
Article:
Mendis, Hashan Roshantha, Audsley, Neil Cameron orcid.org/0000-0003-3739-6590 and Soares Indrusiak, Leandro orcid.org/0000-0002-9938-2920 (2017) Dynamic and Static Task Allocation for Hard Real-time Video Stream Decoding on NoCs. LITES: Leibniz Transactions on Embedded Systems. 1. pp. 1-25. ISSN 2199-2002

https://doi.org/10.4230/LITES-v004-i002-a001

Dynamic and Static Task Allocation for Hard Real-time Video Stream Decoding on NoCs

Hashan R. Mendis, Neil C. Audsley, and Leandro Soares Indrusiak

Real-Time Systems Group, Department of Computer Science, University of York, UK
{hrm506, neil.audsley, leandro.indrusiak}@york.ac.uk

Abstract

Hard real-time (HRT) video systems require admission control decisions that rely on two factors. Firstly, schedulability analysis of the data-dependent, communicating tasks within the application need to be carried out in order to guarantee timing and predictability. Secondly, the allocation of the tasks to multi-core processing elements would generate different results in the schedulability analysis. Due to the conservative nature of the state-of-the-art schedulability analysis of tasks and message flows, and the unpredictability in the application, the system resources are often under-utilised. In this paper we propose two blocking-aware dynamic task allocation techniques that exploit application and platform characteristics, in order to increase the number of simultaneous, fully schedulable, video streams handled by the system. A novel, worst-case response time aware, search-based, static hard real-time task mapper is introduced to act as an upper-baseline to the proposed techniques. Further evaluations are carried out against existing heuristic-based dynamic mappers. Improvements to the admission rates and the system utilisation under a range of different workloads and platform sizes are explored.

2012 ACM Subject Classification On-chip resource management
Keywords and phrases Real-time multimedia, task mapping, network-on-chip
Digital Object Identifier 10.4230/LITES.xxx.yyy.p
Received Date of submission. **Accepted** Date of acceptance. **Published** Date of publishing.

Editor LITES section area editor

1 Introduction

Current multiprocessor System-on-Chip (MPSoC) platforms (many-cores) have tens or hundreds of processing elements (PEs) and often need to support an increased number of applications simultaneously. Network-on-chip (NoC), interconnects have emerged as the promising solution for communication infrastructure of such systems, due to its efficiency and scalability [7]. Future technologies with streaming multimedia applications form a large portion of the application space that exploit these highly distributed on-chip architectures [33]. The processing load imposed by computation-intensive applications such as video decoding can be partitioned into tasks and distributed among multiple PEs on the many-core platform, to improve metrics such as performance, utilisation, energy and to meet timing constraints.

This work looks at the resource-aware embedded systems design problem from the software perspective. Modern MPSoCs have many on-chip resources, such as PEs, communication channels and main memory controllers, which have to be allocated to applications to optimise on high-level metrics such as latency, utilisation and power consumption. Efficient task allocations can reduce NoC usage, and network contention; thereby reducing the response time of the task set and communication energy consumption. These allocations are generally performed at the software level, by a resource manager. On the other hand, from a hardware viewpoint, the NoC communication bandwidth can be reduced (e.g. reducing the link width or frequency) to maximise the NoC utilisation and save area and power consumption. However, this work focuses solely on

software-based runtime task mapping optimisation, as it assumes the hardware implementation platform is fixed and it also offers a less costly and more flexible solution.

The nature of allocating and scheduling application tasks to PEs is considered an NP-hard problem. *Design-time (offline/static) mapping* techniques that have a global view of the system and workload, can be used to optimize the task mapping process at system design-time, such that system resources are efficiently utilised. The complexity and workload characteristics of video decoding applications depend greatly on the temporal and spatial variations in the video stream; hence, the workload is highly dynamic and unpredictable. When dealing with live video processing applications, certain critical application properties are unknown at design-time. Therefore, these highly varying workloads require *runtime (online/dynamic) mapping* techniques, based on light-weight heuristics to allocate and schedule the tasks to PEs whilst the system is operational. Live video decoding systems have hard timing constraints that need to be guaranteed before admitting into the system for decoding. These systems often use deterministic video admission control strategies, which use worst-case timing behaviour of the tasks, resulting in under-utilised systems [24]. In [23], the authors show that by employing efficient dynamic, application and platform aware task to PE mapping strategies, the utilisation levels of hard-real time video decoding systems could be improved. In this work, the mapping approaches in [23] are further evaluated by comparing against an offline mapper and under different platform and workload conditions.

Violation of timing constraints in video processing systems can lead to degraded quality, but the system will continue to operate; hence multimedia systems are generally considered soft real-time (SRT). However, there exists a range of systems that depend on video streams that need to be processed with hard real-time (HRT) guarantees. For example, in vision-based robot control systems, accuracy and functionality of the feedback control systems depend on processing video frames with tight timing restrictions. Another example is in the telesurgery/teleoperation industry [14], where a doctor performs surgery on a patient without physically being in the same location. These safety-critical systems require responsive and reliable communication technology as well as hard real-time guarantees from the video processing systems to function safely. Furthermore, next generation automated video surveillance systems, will require processing and tracking objects in hundreds of video streams in real-time; missing deadlines in these systems would lead to reduced security and delayed response to threats.

Contributions: In order to address the aforementioned issues, we present the following novel contributions:

- A more *precise* definition of the application model [23, 24] used to represent decoding of multiple MPEG-2 video streams is presented, including its *execution and communication characteristics* (Section 3.1). Very few other work consider data parallel video decoding using a scalable, distributed memory, message-passing based communication model.
- A response-time analysis (RTA) based, application-aware, *deterministic admission controller* (D-AC) is presented in [24]. This work (in Section 4) describes, the underlying *D-AC process algorithmically*. Formal definitions of taking into account task graph precedence constraints within the RTA is also presented.
- We present *new evaluations* (Section 7) of the two application-specific, blocking-aware heuristic based runtime mapping approaches introduced in [23]:
  - We present an *upper-baseline* for evaluation - a genetic algorithm (GA) based static/design-time task mapping optimisation approach (Section 6), with a *novel fitness function* that considers video stream schedulability. This design-time mapper is compared against our proposed dynamic mappers and other existing runtime mappers.

- We present new experimental treatments: varying both the *platform size* and workload *computation-to-communication ratio*, which gives new insight to the strengths and weaknesses of each evaluated mapping approach.

The rest of this paper is organized as follows. Section 2 presents related work in task mapping and scheduling. Section 3 introduces the system models. Section 4 presents the deterministic admission controller. The proposed heuristic based runtime task mapping algorithms are described in Section 5, followed by the design-time static mapper in Section 6. Section 7 presents the experimental design and discusses the results. Section 8 concludes this paper.

2 Related work

Mapping of tasks onto PEs broadly falls under two categories: *static* and *dynamic* mapping. Static mappers (executed at design time) have a complete view of the application, workload and platform and attempt to find a suitable task to processor placement to optimise for different metrics such as execution time, throughput, resource utilisation and energy consumption [29]. For example, Butazzo et al. [6] proposes a static mapper that uses a branch-and-bound algorithm to partition and map a taskset with precedence constraints, to reduce the computational resources. However, they assume negligible communication cost between the tasks. Simulated-annealing based, offline, task and memory mapping for mixed-criticality NoCs have been introduced by Giannopoulou et al. [13]. Their optimisation technique accounts for the interferences on the shared memory and the NoC, however they assume a time-triggered NoC with static routes regulated at the source, which is contrary to our priority-based wormhole switching NoC architecture. The static mapper presented in this work uses a RTA and points-based fitness function which evaluates the schedulability of a video stream.

Dynamic mappers (executed at runtime) use heuristics to optimise certain metrics such as application execution time, energy consumption, temperature, reliability and resource utilisation [29]. Carvalho et al. [8] exploit the hop-distance and path load between cores, as a dynamic mapping heuristic to reduce communication packet latency, energy and channel occupation. This work was later extended by Singh et al. [30] to include PEs that can accommodate multiple tasks. Kaushik et al. [21] adapts these heuristics to balance both computation and communication, by using a pre-processing stage to achieve a balanced and reduced task-graph. There also exist hybrid mapping approaches (e.g. [27]), where design-time computed mapping templates are merged with runtime heuristic based decisions to reduce average power dissipation. This work focuses on dynamic mapping strategies that exploit both the application and architecture characteristics.

Ditze et al. [9] presents an extension to the least-laxity-first scheduling algorithm to schedule the MPEG decoding tasks and an admission controller that enforces QoS constraints of parallel video streams. However, their feedback based admission controller does not fully guarantee the schedulability of admitted video streams, but attempts to reduce over-reservation of system resources. Bamakhrama et al. [4] presents an analytical framework to determine the minimum number of processors required to schedule a set of streaming applications with given I/O rates, while guaranteeing the maximum achievable throughput. Similarly, in [1] the critical-paths of task-graphs are mapped onto the PEs first, to reduce the average end-to-end worst-case execution time of a set of streaming applications, such as MP3 and H.263 decoders. They use utilisation tests to determine mapping feasibility and consider the inter-core communication cost as a constant cost per hop on the 2D mesh interconnect. Contrarily, we model the interference patterns of the task communication message flows on a predictable, pre-emptive interconnect. Utilisation based schedulability tests, have been used to guarantee timing properties of streaming applications, when using a contention-free TDMA based communication interconnect [15].

However, inefficient resource reservation in TDMA interconnects lead to unused bandwidth and connection setup overheads limits scalability [12]. In recent work by Dziurzanski et al. [10], RTA based schedulability tests have been used to determine if a taskset is schedulable on a processor. RTA tests take into account the interference caused by higher priority tasks and flows. However, using RTA based tests online, is expensive and in [10], they perform approximate tests to reduce the overhead and use a feedback-based control-theoretic approach to reclaim slack in order to improve admission rates. Similarly, in [25], tasks have been mapped to PEs that have the highest amount of average slack, where the PE slack values are periodically monitored and sent to a global manager. They state that a trade-off has to be made between NoC communication load imposed by slack monitoring and the monitoring frequency which can affect mapping results.

