Dynamic Resource Management of Network-on-Chip Platforms for Multi-stream Video Processing

Hashan Roshantha Mendis

University of York

Computer Science

March 2017

Abstract

This thesis considers resource management in the context of parallel multiple video stream decoding, on multicore/many-core platforms. Such platforms have tens or hundreds of on-chip processing elements which are connected via a Network-on-Chip (NoC). Inefficient task allocation configurations can negatively affect the communication cost and resource contention in the platform, leading to predictability and performance issues. Efficient resource management for large-scale complex workloads is considered a challenging research problem; especially when applications such as video streaming and decoding have dynamic and unpredictable workload characteristics. For these type of applications, runtime heuristic-based task mapping techniques are required. As the application and platform size increase, decentralised resource management techniques are more desirable to overcome the reliability and performance bottlenecks in centralised management.

In this work, several heuristic-based runtime resource management techniques, targeting real-time video decoding workloads are proposed. Firstly, two admission control approaches are proposed; one fully deterministic and highly predictable; the other is heuristic-based, which balances predictability and performance. Secondly, a pair of runtime task mapping schemes are presented, which make use of limited known application properties, communication cost and blocking-aware heuristics. Combined with the proposed deterministic admission controller, these techniques can provide strict timing guarantees for hard real-time streams whilst improving resource usage. The third contribution in this thesis is a distributed, bio-inspired, low-overhead, task re-allocation technique, which is used to further improve the timeliness and workload distribution of admitted soft real-time streams.

Finally, this thesis explores parallelisation and resource management issues, surrounding soft real-time video streams that have been encoded using complex encoding tools and modern codecs such as High Efficiency Video Coding (HEVC). Properties of real streams and decoding trace data are analysed, to statistically model and generate synthetic HEVC video decoding workloads. These workloads are shown to have complex and varying task dependency structures and resource requirements. To address these challenges, two novel runtime task clustering and mapping techniques for Tile-parallel HEVC decoding are proposed. These strategies consider the workload communication to computation ratio and stream-specific characteristics to balance predictability improvement and communication energy reduction. Lastly, several task to memory controller port assignment schemes are explored to alleviate performance bottlenecks, resulting from memory traffic contention.

Contents

List of Tables

List of Figures

Acknowledgements

First and foremost I would like to express my deepest and sincere gratitude to my supervisor Dr. Leandro Soares Indrusiak, for his patience and support throughout my EngD programme. I thank him for taking time to read my thesis and providing valuable feedback. He was always available for discussion and was always a great source of technical knowledge. Without his guidance this thesis would not have been achievable.

I would also like to thank Prof. Neil Audsley for his advice during the early stages of my research. I would like to thank my internal examiner, Dr. Christopher Crispin-Bailey for taking interest in my work, reading my thesis and providing useful comments at each milestone.

I thank my parents and my sister for their endless support and encouragement, for being in my life and caring for me. I want to thank my lovely, dear wife Shu-Chi Liu for her love and patience, for motivating me everyday and for constantly being by my side throughout the highs and especially the lows of this journey. I am blessed to have her in my life.

I would like to thank The Engineering and Physical Sciences Research Council (EPSRC) for funding this research through the UKs Large-Scale Complex IT Systems (LSCITS) programme (grant number EP/F501374/1). I would also like to thank my industrial supervisor Raj Patel, for providing use-cases and providing his insightful thoughts on my research.

My deepest gratitude goes to Dr. Ipek Caliskanelli for her advice and support, especially taking time to discuss the Pheromone Signalling algorithm in detail. A special thanks goes to Dr. Amit Kumar Singh, who took time to provide technical advice on runtime mapping techniques, especially the pre-processing baseline algorithm used in this Thesis. My thanks also to Dr. M. Norazizi Sham Mohd Sayuti for sharing his Genetic Algorithm framework which was extended in this work to act as a static mapping baseline. I also want to thank Dr. James Harbin, Dr. Andrew Burkimsher and Dr. Hashem Ghazzawi for discussing and providing technical guidance on my work on numerous occasions.

Lastly, I would like to thank my dear friends in and outside of York and at home, who have continuously supported me. I thank them for their companionship and wonderful memories.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References. Except where stated, all the work contained within this thesis represents the original contribution of the author. Some parts of this thesis have previously been submitted or published in the following conference/journal papers:

- Chapter 4:
  H.R. Mendis, L.S. Indrusiak, 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", *International Conference on Real-Time Networks and Systems (RTNS)*. ACM, 2014.

- Chapter 5:
  H.R. Mendis, L.S. Indrusiak, N.C. Audsley. "Task Allocation for Decoding Multiple Hard Real-time Video Streams on Homogeneous NoCs", *International Conference on Industrial Informatics (INDIN)*. IEEE, 2015.

  H.R. Mendis, L.S. Indrusiak, N.C. Audsley. "Dynamic and Static Task Allocation for Hard Real-time Video Stream Decoding on NoCs", *Leibniz Transactions on Embedded Systems (LITES)*, vol. 4, no. 2, pp 1-25.

- Chapter 6:
  H.R. Mendis, L.S. Indrusiak, N.C. Audsley. "Bio-inspired Distributed Task Remapping for Multiple Video Stream Decoding on Homogeneous NoCs", *IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, IEEE, 2015.

- Chapter 7:
  H.R. Mendis, L.S. Indrusiak. "Synthetic Workload Generation of Broadcast related HEVC Stream Decoding for Resource Constrained Systems", *International Conference on Signal Processing and Multimedia Applications (SIGMAP)*, 2016.

- Chapter 8:
  H.R. Mendis, L.S. Indrusiak. "Low Communication Overhead Dynamic Mapping of Multiple HEVC Video Stream Decoding on NoCs", *Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms (PARAM-DITAM)*, ACM, 2016.

Chapter 1

Introduction

1.1 Background and motivation

Traditionally, the computation power of a single core processor was improved by increasing the operating frequency. Power dissipation in clocked digital devices is proportional to the clock frequency; therefore, due to on-chip energy density constraints, increasing the processor frequency was no longer feasible. In order to keep Moore's law [1] alive, hardware manufacturers exploited a high number of slower logic gates, leading to parallel devices with duplicated on-chip elements such as general purpose processors, reconfigurable logic, memory and caches, working at lower clock speeds. Thus leading to massively multi-core (commonly referred to as *many-core*) chips with hundreds or thousands of low-frequency, low complexity cores [2]. Today, commercial many-core chips such as the Kalray-MPPA (256 cores) [3] or the Tilera TILE-Gx8036 chip (72 cores) [4] have been integrated into high data throughput applications such as data centre networking equipment, storage devices or broadcast video server products. Many-cores are not only seen in high performance or server devices but also in the consumer electronics domain. Mobile phones with 8-core multiprocessor system-on-chips (MPSoCs) are already in mass production [5], and the release of 10-core MPSoCs have been announced [6]. The International Roadmap for Semiconductors in 2012 [7], have predicted that the number of cores on-chip will double every 26 months. Hence, one of the greatest challenges is in providing the interconnection networks that allow these cores to communicate efficiently. Traditional hierarchical bus or crossbar based interconnects approach saturation as the on-chip data traffic scales up. To effectively tackle this interconnect complexity, a communication-centric design approach, *Network-on-chips (NoCs)*, has become the default choice of scalable, power-efficient, interconnect design paradigm for many-cores [8]. NoCs are inherently shared communication mediums; hence operations such as flow-control, arbitration and routing need to be efficient and fast, where performance is of concern and predictable/analysable where real-time guarantees are required.

The field of embedded multimedia electronics has evolved and grown technological maturity, driven by the ever increasing demand in application and user requirements. Multimedia, and more specifically in the context of this work, *video decoding* has been one of the first applications to embrace the massive parallel processing capability offered by multiprocessing platforms. Video playback and streaming services have become pervasive and an integral part of our life. With recent content delivery infrastructure and hardware advances consumers are now experiencing richer, more immersive content with every new generation of multimedia device. Merely ten years ago portable dedicated video players were becoming popular in the consumer electronic market [9]; now, a 4K-Ultra high definition (UHD) video (i.e. 3840×2160 resolution) can now be streamed and played in real-time on multicore mobile phones [10]. Modern

consumer electronics shows a great demand for computation power with a large constraint on power consumption and device size. After high-definition television (HD-TV) was introduced, video decoding alone now requires an average of 50 giga-instructions per second (GIPS) [11]. Video decoding has always been a computationally expensive process [12]. However, recent dramatic improvements in video compression [13, 14] has made video coding algorithms increasingly complex, making it necessary to utilise the parallelism offered by many-cores.

In Section 1.1.1 of this introductory chapter, the reader will be provided with several use-cases for real-time multi-stream video decoding especially indicating why strict predictability guarantees are required. Section 1.2 will then introduce the scientific research challenges addressed in this thesis and outline the approaches taken by existing work and where less work has been done previously. The thesis hypothesis will be clearly presented in Section 1.3 and Section 1.4 will explicitly outline the novel contributions made in this thesis to address the stated research problems. Section 1.5 will lastly end the chapter with the structure of this thesis.

1.1.1 Use-cases for predictable real-time multi-stream video decoding

Video streams have specific frame-rates, measured in frames per second (fps), which introduce timing requirements for video decoding and playback. The required fps will vary based on the application (e.g. higher fps for HD-TV than low bitrate video communications). Violating these timing constraints can lead to degraded quality of experience (QoE) [15], but the video decoding system can still continue to operate. Due to this reason, conventional video processing systems are considered to be *soft real-time*. However, there are several use-cases with *hard real-time* requirements on video decoding. Furthermore, as multimedia applications become more complex, there starts to exist the need for decoding of *multiple* video streams simultaneously.

The work presented in this thesis (particularly the first two technical chapters) places an emphasis on the management of multiple video streams with hard real-time timing requirements. The latter technical chapters focus on managing video streams with soft real-time timing requirements with complex characteristics. Managing *multiple* video streams with *hard real-time* requirements has not received much attention previously. This section will introduce several use-case scenarios where both hard and soft real-time video decoding of multiple simultaneous video streams are required.

1.1.1.1 Hard real-time decoding of multiple video streams

Hard real-time multiple video stream decoding has direct applicability in applications such as remote tele-surgery, autonomous vehicles and video surveillance. These application scenarios will be described briefly in this section.

Tele-surgery, is where a surgeon performs surgical operations with robotic tools to gain more control over traditional open surgery. *Remote tele-surgery*, is the same as tele-surgery, but the surgeon and patient are separated by significant distances (e.g. [16] [17]). In these systems, the surgeon controls the robotic tools whilst closely monitoring multiple video monitors which display the patient and the robot movement. Therefore, it is crucial that remote tele-surgery requires systems which can decode and playback multiple video streams within the hard timing requirement. Missing any frame decoding deadlines can result in unsafe surgical operations.

In *vision-based autonomous vehicle safety-critical systems*, multiple cameras are connected to the vehicle and the captured video streams are used to identify and actuate emergency measures [18]. These systems can guide and warn the driver of collisions and keep the passengers

safe during emergency situations. In such vision-based vehicle safety systems, the accuracy and functionality of the feedback controls depends on processing video frames with tight timing restrictions. Missing decoding deadlines can introduce latencies into the system, thus reducing its safety.

Next generation *automated video surveillance systems* (e.g. [19]) contain motion-tracking algorithms that make use of the multiple hard real-time video streams. These systems are used to ensure the safety and security of people by tracking hundreds of objects in real-time. For example in [19], the automated video surveillance system is used to notify healthcare services of fallen or injured elderly people living alone. If frames are dropped or their decoding deadlines missed, the motion detection algorithms may not operate correctly, which can result in delayed or failed response to threats/accidents.

1.1.1.2 Soft real-time decoding of multiple video streams

Several soft-real time video streaming applications now require decoding and playback of multiple videos in parallel due to increased user requirements. Unlike in the previously described hard real-time video streams, if soft timing requirements are not met, the system can continue to operate but with reduced QoE and negatively impacting the video service/business.

Multiple soft real-time video stream decoding can be seen in modern multi-view TVs [20], where the TV can stack and display more than one video at the same time. Users can isolate and view a specific video using special polarized glasses, thereby allowing multiple users to watch different videos on the same TV simultaneously. Also, in the digital television (DTV) industry, certain advanced graphical user interfaces (GUIs) now allow multiple video channel playback and video scaling, to aid TV users in selecting a program [21]. These features make use of available system resources to provide a rich and improved user-experience.

Multi-stream decoding is also a common requirement in the live broadcasting (e.g. BBC/Sky) where a wall of multiple videos is displayed and monitored for stream analysis and video errors [22]. In multipoint video conferencing systems [23] several video streams from different client devices need to be decoded and merged into a single frame; thus requiring considerable processing power and incurring delay. Another example is in high-end vehicles, where multiple audio-visual applications with varying quality of service (QoS) and priority/criticality levels need to be decoded and played-back simultaneously. These videos could be from high priority road-surveillance videos from vehicle mounted cameras for the driver, or low-priority entertainment videos for the passengers [24].

1.2 Research problems

This section will provide an overview of the primary scientific research problems addressed in this thesis, as well as highlight existing research work related to these problems. Note that a comprehensive survey of existing related work can be found in Chapter 2, therefore only a general overview of existing research will be given in this chapter.

Figure 1.1 shows a high-level illustration of the research problem investigated in this work, formed from the multi-stream real-time video decoding use-cases described in the Section 1.1.1. The figure shows multiple live video sources which need to be decoded by the many-core decoder and displayed in real-time. In this work, the multi-stream video decoding workload will

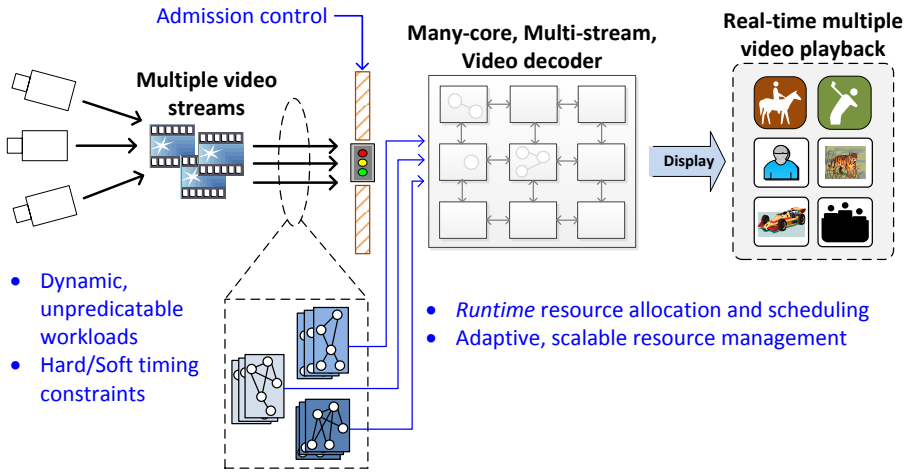


Figure 1.1: Overview of multi-stream video decoding using many-core platform

be modelled as real-time task-sets. This research will primarily look into how dynamic and unpredictable workloads with hard/soft real-time constraints can be allocated and scheduled in an adaptive and scalable manner. In the case where streams have hard timing requirements, an admission controller must provide guarantees before admission. Once admitted, the streams must be allocated efficiently to achieve the required performance and throughput constraint (i.e. fps).

1.2.1 The problem of shared-memory based parallel video decoding

There exists an evident gap between the parallel video decoding research community and the embedded multimedia design research community. Many work in the state-of-the-art in parallel video decoding assume a multi-threaded shared-memory many-core architecture (e.g. [25], [26]). In these architectures, performance improvement saturates as the number of cores and threads increase, due to resource contention. On the other hand, the state-of-the-art NoC-based many-core embedded multimedia systems design assumes a task-level parallel video decoder model (e.g. [27], [28]) which has limited parallelism and flexibility compared with data-level parallel video decoders [29]. These designs also often assume a communication-centric, message-passing based NoC architecture, which are more difficult to program but alleviates the performance bottlenecks of a shared memory, multi-threaded design. In order to improve modern multi-core multimedia systems designs to deliver increased throughput, this separation needs to be reduced.

1.2.2 The problem of resource management for unpredictable workloads

Unpredictable workloads are those which characteristics (e.g. arrival patterns, computation, communication and memory requirements) are only known at runtime and not at design time. Many live video streams fall into this category as they have characteristics that are difficult to determine and predict beforehand. In such scenarios, hard predictability guarantees are difficult to provide and resource management has to be performed at runtime, instead of pruning the allocation search space at design time [30]. As modern video codecs become more sophisticated and include content-adaptive algorithms their dynamicity also increases.

Existing work in resource allocation for *dynamic* workloads focus on *runtime task allocation/re-allocation* (e.g. [30], [31]). Many of these techniques rely heavily on monitoring platform resource usage at runtime in a feedback-loop to adapt to the dynamic workload. However, monitoring system resources in a feedback-loop can in turn lead to higher communication network use. Resource managers that do not employ such feedback monitoring (i.e. open-loop management) have received significantly much less attention in the existing literature. Some of the existing techniques assume exact computation/communication requirements of the workloads are known beforehand (e.g. [32], [33]), which is not the case with live video streams. On the other hand, some assume worst-case workload properties (e.g. [34]) to provide hard real-time guarantees, which can lead to resource over-provisioning and underutilisation. In a multi-stream video decoding context this could mean the rejection of potentially serviceable video streams. Very few existing work have combined end-to-end worst-case response time analysis in runtime resource management schemes (e.g. admission control) for predictable NoC architectures.

1.2.3 The problem of scalable management

Scalability is another challenge addressed in this work to support the continuous increase in the number of processing cores in a platform and the scale of the workload. Existing *centralised* resource management has been successful in the past in smaller systems, but has proven to quickly become a bottleneck as the platform size and workload grows. Existing state-of-the-art *hierarchical* and *fully distributed* techniques also can have complex and *high management overhead* in terms of their communication protocols (e.g. [35–37]). Low overhead fully distributed resource management protocols are required for future large-scale NoCs.

Many existing work consider relatively small applications (e.g. [33]) in their evaluations. However, resource management should also take into account the *scale of the workload*, as the number of tasksets and their rate of arrival can increase as applications become more sophisticated and complex. Runtime task admission and allocation algorithms should therefore still maintain an acceptable execution overhead when faced with large-scale workloads.

1.2.4 The problem of resource sharing

An important aspect of NoC-based many-cores is the sharing of on/off-chip resources (processing elements, communication channels, shared memory, input/output ports etc.) by concurrently running processes or applications. On one hand, many-cores offer increased parallelism to meet the expectations of the embedded and high-performance computing market. On the other hand, the complexity of the software deployed on these systems grow exponentially requiring more shared resources and computation power. Resource sharing leads to higher system utilisation and increases average performance resulting in low-cost design alternatives.

However, resource sharing also leads to *resource contention* causing individual applications to become blocked and wait until a resource is free to use, resulting in unpredictability and delays to execution. The trade-off between the conflicting goals, *predictability* and *performance* is a challenging yet key design decision, driven by use-case requirements. With the emergence of many-core systems, blocking scenarios will effectively become a critical issue in the design of real-time systems. Resource contention due to sharing of resources amongst tasks (i.e. sub-partitions of applications) over multiple cores can easily leave cores in idle states, thereby degrading system performance, impacting schedulability of real-time systems and wasting the

benefits provided by many-cores.

Different task to processing core allocations can result in varying resource contention patterns on the platform [30]. Thus, efficient task allocation (also referred to as *task mapping*) and scheduling policies, combined with application *domain knowledge*, need to be designed to alleviate the causes of resource contention. Many of the existing runtime task allocation techniques consider metrics such as communication cost, utilisation and execution time in their mapping heuristics (e.g. [38], [39], [37]). However, very few runtime mapping heuristics take into account resource contention and even less consider improving predictability metrics (such as reducing *lateness*).

1.3 Research hypothesis

Section 1.2 introduced scientific research challenges and existing research work related to the use cases presented in Section 1.1.1. The main objective of this research is to explore and design predictable and efficient, scalable, low overhead NoC resource management (admission control, allocation/re-allocation) techniques for unpredictable multi-stream video decoding applications. Therefore, the following thesis hypothesis determined the central focus of this research. The thesis hypothesis is divided into two domains with respect to *classical* and *modern* coding tools:

- For videos encoded using classical video codecs: *Application and blocking-aware runtime mapping heuristics combined with a deterministic admission controller can be used to guarantee timing requirements and improve system utilisation for hard real-time video streams; a low-overhead, distributed, remapping technique, can be further used to reduce the lateness, of soft real-time video streams.*
- For videos encoded using modern video codecs: *Application-specific task clustering and mapping combined with better memory controller selection heuristics, can be used to balance communication cost and lateness reduction of soft real-time decoding of complex video streams.*

1.4 Thesis contributions

The following explicitly outlines the novel contributions made in this thesis to address those research problems identified in Section 1.2 and to achieve the thesis hypothesis set above in Section 1.3:

- **A communication-centric data-parallel video decoding application model:** This model suits modern message-passing NoC architectures with distributed local memory. Its hierarchical structure enables modelling characteristics of multiple simultaneous video streams. The initial proposed application model targets *classical* video codecs (e.g. MPEG-2) with certain stream constraints. Latter technical chapters present a more flexible and expressive application model, which both captures the content-adaptive nature of *modern* video codecs (e.g. H.264/High Efficiency Video Coding - HEVC) and block-level characteristics, not been analysed before. Novel, workload generation algorithms are also presented which make use of real stream analysis, to synthetically produce abstract workloads.

- **Deterministic and heuristic-based admission controllers:** This thesis presents a novel deterministic admission controller, which uses end-to-end NoC schedulability analysis to provide hard timing guarantees, for multi-stream video stream decoding workloads. Also, proposed is a soft real-time heuristic-based admission controller which uses task lateness as a metric to balance predictability and utilisation in the system.
- **Application and blocking-aware runtime task mapping techniques:** These two allocation techniques are used to improve the low-utilisation of deterministic admission controllers. Unlike existing mapping techniques, the first proposed approach takes into account task contention and attempts to pack tasks tightly onto a PE whilst trying to reduce deadline misses. The second mapping heuristic adapts the first approach and exploits the limited knowledge of the video stream structure to further improve the system utilisation.
- **A distributed, bio-inspired, task re-allocation technique for NoCs:** This proposed task remapper has low communication overhead and can adapt to varying workloads autonomously, without centralised/cluster-based management. Thus, it is suited for large-scale NoC architectures handling soft real-time workloads. This proposed runtime re-allocation technique is used to load-balance the system as well as further reduce the lateness of accepted video streams.
- **Task clustering/mapping and memory traffic management for HEVC video streams:** Two runtime task clustering and mapping techniques are proposed to improve predictability and reduce NoC communication cost, targeting highly dynamic, soft real-time HEVC video streams with tile-level data parallelism. The first approach attempts to cluster and map the tasks in the longest path of the task-set in close proximity, in order to reduce the NoC usage. The second approach exploits stream-specific video frame relationships to derive a novel clustering technique to balance both the aforementioned objectives. Lastly, several main memory controller port to task allocation heuristics are explored. These heuristics attempt to balance the memory traffic load and also to mitigate memory controller contention.

1.5 Thesis outline

The remaining chapters of this thesis are organised in the following structure:

Chapter 2 surveys the existing literature in three strands: *1)* the video decoding application domain and its complexities; *2)* components and design challenges in predictable many-cores and NoC interconnects; *3)* compare and contrast the state-of-the-art in dynamic resource management techniques and methodologies for NoC-based many-cores.

Chapter 3 firstly present the preliminary system models (application and platform), which form the basis of the evaluation infrastructure in this thesis. Later chapters will refine and lift certain assumptions in these models. Secondly, the evaluation metrics focused in this thesis (related to predictability, performance, energy and efficiency) are defined. This chapter ends with an outline of the overall research problems this thesis is attempting to address.

Chapter 4 provides the first technical contribution in the form of two admission control approaches for multiple video stream applications executed on NoC-based platforms. A

deterministic admission controller is proposed for hard real-time video streams and a heuristic-based admission controller is proposed to trading-off/balance predictability and utilisation for soft real-time streams.

Chapter 5 extends the work in Chapter 4, by presenting two novel hard real-time task allocation heuristics to improve the low-utilisation of deterministic admission controllers. These heuristics consider task blocking and incorporate limited application-awareness to reduce the video streams' end-to-end worst-case response time. They are evaluated against existing NoC task mapping techniques, as well as a design-time task mapper. Extended evaluations related to platform scalability and communication loads are also included.

Chapter 6 explores the feasibility of using a low-overhead, distributed and biologically-inspired task re-allocation protocol, to improve the response-time of video stream decoding with soft timing requirements. This technique is compared against a state-of-the-art hierarchical many-core resource management technique and a centralised manager.

Chapter 7 extends the video decoding application model to consider the complexity of modern codecs, specifically the H.265/HEVC standard. Statistical properties of real video streams such as frame dependency patterns and decoding execution profiles are characterised. A novel bottom-up HEVC video decoding synthetic workload generation framework is introduced.

Chapter 8 utilises the workload model generated in the previous chapter to address challenges in multi-stream HEVC video decoding, with tile-level parallelism and soft timing constraints. Novel, application-aware, runtime task clustering and mapping heuristics are used to balance the overall NoC communication, predictability and mapping overhead. Memory controller to task mapping heuristics are explored to reduce memory traffic congestion in the NoC.

Chapter 2

Literature Survey

This chapter provides an overview and discussion of previous research in fields related to parallel video decoding, network-on-chip and many-core resource management. The sections in this chapter are structured in a top-down approach using a Y-chart methodology as presented in Figure 2.1. Kienhuis et al. [40] introduce this Y-chart approach to prune the embedded systems design space, where the application and architectural models are decoupled and improved iteratively. Similarly, this chapter first introduces parallel video decoding applications in Section 2.1, followed by a discussion of network-on-chip based many-core architectures and their challenges in Section 2.2. Section 2.3 discusses many-core resource management in terms of mapping, scheduling and runtime adaptation mainly for multimedia workloads.

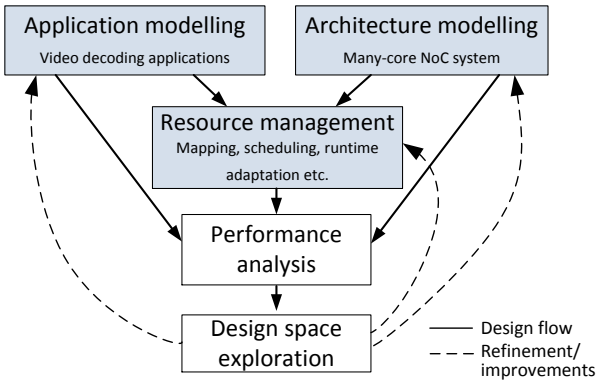


Figure 2.1: Y-chart methodology of reviewing the literature (taken from [40])

2.1 Video decoding applications

An overview of video decoding is presented in this section. The challenges and bottlenecks of these algorithms are analysed and steps taken to parallelise video decoding are discussed.

2.1.1 Video stream decoding overview

Media files are extremely large in their original form (e.g. a 1 minute duration, 4K (3840×2160), 8 bits per pixel (bps) video file, is approximately 33GB large). Therefore, they must be encoded (compressed) before they can be stored or transmitted. Video compression is typically performed by exploiting spatial and temporal redundancies. There exists multiple video compression standards such as MPEG-2, MPEG-4 Part 10 (H.264), H.264/AVC (Advanced video codec) and the latest H.265/HEVC (High Efficiency Video Codec). With each new version of video codec

new coding techniques are used, resulting in increased complexity and requiring more computational resources.

A video stream is a sequence of compressed (encoded) frames (also referred to as pictures). A video sequence can be hierarchically broken down into groups of pictures (GoPs), frames, slices, macroblocks and finally blocks as shown in Figure 2.2a [41]. Frames can be an intra-predicted (*I*-frames) or inter-predicted (*P* or *B* frames). I-frames are reconstructed (decoded) using information within itself. P-frames are forward predicted and require reference data from a recently reconstructed I or P frame. B-frames use bi-directional prediction and can reference past and future I, P or generalised B-frames. For example, in Figure 2.2b, B_2 obtains reference data from I_0 and P_3. *Generalised B-frames* as shown in Figure 2.2d can be referenced by other B-frames, in a hierarchical B-frame coding structure as used by advanced codecs such as H.264/AVC and HEVC [13, 42].

Encoders predict the current P/B frame by using the reference frames and only transmits the residual (difference) and the motion vectors (if any) to the decoder. At the decoder, the residual values are added with the reference frame(s) macroblocks and shifted using the motion vectors (MVs) to decode the frame. As illustrated in Figure 2.2c, a frame's blocks can refer to other blocks in *multiple reference frames* (H.264/AVC onwards). Multiple reference frames improves compression at the expense of higher encoder complexity. The decoded picture buffer (DPB), located in memory stores all decoded frames until they are no longer required by any of the future frames in the encoded video stream. Frames are decoded and displayed in the *display order*, which is different to the *decoding order* at the encoder output, to facilitate bi-directional decoding. Hence, the decoded frames need to be *re-ordered* before displaying.

Figure 2.2b shows an example of a *closed GoP* with 12 frames. Closed GoPs do not have dependencies with other GoPs in the video stream and they reduce error propagation, but provide slightly less compression than open GoPs. Generally, all the GoPs in the video stream have a fixed number of frames and the number of P or B frames within the GoP may change depending on video content variation. According to the MPEG-2 specification, the 12 frame GoP structure given in Figure 2.2b, is recommended to balance compression, facilitate stream random-access and to manage error propagation [43]. Fixed GoP structures are also simpler to implement. They enable a certain level of performance analysis of the decoder beforehand but is poor in coding efficiency as temporal variations in the video are not accounted for. Therefore, adaptive GoP algorithms have been presented to detect valid scene-changes, reduce cumulative encoding errors and improve compression [44, 45]. These techniques vary the GoP length and structure depending on detected scene-change events and temporal variations.

2.1.1.1 Functional blocks of a video decoder

A video *encoder* and *decoder* are complementary pairs of systems, containing coding tools (i.e. algorithms and models) to exploit temporal and spatial redundancies. This thesis focuses solely on decoder resource management, hence the decoder characteristics are inspected closely. A block diagram of a generic MPEG-2 decoder is given in Figure 2.3a. A variable length coder/decoder (VLC/VLD) is used to reduce the codeword length of image data leading to lossless, bit-rate reductions. Quantisation performs lossy compression to reduce a range of values to a single value. 2D inverse discrete cosine transform (IDCT) is frequency to spatial domain transformation technique. Motion compensation is used to decode the inter-predicted frames using the frame data in the DPB. Typically, macroblocks/blocks run through this decoder pipeline, one

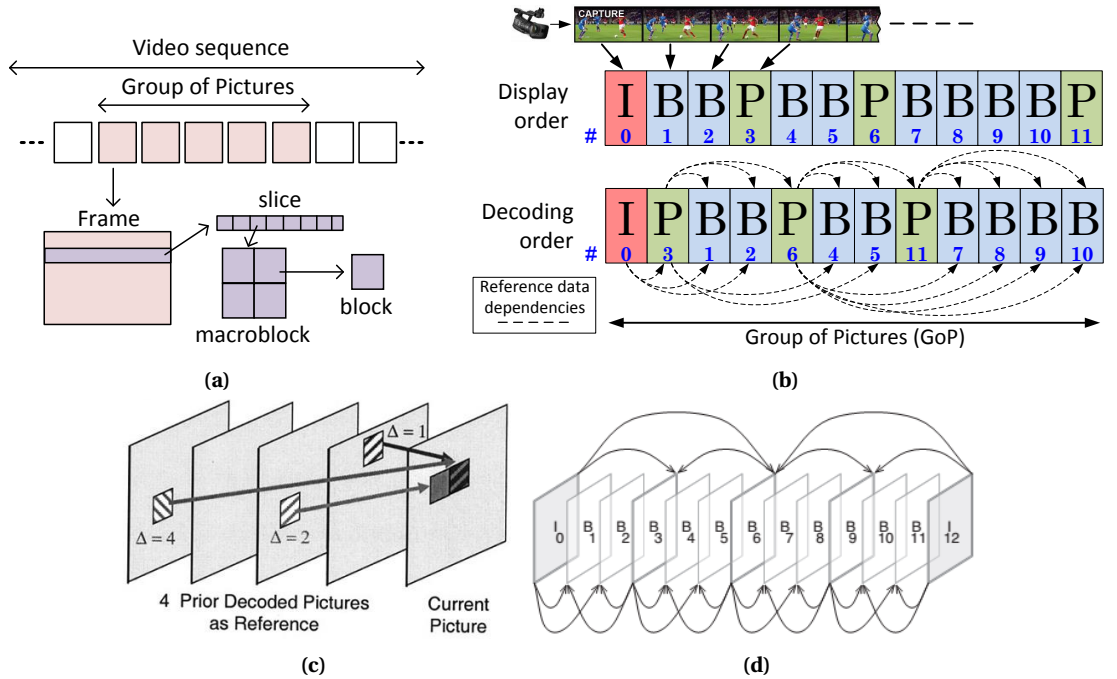


Figure 2.2: (a) Hierarchical structure of a video sequence (b) Frame ordering and reference frame dependency pattern in an open-GoP (c) Multi-frame motion compensation (Δ=reference picture index in the reference picture buffer), taken from [42] (d) Example of using generalised B-frames in a hierarchical B-frame, open GoP structure (taken from [46])

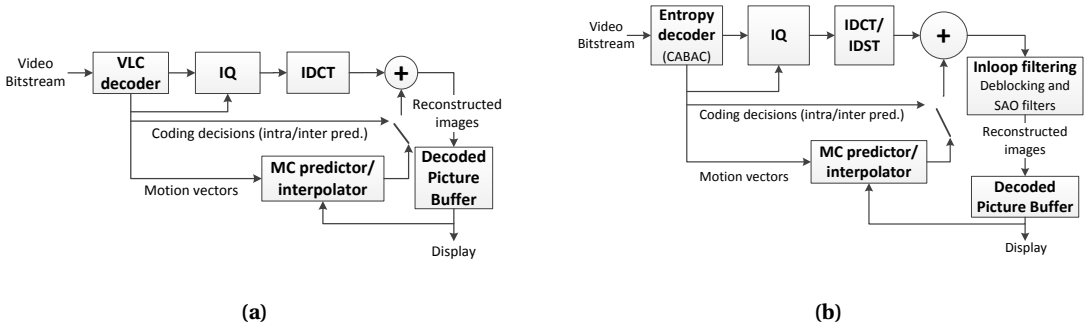


Figure 2.3: High-level block diagrams of: (a) Generic MPEG-2 decoder (adapted from [42]) (b) Generic HEVC decoder (adapted from [47]). (VLC=variable length coding; IQ=inverse quantisation; IDCT/IDST=Inverse discrete cosine/sine transform; MC=motion compensation; CABAC=context-adaptive binary arithmetic coding; SAO=sample adaptive offset)

after the other.

The HEVC decoder (Figure 2.3b) contains a similar set of high-level functions; however, each significantly more complex than MPEG-2. Two main differences in HEVC decoding is the use of a context-adaptive binary arithmetic coder (CABAC) entropy coder, and in-loop filtering. CABAC uses statistical properties to perform lossless compression, and in-loop filtering is used to improve the quality of the image and reduce the visual artefacts introduced during compression [47].

2.1.2 Features of the H.265 (HEVC) coding standard

The preliminary technical chapters in this thesis assume the video stream decoding workloads are MPEG-2 encoded with simple coding tools. MPEG-2 is still in use in industry, specifically in the digital TV domain to encode standard definition (SD) broadcast streams to support legacy

devices and MPEG-4 or H.264/AVC is used for HD streams. H.265(HEVC) has generated huge optimism given the broadcast industry's struggle with lack of bandwidth and the need to deliver HD/UHD content to multiple platforms. HEVC offers many features which make resource management challenging, hence the last two technical chapters of this thesis focus on resource allocation issues surrounding HEVC encoded video streams. This section therefore aims to provide the reader with a brief overview of the key components of the HEVC standard related to this research.

The HEVC coding standard was primarily developed to address compression problems with very high video resolutions (beyond 1920×1080) and promoting the use of parallel architectures. HEVC is able to offer over 50% bit-rate savings than H.264/AVC with approximately similar subjective quality and Peak Signal to Noise Ratio (PSNR) levels [48]. At the time of writing, the third version of the HEVC standard and its extensions have been released [14]. HEVC aims to improve the compression (coding efficiency) by using several improved coding tools (e.g. transform sizes, intra-prediction modes, in-loop filtering changes given in Table 2.1). The macroblock structure in previous codecs have been replaced in HEVC by coding tree units (CTUs). The CTU logical structure can be of the size 64×64, 32×32 or 16×16 depending on the stream parameters. Compared to previous standards (Table 2.1), larger CTU sizes provides better compression for higher resolution videos [48]. CTUs are recursively partitioned using a quadtree structure into smaller coding units (CUs) as shown in Figure 2.4a. I-frames only contain intra-coded CUs, P-frames can contain intra/inter(forward) coded CUs and B-frames can contain intra/inter(forward)/inter(bi-directional) coded CUs. CUs are flagged as Skip if they do not have a residual or motion vector, hence a reference-CU is copied directly. Details of other new coding tools such as merge mode for MVs, asymmetric prediction units, motion interpolation filtering etc. can be found in [47].

Table 2.1: Comparison of MPEG-2, H.265/AVC and HEVC [13, 47]

Features	**MPEG-2**	**H.265/AVC**	**HEVC**
Macroblock size	16×16	16×16	64×64, 32×32, 16×16
Block size	8×8	16×16, 16×8, 8×16,8×8, 8×4,4×4	64×64 to 4×4 (symmetric/asymmetric)
Transform	DCT (8×8)	DCT (8×8, 4×4)	DCT (32×32, 16×16, 8×8, 4×4), optional DST for 4×4
Intra-prediction modes	1	9	35
Reference frames	One	Up to 16 frames	Up to 16 frames
Native parallelisation	None	slice level	slice level, WPP, Tiles
In-loop filtering	None	De-blocking filter	De-blocking and SAO filtering

In addition to the slice-level parallelisation offered in H.264/AVC, HEVC contains several native mechanisms to make the codec better parallelizable [29]. An HEVC frame can optionally be partitioned into *Tiles* (Figure 2.4b), where Tiles are approximately equal rectangular picture regions which can be independently decoded in parallel. When *wavefront parallel processing* (WPP) is enabled in HEVC, each CTU row of a frame can be decoded in parallel. However, each decoding thread can only begin decoding each CTU row with a 2 CTU delay relative to the previous row (Figure 2.4b), due to the entropy decoding synchronisation requirements [47]. A detailed discussion regarding decoder parallelisation is given in Section 2.1.4.

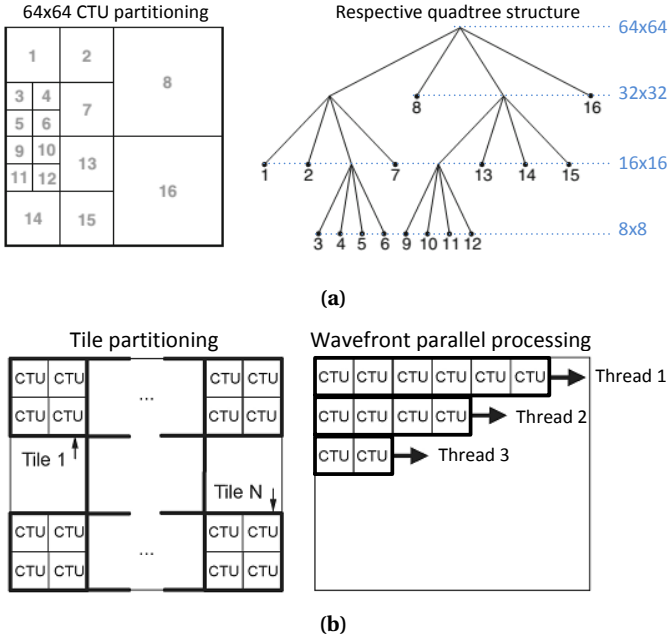


Figure 2.4: (a) HEVC CTU partitioning and respective quadtree structure (taken from [47]) (b) Tiles and Wavefront parallel processing in HEVC (taken from [13])

2.1.3 Complexities of video decoding

New coding tools in each generation of video codec comes at the expense of increased implementation cost. Early work on memory usage and computation complexity analysis of MPEG-2 and H.264 decoding have been carried out by Holliman et al. [49]. They show that total MPEG-2 decoding execution costs are about 8 times slower than H.264 for a video sequence with the same resolution and bitrate. The MPEG-2 decoder spends more time in motion compensation (MC) for low-bitrate videos, but spends more time performing IDCT, VLD and IQ for higher-bitrate videos. Similarly, for H.264, VLD becomes the bottleneck as the bitrate is increased. Motion compensation in MPEG-2 is memory intensive and memory architecture details such as L1/L2 cache size, cache line size, memory bandwidth etc. can impact decoding time. However, in H.264 and H.264/AVC unlike in MPEG-2, MC is mainly computation-bound for low bitrate videos and the peak memory usage highly depends on the number of reference frames per intra-predicted frame [49, 50]. Furthermore, the decoding time of H.264/AVC is also dependent on the quantisation parameter (QP) (i.e. level of quality loss) of the encoded video; where higher QP values decreases the decoding time. In-loop deblocking filtering in H.264/AVC increases memory accesses by 6% and overall processing time increase by 10% [50]. The use of B-frames increases the decoder execution time by 20-40%, hence generally not used by low-latency applications (e.g. video conferencing).

Preliminary decoder performance analysis of the HEVC reference software decoder is performed by Bossen et al. [51]. Their results show that, when performing intra-only HEVC decoding (no P/B frames), the IDCT and in-loop filtering components takes the most amount of time. However, when decoding random-access streams (i.e. P and B-frames are used), the motion-compensation process takes 50% of decoding time, which matches the same trend in results as H.264/AVC decoding [50]. Random-access stream decoding has a higher decoding time due to memory subsystem issues, such as bandwidth saturation and cache misses when fetching reference frame data. Depending on the type of video content and bit-rate required, certain

advanced coding tools can be disabled without sacrificing significant quality loss. Using less complex coding tools would greatly decrease the computation complexity and memory usage of modern codecs [50].

2.1.4 Parallel video decoding

The state-of-the-art in parallel video decoding techniques can be categorised into *task-level*, *data-level* decomposition or a hybrid of the two partitioning approaches [29]. The two approaches are illustrated in Figure 2.5. In task-level decomposition (e.g. [52–54]) the functional components of the decoding algorithm (e.g. IDCT, VLC, IQ etc.) are assigned to different processing elements in the platform. Some of these tasks are pipelined and/or done in parallel. Task synchronisation and a significant amount of communication is required between the tasks to perform the decoding in this technique [54]. Task dependencies and varying task pipeline execution times can also lead to load imbalance. Scalability is difficult to achieve in this parallel approach as decoding higher resolution video streams would require the tasks to be partitioned differently to achieve high throughput demands. Furthermore, each processor/task would need to implement specific hardware/software optimisations to obtain better performance, hence more suited for platforms with heterogeneous processing units [53] or with dedicated hardware units [52].

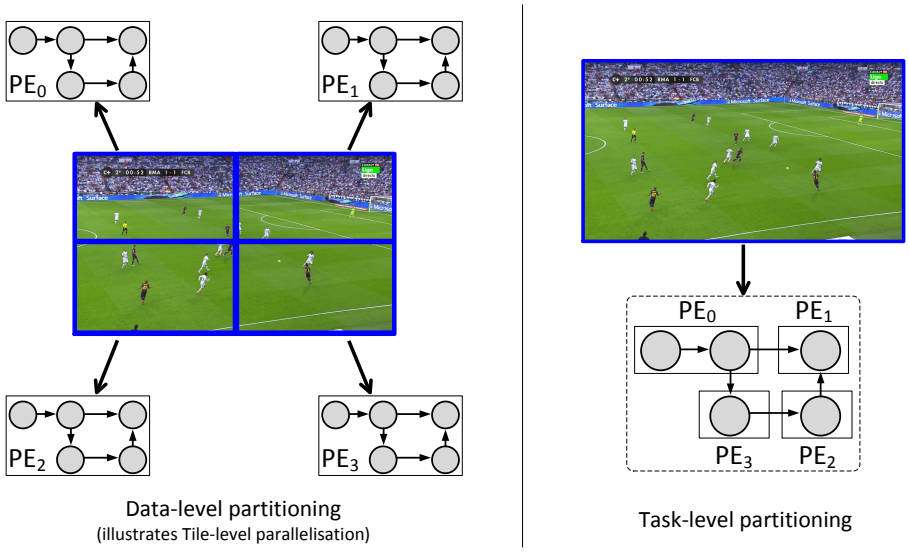


Figure 2.5: Video decoding parallel partitioning approaches: data-level (left) and task-level (right) partitioning

Figure 2.5 illustrates the data-level parallel video decoding approach which is commonly used by the video processing research community (e.g. [25, 26, 29]). In this approach, the data (encoded video stream) is divided into small parts and each assigned to a different processor or processing thread. Partitioning of the video stream data could be coarse-grain (e.g. GoPs/frames/HEVC Tiles) or fine-grain (e.g. macroblocks/CTUs). Meenderinck et al. [29] compare data-level parallelisation techniques at different levels of granularity. *GoP-level* parallelism is simple to implement, but requires a large amount of memory. Frame-level parallelism is a generic approach, applicable to different video coding standards and do not require special signalling or instrumentation in the video bitstream. However, the scalability of *frame-level*

parallelism is limited by the number of parallel B-frames in the GoP and has high communication overhead due to inter-frame data dependencies. Figure 2.6(b) shows the results obtained by Bross et al. [55], when evaluating the scalability of random-access (i.e. P/B frames enabled) vs. intra-only video profiles. Random-access profiles offer superior compression over intra-only videos; however, due to the inter-frame dependencies they scale poorly. The scalability issues with frame-level parallelism is however not a concern when attempting to decode multiple video streams, as idle processors can be utilised to decode frames from an alternate video stream. *Slice-level* parallelism has less dependency issues, but is affected by in-loop filtering; furthermore, increasing the slices in a frame can increase the bitrate. Parallelism at the *macroblock-level* offers a high degree of scalability and lower memory usage, but stream instrumentation and additional delays will be incurred to parse and determine the macroblock start/end locations in the bitstream [29].

