Bio-inspired Distributed Task Remapping for Multiple Video Stream Decoding on Homogeneous NoCs

Hashan R. Mendis*, Leandro Soares Indrusiak† and Neil C. Audsley‡
Real-time Systems Group, Department of Computer Science
University of York
Email: *hrm506@york.ac.uk, †lsi@cs.york.ac.uk, ‡neil.audsley@york.ac.uk

***Abstract*—Centralised management of distributed systems require a significant amount of monitoring traffic to maintain an accurate view of the system global state. The communication overhead of these systems becomes a bottleneck as the number of processing elements in the network and workload increase. State-of-the art in decentralised resource management techniques address this issue by allowing individual or clusters of nodes to make decisions at runtime to manage the dynamic workload. The primary contribution of this paper is using a bio-inspired, distributed, task remapping technique to manage dynamic multiple video stream decoding workloads. Our proposed technique has a low-communication overhead and is used to reduce the cumulative job lateness of the video streams. Secondary contributions include, several improvements to an existing cluster-based resource management approach to introduce awareness of task blocking and relocation distance. We evaluate these two remapping methods by comparing the improvement of job lateness, communication overhead and distribution of utilisation via simulation of several workload patterns.**

I. Introduction

Systems-on-Chip with hundreds of cores are now becoming a preferred target for multimedia applications [1]. High computational load in video decoding can be parallelised and distributed across the different processing elements on the chip to minimise metrics such as overall execution time, power or even temperature. Violating timing constraints of soft real-time video decoding applications can lead to reduced user engagement [2]. Video decoding execution times vary greatly depending on the spatial and temporal attributes of the video [3]. Furthermore, when decoding multiple streams of live video (e.g. multipoint video conferencing, multi-camera real-time video surveillance, multiple view object detection), the workload characteristics are increasingly dynamic and are difficult to predict beforehand. Not respecting the timing requirements of live video streams will negatively impact the quality of experience (QoE). Efficient task allocation is therefore critical in achieving load balance, power/energy minimisation and latency reduction [4]. However, task allocation to optimize multiple metrics, is a NP-complete problem. Centralised resource management, with a master-slave approach is relatively simple to implement and is sufficient for small systems; however, these systems suffer from scalability and redundancy issues [5], [6]. Cluster based resource management techniques have been introduced (e.g. [7]) to overcome the limitations of centralised systems by partitioning the system resources and employing multiple cluster managers. Despite these efforts, the complexity of dynamic applications and large-scale multiprocessor, distributed systems of the future have given reason to investigate fully-distributed, autonomous self-organising/optimising mechanisms [8]–[10]. Such systems should be able to adapt or optimize itself to changing workload and internal conditions and to recover from faults. Many of these systems employ middleware, that perform self-management features by autonomously controlling and adapting task allocation and resource management at runtime.

978-1-4673-8164-2/15/$31.00 © 2015 IEEE

This paper extends and adapts two existing distributed resource management techniques to reduce the lateness of multiple video decoding streams on a Network-on-Chip (NoC) based multicore platform. The novelty in the work presented is twofold. Firstly, the bio-inspired load-balancing algorithm introduced in [11], was adapted to enable dynamic distributed remapping within a network-on-chip (NoC). The proposed technique is fully distributed, does not contain a global manager and each individual processing element (PE) periodically executes a lightweight set of rules which gives it the capability to make task reallocation decisions using its local knowledge. This introduces controlled redundancy into the system where there is no single point of failure or management entity. This work is useful for time-critical applications with dynamic workloads such as live video processing systems which require a high-degree of redundancy and reduced latency to operate. Secondly, we adapt the cluster-based resource management approach given in [12], by introducing task priority and relocation distance awareness.

The rest of this paper is organized as follows. Section II presents related work in multicore resource management. Section III introduces the system models and formulates the problem. Section IV presents the proposed, bio-inspired distributed remapping algorithm. Section V describes the modifications made to the cluster-based resource management and Section VI presents the experimental design and discusses the results. Section VII concludes this paper.

II. Related work

A. Centralised vs. distributed resource management

Achieving effective run-time mapping on multi/many-core systems is a challenging task, particularly when the workload

and their arrival patterns are not known a priori. Resource management decisions and protocols for such systems can be either centralised, decentralised or a combination of both (hybrid). For example in [4], [13], the authors use a global resource manager to monitor the status of the processing elements and use heuristics-based task mapping and scheduling decisions, to minimise network congestion and communication energy. The global managers in these systems have an accurate and up-to-date knowledge of the status of each slave processing node in the system. On the other hand, decentralised resource management techniques such as those in [5], [7] overcome the scalability issues (due to the feedback monitoring traffic communication overhead) in the centralised systems by employing agent-based interaction protocols. In these distributed designs, each individual node use their local knowledge to make efficient task mapping and migration decisions. In both these agent-based systems a series of request-reply messages are exchanged between the agents of the system to determine a suitable processing node to map an application onto. The virtual clusters in [5] have managers, who decide if their cluster is appropriate for an incoming new application. If their cluster is not available, they send a request to other clusters to verify if it is possible to migrate tasks; subsequently, cluster resizing is performed if migration is also not possible. Castilhos et al. [12] extend the work done in [5], by making the cluster resizing approach fully distributed. Compared to a centralised system, their technique reduces the hop distance among communicating tasks resulting in reduced total execution time. The bio-inspired, distributed, remapper presented in this paper does not use virtual clusters; furthermore, the task remapping decisions are fully decentralised.

B. Dynamic task remapping

As shown by several previous work [14], [15], dynamic management approaches are known to be more suitable to adapt, task to processing element allocations at unknown run-time scenarios. Das et al. [15] uses task remapping to reduce the communication energy and introduce reliability for throughput-constrained multimedia applications. Pre-computed task to core mapping selections obtained at design-time are looked up at runtime and used when a fault occurs. Derin et al. [14] proposes several communication and computation aware heuristics that can be used during on-line task mapping. The authors monitor the system status for a MPEG-2 decoder application at runtime and decide when and where to migrate the tasks from a faulty core, such that communication cost and average execution time is minimised. However, they do not consider task migration costs such as the communication overhead required to move the task code and data memory to the new processing node. If proper techniques such as code check-pointing and sufficient interprocessor queues are not available, task migration overhead may severely degrade the quality of service of soft real-time multimedia applications [16]. In [17], tasks are moved around the network at runtime to avoid/reduce network contention in a non-preemptive network on chip. Their remapping algorithm is triggered periodically and uses congestion and communication power aware heuristics to remap the tasks. Here, remapping is performed by changing the entries in a central mapping table. Similarly, the technique proposed in this paper does not involve task migration and hence remapping does not require moving a tasks state from one processor to another. Bio-inspired task management techniques have been explored by Brinkschulte et al. [8], where they introduce an artificial hormone system for task mapping on heterogeneous processing elements, inspired by the hormone system of animals. Initial task allocation is found by exchanging different hormone signals and optimization is done by periodic re-allocation They also show how to guarantee an upper bound for self-configuration time, which is beneficial for real-time applications. They derive a quality measure which includes communication distance, CPU share and suitability of the PE to task mapping; their results showed an average improvement of 10% over the simple load balancing technique on a 4x4 network [18]. Mudry et al. [10] present a fully distributed task allocation approach inspired by the healing and growth process found in bio-organisms. Each node in their network monitors their own load and perform self-scaling functionality; under over-load conditions the nodes are capable of independently locating an unused node in the system and replicate its code into it. Even though this method achieves a high degree of redundancy, the authors have not investigated the impact of the replication overhead (communication cost) on the timing requirements of the applications.