Contrary to many of the discussed related work, in our study we model a homogeneous multicore platform connected via a scalable, network interconnect with priority-based arbitration (similar to QNoC [5]); which makes it easier to predict worst-case network contention scenarios. Furthermore, unlike in existing approaches we do not use any monitoring feedback at runtime, which results in no communication overhead in the resource management technique. To the best of our knowledge, the runtime NoC resource management techniques (i.e. runtime task mapping and admission control), proposed in this work are the only ones aimed at HRT video decoding, to consider both the interference caused by task blocking and to exploit known video stream properties.

3 System model and problem formulation

3.1 Application model

This section outlines the multi-stream video decoding application model, with focus on how the abstract workload is generated. Frames in each video stream, can be of type: I (Intra), P (Predictive) or B (Bi-directional) encoded; i.e. *frame type* denoted $f_t = \{I, P, B\}$. We assume parallel decoding of *multiple* MPEG-2 decoded video streams, with a fixed, independent, group-of-pictures (GoP) structure of *IPBBPBBPBBBB* (decoding order). According to the MPEG-2 specification this 12 frame GoP structure is recommended to balance compression, facilitate reasonable random-access points in the stream and to manage error propagation [11]. Decoding an MPEG stream can be parallelised at different levels of granularity (GoP/frame/slice/macroblock-level) [22]. In our application model, we assume frame-level data parallel decoding, which does not involve stream instrumentation.

As shown in the system overview diagram in Figure 1a, the application model has a hierarchical structure. At the top most level are stream based workflows (W_i), each containing video streams VS_i with arbitrary number of N independent jobs (denoted by J_i). Each of the video streams will have varying resolutions ($res(VS_i)$ = frame height×frame width). A job, represents an MPEG GoP and are modelled as a directed acyclic graph (DAG) with a fixed dependency structure, as depicted in Figure 1b. Each node in the task graph (TG) represents a real-time frame decoding task τ_i and edges represent traffic-flows (*flows* for short), denoted as Msg_i, which are reference data that needs to be sent to one or more dependent tasks (also referred to as *child* tasks). A task's execution can only start *iff* its predecessor(s) (also referred to as *parent* tasks) have completed execution and their output data is available. As shown in Figure 1b, certain tasks in the TG can be executed in parallel (e.g. P_4, B_2, B_3) if all the precedence constraints are met.

A task τ_i is characterised by the following tuple: (p_i, t_i, x_i, c_i, a_i); where p_i is the fixed priority, t_i is the period, x_i is the actual computation cost in terms of execution time, c_i is the worst-case computation cost and a_i is the arrival time of the task τ_i. Tasks are sporadic, preemptive and have fixed priority. Individual task deadlines are unknown; however, each job is considered schedulable

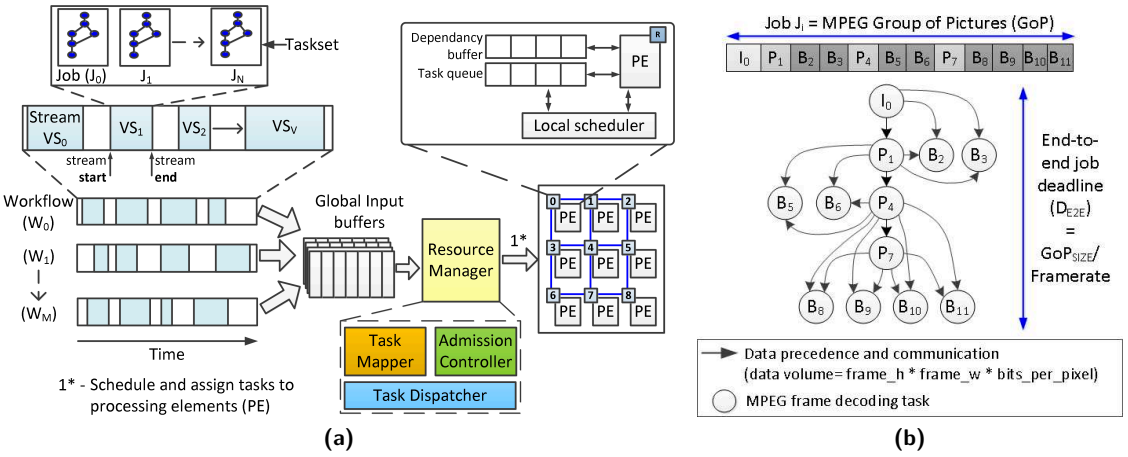

Figure 1 (a) System overview diagram; (b) MPEG GoP data precedence and task communication graph (Communication traffic between tasks and main-memory not illustrated)

if it completes execution on/before its end-to-end deadline ($D_{e2e} = |J_i| / fps$). fps denotes the frame rate of the video stream, which we assume is fixed at 25fps for all video streams. A task upon completing its execution sends its output (i.e. the decoded frame data) as a message flow to the PEs executing its child tasks (dependent task). A message flow, denoted by Msg_i is characterised by the following tuple: (P_i, T_i, PL_i, C_i); where P_i is the priority, T_i is the period, PL_i is the payload and C_i is the basic latency of the message flow Msg_i. It is important to note that, if one or more of a task's children are assigned to a single PE, then only one data flow is sent to the PE in order to avoid flow redundancy. Each task also has memory read and write flows which are not illustrated in Figure 1b. Flows inherit the t_i and p_i of the sender tasks and the PL_i of transmitting the reference frame data is $(res(VS_i) \times bits\ per\ pixel)$.

3.1.1 Deriving the task execution cost

The computation cost of decoding a video frame and the payload of reference data message flow, will greatly depend on the temporal and spatial variations of video streams. MPEG decoding consists of operations such as parse, decode, motion compensation (MC), inverse quantisation (IQ) and inverse discrete cosine transform (IDCT) etc. At the lowest granularity, MPEG contains *blocks* and based on how they are encoded they can be categorised into 9 types as explained by Tan et al. [31]. Depending on the type of frame and decoding steps performed on the frame, the frame type to block type relationship may vary, as shown in Table 1. Based on this information, we can model the frame execution cost as given in Eq. (2). Here, the w_j term denotes the weight of a type j block and the w_0 denotes the constant term in the regression model [31]. As per [31], the regression coefficients w_j are fixed for a given decoder regardless of the type/resolution of the video. We model the number of type j blocks (denoted M_j) as a uniform random variable between $\{0, \max M_j\}$, where $\max M_j$ as defined in Eq. (1), is the maximum amount of blocks for a given video resolution.

Figure 2 shows frame decoding time distributions of 200 jobs (GoPs) which were synthetically generated, as per the execution model described before. The distributions show that P/B-frames have a larger range than I-frames due to the lower amount of coding options used when decoding. Furthermore, I-frames on average take longer to decode because I-frames have a high number IDCT only blocks which have a higher weighting. These distributions correlate well with previous

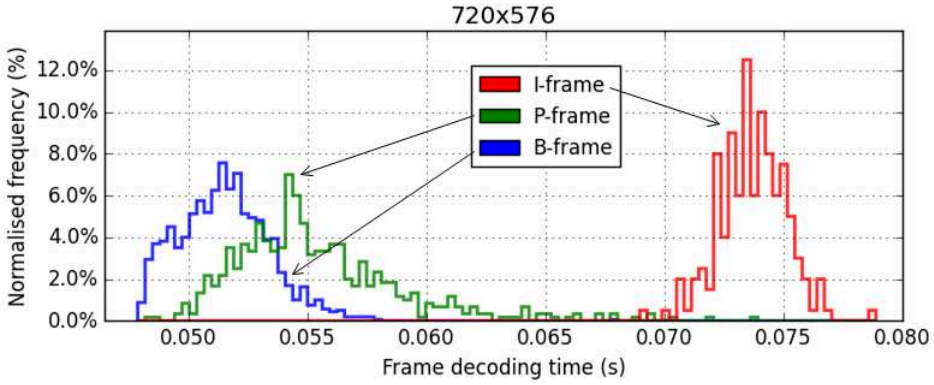

Figure 2 Frame decoding time distribution of synthetically generated workload (720×576 resolution video, 200 jobs)

MPEG real video stream decoding analysis seen in [19]. As defined in Eq. (3), the WCET c_i of a task of type f_t in a video stream is the *maximum* of all f_t type task's execution costs x_i; therefore, different frame types would have a different c_i. For example, in Figure 2, the I-frame decoding task WCET is $\approx 0.08s$, the P-frame decoding WCET is $\approx 0.07s$ and the B-frame decoding WCET is $\approx 0.06s$. We assume the actual execution cost x_i is unknown to the dynamic task mapping algorithm at runtime.

We emphasise that the task WCET that this work is considering assumes all data a task needs is available in the local memory at the start of its execution. The data required is provided by the reference data transfers from parent tasks and reading encoded data from main memory, both of which occur before the task execution. The communication latencies related to these data transfers are dealt with separately as part of the end-to-end response time calculation presented in the preceding sections.

$$\max M_j = \frac{res(VS_i)}{block\_size}, \text{ (0 if block } M_j \text{ not enabled in frame type)} \tag{1}$$

$$x_i = w_0 + \sum_{1 \le j \le 9} w_j \times rand(0, \max M_j) \tag{2}$$

$$c_i(f_t) = \max\{x_i \text{ of } \tau_i \in VS_i \mid \text{frame type} = f_t\} \tag{3}$$

Table 1 Relationship between MPEG block types and frame types (adapted from [31])

Frame type (f_t)	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9
I-frame	X								X
P-frame	X	X	X						X
B-frame	X			X	X	X	X	X	X

3.1.2 Task priority assignment

The task and flow priority assignment between different jobs of the same video stream do not change. The resource manager assigns a priority to each task in the first job (J_0) of the video stream upon admission. These priories are then fixed and the exact same priority values are used

for tasks in subsequent jobs of that stream. This is supported by the fact that each job in the stream has the same number of tasks and dependency structure.

Isovic et al. [18], introduces a quality and dependency-aware frame priority assignment which we have adapted to fit our 12-frame GoP sequence as given in Eq. (4). For a given video stream, task priorities are assigned according to Eq. (5), where tasks of low resolution video streams are given higher priority over high-resolution video streams. Here f_{ix} denotes the frame index within the GoP. This assignment ensures low-resolution video streams will have a lower chance of blocking resulting in lower response-times than high-resolution streams. The ($t_c \times$ offset) component ensures that unique job-level priorities and earliest arrival time is used to select between equal resolution video streams (t_c denotes current time). Flows inherit the priority of their source tasks. A higher p_i value denotes a higher priority.

$$GoP^{pr}_{fr} = \{12, 11, 4, 7, 10, 3, 5, 9, 2, 6, 1, 8\} \tag{4}$$

$$p_i = (res(VS_i) - GoP^{pr}_{fr}[f_{ix}]) + (t_c \times \text{offset}) \tag{5}$$

It is important to note that this work is not trying to solve the priority assignment problem or propose a new assignment scheme, for dynamic workloads. This work purely attempts to optimise on the task allocation, for a given priority assignment. We have selected a sensible assignment policy that reflects common practice in the video streaming industry. The proposed mapping heuristics can work around any other priority assignment approach.