2.1.4.1 Parallel HEVC stream decoding

As described in Section 2.1.2, HEVC contains two native data-parallel techniques: *Tiles* and *WPP*. Both Tiles and WPP require explicit video bit-stream signalling, to indicate where tiles/CTU rows start and end. The scalability of WPP decreases as the number of CTU rows increase, due to synchronisation issues. Tiles start to incur coding losses as the number of Tiles in a frame is increased [56]. Georgakarakos et al. [25] experimentally show that Tile-level parallelisation offer better speed-ups when compared with native WPP as the decoder thread count is increased. Chi et al. [26] improve the WPP technique to mitigate the synchronisation issues and combine it with frame-level parallelism. Their approach, termed overlapped wavefront processing (OWF) show better scalability than Tiles or WPP and the results are given in Figure 2.6(c). Further results from their same work [26], given in Figure 2.6(a) show, parallel decoding of multiple lower resolution videos scale better than a lower number of very high resolution videos. A common trend in all data-parallel approaches shown in Figure 2.6 is that, performance improvement saturates as the number of cores/simultaneous threads are increased. This behaviour is due to the heavy contention on the shared memory sub-system (e.g. memory bandwidth, main memory controllers).

Hybrid decomposition techniques (e.g. [57, 58]) attempt to combine different task-level and data-level parallelisation strategies. In [57], CTU-level wavefront data-parallel processing is employed to decode CTU pixels, but task-level parallelism is used for entropy decoding. They use explicit barrier synchronisation to manage the decoding pipelines. Many hybrid approaches speed-up compute intensive functions of the decoder using architecture specific optimisations such as Single Instruction Multiple Data (SIMD). However, SIMD optimisations are orthogonal to data-level/task-level parallelism as SIMD can be used to improve the performance of each individual core/thread [29]. In [59], SIMD optimisations are used at each decoder stage to obtain a 2.8x-5x speed-up improvement over a combined WPP and frame-level parallel approach.

2.1.5 Challenges in characterisation of video decoding workload

In order to balance complexity and compression performance, most modern video codecs (such as HEVC) are highly content-adaptive [13]. Therefore, different video sequences under certain encoder settings can produce highly dynamic and unpredictable workload characteristics. For example:

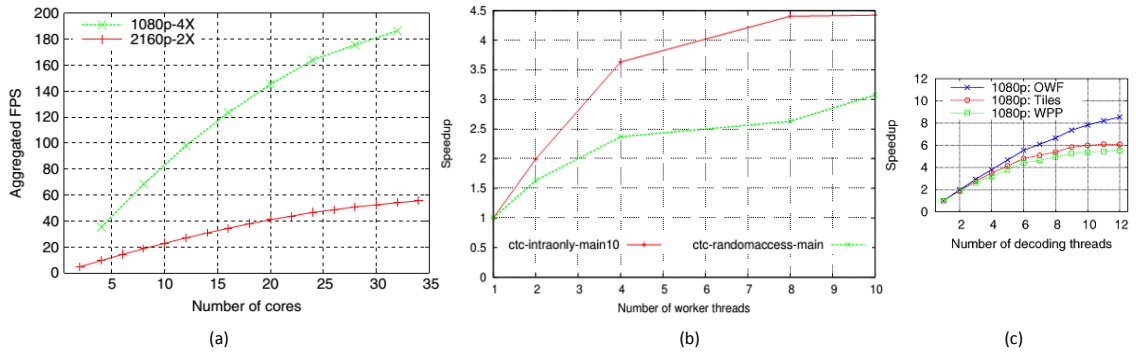


Figure 2.6: HEVC decoder performance results: (a) Using combined OWF and frame-level parallelism (for multiple stream decoding on a Tilera TILE-Gx8036 processor) (taken from [26]) (b) Scalability of intra-only vs. random-access (inter-prediction enabled) videos; speed-ups compared to a sequential decoder (taken from [55]) (c) Scalability comparison between OWF vs. Tiles vs. WPP (taken from [56])

- **Decoding time:** Frame decoding execution cost can vary 2-3 times between consecutive frames in the same stream; however, variation between frames of the same type are small [11]. Decoding time is also generally proportional to resolution. Intra-prediction generally takes more computation time than inter-prediction; higher number of I-macroblocks in a frame can cause the decoding time to increase. I-frame decoding time varies least due to less coding options, and B-frames vary the most [60]. The execution time of certain functional components in the decoder (e.g. interpolation, entropy decoding, de-blocking filtering) can also vary significantly, depending on the type and size of macroblocks in a frame. Hence, the computation cost of video decoding cannot be known a priori.

- **Timeliness:** Timing requirements of a video decoding task is determined by the required *frame rate* (i.e. how many frames need to be displayed per second). For example, a video decoder which has to perform at 25 frames per second (fps) has to decode and display a single frame within 0.04s (1/fps). For video streams that use bi-directional prediction (i.e. B-frames), a frame *re-ordering* delay is introduced to put the frames in the right order for displaying. The re-ordering process complicates real-time analysis of a video decoder as it becomes difficult to determine the decoding deadline of individual frames [60]. Using frame display deadlines implicitly to calculate the decoding deadlines as in [61], can lead to optimistic timing analysis.

- **Arrival rate:** The arrival rate of streaming video depends mainly on the video bitrate and delivery networks. If a video is encoded at a variable bit rate (VBR), the encoder would allow higher bitrates for more complex segments of the video stream and vice versa. VBR means the video stream data will be arriving at the input of the system at a variable rate. In practice the input could be more bursty in nature due to the variability in the transmission medium (e.g. the Internet). However, it is possible to set a maximum allowed bit rate when using VBR to offer a minimum inter-arrival time for packets [62]. Video traffic models which takes into account the non-deterministic behaviour of the transmission mediums and content delivery networks can be found in [63]; however an in-depth analysis is out of scope of this research.

- **GoP-structure:** Video encoders attempt to adapt the GoP structure and length to improve the coding efficiency based on the video sequence motion and scene change [44, 45]. B-frames are used to improve compression by taking advantage of temporal redundancy,

hence for static or slow-motion video segments the number of B-frames in a GoP would be higher than fast-motion videos [46]. The dynamic nature of GoP dependency patterns are further increased from the H.264/AVC standard onwards due to the inclusion of *hierarchical B-frame* structures and *multiple reference frames* [42].

- **Reference frame data:** A macroblock or HEVC CU can be inter or intra coded [46, 47]. Hence, depending on the size and number of inter-coded macroblocks in a frame, the amount of reference data required by an inter-predicted frame can vary. Furthermore, the volume of reference data passed between two frames also can vary on the number of reference frames used to encode an inter-frame.

Buffering is a common technique in video streaming applications to overcome issues with timeliness and throughput. When buffering is used, the user waits until the video frame buffer at the decoder is filled up before starting playback. Hence, the *buffering delay* negatively affects QoE; furthermore, videos freezing/stalling mid-way during playback can impact the QoE factor further [64, 65]. Furthermore, buffering incurs additional memory overhead at the video decoder and in the case of live video streaming, buffering in the middle of playback is not acceptable.

2.1.5.1 Workload models

The highly dynamic characteristics of encoded video makes designing resource management and prediction techniques very challenging. Furthermore, trying to design a realistic and tractable model of video decoding workloads also becomes difficult. Abstract *workload models*, that have accurate stochastic properties to real video streams are necessary, to reliably evaluate different resource management protocols. These workload models facilitates synthetic generation of a large amount of different kinds of video streams, without the need for collecting real traces for a study.

An abstract workload model of a task-level parallel H.264 decoder is introduced in the MCSL benchmark framework [66]. They specify the decoder functional unit execution cost and inter-task communication traffic as actual traces obtained from recorded real video streams as well as data derived from statistical properties of trace-data. However, their workload properties such as execution cost, release patterns, inter-task traffic volumes assume a Gaussian distribution, which may not be accurate with the real underlying distribution. Similar task-level partitioned, video decoder benchmarks are used in several many-core design space exploration works (e.g. [32, 38, 67]); most of which use profiled execution-trace data. However, only a few works consider synthetic workload generation for *data-parallel* video decoding.

The sizes of frames and their decoding times vary, depending on the temporal and spatial correlation in the video. Encoded frame sizes have been assumed to follow a Gamma, Lognormal or Weibull distribution [68]. Tanwir et al. [63] in their survey paper, show that wavelet-based frame-size models offer a reasonable compromise between complexity and accuracy but the model prediction results varied significantly based on the type of encoding. In early research, frame decoding time was assumed to have a linear relationship with the frame size [69]. However, Isovic et al. [60] show that there is a large variance in the decoding time for the same frame size. High variability in decoding times are seen for I/P/B frame-types due to different coding tools and memory access patterns in each type [11]. Therefore, classification based on frame type need to be addressed in the model in order to obtain an accurate representation of video

Table 2.2: Relationship between macroblocks and 8 blocks types (taken from [70])

Block type	M_i	**I**	**P**	**B**	**Bi**	Weight(w_i)
IDCT only	M1	x	x	x	x	8.0e-7
IDCT + Fw motion	M2		x			1.1e-6
Fw motion only	M3		x			4.0e-7
IDCT + Bw motion	M4			x		8.0e-7
Bw motion only	M5			x		3.0e-7
IDCT + Bi motion	M6				x	1.4e-6
Bi motion only	M7				x	5.0e-7
No IDCT, No motion	M8		x	x	x	3.0e-7
Run length decoding	M9	x	x	x	x	1.0e-7
Constant (w_0)						-0.1297e-3

decoding workloads. Tan et al. [70] improve the execution time prediction accuracy by considering other variables within the video stream apart from frame-size and type. They observe that in MPEG-2 decoding, different types of blocks within a macroblock perform different decoding operations related to IDCT and motion compensation. Their observations show 9 different block types (Table 2.2) that can be used to derive the frame decoding cost as per Eq. (2.1). The number of different types of blocks in a frame (M_i in Eq. (2.1)), can be obtained by parsing the MPEG header. The w_i term denotes the weight of a type i block and w_0 is the constant term in their regression model. These weights are essentially coefficients of their regression analysis, and are fixed for a given decoder and platform; the weights have been derived by analysing real video streams. Their evaluations were based on a 2.6GHz Intel Pentium IV platform and using a range of video stream types (e.g. low/high motion, CGI/animation).

$$\text{frame decode time} = w_0 + \sum_{i=1}^{9} w_i \times M_i \tag{2.1}$$

Seitner et al. [71] analyse H.264 video bitstream metrics such as different macroblock-level characteristics (e.g size, type, motion vectors) to estimate the frame/macroblock decoding time and resource utilisation at runtime. However, their analysis does not break down the amount of time spent on the computation/memory subsystems individually, and results are dependent on the optimised decoder and hardware used. A similar approach is made in [72] to estimate the execution time for task-level partitions within 3% normalised error margin, derived using linear regression; however their method incurs stream instrumentation overhead. Much of the existing literature on video workload models do not consider accurately modelling the GoP structures (i.e. dependency patterns). Probabilistic approaches such as markov-chain models have been used to generate varying GoP structures taking into account inter-frame dependency probabilities [61, 73]. However, these techniques do not consider complex coding tools such as multiple reference frames and hierarchical B-frames.

2.2 Many-core platforms

With the increasing demand of high-performance multimedia applications, many-core processor architectures are emerging as an attractive platform to fulfil their high throughput and cost-effective requirements. The advances in microchip technologies and nanotechnology has allowed the integration of multiple processing elements (PEs) onto a single chip, thus enabling the creation of MPSoCs. Homogeneous MPSoC systems have PEs with similar architecture and

performance characteristics such as the Intel Xeon Phi co-processors [74] or the Tilera many-cores [4]. Heterogeneous MPSoCs however could contain PEs with vastly different features and architectures such as the Xilinx Zynq UltraScale+ MPSoC [75], which has a quad-core 64-bit ARM application processor, a dual-core ARM real-time processor, an ARM graphics processing unit and field-programmable gate array (FPGA) logic.

Traditionally, different intellectual property (IP) cores on the chip were inter-connected via point-to-point wires, crossbar, bus or hierarchical bus-based architectures (e.g. [76]). However, as the number of IP cores and their inter-connections on the chip increased, communication quickly became a bottleneck [77]. In addition, power consumption became a limitation in bus based architectures, due to the high capacitive load (and in turn higher energy) as the system size increased [78]. This section of the literature survey will primarily introduce many-core communication interconnects and will outline some challenges related to achieving scalable performance in many-cores.

2.2.1 Network-on-chip interconnects

Networks-on-Chips (NoCs) have been proposed as a more scalable, communication centric approach to address the increasingly complicated communication requirements of modern multicores and future many-cores [8]. NoCs have several clear advantages over the bus architectures as discussed in [77]. Due to shorter wires, less power is consumed and performance does not degrade when scaling. Less contention can be seen for larger networks due to distributed routing. Furthermore, NoCs offer flexibility as the same router design can be reinstated, for larger network sizes. It has been shown empirically, using real workloads, that even for standard definition video stream decoding, a NoC architecture can offer higher concurrency and communication bandwidth compared to a popular shared-bus architecture, resulting in lower decoding times (over a factor of 2) and reduced power dissipation [79]. Therefore, NoCs are the communication architecture of choice for emerging multimedia MPSoC applications, with high bandwidth requirements.

As shown in Figure 2.7 NoCs are composed of *links* which are the physical wires, *IP Cores* which are the processing elements such as memory, processors or other dedicated hardware components, *routers* that are responsible for forwarding packets from one core to another and *network interfaces (NI)* which connect the routers and cores together. Each link and router act as the communication infrastructure that facilitates cores to send/receive data. In a 2D mesh NoC, each router has four pairs of input and output (I/O) ports connecting to neighbouring routers (north, east, south and west) and a fifth I/O port pair connecting to the local IP core. NoCs use multihop communication to pass packets from source to destination core along a specified path in the network. A packet is usually segmented into multiple flow control units (flits). For example, in Figure 2.7, if core (0,0) wanted to communicate with core (2,0), the packets would require traverse across 4 links (including local links).

A NoC is mainly defined by its topology and protocol implemented by it. 2D *mesh* topologies are most commonly used, where nodes are connected as a grid and thereby little effort is needed for expansion of the network. The nodes may be tolerant to a few link failures, however due to its irregularity the corner/edge nodes may have less bandwidth. The edge node issue is subsequently addressed in the *torus* topology; however, unlike mesh networks which have bi-directional links, torus links are unidirectional [77]. NoC protocols dictates the mechanism of data transfer within the network such as flow control, switching mechanisms, routing and

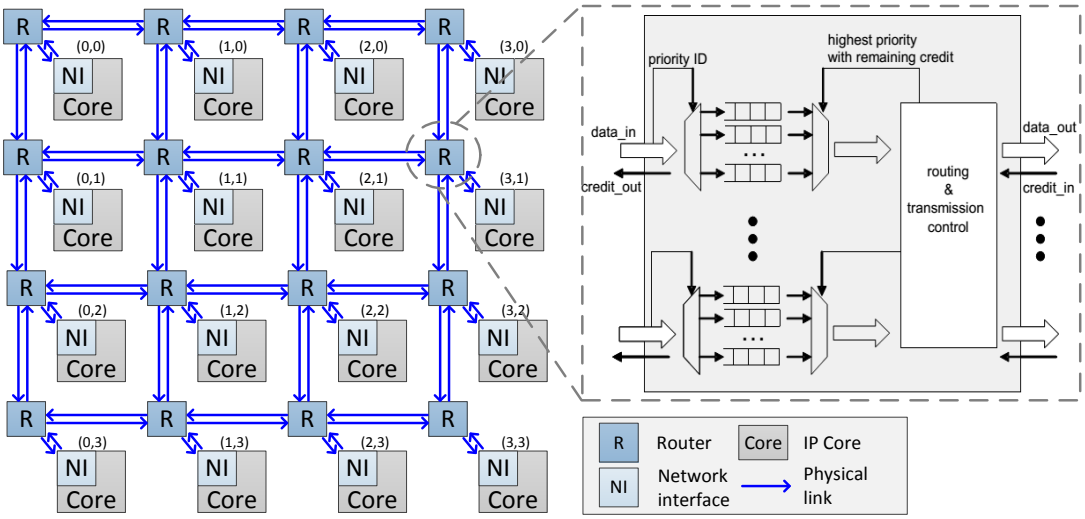


Figure 2.7: High-level architecture of a 2D, 4x4, mesh NoC. Router architecture using virtual channels and priority based arbitration (adapted from [80])

arbitration techniques.

2.2.1.1 Routing, flow control and arbitration

The routing algorithm determines the traversal path of the packets through the network. In deterministic routing such as in *XY-routing*, the packets first travel on x-axis and when the x-coordinate of the destination is reached it continues on the y-axis towards the destination (i.e. rows first, then columns). Deterministic routing is simple to implement, and is commonly used in real-time analysis as it can guarantee that packets always traverse along the same network route between a given source and destination node. NoCs such as Hermes [81] and QNoC [82] employ XY deterministic routing in their NoC routers. In contrast, adaptive routing algorithms determine the path of the packets on a per-hop basis; they are difficult to analyse as the message routes continuously change with respect to the congestion hot-spots in the network [83].

Flow control deals with buffer and channel allocation and is responsible for synchronization between sender and receiver nodes in the network. It is also involved in handling issues of utilizing network resources such as channel bandwidth, buffer capacity and control state efficiently to provide predictable communication services [77, 84]. A switching algorithm as part of the flow control mechanism determines if a flit should be buffered, forwarded or dropped. Unlike in packet switching networks (e.g. Hermes [81], QNoC [82]), circuit switched networks (e.g. AEthereal [85]) reserve a physical path for the data packets before transmission. Packet-switched networks have shown to handle congestion better than circuit switched networks when packet sizes are small [86], although they tend to be energy inefficient due to the complex logic involved. Wormhole switching is a common packet switching technique where a packet is forwarded as soon as the header flit has arrived; thereby having low area overhead (i.e fewer buffers needed), but causes unpredictability and increase in network contention as multiple channels may be occupied [84].

An arbitration strategy is used by the router to select between several simultaneous requests to forward packets. Round robin (RR), first come first serve (FCFS), priority based (PB) and priority based round robin (PBRR) arbitration are some common algorithms used by arbiters to

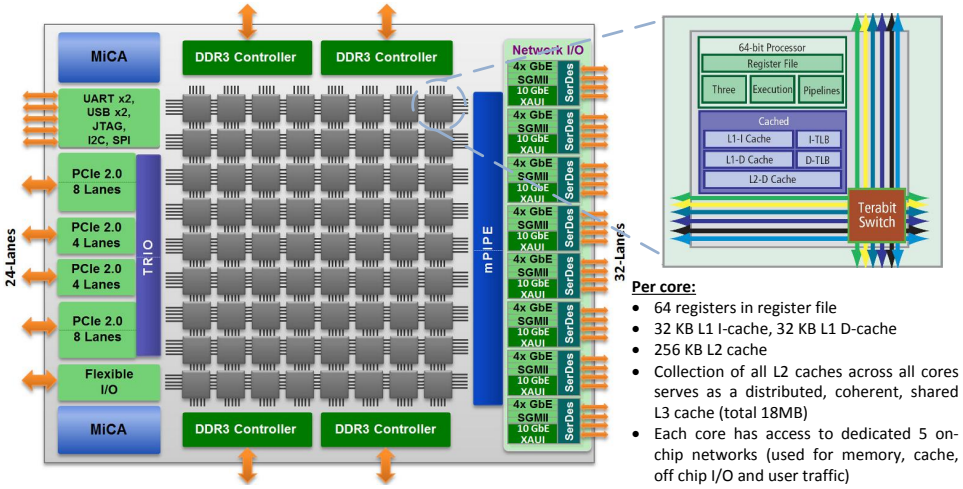


Figure 2.8: 72-core, Tilera Gx72 high level schematic showing memory hierarchy (taken from [4])

select which packet to serve first. Wu et al. [87] presents an improved priority-based arbitration scheme where the contention level of the network is considered when selecting an input channel in the router, thereby helping to relieve congestion hotspots of upstream traffic. Priority-based arbitration schemes are generally used for guaranteed traffic, which is highly attractive for real-time systems where performance bounds need to be known [84].

2.2.2 Predictability in NoCs

For real-time systems that require hard guarantees, the behaviour of tasks and messages sharing different NoC resources such as computation and communication elements need to be predictable. In NoCs such as AEthereal [85], time division multiplexing (TDM) is used to provide guaranteed throughput and latency services with contention-free routing. TDM routers maintain slot-tables to avoid contention, divide bandwidth per link between connections and switch data to the correct output port. TDM routers however scale poorly due to the increase in the slot table size as the NoC size and network load increases.

NoCs such as Hermes [81] and QNoC [82], use a technique called virtual channels (VC), where a physical communication channel is shared logically by separate channels such that multiple packets can use the same path. VCs help to avoid deadlocks, optimise utilisation of wires, improve performance and facilitate message flows with different quality of service (QoS) levels. These advantages however come at the cost of expense of area, power consumption, and production cost of the NoC implementation [77]. VC usage in NoCs is illustrated by Figure 2.7. Each input port has first-in-first-out (FIFO) buffers which stores flits arriving through different virtual channels (i.e. one VC for each priority level). A flit of a given packet will be sent through its respective output port if it has the highest priority among the packets being sent out through that port, and if it has sufficient output port buffer space. Using VCs, associated priority levels and priority-preemptive arbitration the NoC can guarantee throughput to higher priority traffic and it is possible to calculate latency bounds for best-effort traffic [80]. The priority-preemptive arbitration creates a priority ordering which allows the real-time designer to determine when a packet will be blocked. This approach is more efficient than TDM-based NoCs as unnecessary resource reservation is not made. Similar priority-preemptive architectures are often used in real-time multiprocessor research due to its predictability properties (e.g. [80, 88])

2.2.3 Many-core memory challenges

The ever growing gap between memory and processor architecture advances makes it difficult to hide memory latencies as applications became more complex and memory bound. In order to alleviate this issue, modern multicores now employ several levels of memory hierarchy. Registers, scratch-pad memory and caches closer to the processor are small but have faster access times compared to main memory stored off-chip, which are slower but has higher storage space. For example, the Tilera many-core Gx72 chip [4] shown in Figure 2.8 has L1 caches private to each core and distributed shared L2 caches. It also has a dedicated on-chip network, which is used for memory related traffic. In such a NoC based shared memory system, off-chip dynamic random access memory (DRAM) access is translated into a message transaction over the network, and hence incurs round-trip network latencies on every access.

Locality of the data structures on the different memory hierarchy levels need to be taken into account to obtain maximum performance [89]. On-chip cache hierarchies are crucial to achieving fast performance, but in shared-memory architectures providing a consistent view of memory with different caches becomes a problem. Scalable cache coherence protocols have been proposed [90], but bounding the latency and on-chip interference of these coherency techniques on NoC-based systems still remains an open problem. Apart from coherency issues, memory traffic contention in the NoC have also shown to degrade the performance of memory intensive applications, such as video encoding/decoding [49]. Zhuravlev et al. [91] explore cache-aware, thread mapping and scheduling policies to reduce the memory controller and traffic contention; however, they improve average latencies but do not consider worst-case latencies. Memory traffic contention within the NoC rises as the number of cores on the system increases. This poses challenges for real-time multicore systems where upper-bounds for the memory latencies need to be predicted and analysed. A task's worst-case execution time could increase by 300% due to memory access interference even though it only spends 10% of the time fetching from off-chip memory. Nikolic et al. [88] derive upper bounds on memory traffic delays for Tilera-based many-core chips. They assume a limited migration model where tasks can only execute on a subset of the cores and assume static task priority assignment. Architectural efforts have been made in [92] to design a scalable, tree-shaped, TDM interconnect to carry only memory traffic. Using their technique the worst-case latencies of memory transactions to/from the DRAM sub-system can be derived to provide timing predictability.

Due to the limitation on the available number of pins on the many-core chip, the reality of few memory-controllers (MCs) and controller ports for hundreds of cores becomes a concern due to contention at the MCs. Abts et al. [93] show how a suitable MC placement and related routing algorithm combination can improve latency of the workload and bandwidth of the network. Their simulation based experiments show that a diamond based configuration of MCs and a memory-aware routing algorithm can reduce the maximum channel load by 33% and execution time by 56% for uniform random traffic. Placing the data and code of tasks closer to the MCs in order to reduce memory access latencies have also been considered in [94]. They explore a thread-to-MC configuration based on queueing delays at the MC, the communication distance and network load.

2.2.4 Communication models

There are two primary ways data is communicated between parallel tasks in a multiprocessor system - by accessing shared data space and exchanging messages. Tasks in shared memory

platforms communicate by writing and reading from shared global address space common to all processors. Multiple threads can be executed on the different cores of the multicore platform to achieve thread-level parallelism (TLP) where inter-thread communication and synchronisation would occur via the shared-memory. In [95] a speed-up of over 50 is achieved for data-parallel H.264 decoding, using TLP. Many of the parallel video decoding platforms discussed in Section 2.1.4 employ a shared memory communication model. However, shared resource contention related scalability issues (as outlined in Section 2.2.3) eventually decrease performance improvement in shared memory systems.

In the message-passing communication paradigm, parallel processes exchange messages with each other using send/receive data transfer operations. These data transfers could be buffered/unbuffered, blocking/non-blocking, synchronous/asynchronous modes of communication [96]. Unlike in a shared memory model, message passing is only limited by the latency and bandwidth of the on-chip interconnect. Early work by Kranz et al. [97] show that message-passing has a factor of 2-3 improvement over shared memory when transferring large blocks of data between PEs, and is significantly more effective depending on the message sizes and the communication-to-computation cost (CCR) ratio of the workload. The study in [98] show that for NoCs, shared memory approaches do not scale well over 16 processing nodes in a system, but message-passing scale beyond this threshold. However, Casu et al. [99] argue that a hybrid of both paradigms is more cost/area efficient. Their PEs fetch and store private data structures assisted by shared-memory while an on-chip network is available for shared data elements. Message-passing communication has also been used for parallel video encoding/decoding research in the past (e.g: [100, 101]). In [101], GoP-level parallelism is used, while in [100] frame-level parallelism is used and reference picture data is transferred via passing messages in a NoC.

2.2.5 Modelling many-core systems

Early design space exploration is often performed in software based simulations where complexity of the hardware/software design is abstracted. A system could be simulated at different abstraction levels as shown in Figure 2.9; where at each level, the degree of abstraction, cost of modelling and accuracy of the model is different [40]. At the highest level an idea or specification can be turned into an analytical/mathematical model, that does not capture the timing or functional behaviour of the system. In *abstract system level simulations*, fine-grain complexities of the platform and applications are abstracted away. A system is represented as architectural blocks (e.g. processor, interconnect, memory) and applications are considered as tasks and the behaviour is abstracted as simple read/write/execute events. This allows for early design space exploration such as hardware/software partitioning, task allocation, task scheduling etc. [102]. *Discrete event simulations* (DES) are popular models of computing (MoC) used in system level simulations. The DE scheduler processes the system events chronologically according to their associated timestamps. DES has been used in [103] to analyse delays in wormhole based NoCs, with non-uniform number of VCs, using the open source OMNeT++ network simulator. In their simulation they model the router as a collection of connected ports. High-level system simulations have been used by many researchers to quickly evaluate multiprocessor/distributed system resource management techniques [31, 34, 39, 104].

Transaction level modelling (TLM) separates the details of the communication modules from the functional units. TLM models focus only on large-granularity data transfers to speed

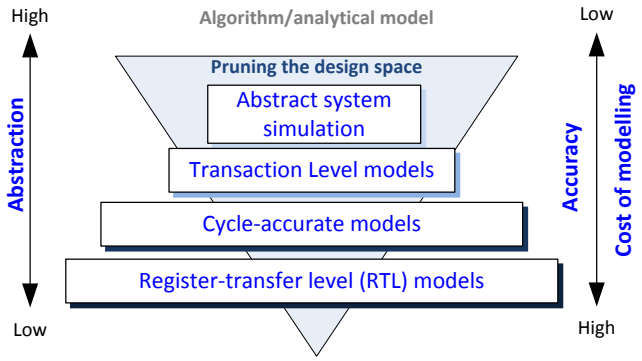


Figure 2.9: Abstraction levels of evaluation platforms (taken from [40])

up simulation time. Hosseinabady et al. [105] proposes a lightweight scheduler that can be used to speed up SystemC-based TLM models to be able to simulate large-scale many-cores. Their approach locally manages the events for each SystemC module in order to reduce the context switching overhead of the SystemC kernel and thereby speed it up by 30%, with no loss of accuracy. Indrusiak et al. [106] present an abstract simulation model of a priority-preemptive NoC, modelled as a single module rather than an array of routers. Their lightweight TLM model takes into account the blocking incurred by different parts of the NoC (i.e. routers, links and PE interfaces), in order to obtain the overall packet transfer latency. Compared to a cycle-accurate model [107], their simulations are 3 orders of magnitude faster but comes at the expense of on average 70% accuracy and overestimate the latency of small packets with long routes.

A *cycle accurate simulator* often extends or complements a functional simulator with timing information and hence can be used to obtain performance estimations for metrics such as latency in number of clock cycles used or power consumption. Popular cycle-accurate NoC simulators include *BookSim* [107] written in C++, *Noxim* [108] and *NIRGAM* [109], both written in SystemC. In Booksim, simulations are restricted to synthetic/trace-driven traffic similar to Noxim. Booksim offers a wider range of topologies and routing algorithms compared to Noxim, however Noxim offer power measurement features not seen in Booksim. Results show only a 5% difference in accuracy in Booksim compared to a register-transfer level (RTL) model but simulation speed and memory usage increase linearly with the modelled network load and number of VCs. Compared to Noxim and Booksim, NIRGAM offers better support for configuring traffic patterns and routers, where users can define their sender and receiver SystemC modules to simulate NoC traffic.

There also exist *full system simulators* (FSS) which model a complete system including the operating system (kernel and user level instructions), the processors and peripherals. For example, the Gem5 [110] full system simulation framework, can simulate execution of real workloads running on multiple PEs interconnected by a NoC. Due to its cycle-accuracy, simulation of complex workloads could lead to minutes or hours. The OVP simulator by Imperas Ltd. [111], is another example of a FSS framework; however, unlike Gem5 [110] it offers instruction/functional level accuracy and no native support for NoCs.

2.3 Resource management in multicore platforms

In multi/many-core systems having a large amount of PEs in the platform brings benefits but also challenges. Tasks that share resources such as PEs, communication interconnects and

shared memory, encounter contention from other tasks. A resource manager (RM) efficiently manages on-chip resources in order to optimise on one or more objectives. When faced with dynamic and unpredictable video streaming workloads, the RM is faced with several *application constraints* (as defined in Section 2.1.5) that needs to be taken into account during the management process. As these application characteristics are unknown at design time, the RM has to make management decisions at *runtime*. Therefore, the runtime management of large-scale complex workloads in a system with many shared resources, becomes an increasingly complex problem to solve.

In the context of this research, the term *resource management* is defined as the process of arriving at the solutions to the following questions:

- **When** do the tasks start execution and in which order?
- **How** do we decide whether to accept/reject new workloads?
- **Which** entity in the system, makes the management decisions?
- **Where** do the tasks get executed?

Likewise, in the context of this research, the primary resource management optimisation variables (objectives) focused are as follows (detailed explanation given in the next chapter - Section 3.2):

- **Predictability:** in terms of optimising the number of schedulable video streams, increasing admission rates and decreasing application lateness.
- **Utilisation:** in terms of decreasing processor idle times and decreasing NoC busy times.
- **Energy-efficiency:** in terms of decreasing NoC communication cost and improving PE busy time distributions.

Resource management can also have several **overheads** which can deteriorate performance. RM overheads therefore need to be taken into account when evaluating and comparing one resource management protocol to another. RM *computation overheads* are primarily related to the processing time required to arrive at a resource management decision. Similarly, RM *communication overheads* are encountered during PE/task monitoring and/or gathering metrics that will be used in the RM decision making process. Section 3.2.4 of the subsequent chapter explains each of the RM overheads in detail.

This section will outline the state-or-the-art resource management techniques and protocols used in the literature to address the concerns above. Before discussing resource management protocols, an introduction to real-time systems and scheduling for multiprocessor platforms needs to be given.

2.3.1 Multiprocessor real-time systems

According to Stankovic et al. [112]: *"Real-time systems (RTS) are those systems in which the correctness of the system depends not only on the logical results of computations, but also on the time at which the results are produced"*. A real-time task has a specified *deadline*, which is the maximum time within which it must finish its execution. If missing a deadline causes catastrophic consequences, the system is defined as *hard* real-time (HRT). On the contrary in *soft* real-time (SRT) systems, producing an output after missing a task's deadline causes system performance degradation, but the system can continue to operate [113].

A real-time task can have several specific characteristics depending on the nature of the application and task model [114]. Release requirements define when the task should become runnable; resource requirements state the hardware/software (e.g. processing elements, dependent data) required per release; scheduling requirements define the timing constraints of the application (e.g. deadlines, periodicity, priority etc.). A task is said to be periodic when it executes on a regular basis and time-triggered; an aperiodic task is when the task is event triggered and hence irregular; thirdly a sporadic task is also an event-triggered task but has a known/specified minimum inter-arrival time. Tasks have a notion of priority, which is a characteristic used by a priority based scheduler to determine which task to execute next. In priority preemptive scheduling, tasks can be preempted by a higher priority task at any time; whereas in non-preemptive scheduling, a running task is executed till completion, it cannot be interrupted. Scheduling provides an algorithm for ordering the use of system resources as well as provides a means for predicting the worst-case behaviour of the system [114]. Many results have been achieved in the area of fixed priority pre-emptive scheduling based on the seminal work of Liu and Layland [115]. Although priority preemptive scheduling generally leads to better schedulability than priority non-preemptive scheduling, preemptive scheduling does have its own drawbacks; such as the need for non-trivial resource access protocols and preemption delays involved in flushing/reloading cache memory [116]. Therefore, there also exists cooperative schemes where tasks can only be preempted at well-defined scheduling points at runtime, thus attempting to strike a balance between fully preemptive and non-preemptive scheduling techniques [117].

Multiprocessor scheduling can be viewed as attempting to solve two problems: the *task allocation* problem and the *task priority assignment* problem [118]. Priority assignment dictates when and in what order tasks and their different invocations should execute. A task can have a single fixed priority for all its invocations, different priorities for each invocation (e.g. earliest deadline first - EDF) or different priorities at different time instants dynamically (e.g. least laxity first - LLF). Multiprocessor scheduling is a much more difficult problem than uniprocessor scheduling, because a task can only use a single processor at a time even when several are free [118]. Hence, optimal uniprocessor priority assignment schemes such as rate monotonic priority ordering (RMPO) [115] or Audsleys' priority assignment [119] algorithms, are not valid. Davis and Burns [118], highlight several challenges in HRT multiprocessor scheduling related to task migration, processor utilisation and resource sharing policies. In their survey they purely consider tasks that do not communicate with one another (i.e. independent tasks).

Static scheduling requires knowledge of real-time task characteristics a priori and may use resources inefficiently; therefore, they are inflexible to failures and overload situation. Thus, the need for dynamic scheduling techniques such as, value-based scheduling [120], proportionate fairness (Pfair) scheduling [121] and earliest deadline zero laxity (EDZL) scheduling exits. Pfair algorithms are considered optimal for periodic task-sets with implicit deadlines (i.e. deadline = period), however Fisher [122], proves that there is no optimal runtime multiprocessor scheduling algorithm for sporadic task-sets with constrained or arbitrary deadlines.

Task allocations can either support migration *(global)*, where tasks and its invocations can be executed on different processors; or no migration (*partitioned*), where each task is allocated to a processor and no migrations are allowed. In the partitioned approach, each processor has its own run-queue, and a task can only interfere other low priority tasks in the same run-queue. In global scheduling, a single priority ordered queue for all processors is maintained, and tasks

and their invocations are allowed to migrate between processors. There also exists hybrid approaches such as clustered/hierarchical scheduling, where multiple processors are combined into a virtual cluster and task migration is only allowed within the cluster [123].

2.3.1.1 Predictability vs. resource efficiency

A task is considered *schedulable* if it completes its execution before its deadline. A wide range of design-time schedulability tests for different real-time tasks models exist, which can be evaluated to guarantee that a task will not miss its deadline [113, 118]. These tests often have conservative assumptions of the task's worst-case timing behaviours. They try to account for any non-deterministic behaviour of the system in the analysis, with less pessimism as possible. The tests also tend to consider the worst-case execution time (WCET) of tasks when performing the schedulability tests to provide *safe* timing guarantees. However, as illustrated in Figure 2.10, the WCET upper bound may be rarely observed in practice. If the analysis has a high degree of pessimism or is overly conservative, tasks will definitely meet their deadlines but system resources will be over-provisioned and/or under-utilised. Conversely, if the analysis is optimistic, certain timing violations may be made, but platform resources (e.g. PEs, NoC) maybe better utilised.

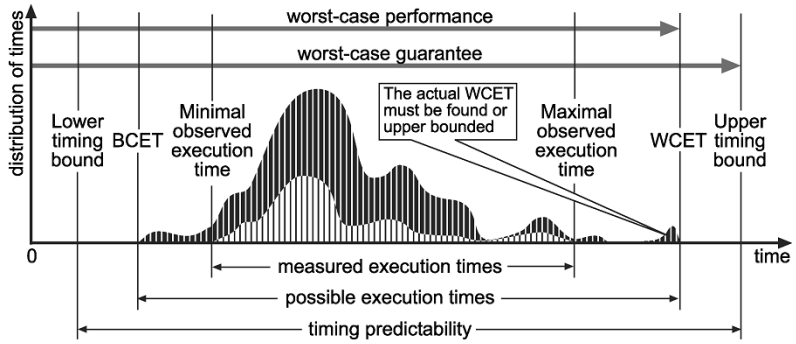


Figure 2.10: Basic notions concerning timing analysis of systems (taken from [124])

SRT systems are often characterized by highly dynamic behaviour and hence offering predictability guarantees while efficiently utilising system resources is non trivial. Therefore, predictability objectives of a SRT system is to minimise the mean and maximum *lateness/tardiness* (i.e. completion time - absolute deadline) of the tasks [125]. Unlike HRT tasks, SRT task properties such as periodicity, deadlines, inter-arrival patterns, computation times etc. are difficult to ascertain offline. Hence, techniques such as reclaiming unused resources, overload management, system adaptability via feedback-based scheduling can be employed to balance predictability and resource utilisation [125].

2.3.1.2 Response time analysis

The *response time* of a task is defined as the elapsed time between the dispatch time and the time it completes execution. The response time includes the duration it was blocked by another task and any time period it was waiting for data. Response-time analysis (RTA) is a standard, sufficient and necessary schedulability test for fixed priority-preemptive systems, where the worst-case timing of a task is analysed [114]. In RTA, the blocking interference caused by

higher priority tasks on lower priority tasks are taken into account when calculating a task's worst-case response time (WCRT).

Audsley et al. [126], present a recurrence relationship as shown in Eq. (2.2), to find the WCRT (denoted r_i) of a fixed priority preemptive task τ_i. In Eq. (2.2), c_i denotes the WCET, t_i denoted the period of τ_i. The set $hp(\tau_i)$ denotes the tasks that have a higher priority and in the same task queue as task τ_i. A task is deemed schedulable if $r_i \leq d_i$, where d_i is the absolute deadline of τ_i with respect to its release time. To derive the WCRT, the analysis assumes, each task experiences the maximum blocking possible by higher priority tasks. This response time analysis has also been adapted to suit symmetric multiprocessor systems using global fixed priority and EDF scheduling [127].

$$r_i^{n+1} = c_i + \sum_{\forall \tau_j \in hp(\tau_i)} \left\lceil \frac{r_i^n}{t_j} \right\rceil c_j \tag{2.2}$$

Similarly, on a many-core NoC interconnect, contention occurs when several message flows try to access the same network resource at the same time. In the case where priority based preemptive arbitration of the routers, the NoC architecture is able to provide guaranteed throughput to high priority message flows. Hence, a message flow (denoted Msg_i) will have at most two interference sources - *direct* and *indirect* interference flows. Direct-interferers (denoted S_{id}) are higher priority traffic-flows that have at least one physical link in common with the observed traffic-flow. Indirect-interferers (denoted S_{ii}) are higher-priority flows that do not share any links with the observed traffic-flow but share at least one link with a traffic-flow in S_{id}. For example in the flow contention scenario in Figure 2.11, Msg_j directly interferes Msg_i if $P_j > P_i$, where P_i denotes flow priority. However, Msg_k indirectly interferes Msg_i, if $P_k > P_j > P_i$.

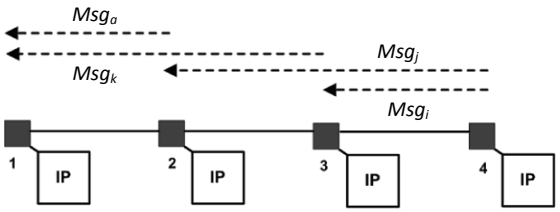


Figure 2.11: Direct and indirect interference on NoC message flows (taken from [128])

Shi et al. [128] introduces an analytical approach to derive an upper bound for the worst-case network latency of a traffic flow in wormhole switching, fixed priority preemptive NoCs, as shown in Eq. 2.3. In this equation J_i^R is the release-jitter and J_i^I is the interference-jitter. The basic latency C_i of a message flow, calculated using Eq. (2.4), is the time taken to transfer from source to destination under the assumption of no contention over the NoC links. In Eq. (2.4), $numHops$ is the number of hops between source and destination, RL denotes the time needed for a flit to traverse a link, $numFlits$ is the payload size in number of flits and HL denotes the time needed to route a packet header.

$$R_i^{n+1} = C_i + \sum_{\forall Msg_j \in S_{id}} \left\lceil \frac{R_i^n + J_j^R + J_j^I}{T_j} \right\rceil C_j \tag{2.3}$$

$$C_i = (HL \times numHops) + (RL \times (numHops - 1)) + (HL \times numFlits) \tag{2.4}$$

The work in [80] extends the priority-preemptive NoC RTA analysis in [128] to include both

communication flows and computation tasks in order to calculate the end-to-end worst case response time (E2ERTA) of task chains. In their synchronous pipeline model, parallel execution of multiple invocations of a task chain is allowed to execute simultaneously over different PEs but disallows the simultaneous execution of more than one invocation of the same task. Different invocations of the task chain runs concurrently in a phase-shifted way. In [80], the response time r_i of the task τ_i that releases the message-flow is considered to be the release jitter of Msg_i, hence $J_i^R = r_i$, as shown in Eq. 2.5. This is assuming the message flow is released immediately after the execution of its source task hence, Eq. 2.5 can be used to calculate the end-to-end response-time of tasks. This analysis becomes useful to determine if a task-set with a given task to PE mapping is schedulable on a fixed priority preemptive system. Both [128] and [80] assume the tasks and flows of an application will be released at simultaneously and do not take into account the precedence constraints of tasks/flows in an application. Dependent tasks in a non-pipelined task chain would not ever interfere with each other and would be released at different points in time after their parent task (i.e. predecessors) have finished their execution. Hence, the analysis in [80, 128] will result in over-estimating the end-to-end WCRT, for a non-pipelined application scenario. Kashif et al. [129] integrates offsets and jitters in to the response time analysis to take into account the precedence relationships in directed acyclic graphs (DAGs) based applications running on priority-preemptive NoCs. However, they still assume all applications in the system start at the same time instant with zero jitter and assume infinite buffers in the NoC.

$$R_i^{n+1} = C_i + \sum_{\forall Msg_j \in S_{id}} \left\lceil \frac{R_i^n + r_j + J_j^I}{T_j} \right\rceil C_j \tag{2.5}$$

2.3.1.3 Predictable online admission control

Admission control and resource reservation have traditionally been used to achieve predictable performance of real-time services. In HRT systems, where application real-time properties are known, sufficient resources are reserved a priori and offline schedulability analysis is performed to ensure all timing constraints can be met [118]. On the other hand, in dynamic real-time systems where the workload patterns are not known at design time, such as in live video decoding (Section 2.1.5), predictability is achieved through online admission control (e.g. [130]). Admission control (AC) is one of the first steps in managing resources in a system. The role of a *predictable* AC is to assert if a new task/task-set can be executed in the system without missing its deadline and not forcing existing tasks to miss theirs. Making a fast AC decision at runtime, taking into account existing resource usage is non-trivial, specially when the application workload is highly dynamic and not known a priori.

Deterministic admission controllers, such as in [131–133], provides a guaranteed and predictable service by taking into account worst-case timing behaviour of the application and platform. However, if the assumptions of the workload execution budgets and/or the assumptions made in the worst-case timing analysis are highly pessimistic, the admission control algorithms will cause severe under-utilisation of system resources. Deterministic admission controllers using online schedulability tests for fixed-priority real-time systems can be categorised into utilisation-based tests and RTA-based tests. Utilisation-based tests can be used to efficiently decide if the system has sufficient capacity to accommodate new tasks [132]. However, they are more suited for uniprocessor systems, often assume an independent task-set model and do

not capture the blocking behaviour of tasks. Dziurzanski et al. [134] uses reduced complexity RTA-based online schedulability tests which are more powerful but have higher overhead than utilisation based tests.