III. System model and problem formulation

A. Application model

The application model (Fig. 1) consists of workflows which resemble a container for parallel video stream decoding requests that may arrive at randomly within a specified inter-arrival time. Hence, the number of parallel video decoding streams processed by the system will vary over time. Video stream consists of an arbitrary number of *N* independent jobs/task-sets. Each job (J_i) represents the MPEG group of pictures (GoP), and is modelled via a fixed dependency task-graph, and takes the structure defined in Fig. 1. Each node in the task-graph is a MPEG-2 frame-level decoding task, and has fixed precedence constraint and communication flow shown via the graph edges. A task can only start execution *iff* its predecessor(s) have completed execution and their output data is available. 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 execution time, c_i is the worst-case computation cost and a_i is the arrival time of the task τ_i. Tasks are preemptive and have a fixed priority. Tasks within a job are assigned fixed mapping and priorities at the start of the video stream; these exact assignments are used for all tasks of all subsequent jobs in a video stream. Tasks of low resolution video streams are given higher priority over high-resolution video streams, to ensure low-resolution video streams have a lower response-time.

The spatial resolution of a video stream will correspond directly to the computation cost of the task and the payload of the message flows. The exact execution time of the tasks are unknown in advance; however, it is assumed that the worst-case computation cost can be estimated. Subtask deadlines are unknown but each job is considered schedulable if it completes execution within its end-to-end deadline ($J_i^r \leq D_{e2e}$). The response-time of a job (denoted J_i^r) is the arrival time of the job to the point in time which all of its subtasks have completed execution. A job is considered late when $(J_i^r - D_{e2e}) > 0$ and

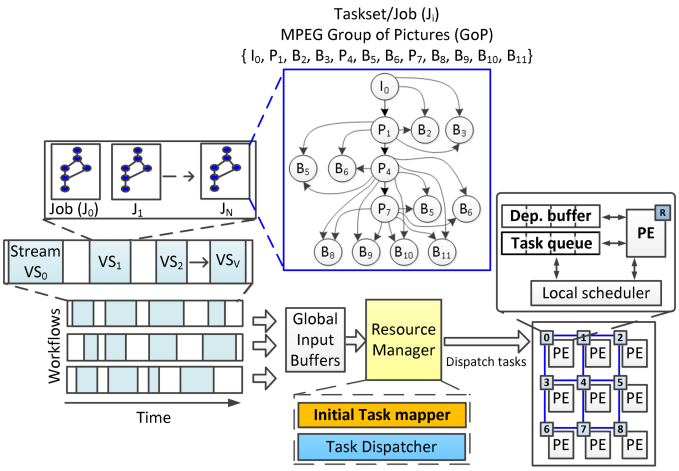

Fig. 1: System overview diagram

late jobs impact the viewing QoE of the real-time video stream. We assume that the arrival rate of jobs are sporadic, and the arrival pattern of new video decoding streams are aperiodic.

Once a task has completed execution, its output (i.e. the decoded frame data) is immediately sent as a message flow to the processing element executing its successor child tasks, as well as to a buffer in main memory. Message flows inherit the priority of their source tasks, with an added offset to maintain unique message flow priorities. 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 message flow Msg_i. The C_i of a message flow as given in Eq. 1 includes the hop-distance and the number of flits (i.e. payload).

$$C_i = (numHops \times arbitrationCost) + (numFlits) \quad (1)$$

B. Platform model

The multi-core platform is composed of P homogeneous processing elements (PEs) connected by a NoC interconnect. Each PE has a task-queue which is contained within the local memory. The PEs are directly connected to the NoC switches which route data packets towards a destination core. We assume the NoC in our platform model uses fixed priority pre-emptive arbitration, has a 2D mesh topology and uses the XY deterministic algorithm for routing such as in [19]. We assume that the NoC link arbiters can preempt packets when higher-priority packets request the output link they are using. This makes it easier to predict the outcome of network contention for specific scenarios. We assume all inter-PE communication occurs via the NoC by passing messages. Once a task is released from a global input buffer, it is sent to the task queue of the assigned PE. The PE upon completing a tasks execution, transmits its output to the appropriate PEs dependency buffer. Once a task has completed, the local scheduler picks the next task with the highest priority with dependencies fulfilled, to be executed next. The resource manager (RM) of the system (Fig. 1), performs *initial task mapping* and priority assignment and *task dispatching* to the PEs. It also maintains a task-to-PE mapping table of the jobs of every admitted and active video stream in the system. The mapping table is essentially a hash-table where keys are task-identifiers and values are node-identifiers. In this work, the terms RM and $dispatcher$ are interchangeable as task dispatching is a functionality of the RM. The main responsibility of the RM is to make initial mapping decisions for new video streams, and to dispatch tasks to the mapped PEs according to the task-to-PE mapping table. Most importantly, the system is open-loop as the RM does not gather monitoring information from the PEs.

C. Problem statement

In a centralised closed-loop system the PEs would continuously feedback the task states such as their completion time to a central manager via status message flows. The central manager would then have an accurate, global knowledge of the system in order to make efficient task management decisions for future workloads. As discussed in [5], [6], these advantages come at the price of higher communication traffic, hot-spots and higher probability of failure and bottlenecks around the centralised manager and performance degradation as the NoC size and workload increases. On the other hand, cluster-based distributed management approaches can offer a certain degree of redundancy and scalability by varying the number of clusters and respectively local cluster managers. However, appropriate cluster size selection is vital to balance communication-overhead/performance; for example, cluster monitoring message flow routes and the cluster manager processing overhead will increase as the cluster size increases. Furthermore, the local cluster managers are still points of failure in the system, where if one of them fails the respective cluster of nodes will severely degrade in performance.

Fully distributed approaches offer higher levels of redundancy and scalability over cluster based approaches for large scale systems, due to not having any central management nodes. However, due to the lack of global knowledge and no monitoring being performed by a centralised authority, the system may be load-unbalanced, and cause jobs to miss their deadlines and become late. To reduce this job lateness of the admitted dynamic varying workload, we propose a bio-inspired distributed task-remapping technique with self-organising properties. This work builds upon an existing bio-inspired load-balancing algorithm by Caliskanelli et al. [11].

