumption. So, this approach makes a feasible system with good quality of services.

3.4 Utilization Factor Model Processor / Core

According to Liu and Layland (Liu and Layland, 2016), the rate of the processor utilization by a task τ_i is:

$$\upsilon_{\tau_i} = \frac{w_i}{T_i} \quad (1)$$

Thus, the utilization factor (or processor utilization) is a percentage of time that processor passes to execute all the N tasks that composed the system, and is

worth:

$$U_{c_k} = \sum_{i=1}^{N} \upsilon_{\tau_i} = \sum_{i=1}^{N} \frac{w_i}{T_i}, \forall k \in [1...P]. \quad (2)$$

A set of N periodic tasks on P cores can be scheduled if the previous condition is valid. According to (Jin and Schiavone, 2007),

$$U_{c_k} \leq 1 \quad (3)$$

The equation (3) is a real-time constraint noted later by RTConst which defines the necessary condition for schedulability of N tasks on P Cores.

3.5 Communication Model

Since tasks are dependent and can exchange messages between them, we denote by $M_{i,j}$ the periodic message sent from τ_i to τ_j.

Based on the work of Bach and al (Bach Duy Bui and Caccamo, 2012), each message $M_{i,j}$ is characterized by:

* A period T$M_{i,j}$: The regular time of arrival,
* Time required to send message W$M_{i,j}$,
* Deadline D$M_{i,j}$,
* This is absolute limit time not be exceeded,
* Size S$M_{i,j}$: Relative size of message in bytes,

The Communication between two cores is ensured by a communication medium which can be bus, NoC, etc.

The network-on-chip (NoC) architecture implies a communication cost between each pair of cores C_k and C_l depending on the distance between them. We denote by $Cost_{C_k,C_l}$ this cost.

We model each NoC architecture by a specific cost matrix Cost such as

$$Cost_{C_k,C_l} = \begin{cases} X_{C_k,C_l} & if k \neq l, \forall k,l \in [1...P]. \\ 0 & \text{otherwise.} \end{cases}$$

where X_{C_k,C_l} is a constant giving the cost of transferring one byte from core C_k to core C_l.

Similar to works (Huang and Raabe, 2011), communication Cost between each pair of cores C_k and C_l, it is necessary to multiply the volume of exchanged data by $Cost_{C_k,C_l}$. Then, the total cost of communications is given by:

$$TotalCost = \sum_{k=1}^{P} \sum_{l=1}^{P} \sum_{i=1}^{N} \sum_{j=1}^{N} Cost_{C_k,C_l} * SM_{i,j} \quad (4)$$

4 Proposed Tasks Allocation Method

We present an allocation strategy of tasks on one core based on Clustering concept which groups communicating tasks on the same or the nearest cores. This core must check real-time constraints *RTConst* (see equation (3)).

4.0.1 Architecture Model

Based on (Bhardwajn and Kumar, 2013) work, we assume that each core C_k (k ∈ [1 . . p]) schedules locally assigned tasks with EDF algorithm and increases the number of tasks to be executed. Given that we are considering a multicore platform, we assume also that each task τ_i can be executed on any less loaded core.

In case of homogeneous multicore architecture (identical cores), we not need of allocation matrix for tasks because can be execute of any cores who checks real-time constraint RTConst.

So, to verify that all tasks are allocated on any cores. We define a condition of tasks allocation Z on cores :

$Z_{i,k}$= task τ_i is allocated on core C_k.

$$\sum_{k=1}^{P} Z_{i,k} = 1, \ \forall i \in [1...N] \quad (5)$$

4.0.2 Strategy Principle

The proposed strategy solves tasks scheduling problem. The idea is to group communicating tasks into different subsets called Cluster (denoted CL) while checking the feasibility constraint at the same time. If there is more than one core, the least loaded one will be chosen to improve the system performance and quality of services. we allocate tasks that communicate frequently on the same or the nearest cores following the equation (3). The objective of the proposed strategy is to ensure these following points:

1. Each task must be allocated at only on the core that can execute it,
2. The load of each core must not exceed its maximum load ($U_{c_k} \leq 1, \forall k \in [1...p]$),
3. The period of each task must not exceed its maximum period ($T_{i_{max}}$),

Before describing the strategy steps, it is necessary to define some preliminaries to be used later.