In deterministic ACs the WCRT of tasks and their deadlines are critical requirements for evaluating the admission decision. In certain distributed real-time systems only the end-to-end deadline of a chain of tasks may be known and individual task deadlines may need to be estimated at runtime [135]. Di Natale et al. [136] slices the overall end-to-end deadline amongst the subtasks in order to maximize the minimum slack of tasks. Kao and Garcia-Molina [137] divides the total remaining slack among the subtasks in proportion to their estimated execution times. Eq. 2.6 shows the calculation of the absolute deadline using their proposed equal flexibility scheme (EQF) subtask deadline assignment scheme; here, x_i represents the task execution time, m represents the total number of tasks in the task-set, a_i is task arrival time and D_{e2e} denotes the task-set end-to-end deadline. A similar approach is taken in [138], where they calculate the subtask deadlines of critical path and non-critical path tasks of a task graph differently. For these deadline assignment schemes to be effective, the task's execution time need to be known or accurately estimated.

$$d_i = a_i + c_i + \left\{ \left[D_{e2e} - a_i - \sum_{j=1}^{m} c_j \right] \times \left[\frac{x_i}{\sum_{j=i}^{m} c_i} \right] \right\} \tag{2.6}$$

Statistical admission control for SRT/best-effort video streaming services have been explored in the past [130, 139]. These work mainly attempt to probabilistically predict the platform load in terms of disk access times and bandwidth measurements for VBR video workloads. These algorithms do not take into account deadlines but attempt to maximise resource usage. There also exists probabilistic admission control approaches which rely on task execution time distributions and quality parameters, to reserve resources at runtime [140]. Ditze et al. [141] propose a method for real-time scheduling and admission control of multiple MPEG video streams which uses continuous re-processing by the admission controller and scheduling the different workloads out-of-phase to each other such that the decoding tasks do not interfere. In case the allocated resources vary over time, their method re-invokes the AC to adjust the resource reservation. This method guarantees QoS, however their analysis is limited to a few concurrent video streams and a uniprocessor system. Online predictability guarantees for video streaming, can also be provided via the video transmission communication network [142] or through operating system support [143] but is out of the scope of this thesis work.

Open-loop scheduling and admission control algorithms such as those discussed above, suffer from the inability to adapt to unforeseen varying system states. They also assume worst-case application characteristics leading to resource under-utilisation [144]. Due to these reasons, feedback control real-time admission and scheduling algorithms has been proposed to adapt the system to dynamic and unpredictable workloads [144, 145]. A feedback control real-time scheduling (FCS) architecture consists of a feedback loop which continuously monitors control variables from the system [145] to adapt the performance based on the required QoS level. For example in adaptive video processing systems, system load, decoding execution time and power usage is monitored to balance image quality and energy consumption [146] or to decide on when to skip frames [147]. Feedback-based admission control strategies have also been recently employed in many-core HRT systems to predict the task-set schedulability, by using the

task slack as a control variable [134]. However, the overhead of the feedback loop in FCS systems increases as the system size and workload increases. Also, higher monitoring rates can improve the accuracy of the system state but also increases the monitoring overhead [37].

2.3.1.4 Video-centric task scheduling

Mastronarde et al. [148], show that a classical EDF scheduling approach does not take into account video quality of the decoded video stream; hence, under heavy load it may induce severe distortion to the video. They introduce a packet-level, priority assignment scheme that takes into account a video distortion reduction metric, which is derived via offline stream analysis. Isovic et al. [147] and Blanch et al. [149] use video-specific metrics such as frame-type, frame-position in the GOP, frame-size and buffer size in order to determine a frame-level priority assignment. They use the priorities to determine which frames to skip decoding in order to meet end-to-end GoP deadlines and to reduce the video quality degradation. Priorities have also been assigned at a coarse-grain, stream-level [18, 150]. In [150], stream priorities are dynamically assigned based on the number of missed video frame deadlines within a specified window of consecutive frames. They use this scheme to maintain the quality of each video stream above a specified tolerable quality level. In [18], a vehicular multi-camera safety system case-study is presented. Higher priorities are assigned to the camera in the direction where a vehicle is approaching. Using these priorities, a runtime resource manager can steal processing resources (i.e. reduce the frame rate) from low-priority cameras.

Moreira et al. [34] investigates real-time scheduling of dataflow programs such as graphics scaling applications, modelled as synchronous dataflow (SDF). They use a time-division multiplexing (TDM) scheduler and a static allocation technique to provide hard timing guarantees. Similarly, Bamakhrama et al. [151] consider using an extension of dataflow graphs to model streaming applications. Their analytical framework, allows the computation of the minimum number of processors needed by different HRT scheduling algorithms, for periodic tasks with varying data dependencies, to achieve maximum throughput and meet all deadlines. In [152], the authors combine a largest task first heuristic with dynamic power management based scheduling scheme to decode SRT H.264 bitstreams with slice-level parallelism. They assume a GoP structure without B-frames that have equally spaced soft deadlines, based on the frame rate. Their task model however, assumes the execution cost of the slice decoding times can be accurately predicted. Energy savings are obtained through setting the speed of the processing cores as low as possible, without missing any deadlines, and shutting off power to unused cores. In [61], the sequence of frames in a GoP is modelled as a deterministic Markov chain. The authors present a two-level scheduler which firstly selects the scheduling and processor frequency per frame and secondly maps frames to processors and set their clock frequencies. The scheduler considers the frame buffer state, dependencies between parent and child frames, number of remaining slices in a frame, power consumed by each processor in each state. They assume an exponentially distributed slice decoding cost and assume frame deadlines and GoP structures are known at design time.

2.3.2 Resource management organisation

Resource management (RM) of on-chip resources such as communication links/bandwidth, computation resources and memory, in a NoC-based many-core platform is a challenging task.

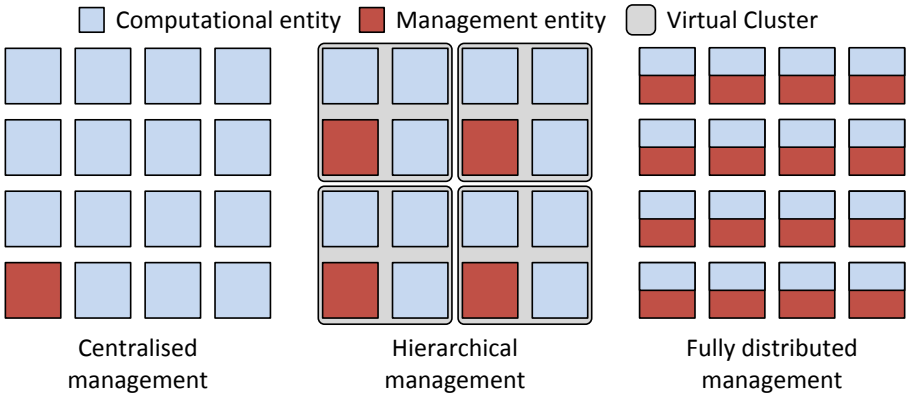


Figure 2.12: Illustration of resource management organisation types

Resources need to be allocated to tasks and message flows at efficiently at runtime and the system must adapt to changing workload conditions. The types of resource management can be broadly classified into three categories: *centralised*, *hierarchical* or *fully distributed*. Figure 2.12 shows the high level differentiation between these classifications, based on which entity in the platform performs the RM duties.

2.3.2.1 Centralised management

Centralised managers have global knowledge of the platform, which allows for a globally coordinated and efficient distribution of resources to applications. For example, in early work by Nollet et al. [153], slave PEs continuously send state information to a single master PE via the NoC. The slaves purely execute user application code and the RM performs runtime management functions. The type of information monitored and the overheads of monitoring depends largely on the optimisation objective, timing and nature of the workload. PE utilisation is another simple but effective metric obtained at runtime from the slave PEs, in order to facilitate task allocation decisions [39]. In some work, the communication between the slave PEs and the central management PE is more complex, such as in [32] where a four-way message request/reply communication protocol is used to decide on a task allocation. In [154], the RM stores and updates several platform resource metrics such as channel occupancy, PE resource usage, local memory usage, which are monitored with the help of resource monitors connected to the NoC ports and PEs. As seen in [155], dedicated control networks can be used for the communication between the RM and the PEs, hence the control messages would not interfere with data traffic; however, an additional network would incur area and energy overheads. In some work, the slave PEs regularly send certain application-specific information to the RM without being requested. For example, Khan et al. [146] feed back the encoded HEVC video frame quality and thread execution times to the RM at runtime, in order to maintain minimal quality degradation and minimise power dissipation. In [156], the RM *periodically* monitors a wide range of metrics such as PE/memory/communication channel usage, network congestion and latency values, to adaptively vary the video frame-rate with respect to (w.r.t) to meeting the deadlines of a video decoding application. The *monitoring frequency* has to be tuned in order to maintain a balance between achieving an accurate view of the system and to keep the *monitoring overhead* low [156].

Centralised management schemes are acceptable for small platforms such as 4-16 core NoCs,

however as the platform size and workload increases, centralised RM shows performance degradation [30]. It suffers from a single point of failure, large monitoring traffic overhead, central communication hotspot around the manager and scalability issues [30]. Due to these reasons, *fully distributed* and *hierarchical* management strategies have been proposed in the literature as scalable and reliable resource management techniques for large-scale many-cores.

2.3.2.2 Hierarchical management

The literature provides several hierarchically distributed RM protocols [35, 36, 146, 157–159]. In these hierarchical management techniques, the many-core system is divided into virtual regions called *clusters*, with each cluster having a cluster manager (CM) which allocates tasks to PEs within its own cluster. Such a scalable, hierarchical management technique is shown to significantly reduce resource management communication and computation overhead, for platform sizes larger than 12×12 [36]. In [36], the cluster managers (agents) are not fixed and can vary, depending on the number of applications. When an application starts execution, its initial node is selected at random. It then tries to randomly allocate regions (i.e. PE resources) on the network to service the application. CMs communicate with CMs in other regions to borrow resources. Long communication routes between distant control agents and applications can be an issue, as it can lead to higher NoC traffic congestion and therefore performance degradation. The distributed management approach by Anagnostopoulos et al. [159], has a lower communication overhead than [36], but solely targets malleable applications (flexible applications, which can be stopped and assigned different number of processors at runtime). In their work each cluster can have one or more 'manager cores' responsible for application workload distribution and a 'controller core' is responsible for assigning applications to managers. A hierarchical parallelisation approach for multi-level H.264 video encoding is given in [160]. A cluster platform is partitioned into groups of computation nodes, each encoding independent GoPs. Within each group, slice-level parallelism is employed. Each processor group has a local manager which requests for new GoPs from a global manager.

Castilhos et al. [157], performs continuous self optimisation by enabling the cluster managers (or agents) to communicate with other cluster agents in the surrounding area. They attempt to reduce processor energy consumption , balance workload distribution and reduce execution time of video decoding and generic synthetic applications. In [157], when the global manager (GMP) receives a new application request it decides which cluster to map the initial tasks of the application. The slave nodes within a cluster (SP) requests a new task from the local cluster manager (LMP) and the LMP in turn makes a request to the global manager (GMP) (as shwon in Figure 2.13(a)). The LMPs periodically monitors the load of each slave node (SP) in its cluster. If there are no resources available in the current cluster, the LMP sends a 'loan request' to other neighbouring LMPs (Figure 2.13(b) - step 1). The LMPs that receive the loan request searches for an available resource in their cluster and if found, reserves the resource and sends a reply back to the requesting LMP (step 2). The requesting LMP decides which resource to select and sends a 'release' message to the other LMPs that were not selected (step -3). Lastly, in step 4 the GMP is notified of the new task mapping. In the authors subsequent work [37], they discuss how a trade-off between accuracy and intra-cluster monitoring overhead can be achieved by tuning the monitoring period.

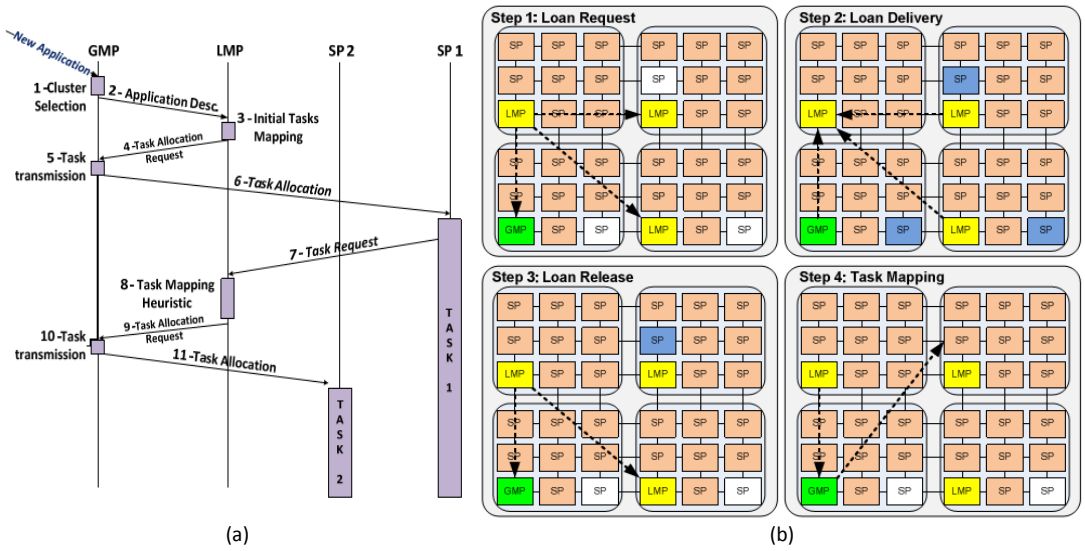


Figure 2.13: Cluster-based resource management protocol : (a) Communication between slave nodes (SP), cluster managers (LMP) and global manager (GMP); (b) Request/reply protocol between neighbour LMPs (taken from [157])

Hierarchical resource management protocols are more scalable than centralised management but they come with their own drawbacks. The inter-cluster communication and intra-cluster monitoring within large sized clusters can lead to high levels of monitoring and control traffic within a NoC. Using smaller clusters intuitively leads to a higher number of cluster-managers which limits the compute-resources in the platform. Furthermore, the system's reliability is still bounded by the number of cluster-managers within a system. Due to these reasons, there exists a body of research focusing on *fully distributed* resource management techniques.

2.3.2.3 Fully decentralised management

Fully distributed resource management techniques are scarce in the many-core resource management literature. Unlike the previously discussed approaches, they employ runtime self-management and resource allocation/re-allocation mechanisms in each node, without the need of one or more management entities. In such systems, the nodes adapt and optimise itself to changing workloads and internal conditions and recover from faults. 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 tasks to miss their deadlines.

In [161], each processing core in the system can migrate individual tasks to neighbouring ones based on the local workload, task sizes, communication requirements of the tasks and network traffic. Results showed that compared to offline RM approaches, their proposed technique provided 25-30% mean different mapping results. A decentralised NoC monitoring infrastructure [162] and fully decentralised core allocation scheme [163] is introduced for *invasive applications* running on NoC-based multicores. Invasive applications are those that can dynamically explore and automatically request/release a certain amount of processing resources by itself during its execution in order to increase its speed-up. Once additional resources have been claimed, the application code is copied and executed on the cores in parallel. Applications

give up resources for other applications only if the gain is substantial over the loss. However, additional hardware in terms of NoC and PE monitoring infrastructure is needed to perform these invasive control functions [162]. Their evaluation results show an additional 2% area overhead and 5% power consumption required for a performance improvement of 6% and communication energy reduction of 27%. In fully distributed schemes, since each node performs control operations the overhead of the software/hardware resource management policies need to be lightweight in order to not interfere with the computation.

2.3.2.4 Bio-inspired autonomous management

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 inspired by biology. Brinkschulte et al. [164] present a distributed, artificial hormone system inspired, middleware framework where periodically each core calculates and communicates its task suitability with other cores in the network, to determine which core to execute a task. However, in their further work [165], it is shown that even though their fully decentralised management technique displays better failure handling, the communication overhead is higher compared to a centralised or hierarchical approach. Bio-inspired, distributed load-balancing reconfigurable hardware and software architectures have been explored by Mudry et al. [166]. Processing nodes are able to monitor their workload and during overload situations, tasks locate unused nodes in the system and replicates its code onto it; similar to cellular division and differentiation in biology. Their algorithm allows a video decoder stream processing pipeline to grow and balance load between its computational nodes automatically using local decisions. Complex policies are required to overcome network flooding due to control message overheads and pipeline overgrowth which would decrease efficiency in the system. An adaptive, immune-inspired NoC architecture is presented in [167], to achieve lower data flow latency and buffer utilisation. The NoC switching, routing, flow control and resources such as buffers, bandwidth and link utilisation can adapt based on the application characteristics. Their architecture requires monitoring and supervising mechanisms at each layer in the NoC.

Bio-inspired algorithms have been evaluated on more mature domains such as wireless sensor networks. For example, Caliskanelli et al. [104] present a distributed load-balancing technique inspired by social insects such as bees. The authors devise a technique which is used to solve the trade-off between service availability and energy consumption. In their technique a node in a network can either be a queen bee (denoted QN) or a worker bee; the QN performs computation and stimulates a pheromone which is propagated throughout the network. All nodes accumulate and pass on pheromones received from the queens, and the dose of the pheromone is decreased at every hop-distance away from the QN. The pheromone level (denoted h_i) for each node decays over time. A node becomes a QN when its h_i drops below a certain threshold. 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 shown in Figure 2.14, the algorithm contains two periodic and one asynchronous event cycles, that are carried out by each node in the network. The differentiation cycle occurs to distinguish a nodes queen status and during the decay cycle the node's hormone level is decremented. The asynchronous, propagation event occurs when a new hormone dose is received by a neighbouring node.

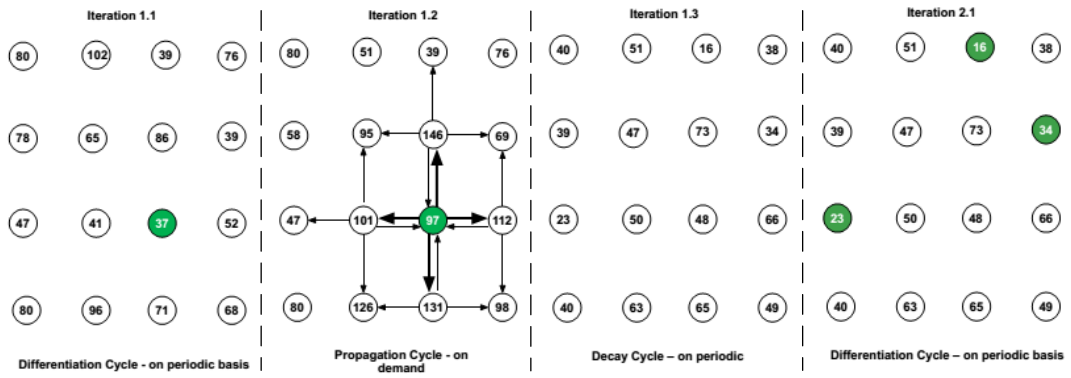


Figure 2.14: Illustration of the pheromone-signalling algorithm: Differentiation and decay cycles and the propagation event. Green nodes indicate queen nodes and the values represent the hormone level (taken from [104])

2.3.2.5 Closed-loop vs. open-loop resource management

Closed/open-loop based classification is orthogonal to the organisation of resource management. In this work, a *closed-loop* RM is referred to as one which has monitoring information sent back to the RM by the system (i.e. either by the PEs/interconnect). Much of the centralised management schemes identified in Section 2.3.2.1, fall under closed-loop resource management, as the slave-PEs send back monitoring information or are requested for state information by the centralised RM (e.g. [153] [154] [146] [32]). In hierarchical-management schemes such as in [37] [159] (Section 2.3.2.2), the cluster managers monitor their slave-PEs gathering information such as utilisation. Therefore, many of the existing hierarchical-management schemes also have a closed-loop resource management protocol. It is difficult to categorise fully-distributed resource management as open/closed-loop, as each node performs RM functionality. Closed-loop systems enable the RM to have an up-to-date view of the system in order to make accurate resource management decisions. Unfortunately, closed-loop systems encounters *communication overheads* as the NoC is utilised for the feedback communication between the RM and slaves. In large-scale platforms, the RM communication overhead due to system monitoring feedback can cause network congestion and eventually lead to severe performance bottlenecks and increase interconnect power consumption.

Unlike closed-loop systems, in *open-loop* resource management, the PEs/network do not send back state/monitoring information back to the RM. In open-loop management, the RM may instead keep track of previous management decisions. The tracked information and predicted/worst-case timing properties and characteristics of the workload can be combined to then estimate the current state of the system and thereby arrive at future task allocation and schedules. The lack of feedback monitoring loop removes the RM communication overhead in open-loop resource management systems leading to better scalability but the RM can have inaccurate view of the system. Open-loop many-core resource management has not had much attention in the literature; however, many of the task allocation and management techniques presented in this work focuses on open-loop management.

2.3.3 Dynamic task mapping

This section first explains the notion of many-core task mapping. Next, an overview of design-time, static mapping techniques are given and their unsuitability in managing dynamic workloads are discussed. *Dynamic task mapping* techniques are then introduced and an in-depth review of the state-of-the-art dynamic mapping heuristics is presented.

Multi/many-core task mapping is an important issue which has a rich body of literature and different taxonomies classified in surveys [30, 168, 169]. Applications are assumed to be partitioned and parallelised into tasks or data-level components. Task graphs (TG) and data-flow graphs (DFG) are common forms of analytical tools to represent applications. The nodes in a TG represents computational tasks and edges indicate communication and dependencies between two tasks. Figure 2.15 illustrates this process of application to TG partitioning and mapping onto a many-core system. Task clustering can be performed during mapping, for platforms that support multitasking. Inter-task communication is carried out via the NoC. Finding the best task placement to optimise on one or more performance metrics (e.g. throughput, deadline misses, energy etc.), can be reduced to a *quadratic assignment problem*, which is a well known NP-hard problem [170].

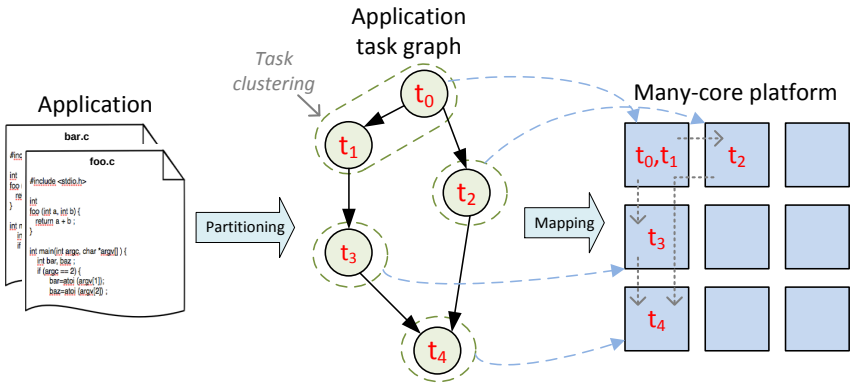


Figure 2.15: Illustration of task to PE mapping

Static task mapping algorithms (also referred to as design-time/offline mapping) are used when the application and platform characteristics are known a priori. Static mapping techniques prune the mapping solution space to find the best placement of tasks on the NoC. The quality of the mappings given by static approaches may be superior to dynamic mapping, which do not have a complete view of the platform and workload. Optimisation techniques such as integer linear programming (ILP) can be used to find an optimal mapping solution for small problem sizes (i.e. small number of tasks and cores) [171]. Search-based techniques such as genetic algorithms (GA) have also been successfully used to find schedulable mapping configurations for hard real-time systems [172]. In [173], Schmitz et al. use a GA to obtain an energy-efficient task mapping configuration on dynamic voltage-scaling enabled PEs, that adheres to specified timing constraints. Whilst their GA fitness function takes into account the power dissipation and communication time of inter-task communication, they do not consider timing characteristics related to network contention. Genetic algorithms can also be easily parallelised, giving the ability to search large problem sizes; however, their solution cannot be guaranteed to be optimal. Branch and bound (B&B) algorithms have also been used to perform static task mapping for NoCs [174]. B&B algorithms first construct a tree structured solution space, which are then topologically searched whilst bounding un-allowable solutions. Deterministic search time

however grows exponentially with the size of the problem [168].

Static mapping techniques cannot be used for workloads with unknown/unpredictable characteristics. Due to the large optimisation overhead of static mapping approaches, they also are not feasible to be used at runtime. The work presented in this thesis considers *dynamic task mapping* to manage the video decoding workload not known at design time, hence a detailed survey of static mapping techniques will not be provided. Interested readers can refer to [30, 168], for a comprehensive review of static mapping approaches.

When the workload is unknown at design time and/or unpredictable, ***dynamic task mapping*** (also referred to as runtime/online/on-the-fly mapping) techniques need to be used. The task allocation needs to adapt to available resources, unforeseeable application and platform changes. Therefore, these greedy algorithms use *heuristics* to determine the task-to-PE mapping at runtime in order to optimise one or more conflicting goals such as energy consumption, execution time, communication latency, reliability and schedulability. As they are executed at runtime, it is mandatory that they are low in complexity and must scale well for larger NoC and task-set sizes, whilst maintaining a good mapping quality.

In this context of this thesis, Table 2.3 summarises the state-of-the-art dynamic resource management (mapping and remapping) techniques according to the following criteria: *1)* interconnection and architecture type; *2)* resource management approach; *3)* was task migration or re-allocation performed?; *4)* key metric in the mapping heuristic; *5)* optimisation goal; *6)* target/evaluated application type; *7)* evaluation platform and *8)* novelty of the technique. The following sub-sections provide a deeper review into the techniques given in Table 2.3, specifically focusing on the novelty of the intelligent heuristics used in the runtime mapping process.

2.3.3.1 Simple bin-packing dynamic mapping heuristics

Several early work in partitioned multiprocessor scheduling considered applying bin-packing heuristics such as first fit (FF), best-fit (BF), next-fit (NF), worst-fit (WF), combined with EDF/RMPO to address the task mapping problem [118]. These heuristics are simple to implement, however they assume independent tasks and the utilisation-based schedulability tests are pessimistic. There exists variations of these packing heuristics such as FF-decreasing (FFD), where tasks are first sorted according to their utilisation/weight and then allocated. In [34], the FFD allocation heuristic is combined with *clustering* of heavily communicating tasks to decrease the number of PEs and communication bandwidth used. First-free node (FFN) and nearest-neighbour (NN) are also two simple heuristics introduced by Carvalho et al. [32]. FFN starts at node (0,0) and simply assigns tasks to the next free node in a column-wise fashion. NN searches for a free/available node in close proximity to the requesting task; the search area is increased spirally with 1-hop increments.

2.3.3.2 Load-balancing dynamic mapping heuristics

Load imbalance between PEs and poor locality of tasks are two major factors which can cause utilisation hot-spots in the system. This could lead to PE failures due to thermal implications, performance degradation and affect latency [31, 37]. Huang et al. [39] use a modified FFD algorithm to perform load balancing whilst ensuring tasks are still schedulable. Tasks from cross-domain HRT applications are first sorted in decreasing order of their resource usage, (a combination of WCET, memory and bandwidth consumption metrics) and allocated to PEs which are sorted in ascending order of resource utilization. Mapping equal number of tasks to each

PE does not necessarily lead to a good load-balance, due to the variation in task execution cost. Brinkschulte et al. [164, 175] show that by considering the suitability of tasks to heterogeneous PEs, the distance between communicating tasks can give up to 20% improvement over a simple equal task-to-PE mapping approach. Mandelli et al. [27] propose a runtime mapping heuristic, which maps the initial task of a DAG based TG to the center of the NoC region with the lowest mean PE load. An incremental region search technique is employed to find a low-utilised placement for non-initial tasks. Their approach is evaluated on general-purpose synthetic and MPEG decoding applications to provide a well-balanced workload distribution whilst reducing communication volume. However, the approaches in [27, 37, 39] assume an accurate value of the new task's communication and computation budgets are known at runtime.

2.3.3.3 Communication/congestion-aware dynamic mapping heuristics

An unoptimised mapping may place communicating tasks away from each other, increasing communication latency, network traffic and congestion. Several NoC communication and congestion-aware heuristics are introduced by Carvalho et al. [32] to reduce the maximum and average load in the network as well as to obtain an even distribution of communication load and reduce packet latency. Their proposed, best neighbour (BN) heuristic, spirally searches a neighbour PE to map communicating tasks; the path load cost function is evaluated to determine the mapping which gives the lowest congestion out of the neighbours in the four directions. Singh et al. [154] extend the work in [32], to include PEs that can accommodate multiple tasks. In their approach, if two communicating tasks are mapped on the same node, then the NoC would not be used for that particular inter-task communication, thus leading to less congestion and communication volume. They evaluate a range of metrics such as total execution time, total platform energy consumption, average channel load, average packet latency; the effect of CCR variation in workload is also investigated. Kaushik et al. [33] present a runtime, pre-processing heuristic which takes the application TG as input and tries to minimize the communication latencies among tasks while simultaneously trying to balance the processing load on the PEs. The pre-processing algorithm is presented in Algorithm 2.1; the first phase merges the TG nodes together as long as their communication load is higher than computation load; the second phase performs load-balancing by merging the nodes in the updated TG. The reduced TG is then mapped on to the PEs in using NN as a heuristic. Their results on a cycle-accurate, 5×5 simulated NoC show 33% reduction in total execution time, 37% reduction in resource usage and 40% reduction in energy consumption compared with a communication-aware mapper, for generic synthetic workload as well as a MPEG-4 decoder. However, the runtime overhead of their algorithm is not given in their evaluation results. As seen in Algorithm 2.1, their technique relies heavily on operations on the application TG (e.g. node merging and searching), which may be slow on large TGs.

Network contention has a negative effect on network latency and performance. With the increase of concurrent task-graphs or applications being executed on-chip, communication density and network contention will arise. According to the classification in [176], *internal contention* occurs when tasks of the same application occupy the same network channel; *external contention* occurs when tasks of different applications occupy the same network channel. When TGs are mapped contiguously, external contention is minimized but internal contention is increased; and vice-versa for non-contiguous allocations. However, two overlapped edges will not always lead to congestion, if they do not share the same channel at the same time. In [176],

Algorithm 2.1: Communication and computation-aware graph pre-processing algorithm (taken from [33])

Input : TG(T,E)
Output: Optimized TG(T,E)
/* Phase - I */
1 **repeat**
2 Find node $t_i \in T$ having maximum computation load max_{pload} from TG(T,E);
3 Find edge $e_i \in E$ having maximum communication load max_{cload} from TG(T,E);
4 **if** $max_{pload} < max_{cload}$ **then**
5 Find computation load of connecting nodes t_p & t_q of edge e_i, i.e. $pload(t_p)$ and $pload(t_q)$;
6 **if** $(pload(t_p) + pload(t_q)) \leq max_{cload}$ **then**
7 Merge t_p & t_q to a single node if their mem. req. are satisfied ;
8 Update TG(T,E) ;
9 **else**
10 break; // goto second phase for optimization, i.e. start from step 13
11 **end**
12 **end**
13 **until** $max_{cload} > max_{pload}$;
/* Phase - II */
14 **repeat**
15 Find communicating nodes t_i and t_j having minimum computation loads $pload(t_i)$ and $pload(t_j)$ from updated TG(T,E) ;
16 **if** $(pload(t_i) + pload(t_j)) < max_{pload}$ **then**
17 Merge t_i & t_j to a single node if their memory requirements are satisfied and update TG(T,E) ;
18 **end**
19 **until** $max_{cload} > max_{pload}$;

in order to minimize the internal/external contention and communication cost, convex/near-convex regions are formed to map the application, and combined with user-level characteristics such as application periodicity, criticality and communication rate. However, the authors only consider a single task per PE and assume exact task communication costs are known.

Fattah et al. [177] attempt to reduce contention for heterogeneous multi-tasking platforms by keeping the mapping area of an application rectangular and carefully choosing where to map the first task. The task with the largest number of edges are mapped to the PE with the largest number of available neighbour PEs, keeping the mapped area as square as possible, to prevent fragmentation. Experiments showed 16% improvement over the allocation method in [176], but computationally more expensive than FF or NN. Similarly, Haghbayan et al. [178], address the first node selection problem (i.e. finding the first PE, around which an application/TG can be mapped). They divide the network into different sized squares to fit potential applications of different sizes; each square region is weighted and searched for a suitable fit to the new application in terms of a unified metric (spatial availability, dispersion and internal congestion). For a large 12x12 network, considering synthetic general-purpose and video processing applications (max 20 tasks in each TG), their method showed 21-28% reduction in congestion and dispersion over the state-of-the-art runtime task mappers. The work in [177], is extended in [179], by modelling the application as multiple rectangles and using a reduced complexity hill-climbing algorithm to find an allocation region. Their work reduced the average allocation fragmentation and communication cost for synthetic traffic, for heavily utilised scenarios.

2.3.3.4 Energy/power/thermal-aware dynamic mapping heuristics

Many mapping approaches have focused on reducing the energy consumption in the system, whilst maintaining acceptable performance. Energy consumption in a MPSoC is mainly due

to computation by the PEs and NoC communication. The volume and path distance of a message flow is directly related to communication energy consumption and hence, energy can be reduced by mapping communicating tasks close together or on the same PE [155, 180]. In [180], the task mapping search is conducted in a bounded network region, to minimise the communication energy; where, communication energy is modelled using the bit-energy metric given in [174]. The bounded region is increased by 1-hop if a suitable PE is not found. In [38] the work is extended by a clustering phase where tasks that exchange a high volume of communication is grouped together and placed on the same PE. Both techniques in [38, 180] are evaluated using image/video processing applications including MPEG4 decoding with functional-parallelism. Castilhos et al. [37] present a hierarchical runtime management and energy monitoring technique to obtain processor and NoC energy data at runtime. The global manager selects the cluster with the lowest accumulated energy to map an incoming application to; similarly within the clusters, the cluster manager selects the PE with the lowest energy consumption to map the initial task to.

In many of the existing work in NoC energy/power-aware dynamic task mapping, the heuristics use power macromodels to estimate the NoC dynamic power consumption. In [174] a bit-level NoC energy model is presented which takes into account the energy consumption of NoC links and routers between source and destination nodes. A comprehensive dynamic power model for a wormhole switched NoC is presented by Palesi et al. [181] and is shown in Eq. 2.7, where the power dissipated by the routers (P_r), network interface (P_n) and links (P_l) are taken into account. The h and f terms in Eq. 2.7, denote the number of hops between source and destination and the number of flits in the message flow. The model assumes that the power required to transmit a header and data flit is the same. In [181], the absolute power values of P_r, P_n and P_l are assumed to be 5.7mW, 5.3mW and 5.23mW respectively, where the buffers, arbitration, routing etc. are taken into account in P_r. It can be seen from Eq. 2.7, that the number of flows, their size (number of flits/bytes), and the route length all contribute and are directly proportional to the NoC power consumption.

$$P_m = \Big[(h+1)(f+1)P_r\Big] + \Big[2(f+1)P_n\Big] + \Big[h(f+1)P_l\Big] \tag{2.7}$$

Certain work has considered reducing the power dissipation of PEs by employing dynamic voltage and frequency scaling (DVFS) and shutting down idle PEs or leading certain areas of the silicon inactive (termed Dark Silicon) [28, 67]. In [67], compile time analysis is used to support a runtime mapping technique, which first assigns the minimum number of PEs to an application such that a throughput constraint is satisfied. Remaining PEs are given to applications which achieve the most energy saving with an additional PE. If the amount of available PEs are not sufficient for the application, then DVFS is employed to minimise the PE energy consumption. However, they do not allow processor sharing which could lead to under-utilised resources. A multi-objective control theoretic approach is explored in [28], to allocate sufficient power to applications at runtime, whilst improving throughput. Different metrics such as the NoC router and PE power measurements, buffer utilisation and application injection rate are fed into the resource manager periodically, to perform DVFS and per-core power-gating. Their technique protects the system from overshooting from power consumption when new applications enter the system. Evaluations are carried out on cycle-accurate simulators using generic synthetic workloads.

Efficient on-chip thermal management is also a concern in multicore systems as thermal hotspots and elevated operating temperatures can lead to device wear-out and failures. In [182], multicore thermal management has been addressed by using runtime learning algorithms assisted by on-chip temperature sensors. The learning agent adapts to workload variations and selects efficient thread-to-core mappings and CPU frequencies. For a set of multimedia applications, their method demonstrated 2-3 times lower system mean time to failure and up to 11% reduction in energy consumption.

2.3.3.5 Timing constraints aware dynamic mapping heuristics

Many of the dynamic mapping techniques that need to provide HRT guarantees perform simple utilisation based tests [34, 39, 163]. They determine if a processor will be overloaded for a given set of mapped tasks. For example, Liu and Layland [115] showed that for their task model, according to RMPO, the sufficient schedulability test is as given in Eq. 2.9; where utilisation is calculated as per Eq. 2.8. c_i is the task WCET, t_i is the task period and n denotes the number of tasks mapped to the target PE. In [163], both processor utilisation and communication bandwidth restrictions are considered when mapping tasks on a NoC platform. Both [39] and [34], assumed a TDMA arbitrated communication network in the target hardware platform. However, as outlined in Section 2.3.1.3, utilisation based tests are sufficient but not necessary to guarantee HRT constraints [118]. Furthermore, unlike RTA-based tests they do not take into account task blocking and do not offer a schedulability result for an individual task. HRT systems that perform RTA require knowledge of certain task characteristics (e.g. WCET, priority, deadlines, periodicity, etc.) in order to guarantee schedulability. If the variance between the WCET and the observed execution time is large, then the system can be under utilised. Furthermore, response-time calculations such as in Eq. 2.5 contains recurrence relations and ceiling operations which incur large execution overhead, and hence the schedulability test needs to be efficient to be able to use online.

$$\text{PE worst-case utilisation: } U(PE_i) = \sum_{i=1..n} \left\lceil \frac{c_i}{t_i} \right\rceil \tag{2.8}$$

$$\text{RMPO schedulability bound: } U(PE_i) \leq n(2^{1/n} - 1) \tag{2.9}$$

Dziurzanski et al. [134] perform approximate schedulability tests to reduce the number of exact RTA executions. RTA is performed per task to core mapping, to ensure hard deadlines can be met. In their approach the slack value of executed jobs are monitored and used within a feedback controller to improve future admission rates. In [158], SRT tasks have been mapped to PEs that have the highest amount of average slack, where the PE slack values are periodically monitored. Ali et al. [183] propose a critical-path first heuristic which attempts to maximise the system utilisation and minimise the end-to-end WCRT of DAG based applications. In their method, all possible paths are extracted from the DAG and mapped according to the order of cumulative weight of the path. However, their utilisation analysis does not capture the dynamic offsets introduced by task precedence constraints and they assume a simple NoC model with high bandwidth and no contention.

2.3.3.6 Runtime mapping techniques for video processing

Many of the previously discussed dynamic task mapping approaches are evaluated on multimedia related benchmarks (e.g. [27, 154, 155]). However, they are general-purpose mapping techniques applicable to a range of application types. Several task mapping work have represented applications using dataflow models of computation (e.g. [31, 34, 151, 183]) which are suited to represent streaming multimedia applications; however, these techniques target generic streaming workloads and do not exploit characteristics of video processing workloads (e.g. dependency structures, execution patterns, data communication volumes etc.).

The default behaviour of popular multi-threaded video decoders such as ffmpeg [184], is to spawn as many threads as there are processing cores on the platform and allow the operating system load balance the cores. A similar approach is taken in [185], where the OS is allowed to map HEVC CTU-level decoding threads on a 12 core system. They observe a saturation in speed-up as the number of threads exceed 8 due to context switching and scheduling overhead. Using profiled information, they are able to analytically derive the minimum number of cores required for a decoding task. Pal et al. [186] propose a dynamic core allocation technique for H.264 slice-level parallel decoder. Each frame is decoded using multiple threads/cores, based on the amount of remaining slack or overshoot of decoding time, the number of cores utilised in the system is dynamically varied to save energy. In [61], slice-level parallel video decoding is carried out, where decoding tasks are not migrated between cores. Stream parsing, mapping and filtering tasks are handled by a single master core and slave cores decode the slices in parallel with goal of minimising the decoding time. In their model they assume fixed GoP dependency structures, bus-based interconnects and shared-memory communication. A similar master-slave approach is employed in [100], where at runtime a dispatching core waits until an idle signal is received from slave cores, before dispatching the reference frame data and compressed frames to a slave core for processing. However, they assume a circuit-switched NoC which could lead to low network utilisation. Blanch et al. [149] introduce several video decoding, task allocation and selection heuristics based on earliest PE availability and task execution costs. Current frame decoding costs are predicted using previous frame execution costs. Evaluations of their techniques of multiple stream decoding on heterogeneous PEs are carried out via simulation.

Khan et al. [146, 187] jointly consider workload balancing and power reduction when mapping HEVC encoding Tiles to homogeneous cores connected by a NoC. They use a FFD bin packing heuristic to maximise the utilisation of the cores whilst minimising the number of cores used. Unlike in [61], their allocation method assumes multiple Tiles can be assigned to a single core, to avoid idle cores. PE frequency and the number of PEs required is adapted to sustain the timing constraints (i.e. frame rate) and application power budget, during HEVC encoding [146]. They continuously monitor the frame-rate of a video in order to determine if more or less power should be given to that encoding thread.

2.3.3.7 Hybrid mapping approaches

Static mapping approaches cannot cope with dynamic workloads, and online mapping approaches that use heuristics cannot be computationally expensive hence the mapping solutions are of lower quality. In hybrid mapping approaches, multiple mapping solutions are derived at design time using computation expensive optimisation approaches, and then applied during runtime considering the workload and resource availability. For example, in [31, 188] a design

space exploration (DSE) is conducted offline of different mappings and corresponding application throughputs and energy consumption values; each of the DSE points are then stored in a database. In [188], the design space is pruned, and inefficient mappings are discarded. In [31], the mappings are classified according to workload scenario/mode groups. For each application, the stored mappings are searched at runtime to find a configuration which satisfies the throughput constraint and uses the minimum number of processors. The hybrid mapping technique by Jung et al. [67] can support intermittent processor failures, hence supporting a higher degree of runtime variability. In [189], the hybrid mapping approach takes into account the shared resource contention. An offline DSE stage is coupled with a formal performance analysis which verifies real-time guarantees. The survey by Singh et al. [30] presents more examples on hybrid mapping techniques. Hybrid approaches can still be inefficient if the application behaviour is too dynamic and the design-time knowledge is limited. Also, it can lead to high runtime and memory usage overhead for storing and searching all the optimal mapping solutions derived at design time.

2.3.4 Runtime task re-allocation

Task re-allocation (also referred to as task migration/remapping) is the process of adaptively changing a task's PE mapping at runtime. Adaptive systems perform task migration/remapping at runtime to improve metrics such as performance, energy savings, load balance, to circumvent system faults etc. A distributed, scalable, task migration policy to avoid thermal hot-spots and workload imbalance in many-core systems is presented in [190]. Each core decides if a migration to a neighbouring core, is beneficial or not in a decentralised manner. Derin et al. [191] consider the problem of task remapping when a faulty core has been detected in the system, for improving fault-tolerance, and to continue operating under degraded performance. When remapping a task in a faulty core, they attempt to remap communicating tasks together, as long as the destination node is not overloaded. However, they do not take into account the migration overhead when deciding to remap a task. In [31], the system continuously monitors the workload execution to identify performance problems, due to bad initial mapping (i.e poor locality and/or load imbalance); migration is performed to improve the performance and energy saving. They propose to remap a video frame decoding task only after it has finished processing a frame, such that migration overhead can be minimised. Holmbacka et al. [192] use a middleware-based migration framework to reduce the system power dissipation whilst maintaining required video decoding QoS levels. Similar to [31], in [192], explicit migration positions are marked in the application code to achieve migration predictability. Harbin and Indrusiak [193], present a dynamic task remapping approach to reduce the network traffic contention, thereby reducing flow latencies and power consumption in a non-preemptive NoC. In their approach, each NoC router keeps track of flow contention and relays this information to a central management core. The central manager then periodically reallocates the tasks according to a congestion-aware heuristic. Tasks with high level of communication are mapped closer together or on the same PE. However, they do not model the overhead of migration and/or remapping process, and intentionally keep the remapping period relatively long to reduce the overheads.

Task migration incurs different overheads based on the implementation, system architecture and communication model. Migration costs are largely dominated by additional interconnect congestion as the task size increases [194]. If a NoC-based distributed memory architecture

with message-passing communication is used, then the task code, data contained in the memory page, and task context has to be transmitted over the NoC from the source PE local memory to the target PE's local memory [195]. Once the transfer has been completed the task has to be re-scheduled in the target PE and all tasks communicating with the migrated tasks should be notified of its new location. Betting and Brinkschulte [196] analytically derive worst-case overhead costs for their decentralised migration process which includes the steps: task termination at source, context transfer, restore procedure at destination, waiting until ready to run and any jitter involved in these functions. However, in [192], the authors assume a shared memory architecture where only a pointer to the task handle needs to be physically moved; hence, the task size or migration distance does not influence the migration overhead. In [194], a replica of the task is available in each core and registered in each local OS; only one replica of the task can be run at a time. When migration is needed, the running task is suspended in the source PE and resumed in the destination PE along with the task-context of the migrated task. This method has low migration overhead but wastes local memory to store task replicas.