3.1.3 Job arrival rate

Video stream start/end times are arbitrary. We assume the video streams have a variable bit rate (VBR), which means the video stream data will be arriving at the input of the system at a variable rate. In reality, the input could be more bursty in nature due to the variability in the transmission medium (e.g. the Internet). However, we assume that the VBR encoding has a user-specified upper bound related to the stream frame-rate; therefore the job arrival rate can be modelled as sporadic with a minimum inter-arrival time. The arrival rate of a job is therefore modelled as per Eq. (6) where J^{rate}_{min} and J^{rate}_{max} are workload parameters, usually set to 1.0 and 1.3 respectively. Decreasing J^{rate}_{min}, would increase the chance of new jobs arriving before the deadline of the previous job has passed, and increasing J^{rate}_{max} too much can make the system more idle. We assume all tasks of a job J_i arrive at the same time instant and hence, the period of all tasks and flows (i.e. t_i and T_i) within a job are equal and set to the minimum inter-job arrival time (i.e. 12/25=0.48s, for 25fps and 12 tasks per job).

$$\text{Arrival rate}(J_i) = rand(J^{rate}_{min} \times D_{e2e}, J^{rate}_{max} \times D_{e2e}) \tag{6}$$

3.2 Platform model

The multi-core platform is composed of P homogeneous PEs connected by a NoC. The NoC platform model uses wormhole packet switching, fixed priority preemptive arbitration, has a 2D mesh topology and uses the XY deterministic algorithm for routing such as in [5]. XY deterministic routing assures that the message flows will always use the same resources of the NoC (i.e. same path) to deliver data from a given specific source and destination. This property is a requirement of the flow response time analysis used by our deterministic admission controller. Therefore, any deterministic routing technique that guarantees a network path is compatible with the presented analysis. The NoC link arbiters are priority-preemptive thus making it easier

to predict the outcome of network contention for specific scenarios. All inter-PE and PE-to-memory communication occurs via the NoC by passing messages. Each PE contains a local memory, a priority-preemptive local scheduler, task queue and a dependency buffer. The PE upon completing a task's execution, transmits its output to the appropriate PEs dependency buffer. Tasks can begin execution on a PE *iff* all its data dependencies have arrived at the dependency buffer. If two tasks are mapped on the same PE it is assumed they do not need to communicate through channels of the NoC; hence the output of the source task will immediately be available at the dependency buffer of its PE. Higher priority tasks that have all dependencies fulfilled can interrupt already running lower priority tasks. Once a task finishes its execution the local scheduler picks the next highest priority task with dependencies fulfilled, to be executed. Similarly, higher priority flows can interfere lower priority flows that share the same network link.

The global input buffers are located in the main memory, and the task data (i.e. the encoded MPEG frame data) must be transmitted from the main memory via the NoC to the respective PE before execution (referred to as *memory read*). Additionally, after the MPEG decoding task has completed, its output (i.e. decoded MPEG frame data) is transmitted to the frame-buffer located in the main memory (referred to as *memory write*). The model assumes an encoded MPEG-2 I-frame (with $\approx 40\%$ frame compression), is twice as big as a P-frame and 4 times as big as a B-frame [19]. The encoded frame byte size and the decoded frame byte size represents the memory read and write traffic payloads respectively. We assume memory read flows have higher priority over data and memory write flows. The platform model assumes 4 main memory controllers (MMC), placed on the four sides of the NoC. A task communicates with the MMC closest to it.

3.3 Open-loop runtime resource manager

The resource manager (RM) of the system (Figure 1a), performs task mapping, priority assignment, admission control (AC) and task dispatching to the PEs. In our platform model we assume the RM is a separate entity/component, however it could also reside in one of the PEs. Task mapping and priority assignment is performed only once for a video stream at the admission of the first job J_0; all subsequent jobs of the video stream follow the initial mapping and priority assignment. The RM also maintains a *runtime mapping table* (TMT) of the jobs of every active video stream in the system. This mapping table contains the following task information:

- Real-time properties: $\{c_i, t_i, p_i\}$
- Non-real-time properties: $\{f_t, f_{ix}, \}$
- Task mapping: indicates which PE a specific task τ_i is mapped to.

The TMT is populated with each task of the first job J_0 of an admitted video stream. Once the video stream has stopped/finished, the task entries related to the stream are removed. The RM also has knowledge of the fixed TG dependency structure used by all video streams. The above TMT information is looked-up during the deterministic admission control and runtime task mapping operations. For example, the task mapping techniques proposed in this work, make use of the information in the TMT (e.g. task WCET, priorities and mapped PE) to determine the worst-case interference for a mapped task. Therefore, unlike in previous work [15, 30], our proposed resource management technique is open-loop and does not require any feedback/monitoring mechanism.

3.4 Problem statement

A deterministic admission controller (D-AC) decides whether to reject or admit a video stream, based on the schedulability of the new and existing video streams. This enables the system to give a hard real-time video stream decoding guarantee to the end-user. However, the D-AC tests result in under-utilised system resources, due to the pessimistic nature of the RTA [23, 24]. With proper task to PE mapping approaches, admission rates and system utilisation can be improved. This is challenging as certain workload characteristics such as execution time, arrival patterns and task and flow interferences are unknown a priori. We present heuristic based runtime mapping approaches that consider the current state of the PEs and the task and flow blocking behaviour in order to minimise communication and computation load of the processing elements. The goal is to develop task mapping heuristics that will lower the worst-case response-time (WCRT) of the video stream such that the D-AC will increase its admission rate, leading to better utilised systems.

4 Deterministic admission control

The deterministic admission controller (D-AC) is invoked when a new video stream request is received. It performs RTA to determine if any of the new or existing/active video streams would miss their end-to-end deadline, by admitting the new video stream. Algorithm 1 shows the steps involved in D-AC decision process. Firstly, upon arrival of a new video stream, the online task mapping heuristic is initiated to assign tasks and flows, processor and priority allocations (line 1). The task mapping details are then added temporarily to the TMT (line 2), to account for the additional resource contention incurred by potentially admitting a new stream. If the video is rejected, then the entries are removed (line 22). After the task mapping, the flows (and their real-time properties) resulting from the mapping can be generated (line 3).

With the information above, the calculation of the WCRT of tasks and flows of all video streams in the system can be initiated (lines 4-11 of Algorithm 1). Higher priority tasks and flows of one stream can block lower priority tasks and flows of other active video streams. A flow Msg_i can have two types of interference sources - *direct* and *indirect* interference. The direct interference flow set (denoted S_{id}) are higher priority flows that have at least one physical link in common with the observed traffic-flow. The indirect flow set (denoted S_{ii}) are higher-priority flows with no shared links with the observed flow but share at least one link with a flow in S_{id}. Eq. (9) given in [16, 28] is used to calculate the upper bound for the worst-case network latency R_i of each traffic flow (Msg_i) in wormhole switching, fixed priority preemptive NoCs. The task WCRT (r_i in Eq. (7), originally introduced in [3]) is used as release jitter J_i^I of message flows. In Eq. (7), $hp(\tau_i)$ is the set of higher priority tasks mapped on the same PE and we have excluded the non-interfering task set $ni(\tau_i)$ due to the known precedence constraints. The basic latency C_i of a message flow, calculated using Eq. (8), is the flow transmission latency when no flow contention is present. In Eq. (8), $Hops$ is the number of hops between source and destination, RL is the link traversal time, $numFlits$ is the payload size and HL denotes the time needed to route a packet header. In Eq. (9), we adjust S_{id}, such that non-interfering flows $ni(Msg_i)$ of flow Msg_i are excluded due to task precedence constraints. The terms t_j and T_j in Eq. (7) and Eq. (9) denote the task and flow periods respectively. The calculated WCRT of tasks and flows are saved in TMT to be used to calculate the WCRT of the video stream job.

$$r_i^{n+1} = c_i + \sum_{\forall \tau_j \in \{hp(\tau_i) \setminus ni(\tau_i)\}} \left\lceil \frac{r_i^n}{t_j} \right\rceil c_j \tag{7}$$

$$C_i = (HL \times Hops) + (RL \times (Hops - 1)) + (HL \times numFlits) \tag{8}$$

$$R_i^{n+1} = C_i + \sum_{\forall Msg_j \in \{S_{id} \setminus ni(Msg_i)\}} \left\lceil \frac{R_i^n + r_j + J_j^I}{T_j} \right\rceil C_j \tag{9}$$

Algorithm 1 Deterministic admission control pseudo-code

Input: TMT : Runtime task mapping table;
Real-time task properties of new video stream request VS_k
Output: Boolean AC-decision : admit/reject
// Perform mapping and derive resulting flow set
1: Map all tasks $\tau_q \in VS_k$ to NoC PEs and assign priorities.
2: Temporarily insert VS_k task mapping details to TMT
3: Derive all valid periodic flows in system (for above mapping configuration) : FT_{temp}
// Calculate WCRT of each task and flow in TMT
4: **for all** $\tau_i \in TMT$ **do**
5: Find interference set $hp(\tau_i)$ (exclude $ni(\tau_i)$)
6: Calculate task WCRT - Eq. (7), save value in TMT
7: **end for**
8: **for all** $Msg_i \in FT_{temp}$ **do**
9: Find interference sets S_{id}, S_{ii} (exclude $ni(Msg_i)$)
10: Calculate task & flow WCRT - Eq. (9) , save value in FT_{temp}
11: **end for**
// Calculate $WCRT(J_i^{CP})$ of all video streams
12: **for all** $VS_i \in TMT$ **do**
13: vs_p : init. structure for all simple paths of VS_i job
14: **for all** $paths \in job(VS_i)$ **do**
15: Calc. $path$ response-time $= \sum_{Msg_q \in path} R_q$; save to vs_p
16: **end for**
17: Critical path of VS_i job : $J_i^{CP} = \max\{vs_p\}$
// Check video stream schedulability
18: **if** $WCRT(J_i^{CP}) \leq D_{e2e}$ **then**
19: $ac\_decision = TRUE$
20: **else**
21: $ac\_decision = FALSE$
22: Remove VS_k details from TMT
23: **return** $ac\_decision$
24: **end if**
25: **end for**
26: **return** $ac\_decision$