IV. Bio-inspired, Distributed Task-Remapping

A. Adapted pheromone signalling algorithm

The distributed, load-balancing algorithm introduced in [11] (henceforth referred to as the *PS algorithm*) is based on the pheromone signalling mechanism as seen in social insects (e.g honey bees). This algorithm has previously been used to improve reliability of wireless sensor networks. We extend and adapt this algorithm to enable distributed remapping of late tasks from NoC PEs that are heavily utilised onto PEs that are under utilised in the near proximity. A node in the network can be classified either as a queen or a worker. Queen nodes (QNs) periodically propagate pheromone (denoted hd) to their network neighbourhood. All nodes accumulate and pass on pheromones received from the QNs, and the dose of the pheromone is decreased at every hop-distance away from the QN. The pheromone level (denoted h_i) for each

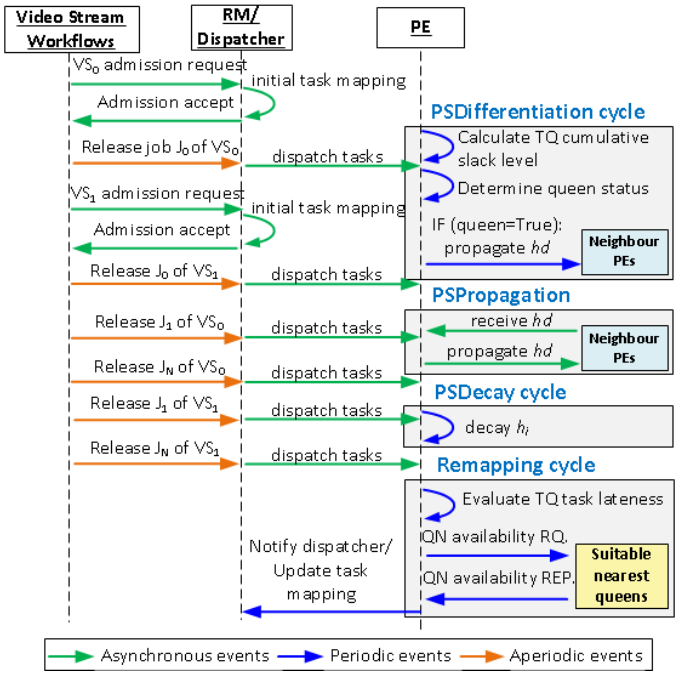

Fig. 2: Sequence diagram of PSRM algorithm related events. Time triggered (periodic) : *PSDifferentiation*, *PSDecay* and *Remapping* cycles; Event triggered: *PSPropagation*

node decays over time. A node becomes a QN when its h_i drops below a certain threshold (denoted Q_{TH}). The range of the pheromone broadcast by the QN is limited to reduce the communication overhead, and hence worker nodes are only aware of nearby QNs. Each node executes a set of simple rules to obtain increased performance on the system as a whole.

As Fig. 2 illustrates, the original PS algorithm contains two periodic and one asynchronous event cycles, that are carried out by each node in the network. The *PSDifferentiation* cycle occurs every T_{QN} seconds to distinguish its queen status. Every T_{DECAY} seconds, the *PSDecay* event occurs to decrement/decay the node hormone level. The *PSPropagation* event occurs when a new hormone dose is received by a neighbouring node. In depth details of these events can be found in [11]. As shown in Fig. 2, the system will continually receive new video decoding requests aperiodically and the jobs within these video streams will be received sporadically. The workload arrival patterns and the PS algorithm periodic event cycles need to be synchronised to obtain a reasonable performance improvement while maintaining low communication overhead.

Algorithm. 1 shows the extensions made to the PSDifferentiation cycle of the PS algorithm. In the original algorithm [11], the Q_{TH} is fixed but in our extension the Q_{TH} is dynamically adjusted depending on the workload mapped on the PE. The *cumulative slack* of the tasks mapped on the PE is used to vary the QN threshold Q_{TH}, such that a node will become a QN if it has enough slack to accommodate additional tasks (line 3). The slack of a task is calculated as the difference between the relative deadline (d_i) and the observed response-time of the task r_i. A negative cumulative slack value indicates the PE does not have any spare capacity to take additional tasks, and hence the node is converted or remains a worker

Algorithm 1: PSRM Algo [PSDifferentiation]. (Periodic : every T_{QN})

/* calc. normalised cumulative TQ slack */
1 $TQ_{Slack} = \frac{\sum_{\forall \tau_i \in PE_{MPT}} (d_i - r_i)}{\sum_{\forall \tau_i \in PE_{MPT}} (d_i)}$;
/* calc. QN threshold */
2 **if** $TQ_{Slack} > 0$ **then**
3 | $Q_{TH} = Q_{TH} \times (1 + (TQ_{Slack} \times Q^{\alpha}_{TH}))$;
4 **else**
5 | $Q_{TH} = h_i \times Q^{\beta}_{TH}$;
6 **end**
/* determine queen status */
7 **if** $h_i < Q_{TH}$ **then**
8 | QueenStatus = TRUE;
9 | $hd = \{0, H_{QN}, QN_{xy}, PE_{MPTinfo}\}$;
10 | Broadcast hd to neighbours;
11 **else**
12 | QueenStatus = FALSE;
13 **end**

node. Line 1 in Algorithm. 1 shows the calculation of the task queue (TQ) cumulative slack (TQ_{Slack}) of the mapped tasks. If TQ_{Slack} is positive, Q_{TH} is incremented by a ratio defined by $(TQ_{Slack} + Q^{\alpha}_{TH})$; where $\{Q^{\alpha}_{TH} \in \Re \mid 0 \leq Q^{\alpha}_{TH} \leq 1\}$ is a parameter of the algorithm. If TQ_{Slack} is negative, then the algorithm ensures the node does not become a QN in this differentiation cycle, by setting Q_{TH} as a proportion of h_i as given in Line 5; here $\{Q^{\beta}_{TH} \in \Re \mid 0 \leq Q^{\beta}_{TH} \leq 1\}$ is also a parameter of the algorithm. The self-organising behaviour of the distributed algorithm (specifically the PSDifferentiation cycle in Algorithm. 1), stabilises the number and position of the QNs in the system, as time progresses and depending on the workload. A node propagates pheromones immediately after it becomes a queen (line 10). We represent the pheromone dose (hd) as a four position vector (line 9) containing the distance from the QN, the initial dosage (H_{QN}), the position of the QN in the network (QN_{xy}) and a data structure ($PE_{MPTinfo}$) containing the p_i and c_i of the tasks mapped on the QN. The worker nodes will receive and store this information as the pheromones traverse through the network.

B. Task remapping

The initial mapping of video stream tasks is performed according to the lowest worst-case utilisation heuristic. A task within a job may be late due to the PE or network route being over-utilised and/or due to the heavy blocking incurred by higher-priority tasks and flows. The goal is to change the task-to-PE mapping of the late tasks, such that these causes of lateness can be mitigated. The task-remapping procedure (Algorithm. 2) is executed by each PE periodically, using only its local knowledge gathered via the pheromone doses. Unlike in task-migration [16], in this work, task remapping is simply an update to a mapping table maintained by the dispatcher. Most importantly, extra overhead is not incurred to move task code/data.

