Adaptive Task Allocation and Scheduling on NoC based Multicore Platforms with Multitasking Processors

Suraj Paul, Navonil Chatterjee, Prasun Ghosal, Jean-Philippe Diguet

Adaptive Task Allocation and Scheduling on NoC based Multicore Platforms with Multitasking Processors

SURAJ PAUL, Indian Institute of Engineering Science and Technology, Shibpur, India
NAVONIL CHATTERJEE, Université Bretagne Sud, Lab-STICC, France
PRASUN GHOSAL, Indian Institute of Engineering Science and Technology, Shibpur, India
JEAN-PHILIPPE DIGUET, CNRS, Lab-STICC, France

The application workloads in modern multicore platforms are becoming increasingly dynamic. It becomes challenging when multiple applications need to be executed in parallel in such systems. Mapping and scheduling of these applications are critical for system performance and energy consumption, especially in Network-on-Chip (NoC) based multicore systems. These systems with multitasking processors offer better opportunity for parallel application execution. Mapping solutions generated at design-time may be inappropriate for dynamic workloads. To improve the utilization of the underlying multicore platform and cope with the dynamism of application workload, often task allocation is carried out dynamically. This paper presents a hybrid task allocation and scheduling strategy which exploits the design-time results at run-time. By considering the multitasking capability of the processors, communication energy and timing characteristics of the tasks, different allocation options are obtained at design-time. During run-time, based on the availability of the platform resources and application requirements, the design-time allocations are adapted for mapping and scheduling of tasks which result in improved run-time performance. Experimental results demonstrate that the proposed approach achieves, on an average 11.5%, 22.3%, 28.6% and 34.6% reduction in communication energy consumption as compared to CAM [18], DEAMS [4], TSMM [38] and CPNN [32], respectively for NoC based multicore platforms with multitasking processors. Also, the deadline satisfaction of the tasks of allocated applications improves on an average by 32.8% when compared with the state-of-the-art dynamic resource allocation approaches.

CCS Concepts: • **Networks → Network on chip**; • **Computer systems organization → Interconnection architectures**; *Real-time systems.*

Additional Key Words and Phrases: Multicore systems, Network-on-Chip, Dynamic resource allocation, Communication energy, Deadline

ACM Reference Format:
Suraj Paul, Navonil Chatterjee, Prasun Ghosal, and Jean-Philippe Diguet. 2019. Adaptive Task Allocation and Scheduling on NoC based Multicore Platforms with Multitasking Processors. *ACM Trans. Embedd. Comput. Syst.* 0, 0, Article 0 (2019), 25 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Advancement in VLSI technology has made it possible for designers to integrate a large number of Processing Elements (PEs), Intellectual Property (IP) cores and memory units (MUs) onto a

Authors' addresses: Suraj Paul, Indian Institute of Engineering Science and Technology, Shibpur, Howrah, West Bengal, India, csuraj.ece@gmail.com; Navonil Chatterjee, chatterjee.navonil@univ-ubs.fr, Université Bretagne Sud, Lab-STICC, Lorient, France; Prasun Ghosal, p_ghosal@it.iiests.ac.in, Indian Institute of Engineering Science and Technology, Shibpur, Howrah, West Bengal, India; Jean-Philippe Diguet, jean-philippe.diguet@univ-ubs.fr, CNRS, Lab-STICC, Lorient, France.

1539-9087/2019/0-ART0 $15.00
https://doi.org/10.1145/1122445.1122456

single chip, resulting in multicore based embedded systems. These systems provide increased parallelism which demand fast communication infrastructure to fulfill the inter-core communication requirements. Network-on-Chip (NoC) paradigm provides the necessary scalable and efficient communication infrastructure between multiple processing cores [7]. NoC based multicore platforms are emerging in implementation of various application domains such as cloud computing, automotive, avionic applications and multimedia and NoC-based FPGA devices are now available [37]. Allocation of the executable tasks of applications onto the available resources is crucial as it affects the system performance. The complexity of the problem increases in multicore systems when multiple applications are executed in parallel to maximize the use of system resources.

The communication dependencies between the tasks and the spatial location of the PEs on a target multicore system are key factors affecting the allocation decisions. For real-time applications, the validity of results not only depends on the logical correctness but also on the time of obtaining the output. Thus, it is important to consider their timing constraints while performing task assignments. As most of the embedded systems are battery operated [13], reducing the energy consumption prolongs the duration of operation of the system. Energy consumed during the exchange of data packets on a NoC based platform is affected by the task allocation decisions such as contiguity of allocated PEs and placement of communicating tasks on PEs located in close vicinity. NoC based multicore systems, executing afore-mentioned embedded applications, exhibit time-varying workloads on the given platform. These variations cannot be predicted accurately during design-time. Such scenarios can occur when applications arrive or depart during run-time or user-driven requests[5]. Offline allocation methods produce sub-optimal solutions as they are unaware of online variations of application workloads. Thus, task allocation strategies that can take into account the varying run-time workload on NoC based multicore systems are essential for embedded applications.

Typically, when the set of applications is known, task assignment and scheduling problem is solved statically (referred to as design-time task allocation). These algorithms typically use intensive search space exploration methods to obtain the optimal mapping for the given applications. However, these approaches are computationally intensive and cannot cope with dynamic application behavior (workload variation) in which different combinations of applications can be executed concurrently over time. Thus, dynamic task allocation techniques (referred to as run-time task allocation) for embedded applications are essential, which take into account the varying workload on NoC based multicore systems.

Task allocation at run-time can be performed either with or without pre-computed task mapping results. Several efficient online heuristics have been proposed for assigning tasks of new applications submitted during run-time for on-the-fly allocation [4][5][32]. But, due to the availability of limited computation time at run-time, task schedulability and optimal mapping may not always be ensured by these heuristics. To overcome the shortcomings mentioned above of static and dynamic task allocation algorithms, hybrid task allocation strategies have become increasingly popular in recent years [19][25][31]. Typically, in this kind of allocation policy, multiple potential allocation solutions are obtained at design time. One of these pre-determined solutions is then selected and applied at run-time depending on the system state. This strategy helps in accomplishing efficient run-time mapping/scheduling decisions. However, most of the existing hybrid mapping approaches reported in literature allocate a single task per processor [19][26][31][39]. These techniques are inadequate for applying in systems where PEs execute multitask operating systems. In such systems, each PE can support multiple tasks based on the amount of its memory [18][32]. Few of the hybrid strategies [30][24] for run-time task assignments target multitasking platform but solve task mapping problem. The challenges of energy-awareness and task scheduling are not addressed which renders these approaches insufficient for allocation of real-time tasks. Thus, an improved hybrid strategy is essential for dynamic task assignment and scheduling of real-time applications targeting such multicore systems. In these systems, considering multiple tasks per PE allocation strategy can potentially result in better utilization of the available computing resources [24]. Also, such an allocation approach facilitates the design of a light-weight run-time platform manager.

On a multicore system with multitasking PEs, each application may have multiple configurations with different usage of platform resources and different performance costs (timing response and energy consumption). The challenges of assigning multiple tasks per PE with a global scheduler on NoC based multicore system involve exploring the spatial distribution of the subset of the PEs chosen for task execution and determining the allocation of tasks within the set of identified PEs with a constraint on energy consumption and deadline satisfaction. The solution space consisting of PEs arranged in regions (contiguous and non-contiguous) is larger in comparison to multicore systems supporting single task assignment per processor. Thus, an efficient approach considering the multitasking capability of the underlying multicore platform is essential for hybrid mapping and scheduling.

In this work, we propose an improved hybrid task allocation and scheduling approach for NoC based multitasking multicore systems to cater to application dynamism at run-time and obtain an improved task allocation for real-time applications. First, as part of the design-time exploration, a set of potential operating conditions are generated for a given set of target applications. It consists of possible shapes of allocation regions of PEs and mapping/scheduling of tasks within these regions. Next, at run-time, this primer allocation information is utilized to determine a suitable allocation configuration of the incoming applications based on the current state of the system. Towards this end, we introduce an improved online algorithm for selecting and adapting the design-time allocation solutions dynamically while meeting the deadline of the tasks and reducing the communication energy consumption. Thus, by leveraging the pre-computed task mapping choices for the target platform, the proposed strategy offers fast online decisions for run-time resource allocation and scheduling for multitasking multicore systems.

The novel contributions of this article are summarized as follows:

- Proposition of an improved methodology for region shape generation for task allocation on a multitasking NoC based multicore platform.
- Augmenting Particle Swarm Optimization (PSO) formulation for task allocation and scheduling considering multitask assignment on processors in a given multicore platform.
- Presenting a dynamic strategy for selection and placement of the set of pre-determined allocation configurations at run-time for satisfying the deadline of the tasks and reducing the energy consumption.
- Evaluation of the proposed approach, demonstrating its effectiveness in terms of communication energy consumption and deadline satisfaction of the allocated applications.

The remainder of this article is organized as follows: Section 2 describes the related works in hybrid task allocation techniques. Section 3 gives prerequisites and problem definition for this work: section 4 details the proposed hybrid task allocation and scheduling algorithm. The test setup and the simulation results are discussed in Section 5. Section 6 concludes.

2 LITERATURE REVIEW

This section presents a review of literature in the domain of dynamic task mapping and scheduling. Algorithms can perform run-time task allocation on-the-fly, i.e., tasks are mapped as and when an application or a set of applications is submitted for execution by the user. It can also be performed at run-time using hybrid task allocation techniques.

2.1 On-the-fly task allocation techniques

In [5][6], a run-time mapping policy incorporating the user behavior in allocation decisions is presented. In [1], dynamic task allocation heuristics are proposed considering different factors such as min/max channel load, average channel load and path load. A dynamic mapping approach employing task assignment in a spiral fashion at run-time has been presented in DSM [20]. It attempts to reduce the time for task communication, reconfiguration and task migration. A run-time mapping approach using distributed agents is mentioned in [9]. In [12], a run-time task allocation scheme for minimizing the internal and external congestion has been proposed using contiguous neighborhood allocation (CoNA) algorithm. The impact of selecting the first node on the quality of mapping obtained by the CoNA algorithm is investigated in [10]. It shows that a square shape is preferred for processor allocation for applications with dependent tasks. A contiguity adjustable square allocation (CASqA) algorithm for run-time task allocation has been proposed in [11]. It provides the flexibility to adjust the contiguity of the allocated processors by expanding the square search region for dynamic task allocation. The afore-mentioned approaches are aimed for single task allocation per processor and are unsuitable for run-time task assignment on multitasking multicore platforms where PEs can support multiple tasks. Furthermore, these techniques solve the task allocation problem and ignore the timing characteristics of task rendering them insufficient for real-time applications. Few works in literature deal with dynamic task mapping on multicore platforms with multitasking processors. Here, multiple tasks can be supported on processors up to to a maximum limit determined by the amount of its available memory. In [32], communicating tasks are mapped in close proximity of each other in a compact manner using a packing strategy. However, the execution time requirement of the tasks is not considered. Consequently, with an increase in the degree of task graph, more tasks get mapped onto the same processor, causing deadline misses. Authors in [18] extend the work presented in [32] by incorporating task migration based mapping improvement policy to shift the tasks from heavy loaded processors to appropriate processors. A run-time unified mapping and scheduling technique presented in [4] reduces the communication energy consumption of the mapped application while meeting the task deadline. All the above on-the-fly mapping approaches focus on fast heuristics for task assignment to take quick on-line decisions as the computation performed by the algorithms may add to the application execution time. Consequently, the quality of the mapping solution obtained by the above methods may be low.

