HAL
open science

A Hybrid Adaptive Strategy for Task Allocation and Scheduling for Multi-applications on NoC-based Multicore Systems with Resource Sharing

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

To cite this version:

Suraj Paul, Navonil Chatterjee, Prasun Ghosal, Jean-Philippe Diguet. A Hybrid Adaptive Strategy for Task Allocation and Scheduling for Multi-applications on NoC-based Multicore Systems with Resource Sharing. 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), Feb 2021, Grenaoble (VIrtual), France. pp.1663-1666, 10.23919/DATE51398.2021.9474079 . hal-03450383

HAL Id: hal-03450383

https://cnrs.hal.science/hal-03450383v1

Submitted on 26 Nov 2021

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

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

A Hybrid Adaptive Strategy for Task Allocation and Scheduling for Multi-applications on NoC-based Multicore Systems with Resource Sharing

Suraj Paul*, Navonil Chatterjee†, Prasun Ghosal* and Jean-Philippe Diguet†
*Indian Institute of Engineering Science and Technology, Shibpur, India
†Lab-STICC, CNRS, UBS Research Center, Lorient, France
Email: suraj.rs2017@it.iiests.ac.in, chatterjee.navonil@cnrs.fr, p_ghosal@it.iiests.ac.in, jean-philippe.diguet@cnrs.fr

***Abstract*—Allocation and scheduling of applications affect the timing response and system performance, particularly for Network-on-Chip (NoC) based multicore systems executing real-time applications. These systems with multitasking processors provide improved opportunity for parallel application execution. In dynamic scenarios, runtime task allocation improves the system resource utilization and adapts to varying application workload. In this work, we present an efficient hybrid strategy for unified allocation and scheduling of tasks at runtime. By considering multitasking capability of processors, communication cost and task timing characteristics, potential allocation solutions are obtained at design-time. These are adapted for dynamic mapping and scheduling of computation and communication workloads of real-time applications. Simulation results show that the proposed approach achieves 34.2% and 26% average reduction in network latency and communication cost of the allocated applications. Also, the deadline satisfaction of the tasks improves on average by 42.1% while reducing the allocation-time overhead by 32% when compared with existing techniques.**

***Index Terms*—Dynamic task allocation, Multicore, Scheduling**

I. Introduction

Network-on-Chip (NoC) characterized by concurrent communication and improved scalability, has emerged as the *de facto* choice of on-chip interconnects in multicore systems. In NoC paradigm, an on-chip network communicates data packets between the PEs using network interface (NI), routers and links. A PE is interfaced to a router through an NI module, while the point to point links connect the routers with each other. Task allocation and scheduling is a challenging problem as it impacts the system performance, especially for NoC based multicore systems. The complexity further magnifies when such systems host PEs with multitasking operating systems. Such platforms offer increased opportunity of parallelism at task and/or application level with multiple allocation configurations. This necessitates design of focus on multi-application deployment to achieve high performance.

In *Real-Time Dynamic Systems (RTDS)* [1] executing real-time applications, the computation and communication workloads often vary on a given system. Offline allocation methods are insufficient in such scenarios as they consider predefined set of applications. Thus, dynamic resource allocation strategies which account for variation in runtime workload are essential for allocating tasks and their communication transactions. Typically resource allocation at runtime can be performed with or without any precomputed result. Several efficient heuristics exist for on-the-fly assignment of tasks of applications i.e. without using design-time results at runtime [2] [3] [4]. However, tasks schedulability and high-quality mapping may not be ensured by these heuristics due to limited online processing. Hybrid resource allocation strategies overcome these bottlenecks. By selecting a predetermined allocation solution and applying at runtime based on the system state, it achieves an efficient dynamic allocation/scheduling. Further, these approaches aid in design of light-weight platform manager for runtime resource assignment. However, most of the existing hybrid task allocation approaches [5] [6] explore allocations considering single task per PE mapping. Such strategies are inadequate for high performance multicore systems which support multiple tasks based on its memory [4]. The solution space in such scenarios is larger compared to multicore systems with single task allocation per processor. Few of the hybrid strategies for dynamic task assignment target multitasking platform but only solve task mapping [7] problem. The challenges of task and communication scheduling are not addressed with renders them unsuitable for dynamic allocation of tasks of real-time applications. Such applications often have dependent task-sets. Once a task completes, it communicates its output data (indicating communication load) to other dependent task(s) for their execution.