Algorithm. 2 illustrates the proposed remapping procedure that utilises the adapted PS algorithm, denoted as $PSRM$. The

Algorithm 2: PSRM Algo [Task remapping].
(Periodic : every T_{RM})

```
   /* find most late task from task queue                    */
1  τ_i^{MAX_L} = MAX({τ_i ∈ TQ | (a_i + d_i) ≤ t_c});
   /* get current blocking for late task                     */
2  B(τ_i^{MAX_L}) = getCurrentBlocking(hp(τ_i^{MAX_L}));
   /* find suitable QNs which offer lower blocking,
      than current blocking                                  */
3  Q^B_List = { };
4  foreach Q_i ∈ Q_List do
       /* get target task blocking factor                   */
5      Self_B_Q = Σ_{∀τ_j ∈ hp(τ_i^{MAX_L})} c_j;
       /* get number of lower priority tasks                */
6      LP_size = |lp(τ_i^{MAX_L})|;
7      if Self_B_Q < B(τ_i^{MAX_L}) then
8          Insert {Q_i, LP_size} to Q^B_List;
9      end
10 end
   /* request for QN availability                            */
11 Avlb_Q^B_List = requestAvailability(Q^B_List);
   /* get available QN that has least amount of
      lower priority tasks                                   */
12 {Q_i^{MIN_LP}, LP_Q^{MIN}} = MIN(Avlb_Q^B_List);
   /* Update dispatcher task-mapping table                   */
13 Notify dispatcher : τ_i^{MAX_L} → PE(Q_i^{MIN_LP});
```

following steps occur at each remapping cycle (seen in Fig. 2). Firstly the task with the maximum lateness τ_L^{MAX} from the PE task queue, is selected as the task that needs to be remapped to a different PE (line 1). The deadline of a task (d_i) is calculated as a ratio of the end-to-end job deadline (D_{e2e}), as given in [20]. Each node is aware of the nearest QNs (Q_{List}), and their mapped tasks, by storing the information received from each pheromone dose hd. In Lines 3-10, the algorithm evaluates the worse-case blocking that will be experienced for the target task τ_L^{MAX} and the number of lower priority tasks that will be blocked, by mapping it onto each $Q_i \in Q_{List}$. Once a list of QNs with lower blocking than the current blocking is obtained (lines 7-9), they are requested (RQ) for their $availability$ (line 11) via a low payload, high priority message flow. The QNs reply (REP) with its availability (i.e. if other worker nodes have been remapped to a QN in that remapping cycle, then the QNs' availability is set to $false$). This avoids unnecessary overloading of QNs. τ_L^{MAX} will then be remapped to the QN with the least number of lower priority tasks (denoted $Q_i^{MIN\_LP}$) from the available QN list ($Avlb\_Q^B_{List}$) (line 12). Finally, the task dispatcher is notified via message flow to update the task mapping table; the dispatcher looks up the task-id in the table and updates the corresponding node-id with the new remapped node-id. When the tasks of the next GoP of the video stream arrives into the system they will be dispatched to the node-id indicated by the updated mapping table. Therefore, remapping will only take effect from the subsequent arrival of the next job in the video stream. Even though there is an update message sent to the dispatcher at a remapping event, the remapping decision is achieved purely using local information at each PE, based on the PSRM algorithm.

Fig. 3 illustrates an example of the remapping procedure in

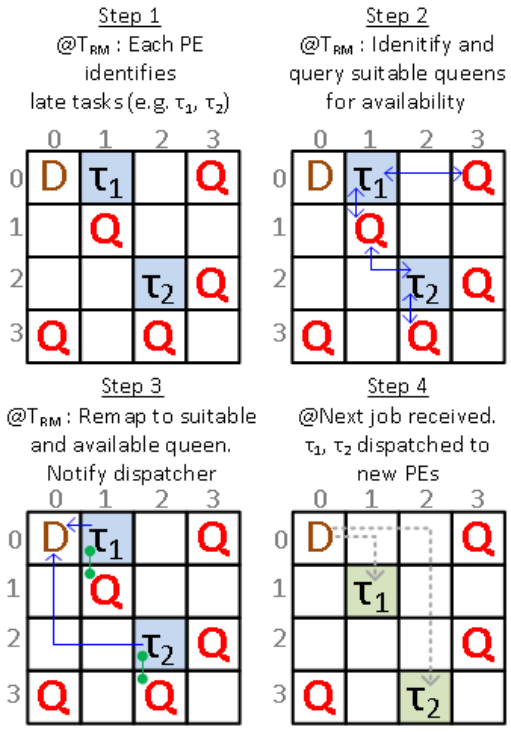

Fig. 3: Task remapping example. (Q=queen nodes; D=Dispatcher; $[\tau_1, \tau_2]$ are late tasks; Blue lines represent communication

Differenciation cycle (T_{QN})	0.22
Decay cycle (T_{DECAY})	0.055
Remapping period (T_{RM})	6.9
Default QN threshold (Q_{TH})	20
QN threshold inc./dec. factors (Q^{α}_{TH}, Q^{β}_{TH})	0.107, 0.01
Pheromone time and hop decay factors	0.3, 0.15
Pheromone propagation range	3

TABLE I: PSRM algorithm parameters

a 4x4 NoC. The (x, y) coordinates refer to the processing node in column x and row y. In step 1 of Fig. 3, at each remapping interval (T_{RM}) each PE identifies the late tasks in their task queues; they are also aware of the position of any nearby QNs due to the pheromone signals. τ_1 and τ_2 on PE(1,0) and PE(2,2) are tasks that are late, at that time instant. In step 2, they determine the suitability of each QN to remap the late tasks to. τ_1 can either be remapped to Q(1,1) or Q(3,0); and τ_2 can be remapped on to either Q(3,2) or Q(1,1) but Q(3,2) is not suitable due to the task blocking behaviour and Q(0,3) is not in the Q_{List} due to distance. In step 2, the nodes request for the suitable QNs' availability; in this instance PE(1,0) obtained a lock on Q(1,1) first. Hence, τ_1 will be remapped onto Q(1,1) and τ_2 will be remapped to Q(2,3). In step 3 the PEs notify the dispatcher via a message flow regarding the remapping. In step 4 the next job arrives and τ_2 and τ_3 are now dispatched to the new processing elements - PE(1,1) and PE(2,3) respectively.

The performance of adaptive algorithms such as PSRM is highly dependent on the selection of a good set of parameters. Manual selection of parameters is not feasible due to the size of the search space. Table I shows several important

parameters obtained via a search-based parameter selection method inspired by [21]. The parameters T_{QN}, T_{DECAY} and T_{RM} and their ratios play a key role in obtaining a good performance from the algorithm. The experimental results during the parameter search process show that the remapping frequency has a significant impact in accuracy and communication overhead. The relationship between these parameters have been investigated extensively in previous work [11], [21]. As a general guideline, to keep the communication overhead low, the event cycles (T_{QN} and T_{RM}) and the QN hormone propagation range must be kept relatively low.