Table 2.3: Summary of state-of-the-art in dynamic task mapping/remapping techniques for many-cores

Ref.	Interconnect	Arch.	RM	Mig.	Key metric in heuristic	Goals	Applications	Eval.	Novelty
[183]	NoC, TDMA (–)	Hom.	Cent.	No	Util.-based schedulability, spiral search	Maximise resources, reduce end-to-end WCRT	Synthetic (streaming app.)	HL sim.	Critical path heuristic
[151]	Bus (–)	Hom.	Cent.	No	First-fit	Latency, Minimise resources	Synthetic (streaming app.)	HL sim.	CSDF graph scheduling
[149]	Bus (–)	Het.	Cent.	No	Earliest available PE, Earliest completion time	Min. frame deadline misses, throughput, video quality	Multi-video decoding	HL sim.	Exec. prediction, frame pri. assignment
[155, 176]	2*NoC (data+control)	Hom.	Cent.	No	Manhattan dist., PE avail.	Comm. cost, energy, exec. time, contention reduction	Synthetic	HL/LL sim.	Region selection User-awarenes
[32]	NoC, RR arb.	Het.	Cent.	No	Path load, PE avail., hop dist.	Packet latency, channel load, exec. time, mapping overhead, energy	Synthetic, multimedia (inc. MPEG-4)	RTL sim	Communication-awareness
[134]	NoC, priority arb.	Hom.	Cent.	No	Schedulability(RTA), slack	Deadlines misses, mapping overhead	Synthetic HPC wokloads	TLM sim.	Control theory, HRT
[177]	NoC, random arb.	Hom.	Cent.	No	PE avail.	Network contention reduction	Synthetic	C_ACC sim. FPGA	First-task & node selection
[178]	NoC, random arb.	Hom.	Cent.	No	PE avail., dispersion, congestion	Avg. packet latency, congestion, energy	Synthetic, multimedia (inc. MPEG-4)	C_ACC sim.	Network partitioned search
[39]	NoC, TDMA (–)	Het.	Cent.	No	FF-decreasing, util. test	Resource utilisation, exec. time	Synthetic	HL sim.	Feedback-based instantaneous utilisation tests
[185]	Bus (–)	Hom.	Cent.	Both	Pipeline on PEs, est. exec. time	Framerate, exec. time, resource utilisation	Real HEVC vid. streams	HW	Hybrid HEVC parallelisation
[67]	NoC, RR arb.	Hom.	Cent.	Yes	Min. PEs, throughput constraint, energy savings	Energy, frame rate	Synthetic Multimedia (inc. H.264 decoding)	C_ACC sim.	Hybrid mapping
[33]	NoC, RR arb.	Hom.	Cent.	No	Task clustering, task comp. and comm. load	Exec. time, load balance, resource util., energy	Synthetic, multimedia (inc. MPEG-4)	C_ACC sim.	TG clustering
[100]	NoC, TDMA	Hom.	Cent.	No	Parallelism degree, balancing comp. and comm.delay	Framerate, exec time	Multi-view H.264 dec. (frame/view-parallel)	HL sim.	Uses message passing. DSE
[61]	Bus (–)	Hom.	Cent.	No	Plat./app. constraints. CPU freq.; slack	Avg. power per PE, deadline misses	Slice/frame-level parallel vid. decoding	HL/C_ACC sim.	Hybrid mapping
[34]	Bus, TDMA (–)	Hom.	Cent.	No	First-fit, task clustering	Resource util., bandwith saving	Synthetic	HL sim.	HRT consideration

Table 2.3 Continued: Summary of state-of-the-art in dynamic task mapping/remapping techniques for many-cores

Ref.	Interconnect	Arch.	RM	Mig.	Key metric in heuristic	Goals	Applications	Eval.	Novelty
[38]	NoC, RR arb.	Hom.	Cent.	No	Comm. volume and energy	Energy, exec. time, latency	Synthetic, multimedia (inc. MPEG-4)	RTL sim	Model-based approach
[186]	Shared bus & mem.	Both	Cent.	Both	PE avail., decode time of prev. frame/slice, slack	Exec. time, energy, latency	Real video streams (H.264), frame/slice level parallel	C_APX sim.	Multicore power mode consideration
[31]	Shared bus & mem.	Het.	Cent.	Yes	Enegy, exec. cost, task locality, periodicity, comm. volume, CCR. PE util.	Exec. time and energy	Synthetic, multimedia (inc. MPEG)	HL/TLM sim.	Hybrid mapping. Workload scenarios
[28]	NoC, random arb.	Hom.	Cent.	No	PE and router util., injection rate, power usage. PE util. NoC congestion	Power, throughput	Synthetic, multimedia (inc. MPEG-4)	C_ACC sim.	Control theory support
[187]	NoC, shared mem. (–)	Hom.	Cent.	Yes	PE frequency, frame rate constraints. FF decreasing	Video quality, minimise resources, power	Real vid. Streams. HEVC encoding	C_APX sim.	Power efficient Tile formation
[154]	NoC, RR arb.	Both	Cent.	No	Hop dist., channel load, task clustering.	Exec. time, energy, packet latency, channel load,	Synthetic	RTL sim	CCR eval. Comm-aware bin packing
[179]	NoC, random arb.	Hom.	Cent.	No	Manhattan dist., App. size	Load distribution, power	Synthetic	C_ACC sim.	Multiple rectangle applications
[146]	NoC, shared mem. (–)	Hom.	Cent.	Yes	Image quality, previous thread exec. time, CPU freq., PE avail.	Video quality, energy, fps per watt, minimise resources	Real vid. Streams. HEVC enc.	C_ACC sim./HW	Feedback-based processor freq. regulation
[175, 196]	NoC, TDMA (–)	Het.	F.Dist	Yes	Load, suitability, hop dist.	Mapping quality, load balance	Synthetic	HL sim.	Bio-inspired, self-X
[163]	NoC (–)	Hom.	F.Dist	No	PE util./load and link bandwidth restrictions. Hop dist.. Path load	Network load, monitoring overhead, mapping success	Synthetic	HL sim.	Child task mapping executed by parent
[161]	NoC, RR arb.	Hom.	F.Dist	Yes	Hop dist., PE load. Channel bandwidth. Spiral search	Link contention/bandwidth, PE load.	Synthetic	FPGA	Hybrid mapping
[162]	NoC, RR arb.	Het.	F.Dist	Yes	NoC util., PE avail.	Speedup, NoC util., energy, h/w overhead	Parallel matrix mult.	FPGA	RM specific HW arch.
[27]	NoC, RR arb.	Hom.	Hier.	No	Link load, hop dist., comm. volume	Load distribution, energy, exec. time, reliability	Synthetic, multimedia (inc. MPEG-4)	RTL sim	Heuristics for initial & non-initial tasks
[37, 157]	NoC, RR arb.	Hom.	Hier.	Yes	Regional energy, Hop dist.	Exec. time, hop dist., energy, load dist., NoC traffic	Synthetic, multimedia (inc. MPEG-4)	C_ACC sim.	Cluster-based, multiple objectives

Table 2.3 Continued: Summary of state-of-the-art in dynamic task mapping/remapping techniques for many-cores

Ref.	Interconnect	Arch.	RM	Mig.	Key metric in heuristic	Goals	Applications	Eval.	Novelty
[158]	NoC, RR arb.	Hom.	Hier.	Yes	Slack, PE util., deadline misses (soft)	Task latency	Synthetic, multimedia (inc. MPEG)	RTL sim	Least slack time sched., both SRT/HRT
[159]	NoC (–)	Hom.	Hier.	Yes	PE util., manhattan dist., remapping perf. gain	Overhead of RM. Speedup	Synthetic, malleable apps	HL sim./HW	Dynamic selection of a manager per app.
[35]	Adaptive NoC.	Het.	Hier.	Yes	PE and link util., PE energy usage. App. resource req. Hop dist.	Exec. time, mapping time, monitorng overhead.	Synthetic, image processing	FPGA	First cluster based RM
[36]	NoC (–)	Hom.	Hier.	Yes	PE avail., remapping perf. gain. Greedy heuristics	Overhead of RM. Speedup	Synthetic, malleable apps	HL sim.	Random regions and cluster managers

Abbreviations/notations used in Table 2.3:

- (–): denotes no contention is assumed
- Resource management (RM) - *Edist*: Fully distributed; *Hier*: Hierarchical; *Cent*: Central
- Architecture: *Hom.*: Homogeneous; *Het.*: Heterogeneous
- Interconnect: *arb.*: Arbitration; *RR*: Round-robin
- *Mig.*: Task migration performed (Yes/No/Both considered)
- Evaluation platform *Eval.*: Cycle-accurate *(C_ACC)* simulations, Cycle-approximate *(C_APX)* simulations, High-level *(HL)*/Low-level*(LL)* simulation, Hardware *(HW)* platform, Register-transfer-level *(RTL)* simulation, Transaction-level model *(TLM)*

2.4 Summary

This chapter presented a literature survey on three domains: parallel video decoding applications; challenges faced in many-core platforms in achieving both predictability and performance; and the state-of-the-art in dynamic resource management for many-core platforms.

Firstly, a basic overview of video encoding and decoding is presented in Section 2.1.1, before moving on to more complex encoding principles used in modern codecs (Section 2.1.2). With each new generation of video codecs the decoding complexity of the videos also increases. As discussed in Section 2.1.4 different parallelisation techniques are used to manage the decoding complexity and achieve the required throughput. Many parallel processing approaches to video decoding assume a shared memory, multi-threaded architecture which offer ease of implementation and flexibility. However, they often face issues with performance scalability due to main memory contention. In contrast, the embedded systems community (as discussed in Section 2.3.3), often assume a distributed memory architecture when decoding video streams. They also assume a functional-parallel partitioning approach to video decoding which is shown by the video processing research community to be inferior to data-parallel decoding.

As shown in Section 2.1.3, the resources required to decode video streams vary considerably based on the spatial and temporal properties in the video and the complexity of the different coding tools used by the encoder. Due to this highly varying nature of video streams, it is increasingly difficult to model/predict the complexity and resource requirements of video decoding workloads at design time (Section 2.1.5). Many of the existing models do not take into account certain workload characteristics such as block-level variations, reference data patterns and frame dependency structures. These properties are required when performing design space exploration for communication-centric platforms such as NoC-based many-cores.

Network-on-chip interconnects, introduced in Section 2.2.1 are the de-facto form of interconnects in many-core platforms that offer improved scalability, power-efficiency and flexibility over traditional bus-based interconnects. However, as applications become more data-intensive, the complex dependencies between applications and tasks within the same application cause network contention. Resource contention can limit application performance but, more importantly, can be hazardous to real-time systems where predictability of network traffic behaviour needs to be analysed and latencies bounded. As discussed in Section 2.2.2, service guarantees in these systems typically are provided by resource reservation or response-time analysis. The issues faced by many-core systems due to memory transactions and off-chip memory traffic is discussed in Section 2.2.3. Many of the MPSoC platforms now have multiple memory controllers on-chip. Efficient memory controller placement and data locality can help to reduce memory traffic latencies. However, the assignment of application/task to memory controller to overcome memory traffic congestion and improve performance is still an open problem.

The final section of the chapter surveys resource management methods for many-core platforms, which involves admission control, scheduling, mapping and remapping of applications. An overview of scheduling and response-time analysis for multiprocessor real-time systems is first presented. Section 2.3.1.3 discusses the merits and drawbacks of deterministic, probabilistic and feedback-based admission control systems in terms of predictability offered vs. adaptability to dynamic workloads. Section 2.3.1.4 shows how previous work on video-centric scheduling techniques can provide better performance and efficient use of resources than conventional multiprocessor scheduling algorithms. In the last decade, considerable effort has

been made to shift from a centralised resource management model to a hierarchical or fully distributed model to reduce monitoring overhead and improve scalability (Section 2.3.2). Out of these efforts, bio-inspired, autonomous management techniques have shown to be a promising research avenue for future large-scale adaptive systems. Many of the reviewed distributed management protocols have complex rules and communication protocols; several depend heavily on architectural changes and only a few consider the low-level issues of communication, such as network resource contention. Sections 2.3.3.1 to 2.3.3.7 discuss the state-of-the-art in heuristic based runtime task allocation techniques for many-cores; a summary of which is presented in Table 2.3. Runtime task mapping techniques are more suited for dynamic/unknown workloads as seen in live video decoding applications. Initial task to processor mapping can later be further improved by adaptive remapping/migration techniques (Section 2.3.4).

Existing techniques in dynamic task mapping follow a closed-loop strategy, depending heavily on monitoring information to guide a mapping heuristic. The periodicity of the monitoring needs to be tuned to balance the traffic overhead and mapping quality. Most of the reviewed techniques attempt to improve performance metrics such as load distribution, power/energy, communication cost, congestion avoidance and average execution time. However, real-time metrics such as task lateness, response-time or deadline misses are rare optimisation objectives in dynamic task mapping. As seen in Table 2.3, many existing mapping heuristics consider PE utilisation and network path load as a metric. Most of the existing work on resource allocation considers simple yet unpredictable NoC arbitration and PE scheduling (e.g. RR, FCFS). This makes it difficult to account for resource contention when performing dynamic task mapping. Furthermore, most existing techniques target a generic application model and do not consider any application-specific task mapping heuristics to improve the mapping quality. This motivates the proposal of new techniques that take into account both resource contention and application awareness when designing a mapping heuristic.

Chapter 3

Evaluation platform, metrics and problem statement

This chapter presents the experimental platform and the metrics used to test the hypotheses of this thesis (Section 1.3). This chapter first describes the application and platform models used for evaluation. The application model is described in terms of its real-time properties, execution profile and arrival patterns. The platform model is defined in terms of its computation and communication elements. Secondly, metrics used to evaluate the proposed resource management techniques are defined. The metrics allow us to quantify the predictability, performance and efficiency of the proposed techniques. Next the simulation-based evaluation methodology is introduced, followed by an outline of the problem statement and objectives of this thesis.

3.1 System model

This section introduces the application and platform models which forms the basis of the experimental framework in this thesis. Assumptions made in the models are described with justification; certain model assumptions will be lifted in the following chapters.

3.1.1 Application model

This section outlines the multi-stream video decoding application model. As described in Section 2.1.1, video frames can be of type: I (Intra), P (Predictive) or B (Bi-directional) encoded; i.e. *frame type* denoted as $f_t = \{I, P, B\}$. Video streams are assumed to be MPEG-2 encoded 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 reduce error propagation [43]. As discussed in Section 2.1.4, decoding a video stream can be performed at different levels of functional and/or data granularities; each technique having their merits and drawbacks. A frame-level data parallel video decoding approach is assumed, as it does not require stream instrumentation and it offers a balance between data dependent communication and parallelism.

As shown in the system overview diagram in Figure 3.1, 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, where a job is denoted as J_i. Video streams have varying resolutions defined as $res(VS_i) = \text{frame\_height} \times \text{frame\_width}$. A job, represents an MPEG GoP and is modelled as a directed acyclic graph (DAG) with a fixed dependency structure, as depicted in Figure 3.2. 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). The weight of a node in the TG is the frame-decoding computation cost and the weight of an edge represents the amount of reference frame data. 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 3.2, 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-chain $= \{\tau_1, \tau_2, ..., \tau_x\}$ is a topologically ordered, non-repeating, set of nodes of the TG where each task sends a message flow to the next task. In this work a task chain is also referred to as a *simple path* or *path*.

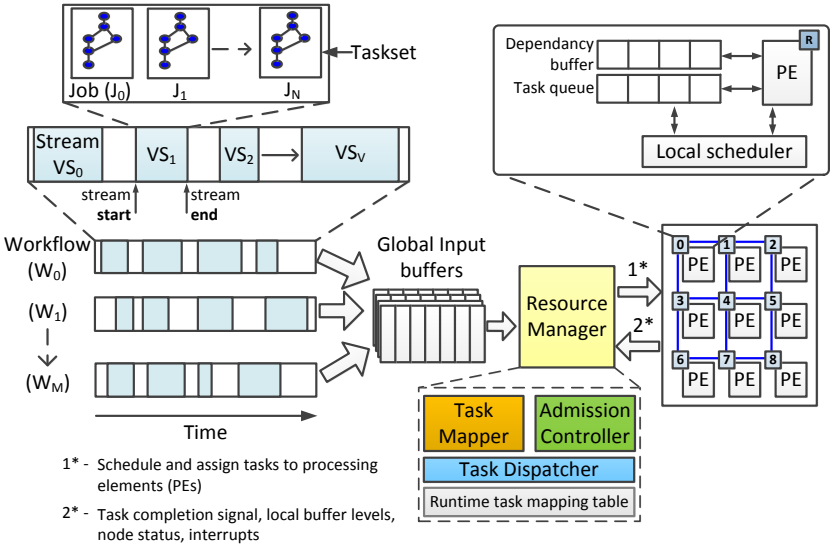


Figure 3.1: System overview diagram

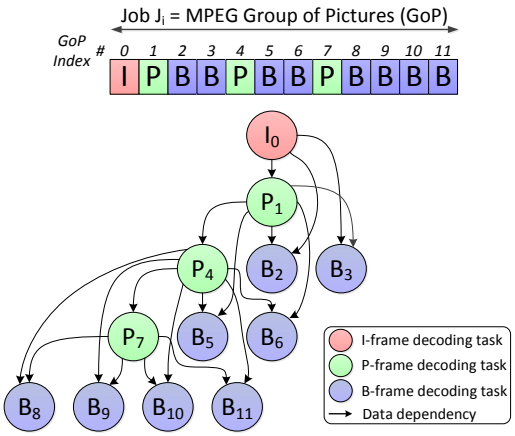


Figure 3.2: MPEG GoP data precedence and task communication graph (weights are not shown in illustration)

A task τ_i is characterised by the following tuple: $(p_i, t_i, x_i, c_i, a_i^t)$; where p_i is the 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, a_i^t is the arrival time (also referred to as release time) of the task τ_i. A task will wait for the dependent data to arrive before execution. Hence, w_i denotes the waiting time and s_i^t, e_i^t denotes the relative start and end times of execution. Tasks are sporadic, preemptive and have fixed priority. Individual task deadlines are unknown; however, each job is considered schedulable 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 is assumed to be fixed at 25fps for all video streams, and $|J_i|$ denotes the number of tasks in a job (cardinality). 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. 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. If one or more of a task's children are mapped to the same PE, then only one data flow is sent out in order to avoid flow redundancy. Flows inherit the t_i and p_i of the sender/source task. The payload PL_i of a message flow (also commonly referred to as *communication volume*) is equivalent to the reference frame data size which is given by Eq. (3.1), where bpp refers to *bits per pixel*.

$$\text{Reference data payload: } PL_{Msg_i} = res(VS_i) \times bpp \tag{3.1}$$

3.1.1.1 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 the video stream. As explained previously in Section 2.1.5.1, MPEG frame decoding time (x_i) can be estimated by the block-level model designed by Tan et al. [70]. Based on this model, x_i is modelled as per Eq. (3.3), which is an adaptation of Eq. (2.1). In Eq. (3.3), the number of type j blocks (denoted M_j) is modelled as a uniform random variable between $\{0, \max M_j\}$, where $\max M_j$ as defined in Eq. (3.2), is the maximum amount of blocks for a given video resolution.

Figure 3.3 shows the distribution of decoding time, per frame type, of 200 synthetically generated jobs (MPEG GoPs). 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 of IDCT only blocks which have a higher weighting. These distributions correlate well with previous MPEG real video stream decoding analysis seen in [60]. As defined in Eq. (3.4), 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. The actual execution cost of the task x_i, is unknown to the system at runtime, and only the worst-case execution cost c_i is known. In a real implementation the c_i can be estimated via offline execution profiling.

$$\max M_j = \frac{res(VS_i)}{block\_size}, \text{ (0 if block } M_j \text{ not enabled in frame type)} \tag{3.2}$$

$$x_i = w_0 + \sum_{j=1}^{9} w_j \times rand(0, \max M_j) \tag{3.3}$$

$$c_i(f_t) = \max\{x_i \text{ of } \tau_i \in VS_i \mid \text{frame type} = f_t\} \tag{3.4}$$

3.1.1.2 Task priority assignment

Assigning priorities to tasks not only enables the analysis of its timing properties (using techniques such as RTA), it also guarantees higher priority tasks do not get interrupted during its execution. The use of task priorities and fixed priority preemptive scheduling, is a design decision chosen to provide the video streams with a high level of timing predictability. Flows inherit

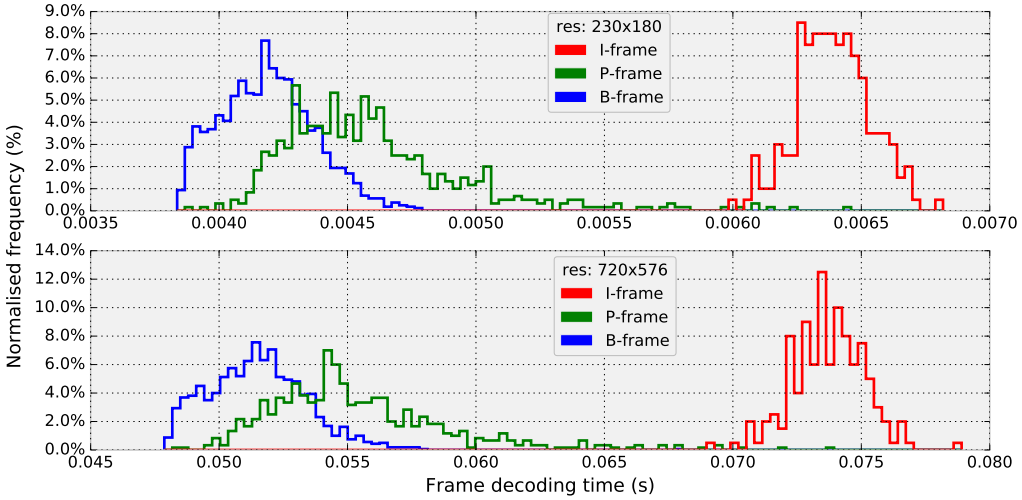


Figure 3.3: Frame decoding time distribution of synthetically generated MPEG decoding workload (top: 230×180, bottom: 720×576 resolution videos, 200 jobs each)

the priority of their source tasks. A higher p_i value denotes a higher priority. Similar to the dynamic mapping schemes presented in this work, the priority assignment between different jobs of the same video stream do not change. The same priority values are used for tasks in subsequent jobs of that stream. This type of policy combined with priority-preemptive scheduling, enables us to analyse the worst-case timing properties of the video streams (as discussed in Section 2.3.1.2).

In this initial application model tasks (i.e. frames) of a video stream are assigned fixed priorities, randomly at design time with the following constraint. It is assumed the priorities are provided by the application or software designer at design time. It should be emphasised that the resource management approaches described in this thesis primarily focus on optimising the placement of the tasks in the system independent to the priority assignment scheme. As such the mapping techniques can work around any given priority assignment scheme. In future chapters a more sensible priority assignment policy is used which is commonly seen in video streaming applications. The problem of optimising priority assignment is out-of-scope to this work.

3.1.1.3 Video/Job arrival rate

In certain HRT multi-stream video systems such as in [17], the number and start time of video streams are fixed and known in advance. However, this may not be true for certain SRT multiple stream decoding systems such as in [22], where video streams can start/stop arbitrarily. To generalise, the application model assumes the start and end time of a video stream VS_i are arbitrary and not known at design time. However, the model assumes two consecutive videos in the same workflow have a random inter-arrival time as specified by Eq. (3.5). VS^{rate}_{min} and VS^{rate}_{max} are model variables (units in seconds) that can be varied to control the temporal separation between two consecutive videos.

$$\text{Inter-video arrival time: } rate(VS_i) = rand(VS^{rate}_{min}, VS^{rate}_{max}) \tag{3.5}$$

As described in Section 2.1.5, video streams typically use VBR encoding to vary the bitrate

according to the content. VBR has a user-specified maximum bitrate, which can be used to represent the minimum inter-arrival time of consecutive frames (ignoring transmission delays). This behaviour is adapted into the application model, such that the job arrival rate is assumed to be sporadic with a known minimum inter-arrival time. In this work, the minimum inter-arrival time of a job is assumed to be equal to D_{e2e}, where $D_{e2e} = |J_i|/fps$ (the display rate). All tasks of a job J_i are assumed to arrive at the same time instant; hence, all tasks in a job have equal periods such that $t_i = D_{e2e}$. The inter-arrival time of consecutive jobs, can therefore be modelled as per Eq. (3.6). The parameters J^{rate}_{min} and J^{rate}_{max} are set to 1.0 and 1.3 respectively for all experiments in this thesis. 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} can make the system more idle.

$$\text{Inter-job arrival time: } rate(J_i) = rand(J^{rate}_{min} \times D_{e2e}, J^{rate}_{max} \times D_{e2e}), \text{ where } \{J^{rate}_{min}, J^{rate}_{max}\} \geq 1.0 \tag{3.6}$$

3.1.1.4 Workload profile

The workload profile of the system is characterised by the following various factors of the hierarchical application model (Section 3.1.1) :

- The maximum number of simultaneous workflows ($|W|$).
- The maximum number of video streams in a workflow ($|VS|$).
- The inter-video ($VS^{rate}_{min}, VS^{rate}_{max}$) and inter-job ($J^{rate}_{min}, J^{rate}_{max}$) dispatch rate : lower values of these parameters can increase the number of parallel video streams decoded by the system.
- The spatial resolution of the video streams: according to Eq. (3.3), higher resolutions will have higher computation requirements.
- The maximum number of jobs per video stream: larger number of jobs will result in long running experiments.

The above factors can be combined to form a quantitative Workload factor WL as per Eq. (3.7). A WL contains multiple workflows (W) and each workflow contains multiple video streams VS_i.

$$\text{Total workload (WL)} = \sum_{W_{i=0}}^{|W|} \sum_{VS_{j=0}}^{|VS|} res(VS_i) \tag{3.7}$$

3.1.2 Platform model

The many-core platform model is composed of P homogeneous PEs (e.g. CPUs) connected by a mesh, NoC interconnect. The NoC platform model uses wormhole packet switching, fixed priority preemptive arbitration, has a mesh topology and uses the static XY deterministic algorithm for routing such as in [82]. The NoC link arbiters are priority-preemptive and as explained in Sections 2.2.2 and 2.3.1.2, they offer means of calculating response-time upper bounds for traffic flows under worst-case conditions.

The platform model assumes inter-PE communication is performed via the NoC by passing messages. Each PE has a priority-preemptive local task scheduler and local memory, which contains a task's queue and dependency buffer. The local memory is assumed to be sufficiently large to hold multiple decoded reference frames. Modern stacked 3D NoC architectures such as

in [197] have a separate layer in which memory cores are located. A PE has access to the memory core directly above it on the memory layer, using short through silicon vias (TSV). This type of architecture can be used where large local memories are required. The PE upon completing a task's execution, transmits its output (i.e. decoded MPEG frame data) to the appropriate PE's dependency buffer. A task can begin execution *iff* all its data dependencies have arrived at the dependency buffer. Each task can only be executed on a single PE and migration is not performed. 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.

Once a task is released from a global input buffer, it is sent to the task queue of the PE, and the respective PE notified via an interrupt. This preliminary version of the platform model does not take into account the delay incurred during task transfer from global input buffer to PE local task queue. The size of the global input buffer, task queue and dependency buffers are model parameters.

3.1.2.1 Closed-loop resource manager

The initial model of the resource manager is assumed to behave in a closed-loop nature (a definition of closed/open-loop resource management can be found in Section 2.3.2.5). The resource manager (RM) of the system in this thesis, has a range of duties in the system:

- *Admission control*: deciding which jobs/video streams to admit/reject
- *Task mapping and priority assignment (MP&PR)*: Mapping tasks to the PEs and assigning fixed priorities to the tasks/flows of the application.
- *Task dispatching*: transferring a task (code and data) from the global buffers to the PEs.

Therefore, the terms RM, task dispatcher (TD), mapper, admission controller (AC) are interchangeable as they are all performed by one entity. It is assumed the RM does not execute frame decoding tasks, it only performs resource management operations (such as mapping/admission control/dispatching etc.). Task dispatching involves the timely release of tasks from the *global input buffers* to the PE task queues. Tasks are held in the global input buffers until their parent tasks have completed execution and their output data is transmitted to the required destination processing node. PEs will notify the RM via interrupts once a task has completed its execution, so that the RM can release child tasks of the completed task. The job structure is fixed and small, hence the overhead of searching for the child tasks is very small. Similarly, the RM notification message payload is not more than 1-2 bytes, hence the notification communication overhead is assumed to be negligible. The RM keeps track of the admitted and active video streams, their real-time task characteristics and the runtime task to PE mapping configuration in the *task mapping table*.

3.1.2.2 Interference in the platform

Figure 3.4, illustrates the blocking behaviour in the platform, due to the priority preemptive PE schedulers and NoC arbiters. Tasks and flows can get blocked by higher priority tasks and flows. Figure 3.4(b) shows an example of the execution of MPEG frame decoding tasks I_0, P_1, B_2 and B_3

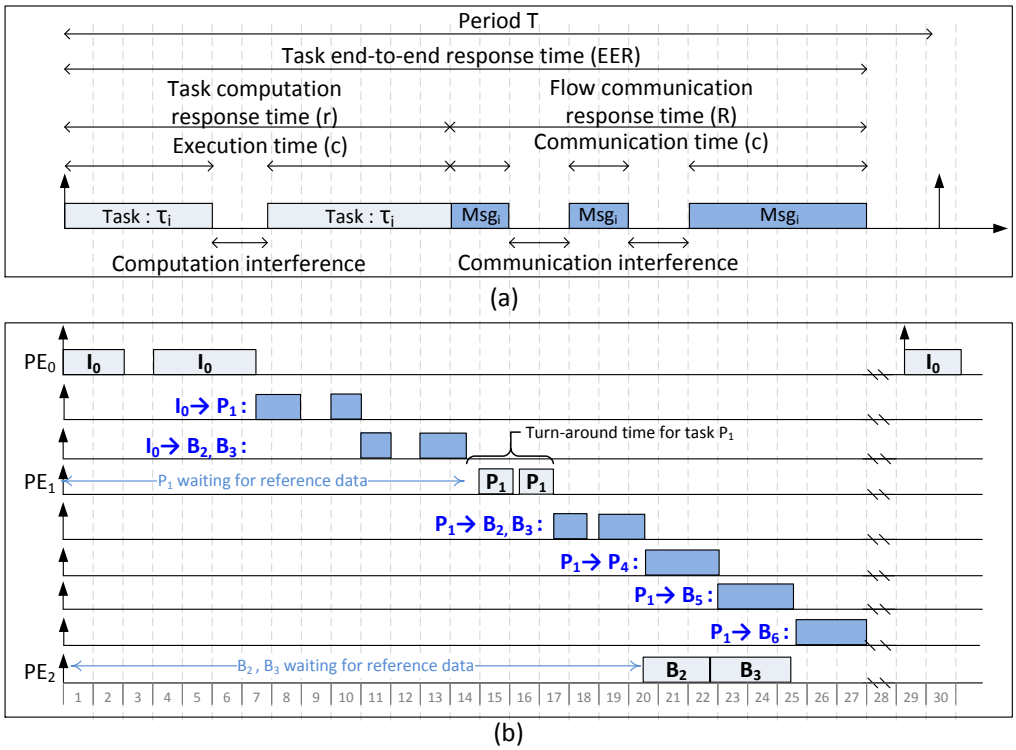


Figure 3.4: (a) End-to-end response time of a communicating task, showing tasks and flows being interfered. (b) Example of 4 MPEG frame tasks executed over 3 processors

on 3 different PEs. I_0 and P_1 are mapped on PE_0 and PE_1 respectively, and both B_2 and B_3 are mapped on the same PE - PE_2. This example follows the data precedence task graph shown in Figure 3.2. In this example B_2 and B_3 are assumed to be mapped on to the same PE; hence, only a single flow is sent by I_0 and P_1 to the destination PE_2, where the B-frame tasks are mapped. However, B_2 and B_3 cannot start execution until PE_2 receives the decoded frame data from P_1. I_0 and P_1 both transmit multiple flows to their child tasks, however these flows cannot be sent in parallel due to the local-link contention. The idle gaps in the PEs and network resources can be exploited by executing tasks and flows from other streams in a pipeline fashion.

3.2 Metrics

Quantitative metrics not only help us to compare and evaluate different resource management policies but can even be used within resource allocation heuristics. In this section the evaluation metrics are described in a generic form, independent to the experimental platform. Rather than choosing purely domain specific metrics, this work attempts to choose metrics that are meaningful and comparable across the real-time and embedded systems domain. Metrics are calculated from the raw measurements that are gathered during the experiment. The metrics used in this thesis is classified into two primary objectives - *predictability* and *utilisation-based performance* metrics. Additional secondary metrics - *overhead* and *energy* metrics are used to quantify the management overheads and energy efficiency of the resource management policies respectively.

3.2.1 Predictability metrics

Predictability metrics are related to timing properties of the workload that is executed by the system. The resource allocation scheme can affect a task/jobs responsiveness, as certain mapping policies may cause higher latencies than others due to interference. In the case of HRT systems, improving the schedulability of the system (i.e. reducing deadline misses) are important. In the case of SRT systems, a few deadline misses are affordable but maximum and average latencies need to be managed.

Task response time (observed): Contrary to execution cost, the response time of a task includes the total time period from release time until the end of execution. This duration includes the waiting time for all dependencies to arrive, the execution cost x_i and the interference caused by task blocking. The *observed* response time of the task should not be confused with r_i, in Eq. (2.2) which is the analytical worst-case response time of a task. For example in Figure 3.4(b), $r_{I_0} = 6.5$, $r_{P_1} = 16.5$ and $r_{B_2} = 21.5$ units. The notation r_{I_0} denotes the response time for the I_0 frame decoding task. The end-to-end response time of I_0, which includes the flows emitted by I_0, is 13.5 from the dispatched time.

Flow response time (observed): This is the duration between flow transfer start and completion. It includes the blocking time due to the interference from direct/indirect flow interferers. In Figure 3.4(b), flow $I_0 \rightarrow P_1$ has a basic latency of 2.5 time units, but a response time of 3.5 time units.

Task turn-around time : This is essentially the duration $e_i^t - s_i^t$, which includes the execution time of the task and the interference from higher priority tasks but does not include the waiting time. As an example, consider task P_1 in Figure 3.4(b); its absolute turn-around time is 3 time units, from the point at which its dependent data had arrived, including its task interference.

Job response time (observed): The observed job response time is the measured time period between a job arriving into the system and all tasks in the job completing execution (denoted J_i^r). This is also referred to as the job end-to-end response time. The maximum J_i^r obtained by several experimental runs can be referred to as the *observed worst-case* job response time, which is most often lower to the analytical worst-case.

Analytical worst-case job response time: The analytical worst-case job response time is the worst-case end-to-end response time of a job that is calculated assuming the worst-case behaviour of tasks and flows in the job. This is essentially the WCRT of the critical-path of the job (denoted $WCRT(J_i^{CP})$). Both the $WCRT(J_i^{CP})$ and J_i^r would vary depending on the task allocation due to the interference patterns. Lower $WCRT(J_i^{CP})$ values are desirable to improve the schedulability of a system and to improve utilisation specific metrics. When a $WCRT(J_i^{CP})$ distribution for multiple jobs is being analysed, a smaller spread is a more desirable in terms of predictability, as it indicates less variability in $WCRT(J_i^{CP})$ between jobs.

Task/job lateness: The task lateness is referred to as the time period after the task has missed its deadline, calculated as per Eq. (3.8), where l_i is task lateness and d_i is the task deadline. The job lateness is the time period after which a job has missed its end-to-end deadline, calculated as per Eq. (3.9). This metric is primarily for SRT systems where a reasonable

number of job deadlines can be missed, however the *maximum* and *mean* lateness of the jobs need to be kept to a minimum. A *positive lateness* result indicates the task or job has missed its deadline. A *negative lateness* is essentially referred to as *slack*, which means the task or job completed its execution before its deadline. In this work, lower maximum and mean lateness values are desirable. The *cumulative job lateness* (C_L^{Jobs}) of an experimental run indicates the overall total lateness incurred by all late jobs in all video streams admitted into a system. C_L^{Jobs} is measured as given in Eq. (3.10); here, the jobs with negative lateness (i.e. slack) are omitted from the calculation.

$$\text{Task lateness: } l_i = e_i^t - (a_i^t - d_i) \tag{3.8}$$

$$\text{Job lateness: } L(J_i) = J_i^r - D_{e2e} \tag{3.9}$$

$$\text{Cummulative job lateness: } C_L^{Jobs} = \sum_{\forall v_i \in VS} \left[\sum_{\forall J_j \in v_i | L(J_j) > 0} L(J_j) \right] \tag{3.10}$$

Number of late video streams: At a coarser granularity, a video stream is considered *late* if one of more of its jobs incurred lateness. In the context of this work, the aim of a SRT resource management scheme is to reduce the number of late video streams.

Number of fully schedulable video streams: A video stream is considered *fully schedulable* if none of its admitted jobs are late (i.e did not miss any deadlines). A resource management scheme with a higher number of fully schedulable video schemes is considered more predictable than one which provides a lower number of fully schedulable video schemes, for the same workload.

Number of rejected video streams: A video stream decoding request can be *rejected* by the system to maintain the predictability level of existing/active video streams in the system. An admission controller could also reject a video stream if it determined the new stream cannot be serviced without missing any deadlines. Reducing the number of rejected video streams is desirable to improve performance/utilisation metrics.

Admission rate: This is the ratio between the number of admitted video streams (fully schedulable or otherwise) over the total video stream decoding requests received.

Many of the above predictability metrics relate to the *worst-case* observed timing characteristics of tasks/jobs. *Average-case* timing measurements can also give a notion of *performance* of a system. For example, a scheduling or task allocation technique which provides a lower mean job response time can be said to offer higher performance than a technique which shows a higher mean job response time. However, due to the unpredictable interference in the system a lower mean response time may not necessarily lead to a lower worst-case response time.

3.2.2 Utilisation-based performance metrics

In the context of this work, utilisation-based metrics are also used to quantify performance. Utilisation metrics are related to the platform and are used to measure how much of the system resources (e.g. PEs, NoC communication channels) are being used. Depending on the context and workload condition, evaluation may focus on different utilisation metrics. For example,

having a balanced utilisation distribution vs. using as few PEs as possible to meet required timing constraints. In all cases however, utilisation levels of the system is driven by admitted workload. In an *ideal* scenario, under high workloads conditions the system resources would be heavily utilised and there would be no videos rejected or late. Inefficient resource allocation policies could lead to PEs being idle, yet some jobs still missing their deadlines. Admitting video streams with extremely high computation/communication requirements in excess of what the system can offer, can lead to job lateness. This work defines certain utilisation metrics as follows:

PE busy time: This is the total time duration that a PE was busy (i.e. active) performing computation. The waiting time for data dependencies to arrive is excluded from this measurement. The *mean* (μ) PE busy time of the system is calculated as the average busy time of all PEs in the platform, as given in Eq. (3.11); also quantified as a percentage of the overall experiment run time, Eq. (3.12). As this metric is a notion of the utilisation of the PEs in the system, the terms *PE busy time* and *PE utilisation* is used interchangeably. In this work, guaranteeing workload worst-case timing requirements whilst maintaining high resource utilisation is preferred.

$$\mu \text{ (system PE busy time)} = \frac{\sum_{i=1}^{P} BusyTime(PE_i)}{P} \tag{3.11}$$

$$\text{Percentage } \mu \text{ (system PE busy time)} = \frac{\mu \text{ (system PE busy time)}}{experiment\ time} \times 100\% \tag{3.12}$$

PE busy time distribution: This is a distribution of busy times for each PE. This metric is used to identify load imbalance, which can introduce issues such as delayed response times and thermal hotspots in the system; specially under high workload scenarios. Under low workload conditions, system designers may decide to utilise as few PEs as possible, in order to save energy. Task clustering based mapping approaches may induce load imbalance but the communication contention in the system can be reduced, hence leading to reduced job response times. Lower standard deviation (σ) or lower variance (σ^2) of the distribution implies more uniform workload distribution across the PEs.

NoC busy time: Similar to the PE busy time metric, the NoC busy time is the total duration in which the NoC links were in use. This metric is calculated as per Eq. (3.13) and Eq. (3.14), where L denotes the number of links in the NoC. Higher NoC busy times can lead to idle PEs waiting for data to arrive before starting execution, thus implying an inefficient task allocation configuration. Increased NoC utilisation could also contribute to NoC congestion and therefore higher response times. This said, allocating all tasks to a single PE can diminish NoC usage, but will also disregard any opportunities for parallelism and can also increase task blocking behaviour.

$$\mu \text{ (system NoC busy time)} = \frac{\sum_{i=1}^{L} BusyTime(NoC_i^{link})}{L} \tag{3.13}$$

$$\text{Percentage } \mu \text{ (system NoC busy time)} = \frac{\mu \text{ (system NoC busy time)}}{experiment\ time} \times 100\% \tag{3.14}$$

3.2.3 Resource management energy-efficiency

The measurements below are used to represent implicit power savings that can be obtained by a specific resource management policy.

Total NoC communication cost: This is a summation of basic latencies of all the flows injected into the NoC by any of the PEs as per Eq. (3.15). NoC communication cost captures the number of flows, their payload volume and route length metrics. It is directly proportional to interconnect power dissipation, as given by NoC energy models in [174, 181]. As discussed in Section 2.3.3.4, several existing task mapping techniques explore task clustering to reduce NoC communication related power dissipation.

$$\text{NoC communication cost} = \sum_{\forall Msg_i \in all\ flows} C_i \tag{3.15}$$

PE busy time distribution : This metric represents the variation of the load on different PEs in the NoC. As discussed in Section 2.3.3, certain energy-aware resource management techniques (e.g. [146, 187]), exploit non-uniform workload distribution to put unused PEs into low-power states. On the other hand, work such as in [27,37,39] focus on uniform distribution of workload to make efficient use of resources, reduce thermal hot-spots and improve system reliability.

3.2.4 Resource management overhead

In this work, resource management typically encounters computation and communication overheads. The computation overheads primarily relates to the execution overhead related with making the task mapping decisions. The communication overheads relates to any use of the NoC interconnect to facilitate the resource management functions. The RM overheads can be used to compare the efficiency of one management technique against another.

Task mapping execution overhead: This is the *normalised* execution time of the task allocation procedure. Runtime task mappers need to be light-weight to not significantly affect the job response time. However, in this work the task mappers are evaluated on an abstract simulator, executed on a desktop system. The actual overhead of mapping on a real platform would be significantly different to the mapping execution time on the evaluation platform. Therefore, adding the mapping execution time into the simulation model will not result in meaningful results. For relative comparison purposes this metric is normalised. A distribution of mapping execution overheads can be obtained by measuring the time taken for mapping each individual job that comes into the system. The *maximum* mapping execution time is of particular importance as it indicates the observed worst-case behaviour of the mapper.

RM communication overhead: This is the NoC communication overhead required to perform the management/monitoring functionalities of the resource manager. As given in Eq. (3.16), this metric is calculated as the sum of the basic latency of all the management control flows (e.g. feedback messages, request/reply protocols etc.). Data communication flows are not taken into account in the calculation. Other properties of the control messages

such as their payloads, number of hops taken per flow (i.e. links used), total number of control message flows, all assist in quantifying the RM communication payload. The cumulative basic latency (Eq. (3.16)) encapsulates all the above factors. Similar to the NoC communication volume metric, a higher communication overhead implies higher NoC energy consumption.

$$\text{RM communication overhead} = \sum_{\forall Msg_i \in control\ flows} C_i \tag{3.16}$$

3.3 Simulation-based evaluation

The resource management and scheduling policies explored in this thesis are evaluated on a discrete-event system-level simulation platform, using an abstract workload model of multiple stream video decoding. As discussed in Section 2.2.5, compared to a cycle-accurate model or a hardware prototype, using an abstract simulation model can accelerate the design space exploration and enables a swift move from algorithm inception to performance evaluation.

There are two primary reasons for selecting simulation based evaluation in this thesis. Firstly, in the early stages of the design flow, exact performance values are less important, and the main goal of this work is to study *relative* performance benefits of different resource management techniques. Secondly, this thesis conducts experiments with a large magnitude of varying multi-stream video workloads, representing different communication/computation costs, arrival patterns and dependency structures etc. It would be highly infeasible to conduct experiments with a similar scale on a prototype system.

A key requirement of simulation-based models is that the characteristics and scale of real workloads and many-core platforms are closely represented. The models should have a reasonable level of abstraction in order to balance simulation time and still give reasonable insight into performance bottlenecks. To this extent, existing abstract video decoding workload models are initially used (as described in Section 3.1.1), and then in Chapter 7 the workload models are extended, to take into account more complex and modern video decoding workload characteristics.

The lightweight and accurate, high-level NoC simulation technique introduced in [106] is used in this work, to model the NoC flow traffic. This technique has similar principles to TLM models where the message flow contention is modelled at a higher abstraction level to avoid lengthy simulations. The entire interconnect is modelled as a single entity and the message flow interference patterns are modelled taking into account the basic latency, priority, shared links and release times of flows. Packets are modelled at the points where they enter/leave the NoC. The state of the NoC is maintained as a list of priority sorted flows. As time progresses, the algorithm activates/deactivates the flows based on their interference sets implemented via an event-driven interface. In terms of accuracy, this technique is over 70% accurate compared to a cycle-accurate model, even for large 10×10 NoC sizes, and can deliver 3-5 orders of magnitude speed-up over an equivalent cycle-accurate model.

It is interesting to investigate the communication and memory subsystem bottlenecks that arise due to higher contention scenarios when the NoC is heavily utilised. A high contention condition can be induced, either by increasing the amount of workload (i.e. increasing the amount of parallel video streams) or by decreasing the NoC bandwidth (e.g. decrease link width

and/or NoC frequency). However, evaluating larger workloads increase the number of simulation events, and in turn negatively affect the simulation speed. Therefore, this work uses the latter approach, where certain experiments assume the NoC communication bandwidth is a scarce resource, with a lower NoC operating frequency relative to the PE frequency. Higher amount of flow interference can also impact the NoC simulation speed, therefore experiments must be carefully designed to balance the simulation runtime, the level of workload and network utilisation evaluated.

3.4 Problem statement

Designing efficient mapping heuristics that take into account different forms of resource contention is a challenging task and an open research question. Inefficient mapping configurations can negatively affect performance and predictability in the system, especially when workloads are highly dynamic. This work focuses purely on runtime task mapping and remapping techniques which are specifically suited for real-time video decoding applications. In order to test the hypothesis defined in Section 1.3, the following problems need to be explored.

- The system must guarantee timing requirements of the video streams upon admission. This is the primary requirement for HRT streams. The dynamic nature of the workload makes providing hard timing guarantees a challenging task, and hence the admission controller needs to assume the worst-case timing behaviour of the workload. As discussed in Section 2.3.1.1, providing high predictability often leads to resource under-utilisation. Therefore, the first research question is, can the admission controller use the system status as a heuristic to offer SRT video streams a reasonable *balance* between *predictability* and resource *utilisation*?

- Targeting purely HRT video streams, can the *task mapper* exploit application and platform knowledge to improve the resource allocation policy in order to *increase* the system utilisation? These heuristics would need to perform equally well for large platform sizes and workloads with variations in communication/computation load. Many of the existing task mapping techniques in the literature (Section 2.3.2.1) follow a closed-loop resource management scheme, where different platform/application metrics are monitored at runtime, to dynamically improve the resource allocation process. Can an *open-loop* resource manager be designed to reduce the overhead introduced by monitoring the system?

- In order to service SRT and best-effort workloads, research work has recently focused on hierarchical and fully distributed resource management techniques (discussed in Sections 2.3.2.2). These offer scalable alternatives to centralised management but still have limitations such as complex protocols and communication overheads. Some of these techniques perform task migration, which re-allocates tasks to different PEs in order to improve performance metrics; however migration overheads are expensive and rely on well defined migration points (checkpoints). Can a low-overhead, fully distributed *remapping* technique be designed which does not perform migration? Can autonomous *bio-inspired* techniques be exploited to assist in the *distributed* resource re-allocation procedure?