In this work, we present an improved hybrid strategy for adaptive task allocation and performance optimization of real-time applications on NoC based multicore systems with multitasking PEs while considering application dynamism at runtime. We address the joint issue of runtime allocation/scheduling of tasks and communication transactions. The salient features of our work are as follows:

- Propose an improved technique for allocation solution exploration at design-time for NoC based multicore systems with multitasking processors.
- Propose an efficient strategy to rapidly identify and adapt the predetermined allocation configurations for satisfying task deadline and reducing communication cost.
- Present an online heuristic for dynamic allocation of task and communication transactions of real-time applications

with link-contention awareness.

II. System Model

We consider a tile based multicore architecture where each tile consists of PEs, routers and links. It is assumed that each tile in the system is homogeneous and consists of general-purpose processors. The tiles are interconnected to each other using on-chip network in 2D mesh topology. The system model is shown in Fig. 1. Each PE consist of a processor

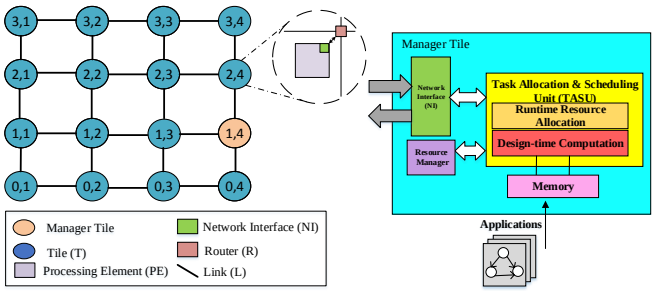

Fig. 1: System model of NoC based multicore platform.

control unit, data-path and memory unit having task and data memory. We have considered a dataflow computation model where PEs communicate by message passing. Each PE can support a maximum of PE_{cap} number of tasks based on its memory. The platform resources are managed by a special-purpose *Manager Tile* (MT) hosting the real-time operating system (RTOS) supporting non-preemptive, multitasking and event-based environment. The task allocation/scheduling algorithm is executed by the Task Allocation and Scheduling Unit (TASU) which decides the PE for task execution. The occupancy status of each PE and link are updated in Resource Manager (RM). The allocation configurations obtained at design-time are stored in offline repository present in memory of MT and fetched by TASU. We have assumed that services for loading task codes onto PEs and status monitoring of PEs and links already exists on the given platform which use a small part of NoC bandwidth or another NoC similar to [2].

The routers use wormhole packet switching to transmit the packets in a pipeline fashion. We have considered that deterministic routing (XY or YX routing) is used owing to its low cost of online implementation. In this work, we have assumed virtual channels (VC) based routers, where each input port consists of two VC. Class0 VC is dedicated for communication using XY policy whereas class1 VC is used by packets routed by YX policy. We have considered each packet has 8 flits with 32 bits/flit. The size of input buffer is 32 bits with buffer depth of 4.

III. Problem Formulation

An application task graph, $A = (T, \mathcal{E})$ is denoting as a directed acyclic graph (DAG), where each entity $\tau_i \in T$ denotes the set of tasks and $e_{k,l} \in \mathcal{E}$ is the set of directed edges, representing communication between tasks. Each task τ_i is characterized by worst-case execution time ex_i and deadline dl_i. sl_i is the slack-time which is the time-margin between the time at which τ_i would complete if it started now, and its deadline time. For an edge $e_{k,l}$, $\delta_{k,l}$ indicates the edge-weight and represents the communication volume in number of packets to be transferred.The NoC topology graph is a directed graph $N = (\psi, L)$ where $PE_i \in \psi$ represents a PE in the topology. Each $l_{i,j} \in L$ represents physical link connecting the routers for processors PE_i and PE_j.