In lines 12-24 of Algorithm 1, the WCRT of the critical path of the job $WCRT(J_i^{CP})$ is calculated. Recall that the video streams use a fixed job structure, hence there are fixed number of simple paths of the TG known a priori. For each path of a video stream job $job(VS_i)$ the summation of the WCRT of all nodes and edges is calculated (line 15); note that the WCRT of the source task is included in R_i as release-jitter r_j. The job critical path J_i^{CP} is the path with

the maximum accumulated cost (line 17). A video stream is granted admission, *iff* the expression given in Eq. (10) is true for the new and all active video streams in the system (lines 18-24 in Algorithm 1); this guarantees that the worst-case timing requirements of all existing and new video streams will be successfully met. It is important to note that the J_i^{CP} and $WCRT(J_i^{CP})$ are properties of a given task-to-PE assignment, hence task mapping is integral to the D-AC decision.

$$WCRT(J_i^{CP}) \leq D_{e2e} \tag{10}$$

4.1 Exclusion of non-interfering tasks and flows

Precedence constraints of the tasks are taken in to account when calculating the task and flow interference. For this analysis, it is assumed there is no overlap between consecutive jobs within the same video stream. The end-to-end RTA given in [16] assumes a synchronous pipeline execution mode, where multiple instances of the TG or portions of the TG (i.e. from a prior job in the same video stream) can be simultaneously executing in the system. In this work, we assume there is no overlap in execution between consecutive jobs of the same video stream. Hence, when deriving the $hp(\tau_i)$ of a task, we can exclude dependent/successor tasks. Likewise, when calculating the direct (S_{id}) and indirect (S_{ii}) flow interference sets the task precedences are taken into account to determine non-interfering flows.

We now formally present the non-interference set of tasks and flows with respect to precedence constraints. Within the application TG, there exists different simple paths (also referred to as *paths*). A simple path is a topologically ordered set of nodes and edges which does not have repeating vertices and are a subset of the TG. For example, the simple path ($P_1 \Rightarrow B_9$) consists of the frame decoding tasks $P_1, P4, P7, B9$ and the flows between them. We define two distinct simple path types *to* and *from* a node in the TG. The *ancestral simple path* (expressed as ($\tau_0 \Rightarrow \tau_i$)) consists of path from root node (τ_0) to the target node τ_i; the *descendant simple path* (expressed as ($\tau_i \Rightarrow \tau_{-1}$)) consists of the path from target node τ_i to any leaf node (τ_{-1}) in the TG. For example in the TG (Figure 1b), if we consider $\tau_i = P4$, then the nodes I_0 and P_1 will lie on the ancestral simple path, and the nodes P_7 and one leaf node $B_{8/9/10/11}$ will lie on the descendant simple path. Hence, the non-interference set of a task τ_i can be defined as per Eq. (11). Similarly flows in ancestral and descendant simple paths will not interfere with the target flow as given by Eq. (12). Here, the ($Msg_i \colon \tau_s \rightarrow \tau_d$) component denotes the target flow Msg_i and its source and destination tasks τ_s and τ_d respectively.

$$ni(\tau_i) = \tau_k \in \{\tau_0 \Rightarrow \tau_i \cup \tau_i \Rightarrow \tau_{-1}\} \tag{11}$$

$$ni(Msg_i \colon \tau_s \rightarrow \tau_d) = Msg_k \in \{\tau_0 \Rightarrow \tau_s \cup \tau_d \Rightarrow \tau_{-1}\} \tag{12}$$

5 Proposed runtime mapping approaches

5.1 Least worst-case remaining slack (LWCRS)

The difference between the task deadline and worst-case computation cost (i.e. task slack), is used as a primary metric when determining the task-to-PE mapping. The heuristic (Algorithm 2) takes into account the worst-case blocking factor introduced by $hp(\tau_i)$ to determine the PE that provides the least worst-case remaining slack (LWCRS) for a target task τ_i. The mappers make use of the information stored in the runtime TMT (as described in Section 3.3), to determine the worst-case blocking for a task. Algorithm 2, iterates through the provided *PE_list* and calculates the following for each task-to-PE mapping: $RemSlack_t$ - the worst-case *remaining* slack (WCRS)

Algorithm 2 *_get_PE_least_slack* pseudo-code - Find PE with the least worst-case remaining slack

Input: τ_i : target task; TMT : copy of the runtime task mapping table; *PE_list* : list of PEs to search
Output: tuple : (result PE, search result (boolean))

1: $PE\_packing = \{\}$
2: **for all** $PE_i \in PE_{list}$ **do**
3: // obtain following from TMT
4: Get $MPT(PE_i)$: tasks already mapped on PE_i
5: Get $hp(\tau_i), lp(\tau_i)$: high/low priority tasks in $MPT(PE_i)$
// get worst-case remaining slack (WCRS) to target task
6: τ_i^{slack} : task slack w.r.t estimated sub-task deadline
7: $RemSlack_t = \tau_i^{slack} - \sum_{\forall \tau_j \in hp(\tau_i)} c_j$
// get WCRS on low-pri mapped tasks
8: $RemSlack_{lp} = \{\}$
9: **for all** $\tau_j \in lp(\tau_i)$ **do**
10: $RemSlack_j = \tau_j^{slack} - \sum_{\forall \tau_k \in hp(\tau_j)} c_k$
11: Insert $RemSlack_j$ to $RemSlack_{lp}$
12: **end for**
// populate local data structure
13: **if** $RemSlack_t > 0$ and $\forall x \in RemSlack_{lp} | x > 0$ **then**
14: $PE\_packing\,[PE_i] = RemSlack_t + \sum RemSlack_{lp}$
15: **end if**
16: **end for**
// if none of the PEs in the list will provide a positive WCRS, then choose the PE with min. utilisation
17: **if** $\forall x \in PE\_packing | x > 0$ **then**
18: PE_j = index of MIN(PE_packing)
19: **return** (PE_j, FALSE)
20: **else**
21: $PE_j = $ *_get_PE_min_util*(TMT, *PE_list*)
22: **return** (PE_j, FALSE)
23: **end if**

of τ_i - taking into account blocking induced by $hp(\tau_i)$ (line 7); $RemSlack_{lp}$ - the WCRS for each of the lower-priority tasks already mapped on the PE ($lp(\tau_i)$), taking into account the (c_i) of τ_i (line 10-11). A weight ($w = RemSlack_t + RemSlack_{lp}$) is then assigned to all PE in *PE_list*(line 14); the PE with the lowest w provides the LWCRS. The heuristic attempts to find the PE mapping that will result in a tight temporal-fit, without missing the deadlines of τ_i nor any of the already mapped tasks. Individual task deadlines are unknown, hence they are estimated via the technique proposed in [20], where the cumulative remaining taskset slack is divided proportional to c_i. If a suitable PE with positive slack is not found, the algorithm will return the PE with minimum utilisation.

5.2 LWCRS-aware mapping

The LWCRS-aware mapping approach (Algorithm 3), makes use of the LWCRS utility function explained in Section 5.1 as well as tries to minimise distance between communicating tasks. The primary objective of the algorithm is to tightly pack (i.e in the temporal domain) each task of the

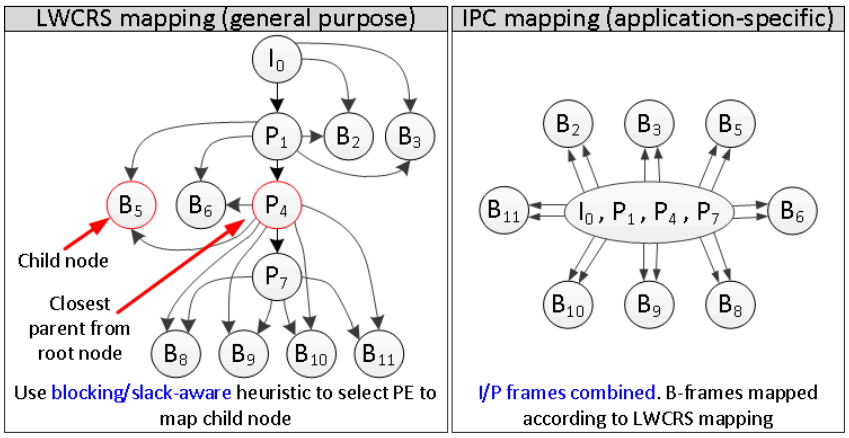

Figure 3 Illustration of LWCRS mapping closest parent selection *(left)* and IPC mapping task grouping *(right)*

job, into the PEs, in order to leave room for tasks of future video streams. The LWCRS mapper will ensure that initial PEs in the NoC will be heavily utilised before selecting the next available PE, whilst not violating the subtask deadlines. This increases the number of simultaneous video streams that the system can handle without missing any deadlines. LWCRS-aware mapping is a general purpose technique, that can be applied to map other types of applications that have dependency/communication patterns.

The algorithm of LWCRS mapping is given in Algorithm 3. The algorithm uses a copy of the runtime task mapping table TMT (maintained by the RM). The existing task-to-PE mappings and mapped task properties are stored in the TMT. Algorithm 2 is used within Algorithm 3, to find a PE which gives the LWCRS (lines 3 and 12). Firstly, the PE which gives the lowest slack is selected to map the root node of the TG (line 3). For all other nodes in the TG, the algorithm maps each node with an increasing hop distances distance from its *closest parent* (lines 9-17). A nodes' closest parent is defined as the node with the longest path from the root node. For example in Figure 3, B5 has 2 parents - P4 and P1; however P4 is the closest parent due to the longer path from the root node (therefore $\tau_{B5}^{PARENT} = P4$). If no suitable PE with remaining slack is found, the algorithm maps the target node to the PE with minimum utilisation (line 17-20). The algorithm attempts to reduce long-communication routes between communicating tasks in order to reduce network congestion and communication costs. Each node in the TG is mapped onto a PE that gives the LWCRS as well as close proximity to τ_i^{PARENT} (lines 6-15). TMT is updated in each iteration.