- Certain constraints have been made in the application and platform models presented in Section 3.1. Assumptions such as fixed GoP structures may be unrealistic when encoding

efficiency is of primary concern (i.e saving bitrate). In addition, modern video codecs such as HEVC can offer built-in data-parallelism features that can be exploited to improve decoding latencies. The platform model as well can be refined to include memory read/write transactions within the application. The problem now is two-fold:

  - Can the application and platform models be adapted to include *memory* related traffic?
  - Can the *application model* in Section 3.1.1 be adapted to represent the communication and computation characteristics, of *HEVC encoded* video streams with content-adaptive complex coding techniques?

- The application model in Section 3.1.1, includes several assumptions regarding the workload. Lifting these assumptions increases the dynamic nature of the workload, and therefore mapping techniques originally designed for HRT workloads may prove inefficient. For future use-cases where SRT, HD/UHD video decoding is required on mobile devices, energy-efficient resource management techniques are required. Furthermore, memory traffic must also be managed appropriately to ensure they do not cause performance degradation. These issues open the following new mapping specific research questions which are explored in this thesis. Can efficient heuristic-based resource allocation techniques be designed, which jointly balance *task lateness* and *communication energy consumption*? Can application-specific heuristics perform better than existing generic allocation techniques? Can the contention caused by *memory* traffic be reduced to improve job lateness?

Chapter 4

Admission control strategies to balance predictability and utilisation

The motivation of the work in this chapter arises from the multiple video stream decoding use-cases presented in Section 1.1.1. Video decoding requests will be received at different points in time into the system at runtime. These videos have timing requirements which will need to be satisfied if admitted. *Buffering* is a common technique in video streaming applications to overcome issues with throughput. However, the *buffering delay* negatively affects QoE; furthermore, videos freezing/stalling mid-way during playback can impact the QoE factor further [64, 65]. Therefore, guaranteeing a reliable video streaming service at the *start* of the stream is required to improve user-experience for SRT video streams and a mandatory requirement for the processing of HRT videos. A predictable and efficient admission controller (AC) is therefore required to decide if the stream should be admitted/rejected. The objective of a predictable AC is to only admit a video stream if it can guarantee that the stream will not incur any lateness throughout the stream's lifetime; nor should the admission cause any lateness to the existing video streams already admitted. The challenge however, is to also maintain a low rejection rate in order to maximise resource utilisation. High level of predictability is required for HRT video streams while SRT videos can tolerate a reasonable degree of lateness.

As discussed in Section 2.3.1.3, many of the existing predictable admission controllers use either deterministic or probabilistic techniques. The probabilistic techniques rely on accurate models of the workload and cannot be used to guarantee hard timing requirements. The deterministic techniques that employ utilisation based tests which are sufficient but not safe. Meaning, they do not consider all timing behaviours of the tasks, omitting the calculation of blocking behaviours.

This chapter presents the first technical contribution of this thesis. Two admission control schemes are presented: a *deterministic AC (D-AC)* and a *heuristic-based AC (Heu-AC)*. The D-AC provides a strict timing guarantee by performing online end-to-end worst-case response time analysis (E2ERTA) of the tasks and flows in the video streams. The Heu-AC attempts to balance predictability, admission rates and efficient resource usage by using heuristics, related to the status of tasks currently active in the system. It has parameters that can be tuned to achieve the required level of trade-off. Both AC schemes are tailored to the multiple video decoding application model as described in Section 3.1.1 and the priority-preemptive NoC many-core platform model introduced in Section 3.1.2.

The remaining sections of this chapter will be as follows. Firstly, the *runtime task mapping*

and priority assignment process (MP&PR) is described. The RM first performs MP&PR which is required by the D-AC process to perform timing analysis on the tasks and flows of the video stream. Secondly, the proposed AC techniques will be described in detail and lastly, they are evaluated using system-level simulation.

4.1 Runtime task mapping and priority assignment

Runtime MP&PR is performed only once for a video stream at the admission of the first job J_0 of a new video stream VS_i. It is reminded that the GoP structure/pattern is fixed for each GoP in the video stream; therefore, there is a fixed task dependency structure for each job J_i of a video stream VS_i. All subsequent jobs of the same video stream follow the initial mapping and priority assignment made for J_0. The task mapping configuration between separate videos will differ as the mapping and priority assignment process is re-invoked for each new video decoding request. The granularity of the mapping configuration is hence at the video stream-level, not at the job-level. A *job-level mapper*, unlike a *video-level mapper* as described here, would invoke the mapping process for each job that arrives into the system and each subsequent job would hence have a different mapping configuration.

Task mapping directly affects the D-AC admission decision but does not influence the Heu-AC decision. Depending on the mapping configuration, the task and flow WCRT may change due to different levels of task/flow interference. Therefore, certain mapping configurations will result in the D-AC admitting a video stream whilst others might result in a rejection. For the purpose of the experiments in this chapter, a simple uniform mapping heuristic termed *least mapped (LM)* is used to map the tasks to the PEs. The LM heuristic is given in Algorithm 4.1; each task in the first job of the video stream is mapped to the PE with the least number of tasks in its task queue. Using a simple mapping heuristic such as LM, allows us to perform a fair evaluation of the basic performance of the D-AC against the Heu-AC, which does not rely on task mapping. Chapter 5 will then show that, more efficient and intelligent mapping schemes can be used to improve the performance of the D-AC.

Tasks are assigned unique priorities according to its resolution as described in Section 3.1.1.2. Similar to task mapping, priorities are assigned to tasks of J_0 of a new video stream, and tasks of all subsequent jobs are assigned the same priorities.

4.1.1 Runtime task mapping table

The RM 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: $\{f_i, f_{ix}\}$
- Task mapping: indicates which PE a specific task τ_i is mapped to $(\tau_i \rightarrow PE_i)$.

Once the tasks of J_0 have been mapped and priorities assigned, their details are saved onto the TMT and used during the E2ERTA analysis performed by the D-AC. If the admission decision is to *reject* the stream, the task entries of the new video stream's J_0 are removed from the TMT. Furthermore, when the video stream has stopped/finished, the task entries related to the stream (VS_i) are removed.

Algorithm 4.1: Least-mapped task mapping heuristic (LM)

Input : TMT - runtime task mapping table,
J_0 - first job of the new video stream (VS_i),
PE_list - PEs and their task queue (TQ) occupancy
Output: TMT with new task mapping

```
/* Map each task in J0 of new video VSi to the PE with the least tasks in its TQ */
```

1 **forall** $\tau_i \in J_0$ **do**
2 Find PE_i from PE_list with the least number of tasks in its TQ
3 Map task τ_i onto PE PE_i
4 Update $PE\_list(\tau_i, PE_i)$ (increment TQ occupancy)
5 Update $TMT(\tau_i, PE_i)$
6 **end**

4.2 Admission control tests

4.2.1 Deterministic admission controller

The D-AC process, is illustrated in Figure 4.1. This process is invoked each time a new video stream decoding request is received by the system. The mapping configuration generated during the MP&PR stage is saved to the TMT and used to perform the E2ERTA of the video streams. This is to determine if by admitting the new video stream, any of the WCRT of new or existing/active video streams would miss their end-to-end deadline. Algorithm 4.2 shows the steps involved in D-AC decision process. Firstly, the task mapping details are added temporarily to the TMT (line 2). If the video is rejected, then the entries are removed (line 24). After the task mapping, the flows (and their real-time properties) resulting from the mapping can be generated (line 3). If two communicating tasks are mapped onto the same PE, then the respective flow between those two tasks are omitted in the response time analysis. Similarly, if multiple children of the same task is mapped onto the same PE, redundant flows are also omitted.

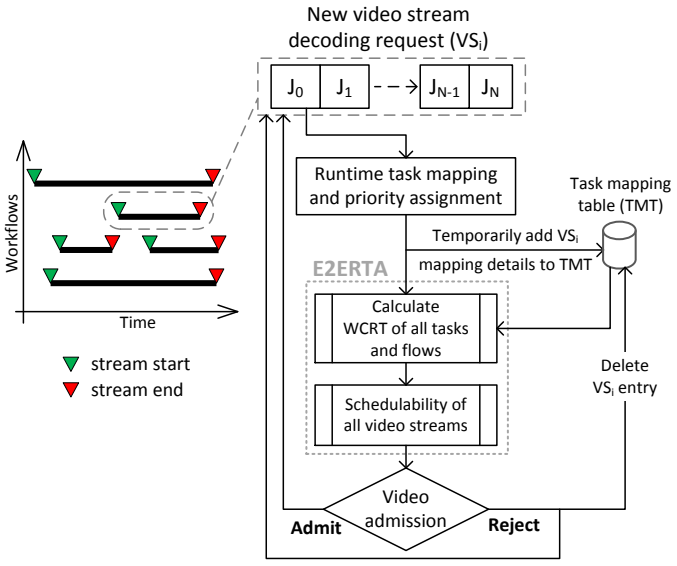


Figure 4.1: Deterministic admission control (D-AC) process

With the task and flow 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 4.2). Eq. (2.2) and Eq. (2.5) can be used to find the WCRT of tasks and flows. The analysis takes into account the blocking/interference factors due to higher priority tasks/flows. However, as the precedence

Algorithm 4.2: Deterministic admission control pseudo-code

Input : TMT : Runtime task mapping table;
J_0 - first job of the new video stream (VS_i),
PE_list - PEs and their task queue (TQ) occupancy
Output: $ac\_decision$ admission control decision (ADMIT/REJECT)

```
   /* Perform mapping and derive resulting flow set                          */
1  Perform MP&PR according to Section 4.1
2  Temporarily insert VS_i task mapping details to TMT
3  FT_temp : derive all valid periodic flows in system (for the mapping configuration)
   /* Calculate WCRT of each task and flow in TMT                            */
4  forall τ_i ∈ TMT do
5  |  Find interference set hp(τ_i) (exclude non-interferers ni(τ_i))
6  |  Calculate task WCRT - Eq. (2.2), save value in TMT
7  end
8  forall Msg_i ∈ FT_temp do
9  |  Find interference sets S_id, S_ii (exclude non-interferers ni(Msg_i))
10 |  Calculate task & flow WCRT - Eq. (2.5), save value in FT_temp ;
11 end
   /* Calculate WCRT(J_i^CP) of all video streams.                           */
12 forall VS_k ∈ TMT do
13 |  vs_p : initialise structure for all simple paths of VS_k job
14 |  job(VS_k) : J_0 of video stream VS_k
   |  /* Calculate WCRT of each path/task-chain in job                       */
15 |  forall path_i ∈ job(VS_k) do
16 |  |  Calculate WCRT of path_i according to Eq (4.3)
17 |  |  Save WCRT(path_i) to vs_p
18 |  end
19 |  Critical path of job(VS_k) : J_i^CP = max{vs_p}
   |  /* Check video stream schedulability                                   */
20 |  if WCRT(J_i^CP) ≤ D_e2e then
21 |  |  ac_decision = ADMIT
22 |  else
23 |  |  ac_decision = REJECT
24 |  |  Remove VS_i details from TMT
25 |  |  Return ac_decision
26 |  end
27 end
28 Return ac_decision;
```

constraints of each job in the application model is known a priori, certain exclusions can be made to the task/flow interference sets as shown in Eq. (4.1) and Eq. (4.2). The process in which the non-interferers are derived is presented in Section 4.2.1.1. The calculated WCRT of tasks and flows are then saved in TMT to be used later to calculate the job WCRT, of all video streams.

$$\widehat{r}_i^{n+1} = c_i + \sum_{\forall \tau_j \in \{hp(\tau_i) \setminus ni(\tau_i)\}} \left\lceil \frac{r_i^n}{t_j} \right\rceil c_j \tag{4.1}$$

$$\widehat{R}_i^{n+1} = C_i + \sum_{\forall Msg_j \in \{S_{id} \setminus ni(Msg_i)\}} \left\lceil \frac{R_i^n + r_j + J_j^I}{T_j} \right\rceil C_j \tag{4.2}$$

In lines 12-27 of Algorithm 4.2, the WCRT of the *critical path* of the job $WCRT(J_i^{CP})$ is calculated. Due to the fixed job structure there are fixed number of *simple paths* (also referred to as *task chains*) 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 16) as per Eq (4.3). Note that the WCRT of the source task is included in R_i as release-jitter r_j, hence only the flow WCRT needs to be accumulated. The WCRT of the leaf node (tasks without any child tasks) task (τ_{-1}) denoted $r_{\tau_{-i}}$ is added to the summation. The job critical path J_i^{CP} is the path with the maximum

accumulated cost (line 19).

$$WCRT(path_i \in J_i) = \left\lceil \sum_{Msg_q \in path_i} R_q \right\rceil + r_{(\tau_{-1})} \tag{4.3}$$

A video stream is granted admission, only if the expression given in Eq. (4.4) is true for the new and all active video streams in the system (lines 18-24 in Algorithm 4.2); this guarantees that the worst-case timing requirements of all existing and new video streams will be successfully met.

$$\text{Admit new } VS_i \leftarrow \text{ if } \forall J_i \in TMT \mid WCRT(J_i^{CP}) \leq D_{e2e} \tag{4.4}$$

4.2.1.1 Exclusion of non-interferers

Unlike in [80, 128], this work takes into account precedence constraints when calculating the task and flow interference. In this analysis, it is assumed there is no overlap between consecutive jobs within the same video stream. Hence, when deriving the $hp(\tau_i)$ of a task, the dependent/successor tasks can be excluded. For example, in the case where both frame decoding tasks B_2 and P_1 are mapped on the same PE, P_1 would not interfere B_2 and vice versa due to the dependency between P_1 and B_2. 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. Excluding non-interferers will make the WCRT analysis tighter but still safe. As the application dependency pattern is known a priori this method is straightforward and much simpler than using an offset-based analysis such as in [129].

As explained in Section 3.1.1, a TG has multiple simple paths. For example the simple path ($P_1 \Rightarrow B_9$) consists of the frame decoding tasks (vertices) P_1, P_4, P_7, B_9 and the message flows (edges) between them. In DAG structures, two distinct simple path types are defined - to and from a node in the TG, termed *ancestral* and *descendant* simple paths [198]. The *ancestral simple path* (expressed as ($\tau_0 \Rightarrow \tau_i$)) consists of the 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 3.2), 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. (4.5). Similarly, flows in ancestral and descendant simple paths will not interfere with the target flow as given by Eq. (4.6). Here, the ($Msg_i : \tau_s \rightarrow \tau_d$) component denotes the target flow Msg_i and its source and destination tasks τ_s and τ_d respectively.

$$ni(\tau_i) = \tau_k \in \{\tau_0 \Rightarrow \tau_i \cup \tau_i \Rightarrow \tau_{-1}\} \tag{4.5}$$

$$ni(Msg_i : \tau_s \rightarrow \tau_d) = Msg_k \in \{\tau_0 \Rightarrow \tau_s \cup \tau_d \Rightarrow \tau_{-1}\} \tag{4.6}$$

4.2.1.2 Conservative E2ERTA analysis

For both tasks and flows, the E2ERTA analysis considers the *maximum interference* to derive an upper bound for their WCRT (i.e r_i and R_i). However, in reality this may not occur and hence makes the analysis safe but conservative. In other words, if a traffic flow or task passes the schedulability test, it will meet its deadline, but if it fails, it may not necessarily miss its deadline.

Furthermore, the WCET of the tasks are considered in the WCRT analysis. If the execution time distribution of the frame decoding tasks have long tails or is skewed (such as in the decoding time model - Figure 3.3), the worst-case execution would definitely be a rare occurrence. Both these factors makes the E2ERTA analysis find WCRT upper bounds which may never be seen in practice/simulation. Hence, the schedulability tests are safe and enables the D-AC to provide hard real-time guarantees, but negatively affect its admission rate and may lead to low resource utilisation.

4.2.1.3 Reducing the overhead of D-AC

As defined in Section 3.2.4, a resource manager can have associated computation and communication overheads. The D-AC does not incur communication overheads but has a computation overhead in terms of the processing time taken to arrive at the admission decision. However, as the admission controller is invoked only at the start of a video stream, a reasonable admission control delay is acceptable. Subjective tests in [65] show that up to 30 seconds of *initial delay* in video playback can be tolerated, before a significant loss in QoE can be observed.

Under certain high workload scenarios, the recurrence relationship calculations (Eq. (4.1) and Eq. (4.2)) to derive the worst-case response time of tasks/flows may take a significant amount of processing time to complete, due to heavy interference. Hence, the D-AC would take longer to arrive at a decision. To avoid this issue, a *time-out* is assigned to the calculation, where an intermediary result is returned when the time-out expires. The time-out needs to be set to a reasonably high value such that a sufficiently accurate worst-case response time can be calculated within the time-limit but at the same time have a relatively fast computation time. In all experiments in this section the time-out of the D-AC is set to 60 seconds.

In order to further manage the overhead of the D-AC, the D_{e2e} timing constraint is checked at different intermediate levels of the E2ERTA calculation. Each time the WCRT of a task/flow is calculated the $\widehat{r}_i \leq D_{e2e}$ and $\widehat{R}_i \leq D_{e2e}$ are checked. Also, for each path in the TG the condition $WCRT(path \in TG) \leq D_{e2e}$ is checked (lines 12-27 of Algorithm 4.2) and if not satisfied the admission is rejected immediately. These intermediate assertions enable the D-AC to arrive at rejection decisions quickly. For example, it is not necessary to calculate the WCRT of job paths, when a single flow WCRT alone has exceeded the D_{e2e}.

Certain existing work in [199], propose techniques to approximate the iterative E2ERTA calculations, thereby lowering the computation complexity. Furthermore, there also exists hardware based implementations of the E2ERTA, which offer up to 3 orders of magnitude faster calculations compared to a software based calculation [200]. These approaches are complementary to the overhead reductions discussed in this section, and can be used in conjunction with the AC and task allocation techniques discussed in this work.

4.2.2 Heuristic-based admission controler

The Heu-AC admission test introduced in this section is essentially a best effort test, which considers the lateness of the tasks admitted. The system is considered to be in an overload state if two conditions are true: (1) if any of the tasks in the *global input buffer (G_IBF)* or *local PE task queues (TQs)* are late; (2) if the G_IBF or the TQs are full. A new video stream decoding request is only admitted if and only if the system is not in an overload state. Therefore, unlike the D-AC, the Heu-AC does not rely on task mapping to perform the admission control process. Similar to D-AC, the Heu-AC process is invoked when a new video stream decoding request is made.

A task can wait in the G_IBF until its parent tasks have been completed. There exists certain scenarios, particularly during heavy workload conditions where the G_IBF does not have any space to accommodate new tasks. If there is no available space in the G_IBF then the admission of new video streams should be avoided and if any jobs are due to be dispatched in that time instant, they will be dropped. Tasks in the local task queues (TQs) can be blocked and waiting for the processor resources. During their waiting period, tasks will incur lateness. A task is considered as late if it finishes its execution after its absolute deadline. However, in our application model the individual task deadlines (d_i) are not known; only the end-to-end job deadline D_{e2e} is known. If the individual task deadlines are assumed to be equal to the end-to-end deadline (i.e. $d_i = D_{e2e}$) then the lateness calculation may be too optimistic, as tasks that are higher up in the task-graph will have longer slack times. To alleviate this issue, the individual absolute task deadline is estimated as a fraction of D_{e2e} as specified in Eq. (4.7) and Eq. (4.8). $l_i(InputBuffer)$ is the estimated *instantaneous lateness* of a task in the *global input buffers* and $l_i(TaskQueue)$ is the estimated instantaneous lateness of a task in the local PE *task queues*. t_c denotes the current time and a_i denotes the arrival time of the task. IBL_α and TQL_α are the ratios used to calculate the relative deadline of the tasks in the global input buffers and task queues respectively.

$$l_i(InputBuffer) = (t_c - a_i^t) - (D_{e2e} * IBL_\alpha), \text{ where } (0 \leq IBL\alpha \leq 1) \tag{4.7}$$

$$l_i(TaskQueue) = (t_c - a_i^t) - (D_{e2e} * TQL_\alpha), \text{ where } (0 \leq TQL_\alpha \leq 1) \tag{4.8}$$

Algorithm 4.3: Heuristic admission control pseudo-code

Input : PE_{all}^{TQ} : Local task queues of all PEs,
G_IB : Global input buffer
Output: *ac_decision* admission control decision (ADMIT/REJECT)

```
1  ac_decision = ADMIT (initialise)
   /* check if global input buffer or task queues are full */
2  if (G_IB||PE_all^TQ) are Full then
3  |  Return ac_decision = REJECT
4  end
   /* check global task queue for task lateness */
5  forall τ_i ∈ G_IB do
6  |  Calculate task τ_i lateness l_i as per Eq. (4.7)
7  |  if l_i > 0 then
8  |  |  Return ac_decision = REJECT
9  |  end
10 end
   /* check local PE task queues for task lateness */
11 forall PE_i^TQ ∈ PE_all^TQ do
12 |  forall τ_i ∈ PE_i^TQ do
13 |  |  Calculate task τ_i lateness l_i as per Eq. (4.8)
14 |  |  if l_i > 0 then
15 |  |  |  Return ac_decision = REJECT
16 |  |  end
17 |  end
18 end
19 Return ac_decision;
```

Every task in global input buffers and the task queues incurs lateness as time progresses. Negative lateness values are considered as slack and are acceptable. Algorithm 4.3 shows the Heu-AC admission decision making process. The algorithm takes the content on the buffer and queues as input parameters. It first checks the lateness of all tasks in the global input buffer

(lines 2-6) and then proceeds to check the lateness of all tasks in all PE TQs (lines 7-14). If any of the tasks are late, the new stream request is rejected else the stream is admitted.

The execution overhead of Heu-AC compared to D-AC, is very small. Even for large workloads the Heu-AC tests arrive at an AC decision within a few seconds, on a standard PC machine (Intel i3 2GHz, 8GB RAM). However, as it can be seen from Algorithm 4.3, the execution time of Heu-AC test is directly dependent on the number of PEs in the system and the number of tasks in the local and global task queues.

4.3 Evaluation

This evaluation section investigates the effect that the different AC tests (heuristic and deterministic) have on predictability and utilisation metrics. The experimental conditions are first defined followed by a discussion of the results. The objectives of this evaluation is two fold:

- To test if the D-AC will give maximum predictability guarantees to the video streams with no admitted videos being late.
- To test if the Hue-AC parameters can be tuned to offer a trade-off between predictability and utilisation.

4.3.1 Experimental design

Platform specification: a 3×3 (i.e. 9 PEs) NoC platform is assumed with the platform characteristics as defined in Section 3.1.2. The PEs are assumed to have an operating frequency of 200MHz and the PE local task queues, dependency buffers and the global input buffer sizes are set to accommodate a maximum of 10 tasks each. Message flow header routing cost is assumed to be 7 clock-cycles, NoC frequency is set at 1GHz and the link width is set to 16 bytes.

Independent variables: The two proposed admission controllers (D-AC and Heu-AC) are evaluated under equal workloads. Different variation of IBL_α and TQL_α values of the Heu-AC are explored. In addition to the proposed AC techniques two more admission controllers are evaluated as baselines:

- No admission control *(No-AC)*: the situation where no admission test is used, however jobs can be dropped due to insufficient buffer space.
- Deadline equal-flexibility AC *(DEQF-AC)*: a heuristic-based admission controller that uses the individual deadline assignment scheme as proposed by Kao and Garcia-Molina [137] (discussed in Section 2.3.1.3). The functionality of DEQF-AC is similar to Heu-AC, however the calculation of the task deadlines are performed using Eq. (2.6).

Workloads: The multi-video stream decoding workload is simulated according to the application model defined in Section 3.1.1. The load profile was characterised as per Sections 3.1.1.3 and 3.1.1.4. The experiments were conducted on a *low* workload of 8 workflows and a *high* workload of 16 workflows. Each workflow is randomly allocated 6/7 video streams with 7/8 jobs per video stream. Video stream resolutions were chosen randomly from common high to low resolutions (e.g. high: 720×576, low: 320×240). The temporal workload parameters are chosen to

ensure a densely packed workflow. The job dispatch rate parameters were set to $J^{rate}_{min} = 1.0$ and $J^{rate}_{min} = 1.5$. The inter-video dispatch rate parameters are set to $VS^{rate}_{min} = 0.48$ and $VS^{rate}_{max} = 0.76$.

Response variables: The predictability and utilisation of the admission controllers were measured in terms of video stream admission rates (i.e. number of fully schedulable, late and rejected streams), job lateness and mean system PE busy time %. Detail description of the metrics can be found in Section 3.2.

Evaluation platform: An abstract system level simulation as specified in Section 3.3 is used to perform the evaluation. Figure 4.2 shows the interaction and execution sequence between the task dispatcher, resource manager, PEs and the NoC model of the simulator. In this closed-loop system (Section 3.1.2.1), the RM releases the child tasks only after a task complete notification is received from the PEs. Simulation experiments were carried out for 35 unique random seeds, which produced a range of workloads within the aforementioned parameter settings. Each seeded simulation run will provide a workload with variations in video resolutions, task execution costs and inter-arrival times.

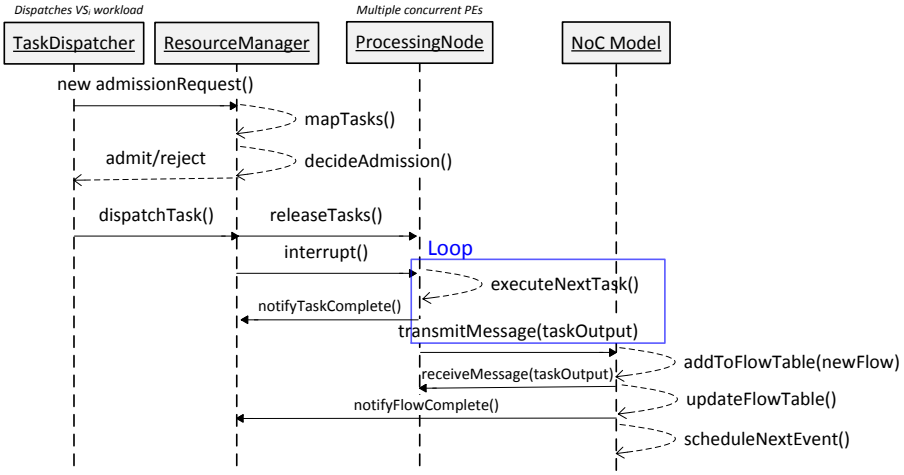
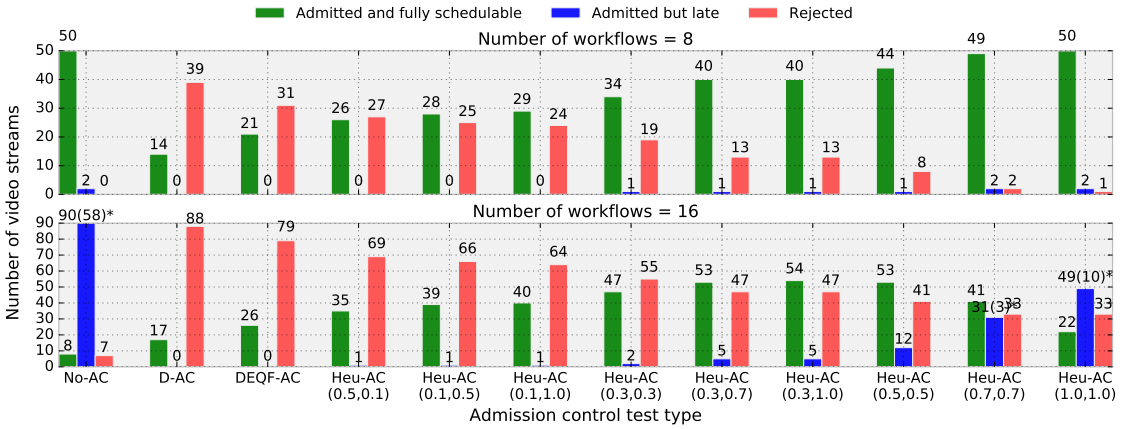


Figure 4.2: High-level execution sequence of abstract simulator

4.3.2 Results discussion

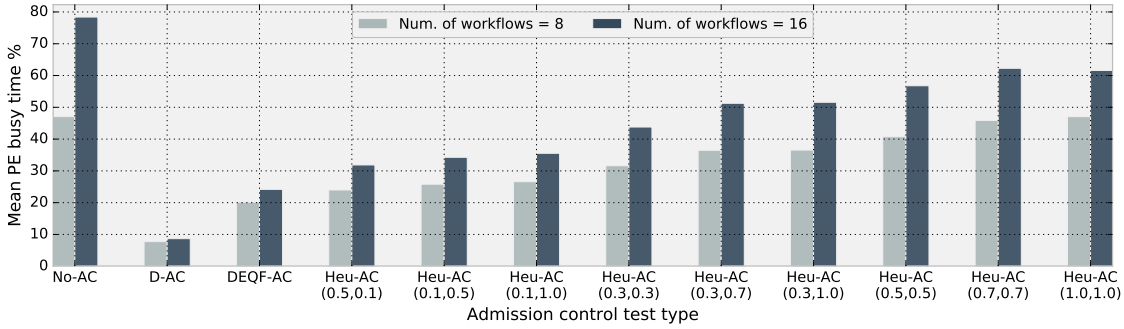
Evaluation results shown in Figure 4.3 illustrate the trade-off between predictability and utilisation made by the different ACs. The x-axis for all sub plots in Figure 4.3 represents the admission controllers under evaluation. The ratios IBL_α and TQL_α of the Heu-AC are given inside brackets. For example *Heu(0.3, 0.7)* denotes the heuristic based admission control test results with $IBL_\alpha = 0.3$ and $TQL_\alpha = 0.7$. Only a subset of $\{IBL_\alpha, TQL_\alpha\}$ combinations are shown in Figure 4.3 to aid the visualisation of the results. Full set of results can be found in Appendix A. Figure 4.3a and Figure 4.3c represent the mean values of 35 seeded simulation runs. The distribution of job lateness values shown in Figure 4.3b represents positive lateness values across all jobs in all seeded simulation runs. The right-y-axis of Figure 4.3b represents the maximum job lateness across all seeds

For all evaluated ACs, the admission rejections increase as the number of workflows is increased (Figure 4.3a). It can also be seen in Figure 4.3c that as more video streams are admitted



(a) Predictability guarantee (video stream admission) results for different admission control tests. Values in brackets (*) denotes number of jobs dropped

(b) Job lateness distribution for different admission control tests



(c) Percentage system busy time, for different admission control tests.

Figure 4.3: Admission control evaluation results - predictability vs. utilisation)

into the system (regardless of whether they are late or schedulable), the PE busy time will increase. For the tested workload the peak system utilisation is $\approx 78\%$ after which the buffers begin to overflow, as shown in the No-AC test case. Both the quantity of admitted video streams and their respective computation complexity correlate directly to a more busy system. Admitting a few high resolution video streams may cause the system to be more busy than when admitting relatively higher amount of low resolution streams. The variation in the job lateness distribution (specially in the No-AC case) is due to the differences in computation cost requirements between streams of different resolutions. Higher resolution streams are more likely to incur positive lateness due to higher execution and communication costs.

The results indicate, when the D-AC is used in both low/high load conditions, none of the

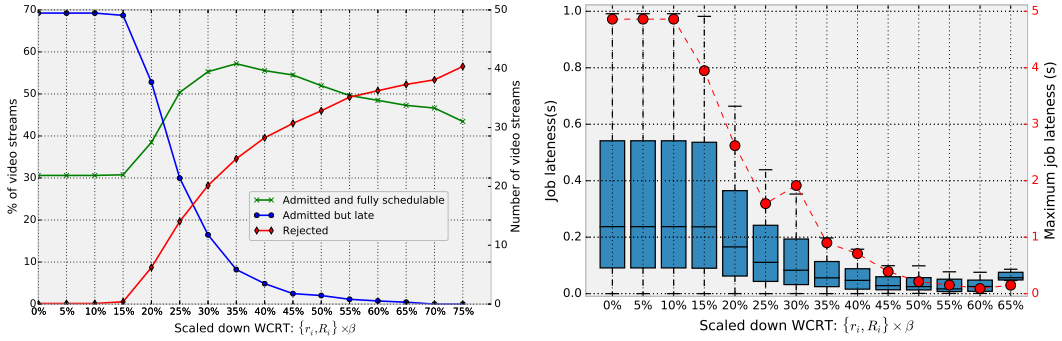
admitted video streams incur any lateness or dropped jobs. However, D-AC has a high rejection rate due to its conservative timing analysis procedure Section 4.2.1.2. Due to the high rejection rate, D-AC shows a very low mean PE busy time (i.e mean PE utilisation). Contrary to the D-AC, the No-AC admits all incoming video streams provided the global input buffer is not full. In the low workload case, No-AC only has a few late streams and very high acceptance rate, but in the high workload scenario it performs poorly with a large amount of late streams, dropped jobs and large job lateness values. The No-AC test therefore provides no timing guarantee to the incoming video streams but has an advantage over other ACs in the form of higher PE utilisation.

The DEQF-AC scheme shows similar service guarantees as the D-AC test, where no admitted streams incur lateness but with a lower rejection rate than D-AC even at high workloads. DEQF-AC also indicates a higher PE busy time than D-AC but lower than the Heu-ACs. The Heu-AC shows a range of different results depending on the IBL_α and TQL_α ratios used. Figure 4.3a illustrates that smaller ratios lie closer to D-AC results while higher ratios show results similar to the No-AC test but with lower rejection rates. The maximum job lateness increase significantly when higher values of IBL_α and TQL_α are used (e.g. Heu-AC(0.5,0.5), Heu-AC(0.7,0.7) and Heu-AC(1.0,1.0)). The PE utilisation results also increase for higher values of the Heu-AC ratio parameters for both low/high workloads. Low Heu-AC ratios on the other hand, show high number of rejections with only very few streams late but at the cost of low PE busy times. Compared to DEQF-AC, the Heu-AC(0.5,0.1) offers about 8% more utilisation (high workload scenario), an average of 9 more fully schedulable streams at the cost of only 1 stream being late. Heu-AC(1.0,1.0) shows more late streams than compared with lower ratios (e.g. Heu-AC(0.7,0.7)), however the PE busy time is comparable. This is mainly due to two aspects - increased NoC congestion (therefore, idle PEs) due to higher admissions and more dropped jobs at higher workloads.

4.3.2.1 Exploring D-AC timing guarantee reduction

Secondary experiments were carried out to investigate the behaviour of the D-AC with respect to relaxing the strictness of the E2ERTA calculations. The effect of scaling down the result of the WCRT calculation in Eq. (4.2) and Eq. (4.1), on reducing the effect of the conservative assumptions of the D-AC is first explored. As shown in Figure 4.4 the task and flow WCRT (r_i and R_i) were reduced by a certain factor β where $0\% \leq \beta \leq 100\%$. In Figure 4.4a, the left y-axis displays the number of streams schedulable, rejected and late as a percentage ratio of the total number of video streams; respectively, the right y-axis displays the number of streams as an absolute value. The ratio of late streams decrease exponentially between $\beta = 15\% - 55\%$ in Figure 4.4a, after which the amount of late streams are negligible. For $\beta = 70\%$ and above, the D-AC shows zero late streams but rejection levels increase. Similarly, wide variations in job lateness values are seen in Figure 4.4b for lower β values as more streams are admitted, but for $\beta > 50\%$ the lateness variation is significantly less.

At this point, it is useful to investigate the predictability of the D-AC if only the task WCRT calculation is carried out in the E2ERTA (i.e. flow WCRT calculation is omitted). However, the benefit of taking into account *both* the task and flow response time in the E2ERTA calculation is only apparent, when the flow basic latency and the task WCET is approximately in the same range and the NoC is more utilised. In order to induce such an experimental condition, the NoC frequency was set to 10MHz, PE frequency scaled up and higher resolution videos were included (e.g. 1280x720) to increase the flow payload size. In this experiment, lines 8-11 of Algorithm 4.2 related to calculating R_i is omitted from the D-AC algorithm. Also, the R_i summation when



(a) **(b)**

Figure 4.4: Effect of scaling down the task and flow $WCRT(r_i, R_i)$: (a) Admission rates; (b)Job lateness

calculating $WCRT(path_i)$ is replaced with only taking into account the WCRT of tasks r_i and not flows.

The group of bar plots labelled *D-AC(tasks only)* in Figure 4.5 shows the admission rates when only the task WCRT based timing analysis was used (WCRT of flows omitted) in the D-AC decision. D-AC(tasks only) shows a higher number of admitted and fully schedulable streams as well as a lower rejection rate than the original D-AC. However, as message flow timing properties was not accounted for, the D-AC(task only) admission controller shows a few streams missed their deadlines. The results in Figure 4.5 also justify that it is crucial to include *both* the task and flow WCRT in E2ERTA calculation and D-AC decision making process, in order to guarantee *hard* timing requirements and avoid stream deadline misses.

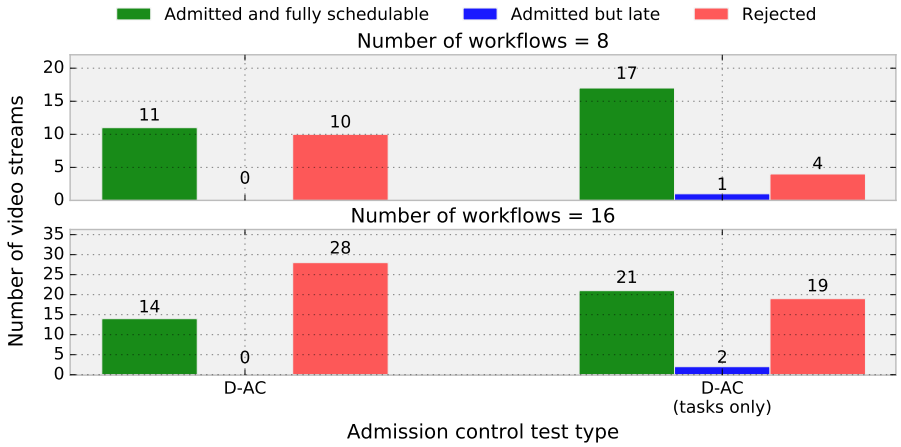


Figure 4.5: Admission rates of D-AC(original) vs. D-AC(tasks only)

4.3.2.2 Results summary

The results confirm that D-AC offer maximum predictable timing guarantees without any disruption to admitted streams. However, this comes at a price of heavily under-utilising the system. Experiments in Section 4.3.2.1 explored variations to the D-AC's E2ERTA calculation to improve the admission rates with low increase to job lateness levels. However, these adaptations of D-AC make the D-AC *unsafe* and unsuitable for HRT video decoding. The No-AC tests increased the system utilisation significantly but at the cost of offering no service guarantees to the user. Heuristic based tests do not attempt to find optimality in both objectives but the IBL_α

and TQL_α ratios can be tuned to offer required levels of predictability and utilisation.

The Heu-AC ratios, IBL_α and TQL_α can be categorised into different ranges namely, *High*: $(1.0 \leq \alpha \leq 0.7)$, *Medium*: $(0.6 \leq \alpha \leq 0.4)$ and *Low*: $(0.5 \leq \alpha \leq 0.1)$. Looking at the range of Heu-AC results given in Appendix A, a general guideline can be given when choosing appropriate values for these ratios. A low value of IBL_α combined with high values of TQL_α provides the best trade-off between utilisation and predictability.

4.4 Summary and novel contributions

To summarise, this chapter presented two novel admission controllers for multi-video stream decoding applications:

- **A deterministic admission controller:** This contribution refers to the deterministic admission controller specified in the first thesis hypothesis in Section 1.3, that is used to provide *hard* timing guarantees for video streams encoded using classical codecs. The D-AC performs end-to-end schedulability analysis of video streams, facilitated by dynamic task mapping to arrive at an admission decision. The original E2ERTA analysis in [80, 128] was adapted to suit the multi-video stream decoding application model. Schedulability is tested at the job-level and non-interferers are excluded to make the analysis tighter.

- **A heuristic-based admission controller:** This contribution is aimed at improving the predictability of *soft real-time* video streams, as specified in the first thesis hypothesis in Section 1.3. The Heu-AC attempts to balance predictability and utilisation of the system by using simple heuristics, unlike the D-AC which performs video stream timing analysis. The Heu-AC queries the lateness of the tasks the global input buffer and the local task queues of the platform to determine the admission decision. Heu-AC admits new video streams only if no other tasks in the system are late and if there are sufficient buffer capacity. The individual task lateness was derived as ratios of the end-to-end job deadline.

The objective of the D-AC was to provide a high degree of predictability which was confirmed in the evaluation section. The D-AC can be used in HRT video stream decoding where missing video stream deadlines are not acceptable. However, the evaluation showed that due to its conservative end-to-end response time analysis, the D-AC severely under-utilises the PEs in the platform. The Heu-AC approach offers a range of utilisation and predictability values such that, the systems designer can choose the IBL_α and TQL_α lateness ratio parameters, depending on the requirements of the video streaming application and/or the load profile. For example higher ratios can be used if higher system utilisation is required or lower ratios can be used when predictable services are required. Therefore, the Heu-AC can be used when soft-timing guarantees of video stream decoding is required. The No-AC baseline test used as a baseline AC in the evaluation section offers best-effort/high performance where missing deadlines/lateness is not a concern.

The low-utilisation issue of the D-AC motivates the research presented in the next chapter. Efficient runtime task mapping schemes are explored to improve the admission rate and resource utilisation of the D-AC in order to efficiently decode HRT video streams.

Chapter 5

Dynamic task mapping for hard real-time video streams

The work presented in this chapter is motivated by the need for efficient resource allocation techniques for data-parallel multiple-stream video decoding with strict hard real time constraints. Section 1.1.1 discussed several use-cases of HRT video decoding. Specially in cases such as in [19] the system should provide deterministic timing guarantees upon admission as well as make efficient use of resources to manage several tens/hundreds of streams simultaneously.

The previous chapter introduced a deterministic admission control technique (D-AC) which can be used to service HRT video streams. However, the evaluation of the D-AC showed that it provides very low admission rates and as a consequence under-utilises the system resources. Runtime task mapping and priority assignment were introduced in the previous chapter (Section 4.1) which are essential to the D-AC process. It is evident that different task to PE mapping configurations produce different task and flow contention patterns in the NoC, leading to timing constraints being successfully met or violated. The work in this chapter is based on the hypothesis that with efficient runtime task to PE mapping approaches, admission rates and system utilisation of the D-AC 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.

In this chapter, two novel heuristic-based runtime task mapping techniques are introduced, that exploit the application characteristics and blocking behaviour, in order to minimise the video stream job WCRT. Lowering the job WCRT will increase the D-AC admission rate, leading to better utilised systems; whilst still maintaining a high level of predictability. To evaluate the dynamic mappers, a baseline search-based, static, HRT task mapping technique is used with a novel points-based fitness function that considers video stream schedulability. The proposed mapping techniques are also evaluated in terms of their scalability and variations in the workload CCR.

The structure of this chapter is as follows: Section 5.1 describes certain refinements to the system model that was presented in Chapter 3. In Section 5.2 a blocking-aware heuristic is introduced which is used by both dynamic mappers presented in this chapter. The first runtime mapper is a general purpose technique and is described in Section 5.3 followed by second and application-specific technique in Section 5.4. The baseline static mapping technique is then introduced in Section 5.5 and an evaluation of the presented task mapping techniques is carried out in Section 5.6.

5.1 System model

The application and platform model used in this chapter follows the primary system model description given in Section 3.1. The following extensions were made to bring the model closer to a realistic video stream decoding system and/or make the evaluation of the task mappers simpler.

5.1.1 Application model refinements

The following three application model refinements are introduced:

- Integrating memory transactions into the task model.
- Video-centric task priority assignment.

5.1.1.1 Memory transactions

As discussed in Section 2.1.3. real video decoding workloads are both computation and data intensive and perform a significant amount of communication with main memory. Even though the amount of shared memory usage is considerably reduced in a distributed memory model as presented in Section 3.1, there still exists certain shared memory transactions that cannot be avoided. At the very least, each decoded frame requires writing out to a frame buffer in main memory to drive a video display. The application model introduced in Section 3.1.1 is therefore adapted to introduce memory read/write transactions in addition to the data dependency communications, as shown in Figure 5.1.

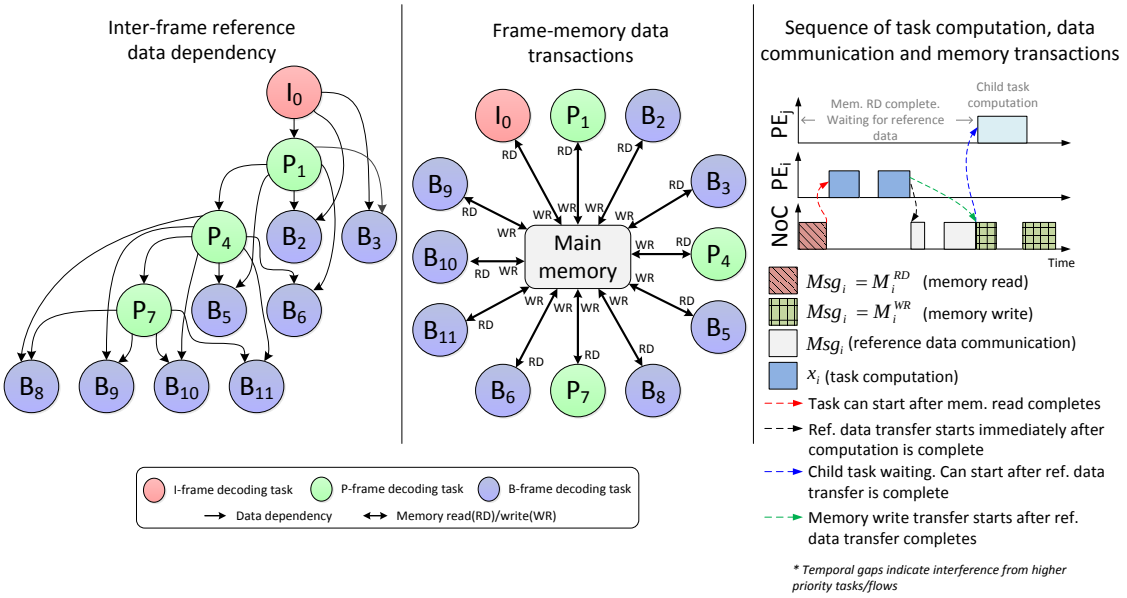


Figure 5.1: MPEG GoP decoding task model with memory transactions

Each task's data (i.e. encoded frame data) needs to be transmitted to its respective PE before starting the task execution. This process is referred to as a *memory read*. After the MPEG decoding task has completed, its output (i.e. decoded frame data) is transmitted to the frame-buffer located in the main memory, referred to as *memory write*. Memory read and write transactions are NoC message flows and are denoted as M_i^{RD} and M_i^{WR} respectively. The payload of a M_i^{RD} is

characterised by the size of the encoded frame as shown in Eq. (5.1). 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; which is similar to the stream analysis shown in [60]. The payload of a M_i^{WR} is essentially the decoded frame data size as given in Eq. (5.2). Memory read flows are given a higher priority over data and memory write flows as the task data is required immediately for a task to start its execution.

$$PL_{M_i^{RD}} = \begin{cases} [res(VS_i) \times bpp] \times 0.4, & \text{if } f_t = I \\ [res(VS_i) \times bpp] \times 0.2, & \text{if } f_t = P \\ [res(VS_i) \times bpp] \times 0.1, & \text{if } f_t = B \end{cases} \tag{5.1}$$

$$PL_{M_i^{WR}} = [res(VS_i) \times bpp] \tag{5.2}$$

The example illustrated in Figure 5.1(right) shows the sequence of computation, communication and memory transactions for a task running on PE_i. M_i^{RD} occurs before the task's computation, after which the task's reference data transfer and M_i^{WR} can start. The M_i^{WR} has to wait until the reference data transfer is complete as it shares the local link and has lower priority. Once the reference data transfer is complete, the child task (mapped to PE_j) can start execution, assuming the child task has completed its memory read (note that the child task's memory read/write transactions are not illustrated).