A set of consecutive links connecting a pair of PEs as per a given routing policy is called a communication route, $\mathcal{R}$. The time delay in transferring a packet through a link is termed as a time slot. The set of time slots during which $e_{i,j}$ occupies the kth link $l^k_{m,n}$ on its route is called link occupation window $LOW^{l^k_{m,n}}_{e_{i,j}} = \{\ p \mid \nu + (k-1) \leq p \leq \nu + (\delta + k - 1)\}$ where ν is the time slot when the first packet is communicated. The link utilization factor, $LUF^{l_{m,n}}_{e_{i,j}}$, gives the time slots over which contention occurs for a link $l_{m,n}$ for transmitting on edge $e_{i,j}$. it is computed as the overlapping time-slots between $LOW^{l^k_{m,n}}_{e_{i,j}}$ and the set of all time-slots over which $l_{m,n}$ is occupied. Using this information, the route utilization factor (RUF) of route $\mathcal{R}$ is defined as:

$$RUF(\mathcal{R}) = \frac{1}{|\mathcal{R}|} \sum_{l_{m,n} \in \mathcal{R}} LUF^{l_{m,n}}_{e_{i,j}} \tag{1}$$

A route with low RUF is desirable as it indicates less contention. The assignment of the task graph on a bounded set of processors ψ is denoted by $map : T \longmapsto \psi$ such that $map(\tau_i) = PE_j$ indicates allocation of τ_i on PE_j. The quality of the mapping is defined in terms of the communication cost (CC) given by Eq. 2:

$$CC = \sum_{\forall e_{i,j} \in \mathcal{E}} \delta_{i,j} * Hop_Count(map(\tau_i), map(\tau_j)) \tag{2}$$

A schedule $\mathcal{S}$ is regarded as the start-time decision of the tasks and communication transaction and defined by function pair (ϕ_t, ϕ_e). $\phi_t(\tau_i, PE_j)$ and $\phi_e(e_{i,j}, l_{x,y})$ give the start-time of execution of τ_i mapped on PE_j and start-time of transmission of $e_{i,j}$ on link $l_{x,y}$. For an application A, we define its operating configuration $C_A = \langle \rho_{size}, S_\rho, CC, A^{ft} \rangle$. ρ_{size} is the number of PEs used for executing the tasks of A. For a multicore platform with processors having multitasking capability, PE_{cap}, $\rho_{size} = [\ |T|/PE_{cap}\ ,\ |T|\ ]$. The spatial distribution of the allocated PEs results in various shapes of the allocation region indicated by S_ρ. A^{ft} represents the finish-time of execution of application A.

A. Problem Statement

Design-time sub-problem: Given a finite set $\mathcal{G}$ of known applications each of which is represented by a DAG $A = (T, \mathcal{E})$ and a topology graph $N = (\psi, L)$, our objective is to ***determine*** the operating configuration $C_A = \langle \rho_{size}, S_\rho, CC, A^{ft} \rangle$ for each application ***such that*** tasks satisfy their deadline and communication cost of allocated application is reduced.

Runtime resource allocation subproblem: For the set of arrived applications $\mathcal{F} = A_1, A_2 ... A_n$ where $F \subseteq \mathcal{G}$, our objective is to ***determine an adaptive resource allocation***

for each application to find the operating configuration $Q = \langle C_{A_1}, C_{A_2} ... C_{A_n} \rangle$ and schedule $\mathcal{S} = (\phi_t, \phi_e)$ with intra-application PE sharing and inter-application link sharing ***such that*** the finish-time of applications is reduced and increase in communication cost at runtime is low while mitigating link-contention during communication transactions of applications.