2.2 Hybrid task allocation techniques

Hybrid mapping techniques have been proposed in the literature to remove the bottleneck of on-the-fly heuristics for run-time task allocation. These approaches exploit pre-computed results obtained offline to perform dynamic task mapping. Numerous efforts have been reported for hybrid mapping techniques. An online execution trace analysis for multimedia applications is proposed for run-time allocation in [31]. It rapidly identifies the mapping for optimizing the throughput, resource usage and energy consumption of the executing applications. [39] proposes an automated design-time exploration engine to enable dynamic application configuration as per the available resources using a priority-based run-time heuristic. Authors in [19] propose a framework for design space exploration for resource management by software reconfiguration on an industrial multicore platform. By a combination of design time and run-time methods, an appropriate system configuration point is selected at run-time which helps to meet the QoS constraints imposed by the user. [36] proposes a scenario-based run-time mapping strategy for homogeneous platforms. In this work, the optimal mappings for inter-application scenarios are explored at design-time and serve as the basis for run-time decision making. Authors in [24] present a model-based framework for run-time system adaptability. The proposed dynamic mapping heuristic aims at run-time power reduction considering multitask allocation and cost of already mapped communicating tasks. The experimental results show that the multitask approach lowers energy consumption. However, this work is unsuitable for real-time task allocation. A hybrid task mapping approach for MPSoCs which focuses on inter-application and intra-application dynamic behavior has been proposed in [25]. During design-time, optimal solutions are obtained considering throughput maximization with or without a predefined energy budget. At run-time, by using the design-time results, it achieves performance improvement and energy savings for multiple applications simultaneously active. In the hybrid mapping approach presented in [30], multiple design points indicating throughput and energy consumption at different resource combinations are explored. This technique is utilized in a run-time algorithm to select a suitable design-point depending on available resources and application throughput. [23] describes a move-based algorithm for run-time mapping targeting dataflow actors on the heterogeneous MPSoCs. In [33] describes a hybrid task mapping and scheduling method by using worst case timing and schedule update with actual execution time at run-time. Its main focus is to provide guaranteed latency for real-time applications.

From the above survey, it can be noted that the existing dynamic allocation approaches employing hybrid strategies for run-time task assignments offer better quality allocation decisions. However, these task allocation techniques for NoC based multicore systems largely consider single task allocation per PE. As a result, they are inadequate for applying in multicore systems with multitasking PEs. The reported hybrid task assignment methods mostly solve only the task mapping problem using run-time results due to which these hybrid approaches are insufficient for task allocation of real-time applications. Further, few works address task mapping/scheduling problem for real-time applications on multitasking platforms but do not consider energy-awareness. In this work, we propose an improved strategy for dynamic task allocation using design-time results. It gives a unified mapping and scheduling solution targeting NoC based multicore platforms with multitasking PEs. The design-time stage generates both contiguous and non-contiguous sets of PE regions for task execution for guiding the PE assignment and scheduling decisions at run-time. This helps to handle the dynamism in application arrival and resource availability encountered at run-time while reducing the communication energy consumption of executing applications and satisfying the task deadline.

3 SYSTEM MODEL

We use a multicore computing platform that is modular and configurable for parallel multi-threaded applications. The system model consists of basic processing nodes, interconnected to each other through a NoC communication infrastructure in 2D mesh, as shown in Fig. 1. The NoC based multicore platform provides computation and communication resources to execute multiple applications.

In this work, we assume that each node is made of general-purpose processors that are considered homogeneous. Therefore, the computation time of the task while executing uninterrupted on any PE is identical. A unique id, PE_i identify each processor. Being connected through an NoC infrastructure, each processing node has a specific position described by its (x, y) integer coordinates. The PEs are independent subsystems having a local processor unit, control unit and memory. Similar to [4][18][32], the size of the available memory in a PE determines the maximum number of tasks, PE_{cap} that can be supported. We have assumed a distributed multiprocessor system model and therefore, we do not consider any shared memory. Thus each node can communicate with any other node in the architecture through messages sent through the NoC routers.

A special node called the *Platform Manager* (PM) runs the RTOS and coordinates the platform activity. PM consists of software modules which include Resource Observer (RO) and Task Allocation and Scheduling Unit (TASU). RO notes the status of the PEs such as occupied or idle. When an

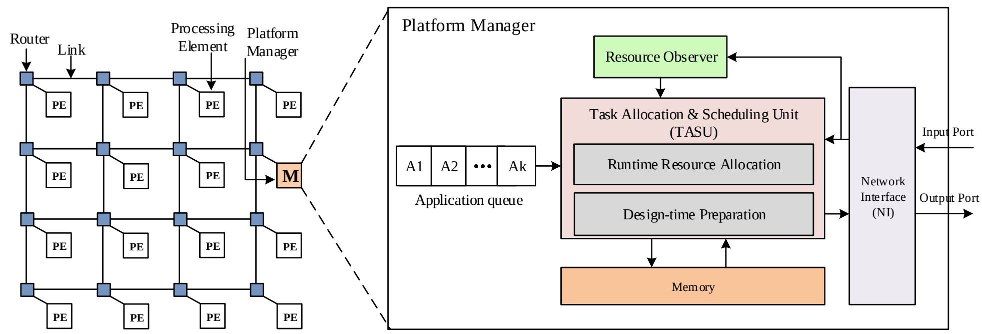


Fig. 1. Overview of the system model

executing task finishes its execution, the PE assigned to it sends its address to the PM to notify their availability which is then updated in RO. TASU runs the proposed adaptive task allocation algorithm. The pre-computed allocation configurations are stored in the offline repository present in memory of PM. These are fetched by TASU at run-time and used to allocate the tasks of each application on a region of PEs. The tasks of applications are dispatched from PM to the selected PEs and executed on these PEs. Since this article deals with algorithm for task allocation and scheduling during design-time/run-time stages on multitasking processors, it is assumed that suitable mechanisms for loading of task code onto the PE memory and contention delay avoidance during packet transfer already exist on the given platform similar to [3][5][21]. We have considered that no incoming application workload is executed in PM to minimize any time overhead imposed on application management. Furthermore, OS supports non-preemptive multitasking and event-based programming environment. It monitors the arrival of the new applications and makes the task allocation decision only when a new event occurs i.e. an application enters or leaves the system. It is assumed that the platform has sufficient resources to allocate the tasks of incoming applications onto the PEs.

4 PROBLEM REPRESENTATION

This section presents the necessary definitions and notations required to represent the problem of dynamic resource allocation for real-time applications using design-time analysis.

Application Task Graph: An application is represented as a directed acyclic graph $G = (\Gamma, \mathcal{E})$. Γ denotes the set of tasks associated with the application.

A task $\tau_i \in \Gamma$ has following parameters: (ex_i, dl_i). ex_i is considered as worst-case execution time (WCET) taken by τ_i which remains fixed while executing on a PE. dl_i is the deadline of the task which is considered with respect to the time of arrival of the application. In addition, we denote the slack-time associated with the task by sl_i. It refers to the difference between the time at which a task would complete if it started now and its deadline time. sl_i is given by Eq. 1.

$$sl_i = dl_i - (current\ time + remaining\ task\ execution\ time) \tag{1}$$

Being a dynamic attribute of a task, sl_i depends on the run-time situation. The results obtained on completion of task execution is transmitted as encapsulated messages. In this work, we have considered soft real-time tasks.

$\mathcal{E}$ is the set of directed edges $e_{kl} = 1 \leq k, l \leq |\Gamma|$ representing the communication between the tasks τ_k and τ_l. The communication volume between task τ_k and τ_l is denoted by the edge weight

w_{kl}. A given task, except the root task, can begin its execution on a PE when it is available and the required data from all its parent tasks reaches the PE.

Topology Graph: The topology graph is an undirected graph $\mathcal{H} = (P, L)$. P representing the set of processors present in the NoC topology. Each $PE_j \in P$ is attached to the on-chip network through a router. The bidirectional communication link connecting the routers for PE_j and PE_k is represented by an edge $l_{jk} \in L$.

Resource Assignment:The *resource assignment* χ for a given application G on a finite set of processors is defined by the function pair $\mathcal{R} = (map, start)$. $map()$ and $start()$ indicate the spatial and temporal assignment of the tasks of G onto a set of PEs of the multicore platform.

Spatial Allocation: The spatial allocation of application G on the set of processors P is represented by the function map: $\Gamma \longmapsto P$. $map(\tau_k) = PE_j$ indicates that task τ_k is allocated to $PE_j \in P$ for execution.

Temporal Allocation: The starting time of the task is given by its temporal allocation. This involves obtaining the schedule of tasks present in application G on its allocated processor. This is determined by the function $start : \Gamma \longmapsto \mathbb{Q}_+$ which gives the start time of tasks. Thus, $start(\tau_k)$ represents the time at which task τ_k begins execution on a PE given by $map(\tau_k)$.

If $finish()$ represents the time at which task completes its execution, then

$$finish(\tau_k) = start(\tau_k) + ex_k \tag{2}$$

If $finish(\tau_k) \leq dl_k$, then τ_k meets its deadline. For a multitasking PE PE_k, multiple tasks may be assigned to it. While scheduling a task τ_i, the earliest time at which the particular processor is available for its execution is given by the function EAT (earliest available time) defined as:

$$EAT(PE_k) = \max_{\forall \tau_i \in \Gamma | PE_k = map(\tau_i)} finish(\tau_i) \tag{3}$$

Communication Energy: Communication energy is defined as the amount of energy consumed for transferring data packets from the source to the destination PE. It depends upon (i) distance between the processors executing the communicating tasks (ii) data volume and (iii) energy required to transfer a single bit through the NoC. Similar to existing works [3][5][32], we assume constant energy is consumed in one bit transmission. Let E_r and E_l be the energy consumption for one bit transmission in a router and a link, respectively. The number of routers N_r and links N_l traversed in the communication path is calculated using the hop count between the source-destination pair. Communication energy, E_{comm}, in uJ for allocated applications on a given NoC platform is estimated:

$$\begin{aligned} E_{comm} &= \sum_{\forall G_i} \sum_{\forall e_{ij} \in G_i} (E_r * N_r + E_l * N_l) * w_{ij} \\ N_r &= HC(map(t_i), map(t_j)) + 1 \\ N_l &= HC(map(t_i), map(t_j)) \end{aligned} \tag{4}$$

where, w_{ij} is the data volume between tasks t_i and t_j in megabits. It is the total data that needs to be transferred between the tasks in packetized form where each packet consists of various flits. HC denotes the hop-counts.