5.1.1.2 Video-centric task priority assignment

The original task model presented in Section 3.1.1.2 and later used in Chapter 4, assumes a random task priority assignment. In such an assignment, there may be situations in which the response time (and therefore the QoE) of low resolution videos may be severely degraded, due to blocking incurred by several long running high resolution streams. This may seem counter-intuitive from a user perspective. In this chapter, a more sensible assignment policy is followed, which reflects common practice in the video streaming industry. However, it is worth emphasising that this work is not trying to propose a novel priority assignment scheme for dynamic workloads, but only attempts to optimise on the task allocation, for a *given* priority assignment. The proposed mapping heuristics can work around any other priority assignment approach.

The priorities for the video decoding tasks are assigned according to Eq. (5.4), where tasks of *low resolution* video streams are given *higher priority* over high-resolution video streams. In Eq. (5.4), f_{ix} denotes the frame index within the GoP. The $(t_c \times offset)$ component ensures a first-come-first-served (FCFS) selection between equal resolution video streams (t_c denotes current time). The quality and dependency-aware frame priority assignment given in [147] is used to assign priorities between frames within a GoP as shown in Eq. (5.3). Similar to the random priority assignment scheme, task priorities are assigned upon admission of a new video stream and the same priority values are used for tasks in subsequent jobs of that stream.

$$GoP_{fr}^{pr} = \{12, 11, 4, 7, 10, 3, 5, 9, 2, 6, 1, 8\} \tag{5.3}$$

$$p_i = (res(VS_i) - GoP_{fr}^{pr}[f_{ix}]) + (t_c \times offset) \tag{5.4}$$

5.1.2 Platform model refinements

This chapter presents the following refinements to the platform model:

- Platform model changes to accommodate memory transactions.
- An open-loop resource management scheme.
- Removing buffer space constraints.

5.1.2.1 Memory interfaces

As explained in Section 5.1.1.1, tasks read and write to/from the global input buffers location in main memory. Memory read/write transactions occur via the NoC and to/from the memory controllers located around the NoC. As shown in Figure 5.2, the system has 4 main memory controllers (MMC) located on the edges of the platform and each MMC has dual ports. Such a multiple memory controller configuration is similar to the Tilera many-core architecture [4]; although they have four controllers arranged only above/below the NoC. The model assumes every MMC can access the entire main memory, as is the case on Tilera many-cores when the memory is not striped. Any PE can access any MMC on the platform and memory transaction traverse in a multi-hop manner across the NoC similar to data traffic flows. In this model it is assumed that tasks by default communicate with the MMC port (MMCP) closest to it.

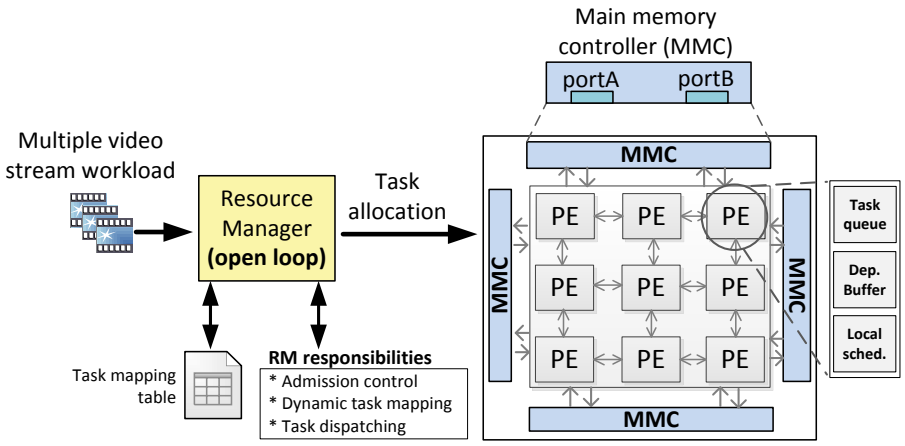


Figure 5.2: Refined system model (with memory controllers and open-loop RM)

5.1.2.2 Open-loop resource management

Section 3.1.2.1 introduced a closed-loop RM which queries for the system status and receives feedback from the PEs. As discussed in Section 2.3.2.1 and Section 2.3.2.5, this type of centralised, closed-loop approach is not suitable for large-scale NoCs where the feedback/monitoring overhead would cause network congestion and degrade performance [30, 35, 36]. Therefore, in this section, an open-loop RM is presented which does not rely on feedback to perform its duties. The high level sequence diagram of the simulation implementation of the open-loop RM is shown in Figure 5.3. Comparing Figure 5.3 with the previous closed-loop implementation in Figure 4.2 shows several differences as well as similarities. As normal, the RM performs admission control, task dispatching, mapping and priority assignment functions. After mapping the tasks the RM performs admission control, and if the stream is admitted the respective tasks

are released to the mapped PEs. Tasks no longer notify the RM regarding their completion, instead all tasks are released together to their mapped PEs by initiating memory read transactions from the main memory. During release, the tasks are tagged with information regarding the location of their child tasks, so that after a task completes execution it knows which PE to send its reference data to. The tasks also send their output to the main memory.

Note that the D-AC does not rely on the feedback loop in previous system model, hence the open-loop mechanism introduced here does not affect it in any way. On the other hand the runtime task mapping techniques would now rely solely on the information in the TMT to make the runtime mapping decisions, hence their mapping quality would be reduced. Using an open-loop RM is a key design decision, chosen to trade-off mapping quality to achieve reduced usage of the NoC, leading to lower congestion and energy consumption. Furthermore, in terms of scalability, as discussed in Section 2.3.2.1, when the NoC size and workloads increase the continuous feedback to the RM as seen in a closed-loop system can lead to network congestion bottlenecks.

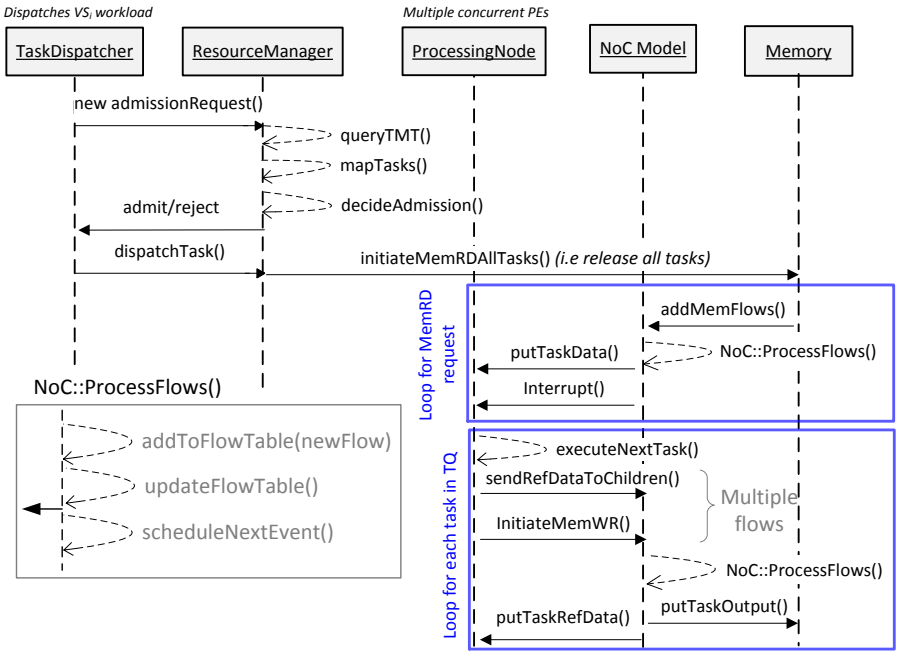


Figure 5.3: Refined system model (with memory controllers and open-loop RM)

5.1.2.3 Infinite buffer space

In Chapter 4, it was assumed that the buffers in the system (i.e. global input buffers, PE local task queues and dependency buffers) have a finite space to accommodate tasks and dependency data. If these buffers are full, video streams are rejected and new jobs of active streams may be dropped. Dropping of jobs due to buffer overflow reduces the utilisation of the system and presents a confounding variable to any performance evaluation. Hence, from this chapter onwards it is assumed that the above mentioned buffers have infinite space. Although this is not a realistic assumption, it simplifies the evaluation of the resource allocation techniques. Also, it opens up the possibility for designers to evaluate which buffer size a system should have, to prevent performance bottlenecks.

5.1.3 Memory-specific response time analysis refinements

The memory transactions introduced in the application model need to be integrated into the job E2ERTA given in Section 4.2.1 to ensure the predictability guarantee of the D-AC is not reduced. As shown in Figure 5.1 each task now has a new inbound and outbound edge representing the memory read and write transactions. This would also add two nodes (start and end) to all the paths of the initial TG. For example a non-memory aware TG would have an end-to-end path such as $(I_0 \Rightarrow P_1 \Rightarrow B_2)$ but now it would be changed to $(M^{RD}_{I_0} \Rightarrow I_0 \Rightarrow P_1 \Rightarrow B_2 \Rightarrow M^{WR}_{B_2})$. Also, there would exist short paths including only one computation $(M^{RD}_{B_2} \Rightarrow B_2 \Rightarrow M^{WR}_{B_2})$. The memory read/write transactions would be modelled as new edges in the application TG with weights indicated by the flow payloads $(PL_{M^{RD}_i}, PL_{M^{WR}_i})$. These new memory related paths need to be taken into account when determining the worst-case response time of a job (i.e. the $(WCRT(J^{CP}_i)$ calculation in Algorithm 27). Similarly, in this new model, the memory read/write transactions are taken into account when measuring a job's *observed* end-to-end response time via simulation.

5.2 Least worst-case remaining slack heuristic

This section defines the least worst-case remaining slack (LWCRS) heuristic, which will facilitate the two runtime task mapping techniques presented in the following sections. LWCRS attempts to select a PE with a *balanced-blocking* approach, considering both the blocking incurred to a target task and the interference caused to lower priority tasks from the target task. It takes into account the blocking behaviour of the tasks already mapped onto a given PE to approximate the remaining slack a PE has to accommodate a new task. The objective is to find the PE mapping that will result in the tightest temporal-fit, without missing the deadlines of the target task nor any of the already mapped tasks. Algorithm 5.1 provides the utility function used to find the task-to-PE mapping which will give the LWCRS for the target task. The *worst-case slack* of a task is the difference between the task deadline and worst-case computation cost. This function is integral to both runtime mapping strategies introduced in this chapter.

Algorithm 5.1 iterates through the provided *PE_list* and calculates the following for each task-to-PE mapping: (a) $RemSlack_t$ - the worst-case remaining slack (WCRS) for the given target task - blocking incurred by higher-priority tasks already mapped on the PE are taken into account (line 6); (b) $RemSlack_{lp}$ - the WCRS for each of the lower-priority tasks already mapped on the PE, taking into account the WCET (c_i) of the target task (line 7-11). Individual task deadlines (d_i) are calculated using the DEQF scheme [137]. The cumulative WCET of higher priority tasks are used to determine the amount of worst-case blocking for a target task. A weight is then assigned to each PE in *PE_list*, which is equal to the sum of $RemSlack_t$ and $RemSlack_{lp}$, for each of the searched PEs (line 13). The PE with the lowest weight is selected and returned (line 17,18). If no PE is found with *positive* worst-case slack the PE with the lowest utilisation is selected and returned (line 20-22). As the WCET of tasks are used to calculate the slack and the deadlines, the calculations are conservative but safe.

5.3 LWCRS-aware runtime mapping

The LWCRS aware mapping algorithm (henceforth denoted as LWCRS) makes use of the utility function introduced in Section 5.2. It attempts to temporally pack tasks tightly as well as minimise the distance between communicating tasks to reduce network congestion. The temporal

Algorithm 5.1: *_get_PE_least_slack* pseudo-code - Find PE with the least worst-case remaining slack

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

1 $PE\_packing = \{\}$ structure to store PE weights
2 **forall** $PE_i \in PE\_list$ **do**
/* obtain the following from TMT */
3 Get $TMT(PE_i)$: tasks already mapped on PE_i
4 Get $hp(\tau_i), lp(\tau_i)$: high and low priority tasks in $TMT(PE_i)$ w.r.t τ_i
/* get worst-case remaining slack (WCRS) to target task */
5 $\tau_i^{slack} = d_i - c_i$ (task slack w.r.t estimated sub-task deadline)
6 $RemSlack_t = \tau_i^{slack} - \sum_{\forall \tau_j \in hp(\tau_i)} c_j$
/* get WCRS on low-pri mapped tasks */
7 $RemSlack_{lp} = \{\}$
8 **forall** $\tau_j \in lp(\tau_i)$ **do**
9 $RemSlack_j = \tau_j^{slack} - \sum_{\forall \tau_k \in hp(\tau_j)} c_k$
10 Insert $RemSlack_j$ to $RemSlack_{lp}$
11 **end**
/* assign weights to each PE - combination of blocking factors */
12 **if** $RemSlack_t > 0$ *and* $\forall x \in RemSlack_{lp} | x > 0$ **then**
13 $PE\_packing[PE_i] = RemSlack_t + \sum RemSlack_{lp}$
14 **end**
15 **end**
/* choose PE with lowest positive weight, if any exists */
16 **if** $\forall x \in PE\_packing \mid x > 0$ **then**
17 PE_j = index of MIN(PE_packing)
18 Return (PE_j, FALSE)
19 **else**
/* else choose PE with lowest utilisation */
20 Sort PE_list by increasing order of PE utilisation (Eq. 2.8)
21 $PE_j = PE\_list[0]$
22 Return (PE_j, FALSE)
23 **end**

packing enables parallel video streams to be pipelined on the PEs, such that the initial PEs in the NoC will be heavily utilised before selecting the next available PE. This allows the system to potentially admit and handle a higher number of simultaneous video streams, without missing any deadlines. This type of mapping is dissimilar to the LM mapper used in Section 4.1, where the workload is distributed evenly, and makes use of less PEs as possible whilst attempting to ensure deadlines are not violated. LWCRS is a *general purpose* technique, that can be applied to map other types of applications, modelled as DAGs and scheduled using fixed priority-preemptive scheduling.

The algorithm of LWCRS mapping is given in Algorithm 5.2. The algorithm uses a copy of the TMT. The *_get_PE_with_least_slack* utility function is used to find a PE which gives the LWCRS for each task in the job (lines 3 and 11). 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 within increasing hop distances from its *closest parent* (lines 9-17); the helper function *getPENeighbours()* is used for this purpose. A node's closest parent (τ_i^{PARENT}) is defined as the node with the longest path from the root node. For example in Figure 5.4, 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

Algorithm 5.2: LWCRS-aware mapping heuristic algorithm pseudo-code

```
   Input  : all tasks in the job (J_0) sorted in the frame decoding order in the GoP,
            TMT : the runtime task mapping table,
            PE_list : list of PEs to search
   Output: Updated TMT with new task to processing element mapping (τ_i → PE_i)
1  forall unmapped tasks : τ_i ∈ J_0 do
       /* if task is the root node - τ_0                                          */
2      if τ_i == τ_0 then
3          (PE_i, found)=get_PE_least_slack(τ_i, TMT, PE_list) /* call Algorithm 5.1 */
4          Map (τ_i → PE_i); Update TMT
5      else
           /* obtain following from TMT                                           */
6          Get τ_i^PARENT : closest parent task
7          Get PE_i^P : PE that τ_i^PARENT was mapped onto
           /* get neighbouring PEs at increasing hop_counts                       */
8          for hc = 1 to MAX_HOPS do
9              N_PE_i^P = getPENeighbours(PE_i^P, hc)
10             Append PE_i^P into N_PE_i^P
11             (PE_i, found) = get_PE_least_slack(τ_i, TMT, N_PE_i^P) /* call Algorithm 5.1 */
12             if found == TRUE then
13                 Map (τ_i → PE_i); Update TMT
14                 break loop; Go to step 1
15             end
16         end
           /* if no suitable PE is found, select closest lowest utilised PE       */
17         if found == FALSE then
               /* else choose neighbouring PE with lowest utilisation             */
18             N_PE_i^P = getPENeighbours(PE_i^P, hop_count = 1)
19             Sort N_PE_i^P by increasing order of PE utilisation (Eq. 2.8)
20             PE_i = N_PE_i^P[0]
21             Map (τ_i → PE_i); Update TMT
22         end
23     end
24 end
25 Return updated TMT
```

node to the PE with minimum utilisation (line 17-22). 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}. The TMT copy is updated at runtime in each iteration of the main loop.

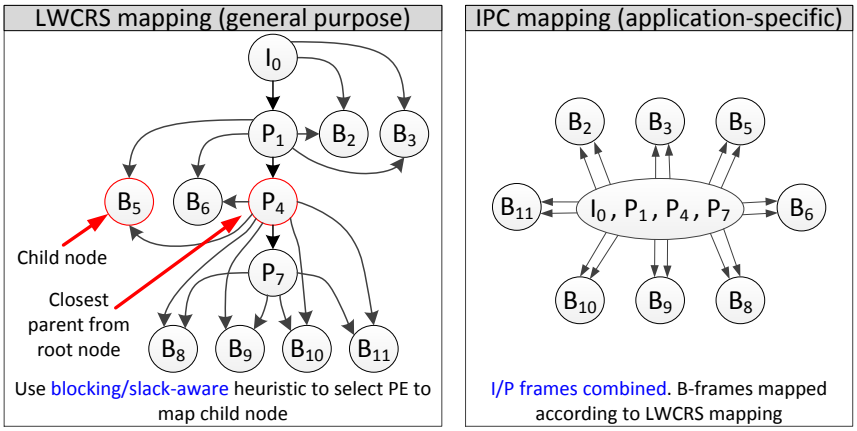


Figure 5.4: Illustration of LWCRS mapping closest parent selection *(left)* and IPC mapping task grouping *(right)*

5.3.1 LWCRS mapping complexity analysis

The worst-case time complexity of LWCRS mapping algorithm ($W_C(LWCRS)$) can be calculated as Eq. (5.5). The term $|J_i|$ denotes the number of tasks in a job, MAX_HOPS is the maximum hop count for a given network size, $|PE|$ is the number of PEs in the platform and $|PE_{TQ}|$ represents the maximum number of tasks in a PEs task queue. In our current application model the $|J_i| = 12$ (12 frames per GoP). In the case of a regular 2D mesh NoC wherein the rows and columns are equal, the MAX_HOPS can be reduced to $MAX\_HOPS = (2\sqrt{|PE|} - 2)$, thereby giving Eq. (5.6). By dropping the constant terms and variable coefficients, the $W_C(LWCRS)$ can be reduced to Eq. (5.7) and further reduced to Eq. (5.8) by dropping the lower order term. Finally, $|TQ| \leq |PE|$ can be assumed as the D-AC would not allow the system to be overloaded, thereby giving the worst-case runtime complexity of LWCRS as Eq. (5.9), which is less than $O(n^3)$ but worse than $O(n \log n)$.

$$W_C(LWCRS) = O\left[|J_i| \times MAX\_HOPS \times |PE| \times |TQ|\right] \tag{5.5}$$

$$W_C(LWCRS) = O\left[12 \times (2\sqrt{|PE|} - 2) \times |PE| \times |TQ|\right] \tag{5.6}$$

$$W_C(LWCRS) = O\left[\left((|PE|)^{3/2} - |PE|\right) \times |TQ|\right] \tag{5.7}$$

$$W_C(LWCRS) = O\left[(|PE|)^{3/2} \times |TQ|\right] \tag{5.8}$$

$$W_C(LWCRS) = O\left[(|PE|)^{5/2}\right] \tag{5.9}$$

Since the mapping algorithm is executed only once for a video stream its runtime overhead (in the order of a milliseconds) is negligible compared to the duration of a video (order of minutes/hours).

5.4 Clustered I and P frames mapping

Unlike the LWCRS mapper, the clustered I and P frames mapping heuristic (henceforth denoted as IPC) exploits known application-specific TG dependency patterns. By inspecting the video job TG (Figure 5.1), it is clear that the I and P frames lie on the longest-path in the TG. The longest path of the TG is the simple path with most number of non-repeating nodes (ignoring node and edge weights). Even if the node and edge weights are taken into account, the task chain $I_0 \rightarrow P_1 \rightarrow P_4 \rightarrow P_7$ also lies in the critical path of the TG (assuming no B-frame task blocking is encountered). Therefore, by grouping the I and P frame tasks together the critical path WCRT of the TG is essentially reduced. Searching for the critical path of the TG is not necessary as the dependency structure of the TG is known a priori. The B-frames have no inter-dependencies, hence they can be processed in parallel. The B-frame decoding tasks can occupy smaller temporal gaps in the execution schedule as their computation cost is lower than I/P frames.

Figure 5.4 (right) illustrates the TG after the grouping of I and P frames. Clustering the I/P frames together has two distinct advantages : (a) it reduces the NoC congestion/interference as less 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, without waiting for message flows. The IPC mapping technique works as described in Algorithm 5.3. 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 1-hop distance. The LWCRS heuristic (Algorithm 5.1 is used to select a PE within the 1 hop distance region.

Algorithm 5.3: IPC mapping heuristic algorithm pseudo-code

Input : all tasks in the job (J_0) sorted in the frame decoding order in the GoP,
TMT : the runtime task mapping table,
PE_list : list of PEs to search
Output: Updated TMT with new task to processing element mapping ($\tau_i \rightarrow PE_i$)

```
/* first find lowest utilised PE                                              */
```
Sort PE_list by increasing order of PE utilisation (Eq. 2.8)
$PE_{IPC} = PE\_list[0]$
forall *unmapped tasks :* $\tau_i \in J_0$ **do**
```
    /* if task is either a I or P-frame decoding task map to lowest utilised PE */
```
    if $f_t(\tau_i) == \{I \| P\}$ **then**
        Map ($\tau_i \rightarrow PE_{IPC}$); Update TMT
    else
```
        /* map B-frames at 2-hop distances from their closest parent          */
```
        Get τ_i^{PARENT} : closest parent task
        Get PE_i^P : PE that τ_i^{PARENT} was mapped onto
        $N\_PE_i^P$ = $getPENeighbours(PE_i^P, hop\_count = 1)$
        $(PE_i, found)$ = ***get_PE_least_slack***$(\tau_i, TMT, N\_PE_i^P)$
        Map ($\tau_i \rightarrow PE_i$); Update TMT
    end
end
Return updated TMT

5.4.1 IPC mapping complexity analysis

The IPC mapping algorithm uses the LWCRS heuristic given in Algorithm 5.1. Therefore, its worst-case timing complexity $W_C(IPC)$ analysis follows a similar derivation to $W_C(LWCRS)$ as given in Eq. (5.10) and can be reduced to Eq. (5.11). Similar to $W_C(LWCRS)$, if $|TQ| \leq |PE|$ is assumed, then $W_C(IPC)$ further reduces to Eq. (5.12).

$$W_C(IPC) = O\left[|J_i| \times |PE| \times |TQ|\right] \tag{5.10}$$

$$W_C(IPC) = O\left[|PE| \times |TQ|\right] \tag{5.11}$$

$$W_C(IPC) = O\left[(|PE|)^2\right] \tag{5.12}$$

5.5 Baseline static task mapping

This section presents a search-based static HRT mapping optimisation technique, used as an upper baseline to evaluate the dynamic task mappers presented in the previous two sections. This static mapper is an extended version of the technique introduced by Sayuti et al. [172]. As discussed in Section 2.3.3, static mapping algorithms are suitable when a complete view of the application workloads and platform conditions are known at design time. This static HRT mapper is purely used as an upper baseline, to evaluate the performance of the proposed run-time mappers. An indication of a good runtime mapping heuristic is one which shows results as close as possible to the upper-baseline. However, recall that static mappers not only rely on full knowledge of the workload, but also incur a considerable runtime execution overhead; hence, they are not suitable for use in runtime mapping.

In [172], the authors use a multi-objective genetic algorithm (GA) based optimisation strategy to optimise the task to PE mapping approach. GAs start with a random initial population of candidate solutions and gradually evolves the populations towards the global optimum using a given fitness statistic. As illustrated in Figure 5.5, the algorithm in [172] uses standard evolutionary GA pipeline constructs such as single-point crossover, bit-flip mutation and binary tournament selection to generate a new population for each generation. Each GA individual is represented by an integer-based chromosome structure indicating the PE mapping for each task. The index of the gene represents the task index and the value of the gene indicates the PE index. Hence, the length of the chromosome varies with respect to the number of tasks in the workload. Elitism is used to ensure the best individual of each generation is advanced to the next generation. However, due to the random nature of solution development, GAs do not guarantee optimality of the best solution which it finds.

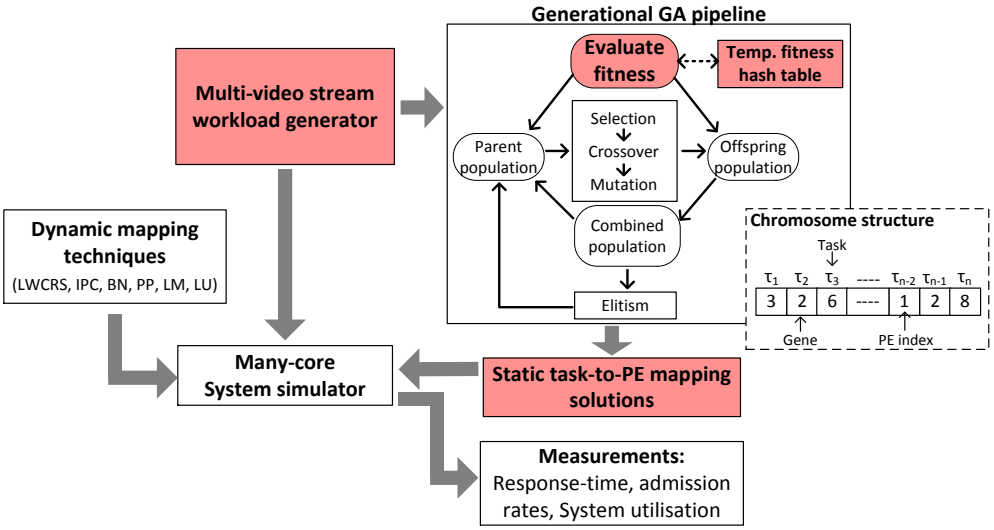


Figure 5.5: Illustration of the GA pipeline from [172] adapted and integrated to the experimental design flow

The GA-based static HRT mapper given in [172] was adapted and integrated with our application/platform model and metrics. The shaded components in Figure 5.5 indicate the changes made to the GA and the design flow with respect to the GA in [172]. In the experimental design flow, the abstract video stream 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. This allows evaluation of both the dynamic and static mappers under the same workload. This adapted GA-based static mapping technique is henceforth referred to as *GA-MP*. The following extensions were added to the GA framework in [172]:

Application specific extensions: A direct integration of the GA framework in [172] without several application specific adaptations is not possible. For example, the precedence constraints of the TG, as discussed in Section 4.2.1.1, were taken into account when calculating the task/flow interference sets. Memory read/write traffic has been incorporated into the set of flows in the application and the task mapping configurations (Section 5.1.1.1). Furthermore, as mentioned in Section 4.2.1, flow-level optimisations were carried out such as redundant flows are removed for each mapping configuration when multiple children are mapped to same PE.

Points-based GA fitness function: Unlike in [172], this work is concerned with the end-to-end schedulability of an entire video stream rather than individual tasks/flows. Therefore, a points-based single-objective fitness function as described in Algorithm 5.4 is used for this purpose. Firstly, the $WCRT(J_i^{CP})$ of every video stream needs to be calculated. These calculations are similar to the E2ERTA performed by the D-AC (Section 4.2.1). The loop in lines 2-10 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 6 and 8 of Algorithm 5.4 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 enables the GA to pick individuals with task mappings that have lower distributions of $WCRT(J_i^{CP})$ and penalise individuals who have unschedulable video streams. For example consider the following scenario. Two GA individuals, A and B have equal number of fully schedulable streams and one unschedulable video stream each, but they have fitness scores -1.65 and -1.15 respectively. This indicates that A's unschedulable video missed its deadline by a lower margin than B's unschedulable video. Hence, using a points-based fitness score the individual A is preferred over B, whereas if an integer based fitness score is used, both these individuals will be ranked equally.

Algorithm 5.4: Points-based GA fitness function

Input : - Real-time characteristics of all tasks and flows of every video stream VS_i for a given workload (WL).
- Task to PE mapping configuration (i.e. one individual of the GA population)

Output: points-based fitness score

```
   /* Calculate points for all videos in workload (WL)                    */
1  points = 0
2  forall VS_i ∈ 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 ≥ D_e2e then
           /* unschedulable                                                */
6          points += 1 × (WCRT(J_i^CP) − D_e2e) / D_e2e
7      else
           /* fully schedulable                                            */
8          points += −1 × (D_e2e − WCRT(J_i^CP)) / D_e2e
9      end
10 end
11 Return points
```

GA implementation optimisations: Due to the extensions above, the fitness evaluation of the GA becomes more complex. A hash table of task mapping solutions and corresponding fitness scores are maintained and looked-up to avoid repeated evaluation of the same gene. The hash table is cleared after its number of entries reaches a fixed threshold to reduce the memory footprint. 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 : $\max_{VS_i \in WL}(WCRT(J_i^{CP})) < D_{e2e}$

5.6 Evaluation

The evaluation section of this chapter has three separate experiments covering three objectives with respect to measuring the performance of the proposed dynamic mapping techniques.

- Experiment A ***(ExpA)*** - Evaluate the improvement to the admission rate and system utilisation of the D-AC approach when using the proposed LWCRS and IPC runtime mapping techniques.
- Experiment B ***(ExpB)*** - Compare LWCRS and IPC to the GA-based static HRT task mapping technique (GA-MP) presented in Section 5.5.
- Experiment C ***(ExpC)*** - Investigate the performance of LWCRS and IPC with respect to different workload CCRs and scalability in terms of increasing NoC sizes.

5.6.1 Experimental design

The application and platform models used in the evaluation follow the characteristics described in Section 5.1.The preliminary experiments assume a 3×3 NoC platform but larger NoC sizes are also explored as per Table 5.1. The PEs are assumed to have an operating frequency of 200MHz. Message flow header routing cost is assumed to be 7 clock-cycles, NoC frequency is set at 10MHz and the link width is set to 16 bytes. As explained in Section 3.3, a lower NoC frequency (i.e. low bandwidth) is assumed, in order to induce a reasonable amount of network utilisation/congestion. Such conditions cannot be created using a higher NoC frequency for the same workload level. Table 5.1 summarise the experimental conditions, parameters and metrics used to explore the evaluation objectives described above. The abstract simulation framework as described in Chapter 3 and Section 4.3.1 is used in these experiments. However, the sequence of events now follows the open-loop RM process as shown in Figure 5.3. The D-AC admission controller is used for all experiments except in *ExpC*, where no admission control (No-AC) is used. When the D-AC is used, all admitted video streams are fully schedulable.

Table 5.1: Summary of LWCRS and IPC experimental evaluation design parameters

Eval. obj.	Workload configuration	NoC size	Independent variables	Response variables	AC
ExpA	increasing workloads: $WL = \{4.3\times10^4 .. 3.0\times10^6\}$, $\lvert WL\rvert = 460$, $\max\lvert VS\rvert = 9$ (per WL)	3×3	*Mapping techniques:* LWCRS, IPC, PP, BN, LU, LM	Admission rate, PE busy time	D-AC
ExpB	increasing workloads: $WL = \{54.2\times10^4 .. 2.3\times10^6\}$, $\lvert WL\rvert = 10$, $\max\lvert VS\rvert = 8$ (per WL)	3×3	*Mapping techniques:* LWCRS, IPC, PP, BN, LU, LM, GA-MP	Admission rate, PE busy time, NoC busy time	D-AC
ExpC	$\lvert VS\rvert = \{9, 25, 49, 81\}$ (proportional to NoC size), Therefore WL is random (but bounded by $\lvert VS\rvert$)	3×3, 5×5, 7×7, 9×9	*Mapping techniques:* LWCRS, IPC, PP, BN, LU, LM *Varying CCR levels:* $CCR(WL) = \{0.001, 0.5, 1.0, 1.5, 2.0\}$	Analytical $WCRT(J_i^{CP})$	No-AC

5.6.1.1 Baseline task mapping techniques:

The following dynamic task mapping techniques existing in the literature are used as baselines.

- Best neighbour heuristic **(BN)**: as defined in [32]. 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.
- Pre-processing based communication and computation balancing heuristic **(PP)**: as defined in [33]. As the TG dependency pattern is fixed and known, the pre-processing stage of PP (Algorithm 2.1) is performed at design time.

Evaluation is also performed against two load-balancing task allocation heuristics which attempt to evenly distribute the load of the application across available PEs:

- Least mapped heuristic **(LM)**: selects the PE with the minimum number of mapped tasks according to the TMT. This mapper was used in the work presented in Chapter 4, and the pseudo code given in Algorithm 4.1.
- Least utilised heuristic **(LU)**: similar to LM, this mapper iterates through all tasks in the job and maps each task to the lowest utilised PE. The worst-case PE utilisation is estimated via Eq. (2.8).

The GA-MP static mapper with functionality as explained in Section 5.5, is used as an upper baseline to evaluate the performance of the proposed mappers. For all experiments the crossover and mutation rates were fixed at 0.5 and 0.01 respectively, similar to the GA in [172]. Higher workloads will be executed with a larger number of GA evaluations to suit the increasing complexity of the search problem; where the number of evaluations = number of generations $\times$ population size.

5.6.1.2 Workload conditions:

The workload conditions are varied based on the experiment type as shown in Table 5.1. In all workloads video stream resolutions are selected at random from a range of resolutions (e.g. high: 720×576, low: 230×180), with 1 video per workflow and 5 GoPs per video. Simulations were carried out for 30 unique seeds for every workload condition, which results in varying video stream arrival patterns and task execution costs. For the evaluation objective *ExpA*, an increasing workload WL profile is used with 460 different workload levels with a maximum of 9 video streams each. The WL level is calculated as per Eq. (3.7). The term $|VS|$ in Table 5.1 denotes the number of simultaneous video streams. For *ExpB*, 10 increasing workload levels were used to compare the dynamic mappers against the GA-MP static mapper. As each of the 10 workloads have 30 randomly seeded variations, the GA-MP task mapping optimisation is carried out for each of the seeds. The workload configuration used for *ExpC* varied the $|VS|$ proportional to the size of the NoC. Each seeded simulation run will have a random WL constrained by $|VS|$.

5.6.1.3 Varying CCR levels:

In *ExpC*, the CCR level of the workload and the platform size are varied. In reality the CCR of a workload can also vary depending on platform modifications. For example if optimised/specialised hardware components (e.g. accelerators) are used in the PEs to accelerate the computational elements in the video decoder the CCR would be high (i.e. communication-bound). On the other hand if a high bandwidth or faster NoC was used the CCR would be low (computation-bound). Task mapping heuristics that target workloads with a low-CCR (such as load-balancing

techniques) may perform poorly when the CCR is high. 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. (5.13). In this calculation, the flow's C_i and the task's c_i are used as the edge and node costs of the TG 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. (5.14)). To vary the CCR, the task computation cost is kept constant and gradually the NoC frequency is varied to change the cost of the edges. Increasing the NoC frequency will reduce the communication cost thereby decrease the CCR and vice versa to increase the CCR.

$$\text{CCR of a video stream: } CCR(VS_i) = \frac{\text{Total cost of TG edges}}{\text{Total cost of TG nodes}} = \frac{\sum\limits_{\forall edges \in J_i} C_i}{\sum\limits_{\forall \tau_i \in J_i} c_i} \tag{5.13}$$

$$\text{CCR of a workload: } CCR(WL) = \frac{\sum\limits_{\forall VS_i \in WL} CCR(VS_i)}{|VS|} \tag{5.14}$$

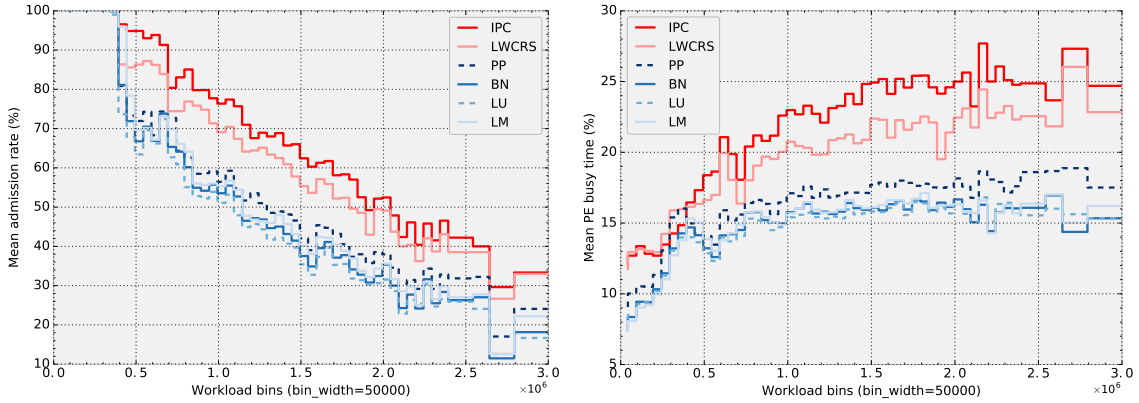
5.6.2 Results discussion

5.6.2.1 ExpA - Admission rates vs. PE busy times

The results from *ExpA* are shown in Figure 5.6. Each data sample in the distributions represents the mean admission rate (Figure 5.6a) and mean PE busy time (Figure 5.6b) as percentages per workload level. The x-axis represents the different workload levels (WL) separated into equal width bins; the y-axis steps represents the mean of the data in each bin.

Admission rates decrease as the workload increase, because for high workloads the D-AC cannot guarantee the timing requirements will be met, and hence more rejections will be made. Around the mid-high workloads the proposed mapping techniques (IPC and LWCRS) show an improvement of about 10%-15% over the baseline runtime mappings. Since the WCRT of the jobs are lower when these two mappers are used, the D-AC will admit more video streams. Overall, IPC performs better than LWCRS for all workloads, showing a 2-8% improvement. However, recall that IPC is an application specific heuristic which makes use of known characteristics of the application TG unlike LWCRS which is application agnostic. The admission rate improvement of IPC and LWCRS over the baselines drop slightly during certain higher workloads (e.g. $2-2.25\times10^6$). PP shows slightly better admission rates than the other baselines for all workload levels because it performs task grouping. The LU, LM and BN techniques show comparable admission rates.

Higher admission rates result in more tasks being processed by the system, leading to increased PE utilisation as depicted in Figure 5.6b. As the workload increases, the PE utilisation also increase up to a certain level (approx. 1.0×10^6), after which the PE busy time starts to level off, due to the decline of the admission rates. The proposed IPC and LWCRS task mappers show a 5%-10% improvement in utilisation over the LM, LU, BN and PP mappings for workloads over 1.0×10^6. IPC utilises the system more than LWCRS for workloads higher than 0.5×10^6. PP utilises the PEs more than the other compared baselines, especially at higher workload levels (i.e. over 2.3×10^6). The PE busy time is a function of both the number of video streams admitted and their spatial resolution; thus for example IPC and LWCRS have similar admission rates at 2.8–3.0×10^6 but different PE busy times.



(a) **(b)**

Figure 5.6: Performance of the runtime mapping techniques for a range of workload levels (a) Admission rates, (b) PE busy time

5.6.2.2 ExpB - NoC usage and comparison against GA-MP baseline

Figure 5.7 shows the results from *ExpB*, where predictability and performance of the different task-mapping heuristics in terms of mean admission rate and mean PE and NoC busy time are given, under 10 different workload levels. Similar to Figure 5.6, the proposed IPC and LWCRS mappers show a significant improvement in admission rates and PE busy times over the baseline runtime mappers. The trend of the results are similar to ExpA (Figure 5.6), however in *ExpB* the NoC busy time is also measured. The NoC busy time results shown in Figure 5.7(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. Both IPC and LWCRS attempt to reduce task communication; hence the mappings that they perform result in higher PE utilisation while the baseline mappings have a higher NoC utilisation due to more communication. IPC utilises the NoC for inter-task communication more than LWCRS. LWCRS produces a tighter grouping of tasks than IPC resulting in lower number of PEs being used. 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. LWCRS could still have memory traffic congestion, leading to higher $WCRT(J_i^{CP})$ and reduced admission rates compared with IPC. 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 on PEs that cause higher interference.

In the Figure 5.7, GA-MP results denote the task mapping using the genetic algorithm based static hard real-time mapper. GA-MP has full knowledge of the task characteristics and is used only as an upper baseline. The number of GA evaluations taken to obtain these results are given in Table 5.2; here, evaluations refer to the number of generations $\times$ the population size. The mapping search space increases proportionally with the workload level, therefore a higher number of evaluations were required to obtain a reasonable admission rate for higher workloads. Similar to the dynamic mappers, the GA-MP's 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, a gradual decrease in relative improvement is noticeable as the workload level increases. 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 have comparable admission rates, but the GA-MP has higher PE busy times (e.g. 225.4×10^4). This is because in certain scenarios the GA-MP obtains a mapping which makes the D-AC reject lower

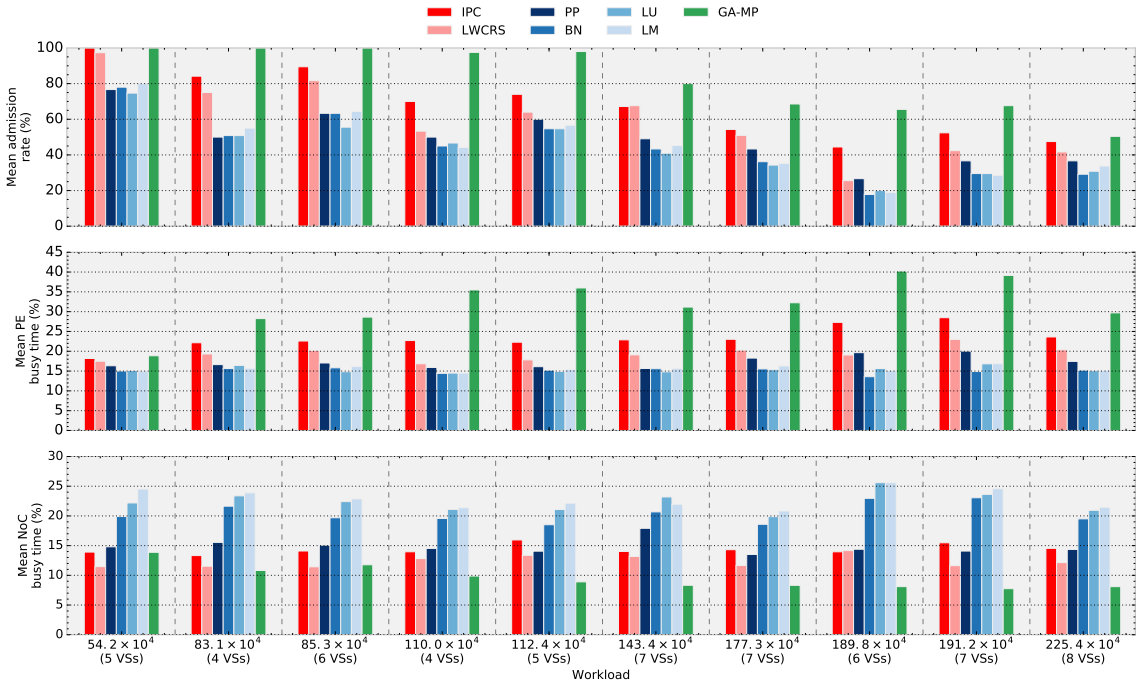


Figure 5.7: Evaluation results of the dynamic mappers vs. static mapper: (*Top:* Admission rate; *Center:* PE busy time; *Bottom:* NoC busy time)

resolution videos but accept higher resolution streams, thus giving rise to higher PE busy times. Further investigation into the mapping configuration by the GA-MP showed that it used only a few PEs per job (on average 2 or 3) and many of the best solutions map children of the same parent together on the same PE. This leads to lower traffic flows, contention and therefore lower job response times and admission rates.