C. Disadvantages of the lack of global state

The platform model has fixed priority preemptive NoC arbiters and local schedulers. Hence, tasks and flows can be blocked by higher priority tasks and flows. The remapping heuristic takes into account the new tasks' blocking incurred by a possible remapping (lines 4-10 of Algorithm. 2). However, since the processing nodes lack a global view of the communication flows, the remapping heuristic cannot take into account the change in the overall network *communication interference pattern* caused by the reallocation of the tasks. Therefore, there are situations where remapping a task can result in an increase in the lateness. As shown in Fig. 2 and Algorithm. 1, every T_{QN} time units the worker nodes get updates from all QNs in close proximity to them. However, between subsequent *PSDifferentiation* events, the workload of the QN can change rapidly when the system is heavily utilised, which may lead to inaccurate local knowledge regarding the nearby QNs. Furthermore, late tasks should be remapped ideally before the next job invocation. However, the remapping event is periodic (i.e. every T_{RM} seconds) which allows the remapping overhead to be kept at a minimum, but does not guarantee synchronisation with the workload arrival pattern. Longer periodic events may lead to inconsistency in data and states, but are used to keep the communication overhead at a minimum.

V. Virtual cluster-based task remapping

This section explains the improvements introduced to the cluster-based resource management technique presented by Castilhos et al. [12]. In their work the multiprocessor system on chip (MPSoC) is divided into virtual clusters (VC), and each VC has a local manager (LMP) which performs task monitoring, migration and communication with the other LMPs. The other PEs in the network are referred to as *slave PEs*. A PE sends an status update to its respective cluster LMP every time a task completes execution, therefore every LMP has accurate knowledge regarding the nodes in its cluster. Each LMP is responsible for mapping tasks inside the cluster. In the case when there are no free PEs available in the cluster, the LMP sends a *loan request* to the other neighbour clusters. When the neighbouring LMPs receive this loan request, they search for an available PE in their cluster. If an available PE is found, the LMP reserves this PE and sends the location of the PE as a *loan reply* to the requesting LMP. Once the requesting LMP has received all the replies from the other LMPs, it picks the closest slave PE to map the task to. This algorithm was then directly applied to our resource management problem by initially attempting to remap late tasks to a PE in the same cluster and failing to do so, communicate with neighbouring clusters to remap to a remote cluster. By doing so we hope to reduce the overall lateness of the video decoding jobs admitted into the system. However unlike in [12] we do not perform re-clustering or task-migration and hence the initial cluster size remains fixed throughout the operation of the system. In our platform model, the LMPs perform task execution similar to the slave PEs as well as carry out monitoring and periodic task remapping procedures; however, we do not model the execution overhead taken to execute the resource management functionality.

Cluster size	2x5
Remapping period (T_{RM})	6.9
Max. number of late tasks to remap	7

TABLE II: CCPRM$_{V2}$ algorithm parameters

The initial mapping of video stream tasks is performed according to the lowest worst-case utilisation heuristic. Similar to the PSRM algorithm in Section IV, each LMP performs remapping at periodic time intervals and notifies the task dispatcher of any task remapping decision. Furthermore, the notion of $availability$ in our model depends on PE utilisation. Hence, each LMP will first attempt to find an underutilised PE in its own cluster to remap late tasks to; if one is not found it would send a loan request to neighbouring LMPs. The neighbouring LMPs would subsequently initiate the same utilisation-based search on their cluster. We denote this initial algorithm as $CCPRM_{V1}$ as it has minimal change to the original cluster algorithm given in [12].

A. Proposed modifications to the cluster-based resource management technique

Preliminary experiments showed several limitations of the original CCPRM$_{V1}$ algorithm, which we highlight below along with proposed improvements:

- There are situations where tasks are relocated further away from its communicating child/parent tasks. These remapping decisions will result in traffic flows with longer routes and may cause unwanted interference to lower-priority traffic flows.
  Suggested improvement : Limit task relocation to 2 hop distance.
- In a priority preemptive platform, task remapping can have negative effects if the blocking factor (Eq. 2) is not taken into account. If the blocking factor incurred by higher priority tasks in the new PE is more than the current blocking, the tasks response-time would increase. Likewise, lower priority tasks on the new PE will be affected by the task remapping, causing them to incur lateness.
  Suggested improvement : balanced blocking heuristic - ensure the new PE will produce a lower blocking factor as well as has the minimum number of low-priority tasks, similar to the approach given in the PSRM Algorithm 2 (lines 4-10).

$$\text{Blocking factor} = \sum_{\forall \tau_i \in hp(\tau_i)} x_i \qquad (2)$$

- The selected PE to remap the late task might have a negative slack value, as defined by TQ_{Slack} in the PSRM Algorithm 1 (line 1). This indicates that a majority of the tasks mapped on the new PE may already be late. **Suggested improvement** : Ensure a late task is only remapped to a PE with positive slack.
- As illustrated in [12], the original location of the LMP is at the corner of the cluster. This results in longer routes for monitoring traffic from PEs at the furthest edges of the cluster and may impact data traffic in NoCs with larger cluster sizes.
  Suggested improvement : Place the LMP at the center of the virtual cluster.

The modified algorithm(denoted $CCPRM_{V2}$) has 3 important parameters: cluster-size, remapping period and number of late tasks to remap. The parameters given in Table II were selected as the best after a parameter search, similar to the one carried out for PSRM (Section IV-B). A large cluster size will result in more monitoring traffic transmitted to a central node in the cluster. A larger number of late tasks to remap would cause the heavy variation in the load which in some cases may be undesirable. It is important to note that like PSRM, the remapping performed by CCPRM$_{V2}$ will change the interference patterns of the message flows; hence certain tasks and flows may incur lateness while others will decrease in lateness.

VI. Evaluation

A. Experiment design

A discrete-event, abstract simulator with a light-weight, NoC simulation component as explained in [22] has been adopted, to simulate a 10x10 NoC and video stream workload as explained in Section III. The level of workload was configured such that there would be an upper limit of 103 parallel video streams at any given time in the simulation. Experiments were performed under 30 unique workload situations, where the number of videos per workflow, their resolutions and arrival patterns vary based on the randomiser seed used in each simulation run. The computation to communication ratio of the workload was approximately 2:1. The resolution of the video streams were selected at random from a list of low to high resolutions (e.g. from 144p to 720p). The inter-arrival time of jobs in a video stream were set to be between 1 to 1.5 times the D_{e2e}. Tasks were initially mapped to the lowest utilised PE (according to worse-case utilisation) and priority assignment of the tasks followed a scheme were the lowest-resolution tasks get the highest priority. This initial mapping and assignment scheme were constant variables for all evaluations.