Operating Condition: Each application has a specific operating condition which consists of its allocation policy, used resources and energy consumption. A suitable operating condition for an application is essential for correct application behavior during run-time such that all the tasks in a given application satisfy their respective deadline. We define the operating condition, C_G of application G with the following tuple:

$$C_G = \left\langle \psi, \rho, \mathcal{S}_\rho, E_{comm}, G^{ct} \right\rangle \tag{5}$$

An application, G can have two types of *allocation mode* denoted by $\psi = EA$ or FA, where EA represents *communication energy aware* mode in which tasks of G are assigned to a PE in a given multicore platform such that communication energy of the application is minimized. FA denotes *finish-time aware* allocation mode which aims at reducing the finish-time of the tasks present in application. ρ indicates the number of PEs used for assigning the tasks of application G. In a multicore platform with processors having multitasking capability PE_{cap}, $\rho = R_{min}...R_{max}$ where $R_{min} = |\Gamma|/PE_{cap}$ and $R_{max} = |\Gamma|$. The number of allocated PEs is n where $n \subset \rho$. For a given NoC based multicore platform, these n PEs can have different spatial distribution resulting in different shapes of region of allocation. This is indicated by $\mathcal{S}_\rho$. G^{ct} represents the completion time of execution of the allocated application G.

4.1 Problem Formulation

We formally state the hybrid resource allocation problem addressed in this work as follows.

4.1.1 Design-time sub-problem.

Given the following as inputs:

(1) A set $\mathbb{G}$ of known applications each represented by a directed acyclic graph $G = (\Gamma, \mathcal{E})$.
(2) A target NoC based multicore platform given by $\mathcal{H} = (P, L)$ with each processor having maximum task capacity PE_{cap}.
(3) Timing information of the task τ_i of the applications
  - execution time, $ex_i > 0$
  - deadline, $dl_i \geq ex_i$

Determine the operating conditions for each of these application to find:

- number of PEs ρ used for allocating the tasks in $G(\Gamma, \mathcal{E})$
- shapes of allocation region, $\mathcal{S}_\rho$ for all values of ρ
- $map : \Gamma \longmapsto P \; | PE_{tasks} \leq PE_{cap}$

such that

(1) tasks meet their deadline.
(2) E_{comm} of the application is reduced.

4.1.2 Run-time resource allocation sub-problem.

Given the following as inputs:

(1) A set of arrived applications $Q = \{G_1, G_2...G_n\}$ where $Q \subset \mathbb{G}$
(2) A target NoC platform $\mathcal{H} = (P, L)$ where processors have task capacity PE_{cap}.
(3) A set of operating conditions identified at design-time for each application.

Determine an adaptive resource allocation at run-time to identify the operating conditions for each of the arrived applications:

$$\gamma = \left\langle C_{G_1}, C_{G_2}...C_{G_n} \right\rangle, C_{G_i} \in C_G \wedge 1 \leq i \leq n \tag{6}$$

such that during execution of the arrived application on $\mathcal{H}(P, L)$:

(1) Finish-time of the allocated application G_i^{ct} is reduced $\forall \; G_i \in Q$
(2) Increase in E_{comm} of the allocated applications at run-time is low.
(3) $\sum\limits_{G_i \in \mathbb{G}} C_{G_i}(\rho) \leq |P|$

(4) Overhead of taking run-time mapping and scheduling decision is low.

5 PROPOSED APPROACH

In this section, we describe the proposed allocation strategy for NoC based multicore platform.For mapping and scheduling applications on a multicore platform containing PEs with multitasking capability, a hybrid scheme is adopted. We first derive the allocation templates of individual applications at design-time with a focus on energy consumption and deadline satisfaction of each task. Next, we use an improved run-time heuristic for online adaptation of the design-time identified allocation decisions according to the platform resource availability and application requirements. The details of the proposed design-time and run-time strategy are as described next.

5.1 Design-Time Preparation

In the design-time exploration stage, a set of operating points are determined which consist of allocation templates for the tasks of the application. It consists of various allocations of tasks that infer different scheduling solutions, based on EAT of processors that gives different completion time of each application. As the order and time instant of arrival of applications are unknown the applications sharing the given platform are unavailable at this stage. Thus, for the given set of applications, the allocation decision is done during design-time considering individual application execution. This involves *spatial allocation configuration* of tasks and *allocation template formation* using the allocated PEs. These steps are as follows.

5.1.1 Spatial allocation configuration: In this step, the spatial allocation configuration for tasks of each of the applications is obtained. A configuration is characterized by (i) allocation size, which is the number of the selected PEs for task execution of the given application, and (ii) allocation shape, which is the spatial distribution of the chosen PEs. This is determined by the number of tasks present in each of the applications and the multitasking capability of the PEs. The afore-mentioned information is used to find the various configuration for allocating tasks of applications as described below.

Spatial allocation on a multicore platform involves selecting the allocation region, which consists of a set of PEs for task execution. The region is characterized by the relative position and the number of PEs forming the region. Therefore, these regions can have different shapes and sizes which is required to be determined. Algorithm 1 describes the proposed selective region shape generation (SRSG) approach employed to identify these regions for any given application. First, the algorithm enumerates the possible sizes of each region R. This is done by using the number of tasks N present in the application and the multitask capability PE_{cap} of the given multicore platform. For an application with N tasks the region size varies between N/PE_{cap} to N. Next, for each of these region sizes, the distribution of PEs is determined. Please note that we consider contiguous PE selection for the allocated region. The quality of the allocated region is governed by the average manhattan distance (AMD) between the PEs [12]. A compact region gives smaller AMD value compared to a scattered region of PEs. The proposed algorithm accomplishes the objective mentioned above by selectively growing the allocation region shape. At each intermediate level of the region growth, the set of nodes, S_j satisfying the contiguity constraint is selected. However, this may give rise to identical regions i.e. regions having the same spatial pattern of PEs but with different orientations. For a 2D mesh NoC based multicore platform, rotations in steps of 90° can be used to identify the identical regions. Such regions are excluded from further exploration. This method ensures that the set of PEs determined by Algorithm 1 consists of unique region shapes. Such regions are enlisted in *Shape_list* for the given application and target multicore platform. These steps are repeated for each of the region sizes to find regions of various shapes.

ALGORITHM 1: ***Selective Region Shape Generation***

Input : $G = (\Gamma, \mathcal{E})$; PE_{cap}; AMD
Output: A set of allocation region shapes consisting of different number of PEs

```
Shape_list ← ∅; R ← ∅ ;S_j ← ∅; j=0;
R_max = N; \\N is total number of tasks in G = (Γ, ℰ)
R_min = N/PE_cap;
for i = R_min to R_max do
    while size of generated region ≠ i do
        R ← choose a PE with maximum free neighbour;
        for each adjacent PE location (x,y) of R do
            R` = R ∪ (x, y);
            h = ComputeAMD(R`);
            if h < AMD then
                S_j ← R`; \\include the PE locations in generated shape
            j++;
    for each shape S_j do
        if S_j is identical with shape S_k where j ≠ k then
            remove S_j; \\check and remove identical shapes
        else
            Shape_list ← Shape_list ∪ S_j;\\update the list of generated shapes
    return Shape_list;
```

5.1.2 Allocation template formation: In this step, the regions obtained by spatial allocation process are used to determine the allocation templates i.e. the allocation of the tasks of application onto the PEs constituting these regions. It involves allocation of tasks and their sequence of execution, considering the task precedence constraints and their timing characteristics according to EAT values of PEs (refer Eq. 3). To explore the mapping and scheduling of tasks on the PEs, we use a particle swarm optimization (PSO) based approach, which is popular for tasks allocation [27][14]. In a PSO based method, multiple candidate solutions co-exist and collaborate simultaneously. In this work, we have used a discrete particle swarm optimization (DPSO) to solve the problem of tasks to PE mapping and scheduling. First, we present a brief overview of the DPSO scheme.

Let p_k^i denote the i^{th} particle in its k^{th} iteration. It can be represented in n-dimensional space as $\langle p_{k,1}, p_{k,2} \ldots p_{k,n} \rangle$. Every particle has its corresponding local best solution which the particle has seen over the generations. $gbest_k$ is the best particle present until generation k. Eq. 7 gives the new position of the particle as follows:

$$p_{k+1}^i = (c_1 * I \oplus c_2 * (p_k \longrightarrow pbest^i) \oplus c_3 * (p_k \longrightarrow gbest_k)).p_k^i \tag{7}$$

In this equation, the minimum length sequence of swapping to be applied on elements of a to change it to b is indicated by $a \longrightarrow b$. $\oplus$ sign is the fusion operator. $a \oplus b$ is equal to the sequence in which the sequence of swaps in a is followed by the sequence of swaps in b. c_1, c_2 and c_3 are the inertia, self-confidence and swarm confidence values. The factor $c_i * (a \longrightarrow b)$ means that the swaps in the sequence are applied with probability c_i. The sequence of identity swaps is indicated by I which indicates the inertia of particle to retain its current configuration. To generate p_{k+1}^i from p_k^i, final swap corresponding to $c_1 * I \oplus c_2 * (p_k \longrightarrow pbest^i) \oplus c_3 * (p_k \longrightarrow gbest_k)$ is applied. As reported in [17], DPSO particle system converges to a solution if the following condition is satisfied:

$$(1 - \sqrt{c_1})^2 \leq c_2 + c_3 \leq (1 + \sqrt{c_1})^2 \tag{8}$$

We have experimented with different values of c_1, c_2 and c_3. The results reported in this work correspond to $c_1 = 1, c_2 = 0.5$ and $c_3 = 0.5$. Other values of c_1, c_2 and c_3 affects the convergence rate while giving the same final results. Next, we describe the formulation of the particle to address the mapping and scheduling problem. Input to our formulation is the application task graph and the shape of the PE region determined during the previous step.

Particle representation: For each task present in the application, a PE from the set of PEs constituting the region shape needs to be assigned. In the proposed PSO formulation, a particle is a sequence of integers represented as an array. The length of the particle, representing the array index, is equal to the number of tasks present in the application. Each element of the array indicates the PE id (PE_{id}) considered for executing the task. Fig. 2 illustrates a typical particle formulation for mapping an application with 6 tasks on a platform having 4 PEs with $PE_{cap} = 2$. The length of the particle (array size) is 6 which is equal to the number of tasks. The numbers inside the boxes indicate the PE_{id}, whereas the numbers shown outside the boxes (array indices) represent task id τ_i. Each of the PE_{id} can be repeated depending on the task capacity of the processors of the target platform. By this particle structure, we represent multitask assignment on the PEs of the given multicore platform.