Table 5.2: Number of total GA evaluations for different workloads (corresponding to results shown in Figure 5.7)

Workload (WL)	**# Evaluations**	**Workload** (WL)	**# Evaluations**
54.2×10^4	5000	143.4×10^4	37800
83.1×10^4	6000	177.3×10^4	40000
85.3×10^4	7200	189.8×10^4	40000
110.0×10^4	27000	191.2×10^4	48400
112.4×10^4	32000	225.4×10^4	50000

GA-MP performance with respect to workload level: As the workload increases, the number of the tasks and flows, their computation and communication costs increase (Table 5.3), causing the complexity of the optimisation problem also to increase. To illustrate this, the GA-based mapping optimisation is executed for the ten different workloads with a fixed number of generations and population size (500, 200 respectively); the search terminates when an acceptable mapping solution is found. GA-MP execution runtime for the different workloads are given in Table 5.3. The total execution time 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. Both, the number of tasks and flows and their computation/communication costs attribute to the runtime of the GA-MP. For example $WL = 189.8 \times 10^4$ shows a lower runtime, because $|VS| = 6$ showing that both the number of simultaneous videos 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.

Figure 5.8 shows the generational progress of the GA for the workloads described in Table 5.3 for a single seed. Recall that lower points-based scores relate to GA solutions with higher fitness. There is a dramatic improvement in fitness score in the first 100 generations after which the search converges. Certainly GA parameter tuning can be used to avoid fast convergences but may result in a longer runtime. As the workload level increases, the search space and complexity increases and the GA is unable to find a solution within a reasonable number of evaluations (i.e. runtime). Lower workloads (e.g. $WL = 85.3 \times 10^4$) find a solution quickly and stop the GA progression; in $WL = 54.2 \times 10^4$ the solution is found in 1 generation. It is also important to note that for a given workload, the GA progression trend and their convergence will vary slightly depending on the fitness of the initial random population of the GA. Figure 5.8 shows the progression of a single test run with a single instance of a random initial population.

Table 5.3: GA-MP runtime performance for different workload levels (GA progress shown in Figure 5.8)

Workload (WL)	$\|VS\|$	**Total # tasks**	**Total # flows**	**Runtime (hrs.)**	**# Evals.**	**Solution found**
54.2×10^4	5	60	215	0.31	0.2×10^3	Yes
83.1×10^4	4	48	172	1.22	3.4×10^3	Yes
85.3×10^4	6	72	258	9.05	4.6×10^3	Yes
110.0×10^4	4	48	172	1.68	6.2×10^3	Yes
112.4×10^4	5	60	215	5.26	6×10^3	Yes
143.4×10^4	7	84	301	99.32	10×10^4	No
177.3×10^4	7	84	301	71.68	10×10^4	No
189.8×10^4	6	72	258	37.17	10×10^4	No
191.2×10^4	7	84	301	96.54	10×10^4	No
225.4×10^4	8	96	344	114.49	10×10^4	No

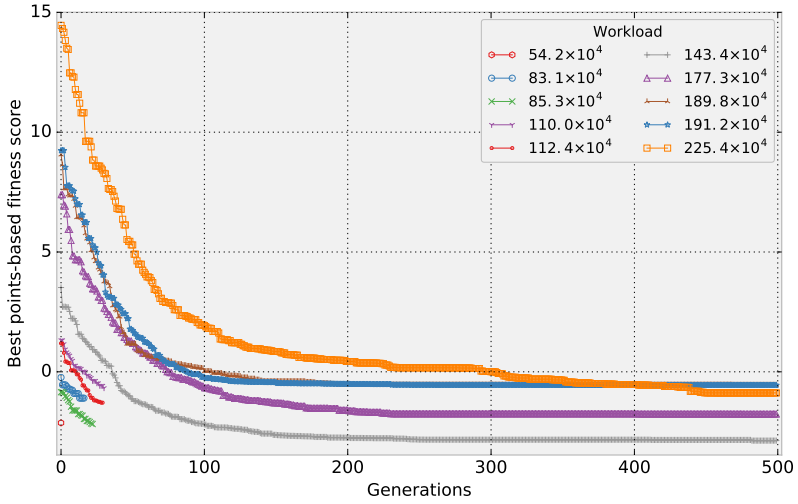


Figure 5.8: GA-MP points-based score convergence over generations, for different workload levels (workload conditions given in Table 5.3)

5.6.2.3 ExpC - Evaluations with respect to CCR and NoC size variations

Figure 5.9 shows the results from *ExpC*, where the analytical $WCRT(J_i^{CP})$ variation (i.e. predictability) in the different dynamic mapping techniques are evaluated for different NoC sizes and workload CCRs. The analytical $WCRT(J_i^{CP})$ or simply $WCRT(J_i^{CP})$, is calculated after all

streams have been mapped and admitted. The box plots represent the $WCRT(J_i^{CP})$ distribution of all streams for all seeded simulation runs and the line plot with circular markers represents the distribution means. Note that in *ExpC*, admission control is disabled (No-AC); hence, a significant amount of streams are admitted but late. Lower $WCRT(J_i^{CP})$ is preferred and correlates to a potential higher admission rate, if a D-AC was used. No-AC is used to observe the performance of the mappers without the influence of an admission controller.

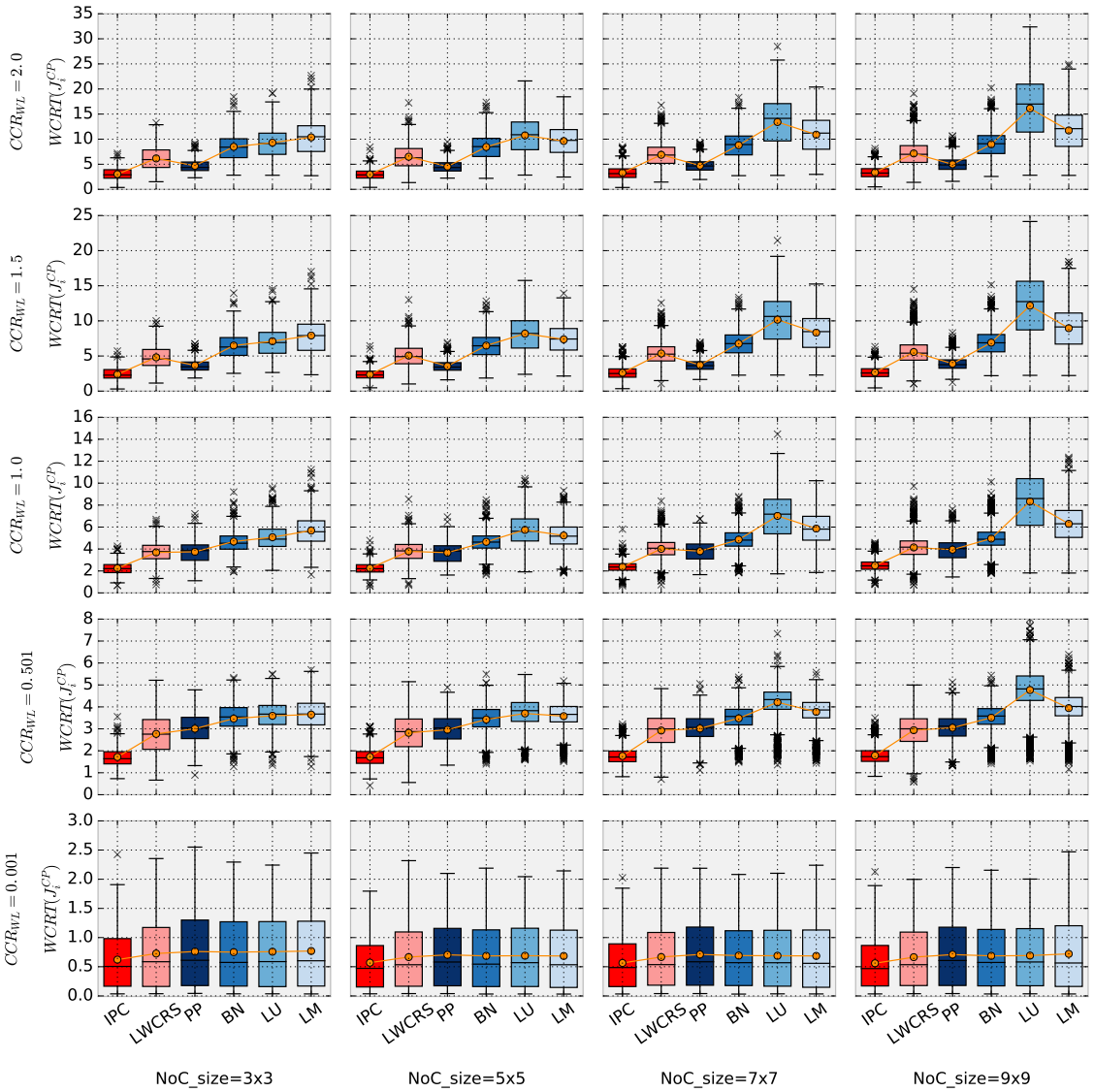


Figure 5.9: Distribution of analytical $WCRT(J_i^{CP})$ obtained using different dynamic mapping approaches, under varying CCR and NoC sizes. Mean values shown by line plot

$CCR(WL) < 1.0$ denotes computation-bound workloads and $CCR(WL) > 1.0$ denotes communication-bound workloads. The $WCRT(J_i^{CP})$ of all mappers increase as the CCR increases, as the communication latency has effectively increased. IPC mapping method perform relatively better than all the baselines in all conditions. LWCRS performs worse than PP for higher CCR and NoC sizes, but shows better results than BN/LU/LM in all conditions. The PP heuristic performs well as it performs better grouping of tasks, but still shows a slightly higher $WCRT(J_i^{CP})$ distribution when compared with IPC. 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 larger distribution range. BN performs better than LM and LU in higher CCRs, as it takes into account the communication channel load as a metric but is worse than PP/LWCRS/IPC as it does not perform task grouping.

5.6.2.4 Results summary

The results from the evaluation confirm that the proposed dynamic mapping heuristics, IPC and LWCRS show an improvement of about 10%-20% in mean admission rates and about 5%-15% improvement in PE busy times, when compared against other existing heuristic based dynamic task-mappers. Furthermore, these improvements can be obtained at a lower usage of the NoC, which could potentially lead to lower power consumption in the system. For larger workloads, the proposed dynamic mappers are only 5%-10% worse than the mapping solution which took 90-100 hours to achieve, by the GA-MP upper baseline.

Results also show that mapping heuristics that rely purely on communication/computation load do not scale well as the NoC size and workload increase. This work also 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.

5.7 Summary and novel contributions

To summarise, this chapter presented the following two novel dynamic task mapping heuristics for hard real-time video stream decoding applications, on NoC-based platforms:

- **LWCRS:** a general-purpose, blocking-aware mapping technique that attempts to tightly pack tasks in the temporal domain. It uses the worst-case remaining slack of tasks as a metric in the mapping process.
- **IPC:** an application-aware mapping technique that clusters the I and P frame decoding tasks (which lie on the critical path) and maps them to a single PE. The remaining B-frame decoding tasks are mapped according to LWCRS.

Both IPC and LWCRS can be used to improve the utilisation of the D-AC by reducing the end-to-end worst-case response time of video stream jobs and thereby admitting more video streams. The proposed runtime mappers are open-loop, meaning they do not rely on monitoring information from the platform or application which is unique unlike existing centralised resource management techniques, surveyed in the literature (Section 2.3.3). Both proposed mappers showed significant improvement over existing runtime mappers.

This work also presented a point-based, WCRT-aware fitness function and experimental design flow which can be used in conjunction with static hard real-time, task mapping, search techniques (such as a genetic algorithm) to suit multi-stream video decoding applications with task dependencies and unknown execution times. This static HRT mapper was used as an upper baseline to evaluate the proposed dynamic mappers and it was able to achieve significantly better mapping solutions for low workloads within an acceptable search time. However, it showed

only a minor improvement over the dynamic mappers for high workloads even after 100 hours of search time. Evaluations were also carried out for larger NoC sizes and varying workload CCR levels. This evaluation showed that it is crucial to take into account the workload CCR and efficient task clustering in the task mapping process, especially if the communication and computation load of an application has high variability.

Chapter 6

Distributed task remapping

The experimental results from Chapter 5 demonstrated how increasing the number of processing cores on a NoC can facilitate processing of higher workloads levels. However, with the increase in the number of cores and concurrent applications the centralised resource manager faces scalability issues and performance bottlenecks as discussed in Section 2.3.2.1 of the literature survey. Hierarchical, cluster-based, distributed resource management techniques have been proposed by many researchers (Section 2.3.2.2) to overcome several limitations of a central management unit. The complexity and dynamic nature of multiple video stream decoding applications and the availability of large-scale distributed NoC architectures motivates the investigation of fully-distributed, autonomous self-organising mechanisms. Even though a handful of such fully distributed techniques do already exist for NoC-based platforms (Section 2.3.2.3), they either rely on complex interaction protocols [163], have high communication overhead [165] and/or require specialised hardware [162, 167].

The work in this chapter presents a bio-inspired, fully distributed, low-overhead task remapping technique that is an extension of the pheromone signalling-based load-balancing protocol introduced by Caliskanelli et al. [104] (described briefly in Section 2.3.2.4). The proposed remapping technique dynamically adapts the mapping of SRT multiple video stream decoding tasks on the NoC-based platform in order to reduce the overall job lateness. Each individual PE periodically executes a lightweight set of rules which gives it the capability to make autonomous task reallocation decisions using its local knowledge. This introduces a high-degree of controlled redundancy into the system, as there is no single point of failure or management entity. The adaptive nature of the distributed system resource management policy proposed makes it difficult to analyse task/flow latencies and hence, the system is not able to provide hard timing guarantees. Therefore, the work in this chapter solely targets SRT or best-effort video stream decoding workloads where the objective is to minimise the resource management overhead whilst reducing the workload lateness.

The second contribution in this chapter is an adaptation of a hierarchical, cluster-based resource management approach as given in Castilhos et al. [157] (an overview given earlier in Section 2.3.2.4). This RM technique is extended to introduce task blocking and relocation distance awareness and is used as a baseline to the proposed bio-inspired remapping technique.

The remainder of the chapter is organised as follows. Section 6.1 describes the changes to the system model that is used in this chapter. In Section 6.2 the protocols of the proposed bio-inspired distributed remapper is defined followed by a description of the cluster-based remapping technique in Section 6.3. An evaluation of the presented task remapping techniques is carried out in Section 6.4.

6.1 System model refinement

Both the application and platform models used in this chapter follow the previous definitions provided in Section 5.1, with a few exceptions as stated in this section.

6.1.1 Omitting the memory transaction model

The previous chapter integrated high-level abstract memory read/write transactions within the application model. The primary experiments carried out by this chapter omit those memory transactions, in order to improve the simulation speed for evaluating both remapping techniques discussed in this chapter. However, the secondary experiment gives an indication of the predictability of PSRM when memory traffic is modelled. The simulation speed of a discrete-event simulation increases proportionally to the number of simulated events. Both the bio-inspired and cluster-based remappers require parameter tuning and hence require several thousands of simulation runs. However, this speed/accuracy trade-off, does not compromise the evaluation as both remappers are evaluated under the same experimental conditions. Furthermore, the state-of-the-art techniques in distributed resource management (e.g. [37, 157, 159]) also do not take into account the main memory traffic in their evaluations. The author acknowledges that memory transactions can impact task mapping performance, and therefore further investigations into memory traffic bottlenecks are carried out in Chapter 8.

6.1.2 Disabled admission control

For all experiments conducted in this section, the admission controller is disabled (i.e. No-AC). The remapping techniques presented in this section can be employed in conjunction with a soft real-time/best effort admission controller (e.g. Heu-AC). However, disabling the admission controller allows us to investigate the performance and behaviour of the remappers under heavy load conditions.

6.1.3 Job-level task mapping/remapping

In the previous chapters, runtime task mapping was constrained to the *stream-level*, in order to provide hard timing guarantees by the D-AC. The application model in this chapter assumes soft real-time video streams; hence, hard timing guarantees are not required. The allocation of tasks of each consecutive job of the same video stream can be changed/updated at runtime by the remapping algorithm if needed, to improve the performance/predictability. The mapping configuration in this chapter can therefore vary at the *job-level*.

A generic high-level view of the distributed task remapping process is illustrated in Figure 6.1. In this model, the tasks of the first job J_0 of a new stream VS_i are mapped and assigned priorities upon admission as per the previous chapters. For all work presented in this chapter, the initial mapping of the tasks follow the LU dynamic task mapping heuristic described in Section 5.6.1.1. Task priorities are assigned as per the video-centric priority assignment policy in Section 5.1.1.2. This initial MP&PR configuration is then saved onto the runtime task mapping table (TMT) and tasks of J_0 are dispatched to the respective PEs. When new subsequent jobs J_i of the stream VS_i are received, the task dispatcher queries the TMT to retrieve the mapping and priority assignment made for J_0. Hence, the same mapping and priority configuration is used for the remaining jobs in the stream unless the remapping procedure changes a task's allocation.

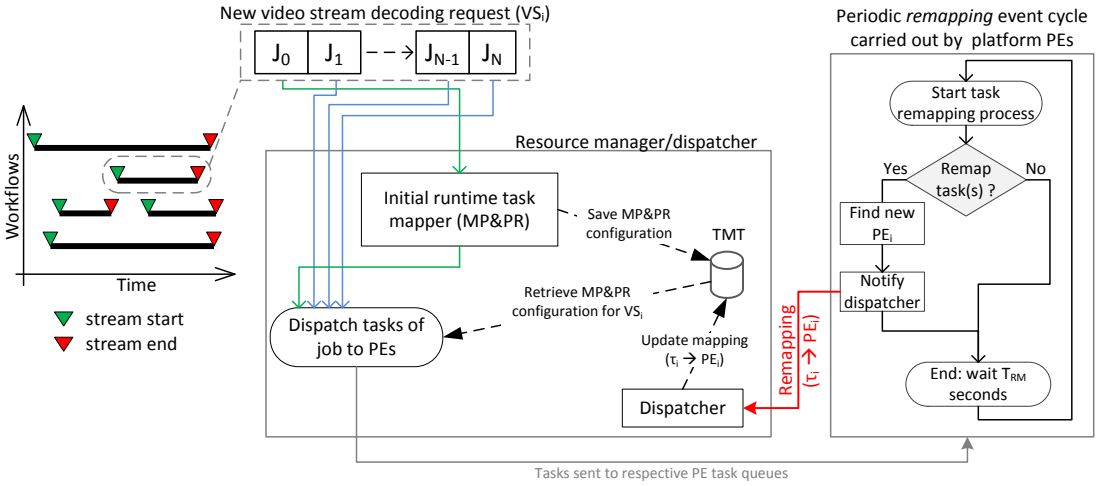


Figure 6.1: Video stream task remapping illustration

6.1.4 Low-overhead remapping procedure

Both remapping procedures presented in this chapter use a periodic remapping event and signalling scheme as shown in Figure 6.1. In the bio-inspired remapping technique any PE in the platform can notify the dispatcher regarding a task remapping, but in the cluster-based remapping technique only the cluster managers notify the dispatcher. The dispatcher looks up the task's id in the TMT and updates the corresponding PE with the new remapped PE id. When the next job is received the tasks are sent to the new PEs, as indicated by the updated TMT. Hence, remapping will only take effect from the subsequent arrival of the next job in the video stream.

Unlike the existing task migration techniques in the literature (Section 2.3.4), this novel remapping procedure does not require task code/data to be moved from one PE to another. Task remapping is simply an update to the TMT maintained by the dispatcher. The dispatcher notification message overhead is very minimal as its payload is only a few bytes and only transmits each remapping cycle.

6.2 Bio-inspired distributed task remapping

This section will first describe the mechanisms of the original pheromone-signalling (PS) based load balancing algorithm [104], henceforth referred to as *PSAlgo*. This algorithm is inspired by social-inspects (e.g. bees) and has previously been used to improve reliability of wireless sensor networks. Extensions to PSAlgo are made in this chapter in order to enable distributed remapping of late tasks from NoC PEs that are heavily utilised onto PEs that are under utilised in close proximity. The remapping algorithm parameter selection guidelines are also given in this section as well as a discussion of the difficulties faced with this technique.

6.2.1 Overview of the PS distributed load-balancing algorithm

An overview of PSAlgo was given in Section 2.3.2.4. To recap, each PE in the network can either be classified as a queen node (QNs) or worker node (WN). QNs periodically propagate pheromones (denoted *hd*) to neighbouring PEs. All nodes accumulate and pass on pheromones

received from the QNs to their neighbours, and the dose of the pheromone is decreased at every hop-distance away from the QN. The pheromone level (denoted h_i) for each 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.

The original PSAlgo [104], contained two periodic events and one asynchronous event. An additional remapping event has been added as part of the algorithm extensions. All events are shown by the sequence diagram in Figure 6.2 and these events are carried out by each PE in the NoC. The periodic *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 [104] and pseudo-code of these events are given in Appendix B.1.

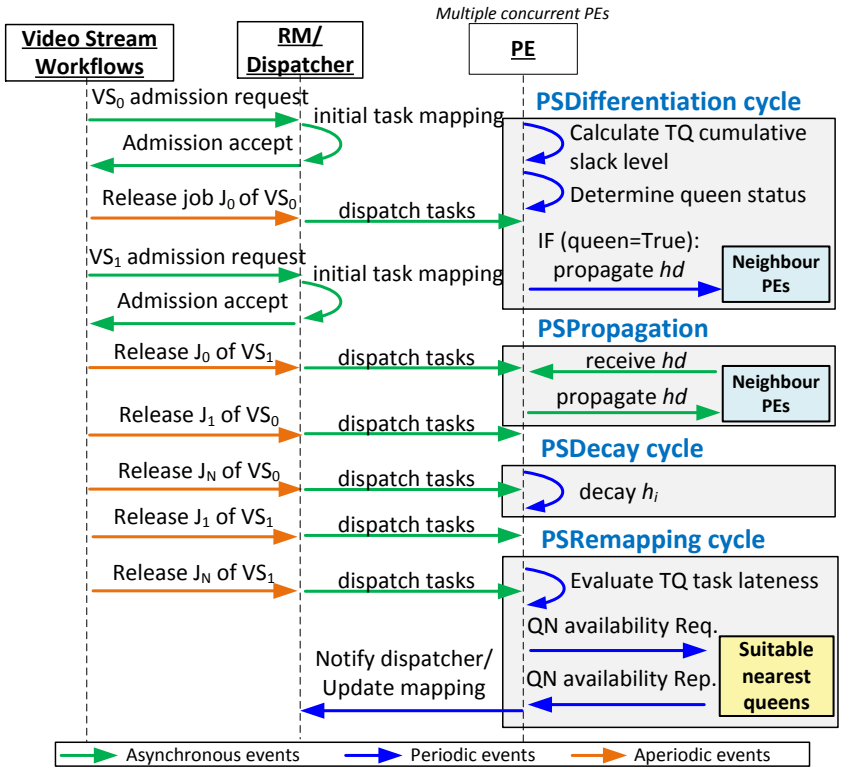


Figure 6.2: Sequence diagram of PSRM algorithm related events. Time triggered (periodic) : *PSDifferentiation*, *PSDecay* and *PSRemapping* cycles; Event triggered: *PSPropagation*

6.2.2 Extensions to the PS algorithm

In Figure 6.2, the event sequence of the bio-inspired remapping technique (termed *PSRM*) is presented. The PSPropagation and PSDecay cycles follow the original algorithms given in [104]. Figure 6.3 illustrates the hormone propagation when the hop distance limit is set to 3. At each hop the level of hormone in the pheromone message decreases, hence PEs further away from a QN will receive a lower hormone dose than those near to a QN. To reduce the communication overhead, each pheromone message only propagates in a forward direction. For example, in

Figure 6.3, PE(1,3) does not pass pheromones back to PE(1,2) and the QN does not receive its own hormone back, which it propagates. All control flow messages of PSRM including the dispatcher notification messages have a higher priority than data communication message flows.

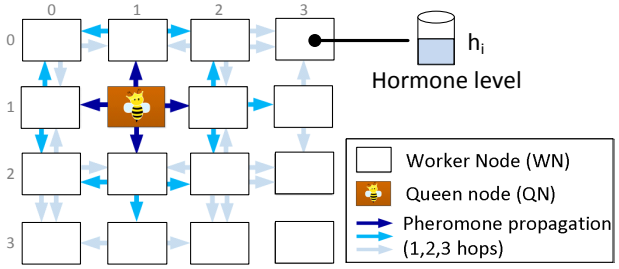


Figure 6.3: PSRM pheromone propagation

PSDifferentiation has been adapted to take into account the load of the tasks mapped onto a PE. An additional periodic remapping cycle (referred to as *PSRemapping*) has been added to determine the late tasks and remap them to new PEs. As shown in Figure 6.2, the system will continually receive jobs from active streams and new video decoding requests as well. The PSRM event cycles will occur in parallel to the videos/jobs arriving. The objective of these extensions are to allow WNs to remap late tasks of the video streams to a nearby QNs who have slack (idle temporal gaps) to accommodate more tasks.

Algorithm 6.1 shows the extensions made to the PSDifferentiation cycle of the PS algorithm. In this extension, the Q_{TH} is *dynamically adjusted* depending on the *normalised cumulative slack* (MPT_{slack}) of the tasks mapped on the PE. MPT_{slack} is calculated in line 2. A PE will become a QN if it has enough slack to accommodate additional tasks (lines 3-7). PEs keep track of a task's observed response time $(e_i^t - a_i^t)$ from its previous invocation, thereby enabling the estimation of the task slack $[d_i - (e_i^t - a_i^t)]$. A negative MPT_{slack} indicates the PE does not have any spare processing capacity to take additional tasks, and hence decreases the queen threshold level; thereby, decreasing the likelihood of the node becoming a QN. Likewise, a positive MPT_{slack} increases the queen threshold level. The amount in which Q_{TH} is incremented/decrement is controlled by the parameters Q_{TH}^{α} and Q_{TH}^{β}.

The self-organising behaviour of the distributed algorithm (specifically the PSDifferentiation), stabilises the number and position of the QNs in the system, as time progresses and depending on the workload. It should be emphasised that a QN will return into a WN state if it becomes overloaded. A node propagates pheromones immediately after it becomes a queen (line 10 of Algorithm 6.1). The pheromone dose (*hd*) is represented as a four element vector (line 10) containing the distance from the QN, the initial dosage (H_{QN}), the position of the QN (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.

6.2.2.1 PSRM task remapping cycle

Tasks incur lateness due to the resource over-utilisation and/or due to task/flow blocking. The goal is to change the mapping of the late tasks, such that these causes of lateness can be mitigated. The PSRemapping cycle is presented in Algorithm 6.2 and is executed by each PE periodically, using only its local knowledge gathered via the pheromone doses.

Algorithm 6.1: PSRM Algo [PSDifferentiation] (Periodic : every T_{QN})

Input : PE_{MPT} - tasks mapped onto the PE,
T_{QN} - QN differentiation period,
h_i - local pheromone level,
$Q_{TH}, Q_{TH}^{\alpha}, Q_{TH}^{\beta}$ - local QN threshold and increment/decrement factors,
Output: hd - updated pheromone dose,
$QueenStatus$ - QN status (boolean),

1 **while** *true* **do**
2 Calculate normalised cumulative slack of mapped tasks: $MPT_{slack} = \frac{\sum_{\forall \tau_i \in PE_{MPT}} [d_i - (e'_i - a'_i)]}{\sum_{\forall \tau_i \in PE_{MPT}} (d_i)}$;
/* calculate QN threshold based on the slack level */
3 **if** $MPT_{slack} > 0$ **then**
4 $Q_{TH} = Q_{TH} \times (1 + (MPT_{slack} \times Q_{TH}^{\alpha}))$;
5 **else**
6 $Q_{TH} = h_i \times Q_{TH}^{\beta}$;
7 **end**
/* determine queen status */
8 **if** $h_i < Q_{TH}$ **then**
9 QueenStatus = TRUE;
10 Broadcast $hd = \{0, H_{QN}, QN_{xy}, PE_{MPTinfo}\}$ to neighbouring PEs;
11 **else**
12 QueenStatus = FALSE;
13 **end**
14 wait for T_{QN};
15 **end**

The task with the maximum lateness τ_L^{MAX} is selected from the mapped tasks on the PE, as the task that needs to be remapped to a different PE (line 2). d_i is calculated using the DEQF scheme [137]. Each node is aware of the nearest QNs (denoted Q_{List}), and their mapped tasks, by storing the information received from each pheromone dose hd. A *balanced blocking heuristic* (similar to LWCRS in Chapter 5) is used to select a QN out of Q_{List} to remap τ_L^{MAX}. The heuristic attempts to find a QN, that can provide a lower blocking than the current blocking $B(\tau_L^{MAX})$ (lines 5-11), and also with the minimum number of low priority tasks which will be affected by the remapping. To avoid overloading QNs in a single remapping cycle, only a single WN can select a particular QN at a remapping event. Hence, the QN *availability* (lines 12-14) is checked before selection. There are situations where a QN can become overloaded; for example, when its amount of cumulative slack reduces over time after several remapping events. In these conditions, the QN will return to a WN state (as described by the PSDifferentiation cycle in Algorithm 6.1).

Finally, the task dispatcher is notified via a message flow to update the TMT. In the next job invocation the dispatcher will retrieve the updated mapping from the TMT and dispatch the tasks to the new PE. It should be made clear that even though there is an update message sent to a centralised dispatcher, the remapping decision is achieved in a decentralised manner, purely using local information at each PE.

Figure 6.4 illustrates an example of the remapping procedure in a 4x4 NoC. At a remapping event (step 1 of Figure 6.4), the PEs identify the late tasks mapped onto them. In step 2, the worker nodes PE(1,0) and PE(2,2) determines the suitability of each QN to remap the late tasks (τ_1 and τ_2). 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. Also in step 2, the nodes request for the suitable QN's availability; in this instance PE(1,0) obtains a lock on Q(1,1) first. Therefore, τ_1 will be remapped onto Q(1,1)

Algorithm 6.2: PSRM Algo [PSRemapping - Task remapping].
(Periodic : every T_{RM})

Input : PE_{MPT} - tasks mapped onto the PE,
T_{RM} - remapping period,
Q_{List} - list of QNs in close proximity and their mapped tasks
Output: Remapped tasks

while *true* **do**
  /* find most late task from mapped tasks */
  $\tau_i^{MAX\_L} = MAX(\{\tau_i \in PE_{MPT} \mid (a_i^t + d_i) \leq t_c\})$;
  /* get current blocking for late task */
  Current blocking: $B(\tau_i^{MAX\_L}) = \sum_{\forall \tau_j \in hp(\tau_i^{MAX\_L})} c_j$
  /* find suitable QNs which offer lower blocking, than current blocking */
  $Q_{List}^B = \{\}$
  foreach $Q_i \in Q_{List}$ **do**
    /* get target task blocking factor */
    Amount of self blocking if mapped to QN_i: $Self\_B_Q = \sum_{\forall \tau_j \in hp(\tau_i^{MAX\_L})} c_j$
    Amount of lower priority tasks mapped on QN_i: $LP_{size} = |lp(\tau_i^{MAX\_L})|$
    /* Insert into suitable QN list, only if lower than current blocking */
    if $Self\_B_Q < B(\tau_i^{MAX\_L})$ **then**
      Insert $\{Q_i, LP_{size}\}$ to Q_{List}^B
    end
  end
  /* request for QN availability - via request/reply message flows */
  $Avlb\_Q_{List}^B = requestAvailability(Q_{List}^B)$
  /* get available QN that has least amount of lower priority tasks */
  $\{Q_i^{MIN\_LP}, LP_Q^{MIN}\} = MIN(Avlb\_Q_{List}^B)$
  /* Update dispatcher's task mapping table */
  Notify dispatcher : $\tau_i^{MAX\_L} \rightarrow PE(Q_i^{MIN\_LP})$
  wait for T_{RM}
end

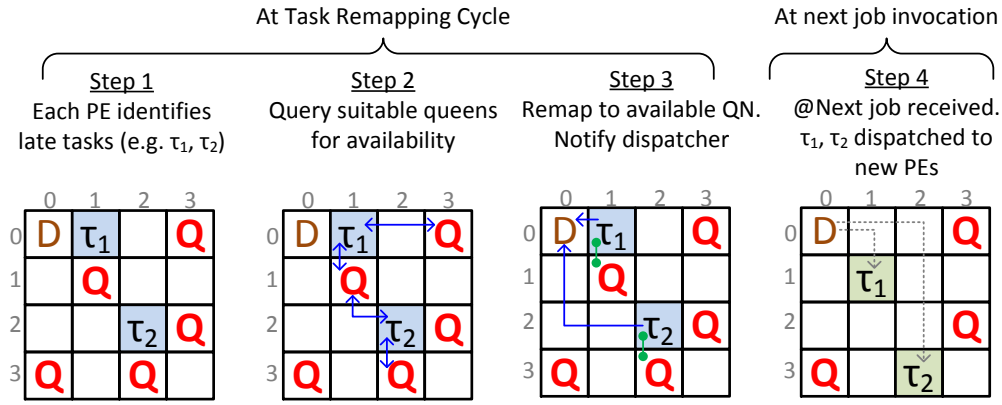


Figure 6.4: PSRM task remapping example. (Q=queen nodes; D=Dispatcher; $[\tau_1, \tau_2]$ are late tasks; Blue lines represent communication

and τ_2 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 the tasks are now dispatched to the new processing elements PE(1,1) and PE(2,3).

6.2.3 PSRM parameter selection

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. For this work, a simple random search based parameter tuning approach

(evaluated 150 unique random parameter combinations) proved sufficient to obtain reasonable performance. A more efficient parameter selection mechanism for PSAlgo can be found in [201] but relies upon an optimised simulation framework with lower accuracy to speed-up the fitness evaluation. Table 6.1 shows the parameters obtained via a simple random search based selection method. Results for all tested random parameter combinations can be found in Appendix B.2.

Table 6.1: PSRM algorithm parameters - for a 10×10 NoC

PSRM parameter	**Value**
Differenciation cycle period (T_{QN})	0.94 (s)
Decay cycle period (T_{DECAY})	0.313 (s)
Remapping period (T_{RM})	3.8 (s)
Default QN threshold (Q_{TH})	20
Initial hormone dose by QN (H_{QN})	5
QN threshold inc./dec. factors (Q^{α}_{TH}, Q^{β}_{TH})	0.203, 0.3
Pheromone time and hop decay factors	0.2, 0.217
Pheromone propagation range (pd)	2 (hops)

Several intuitive guidelines were derived from the parameter tuning. T_{QN} must be smaller than T_{RM}, but high enough not to cause significant signalling overhead in the communication network. T_{DECAY} should be 2-4 times lower than T_{QN}. The remapping period (T_{RM}) should be much larger than the job minimum inter-arrival rate, to ensure QN position stabilisation. A lower T_{RM} results in the dispatcher notification overhead to cause network congestion. Likewise, the QN hormone propagation range should be kept low to reduce the protocol communication overhead but at the same time giving sufficient capability to discover nearby QNs. The Q^{α}_{TH} and Q^{β}_{TH} factors must be kept low to reduce large variations in h_i. The relationship between the original PSAlgo parameters have been investigated extensively in previous work [104, 201].

6.2.4 Complexity and overhead analysis of PSRM

PSRM incurs very low computation complexity in the PSPropagation and PSDecay cycles as each are a short sequence of simple ALU operations. The PSDifferentiation has $O(|MPT|)$ worst-case complexity where $|MPT|$ denotes the number of tasks mapped on the PE. The PSRemapping has a worst-case complexity of $O(|MPT| \times |Q_{list}|)$ where, $|Q_{list}|$ is bounded by the propagation range. However, unlike cluster-based or centralised management approaches, each PE carries out the resource management event cycles, hence the complexity variables $|MPT|$ and $|Q_{list}|$ are very small. The communication overhead of the PSDifferentiation and PSPropagation events are directly proportional to the number of QNs in the network, the propagation range pd and the differentiation period T_{QN}. The number of hd messages injected into the NoC due to hormone propagation by a single QN and its hc hop neighbours can be estimated by Eq. (6.1). Msg^{hd}_{hc} denotes the number of messages generated at hop count hc. Each hd has a very small payload (16 bytes) compared to the inter-task data message payload. In PSRM, each WN helps to propagate pheromone messages, therefore the PS messages are of 1 hop each; when a WN receives a PS message from another WN or a QN it creates another message and passes it along to its own neighbours. Therefore, the PS algorithm inherently avoids message flows with long routes. The propagation range (pd) parameter determines how far a specific PS message emitted by a QN can travel. Larger pd values can significantly increase the congestion in the network but improves QN discovery. A smaller pd can limit the interference to the data traffic.

$$\text{Number of } hd \text{ messages (by 1 QN): } Msg_{hc}^{hd} = \begin{cases} 4 & \text{if } hc = 1 \\ Msg_{hc-1}^{hd} + [4 \times 3^{(hc-1)}] & \text{if } 1 < hc \leq pd \end{cases} \tag{6.1}$$

The amount of workload admitted into the system will marginally affect the position and number of QNs in a network but the worst-case scenario where all PEs are QNs would only occur in very low load conditions. The dispatcher notification communication overhead is small compared to the pheromone propagation overhead as the T_{RM} is much larger than T_{QN}. Further results and discussion of the communication overhead of PSAlgo with respect to the network size and T_{QN} can be found in [202].

6.2.5 Challenges of PSRM

The primary drawback of PSRM (and any other fully distributed management techniques) is lack of global state of the system. The remapping heuristic cannot take into account the change in the *network contention patterns* caused by the reallocation. Therefore, there are situations where remapping a task can result in an increase in the lateness. The second drawback of this technique is related to event synchronisation. Between consecutive PSDifferentiation events, the QN's load can change rapidly when the system is heavily utilised. This 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.

Lastly, PSRM is sensitive to the parameters used and the network size. The parameters need to be tuned for a specific network size and ideally for a known workload profile to obtain better results. In this work however, a fixed/generic parameter set (Table 6.1) was derived (via tuning) for a range of random workload profiles (i.e. different arrival patterns, execution costs, resolutions etc.)

6.2.6 Application of PSRM

This section reflects upon the applicability of PSRM and describes scenarios and conditions where PSRM can deliver good performance and where it might be less able to meet expectations. PSRM's performance is dependent on the type of workload scenario. If the workload is less diverse and more characteristics of the workload are known beforehand, the PSRM parameter tuning can yield a better performing task remapper. For example, if the actual workload scenario consists of a fixed number of streams and workflows, with only one or two video resolutions then the diversity of the workload can be significantly reduced. Similarly, long-running video streams rather than short videos are more desirable as the pheromone-signalling algorithm may take several differentiation cycles to stabilise and a suitable number of nodes become QNs. The type of scenario described above is commonly seen when the properties of the streams are controlled by the same video decoding service (e.g. multi-stream real-time video

surveillance, multi-stream broadcast video monitoring systems). However, note that when evaluating the remapping techniques presented in this chapter, a random set of workload conditions are induced to cover a range of possible application scenarios.

PSRM can also perform better for video streams with less number of B-frames or bi-directional predicted coding disabled completely. In these streams, the task-graphs (i.e. GoPs structures) will have less number of edges, leading to lower communication between tasks. In such kinds of workloads, the negative impact of task reallocation due to not considering network contention (as described in Section 6.2.6), can be made less severe.

The level of lateness of the workload also has an impact on the effectiveness of PSRM. The availability of QNs is dependent on the amount of available slack in a node's task queue. Therefore, if the system is heavily loaded and over-saturated, none of the nodes may have available slack; this can gradually decrease the number of QNs, leading to short intervals of poor PSRM performance. Of course, a simple heuristic based system admission-controller can be then be used in conjunction with PSRM, to avoid encountering such overload situations.

PSRM's performance is also dependent on the scale of the platform and the workload. In smaller platform sizes (e.g. 4×4 and lower) the communication overhead of PSRM can outweigh the performance benefits, therefore it is mostly suitable for large scale platforms. Likewise, for smaller workloads (e.g. when the number of parallel streams are significantly smaller than the number of cores on the platform), using a simple first-fit load-balancing initial mapping heuristic may be sufficient.

6.3 Cluster-based distributed task remapping

In this section a cluster-based remapping technique is described, which is used as a baseline to evaluate PSRM. Section 2.3.2.2 provided a brief overview of the cluster-based resource management technique (referred to as CCPRM_{V1}) introduced by Castilhos et al. [157]. In their work, the NoC is partitioned into virtual clusters and each cluster has a local manager (LMP) which manages the slave PEs in its cluster. LMPs perform frequent monitoring on the PEs to obtain knowledge about a cluster. There is also a global manager (GMP) which performs duties similar to the centralised dispatcher. As illustrated in Figure 2.13, the LMPs carry out a *3-way request/reply/release* interaction protocol to find resources in other clusters when there are no slave PEs available in the current cluster to service a task. Similarly, in this work, during a periodic remapping event the LMPs first try to remap a late task within its own cluster; failing which, the LMP communicates with the other LMPs to find an available slave PE in a remote cluster. Here, *availability* refers to a slave PE with lower utilisation. Similar to PSRM, the dispatcher is notified of any task remapping. By doing so, the overall lateness of the video stream jobs can be reduced.

Unlike in [157], this adaptation does not vary the cluster size at runtime or perform task migration. The LMPs perform task execution similar to the slave PEs as well as carry out monitoring and periodic task remapping procedures. The resource management execution overhead of LMPs is not taken in to account in the model. Similar to PSRM, all control flow messages of CCPRM_{V1} (including the dispatcher notification messages) have a higher priority than data communication message flows.

Table 6.2: $CCPRM_{V2}$ algorithm parameters - for a 10×10 NoC

$CCPRM_{V2}$ parameter	**Value**
Cluster size	2×5
Remapping period (T_{RM})	7.2
Max. late tasks (per cluster) to remap	5

6.3.1 Improvements to $CCPRM_{V1}$

Preliminary experiments showed several limitations of the original $CCPRM_{V1}$ algorithm, which are addressed as follows in the improved version of the algorithm, denoted $CCPRM_{V2}$.

- The task relocation distance (previously unconstrained) is now limited to 2 hops to reduce long inter-task communication routes, which will cause higher interference in the NoC.
- In $CCPRM_{V1}$, late tasks were remapped to a new slave PE (either in own or remote cluster) following the LU heuristic. In $CCPRM_{V2}$, this is improved by integrating the balanced-blocking heuristic used in PSRM (lines 4-10 of Algorithm 6.2) to select the new slave PE. Furthermore, $CCPRM_{V2}$ ensures that a late task is only remapped to a PE with positive cumulative slack.
- In [157], the original location of the LMP is at the corner of the cluster, as illustrated in Figure 2.13(b). In order to reduce long monitoring traffic flow routes, the LMP is placed at the center of the cluster.

6.3.2 $CCPRM_{V2}$ parameter selection

$CCPRM_{V2}$ has 3 important parameters: cluster-size, remapping period and number of late tasks to remap. The parameters given in Table 6.2 were selected as the best after a parameter search, equivalent to the random search carried out for PSRM (Section 6.2.3). Large cluster sizes result in more monitoring traffic transmitted to the LMP and longer communication routes. A larger number of late tasks to remap would cause the heavy variation in the load which in some cases may be undesirable. Similar to 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.

6.3.3 Complexity and overhead analysis of $CCPRM_{V2}$

In $CCPRM_{V2}$, the computation overhead is incurred only by the LMPs in each cluster. At each remapping cycle, the search for a local cluster slave PE has $O(|cluster| \times |MPT|)$ worst-case complexity. A similar worst-case complexity is taken to select a slave PE when an LMP receives a loan request. The RM communication overhead of $CCPRM_{V2}$ is due to two factors: the cluster monitoring overhead and the loan request/reply/response protocol. Slave PEs notify its cluster LMP when a task completes, hence the monitoring overhead is directly proportional to the amount of workload (i.e. number of tasks admitted). The loan request and reply interactions (steps 1 and 2 of Figure 2.13) incur $(|LMPs| \times (|LMPs| - 1))$ message flows and the loan release (steps 3 of Figure 2.13) takes $(|LMPs| \times (|LMPs| - 2))$ messages. $|LMPs|$ denotes the number of LMPs in a network. Hence, the inter-LMP interaction is directly proportional to the number of clusters in the NoC. The loan messages and monitoring feedback messages may have long routes depending on the distance of the slave nodes and LMPs. Both the monitoring message payload

(i.e. slave to LMP) and the inter-LMP loan communication protocol payload is assumed to be 16 bytes.

6.4 Evaluation

This evaluation section is separated into primary and secondary experiments as follows:

- Primary evaluation ***(ExpA)***: evaluate predictability and performance of PSRM against the baseline remapping techniques.
- Secondary investigation of PSRM ***(ExpB)***: to evaluate the impact on job lateness with memory modelling disabled vs. enabled.

6.4.1 Experimental design

The abstract simulation framework as described in the previous chapters is used in these experiments with the application and platform characteristics as defined in Section 6.1. All experiments assumed a 10×10 NoC platform. The PEs are assumed to have an operating frequency of 200MHz. Message flow header routing cost is assumed to be 7 clock-cycles, NoC frequency is set at 500MHz and the link width is set to 16 bytes. The task remapping protocols evaluated in this experiment (e.g. PSRM and CCPRM) will inject a high number of control message flows into the network, which will result in a high amount of network contention if the NoC bandwidth is low. As explained in Section 3.3, the simulation speed can decrease under heavy contention scenarios. Therefore, a relatively higher NoC frequency is selected to balance contention and simulation speed.

The level of workload was configured such that there would be a peak load of 103 parallel video streams (slightly more videos to the number of total PEs). Video stream resolutions were selected at random from a range of resolutions (e.g. high: 720×576, low: 230×180), with 1 video per workflow, maximum 5 jobs per video. Each experiment was run for 30 unique seeds. Admission control is disabled (i.e. No-AC), and an open-loop task dispatching and resource management process is carried out in all experiments (similar to Section 5.1.2.2). Table 6.3 summarise the experimental conditions, parameters and metrics used to explore the aforementioned evaluation objectives.

Job lateness improvement, video stream schedulability, resource management communication overhead and PE busy time distribution are the main response variables measured in the experiments. The *job lateness improvement* indicates the reduction of the cumulative job lateness (C_L^{Jobs}) when remapping is used (Section 3.2.1). The *RM communication overhead* of all remappers include the protocol control messages as well as the dispatcher notification message flows (Section 3.2.4).

6.4.1.1 Baseline remapping techniques:

The following remapping techniques are used as baselines to evaluate PSRM:

- CCPRM_{V1} - the original cluster-based remapping technique, with a cluster size of 2×5 (i.e 10 clusters).