1) Metrics: The experiments have multiple dependent variables as described below:

- *Total number of fully schedulable video streams* : is the number of all admitted video streams that have no late jobs (i.e $J_i^r \leq D_{e2e}$).
- *Cumulative job lateness* (C_L^{Jobs}) : is calculated as the summation of lateness of all the late jobs from every video stream (v_i) admitted to the system (Eq. 3). In Eq. 3, J_i^L is a late job and VS denotes all the video streams admitted to the system. We measure the job lateness with remapping enabled/disabled, hence a reduced C_L^{Jobs}, when remapping is enabled is considered an improvement to the system. This metric gives us a notion of how the remapping technique reduced the lateness of the unschedulable video streams, which directly affects the QoE of the video stream.
- *Communication overhead* : is calculated as the sum of the basic latencies (C_i) of every control signal in the respective remapping technique. In the PSRM algorithm these are the pheromone broadcast and QN availability request signals. In the cluster-based technique the PE status update traffic and the inter-cluster communication traffic contributes to the overhead. Furthermore, the task dispatcher notification messages in all the remapping techniques are included in the overhead. Lower communication overheads lead to less congested networks as well as lower communication energy consumption [15].
- *Distribution of PE utilisation*: is calculated by the measured total busy time for every PE on the network during a simulation run. PE utilisation gives a notion of the workload and a lower variation in workload distribution is desirable. Overloading a single resource and/or having a high number of idle PEs, are undesirable properties which may lead to reduced reliability and increased wear-and-tear.

$$C_L^{Jobs} = \sum_{\forall v_i \in VS} \left[\sum_{\forall J_i^L \in v_i} (J_i^r - D_{e2e}) \right] \tag{3}$$

$$\text{Comms. overhead} = \sum_{\forall msg_i \in ControlMsgs} C_i \tag{4}$$

2) Baseline remapping techniques: The proposed PSRM distributed remapper is evaluated against the following baselines :

- $CCPRM_{V2}$ - is the improved cluster-based management, with a cluster size of 2x5 (i.e 10 clusters).
- *Centralised management* - is essentially CCPRM$_{V2}$ with only one 10x10 cluster. A single LMP receives status updated from every slave PE in the network and performs periodic remapping as outlined in Section V-A. The LMP notifies the task dispatcher of any remapping decisions.
- A *random remapper* - is a remapping scheme where, every remapping interval each PE selects the most late task in its task queue and randomly selects another node on the network to remap to. The task dispatcher is notified of the remapping event.

B. Experimental results

The comparison of CCPRM$_{V1}$ and CCPRM$_{V2}$ for the C_L^{Jobs} metric is shown in Fig. 4(a). In this plot a positive improvement indicates that task remapping has helped to reduce the cumulative job lateness in the admitted video

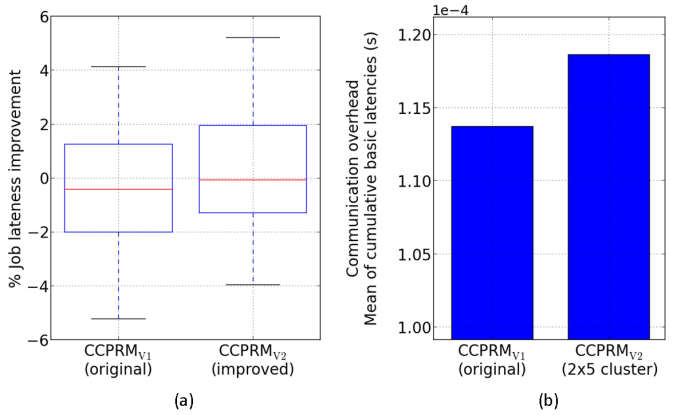

Fig. 4: Comparison of CCPRM$_{V1}$ (original) and CCPRM$_{V2}$ (improved) (a) Cumulative job lateness improvement, (b) Communication overhead

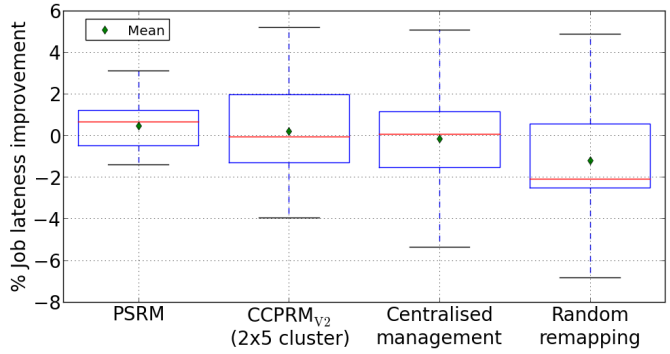

Fig. 5: Distribution of cumulative job lateness improvement after applying remapping.

streams. A negative improvement indicates that the remapping has instead worsened the lateness of the jobs. Each sample in the distribution corresponds to a simulation run with a unique workload. It is clear that the modifications made to the original CCPRM$_{V1}$ technique has resulted in an improvement in reducing job lateness. In CCPRM$_{V1}$ a majority of the data shows negative improvement, while CCPRM$_{V2}$ shows more positive job lateness improvement. However, this improvement has costed a 4% increase in communication cost. Certain constraints in the local remapping decisions in CCPRM$_{V2}$ would result in more communication with neighbouring clusters which might explain the increased overhead.

Fig. 5 shows the distribution of cumulative job lateness improvement for each of the remapping techniques. Firstly, all the techniques show both negative and positive improvements; hence, under certain workload situations the remapping techniques have failed to improve the lateness of the jobs. However, a majority of the distribution in both PSRM and CCPRM$_{V2}$ are in the positive improvement region. PSRM has a smaller spread in lateness compared to the baselines. The upper quartile and a significantly large proportion of the inter-quartile range (IQR) falls in the positive improvement area, which is not seen in any of the baselines. In over 60% of the workload scenarios PSRM will produce positive improvement to the job lateness of the video streams but the actual improvement is small (up to 3%-4%). Futhermore, in Fig. 6, we can see PSRM

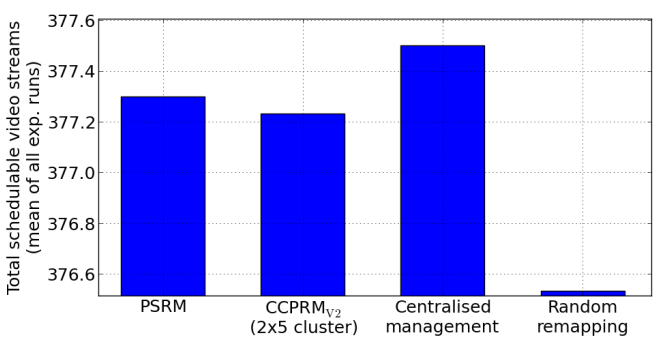

Fig. 6: Comparison of fully schedulable video streams for each remapping technique

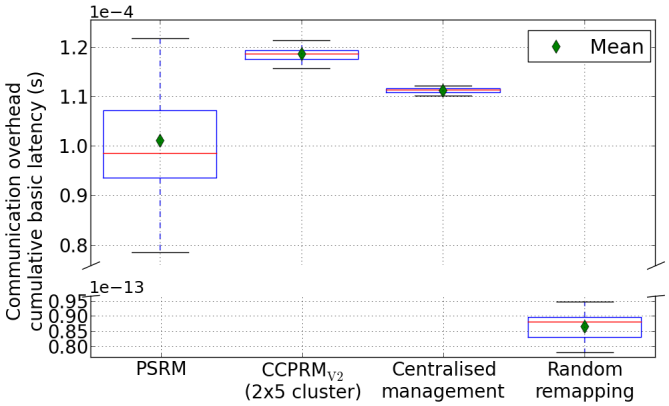

Fig. 7: Communication overhead of the remapping approaches

is marginally better than the CCPRM$_{V2}$ in the number of fully schedulable video streams. CCPRM$_{V2}$ shows a better job lateness improvement over the centralised management, because the monitoring traffic is shorter in route-length and hence is less disruptive to the data communication. We can see that the centralised management has the highest number of schedulable video streams out of the evaluated remappers. This could indicate that CCPRM$_{V2}$ and PSRM gave significant job lateness improvements only to a few video streams while the centralised management was able to make minor improvements to multiple video streams. The random remapper shows the worst results with a majority of the experiments resulting in negative improvements and produces the lowest number of fully schedulable video streams. It was interesting to note that there were a few scenarios where random remapping produced significant job lateness improvements, which is seen by the high upper whisker in the box plot (Fig. 5).