* *Cluster (CL).*
  Group of tasks that exchange data with each other.
  Example: τ_1, τ_2 and τ_3 are grouped in CL_1, (See figure 4).
* *Core Processor (CP).*
  Is the core which can execute all the tasks in the same Cluster such that its utilization rate is less than or equal to 1.
  Example: Looking at Figure 4, we notice that core C_1 can execute all tasks of Cluster CL_1. So, if its utilization rate (supporting all these tasks) is less than or equal to 1, then it is a CP.
* *CPCL.*
  Cluster *CL* which can be schedulable with one or

Table 1: Comparison of communication cost related to tasks increasing.

tasks N	Cores P	Alg.(Bhardwajn and Kumar, 2013)	Alg.(Govil, 2011)	Our.Alg
6	4	106	101	90
7	4	140	135	112
8	4	188	169	135
9	4	243	224	201
10	4	259	251	222
11	4	333	319	290
12	4	378	365	345
13	4	435	401	375
14	4	480	465	450
15	4	518	502	472
16	4	533	522	500
17	4	565	545	523
18	4	588	576	552
19	4	617	591	550
20	4	652	620	592
Total		6035	5786	5409

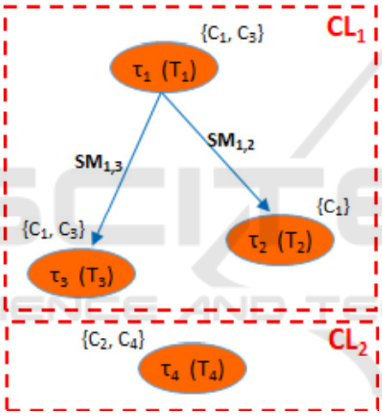

Figure 4: Example of cluster CL.

more Core Processors CP,
Example: If the cluster CL_1 can be scheduled on C_1 ($U_{C_1} \leq 1$), so it is called CP and it becomes a *CPCL* (see figure 4).

* *Traffic Data (TD).*
Amount of data transferred (in bytes) of Cluster *CL* in unit time (ut), (see Figure 4).
Example:

$$TD(CL_1) = ut * (\frac{SM_{1,3}+SM_{1,2}}{T_1}) \text{ and } TD(CL_2) = 0. \quad (6)$$

5 SIMULATION RESULTS

A system RTSys, can be implemented by N dependent and periodic tasks to be assigned to P cores linked by a network-on-chip (NoC) in order to ensure the communication between dependent tasks. Reducing the cost of communications between cores becomes a major concern to ensure high-performance and reliability of such systems.

To show efficiency of proposed strategy, we carried comparisons with allocation algorithms of Poornima and al (Bhardwajn and Kumar, 2013), kapil (Govil, 2011). This section therefore makes possible to situate the proposed strategy in relation to algorithms mentioned above. We propose to implement the three approaches using the same case study presented in section (Bhardwajn and Kumar, 2013).

The objective is to allocate tasks on different cores and calculate the total communication cost of each approach. The 1-D Mesh network (Konstantakopoulos, 2007) communication medium is used to connect the presented cores in the previous example.

5.1 Comparison of Algorithms by Increasing the Number of Tasks

First, the comparison is performed by fixing the number of cores to 4 with a variation number of tasks. So, the table 1 represents the communication cost for each algorithm.