5.3 I and P frames combined mapping (IPC)

Unlike the LWCRS-aware mapper, IPC exploits known application-specific communication and dependency patterns. By inspecting the video job TG (Figure 3), we can see that the I and P frames lie on the longest-path in the TG. We define longest-path as the path in the TG with most number of non-repeating nodes (assuming edge weights are equal). Furthermore, the path $I_0 \rightarrow P_1 \rightarrow P_4 \rightarrow P_7$ is most often the critical path (CP) of a job, assuming B-frame decoding tasks do not experience severe blocking. Figure 3 (right) illustrates the TG after the grouping of I and P frames. Combining the I/P frames together has two distinct advantages : (a) it reduces the NoC congestion/interference as fewer flows need to be injected into the NoC; (b) it reduces the end-to-end response time of a job since the TG's potential CP is executed as soon as possible,

Algorithm 3 LWCRS-aware mapping heuristic algorithm pseudo-code

Input: all tasks in the job (J_0), TMT : copy of the runtime task mapping table
Output: task to processing element mapping : MPG ($\tau_i \rightarrow PE_i$)

for all unmapped tasks : $\tau_i \in J_0$ **do**
  if τ_i is *root_node* **then**
    (PE_i, $found$)=***__get__PE__least__slack***(τ_i, TMT, PEs)
    Map ($\tau_i \rightarrow PE_i$); Update TMT;
  else
    // obtain following from TMT
    Get τ_i^{PARENT} : parent task with max route cost.
    Get PE_i^P : PE that τ_i^{PARENT} was mapped onto
    // get neighbours of increasing hop_counts
    for $hc = 1$ to MAX_HOPS **do**
      $N\_PE_i^P = \_get\_neighbours(PE_i^P, hc)$
      Append PE_i^P into $N\_PE_i^P$
      (PE_i, $found$) = ***__get__PE__least__slack***(τ_i, TMT, $N\_PE_i^P$)
      if $found$ is $TRUE$ **then**
        Map ($\tau_i \rightarrow PE_i$); Update TMT;
        break loop; Go to step 1;
      end if
    end for
    // if no suitable PE is found, select closest min. util. PE
    if a suitable PE is NOT found **then**
      $N\_PE_i^P = \_get\_neighbours(PE_i^P, hop\_count = 1)$
      $PE_i = \_get\_PE\_min\_util$(TMT, $N\_PE_i^P$)
      Map ($\tau_i \rightarrow PE_i$); Update TMT
    end if
  end if
end for
return TMT

without waiting for message flows. The IPC mapping technique works as follows. Firstly, the I and P frame decoding tasks of the job are grouped and mapped to the lowest-utilised PE on the platform. The B-frame decoding tasks are mapped as close to their parent tasks with a 2 hop distance constraint. The LWCRS heuristic (Algorithm 2 described in Section 5.1) is used to select a PE within the 2 hop distance region. B-frames have no inter-dependencies, hence they can be processed in parallel. The B-frame decoding computation cost is lower than I/P frames, hence they can occupy smaller temporal gaps in the PE task queues.

6 Static hard real-time mapper

Static-mapping algorithms are used for multiprocessor systems when the application characteristics and workloads are known at design-time. To evaluate our heuristic based dynamic mapping techniques, we compare against a static search-based hard real-time (HRT) mapper as introduced by Sayuti et al. [26]. We adapt and modify this static mapper to suit our application and platform model. In this particular technique, the authors use a genetic algorithm (GA) based optimisation strategy to optimise the task to PE mapping approach. GAs have been used in the past to optimise task allocation for multi/many-core systems [2, 32]; however the work done in [26] was the first to explore task mapping of hard real-time tasks whilst optimising multiple objectives.

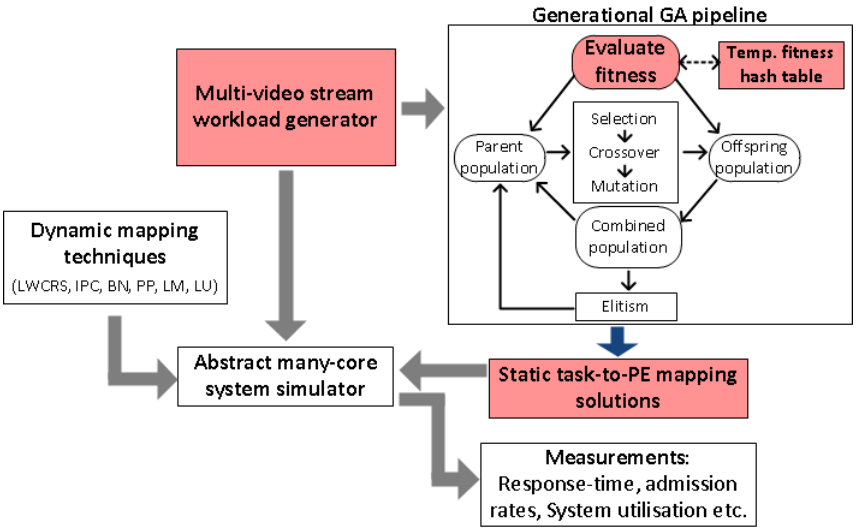

Figure 4 The GA pipeline from [26] adapted and integrated to the experimental design flow

GAs start with a random initial population of candidate solutions and gradually evolves the populations towards the global optimum using a given fitness statistic. In Sayuti et al. [26] the GA individual is represented by an integer-based chromosome structure indicating the PE mapping for each task. As illustrated in Figure 4, their algorithm uses simple evolutionary GA pipeline constructs such as single-point crossover, bit-flip mutation and binary tournament selection to generate a new population for each generation. They use elitism to ensure the best individual of each generation is advanced to the next generation. A multiple objective fitness function (application schedulability - Eq. (9) and power dissipation for every individual mapping solution) is used to guide a random search towards solutions of increasing fitness. It is important to note however that due to the random nature of solution development, GAs do not guarantee optimality even when it may be reached.

This static HRT mapper is purely used as an upper baseline, to evaluate the proposed heuristic based mapping techniques. If workload characteristics are known at design time, we can use a static HRT mapper to find a mapping configuration that will optimise our chosen metrics. This obtained experimental upper bound is valuable to assess the performance of the dynamic mappers results for a given workload. Dynamic mappers rely on fast heuristics and do not have complete knowledge of the workload at runtime; hence an indication of a good heuristic is one which shows results as close as possible to the upper-bound. However, recall that static mappers not only rely on full knowledge of the workload, but also incur a considerable runtime execution overhead. In this work we show via experimentation that even for small NoCs, under low workload conditions the static mapper will take up to several hours to find a reasonably optimised mapping configuration. The computation complexity of the fitness function will increase exponentially as the workload increases. For these reasons, GA-based static HRT mappers are not suitable for use in runtime mapping.

The GA-based HRT mapping optimisation given in [26] has been significantly, adapted to be able to integrate it with our application model and metrics. The red shaded components in Figure 4 indicate the changes made to the GA and the design flow with respect to the GA in [26]. In the experimental design flow, the same workload is first generated and input into the GA pipeline to obtain mapping solutions. The final GA mapping solutions are then taken and used in the system simulator to obtain performance measurements and compare against the mapping

results provided by the dynamic mappers. The following sub-sections outline the adaptations done to the GA design in [26].

6.1 Points-based GA fitness function

Unlike in [26] we are concerned with the end-to-end (E2E) schedulability of an entire video stream rather than individual tasks/flows. We use a novel points-based single-objective fitness function as described in Algorithm 4. The loop in lines 3-12 evaluates and accumulates the points for each VS_i in the workload (WL). Video streams that have a higher $WCRT(J_i^{CP})$ than their deadline are given a positive point based on the amount of which they have missed their deadline (i.e. $WCRT(J_i^{CP}) - D_{e2e}$). On the other hand video streams that are fully schedulable are awarded a negative point relative to their slack (i.e. $D_{e2e} - WCRT(J_i^{CP})$). The ratios in line 7 and 10 of Algorithm 4 indicates the extent to which a video stream is unschedulable/schedulable. Individuals with negative points have a higher fitness score than those with positive points. This points-based fitness scoring system would enable the GA to pick individuals with task mappings that have lower distributions of $WCRT(J_i^{CP})$. For example consider the following scenario. Two GA individuals, A and B have equal number of fully schedulable streams and one unschedulable stream each, but they have fitness scores -1.65 and -1.15 respectively. This indicates that As unschedulable video missed its deadline by a lower margin than Bs unschedulable video. This differentiation can not be made if an integer based fitness score is used, where both these individuals would be ranked equally.

Algorithm 4 Points-based GA fitness function

1: $points = 0$ //Calculate points for all videos in workload (WL)
2: **for all** $VS_i \in WL$ **do**
3: Calculate WCRT of all sporadic tasks and flows of VS_i
4: Find J_i^{CP} of VS_i
5: **if** $J_i^{CP} \geq D_{e2e}$ **then**
6: // unschedulable
7: $points \mathrel{+}= 1 \times \frac{WCRT(J_i^{CP}) - D_{e2e}}{D_{e2e}}$
8: **else**
9: // fully schedulable
10: $points \mathrel{+}= -1 \times \frac{D_{e2e} - WCRT(J_i^{CP})}{D_{e2e}}$
11: **end if**
12: **end for**
13: **return** *points*

6.2 Application-specific adaptations

In addition to the novel fitness function, we introduce certain extensions to the application model in [26] to accommodate our application specific task-model. For example, precedence constraints were taken into account when calculating the task/flow interference sets (Section 4.1). Memory read/write traffic has been incorporated such that for each mapping configuration (i.e. each chromosome) tasks-to-MMC selection (Section 3.2). Redundant flows are removed for each chromosome when multiple children are mapped to the same PE (Section 3.1).