PSRM shows a significant communication overhead reduction when compared to the baselines (Fig. 7). The mean and IQR of PSRM communication overhead is lower than the baselines but the larger variance of the results is due to the different range of workloads and their effect on the QN differentiation cycle in each experimental run. The maximum overhead is comparable to that of CCPRM$_{V2}$. Both the CCPRM$_{V2}$ and centralised management show a higher and narrower distribution of communication overhead than PSRM. A higher upper whisker in PSRM shows that under certain workload scenarios the overhead can be costly and similar to the CCPRM$_{V2}$ baseline. The lower communication overhead

distribution of the centralised manager when compared with CCPRM$_{V2}$, is due to the lack of inter-cluster communication. In the centralised management scheme communicating tasks mapped at the middle of the NoC will suffer due to the network congestion caused by the incoming monitoring traffic. Furthermore, these traffic flows will occupy longer routes than CCPRM$_{V2}$. Furthermore, we are shown in [7], that the centralised managers' communication overhead issues become severe after the NoC size exceeds 12x12. The random mappers' communication overhead is many orders of magnitude lower than the others as it only incurs overhead when notifying the task dispatcher regarding remapping decisions.

The PE utilisation distribution shown in Fig. 8(a), indicates the PEs with higher utilisation using lighter shade, while the darker shades show PEs with low utilisation levels. The data shown in this plot are normalised such that each remapping technique is relative to each other. PSRM shows a slightly similar variation in the workload distribution to CCPRM$_{V2}$ with only a single PE with extremely high utilisation and a few with very low utilisation. The curves fitted to the histogram data shown in Fig. 8(b) indicates that all four remapping techniques have a similar spread of workload distribution. However, closer examination to the statistical properties of the distributions (given in Table III), indicate that centralised management has the lowest variance and the mean utilisation. The random remapper has the highest variance and mean utilisation. The frequency spikes of the centralised and random remappers in Fig. 8(b) at 0.8, 0.4 and 0.7 probably give rise to these statistical properties. PSRM shows a higher mean utilisation and lower distribution variance when compared with CCPRM$_{V2}$.

Overall the results indicate the PSRM technique helps to reduce lateness in the video stream jobs and to increase the number of schedulable video streams when compared with the CCPRM$_{V2}$ remapper. It is important to note that this improvement, even though is marginal, comes at a much lower communication overhead (up to 30% lower than the cluster-based and centralised approaches). A higher maximum lateness improvement can be obtained using CCPRM$_{V2}$, but only in 40%-50% of the workload scenarios. Communication overhead of CCPRM$_{V2}$ may grow as the cluster sizes increase, however in the PSRM technique this overhead will vary depending on the distribution of QNs in the network. Also, unlike in the baseline approaches, in PSRM the pheromone signalling messages are usually short (only a few hops) regardless of the NoC size increases. A centralised resource manager can help to evenly distribute the workload much better than PSRM, because of its global knowledge of the PE status and the mapped tasks. However, PSRM shows better workload distribution when compared with a cluster-based approach. Unlike in the cluster-based management, in PSRM, there are no resource managers in the network; each node executes a simple set of rules using only local knowledge to collectively improve the performance. The execution cost of the remapping event (Algorithm 2) is bounded by the number of QNs in the local vicinity and the number of tasks mapped on the node. In the cluster based approach each LMPs execution overhead for management functions (such as remapping, inter-core communication, monitoring etc.) would increase as the cluster size increases. Unlike in the centralised approach, PSRM is decentralised hence has no single point of failure

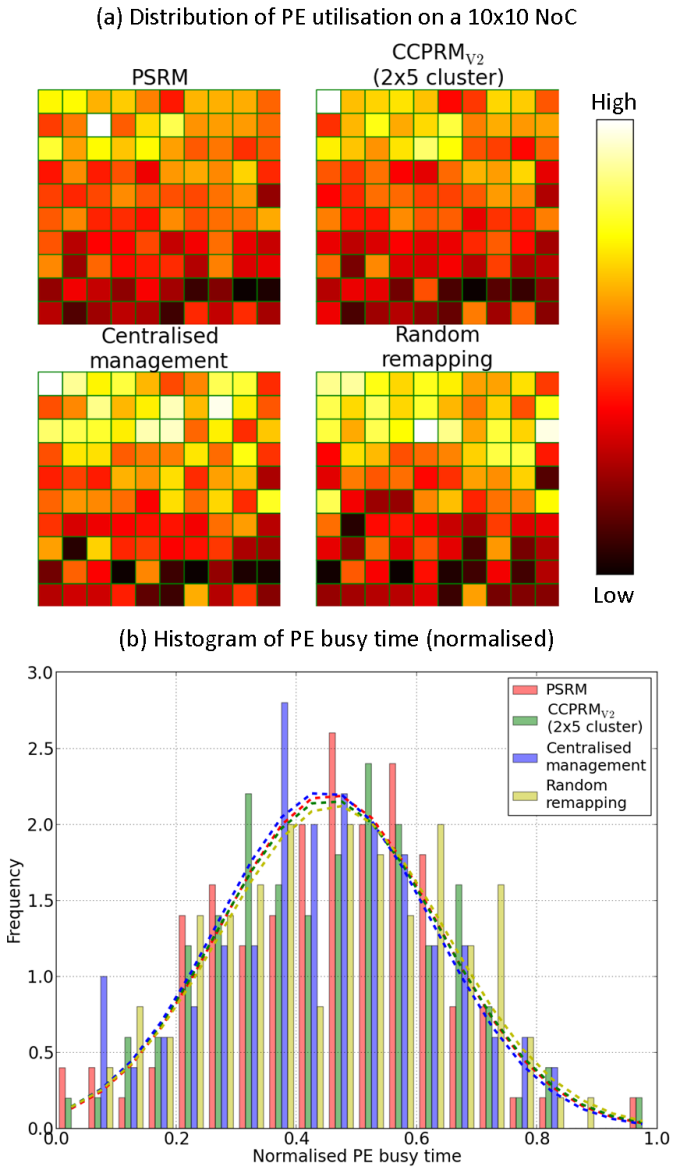

Fig. 8: Comparison of PE utilisation for all remapping techniques. (a) Distribution of PE utilisation across a 10x10 NoC (b) Histogram of PE busy time (normalised; 20 bins)

	mean	var.
PSRM	0.455	0.033
CCPRM$_{V2}$	0.454	0.034
Centralised management	0.447	0.032
Random remapper	0.462	0.035

TABLE III: PE utilisation distribution statistics. Lower variance (var.) = better workload distribution

or an isolated communication congestion area. Each PE has the capability of performing remapping and becoming a QN, hence the level of redundancy in the system is greater than in the baseline remappers.