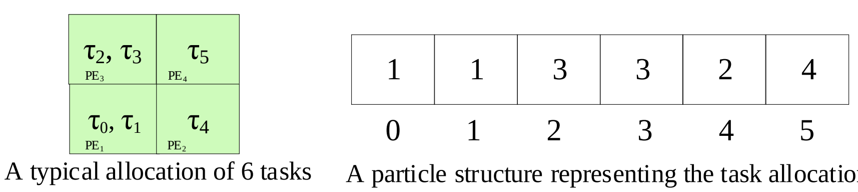


Fig. 2. Particle structure for representation of task to PE assignment

In this work, we focus on the communication energy reduction and deadline satisfaction of the tasks of the application. In the fitness function formulation, we consider the communication energy of the mapping and timing constraint satisfaction for the tasks, which are represented in the particle. The fitness of a particle p_i, $FF[p_i]$ is given by Eq. 9. B_i and α are binary factors. $B_i = 1$ if p_i represents schedulable task assignment i.e. tasks can satisfy their deadline on the identified PEs, otherwise $B_i = 0$.

$$FF[p_i] = \alpha * E_{comm}[p_i] * B_i + (1 - \alpha) * B_i \tag{9}$$

α indicates if the application has dependent tasks (α=1) or independent tasks(α=0). It is determined by the edges present in the task graph. The particles with small non-zero fitness value are preferable. This is because we are interested in task allocations which give low communication energy consumption while meeting the deadline of the tasks.

Evolution of particles: The initial generation of PSO comprises of particles which represent task to PE assignment such that the tasks are schedulable. To achieve improvement in further evolutions, local best of each particle *lbest* and global best *gbest* of a given generation is considered. The global best of the generation is initialized with the particle having the lowest non-zero value of fitness function. Such a particle satisfies the task deadline and results in least communication energy. The second generation of particle is obtained by random exchange of tasks (swap operations) between the selected PE pair(s). Subsequent generations are evolved by changing the particles through a sequence of swap operations [34][16]. The swap sequences are identified to align a particle with its local best and global best values by applying on the original particle with different probabilities. The local best value of a particular particle is updated whenever the communication energy of the particle is lesser than its previous value and meets deadline of its task. This is due to the fact that although the swap operators help in obtaining particles with reduced communication cost but may result in task schedule which may not satisfy their timing constraint. In such a case, the particle may be rejected or the swap operation(s) may not be allowed. In this work, we have not allowed such swap operations. The global best value of a generation is updated when the corresponding fitness value of the present generation is better i.e. smaller in magnitude compared to the previous generation.

5.1.3 Routing for allocation templates: The generated regions are of different shapes and sizes. Let us consider a NoC based multicore platform where multiple applications are executed in parallel. In the case of regions with irregular shape, packet communication using dimension order routing may allow packets to pass through the routers associated with PEs belonging to different region. This will lead to traffic overlap between the packets of different applications, which may affect the performance of the applications running in two different regions. Thus, we use table-based routing algorithm for routing packets between a pair of communicating nodes. Here, the route of the communication packets of the tasks of assigned application are confined to the routers associated with PEs present in its allocated region. Also, tasks belonging to different applications do not share a PE. This avoids the situation of tasks using the same link and thereby avoids deadlock. We use Lee's shortest path routing algorithm to determine the routes for the communication nodes. This information is stored in a routing table present in all the routers. Based on the source and destination node, the routers (both sender/receiver and intermediate) route the packets within the allocated region.

The allocation decision consisting of spatial (PE for tasks), temporal allocations (scheduling of tasks) and routing of the data packets are stored in the offline repository as allocation templates for each application. This information is exploited during run-time to carry out online resource assignment which is explained next.

5.2 Run-time resource allocation

Fast decision making is required during assigning the tasks of incoming applications to PEs to ensure that task allocation and scheduling process has low time overhead. We use the design-time results to obtain a light-weight scheme for run-time task assignments. It consists of determining the suitable set of PEs to map the tasks of an application and scheduling their execution at run-time.

As the arrival-time of applications is not known apriori, the run-time scenarios cannot be predicted. Following factors affect the run-time task allocation: (a) the availability of PEs when a new application requests for execution and (b) timing characteristics of tasks. Due to these factors, the design-time decisions may not always be suitable for run-time implementation. Thus, when an application is submitted at run-time, the allocation of the tasks of application needs to be customized. We have developed an Online Allocation Reconfiguration (OAR) strategy for online task allocation driven by design-time results and run-time resource availability (presented in Algorithm 2).

5.2.1 Allocation region size selection at run-time: The selection of the set of PEs for allocation of tasks at run-time is carried out in two modes (i) energy-aware and (ii) finish-time-aware. In energy-aware region selection mode, the region selection and mapping help to reduce the communication energy consumption of the application being mapped on the multicore platform. This is achieved by using a heuristic approach which selects a suitable allocation template from the repository. The selection procedure is implemented in ascending order of the region sizes starting from the least size allocation template (R_{min}) up to the number of available PEs ($Avail_{size}$) on the platform. The search procedure is iterated till a suitable region size is found to fit in the available set of PEs present in the multicore platform at run-time.

ALGORITHM 2: ***Online Allocation Reconfiguration***

Input : $G = (\Gamma, \mathcal{E})$; $\mathcal{H} = (P, L)$; $Shape\_list$

Output: Reconfigured allocation and scheduling decisions of tasks of application on the available PEs at run-time

1 $PE\_Avail = get\_avail\_Proc(P)$;
2 $PE\_Cust = \emptyset$; \\set of PEs in customized PE region selected at run-time for task allocation
3 $Avail_{size}$ = size of PE_Avail;
4 obtain the region sizes R_s of G from repository;
5 **if** *energy-aware allocation mode is used* **then**
6   **if** $Avail_{size} < R_{max}$ **then**
7   **for** $R_s = R_{min}$ *to* $Avail_{size}$ **do**
8   **for** *each shape* $S_i \in Shape\_list$ *with size* = R_s **do**
9                 $(PE\_Cust, status) = Online\_Allocation\_Adapt(S_i, PE\_Avail)$;
10   **if** *status is TRUE* **then**
11   **goto** line 24;
12   **else**
13   **goto** line 8 with $R_s = R_{max}$;
14 **else**
15   **if** *finish-time aware allocation mode is used* **then**
16   **if** $Avail_{size} < R_{max}$ **then**
17   **for** $R_s = Avail_{size}$ *to* R_{min} **do**
18   **for** *each shape* $S_i \in Shape\_list$ *with size* = R_s **do**
19                     $(PE\_Cust, status) = Online\_Allocation\_Adapt(S_i, PE\_Avail)$;
20   **if** *status is TRUE* **then**
21   **goto** line 24;
22   **else**
23   **goto** line 18 with $R_s = R_{max}$;
24 **for** *each task* $\tau_i \in \Gamma$ **do**
25   τ_{sel} = task with earliest deadline among the tasks ready for execution;
26   PE_{target} = PE assigned to most communicating parent task of τ_{sel};
27   PE_{sel} = PE $\in PE\_Cust$ with $EAT(PE) < slack - time(\tau_{sel})$ and nearest to PE_{target};
28   assign start-time of τ_{sel} on PE_{sel};
29   update EAT(PE_{sel});

The second region selection mode consists of finish-time-aware allocation template selection. Here, we select an allocation size which results in the least finish-time of the application. First, the regions with size equal to the number of PEs available on the platform is checked. Then the algorithm considers pre-computed regions in descending order of their region sizes. The availability of free PEs on the platform affects the parallel execution of tasks present in a given application. Larger sized allocation region, offer a better opportunity for simultaneous execution of these tasks due to the availability of PEs and hence improve the finish-time of the application. This process is repeated until a suitable region size fits into the available PEs during run-time. In cases where $Avail_{size}$ is greater than the size of any of the allocation regions, then the templates of highest region size is selected.

5.2.2 Run-time adaptation of task allocation: In this step, the actual task allocation and scheduling on the PEs is determined at run-time. Towards this end, an appropriate shape of the allocation region for a chosen region size needs to be decided for using during run-time scenario. The region shapes pre-determined at design-time may not always be appropriate for directly adopting on the available PEs of the platform. Thus, it is necessary to customize the shape of the selected allocation template. This problem of run-time adaptation of allocation template is solved by invoking function *Online_Allocation_Adapt()* in lines 8, 18 of Algorithm 2. It reconfigures the offline mapping and scheduling decisions by identifying suitable PEs. The task having earliest deadline among the tasks ready for execution is selected for allocation. PEs with available task capacity are considered for mapping task(s) if their EAT is within the slack-time margin (refer Eq. 1) of the task to be mapped. Such a PE is chosen as the PE hosting the most communicating parent task or in its close vicinity. Tasks are then assigned start-time of execution on the identified PEs. *Online_Allocation_Adapt()* employs the following heuristics for the above process of allocation and scheduling:

A) $best\_fit()$: In this scheme, the pre-determined allocation template found at design-time can be directly applied to the set of available PEs at run-time. This occurs in the scenario when the distribution of available PEs on the platform is such that the shapes of the allocation region can be accommodated without any change in spatial allocation. This scheme provides the opportunity to maximally exploit the design-time results with incremental changes. These changes involve updating the start-time of tasks depending of the EAT of the selected PEs for task execution.

B) $reorient\_fit()$: During run-time, the PEs availability depends on the timing characteristics of the tasks executing on them. As tasks can have different execution times, the spatial distribution of the available PEs may not readily match with the allocation templates. Reorient fit heuristic is used in this condition for adapting the design-time allocation templates according to the run-time resource distribution. Here, the allocation templates are customized by re-orientation of allocation region to fit the spatial distribution of the available set of PEs. In the context of the mesh-based NoC platform, it is sufficient to consider rotating the allocation templates by 90° towards left or right along with mirroring in the horizontal or vertical direction.

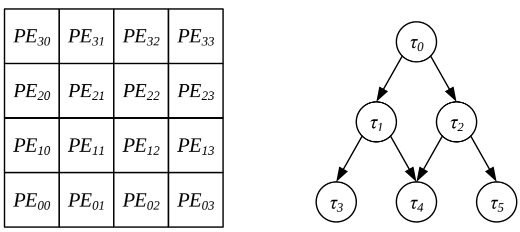
C) $flexible\_fit()$: The distribution of available PEs can be scattered on the platform. As a result, the pre-determined allocation templates maynot be fit to those PEs by directly applying on the available set of PEs (BF) or by re-orientation (RF). To overcome this, $flexible\_fit()$ heuristic is used for customizing the allocation templates. For a selected region size, the allocation template is chosen which has most PE locations similar to the distribution of the available PEs on the platform. Next, the mapping and scheduling decisions of the tasks in the allocation region are customized. This is done by considering the PE executing the most communicating parent task or in its close vicinity. Such PEs are selected with EAT less than the available slack time of the task. This policy helps to reduce the rise in communication energy consumption while meeting task deadline.