6.3 GA design optimisations

Due to the extensions above, the fitness evaluation of the GA becomes more complex. To alleviate this issue, we limit the recursion depth of the WCRT analysis in Eq. (9). Furthermore, a hash table of task mapping solutions and corresponding fitness scores are maintained and looked-up to avoid having to evaluate the same gene more than once. Subsequently, the GA evolution cycle terminates immediately if it encounters an individual with an acceptable mapping solution. Such a mapping solution will result in all the admitted video streams to be schedulable; in other words, the maximum $WCRT(J_i^{CP})$ of all the video stream is less than the E2E deadline : $\max_{VS_i \in WL}(WCRT(J_i^{CP})) < D_{e2e}$

7 Evaluation

7.1 Experiment design

We wish to evaluate the performance of the proposed task mapping schemes in terms of admission rates and PE utilisation. Experimental evaluation is performed through a discrete-event, abstract simulation of a 3x3 NoC platform with the characteristics described in Section 3. The PEs are assumed to be operating at 200MHz and the NoC frequency is set to 10MHz with 7 clock-cycle routing latency and 16 byte link width. A lower NoC frequency (i.e. low bandwidth) is assumed, in order to induce an experimental condition with a reasonable amount of network utilisation/congestion. The NoC and the PEs use priority-preemptive arbitration and scheduling respectively. The light-weight NoC simulation component described in [17] is used to model the NoC communication traffic and interference patterns. The D-AC in Section 4 is used for all the simulation runs of this experiment.

Synthetic abstract video streams are used as workloads for all experiments as described by the application model in Section 3.1. The task execution costs are calculated using the block-level frame decoding cost model described in Section 3.1.1. All synthetic streams have a fixed frame rate of 25fps and 12 frame GoPs. The inter-arrival time of the video stream jobs are uniformly distributed between $1.0 \times D_{e2e}$ and $1.3 \times D_{e2e}$.

7.1.1 Varying workload

The total workload introduced to the system can be defined as a summation of the resolutions of all the video streams admitted and active in the system, as shown in Eq. (13). Mapping approaches for a range of workload values are evaluated; starting from a single video stream with 230×180 resolution to 9 parallel video streams with 720×576 resolution. Each experiment contains 30 simulation runs with different random seeds, which results in varying video stream arrival patterns and task execution costs.

$$\text{Total workload value} = \sum_{\forall VS_i \in WL} [frame\_h(v_i) \times frame\_w(v_i)] \tag{13}$$

7.1.2 Varying communication-to-computation ratio and NoC size

The mapping techniques explored in this paper use runtime heuristics such as communication path load, hop-distance and PE utilisation. Hence, it is interesting to explore the performance of these techniques, when the application communication-to-computation ratio (CCR) varies. For example, if an application is computation-bound (low CCR) then standard load-balancing heuristics may be sufficient. On the other hand if the application is communication-bound (high

CCR) then communication-aware heuristics may perform better. Several previous work in the state-of-the-art in dynamic task mapping [1, 21] does not consider the influence of varying CCR in their results, which may lead to biased results.

A single video stream can be represented by a sporadic TG as shown in Figure 1b. Thus, the CCR of a single video stream can be calculated as the ratio between the total cost of the communication edges over the total task cost in the TG, as shown in Eq. (14). Here, the communication basic latency and the task WCET are the communication and task costs respectively. The CCR of a workload can then be defined as the mean CCR of all the parallel video streams included in the workload (Eq. (15)). To change the CCR, we keep the task computation cost constant and gradually vary the NoC frequency. To evaluate the scalability of the proposed mappers we perform experiments under different NoC sizes and CCR combinations. Data from 30 uniquely seeded simulation runs are obtained. Each run consists of a fixed number of simultaneous video streams but with different arrival patterns and varying task execution costs.

$$\underbrace{CCR(VS_i)}_{\substack{\text{CCR of a single}\\ \text{video stream}}} = \frac{\text{Total edges cost}}{\text{Total nodes cost}} = \frac{\sum\limits_{\forall e \in edges} C_i}{\sum\limits_{\forall \tau_i \in J_i} c_i} \tag{14}$$

$$CCR_{WL} = \frac{\sum\limits_{\forall VS_i \in WL} CCR(VS_i)}{|VS|} \tag{15}$$

7.1.3 Metrics

For each simulation run we measure the video *admission-rates* and *PE and NoC busy times*. The admission rate is calculated as a ratio between the admitted video streams over the total video stream decoding requests. A D-AC ensures all admitted videos will be fully schedulable. The percentage PE busy time (also referred to as PE utilisation) is measured as the ratio between the total active (busy) time of all PEs in the system over the total simulation time. The percentage NoC busy time (also referred to as NoC utilisation) is the ratio between the total active time of the NoC links over the total simulation time.

The objective is to increase the admission rate of the system and thereby decrease the PE idle-time. Higher admission rates using lower NoC usage is advantageous because it can potentially lead to lower power consumption.

7.2 Baseline mapping heuristics

7.2.1 Dynamic Mapping Heuristics

The path-load based best-neighbour (BN) heuristic defined in [8], and the *pre-processing (PP)* based algorithm defined in [21] are used as baselines. The original BN algorithm was adapted to support multiple tasks and have used PE utilisation to determine available PEs, while maintaining path-load as the main heuristic. Since the dependency pattern of the tasks are assumed to be known beforehand, the pre-processing stage of the PP algorithm is performed at design-time. While PP takes into account both the communication and the computation properties of the tasks the BN heuristic focuses mainly on communication link congestion. Evaluation is also performed against two load-balancing task allocation heuristics which attempt to evenly distribute the load of the application across available PEs. The *lowest utilised (LU)* heuristic iterates through all tasks in the job and maps each task to the analytical lowest utilised PE. The worst-case utilisation of a PE is measured as given in Eq. (16), where $MPT(PE_i)$ denotes all tasks mapped on PE_i.

Since we do not know the actual execution cost of the tasks, we use the worst-case computation cost (c_i). At the end of each iteration the respective local mapping table is updated with the new task-to-PE mapping. Finally, the *least mapped (LM)* heuristic, selects the PE with the minimum number of mapped tasks according to the runtime task-mapping table.

$$U = \sum_{\forall \tau_i \in MPT(PE_i)} \left\lceil \frac{c_i}{t_i} \right\rceil \tag{16}$$

7.2.2 Static GA-based HRT mapper

The GA-based HRT mapper (GA-MP) described in Section 6 is used to act as an upper-baseline for our proposed task mapping techniques. The workload consisting of multiple video decoding streams are generated and fed into both the GA-MP and the abstract simulator (Figure 4). Task-to-PE mappings solutions obtained via the GA-MP are then evaluated in the discrete-event abstract simulator. The performance of the static mappings obtained from the GA-MP are then compared with the dynamic mapping strategies. The crossover and mutation probability rates are important parameters in the GA-pipeline and for all our experiments we use (0.5, 0.01) respectively. Higher workloads will be executed with a larger number of GA-evaluations to suit the increasing complexity of the search problem.

7.3 Discussion of experimental results

7.3.1 Dynamic mapper performance under workload variation

Figure 5 shows the performance of the different task-mapping heuristics in terms of mean admission rate and mean PE and NoC busy time, as the level of the workload is increased. Results obtained by using the dynamic task mappers (i.e. IPC, LWCRS, PP, BN, LM and LU) are first discussed. The admission rates for all mapping types decrease as the workload is increased, because for high workloads the AC cannot guarantee the timing requirements will be met, and hence rejections will be made. Using the IPC and LWCRS task mapping heuristics the WCRT of the jobs can be reduced, thus allowing the D-AC to admit a higher number of video streams than the baseline mappers. When comparing with the baseline heuristics, the proposed IPC and LWCRS dynamic mapping heuristics provide a 10%-20% improvement in low workloads, and an average of 5% improvement in high workloads, for admission rates. It is important to note that even though IPC outperforms LWCRS, IPC is an application specific heuristic which makes use of known characteristics of the video stream.

The PP heuristic outperforms the other baselines (BN, LM, LU) because it attempts to balance computation and communication by grouping the tasks in a job. The admission rates when using BN drops lower than LU and LM in the highest workload levels because the path-load heuristic alone is not sufficient. Higher admission rates result in more tasks being processed by the system, leading to increased PE utilisation as depicted in Figure 5(middle). However, the PE utilisation is a function of both the number of video streams admitted and their spatial resolution (e.g. 4 high resolution streams will yield a higher utilisation than 5 lower resolutions streams). With respect to the dynamic mapping heuristics, we can see that as the workload increases, the PE utilisation also increases; however in workload level 225.4×10^4 we see a decline because the admission rates are low as well as the admitted video streams are of a lower-resolution. The proposed IPC and LWCRS heuristics outperform the baseline dynamic mappers, except in the case where PP performs better than LWCRS at workload level 189.8×10^4, where the admitted video streams by

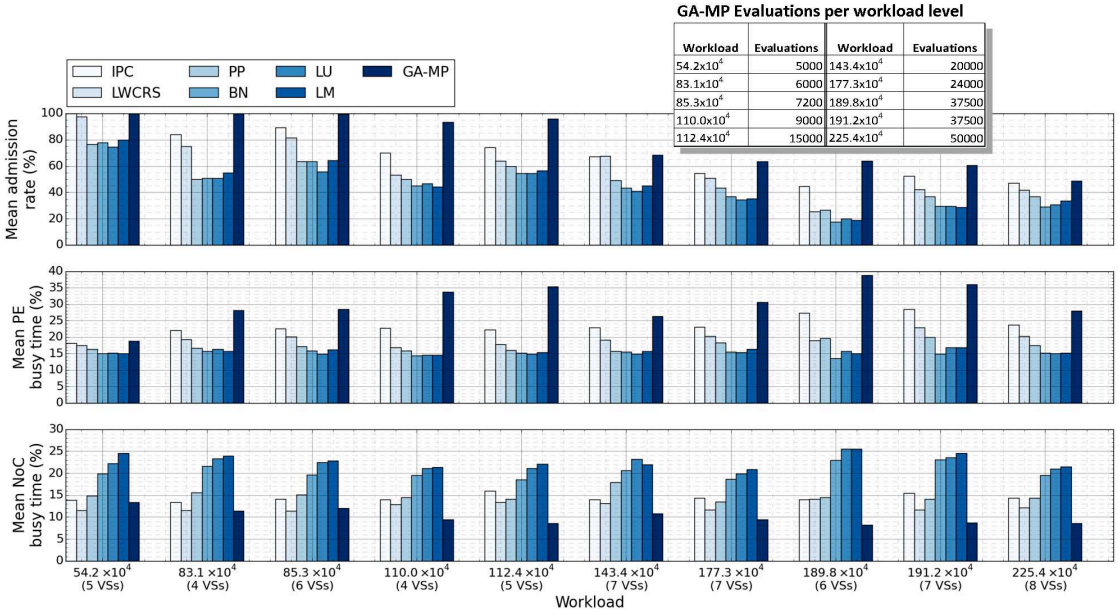

Figure 5 Workload vs. (*Top*: Admission rate; *Middle*: PE busy time; *Bottom*: NoC busy time)

PP is of higher resolution than LWCRS. Across the different workloads, the proposed dynamic mapping methods show an improvement of 5%-15% in PE busy time for workloads over 83.1×10^4

The NoC busy time results shown in Figure 5(bottom) complement the PE busy time results and offers further insight into the mapping behaviour. Lower PE busy times indicate the PEs are busy waiting for data to arrive at the local buffers, thus increasing the NoC usage. Out of the proposed dynamic mappers, IPC utilises the NoC more than LWCRS. LWCRS produces a tighter grouping of tasks than IPC resulting in lower number of PEs being used. Tightly grouping tasks reduces the number of data flows but does not reduce memory flows; therefore now memory traffic becomes a bottleneck, primarily congesting the local-link. Hence, LWCRS could still have a higher $WCRT(J_i^{CP})$, leading to reduced admission rates compared with IPC. In IPC, because 4 tasks in the job (i.e. I_0, P_1, P_4, P_7) are always mapped together, the algorithm will try to find other PEs to map the B-frame tasks. This leads to more PEs being used than LWCRS and thus a higher NoC busy time than LWCRS. LU, LM mappers have the highest NoC usage due to the sparse distribution of tasks on all PEs. PP shows similar NoC usage to IPC, but it does not consider blocking, hence, it might place the grouped tasks to PEs that might cause higher interference.