IV. Proposed Approach

Algorithm 1: *Adaptive Resource Allocation for Tasks and Communications* $(ARA-TC)$

Input : Application $G(T, E)$, Topology Graph $N(\psi, L)$
Output : Runtime allocation/scheduling of tasks and communication

1 $\rho_{size}^{max} = |T|$; $\rho_{size}^{min} = |T|/PE_{cap}$;
2 $Proc_sel \leftarrow \emptyset$; $Link_sel \leftarrow \emptyset$; $RegionShape_list \leftarrow \emptyset$;
/* Design-time Preparation stage */
3 **for** *i*= ρ_{size}^{min} *to* ρ_{size}^{max} **do**
4 $\quad RegionShape_list = Selective_Region_Shape_Generation(\rho_{size}, \psi)$;
5 **end**
6 **for** *each shape* $S_i \in RegionShape_list$ **do**
7 $\quad AT^j = Discrete_Particle_Swarm_Optimization(T, S_i)$
8 **end**
/* Online Customization of Allocation */
9 $Proc_Avl$=$get_available_PE()$;
10 $Avail_size$=size of $Proc_Avl$;
11 **for** $\rho_s = Avail_size$ *downto* ρ_{size}^{min} **do**
12 $\quad$ **for** *each allocation template* AT^j *with size* ρ_s **do**
13 $\quad\quad (Proc_sel, Link_sel, STATUS) = Online_Allocation_Adapt(AT^j, Proc_Avl)$;
14 $\quad\quad$ **if** *STATUS==TRUE* **then**
15 $\quad\quad\quad$ **goto** line 19;
16 $\quad\quad$ **end**
17 $\quad$ **end**
18 **end**
19 assign start-time of tasks on PE allocated from set $Proc_sel$;
20 assign start-time of edges on links allocated from set $Link_sel$;

Algorithm 1 presents the proposed adaptive resource allocation strategy for tasks and communications $(ARA-TC)$. In the offline stage, we determine a set of operating configurations consisting of allocation templates and different schemes of task execution times. Each template contains various assignment/scheduling decisions of tasks, resulting in different communication cost and finish-time of each application. It involves *spatial allocation configuration* and *allocation template formation* using the identified PEs. First, the algorithm invokes $Selective_Region_Shape_Generation()$ in line 4 to find various spatial allocation configurations. It first enumerates the allocation region size (ρ_{size}) for task execution for an application. The allocation region shape is grown selectively in an incremental manner by considering the neighbouring tiles during each intermediate level of region growth. The identical region shapes having the same pattern of PEs but with different orientations are excluded from further growth to generate unique region shapes. Next, the allocation templates (AT) are synthesized (line 7), wherein the allocation and scheduling of tasks to PEs present in each generated region is determined. In this work, this is accomplished by employing well-known discrete particle swarm optimization (DPSO) [8]. We have modified the basic DPSO to solve task allocation and scheduling for multitasking PEs. An array of integers indicating id of PEs present in allocation region is used to represent a particle p_i. In particle formulation, we allow PE ids to be repeated in an array based on the task capacity of processors. For p_i its fitness function, $ff[p_i]$ is considered as $ff[p_i] = CC[p_i] * B_i$ where binary variable $B_i = 1$ indicates the task allocation given by the p_i is schedulable i.e. satisfies its deadline, else $B_i = 0$. The particles with small non-zero fitness value are preferable. In this stage, table-based routing is used to confine the packets to routers present within the allocation region.