Using these heuristics, the design-time results are customized in accordance with the prevalent run-time scenario for online resource allocation of the tasks of application.

5.3 Working Example

In this sub-section, we explain the working of the proposed algorithm. Let us consider an application A_1 consisting of six tasks $\tau_0, \tau_1, \ldots \tau_5$ as shown in Fig. 3a. In the table depicted in Fig.3a, the execution time and deadline of each task are shown. We assume that the target platform for execution of this application is a 4×4 NoC based multicore platform with maximum $PE_{cap} = 2$ for each PE. The design-time analysis as detailed in subsection 5.1 is carried out to obtain the spatial allocation and configuration of the tasks. The possible sizes of allocation region and the relative distribution of the PEs constituting these regions are depicted in the repository presented in Fig. 3b. The AMD and the application finish-time of the resulting allocation templates are shown in the table given in Fig. 3b. We find that for assigning the tasks of the given application, allocation region size can vary between 3 to 6. In this example, we assume the upper limit on the AMD of the allocation shape to be 4.0 during region shape generation. Therefore, we illustrate the repository for region size (R_{size})

upto 5. Each of these generated allocation regions is used for obtaining the allocation template of the tasks showing various timing response for the target application.



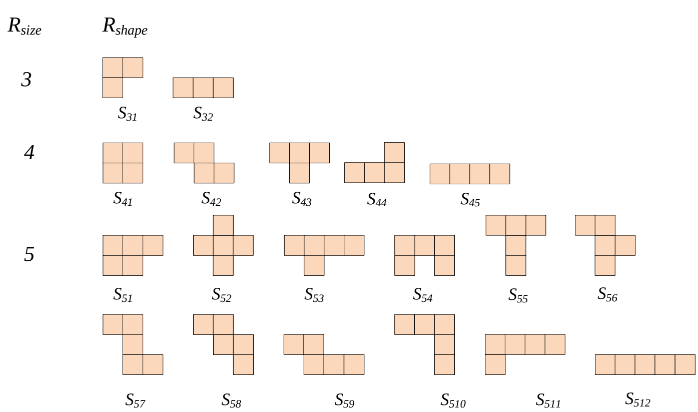
A 4x4 multicore platform

Application A_1

Tasks	Execution time (msec)	Deadline (msec)
τ_0	1.0	1.5
τ_1	0.5	2.0
τ_2	0.5	1.5
τ_3	1.5	3.2
τ_4	1.0	4.3
τ_5	1.5	4.8

Task timing characteristics

(a)

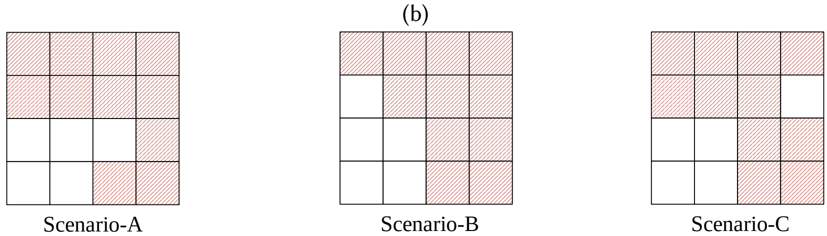


Different shapes of the regions

R_{shapes}	AMD	Finish Time (msec)	$E_{comm}(uJ)$
S_{31}	1.33	4.0	97.11
S_{32}	1.33	4.0	167.40
S_{41}	2.0	3.5	205.01
S_{42}	2.5	3.5	178.51
S_{43}	2.25	3.5	260.1
S_{44}	2.5	3.5	336.39
S_{45}	2.5	3.5	341.95
S_{51}	3.2	3.0	341.95
S_{52}	3.2	3.0	341.95
S_{53}	3.6	3.0	358.61
S_{54}	3.6	3.0	369.72
S_{55}	3.6	3.0	391.93
S_{56}	3.6	3.0	391.93
S_{57}	4.0	3.0	397.47
S_{58}	4.0	3.0	397.47
S_{59}	4.0	3.0	397.47
S_{510}	4.0	3.0	397.47
S_{511}	4.0	3.0	341.95
S_{512}	4.0	3.0	397.47

Typical shape characteristics

(b)



(c)

Fig. 3. Working of the proposed adaptive resource allocation strategy

Next, we explain the run-time selection and placement of the differently shaped regions from the repository generated at the design-time. We assume a time-aware allocation strategy. As online scenarios cannot be predicted during static analysis, therefore in some cases, reconfiguration of the predetermined allocation region is performed. The run-time mapping scenarios are depicted in Fig. 3c. We assume that at run-time, some applications are already active on PEs of the target multicore platform. This has been depicted by shaded regions of red color to indicate that 100% PE capacity are already used. In this example, we consider three run-time scenarios-A to C for illustrating the proposed run-time adaption and reconfiguration approach. In these scenarios, we depict a typical situation with five available PEs for task allocation and scheduling. The proposed algorithm adapts the pre-computed allocation information for A_1 to take decisions on task allocation at run-time. First, the region size of the allocation template is decided. As time-aware allocation strategy is used, the allocation templates are evaluated in decreasing order of region size. As a result, the region size comprising of 5 PEs is evaluated for task assignment. In case sufficient PEs for required region size ($R_{size} = 5$) is not available, the algorithm considers the next smaller size region for allocation. The working of online shape adaptation heuristics using different run-time scenarios is described next.

Consider the scenario-A shown in Fig. 3c. The proposed algorithm first employs the $best\_fit()$ heuristic to select an allocation template of $R_{size} = 5$. Here, the shape S_{51} is selected as it has the least AMD, and it can fit into the current distribution of PEs in scenario-1. Hence the task of A_1 is allocated and scheduled using S_{51} allocation template. Next, consider the run-time situation as shown in scenario-B. In this case, $best\_fit()$ heuristic could not select a suitable allocation template. This is because none of the shapes (in $R_{size} = 5$) match directly with the pattern of the available PEs in the run-time scenario. As a result, the pre-computed decisions are required to be customized based on the current system state. The proposed algorithm selects an allocation region for tasks of A_1 using $reorient\_fit()$ heuristic. The proposed algorithm considers the re-orientations of the allocation template of S_{51}. It fits within the available PEs with a 90° anti-clockwise re-orientation.

Scenario-C (refer Fig. 3c) depicts the case where the strategies used by $best\_fit()$ and $reorient\_fit()$ heuristics to select a suitable allocation template could not identify one for assigning the tasks of A_1. In such cases, $flexible\_fit()$ heuristic is used by the proposed algorithm. The spatial location of available PEs on the platform is most similar to template S_{51}. Hence, this template is used for driving the task allocation steps. While mapping the tasks of A_1, the proposed algorithm selects $PE_{00}, PE_{01}, PE_{10}$ and PE_{11}. However, all the tasks of A_1 cannot be fitted in the selected PEs. For mapping the remaining task, a PE is chosen during run-time in the close vicinity of the PE executing its most communicating parent task. This reduces the increase in the communication energy of the application while meeting its deadline. As all the adjacent PEs are busy (hosting maximum number of tasks), therefore PE_{23} is selected for executing the remaining task. However, this type of mapping favouring reduction of finish-time may result in a penalty on communication energy consumption of the mapped application. In such condition, selecting the allocation template with smaller region size can help in saving communication energy consumption on the given NoC based multicore platform. For e.g. if we use $R_{size} = 4$, it results in 18.3% lower communication energy consumption compared to using template S_{51} with reconfiguration. However, it results in 10.2% average rise in finish-time of task of application.

6 PERFORMANCE EVALUATION

We have evaluated the performance of the proposed algorithm experimentally. The following subsection describes the experimental setup and simulation results.

6.1 Test Setup

We have used a C++ based simulator to perform application mapping/scheduling in NoC based multicore system. The simulator can simulate different-sized 2D mesh NoC topology and is based on the previous works [3][4]. We have modified the simulator for implementing the proposed task allocation strategy. The simulation process is divided into the following steps. In the first step, we implement the design-time allocation template generation. This consists of (i) formation of different regular and irregular shapes consisting of contiguous processor allocation, (ii) a particle swarm optimization (PSO) based mapping and scheduling of tasks onto the different generated shapes and (iii) determination of routing information of the mapped tasks. The output results obtained at

Table 1. Various benchmark and synthetic applications

Application Type	Applications	Total tasks
Benchmark applications	263ENC MP3DEC	12
	263DEC MP3DEC	14
	MP3ENC MP3DEC	13
	MPEG4	12
	PIP	8
Synthetic applications	TGFF1	45
	TGFF2	60
	TGFF3	85
	TGFF4	100

Table 2. Simulation settings for ORION 3.0

Parameter	Value
Technology	45 nm
Transistor	NVT
V_{dd}	1.0 v
Router Frequency	250 MHz
No. of pipeline stages	4
Flit width	32
Link wire layer type	Global
Link wire width & spacing	DWIDTH_DSPACE

Table 3. Network settings

Parameter	Value
Packet size	64 flits (32 bits per flit)
Buffer depth	8
Selection logic	Random
Traffic	Table based
Warm up time	5000 clk cycles
Simulation time	200000 clk cycles

Table 4. Test scenarios and their initial conditions

Test scenarios	Initial condition
Scenario-1	All PEs are available
Scenario-2	10% of platform PEs occupied
Scenario-3	25% of platform PEs occupied
Scenario-4	40% of platform PEs occupied
Scenario-5	55% of platform PEs occupied

design-time are stored as repository files. The second step of the simulation is performed at the run-time, where different applications are submitted dynamically for execution on the NoC based multicore platform. The simulator selects an appropriate allocation template from the repository and customizes it based on the run-time scenarios. We consider 8×8 NoC based multicore platform for performing the experiments with one of the processors is selected as the Platform Manager. Experiments are conducted using benchmark applications such as MPEG4, MWD, MP3ENC, 263DEC and 263ENC [28]. However, the availability of large size benchmarks is limited. We use synthetic applications generated by the TGFF tool [8] to obtain the task graphs with a higher number of tasks. The number of tasks is varied between 5 to 100. Also, the benchmark and synthetic applications, as shown in Table 1, have been randomly combined to form different workloads for allocation. Simulation has been carried out using an Intel i7 processor running at 3.0 GHz. We have considered various test scenarios, which consist of initial conditions, as depicted in Table 4. These initial conditions indicate the different numbers of already occupied PEs when an incoming application requires allocation at run-time. The input to the simulator consists of an input file, which typically contains the number of applications, size of each application, execution time of tasks and their deadline. In our experiments, the computation time requirement of the tasks has been uniformly distributed between $50ms$ to $250ms$. Please note that the deadline allocation for each task is similar to [3][4].