VII. Conclusion and future work

This paper presented several significant extensions/adaptations to a fully distributed bio-inspired resource management technique as well as to a cluster-based management scheme. We have shown how these schemes can be applied to a multiple video stream decoding application with several unknown dynamic workload characteristics, on a NoC-based multicore. This work presents novel, low communication overhead, task-remapping strategies to progressively distribute the workload in the network and to reduce the overall job lateness.

The experimental results have shown that the bio-inspired remapper gives a marginal (2%-4%) improvement in lateness reduction but incurs 10%-30% lower communication overhead and minor improvement to workload distribution than the baseline cluster-based and centralised management schemes. The centralised management allows the system to increase the number of total schedulable video streams, but the improvement to the cumulative job lateness of the late video streams is poor and the communication overhead is higher than the proposed technique. The benefits of the centralised management degrade as the scale of the network and workload increase [5]. Results show that the proposed PSRM approach give a marginal benefit in reducing the cumulative job lateness of the video streams when compared against the CCPRM_{V2} cluster based resource management approach; however it is important to note that the improvement is obtained using a significantly less (up to 30% lower) communication overhead than the cluster based approach. A reduced communication overhead may lead to lower energy consumption [15] and less congested communication network, making PSRM more efficient than the cluster-based approach. Furthermore, unlike in the centralised or cluster-based approaches the proposed PSRM remapping technique does not depend on a single or group of management entities. Each node is independent and capable of relocating late tasks to improve the overall job latency, hence adopting this technique introduces a high degree of redundancy for NoC-based multi/many-cores that require reliable and timely operation.

Future work can include accounting for the communication traffic contention patterns in the remapping heuristics to improve the performance metrics. Further extensions to this work would include exploring distributed dispatching and admission control decisions depending on the status of queen nodes.

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] Y. Takeuchi, Y. Nakata, H. Kawaguchi, and M. Yoshimoto, "Scalable parallel processing for H.264 encoding application to multi/many-core processor," in *Int. Conf. on Intelligent Control and Information Processing (ICICIP)*, Aug. 2010.

[2] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang, "Understanding the Impact of Video Quality on User Engagement," in *SIGCOMM Conf.*, ser. SIGCOMM '11. ACM, 2011.

[3] D. Isovic, G. Fohler, and L. Steffens, "Timing constraints of MPEG-2 decoding for high quality video: misconceptions and realistic assumptions," in *Euromicro Conference on Real-Time Sys.*, 2003.

[4] E. de Souza Carvalho, N. Calazans, and F. Moraes, "Dynamic task mapping for MPSoCs," *IEEE Design Test of Computers*, vol. 27, pp. 26–35, 2010.

[5] M. A. Al Faruque, R. Krist, and J. Henkel, "ADAM: run-time agent-based distributed application mapping for on-chip communication," in *Design Automation Conference*, 2008.

[6] A. K. Singh, M. Shafique, A. Kumar, J. Henkel, A. Das, W. Jigang, T. Srikanthan, S. Kaushik, Y. Ha, and A. Prakash, "Mapping on multi/many-core systems: survey of current and emerging trends." in *Design Automation Conference*, 2013.

[7] S. Kobbe, L. Bauer, D. Lohmann, W. Schröder-Preikschat, and J. Henkel, "DistRM: distributed resource management for on-chip many-core systems," in *Int. Conf. on Hardware/software codesign and sys. synthesis (CODES+ISSS)*, 2011.

[8] U. Brinkschulte, M. Pacher, and A. Von Renteln, "Towards an artificial hormone system for self-organizing real-time task allocation," in *Software Technologies for Embedded and Ubiquitous Sys.* Springer, 2007, pp. 339–347.

[9] W. Trumler, T. Thiemann, and T. Ungerer, "An artificial hormone system for self-organization of networked nodes," in *Biologically Inspired Cooperative Computing*. Springer, 2006.

[10] P.-A. Mudry and G. Tempesti, "Self-scaling stream processing: A bio-inspired approach to resource allocation through dynamic task replication," in *Adaptive Hardware and Systems, NASA/ESA Conference on*. IEEE, 2009.

[11] I. Caliskanelli, L. S. Indrusiak, F. Polack, J. Harbin, P. Mitchell, and D. Chesmore, "Bio-inspired load balancing in large-scale WSNs using pheromone signalling," *Int. Journal of Distributed Sensor Networks*, 2013.

[12] G. Castilhos, M. Mandelli, G. Madalozzo, and F. Moraes, "Distributed resource management in NoC-based MPSoCs with dynamic cluster sizes," in *IEEE Computer Society Annual Symp. on VLSI*, 2013.

[13] C.-L. Chou, U. Ogras, and R. Marculescu, "Energy- and performance-aware incremental mapping for networks on chip with multiple voltage levels," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Sys.*, vol. 27, pp. 1866 –1879, 2008.

[14] O. Derin, D. Kabakci, and L. Fiorin, "Online task remapping strategies for fault-tolerant Network-on-Chip multiprocessors," in *IEEE Int. Symp. on Networks-on-Chip (NOCS)*, 2011.

[15] A. Das, A. Kumar, and B. Veeravalli, "Energy-Aware Communication and Remapping of Tasks for Reliable Multimedia Multiprocessor Systems." in *ICPADS*, 2012.

[16] A. Acquaviva, A. Alimonda, S. Carta, and M. Pittau, "Assessing task migration impact on embedded soft real-time streaming multimedia applications," *EURASIP Journal of Embedded Sys.*, pp. 1–15, 2008.

[17] J. Harbin and L. Indrusiak, "Dynamic task remapping for power and latency performance improvement in priority-based non-preemptive Networks On Chip," in *Int. Workshop on Reconf. and Communication-Centric Systems-on-Chip (ReCoSoC)*, 2013.

[18] U. Brinkschulte, A. von Renteln, and M. Pacher, "Measuring the quality of an artificial hormone system based task mapping," in *Int. Conf. on Autonomic Computing and Comm. Sys.*, 2008.

[19] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *Journal of Sys. Arch.*, vol. 50, pp. 105–128, 2004.

[20] B. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," *IEEE Trans. on Parallel and Distributed Sys.*, vol. 8, pp. 1268–1274, 1997.

[21] I. Caliskanelli and L. Indrusiak, "Search-Based Parameter Tuning on Application-Level Load Balancing for Distributed Embedded Systems," in *IEEE Int. Conf. on High Perf. Comp. and Comms. on Embedded and Ubiquitous Computing (HPCC_EUC)*, 2013.

[22] H. R. Mendis, L. S. Indrusiak, and N. 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 *Proc. of the 22nd Int. Conf. on Real-Time Networks and Sys.*, 2014.