To address the varying availability of PEs and different execution time of arrived tasks, we propose an improved online scheme shown in lines 9-20 of Algorithm 1, which customizes the design-time decisions. First, the function $get_available_PE()$ prepares the set of available PEs, $Proc_Avl$ as shown in line 9. Then, the precomputed allocation regions are explored in descending order of their sizes (lines 10-11). This is iterated until a suitable region size is found which fits into the available set of PEs. Such a runtime policy prefers allocation templates which results in the least value of finish-time. Among the tasks ready for execution, EDF strategy is used for selecting tasks for assigning on PEs. The PEs having available task capacity and located in the close vicinity of PE hosting the most communicating parent task are considered. Such a PE is chosen if its earliest available time is less than slack-time margin of the task to be allocated. The start-time of task execution is then assigned to the identified PE. This process of online selection and adaptation of design-time results is effected by function $Online_Allocation_Adapt()$ using a heuristic strategy (line 13). It involves fitting the precomputed allocation templates to the PE distribution by reorienting the allocation regions or customizing the allocation region most similar to the available set of PEs. However, during the above runtime reconfiguration approach, the network links associated with the selected PEs may be already occupied by data packets of other communicating tasks leading to traffic contention where multiple real-time applications can be concurrently active. To mitigate link contention, the online strategy first evaluates the RUF of route $\mathcal{R}$ (refer Eq.1) determined using XY routing policy between the source and destination PEs. If the route has zero RUF, then the $\mathcal{R}$ is assigned for data packets. Else, the RUF for the route obtained by YX routing is evaluated. The proposed runtime approach selects a candidate PE in close vicinity which has RUF zero or least non-zero value. This ensures that links with low/no contention are selected for communication traffic at runtime.

V. Results and Discussions

To evaluate the effectiveness of the proposed approach, a C++ based simulator is used, which is based on [9] [1] and modified to implement the proposed adaptive task allocation strategy. We have considered a NoC platform of 8×8 size in simulations considering both real-time and synthetic applications. Real-time applications such as 263enc, 263dec, MPEG and PIP [10]. Synthetic graphs are generated using TGFF [11] tool, such as TGFF(1-4) consisting of 25, 35, 45 and 55 tasks respectively. Simulations have been conducted

on 1000 test scenarios generated randomly by combination of 100 applications each consisting of [20 100] tasks and [50 150] edges. For the tasks, the execution time requirement is uniformly distributed between 50 to 500 clock cycles and their deadlines are assigned. Next, we discuss the experimental results on the comparison of the performance of the proposed runtime allocation/scheduling approach with other competitor algorithms for contention-aware dynamic allocation.

Fig. 2 compares the communication cost of the task allocations resulting from various runtime approaches. The results are normalized with respect to CA-NN [3] method considered as baseline. The applications allocated by the proposed approach achieves 23.6% and 35.3% improvement in communication cost on an average when compared with the allocations given by PL [12] and DTMCS [13] strategies respectively. This is because the proposed dynamic resource allocation strategy exploits better mapping decisions taken at design-time for applying in runtime. This results in contiguous task allocation with reduced sharing of on-chip links.

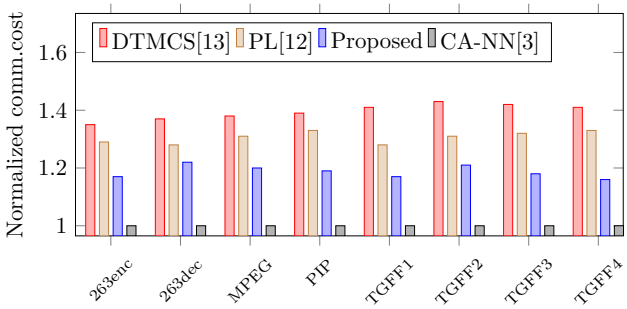

Fig. 2: Comparison of communication cost.

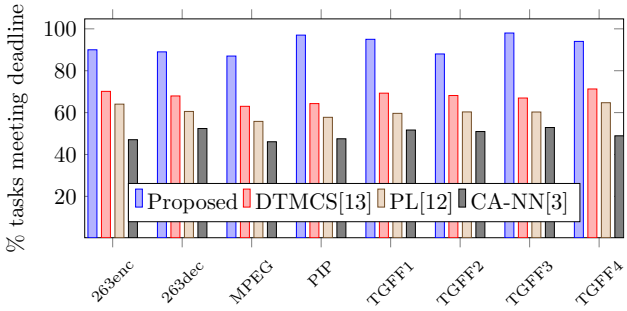

Fig. 3: Deadline performance of various algorithms.