Using this information, the simulator allocates the tasks belonging to an application to the available processors as determined by the task allocation algorithm. There are different parameters based on which the quality of allocation results is assessed. These parameters include communication energy, deadline satisfaction of tasks and average communication latency of the allocated application. The energy consumption for packet traversal (E_{comm}) has been computed using Eq. (4).

We have used link energy (E_{link} = 3.12×10^{-13} J/bit) and router energy (E_{router} = 5.24×10^{-12} J/bit) derived from ORION 3.0 power model [15]. The parameter settings for ORION is shown in Table 2. We have used Noxim [2], a cycle accurate simulator, to determine the overall network performance considering an 8×8 2D mesh interconnection. The configuration of Noxim simulator is shown in Table 3.

6.2 Evaluation of proposed run-time mapping and scheduling algorithm

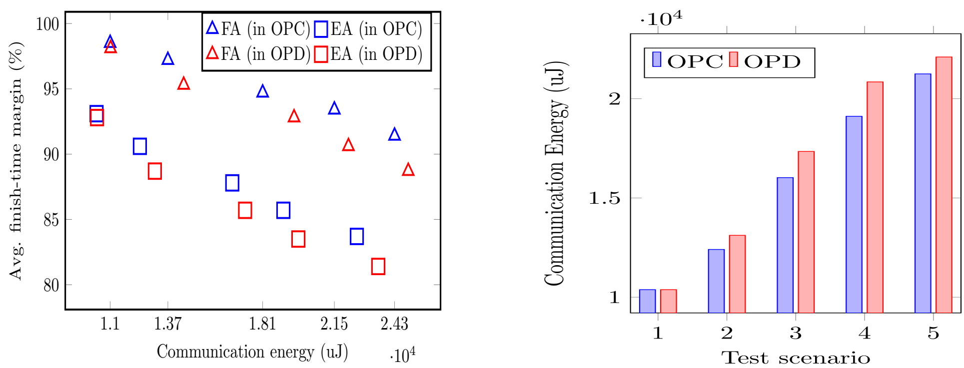
In this subsection, we evaluate the performance of the proposed dynamic task allocation and scheduling algorithm in terms of communication energy of the allocated applications, finish-time of the mapped tasks and average packet latency of the tasks of the applications. To evaluate the adaptivity of the proposed approach to run-time conditions, application workloads are assumed to arrive randomly at any time with varying availability of PEs as indicated by the different test scenarios. While selecting PEs for task allocation, the occupied PEs are unavailable for new task assignments. For each initial condition corresponding to a test scenario, two cases are considered. In the first case, OPC (occupied PEs-contiguous), we consider the already occupied PEs to be present in contiguous regions on the platform. For experimentation, such regions are assumed to be located on top-left, top-right, bottom-left, bottom-right and centre region of the multicore platform. In the second case, the occupied PEs are considered to be distributed randomly on the multicore platform. This case is referred to as OPD (occupied PEs distributed) in the results. The results for the OPD case are averaged over 100 random arrangements of occupied PEs for each test scenario.

6.2.1 Effect of allocation modes on performance. We have evaluated the effect of allocation modes on energy-consumption and finish-time of the allocated applications. We consider different test scenarios with varying numbers of occupied PEs in the given NoC platform, as shown in Table 4. For each of the test scenarios, we have used energy-aware (EA) mode and finish-time aware (FA) mode separately for selection of allocation regions. The resulting communication-energy and average finish-time margin of the allocated applications is shown in Fig 4a. For any application workload with N tasks, AFTM is given by Eq. 10. It indicates the time margin by which tasks complete their execution within their corresponding deadline.

$$AFTM = \frac{1}{N} \sum_{\forall \tau_i \in \Gamma} \frac{dl_i - finish(\tau_i)}{dl_i} \tag{10}$$

It is found that the task allocations using EA mode shows on an average, 17.3% less communication energy consumption compared to the case when the same tasks are assigned using FA mode during region selection. However, AFTM of tasks allocated by using EA mode is, on an average, 20.4% lesser compared to that of allocations using FA mode. Thus, EA mode is suitable for communication energy reduction while FA is more useful for improving the timing performance of tasks.

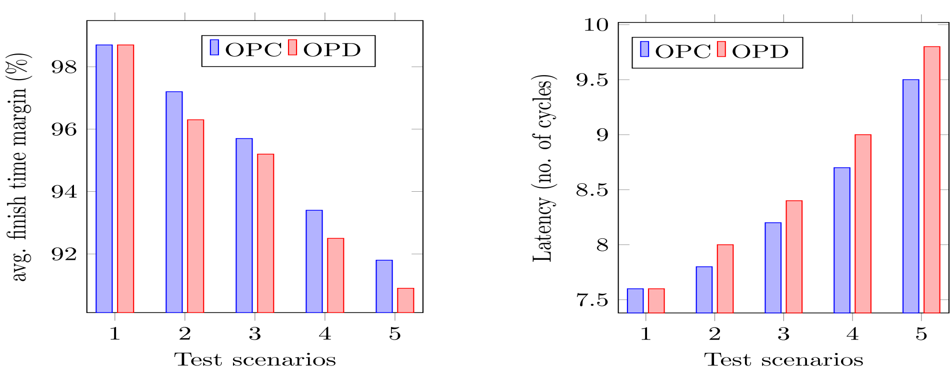
6.2.2 Communication Energy. In this subsection, we evaluate the performance of the energy-aware mode of the proposed dynamic task allocation strategy in terms of the communication energy consumption of the allocated applications. Fig. 4b shows the communication energy consumed by the mapped application during their execution. On an average, test-scenarios 2, 3, 4 and 5 with already occupied PEs in the initial condition distributed on the platform, result in 10.6%, 19.4%, 27.1% and 36.3% more communication energy consumption as compared to test-scenario 1, respectively. Test scenario-1 gives the least communication energy of the allocated application as all the PEs are considered to be available for task assignment. As the number of already occupied PEs of the platform increases progressively from test-scenarios 2 to 5, the mapped region becomes less compact due to the non-availability of free PEs in the regular region for task assignment. This increases the communication energy of the mapped application. For e.g. in test scenarios with OPD cases, it is found that the communication energy consumption is, on an average, 21.3% higher as compared to the applications mapped in the same test scenario with OPC case.



(a) Effect of allocation mode on performance

(b) Comm. energy variation with EA mode

Fig. 4. Evaluation of allocation modes and communication energy performance with PE availability



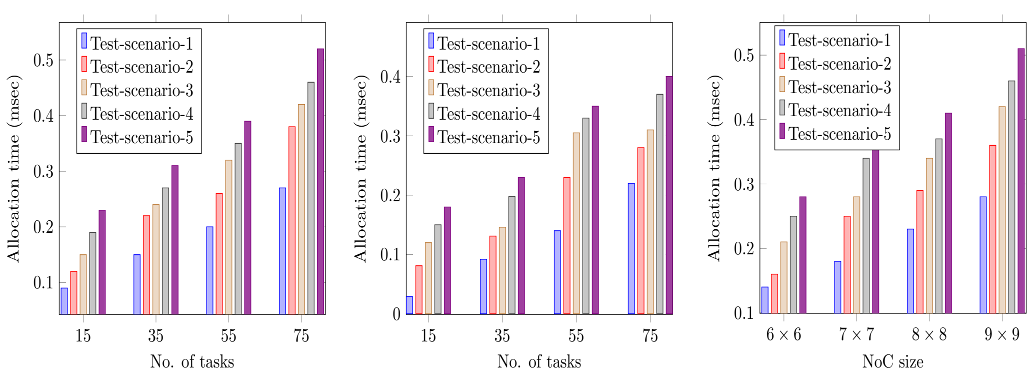
(a) Avg. finish-time of applications with FA mode

(b) Network latency with FA mode

Fig. 5. Impact of PE availability on task finish-time and network performance of allocated applications

6.2.3 Timing performance. The finish-time aware mode of the proposed approach for run-time task allocation considers the completion time of the tasks while choosing an allocation region for their execution. The time margin by which the tasks satisfy their deadline is affected by the size and shape of the allocated region on the platform. Fig. 5a explores the effect of allocated region on the timing performance of the applications in terms of average finish time margin (AFTM) of their tasks. On average, AFTM drops by 20.7% when the occupied PEs considered in the test scenarios rises by 15%. When tasks are allocated, in test-scenarios with higher number of occupied PEs in initial conditions, the allocated region size is more as the PEs located sparsely are selected for task execution. This increases the chance of more tasks being allocated to the already occupied PEs depending on (i) capacity of multitasking and (ii) EAT of the selected PE. Such tasks can start their execution only after the earlier assigned task(s) finish their execution. Additionally, the AFTM is also affected by the contiguity of the already occupied PEs on the platform. For a given test scenario, the AFTM of the scheduled tasks, is on an average, 15.3% lower in case of test-scenarios with distributed patterns compared to contiguously occupied PEs. The difference is more prevalent when the test-scenario has the occupied PEs present in contiguous region. Consequently, better opportunity for simultaneous execution of the allocated tasks of application exists due to the availability of PEs. More number of scheduled tasks complete their execution within their deadline. As a result, the AFTM of the tasks increases when the initial condition has occupied PEs in contiguous region.

6.2.4 Communication Latency. The average network latency of the allocated applications under various test scenarios is shown in Fig.5b. It can be observed that with 15% increase in PE occupancy in the initial condition of the test scenarios, the latency of the allocated application rises by 19.2%. As more number of available PEs are present in distributed manner across the platform, the resultant allocated regions are irregular shaped. The communicating tasks get assigned to PEs located sparsely. Data packets from source to destination PEs use more network resources, such as routers and links, for communication between the dependent tasks. Further, it is observed that occupied PEs present in regular regions show reduced communication energy compared to occupied PEs distributed across the platform. On an average the allocated applications show 23.5% higher latency in test scenarios with OPD as compared to OPC.



(a) Allocation time in OPD test scenarios
(b) Allocation time in OPC test scenarios
(c) Scalability with number of PEs

Fig. 6. Scalability of proposed approach with no. of tasks and processors

6.2.5 Scalability. The scalability of the proposed approach with the number of tasks is presented in Fig. 6. We have varied the number of tasks for allocation and the corresponding allocation time taken for executing the proposed method is reported. We have conducted three sets of experiments. In the first set of experiment, we evaluate the allocation time of the tasks considering OPD case. The results are shown in Fig. 6a. It is found that as the number of tasks for allocation increases, the allocation time also rises. On an average, 6.7% rise in allocation time occurs with 10% increase in number of tasks. This is because the scheduler needs to execute the algorithm more number of times to complete the task allocation process. However, for a given number of tasks, the time taken by the scheduler to finish mapping and scheduling decision increases with more number of already occupied PEs present in test scenario in OPD. It is attributed to frequent customization carried out by the $flexible\_fit()$ heuristic for run-time adaptation of task allocation when the PE availability does not match with the pre-computed region shapes. This increases when more number of already occupied PEs are present distributed in the platform.