Experimental results on our strategy show that it offers 10.5% ($\frac{6035-5409}{6035} * 100$) communication cost profit compared to algorithm proposed by poornima and al. (Bhardwajn and Kumar, 2013). It offers 6.5% ($\frac{5786-5409}{5786} * 100$) profit compared to algorithm proposed by kapil. (Govil, 2011).

So, Kapil algorithm (Govil, 2011) is an improved extension of the one proposed by poornima and al. (Bhardwajn and Kumar, 2013).

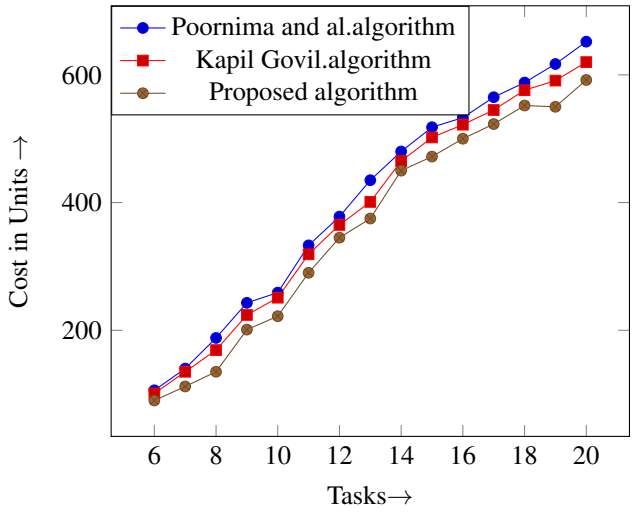

Figure 5: Cost of communication when tasks are in an increasing order and number of cores is 4.

Table 2: Comparison of communication cost when number of cores is in increasing order.

Cores P	Tasks N	Alg.(Bhardwajn and Kumar, 2013)	Alg.(Govil, 2011)	Our. Alg
4	50	645	620	608
5	50	672	640	630
6	50	669	640	625
7	50	678	650	638
8	50	675	649	636
9	50	672	647	635
10	50	682	655	646
11	50	688	666	658
12	50	685	665	659
13	50	685	660	655
14	50	678	658	650
15	50	675	655	645
Total		8104	7805	7685

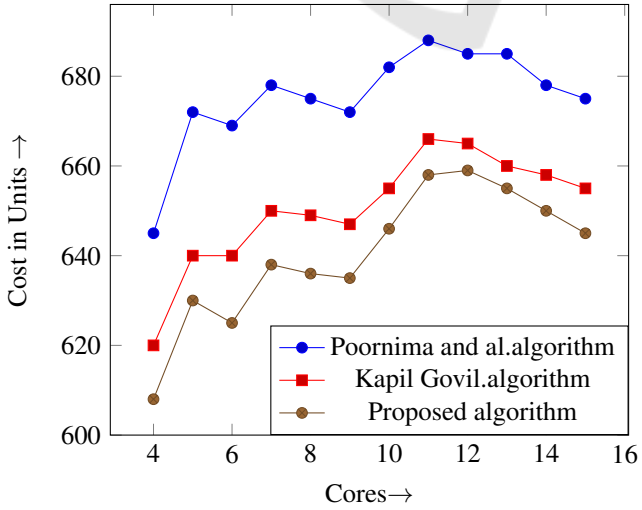

Figure 6: Cost of communication when cores are in an increasing order and number of tasks is 50.

Figure 5 presents curves which compare communication cost in each approach.

5.2 Comparison of Algorithms by Increasing the Number of Cores

Then, we increase the number of cores by fixing the number of tasks to 50. Thus, the table 2 represents the communication cost for each algorithm.

When measuring the profit obtained in terms of communication cost, we find that our strategy offers a little more than 5% $(\frac{8104-7685}{8104} * 100)$ of profit compared to poornima algorithm and 2% $(\frac{7805-7685}{7805} * 100)$ profit compared to that proposed by kapil. Figure 6 presents curves which compare communication cost in each approach.