Fig. 3 shows the results of deadline performance. On an average, 31.2% more tasks satisfy their deadline using the proposed runtime allocation algorithm when compared with the allocations obtained by DTMCS. When compared with PL and CA-NN algorithms, the proposed runtime approach results in 45.2% and 51.7% more tasks completing execution within their deadline, respectively. A comparison of the average communication latency is shown in Fig. 4. The cycle accurate simulator *Noxim* [9] is used with settings as described in Section 3 to evaluate the network performance. It is observed that, on an average, the proposed algorithm reduces the communication latency of the mapped applications by 26.8%, 34.1% and 41.6% when compared with DTMCS, PL and CA-NN algorithms respectively.

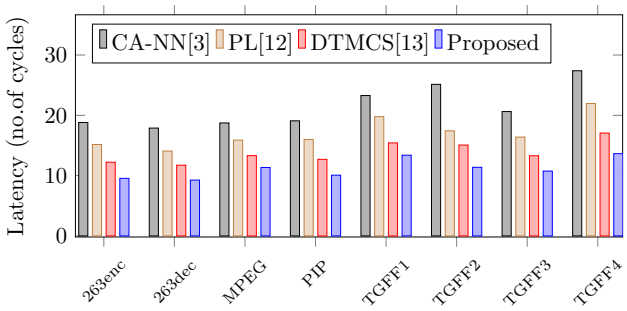

Fig. 4: Comparison of average network latency.

Experimental results also demonstrate that the proposed online allocation customization algorithm has low run-time overhead compared to the state-of-the-art contention-aware approaches. The results could not be presented here due to the paucity of space.

VI. Conclusion

In this work, we have presented a contention-aware hybrid allocation and scheduling strategy for NoC based multicore platform with multitasking processors. Compared with recent communication-ware dynamic allocation algorithms, the proposed runtime resource allocation strategy shows improved performance in latency, task deadline satisfaction and communication cost.

References

[1] N. Chatterjee *et al.*, "Deadline and energy aware dynamic task mapping and scheduling for network-on-chip based multi-core platform," *J. Syst. Archit.*, 74 (2017): 61–77.

[2] C. L. Chou and R. Marculescu, "Run-time task allocation considering user behavior in embedded multiprocessor networks-on-chip," *IEEE TCAD*, 29.1 (2010): 78–91.

[3] A. Singh *et al.*, "Communication-aware heuristics for run-time task mapping on noc-based mpsoc platforms," *J. Syst. Archit.*, 56.7 (2010): 242–255.

[4] T. Maqsood *et al.*, "Dynamic task mapping for network-on-chip based systems," *J. Syst. Archit.*, 61.7(2015): 293–306.

[5] G. Mariani *et al.*, "An industrial design space exploration framework for supporting run-time resource management on multi-core systems," DATE'10: 196–201.

[6] W. Quan and A. D. Pimentel, "A hybrid task mapping algorithm for heterogeneous mpsocs," *TECS*, 14.1 (2015): 14.

[7] L. Ost *et al.*, "Power-aware dynamic mapping heuristics for noc-based mpsocs using a unified model-based approach," *TECS*, 12.3 (2013): 75.

[8] P. K. Sahu *et al.*, "Application mapping onto mesh-based network-on-chip using discrete particle swarm optimization," *IEEE TVLSI*, 22.2 (2014): 300-312.

[9] V. Catania *et al.*, "Noxim: An open, extensible and cycle-accurate network on chip simulator," ASAP'15: 162–163.

[10] N. Chatterjee *et al.*, "Fault-tolerant dynamic task mapping and scheduling for network-on-chip-based multicore platform," *TECS*, 16.4 (2017): 108.

[11] R. P. Dick *et al.*, "TGFF: Task graphs for free," CODES/CASHE'98: 97–101.

[12] E. Carvalho and F. Moraes, "Congestion-aware task mapping in heterogeneous mpsocs," in *2008 International Symposium on System-on-Chip*, Nov 2008, pp. 1–4.

[13] H.-L. Chao *et al.*, "Dynamic task mapping with congestion speculation for reconfigurable network-on-chip," *TRETS*, 10.1 (2016): 3.