In the second set of experiment, we have repeated the allocation of the tasks but considering the OPC test cases in regular shapes. As depicted in Fig. 6b, it is observed that there is an average increase of 4.2% in allocation time with 10% increase in number of tasks. However, the rise in allocation time is less compared to first set of experiments with OPD. This is due to the fact that in case of test-scenario with OPC, the remaining PEs are available in contiguous regions. Thus in most of the cases, the proposed algorithm performs incremental customization by invoking $best\_fit()$ and $reorient\_fit()$ heuristics. As a result, time consumed in completing the task allocation is lesser compared to former set of experiments.

The third set of experiment evaluates the scalability of the proposed run-time approach with number of processors. The platform size is varied from 6×6 to 9×9 while fixing the number of allocated tasks to 60. As shown in Fig 6c, the allocation time increases for each test scenario as the size of the target platform increases. More number of PEs are present in larger sized platform, so exploring and selecting a suitable set of PE for task allocation and scheduling consumes more time. It is observed that on an average, for a given test scenario, the allocation time rises by 7.2% as the platform size is changed from 6×6 to 9×9 which is reasonable compared to the increase in number of PEs ($\times 2.2$).

6.3 Comparison with existing works

In this section, we compare the performance of the proposed dynamic allocation approach with the selected state-of-the-art run-time allocation methods reported in the literature for NoC based multicore systems having processors with multitasking capability. We first present the comparison results of the run-time performance of the proposed hybrid mapping technique followed by the discussion on the performance of the design-time computation stage of the algorithm. Deadline and Energy-aware Mapping and Scheduling (DEAMS) [4] approach uses an online heuristic to assign and schedule the tasks on multicore platforms. It selects a PE for assigning tasks of a submitted application while considering the timing characteristics of the tasks. The resultant allocation chooses an occupied PE for executing an unmapped task provided the task has enough slack time to finish within its deadline. Communication aware Packing based Nearest Neighbour (CPNN) described in [32] attempts to allocate the maximum communicating tasks on the same PE. The run-time approach described in [18] performs dynamic allocation of tasks to PEs and, thereafter, performs iterative improvement on the task mapping decisions by a Communication-Aware task Migration (CAM) strategy. The Two-Step Multi-application Mapping (TSMM) heuristic mentioned in [38] uses application mapping followed by task allocation within the selected region of PEs to minimize the communication energy and execution time of each application. The comparison results have been discussed in the following subsections.

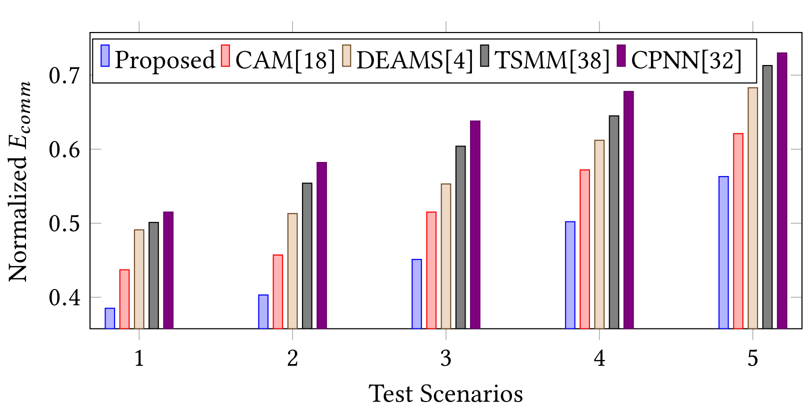


Fig. 7. Comparison of communication energy performance for various dynamic mapping methods

6.3.1 Communication Energy. Fig.7 shows the comparison of the communication energy consumed by the applications allocated by various dynamic allocation methods concerning the run-time heuristic of the proposed approach. The results are expressed as normalized values for Nearest-Neighbour (NN) [1] method considered as the baseline method. From test-scenarios-1 to 5, the set of available PEs on the platform becomes progressively distributed across the platform. All the compared algorithms result in task allocation, which shows rise in communication energy consumption. However, the proposed algorithm out-performs the other run-time algorithms. It results in 24.2% reduction in communication energy of the allocated applications. When compared with CAM, DEAMS, TSMM, and CPNN techniques, the task allocations resulting from the proposed approach achieve 11.5%, 22.3%, 28.6% and 34.6% average reduction in communication energy.

6.3.2 Deadline Performance. Fig. 8 reports the results of comparison of number of tasks meeting their corresponding deadline. Based on our simulations, we observe that using the proposed method, on an average 19.4%, 28.2% and 37.7% more number of tasks satisfy deadline compared to the cases when the same tasks are allocated and scheduled by DEAMS, TSMM and CPNN techniques, respectively. CPNN, TSMM and CAM algorithms solely focus on communication dependency between the tasks and ignore their timing characteristics while mapping them onto the same PE in the multicore platform. This policy causes an increase in the finish-time of the tasks, which negatively impacts their deadline performance. Tasks allocated dynamically using DEAMS show better results compared to the methods mentioned above and result in deadline misses when the task slack-time is low. The proposed dynamic allocation approach exploits the pre-computed decisions of mapping and scheduling for the run-time assignment of the tasks on the multitasking PEs. It uses the timing characteristics of tasks and selects suitable PEs closely located for run-time allocation customization. Thus, the number of tasks completing execution within their deadline increases.

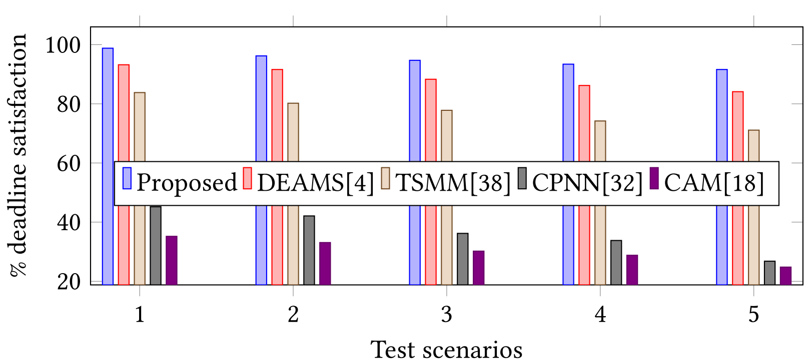


Fig. 8. Deadline performance comparison of different dynamic allocation approaches

6.3.3 Allocation time. It is essential to consider the cost of an online algorithm, i.e., the time spent by the Platform Manager to apply the allocation policy to tasks of the arrived application. A comparison of time-taken by various algorithms for dynamic task allocation is depicted in Fig.9 under various test scenarios. Simulation has been carried out on an Intel i7 processor running at 3.0 GHz frequency. The proposed approach gives an improved performance concerning the time consumed in the allocation of tasks of the application. When compared to DEAMS, the run-time strategy of the proposed dynamic allocation approach gives a 14.1% average improvement in allocation time. The proposed approach takes into account the resource availability at run-time and performs incremental changes in allocation templates for quick on-line mapping and scheduling. The allocation time of the proposed algorithm shows 12.4% and 28.7% average reduction over CPNN and CAM algorithms, respectively. CPNN and CAM find only the mapping solution in each iteration by selecting appropriate PEs for communication energy reduction. Furthermore, the iterative task re-mapping performed by CAM adds to the allocation time of the tasks. TSMM results in 18.7% more allocation time on an average as compared to the proposed algorithm. This is attributed to the use of two sequential steps in TSMM at run-time, which involve application region selection and task mapping within the selected region.

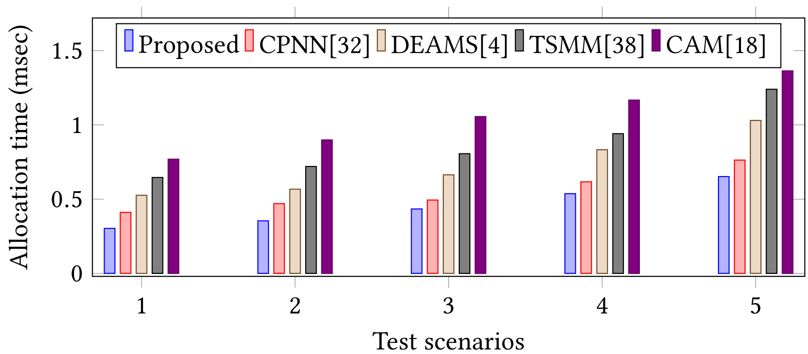


Fig. 9. Allocation time of different run-time heuristics

6.3.4 Design-time performance comparison. We compare the performance of different algorithms for design space exploration to allocate tasks onto multitasking processors in a NoC based multicore platform. In this work, we have used PSO to assign tasks on PEs in the multicore platform. PSO has been reported to have faster convergence than other evolutionary techniques like the Genetic Algorithm (GA) [35] and it is capable of working with a population of relatively small size [28]. Thus, we use existing PSO based DSE approaches for comparing with our proposed allocation strategy. We have considered, discrete PSO (DPSO) based application mapping technique [14] [28] to compare with the proposed PSO based mapping for the allocations found by the design-time method on benchmark applications. Besides, a GA based mapping GBMAP, presented in [35], is also used to compare the mapping results. Table 5 depicts the the exploration time (in sec) and communication cost (CC) i.e. the product of hop counts and data volume of the resultant mapping. The CC values are normalized with respect to NMAP[22] technique, which is popularly used for comparing the application allocation results. For the sake of comparison, in our experiments, we have considered a multicore system with a capacity of one task per PE. The proposed approach gives 23.3%, 16.6%, and 11.1% lower communication cost on an average, as compared to NMAP, PSMAP and DPSO methods respectively. NMAP, PSMAP, and DPSO approaches map the tasks of a given application onto a fixed rectangular-shaped region. Other shaped are not explored in these works. The solution space consisting of PEs arranged in contiguous and non-contiguous regions is larger than fixed rectangular-shaped allocation regions. We address the shape generation and task allocation during design-time separately. This helps to effectively explore the larger search-space inherent in case of rectangular and non-rectangular regions. Therefore, the quality of the resultant mapping using our approach is better compared to other works mentioned in the literature.