7.3.2 Static mapper performance under workload variation

In the Figure 5, GA-MP results denote the task mapping using the genetic algorithm based static hard real-time mapper. The number of GA evaluations taken to obtain these results are given in Figure 5(top-right table). GA-MP has full knowledge of the task characteristics and is used as an upper baseline. The trend of the results closely match the dynamic mappers (i.e. admission rates decrease and PE busy times increase as the workload is increased). Even though the GA-MP outperform all the dynamic mappers at every workload level, we notice a gradual decrease in relative improvement as the workload level increases (e.g. at workload 225.4×10^4 the IPC and GA-MP show comparable mean admission rates). Under certain conditions the GA-MP and the proposed dynamic mappers show similar results in admission rates, but the GA-MP

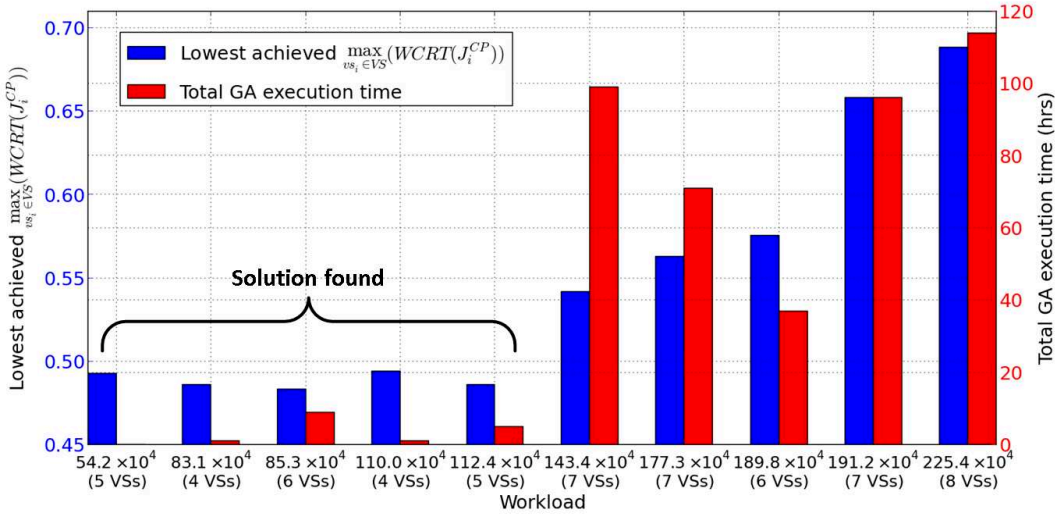

Figure 6 GA-MP performance analysis for different workload levels

has higher PE busy times (e.g. at workload 143.4×10^4 LWCRS show similar admission rates but poor PE utilisation). This is because in certain scenarios the GA-MP obtains a mapping which rejects lower resolution video streams but accepts higher resolution streams, thus giving rise to higher PE busy times. We noticed that GA-MP uses only a few cores per job (on average 2 or 3 cores). Furthermore, the GA tries to map children of the same parent together on the same PE, thus avoiding redundant data traffic flows. Due to these reasons, the GA-MP mapping significantly reduces the number of flows injected into the NoC. This reduces the NoC busy time, flow contention and therefore leading to higher admission rates.

As the workload increases, the number of the tasks and flows as well as their computation and communication costs increase; hence increasing the complexity of the optimisation problem. To compensate, the number of evaluations also needed to be increased to obtain a reasonable performance level. To illustrate this, the GA-based mapping optimisation is executed for the ten different workloads with a fixed generation and population size (500, 200 respectively). The search terminates when an acceptable mapping solution is found. Results (Figure 6) show that the total execution time (red bars) of the GA increases exponentially as the workload level is increased and all except the lowest workload level show an execution time in the order of tens of hours. Workload 189.8×10^4 shows a lower execution time, because the number of parallel video streams are 6 even though the workload value is higher. Note that there is a sudden increase in GA execution time from 112.4×10^4 to 143.4×10^4. This is because the number of video streams have increased from 5 to 7 and also because the GA does not terminate early, as no solution can be found for the higher workload within the fixed number of evaluations. Hence, both the number of parallel video streams and their resolutions are directly related to the GA execution time. The GA was able to find a satisfactory solution for workloads up to 112.4×10^4, however a solution was not found for the larger workloads. Similar to the execution time, we see that the lowest achieved $\max_{VS_i \in WL}(WCRT(J_i^{CP}))$ gets worse as the workload level increases; showing that as the search space and complexity increases the GA evaluations become computationally expensive as well as is unable to find a solution within a reasonable time frame.

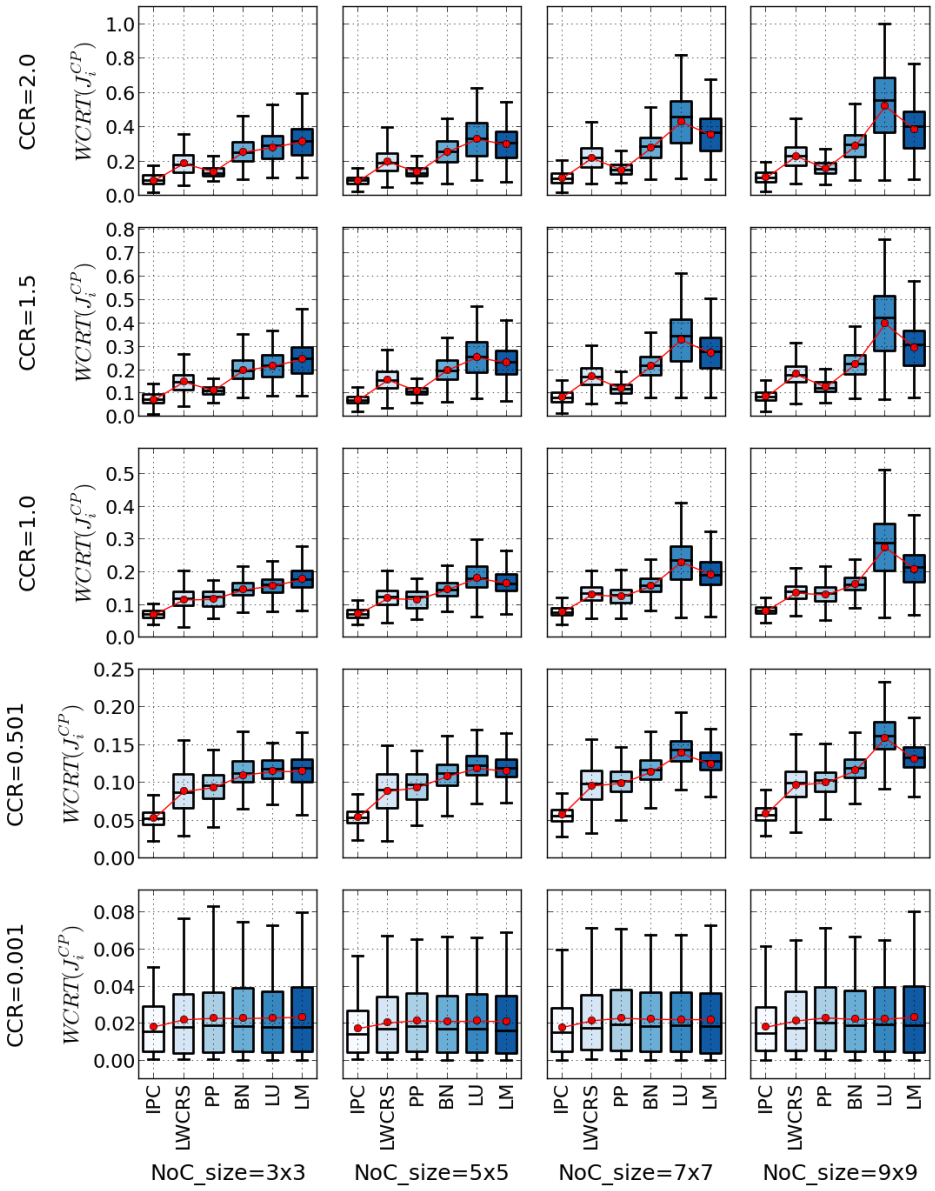

Figure 7 Normalised analytical $WCRT(J_i^{CP})$ obtained using different dynamic mapping approaches, under varying CCR and NoC sizes.