The simulations show that our strategy is better than those proposed in (Bhardwajn and Kumar, 2013) and (Govil, 2011). This is explained by efficiency of strategy since it offers several heuristics which gives more flexibility to placement level.

5.3 Evaluation

It can be observed from Figure 5 that the values of the total optimal cost obtained by the present algorithm are better compared to those obtained in (Govil, 2011) and (Bhardwajn and Kumar, 2013), in the case, when the number of cores is kept fixed and the number of tasks is taken in an increasing order. The similar observation can also be made from Figure 6 in the case when the number of tasks is fixed and the number of cores is taken in increasing order.

Thus, it is concluded that the present algorithm results have better optimal cost in both cases.

6 CONCLUSION

In this paper, the problem of periodic tasks allocation on a homogeneous multicore architecture using tasks clustering, is discussed. As the task allocation problem is known to be NP-hard.

Our strategy proposes an allocation of tasks which reduces cost of communication between cores and also suggests reducing distance between tasks if the allocation on same core is not possible to obtain the system feasibility.

From the experimental results, we conclude that the proposed solution improves the cost communication in the whole system while keeping its feasibility.

REFERENCES

Alan, B. and Robert, D. (Jan 2017). Mixed criticality systems-a review. *Department of Computer Science, University of York, Technical. Report*, pages 1–69. http://www-users.cs.york.ac.uk/burns/review.pdf.

Bach Duy Bui, R. P. and Caccamo, M. (2012). Real-time scheduling of concurrent transactions in multidomain ring buses. *IEEE Transactions on Computers*, 61(9):1311–1324.

Bhardwajn, P. and Kumar, V. (2013). An effective load balancing task allocation algorithm using task clustering. *International Journal of Computer Applications*.

Burns, A. and Davis, R. (November 2017). A survey of research into mixed criticality systems. *ACM Comput*, 82(37):37–50.

Gammoudi, A. Benzina, M. K. and Chillet, D. (2015). New pack oriented solutions for energy-aware feasible adaptive real-time systems. *Intell. Softw. Methodologies Tools Techn*, page 73–86.

Govil, K. (2011). A smart algorithm for dynamic task allocation for distributed processing environment. *International Journal of Computer Applications*, page 13–19.

Houssein, H. E. and Hadi, M. A. E. (2016). Energy efficient scheduler of aperiodic jobs for real-time embedded systems. In *J. Autom. Comput*, page 1–11.

Hu, J. and Marculescu, R. (2005). Energy-and performance-aware mapping for regular noc architectures. *IEEE Trans. Comput.-Aided Design Integr.Circuits Syst*, 24(4):551–562.

Huang, J. and Raabe, A. (2011). Energy-aware task allocation for network-on-chip based heterogeneous multiprocessor systems. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, page 447–454. IEEE.

Jin, S. and Schiavone, G. (2007). *A performance study of many cores task scheduling algorithms*. London, 2nd edition.

Joel Goossens, E. G. a. L. C.-G. (November 2016). Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms. *Real-Time Systems*, 52(6):808–832.

J.Stankovic (1988). *Misconceptins about real-time computing*. IEEE Computer, London, 2nd edition.

Konstantakopoulos, T. K. (2007). *Energy scalability of on-chip interconnection networks*. PhD thesis, Massachusetts Institute of Technology.

Liu, C. L. and Layland, J. W. (2016). Scheduling algorithms for multiprogramming in a hard-real-time environment. volume 12, page 101–111.

M.Alabau and Dechaize, T. (1991). *Ordonnancement temps-réel par échéance*. London, 2nd edition.

S. Meskina, N. Doggaz, M. K. and Z. Li, M. f. (2017). for smart grids recovery. *EEE Trans. Syst. Man. Cybern. Syst*, 47(7):1284–1300.

Zhang, H. (2012). *ordonnancement de tâches temps réel dans les systèmes multicoeur*. PhD thesis, Université de Nantes.