7 CONCLUSION

We have proposed a hybrid mapping and scheduling algorithm for NoC-based multicore platform with multitasking processors. Our approach consists of design-time allocation exploration followed by run-time selection and configuration of the appropriate allocation template. The allocation

Table 5. Communication cost (cc) & DSE-time (in sec) by different DSE approaches for hybrid strategy

Benchmarks	NMAP[22]		GBMAP[35]		PSMAP[29]		DPSO[14] [28]		Proposed	
	CC	DSE-time	CC	DSE-time	CC	DSE-time	CC	DSE-time	CC	DSE-time
MPEG4	1	0.016	0.94	0.03	0.93	0.04	0.93	2.10	0.912	2.33
263ENC MP3DEC	1	0.012	1	0.13	1	0.26	1	2.08	1	2.36
PIP	1	0.01	1	0.01	1	0.01	1	0.42	1	0.47
MP3ENC MP3DEC	1	0.01	1	0.21	1	0.32	1	1.97	0.821	2.22
263DEC MP3DEC	1	0.01	0.987	0.16	0.969	0.23	0.969	2.09	0.903	2.31

explored during design-time consider the multitasking capability of the processors and task timing constraints of the given applications. The proposed approach adapts to the dynamism of the application workload by exploiting these solutions at run-time, based on the resource availability and application timing requirements. We have evaluated our strategy and compared its performance with other task mapping and scheduling algorithms reported in the literature. Experimental results indicate its effectiveness in terms of communication energy consumption and deadline satisfaction of the allocated applications. Thus, the proposed strategy is suitable for run-time allocation of dynamic workloads with multiple applications with real-time constraints on NoC-based multicore systems. In future, we plan to extend this work to consider the actual execution time of tasks at run-time by updating the pre-computed decisions. Also, the proposed hybrid approach can be augmented to limit the mutual interference between tasks for hard real-time systems.

REFERENCES

[1] E. Carvalho, N. Calazans, and F. Moraes. 2010. Dynamic Task Mapping for MPSoCs. *IEEE Design Test of Computers* 27, 5 (Sep. 2010), 26–35. https://doi.org/10.1109/MDT.2010.106

[2] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti. 2016. Cycle-Accurate Network on Chip Simulation with Noxim. *ACM Trans. Model. Comput. Simul.* 27, 1, Article 4 (Aug. 2016), 25 pages.

[3] N. Chatterjee, S. Paul, and S. Chattopadhyay. 2017. Fault-Tolerant Dynamic Task Mapping and Scheduling for Network-on-Chip-Based Multicore Platform. *ACM Trans. Embed. Comput. Syst.* 16, 4, Article 108 (May 2017), 24 pages.

[4] N. Chatterjee, S. Paul, P. Mukherjee, and S. Chattopadhyay. 2017. Deadline and energy aware dynamic task mapping and scheduling for Network-on-Chip based multi-core platform. *Journal of Systems Architecture* 74 (2017), 61 – 77.

[5] C. Chou and R. Marculescu. 2010. Run-Time Task Allocation Considering User Behavior in Embedded Multiprocessor Networks-on-Chip. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Syst.* 29, 1 (Jan 2010), 78–91.

[6] Chen-Ling Chou and Radu Marculescu. 2008. User-aware Dynamic Task Allocation in Networks-on-chip. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '08)*. ACM, New York, NY, USA, 1232–1237.

[7] W. J. Dally and B. Towles. 2001. Route packets, not wires: on-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*. 684–689. https://doi.org/10.1109/DAC.2001.156225

[8] Robert P Dick, David L Rhodes, and Wayne Wolf. 1998. TGFF: task graphs for free. In *Proceedings of the 6th international workshop on Hardware/software codesign*. IEEE Computer Society, 97–101.

[9] Mohammad Abdullah Al Faruque, R. Krist, and J. Henkel. 2008. ADAM: Run-time agent-based distributed application mapping for on-chip communication. In *2008 45th ACM/IEEE Design Automation Conference*. 760–765.

[10] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila. 2013. Smart hill climbing for agile dynamic mapping in many-core systems. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.

[11] M. Fattah, P. Liljeberg, J. Plosila, and H. Tenhunen. 2014. Adjustable contiguity of run-time task allocation in networked many-core systems. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 349–354.

[12] M. Fattah, M. Ramirez, M. Daneshtalab, P. Liljeberg, and J. Plosila. 2012. CoNA: Dynamic application mapping for congestion reduction in many-core systems. In *2012 IEEE 30th International Conf. on Computer Design (ICCD)*. 364–370.

[13] J. Hu and R. Marculescu. 2004. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *DATE, 2004. Proceedings*, Vol. 1. 234–239.

[14] Soumya J., Ashish Sharma, and Santanu Chattopadhyay. 2014. Multi-Application Network-on-Chip Design Using Global Mapping and Local Reconfiguration. *ACM Trans. Reconfigurable Technol. Syst.* 7, 2, Article 7 (July 2014), 24 pages.

[15] A. B. Kahng, B. Lin, and S. Nath. 2015. ORION3.0: A Comprehensive NoC Router Estimation Tool. *IEEE Embedded Systems Letters* 7, 2 (June 2015), 41–45. https://doi.org/10.1109/LES.2015.2402197

[16] Kang-Ping Wang, L.Huang, Chun-Guang Zhou, and W. Pang. 2003. Particle swarm optimization for traveling salesman problem. In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics*, Vol. 3. 1583–1585.

[17] G. Luo, H. Zhao, and C. Song. 2008. Convergence Analysis of a Dynamic Discrete PSO Algorithm. In *2008 First International Conference on Intelligent Networks and Intelligent Systems*. 89–92. https://doi.org/10.1109/ICINIS.2008.100

[18] Tahir Maqsood, Sabeen Ali, Saif U.R. Malik, and Sajjad A. Madani. 2015. Dynamic task mapping for Network-on-Chip based systems. *Journal of Systems Architecture* 61, 7 (2015), 293 – 306. https://doi.org/10.1016/j.sysarc.2015.06.001

[19] G. Mariani *et al.* 2010. An Industrial Design Space Exploration Framework for Supporting Run-time Resource Management on Multi-core Systems. In *DATE '10*. 196–201.

[20] Armin Mehran, Ahmad Khademzadeh, and Samira Saeidi. 2008. DSM: A Heuristic Dynamic Spiral Mapping algorithm for network on chip. *IEICE Electronic Express* 5, 13 (2008), 464–471.

[21] Hashan Mendis, Neil Audsley, and Leandro Indrusiak. 2017. Dynamic and Static Task Allocation for Hard Real-Time Video Stream Decoding on NoCs. *Leibniz Transactions on Embedded Systems* 4, 2 (2017), 01–1–01:25.

[22] S. Murali and G. De Micheli. 2004. Bandwidth-constrained mapping of cores onto NoC architectures. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, Vol. 2. 896–901 Vol.2.

[23] Thanh Dinh Ngo, Kevin J. M. Martin, and Jean-Philippe Diguet. 2017. Move Based Algorithm for Runtime Mapping of Dataflow Actors on Heterogeneous MPSoCs. *Journal of Signal Processing Systems* 87, 1 (01 Apr 2017), 63–80.

[24] L. Ost *et al.* 2013. Power-aware Dynamic Mapping Heuristics for NoC-based MPSoCs Using a Unified Model-based Approach. *ACM Trans. Embed. Comput. Syst.* 12, 3, Article 75 (April 2013), 22 pages.

[25] Wei Quan and Andy D. Pimentel. 2015. A Hybrid Task Mapping Algorithm for Heterogeneous MPSoCs. *ACM Trans. Embed. Comput. Syst.* 14, 1, Article 14 (Jan. 2015), 25 pages. https://doi.org/10.1145/2680542

[26] Wei Quan and Andy D. Pimentel. 2015. A Hybrid Task Mapping Algorithm for Heterogeneous MPSoCs. *ACM Trans. Embed. Comput. Syst.* 14, 1, Article 14 (Jan. 2015), 25 pages. https://doi.org/10.1145/2680542

[27] Pradip Kumar Sahu and Santanu Chattopadhyay. 2013. A survey on application mapping strategies for Network-on-Chip design. *Journal of Systems Architecture* 59, 1 (2013), 60 – 76. https://doi.org/10.1016/j.sysarc.2012.10.004

[28] P. K. Sahu, T. Shah, K. Manna, and S. Chattopadhyay. 2014. Application Mapping Onto Mesh-Based Network-on-Chip Using Discrete Particle Swarm Optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22, 2 (Feb 2014), 300–312. https://doi.org/10.1109/TVLSI.2013.2240708

[29] P. K. Sahu, P. Venkatesh, S. Gollapalli, and S. Chattopadhyay. 2011. Application Mapping onto Mesh Structured Network-on-Chip Using Particle Swarm Optimization. In *2011 IEEE Computer Society Annual Symp. on VLSI*. 335–336.

[30] Amit Kumar Singh, Akash Kumar, and Thambipillai Srikanthan. 2013. Accelerating Throughput-aware Runtime Mapping for Heterogeneous MPSoCs. *ACM Trans. Des. Autom. Electron. Syst.* 18, 1, Article 9 (Jan. 2013), 29 pages.

[31] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel. 2016. Resource and Throughput Aware Execution Trace Analysis for Efficient Run-Time Mapping on MPSoCs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 1 (Jan 2016), 72–85. https://doi.org/10.1109/TCAD.2015.2446938

[32] Amit Kumar Singh, Thambipillai Srikanthan, Akash Kumar, and Wu Jigang. 2010. Communication-aware Heuristics for Run-time Task Mapping on NoC-based MPSoC Platforms. *J. Syst. Archit.* 56, 7 (July 2010), 242–255.

[33] S. Skalistis and A. Simalatsar. 2017. Near-optimal deployment of dataflow applications on many-core platforms with real-time guarantees. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. 752–757.

[34] J. Soumya, K. Naveen Kumar, and Santanu Chattopadhyay. 2015. Integrated core selection and mapping for mesh based Network-on-Chip design with irregular core sizes. *Journal of Systems Architecture* 61, 9 (2015), 410 – 422.

[35] M Tavanpour, A Khademzadeh, S Pourkiani, and Mohammad Yaghobi. 2010. GBMAP: An evolutionary approach to mapping cores onto a mesh-based NoC architecture. *Journal of Communication and Computer* 7 (01 2010), 1–7.

[36] Wei Quan and A. D. Pimentel. 2013. A scenario-based run-time task mapping algorithm for MPSoCs. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.

[37] Inc. Xilinx. 2019. Versal: The First Adaptive Compute Acceleration Platform (ACAP) (WP505-v1.0.1). *White Paper*. https://www.xilinx.com/support/documentation/white_papers/wp505-versal-acap.pdf

[38] B. Yang, L.Guang, T. Santti, and J. Plosila. 2013. Mapping multiple applications with unbounded and bounded number of cores on many-core networks-on-chip. *Microprocessors and Microsystems* 37, 4 (2013), 460 – 471.

[39] C. Ykman-Couvreur *et al.* 2011. Linking run-time resource management of embedded multi-core platforms with automated design-time exploration. *IET Computers Digital Techniques* 5, 2 (March 2011), 123–135.