7.3.3 Scalability and CCR variation evaluation

Figure 7 shows the performance variation in the different dynamic mapping techniques when evaluating for scalability and different communication loads. The y-axis of each subplot in Figure 7 displays the normalised calculated analytical $WCRT(J_i^{CP})$ of the video streams for different mapping techniques under different NoC sizes and varying CCR values. $CCR < 1.0$ denote computation-bound workloads while $CCR > 1.0$ denotes communication-bound workloads. In this experiment, we disable the admission-controller, hence all generated video streams are admitted; still, lower analytical $WCRT(J_i^{CP})$ distributions are preferred as this will lead to higher number of video streams being schedulable. The level of workload is kept proportional to the num-

ber of PEs in the platform, such that on average (over 30 seeded runs), the workload=2.2×10^5 per PE. The analytical $WCRT(J_i^{CP})$ of all mappers increase as the CCR increases, since the communication latency has effectively increased. LU and LM are computation-centric mappers, and therefore their performance deteriorates significantly under higher CCR conditions. Furthermore, LU and LM may map communicating tasks further apart as the NoC size increases, which results in a broader distribution of data points. BN does not group tasks together, but takes into account the communication channel load as a metric; hence, it performs better than LM and LU in higher CCRs but still shows a higher $WCRT(J_i^{CP})$ distribution when compared to PP, LWCRS and IPC. The proposed IPC mapping method perform relatively better than all the baselines in all conditions while the proposed LWCRS method performs better than BN, LU and LM. The PP heuristic which attempts to balance computation and communication, is a strong competitor to the proposed mapping techniques. It performs better than LWCRS for $CCR > 0.5$ due to better grouping of the TG, but still shows a slightly higher $WCRT(J_i^{CP})$ distribution when compared with IPC. However, the lower bottom whiskers of LWCRS tells us that in certain workload conditions it can produce a lower $WCRT(J_i^{CP})$ than PP.

8 Conclusion

This paper formally describes a multi-stream video decoding application model and an algorithm for a deterministic admission controller, which uses video stream schedulability tests. A novel point-based, WCRT-aware fitness function for a design-time hard real-time task mapper is presented and compared against dynamic mapping techniques. This work describes two application and platform aware runtime task mapping strategies, that attempt to decrease the end-to-end response-time of the video stream decoding jobs. The first (LWCRS) technique attempts to tightly pack tasks in the temporal domain by using a novel worst-case remaining slack-aware metric of the tasks. The second technique (IPC) groups the I and P frame decoding tasks and maps them to a single PE, and the remaining tasks according to LWCRS. The techniques improve the admission rates of a hard real-time deterministic admission controller and thereby increasing system utilisation. We also present extended evaluation of these two mappers against other existing runtime mappers, under varying platform sizes and communication-to-computation loads.

Simulations carried out reflect that the proposed dynamic task mappers show an improvement of about 10%-20% in mean admission rates and an improvement of about 5%-15% in PE busy times, when compared against other existing heuristic based dynamic task-mappers. Furthermore, better admission rates and PE utilisation can be obtained at a lower usage of the NoC, which could potentially lead to lower power consumption in the system. The results from the static hard real-time GA-MP mapping shows that for lower workloads a suitable mapping solution can be achieved within a reasonable amount of time (3 hours on average). However, for larger workloads the GA-MP can take on average 100 hours to achieve a task-to-PE mapping which is only 5%-10% better than the proposed dynamic mappers.

Results also show that mapping heuristics that rely purely on communication/computation load does not scale well as the NoC size and workload increases. This work shows how the dynamic mappers behave under different CCR workloads. LWCRS and IPC group tasks together to minimise communication and to reduce the computation interference; they perform significantly better than the BN, LU and LM baselines and marginally better than the PP mapping technique under high CCR workloads. By taking into account task and flow blocking factors, better mapping decisions can be achieved. For communication and memory bound applications such as parallel video stream decoding, the performance results of the mapping techniques at higher orders of CCR are of particular interest. Potential further work in this area will be to explore combined

priority assignment and mapping techniques to reduce the worst-case response time further.

Acknowledgement

We would like to thank the LSCITS program (EP/F501374/1) and DreamCloud project (EU FP7-611411), for funding this research and RheonMedia Ltd. for providing industrial case studies.

References

1 Hazem Ismail Abdel Aziz Ali, Luís Miguel Pinho, and Benny Akesson. Critical-path-first based allocation of real-time streaming applications on 2d mesh-type multi-cores. In *RTCSA conf.*, 2013.

2 Giuseppe Ascia, Vincenzo Catania, and Maurizio Palesi. A multi-objective genetic approach to mapping problem on network-on-chip. *J. UCS*, 12(4):370–394, 2006.

3 N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Eng. Journal*, 8:284–292, 1993.

4 Mohamed A. Bamakhrama and Todor P. Stefanov. On the hard-real-time scheduling of embedded streaming applications. *Design Automation for Embedded Sys.*, 17:221–249, 2013.

5 Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. QNoC: QoS architecture and design process for network on chip. *Journal of Sys. Arch.*, 50:105–128, 2004.

6 Giorgio Buttazzo, Enrico Bini, and Yifan Wu. Partitioning real-time applications over multicore reservations. *IEEE Trans. on Industrial Informatics*, 7:302–315, 2011.

7 W.J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Design Automation conf. (DAC), 2001. Proceedings*, pages 684–689, 2001.

8 E.L. de Souza Carvalho, N.L.V. Calazans, and F.G. Moraes. Dynamic task mapping for MPSoCs. *IEEE Design Test of Computers*, 27:26–35, 2010.

9 Michael Ditze, Peter Altenbernd, and Chris Loeser. Improving resource utilization for MPEG decoding in embedded end-devices. In *Proc. of the 27th Australasian Conf. on Computer Science*, 2004.

10 Piotr Dziurzanski, Amit Kumar Singh, and Leandro Soares Indrusiak. Feedback-based admission control for hard real-time task allocation under dynamic workload on many-core systems. In *Architecture of Computing Systems – ARCS 2016*, Lecture Notes in Computer Science, pages 157–169. Springer International Publishing, Apr 2016.

11 ETSI. ETSI TS 101 154 v1.10.1 (2011-06) - digital video broadcasting (DVB) - specification for the use of video and audio coding in broadcasting applications based on the MPEG-2 transport stream. Technical report, European Telecommunications Standards Institute (ETSI), Jun 2011.

12 Mohammad Abdullah Al Faruque and Jörg Henkel. QoS-supported On-chip Communication for Multiprocessors. *Int. Journal of Parallel Programming*, 36:114–139, 2008.

13 Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, Lothar Thiele, and Benoit Dupont de Dinechin. Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources. *Real-Time Systems*, 52:399–449, Jul 2016.

14 Blake Hannaford, Jacob Rosen, Diana W Friedman, Hawkeye King, Phillip Roan, Lei Cheng, Daniel Glozman, Ji Ma, Sina Nia Kosari, and Lee White. Raven-ii: an open platform for surgical robotics research. *Biomedical Engineering, IEEE Transactions on*, 60:954–959, 2013.

15 J. Huang, A. Raabe, C. Buckl, and A. Knoll. A workflow for runtime adaptive task allocation on heterogeneous MPSoCs. In *DATE conf.*, 2011.

16 L.S. Indrusiak. End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration. *Journal of Sys. Arch.*, 60:553–561, 2014.

17 L.S. Indrusiak, J. Harbin, and O. Marchi Dos Santos. Fast simulation of networks-on-chip with priority-preemptive arbitration. *ACM Trans. Design Automation on Electronic Systems*, 20(4):1–22, September 2015.

18 D. Isovic and G. Fohler. Quality aware MPEG-2 stream adaptation in resource constrained systems. In *ECRTS conf. 2004*, 2004.

19 D. Isovic, G. Fohler, and L. Steffens. Timing constraints of MPEG-2 decoding for high quality video: misconceptions and realistic assumptions. In *Euromicro conf. on Real-Time Sys.*, 2003.

20 B. Kao and H. Garcia-Molina. Deadline assignment in a distributed soft real-time system. *IEEE Trans. on Parallel and Distributed Systems*, 8:1268–1274, 1997.

21 S. Kaushik, A.K. Singh, and T. Srikanthan. Computation and communication aware run-time mapping for NoC-based MPSoC platforms. In *SOC Conf.*, 2011.

22 Cor Meenderinck, Arnaldo Azevedo, Ben Juurlink, Mauricio Alvarez Mesa, and Alex Ramirez. Parallel scalability of video decoders. *J. Signal Process. Syst.*, 57:173–194, November 2009.

23 H. R. Mendis, Neil C. Audsley, and L.S. Indrusiak. Task allocation for decoding multiple hard real-time video streams on homogeneous NoCs. In *INDIN conf.*, 2015.

24 H. R. Mendis, L.S. Indrusiak, and Neil C. Audsley. Predictability and utilisation trade-off in the dynamic management of multiple video stream decoding on network-on-chip based homogeneous embedded multi-cores. In *RTNS conf.*, 2014.

25 M. Ruaro, G. Madalozzo, and F. G. Moraes. A hierarchical lst-based task scheduler for noc-based mpsocs with slack-time monitoring support. In *2015*

IEEE International conf. on Electronics, Circuits, and Systems (ICECS), Dec 2015.

26 M.N.S.M. Sayuti and L.S. Indrusiak. Real-time low-power task mapping in Networks-on-Chip. In *2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2013.

27 Andreas Schranzhofer, Jian-Jian Chen, and Lothar Thiele. Dynamic power-aware mapping of applications onto heterogeneous mpsoc platforms. *IEEE Trans. on Industrial Informatics*, 6:692–707, 2010.

28 Zheng Shi, Alan Burns, and Leandro Soares Indrusiak. Schedulability analysis for real time on-chip communication with wormhole switching:. *Int. Journal of Embedded and Real-Time Comms. Sys.*, 1:1–22, 2010.

29 Amit Kumar Singh, Muhammad Shafique, Akash Kumar, Jörg Henkel, Anup Das, Wu Jigang, Thambipillai Srikanthan, Samarth Kaushik, Yajun Ha, and Alok Prakash. Mapping on multi/many-core systems: survey of current and emerging trends. In *DAC conf.*, 2013.

30 Amit Kumar Singh, Thambipillai Srikanthan, Akash Kumar, and Wu Jigang. Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms. *Journal of Sys. Arch.*, 56:242–255, 2010.

31 Ying Tan, Parth Malani, Qinru Qiu, and Qing Wu. Workload prediction and dynamic voltage scaling for mpeg decoding. In *Proc. of the 2006 Asia and South Pacific Design Automation Conf.* IEEE Press, 2006.

32 Mitchell D Theys, Tracy D Braun, HJ Siegal, ANTHONY A Maciejewski, and YK Kwok. Mapping tasks onto distributed heterogeneous computing systems using a genetic algorithm approach. *Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences*, pages 135–178, 2001.

33 Wayne Wolf. *Multimedia applications of multiprocessor systems-on-chips*, pages 86–89. IEEE Computer Society, 2005.