Table 6.3: Experimental design summary of evaluating remapping techniques

Eval. obj.	Independent variables	Response variables
ExpA	PSRM, $CCPRM_{V1}$ (2×5 cluster size), $CCPRM_{V2}$ (2×5 cluster size), Centralised management, Random remapper	Job lateness improvement, Number of schedulable video streams, RM communication overhead, PE busy time distribution
ExpB	PSRM: memory modelling enabled/disabled	Job lateness improvement

- $CCPRM_{V2}$ - the improved cluster-based remapping technique, with a cluster size of 2×5 (i.e 10 clusters). The modifications as outlined in Section 6.3.1 are included in this cluster based remapper.
- *Centralised management* - this is essentially $CCPRM_{V2}$ with a single 10×10 cluster. A single manager receives status updated from every slave PE in the network and performs periodic remapping. The central manager notifies the task dispatcher of any remapping decisions.
- *Random remapper* - every remapping event each PE selects the most late task in its task queue and randomly selects another PE to remap to. The task dispatcher is notified of the remapping event.

6.4.2 Results discussion

6.4.2.1 ExpA - Comparison of PSRM and baseline remapping techniques

In ExpA, the predictability and communication overhead of the remapping techniques are evaluated. Figure 6.5 shows the distribution of cumulative job lateness improvement for each of the remapping techniques evaluated. Each sample in the distribution corresponds to the C_L^{Jobs} improvement in a unique seeded simulation run. A positive job lateness improvement indicates that task remapping has helped to reduce the cumulative job lateness of the video streams. Negative improvement indicates that the remapping has instead worsened the lateness of the jobs, due to the unpredictable network contention changes that occur due to task reallocation (Section 6.2.6). All the techniques show both negative and positive improvements. Unlike the baseline remappers, a majority of the PSRM distribution lies in the positive improvement region. However, only a maximum of 2.6% positive lateness improvement can be achieved using PSRM, while ≈5% improvement can be achieved if $CPRM_{V2}$ is used. The narrower spread in the PSRM job lateness distribution indicates that PSRM can offer lower variability in timeliness (i.e. better predictability) compared to the cluster-based and random remapping baselines. The results in Figure 6.5 also show the minor improvement of $CCPRM_{V2}$ over $CCPRM_{V1}$. The modifications enable $CCPRM_{V2}$ to marginally decrease the inter-quartile range (IQR) variability and increase the mean and maximum positive job lateness improvement over $CCPRM_{V1}$. The central manager shows the lowest amount of job lateness variability even though majority of the distribution falls in the negative lateness region. Random mapping performs poorly as most of its improvement lies in the negative region, even though it managed to achieve the highest job lateness improvement (≈6%) out of the evaluated remappers.

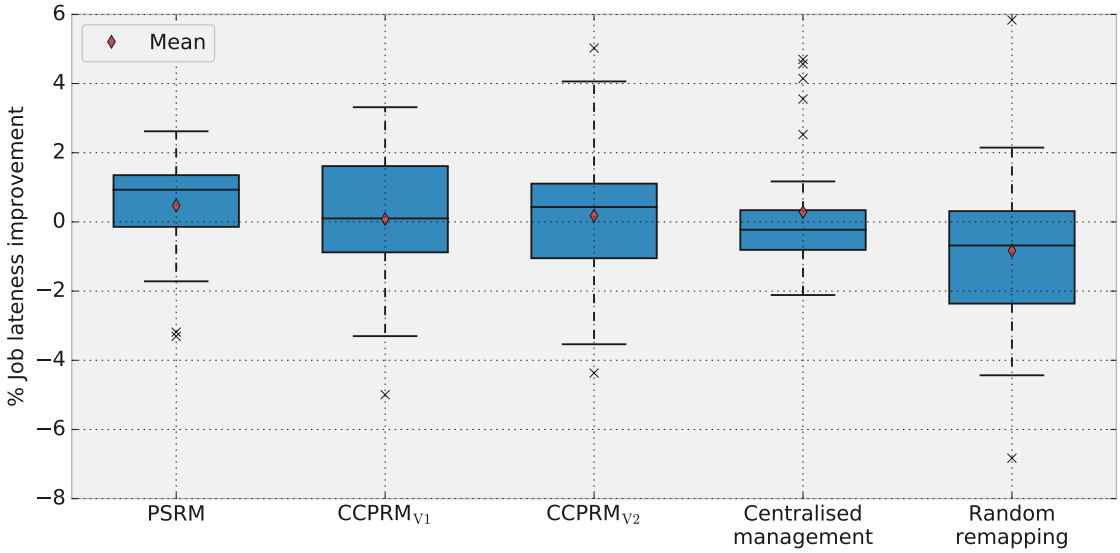


Figure 6.5: PSRM vs. baselines evaluation - job lateness improvement %

Table 6.4 shows the number of schedulable video streams and the total number of job deadline misses (of the unschedulable streams), for all remapping techniques evaluated. A video stream is considered unschedulable if at least 1 or more of its jobs miss their deadlines. A higher number of schedulable video streams and a lower number of job deadline misses represent better predictability. Therefore, both the results in Figure 6.5 and Table 6.4, have to be considered in conjunction to better investigate the predictability of the remapping techniques. PSRM shows marginally higher number of schedulable streams and lower deadline misses than the baselines. The improvement of CCPRM$_{V2}$ over CCPRM$_{V1}$ is more visible in the metrics given in Table 6.4. The central manager has a higher number of fully schedulable video streams than CCPRM$_{V2}$, but at the cost of increasing the job lateness of the unschedulable videos.

Table 6.4: Video stream schedulability results for different remapping types, across all seeded experimental runs

Remapper type	**Cumulative # schedulable video streams (all seeds)**	**Cumulative # job deadline misses (all seeds)**
PSRM	11227	12163
CCPRM$_{V1}$	11208	12247
CCPRM$_{V2}$	11216	12180
Centralised	11219	12206
Random	11209	12239

Figure 6.6 shows the distribution of RM communication overhead of the remapping techniques. Different aspects of the communication overhead are measured. The cumulative basic latency, cumulative payload, total number of message flows and number of hops traversed of control flows are indicated in Figure 6.6. PSRM shows a significant communication overhead reduction in all factors, when compared to CCPRM$_{V2}$, CCPRM$_{V1}$ and centralised management. Both PSRM and the cluster-based remappers on average utilise only a few links (i.e. hops) per control flow; however as the cluster-based remapping/management techniques require 3 times more control flows to operate, their overall communication overhead significantly increases. The slave monitoring feedback messages form majority of the control messages of the cluster-based remappers.

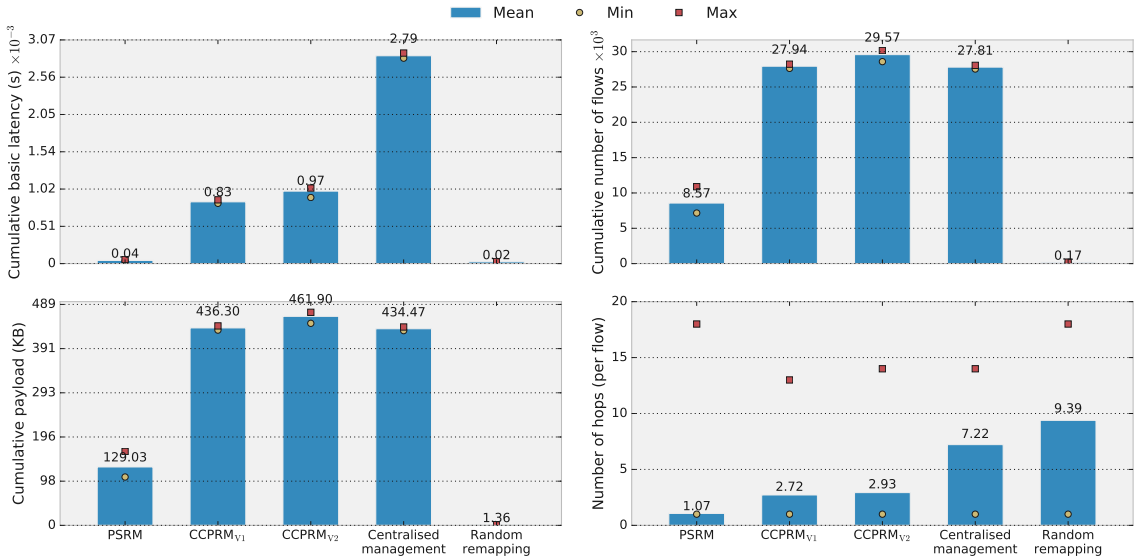


Figure 6.6: PSRM vs. baselines evaluation - RM communication overhead factors (mean/max/min statistics shown across all seeds). *Top-left:* cumulative basic latencies, *Top-right:* cumulative number of control flows injected, *Bottom-left:* cumulative payload of all control flows, *Bottom-right:* number of hops per control flow

Central management has the higher communication overhead, in terms of cumulative basic latencies even though it has slightly lower cumulative payload and total number of control flows compared to the cluster-based remappers. This is mainly due to the long communication routes (high hop count) used by the control message flows; slave PEs further away from the management node will take longer routes to send monitoring feedback. Route lengths will increase significantly, as the NoC size increases, and the number of flows and cumulative payload will increase as the level of workload increases. As shown in previous work (e.g. [36]), the slave to central manager communication will become a severe performance bottleneck for NoC sizes larger than 12×12. Random remapping has the lowest overall communication overhead, as it incurs only control flows to notify the dispatcher. However, it has a high number of hops per control flow, thus interfering more data traffic flows than the other remappers.

The PE busy time distribution shown in Figure 6.7a, denotes the PEs with higher mean busy times using lighter shades whilst the darker shades show PEs with low busy levels. The data shown in this plot are normalised such that each remapping technique is relative to each other. An ideal/balanced workload distribution would be one in which all PEs represent a similar shade. A histogram view of the mean PE busy time is shown in Figure 6.7b, where the PE busy times measurements are categorised into equal sized utilisation levels. The standard deviation (σ) and mean statistics of the PE utilisation distribution across all nodes is given in Table 6.5. Remappers with lower σ values have a more balanced workload distribution than higher σ values. Overall all remappers show a similar workload distribution pattern, with a majority of the top region of the NoC utilised more than the bottom region; due to the nature of the initial mapping. PSRM does not have any nodes at very high utilisation levels (i.e. 90%-100%) unlike the baselines but have several nodes which have very low utilisation levels. PSRM also has comparable workload distribution variability to the cluster-based remappers. The centralised management show a marginally better workload distribution than the other remappers indicated by its low standard deviation.

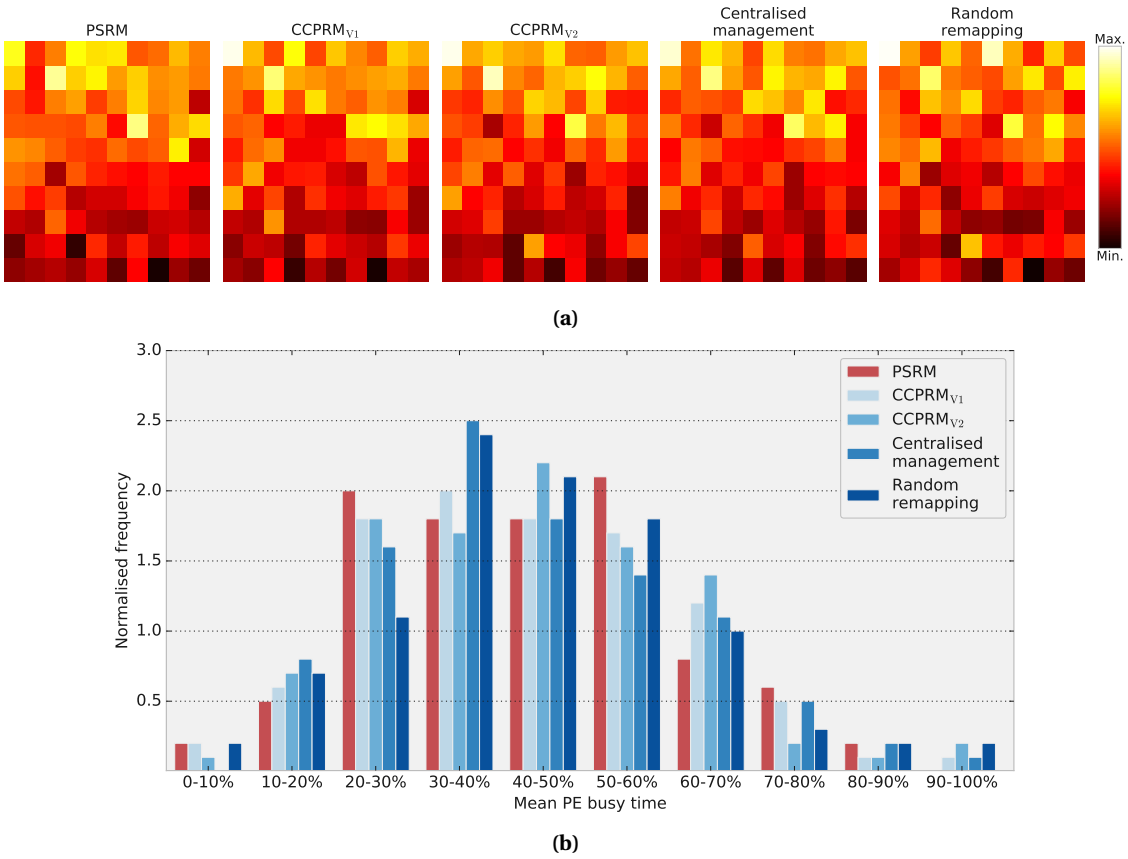


Figure 6.7: PSRM vs. baselines evaluation: (a) 2D visualisation of the normalised mean PE busy time distribution (10×10 NoC), (b) Histogram of the normalised PE busy time distribution

Table 6.5: PE busy time distribution results (average over all experimental runs)

Remapper type	Std. dev (σ) of PE busy time	Mean PE busy time %
PSRM	0.1094	43.67
CCPRM$_{V1}$	0.1094	43.18
CCPRM$_{V2}$	0.1097	43.07
Centralised	0.1078	43.07
Random	0.1108	43.36

6.4.2.2 ExpB - Effect of memory modelling on PSRM

In ExpB, the effect of modelling the memory traffic on the PSRM remapping technique is investigated. The distribution of job lateness improvements when PSRM is used is shown in Figure 6.8, when memory transactions modelling is enabled and disabled. Both treatments use equivalent parameters for PSRM. NoC usage is reduced when two or more communicating tasks are mapped/remapped on to a single PE but the memory transactions would still occur over the NoC. Due to this reason, when memory modelling is enabled the performance of the PSRM technique slightly worsens. The maximum negative lateness improvement of PSRM increases from ≈-3.2% to ≈-5.2%, when memory traffic is modelled. However, note that as the simulation time significantly increases when memory traffic is modelled, the parameters used for both treatments were the same as shown in Table 6.1. Hence, the performance of PSRM (with mem. enabled) can be improved further by parameter tuning. From these results it can be inferred that, the simulations can marginally loose accuracy (by about 1-2%) by not modelling

memory traffic, but gain the benefit of almost doubling the simulation speed. The remaining experimental evaluations do not consider memory traffic modelling.

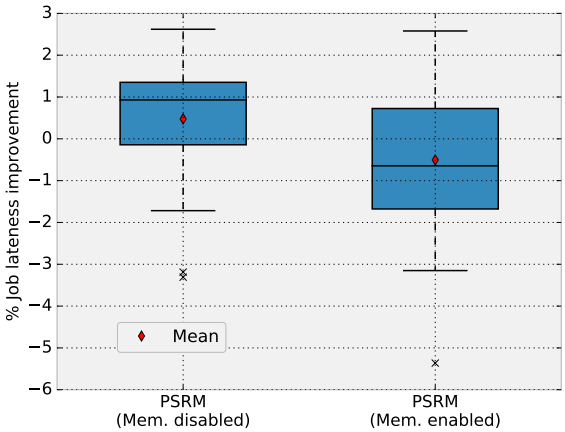


Figure 6.8: PSRM evaluation memory disabled vs. enabled

6.4.2.3 Results summary

Overall, the results indicate the PSRM fully distributed remapping technique helps to reduce lateness in the video stream jobs and to marginally increase the number of schedulable video streams (i.e. improve predictability) when compared with the the baseline remappers. The maximum job lateness improvement achieved by PSRM is very small (2.6%) but shows a reduction in job lateness improvement variability when compared with the cluster-based and random remapping techniques. Lower timing variability is desirable in the context of predictable systems. However, this minor improvement in job lateness comes at a much lower communication overhead than the cluster-based and central management baselines. On average, PSRM uses 3 times less control message flows, 3 times less cumulative payload and about half the average number of links than the $CCPRM_{V2}$ cluster based remapper. These differences results in a large overall reduction in RM communication overhead in terms of cumulative basic latency.

The centralised manager dominates over the cluster-based remappers in terms of number of schedulable video streams. It also performs the best amongst the evaluated remappers in reducing the job lateness improvement variability and shows the best balanced PE workload distribution. However, central management incurs the largest amount of overall RM communication overhead, as it requires a high number of control message flows, each of which on average have long routes. This limitation of the central manager becomes a serious issue as the NoC size scales. Random remapping on the other hand, shows the lowest communication overhead but performs poorly in terms of predictability.

6.5 Summary and novel contributions

To summarise, this chapter investigated the following two runtime task remapping techniques, in order to improve the predictability and performance of SRT video stream decoding:

- PSRM - a fully distributed, low overhead, bio-inspired task remapping technique. This is an extension of the pheromone signalling based load balancing technique introduced in [104]. It was adapted to perform dynamic task remapping to improve job lateness in NoC based platforms.

- $CCPRM_{V2}$ - a cluster-based task remapping technique for NoCs is presented as a direct baseline to PSRM. This was first introduced as a hierarchical resource management technique in [157] and is adapted/improved in this work to facilitate task remapping and to take into account task blocking behaviour.

Both presented techniques make use of a balanced blocking heuristic similar to the LWCRS task mapper (in Chapter 5) when selecting a new PE to remap a late task. As discussed in Section 6.1.3, unlike in existing state-of-the-art dynamic resource management techniques, the remapping procedure in this chapter do not migrate tasks, therefore they have a much lower communication overhead.

PSRM relies on local knowledge of each PE to make decentralised remapping decisions. In comparison, the cluster-based approach has a broader view of the NoC state, as the LMPs monitor each slave PE in the cluster and interact with other LMPs to make a remapping decision. The two remappers are evaluated with a centralised remapping scheme as well as a random remapper. Evaluation results show that the proposed fully distributed PSRM remapping technique is marginally better in terms of predictability when compared to the baseline remappers. However, PSRM incurs much less communication overhead than the cluster-based/centralised remappers and comparable PE workload distribution to the cluster-based management approach. The results also suggest that it is possible to employ runtime task remapping to improve the initial mapping, albeit not in all workload conditions, due to unpredictable NoC traffic contention patterns.

Chapter 7

Extending the application model to support modern video coding tools and standards

The previous chapters presented dynamic resource management methods on video streams encoded using *classical codecs* (e.g. MPEG-2). Even though MPEG-2 is still being used to encode standard definition video streams, *modern codecs* such as HEVC are gradually being adopted to address the compression challenges faced with ultra-high definition video streaming. With the adoption of HEVC, both the encoder and decoder have increased in computation complexity and memory requirements. Moreover, modern encoding algorithms such as the use of hierarchical B-frame structures, random-access profiles, adaptive scene change detection, multiple reference frames and more variability in block sizes makes video decoding workloads highly dynamic and complex. To address these challenges, better resource management techniques are required. At early stages of the design space exploration process, simulation-based investigation is a common practice. Therefore, it is crucial to have tractable, realistic and abstract models of the actual application workload.

The work in this chapter is motivated by the lack of data-parallel, HEVC video decoding workload models in the literature (Section 2.1.5.1). The application model used in the previous chapters are extended in this chapter to model a higher degree of variability in the video stream workload characteristics. These models will then be used in the following chapter to investigate runtime task mapping techniques for parallel HEVC decoding.

Trace-driven analysis of real HEVC decoding workloads are carried out at the GoP, frame and coding unit (CU) levels. DAG-based synthetic HEVC decoding workload generation algorithms are presented, that use statistical distribution models that closely represent video decoding workload characteristics. To the best of the author's knowledge, this is the first work to characterise HEVC decoding workloads at the block-level as well as capturing the properties of GoP-level task graphs dependency patterns, for different types of video streams.

The remainder of the chapter is organised as follows. Firstly the refinements introduced to the application model are clearly indicated in Section 7.1. Section 7.2 describes the methodology used to analyse HEVC video streams and generate synthetic workloads. A description of the video streams used and the codec set-up is given in Section 7.3. A GoP-level analysis is conducted in Section 7.4 followed by a CU level analysis in Section 7.5 and a communication volume analysis in Section 7.6. Breakdown of the CPU and memory usage of the decoder is discussed in Section 7.7. In Section 7.8 the algorithms to generate the synthetic HEVC decoding workloads are presented. Finally, an evaluation of the presented workload generation technique

is carried out in Section 7.9.

7.1 Application model refinements

The work in this chapter lifts certain assumptions made in the application model used in the previous chapters.

- *Dynamic GoP structures:* the GoP frame pattern and dependency structure no longer will follow the 12 frame structure given in Section 3.1.1. The number of frames and their dependency patterns within a job can now change between consecutive jobs in the same video stream.
- *Task execution cost:* the frame decoding task execution cost will be derived via an accumulation of lower-granularity CU decoding costs. In the previous model (Section 3.1.1.1), the block-level decoding cost was assumed to be fixed and the number of different blocks per frame were assumed to be random. In this refinement, the CU decoding cost will be sampled from an analytical distribution representing the real CU decoding time. The type and number of CUs within the frame model will also be representative of real streams. Execution time at the CU-level granularity will facilitate deriving even a coarser-grain, Tile-level decoding cost.

  This CU-level decoding information will be used in the following chapter to investigate HEVC tile-level task mapping.
- *Dynamic task reference data:* the model in Section 3.1.1 assumes a fixed reference data payload. This is not realistic as the amount of reference data will vary depending on the motion in the video. From this chapter onwards, the model assumes a variable amount of reference data volume per inter-frame.

7.2 HEVC stream analysis and synthetic workload generation methodology

Similar to other video coding standards, HEVC has a hierarchical structure (Section 2.1.2). CUs exist at the lowest granularity and frames and GoPs are at the highest level of the hierarchy. Different characteristics at each level contribute to the complexity of the decoding process. For example, depending on the motion in the video the number of P and B frames in a GoP as well as their inter-frame dependencies can vary. CU-level granularity analysis is required in order to derive an accurate frame decoding model. For example, when modelling an I-frame, a higher proportion of smaller CU sizes may need to be used, while having no P/B/Skip-CUs. A fine-grain workload characterisation is also beneficial for design space exploration of CU/WPP/Tile-level parallel decoding of HEVC streams.

Figure 7.1 illustrates HEVC workload characterisation methodology which includes the statistical analysis of real HEVC video streams and the synthetic HEVC decoding workload generation. Video sequences of varying content are first obtained and HEVC encoded. Encoder settings are configured to represent broadcast related streams suited for decoding on resource constrained systems. During the encoding and decoding process, the following characteristics of the different video streams are captured and analysed:

- **GoP-level characteristics:** Unlike the application model used in the previous chapters, this model considers dynamic/adaptive GoP dependency patterns. The distribution of the number of P/B frames in a video stream with respect to the scene content are captured per video stream. Reference frame distances (i.e. distance between parent and child frame decoding tasks in a GoP) and hierarchical/contiguous B-frame groupings are analysed.
- **Frame and CU-level characteristics:** CU characteristics such as distribution of CU sizes, types per intra/inter frame are captured. When decoding the streams, CU-level decoding time is profiled and analysed in order to derive a frame-level decoding cost (i.e. the weight of the nodes in the task graph)
- **Inter-task communication characteristics:** Reference data volume per frame/CU-type (i.e. inter-task communication flow payload) and encoded frame size analysis (i.e. task memory read traffic payload) are captured.

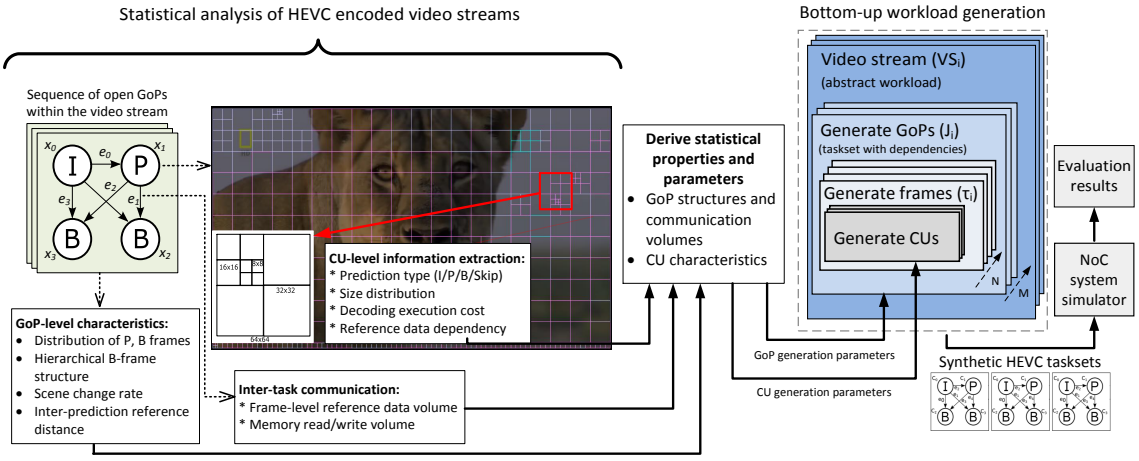


Figure 7.1: DAG-based HEVC workload generation methodology

The statistical properties obtained from the video stream trace analysis are then used as parameters for the workload generation process. The bottom-up workload generator builds the task sets from a lower granularity (i.e. CU-level) up to a higher granularity (frames, GoPs). Multiple CUs are generated to construct a frame and multiple frames combined according to the inter-frame reference pattern, to generate a GoP.

7.3 Video sequences and codec tools under investigation

The video sequences under investigation were chosen to represent varying levels of spatial and temporal video characteristics. The selected video streams represent typical live (e.g. sport, speech) or on-demand (e.g. movie, documentary) broadcasting material. Below are the video sequences selected for this study (snapshots presented in Figure 7.2):

- **FastFurious5** (Action, 720p, 30fps, 15mins): Heavy panning/camera movement, frequent scene changes.
- **LionWildlife** (Documentary, 720p, 30fps, 15mins): Natural scenery, medium movement scenes, fade in/out, grayscale to colour transitions.

- **Football** (Sport, 720p, 30fps, 15mins): camera perspective mostly on field, camera panning, occasional close-ups on players/spectators. Large amounts of common single colour background, combined text and video.
- **ObamaSpeech** (Speech footage, 720p, 24fps, 10mins): Constant, non-uniform background; uni-camera and single person perspective, head/shoulder movement.
- **BigBuckBunny** (CGI/Animation, 480p, 25fps, 9mins): Wide range of colours, moderate scene changes.
- **ColouredNoise** (Pseudo Random coloured pixels, 780p, 25fps, 10mins): Low compression efficiency, useful for analysing *worst-case* characteristics of an encoder/decoder.

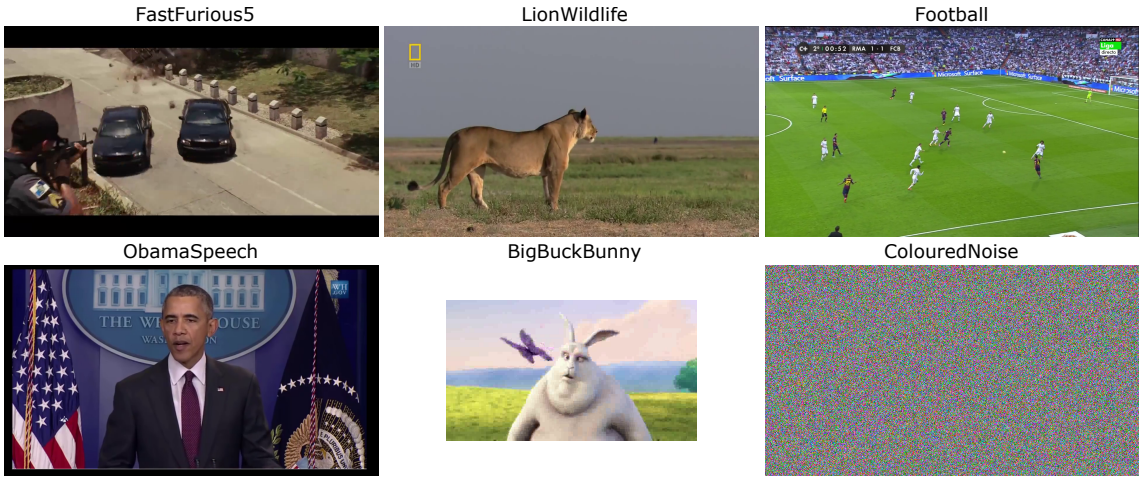


Figure 7.2: Video sequence snapshots

7.3.1 Encoder and decoder settings

The video streams were encoded using the open source x265 encoder (v1.7) [203]. x265 has shown to produce a good balance between compression efficiency and quality , by incorporating several advanced coding features such as adaptive, hierarchical B-frame sequences [204]. The default settings of x265 were complimented with additional settings, to suit resource-constrained decoding platforms (i.e embedded platforms with low memory), targeted at broadcast/video streaming applications. The encoder settings are specified in Appendix C.1. As discussed in Section 2.1.3, fetching reference frame data from main memory degrades decoding performance especially when multiple reference frames are used. Therefore, inter-prediction has been restricted to 1 forward and 2 backward references. Higher number of B-frames in a GoP offers better compression but decreases motion related quality. To strike a balance, the encoder was configured to use a maximum of 4 contiguous B-frames.

Key-frames (random access-points in the video) provide the ability to move (e.g. fast-forward, pause and rewind) within a video stream. x265 by default treats all I-frames as key-frames if the closed-GoP (self-contained/independent GoPs) setting is chosen. Closed-GoPs offer less compression than open-GoPs, but reduce error propagation during data losses, hence more suited for broadcast video. To ensure a balance between compression and random-access precision, maximum I-frame interval of 35 frames was chosen. Weighted prediction was disabled to increase decoder performance and B-frames were allowed to have intra-coded blocks in order to be efficient during high motion/scene-change sequences.

Decoding execution trace data was obtained using the open source, OpenHEVC [205] decoder with the minimal settings given in Appendix C.1. The platform used for decoding was a laptop with Intel Core i7-4510U 3.1GHz CPU, 4MB L3-cache and 8GB of DDR3 RAM, running Ubuntu 14.04 Linux OS. In order to eliminate any inter-thread communication and synchronisation latencies which might affect decoding time measurements, multi-threading was disabled. Also, dynamic power management mode was switched off, such that the processor operated at a fixed frequency. System cache was flushed before decoding each video.

7.4 Adaptive GoP characteristics

The structure of a GoP is directly related to the scene changes and motion in the video. As seen from Figure 7.3, the ratio of the number of P and B frames per GoP (denoted as P/B ratio) of the *Football* changes with respect to high-motion scenes or during scene-changes. The scene change rate is measured as $1/GoP_{diff}$, where GoP_{diff} refers to the mean number of GoPs between scene change events. The scene change rate in each tested video stream is given in Table 7.1. This metric gives a notion of how often the structure of the GoP changes with respect to time. The *FastFurious5* video has the highest scene change rate while the *Football* video has the lowest. P-frames give higher compression than I-frames, hence during scene changes, the x265 encoder increases the P/B ratio, whilst keeping the GoP length constant. Figure 7.4 shows two example GoPs from the *BigBuckBunny* video; the bottom GoP refers to a scene with high-motion (more P-frames).

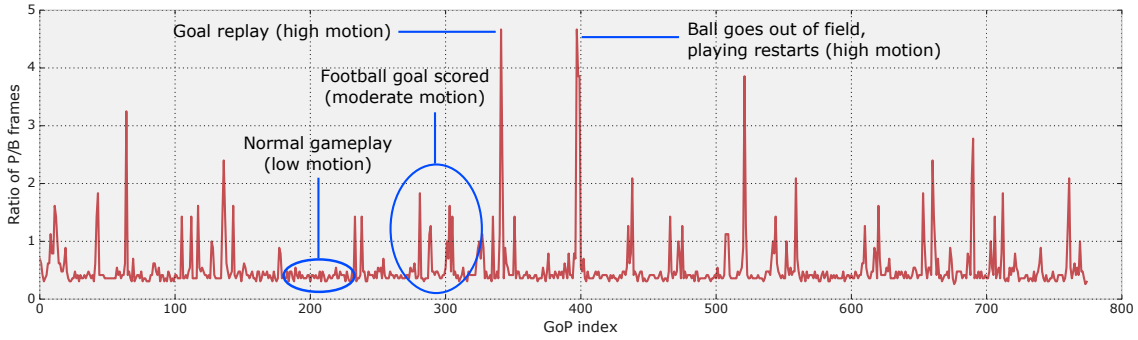


Figure 7.3: Ratio of P:B frames within a GoP varying with respect to motion and scene changes in the video (*Football* video)

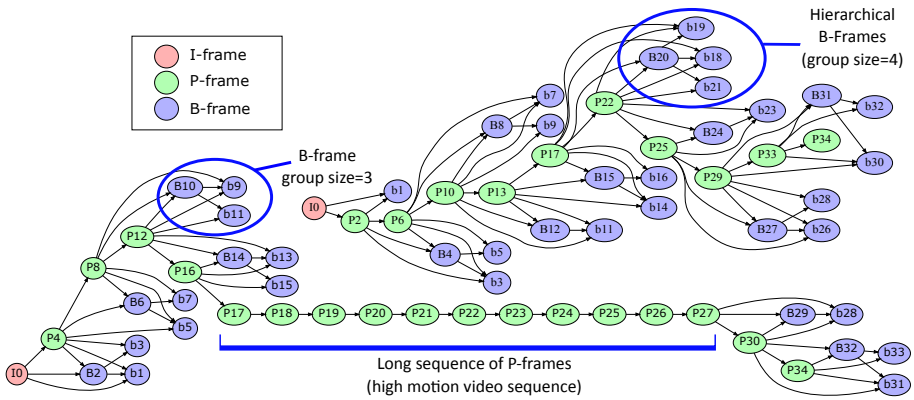


Figure 7.4: Example structure of a GoP (from *BigBuckBunny*). Top: low number of P-frames, Bottom: high number of P-frames. (NB: edges and nodes are unweighed)

Table 7.1: Scene change rate for all video sequences analysed. (GoP_{diff} = mean number of GoPs between scene changes)

Video	Scene change rate ($1/GoP_{diff}$)
FastFurious5	0.45
BigBuckBunny	0.30
LionWildlife	0.20
Football	0.05
ObamaSpeech	0.00
ColouredNoise	n/a

7.4.1 Distribution of different frame types

The number of P and B frames in a GoP for each video sequence is shown as a histogram in Figure 7.5. The number of P-frames in a GoP (denoted nP) is modelled as an exponentiated-Weibull (exp-Weibull) distribution [206], where the probability density function (PDF) of the exp-Weibull distribution is given as Eq. 7.1 with shape parameters a and c. Additional *scale* and *location* parameters defines the relative size and position of the PDF; they are specific to the statistics package used (i.e. SciPy).

$$f(x) = ac\,(1-\exp -x^c)^{a-1}\exp -x^c x^{c-1} \tag{7.1}$$

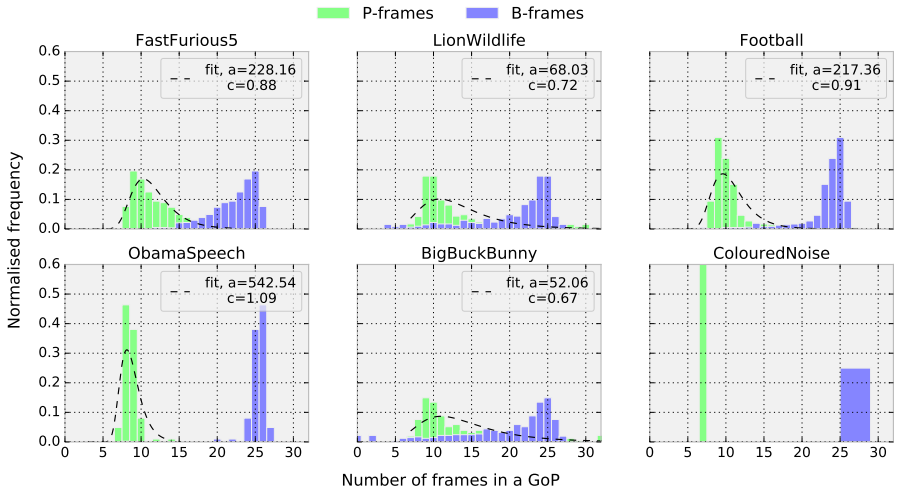


Figure 7.5: Distribution of P and B frames within a GoP, for all video sequences (distributions fitted to the exp-Weibull PDF with shape parameters (a,c), location=0, scale=1.5)

Preliminary distribution fitting using the Gamma and Gumbel PDFs provided inaccurate fits, motivating the use of exp-Weibull distribution due to its long-tail, right-skewed density and flexibility in shape. From Figure 7.5, it is clear that most of the video sequences fit the exp-Weibull distribution except for the case of *ColouredNoise*, where there is no variation in the number of P or B frames. As the GoP length N is fixed, the number of B-frames ($nB = (N-1) - nP$), is an inverted version of the number of P-frames in a GoP. For low-motion videos (e.g. *ObamaSpeech*), the number of P and B-frames show less variation and high values of exp-Weibull parameters (c,a) are seen. The inverse observation can be made regarding high-motion videos (e.g. *BigBuckBunny*).

7.4.2 Contiguous B-frames and reference frame distances

Figure 7.6a shows the proportion of different contiguous B-frames in a GoP. Low-motion videos (e.g. *ObamaSpeech*), show a high level of contiguous B-frames, while high motion videos (e.g. *FastFurious5*), the proportions are uniform. The proportion of contiguous B-frames have a direct impact to the reference distance of a frame. Reference distance (RFD) is calculated as the absolute difference between the GoP index of the current frame and its parent/dependent frame(s). Larger RFDs correlate with higher contiguous B-frames in a GoP. Higher average RFD in a video stream, means the decoder has to keep a decoded frame longer in the DPB, which would result in high buffer occupancy and hence a larger main memory requirement.

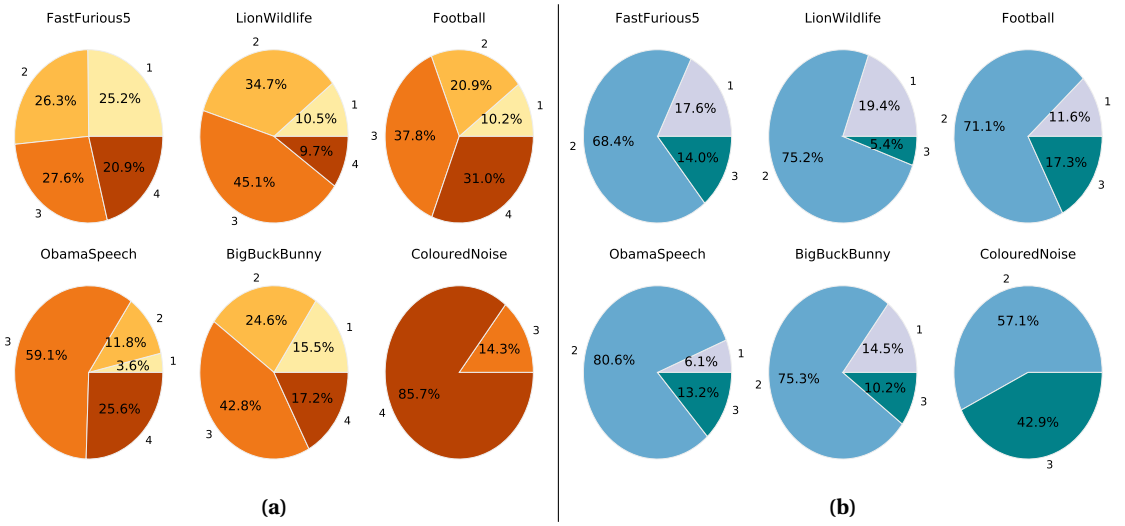


Figure 7.6: (a) Number of contiguous B-frames in a GoP, (b) Reference distances for B-frames in different video sequences.

Generally, P-frames do not refer to B-frames and if the encoder is restricted to have only 1 reference frame for the forward direction, then a P frame refers to the closest previous I/P frame (see Figure 7.4). B-frames referred to past and future I/P/B frames with a RFD characteristic as shown in Figure 7.6b. The most common B-frame RFD is 3, as the maximum number of contiguous B-frames (nB_{max}) in the GoP is constrained to 4. Further analysis showed that a contiguous sequence of B-frames referred only to I/P-frames in close temporal proximity; therefore long edges in the task graph are not present. A correlation exists between the RFD ratios shown in Figure 7.6b and the number of contiguous B-frames shown in Figure 7.6a. For example, less than 10% of contiguous B-frames in *LionWildlife* are of size 4, which leads to a low number of frames having a RFD of 3.

7.5 HEVC coding unit characteristics

As described in Section 2.1.2, HEVC frames are logically structured as coding tree blocks (CTB) and each CTB is recursively sub-partitioned into coding units (CU) in a quad-tree manner. In this section the CU-level characteristics: types, sizes and decoding times are captured and analysed for each video stream. These fine-grain CU properties are used to form the coarse-grain frame decoding time model.

7.5.1 CU sizes and types

HEVC CUs can be of the size 64×64, 32×32, 16×16, 8×8, 4×4 (intra only) and can use either intra (I-CU) or inter (P/B/Skip-CU) prediction. The size and type of CUs are defined further in Section 2.1.2. Figure 7.7a shows that intra and inter predicted frame types differ significantly in their use of different CU sizes. There are a higher number of smaller CUs in videos that have fine-grained information in the picture (e.g. individual players and background spectators in *Football* or detailed background in *ObamaSpeech*). Overall, 64×64 CUs are least used than other sizes, however inter-frames show significantly more use of 64×64 CUs than intra-frames. The encoder has failed to use small CU sizes for the *ColouredNoise* video.

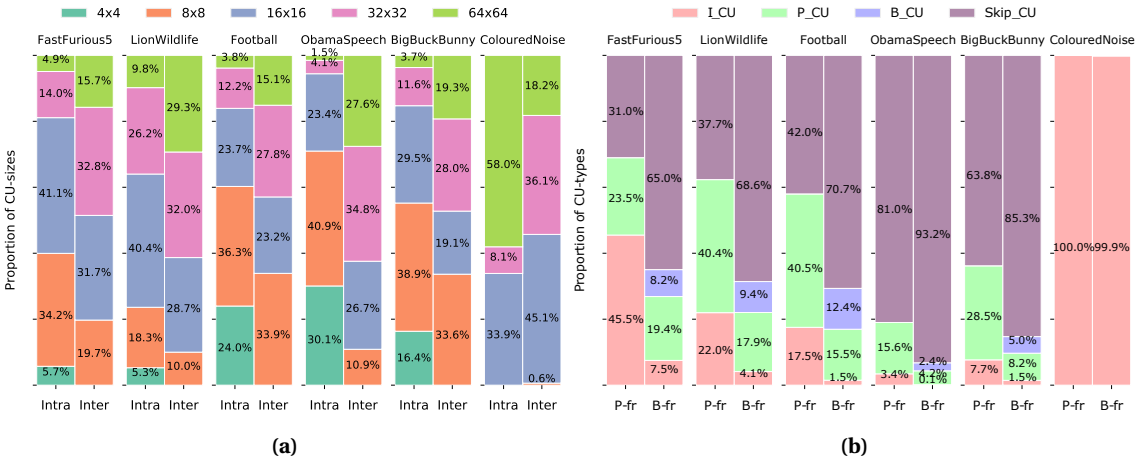


Figure 7.7: (a) Proportion of CU sizes within each video sequence (per Intra/Inter frame type) (b) Proportion of CU types within each video sequence (per Inter frame type)

As seen in Figure 7.7b, except for *ColouredNoise*, the encoder has exploited heavy use of Skip-CUs to offer better compression. The number of I-CUs are higher in video sequences with high-motion. The inverse appears to be true of Skip-CUs; for example in *ObamaSpeech*, the amount of Skip-CUs are between 80-93%. In general, the number of P-CUs are 2-3 times the amount of B-CUs, and P-frames show a higher amount of I-CUs than B-frames. The encoder has failed to use inter-prediction to compress the *ColouredNoise*, where 99% of the video has been coded using intra-CUs.

7.5.2 CU decoding time

The distributions of CU-level decoding time are given in Figure 7.8. I,P and B-CU decoding times are fitted to an exp-Weibull distribution (parameters given in Appendix C.2). Skip-CUs can belong to either P or B frames; hence, the Skip-CU decoding times are multi-modal. The Skip-CU decoding time is modelled as a high-order polynomial function (coefficients are given in Appendix C.2). High B-CU decoding times are due to complex transformations, in bidirectional inter-prediction. B-CUs decoding times are about 2-3 times larger than I/P-CUs. The CU decoding time is dependent on the CU-size and content of the video sequence. This is evident in the *Football's* B-CU decoding time, where compared to others has a lower variance in decoding times, because of a higher proportion of inter-frame 8×8 CUs in the video stream. A trend can be seen in the I-CU decoding time and the exp-Weibull shape parameters; where high-motion, high scene-change videos such as *FastFurious5* and *LionWildlife* have a lower a and higher c, leading to wider distributions. Overall Skip-CUs have the lowest decoding time compared with

the other CU-types. The I-CU decoding time in *ColouredNoise* gave the highest execution time out of all the videos; giving an observed worst-case CU decoding time of 4.94×10^{-4} s. The Skip-CU decoding in *Football* shows the lowest decoding time (1.08×10^{-6} s), which shows us that decoding CUs can have up to two orders of magnitude variation, depending on the type of video and CU-types/sizes used at the encoder.

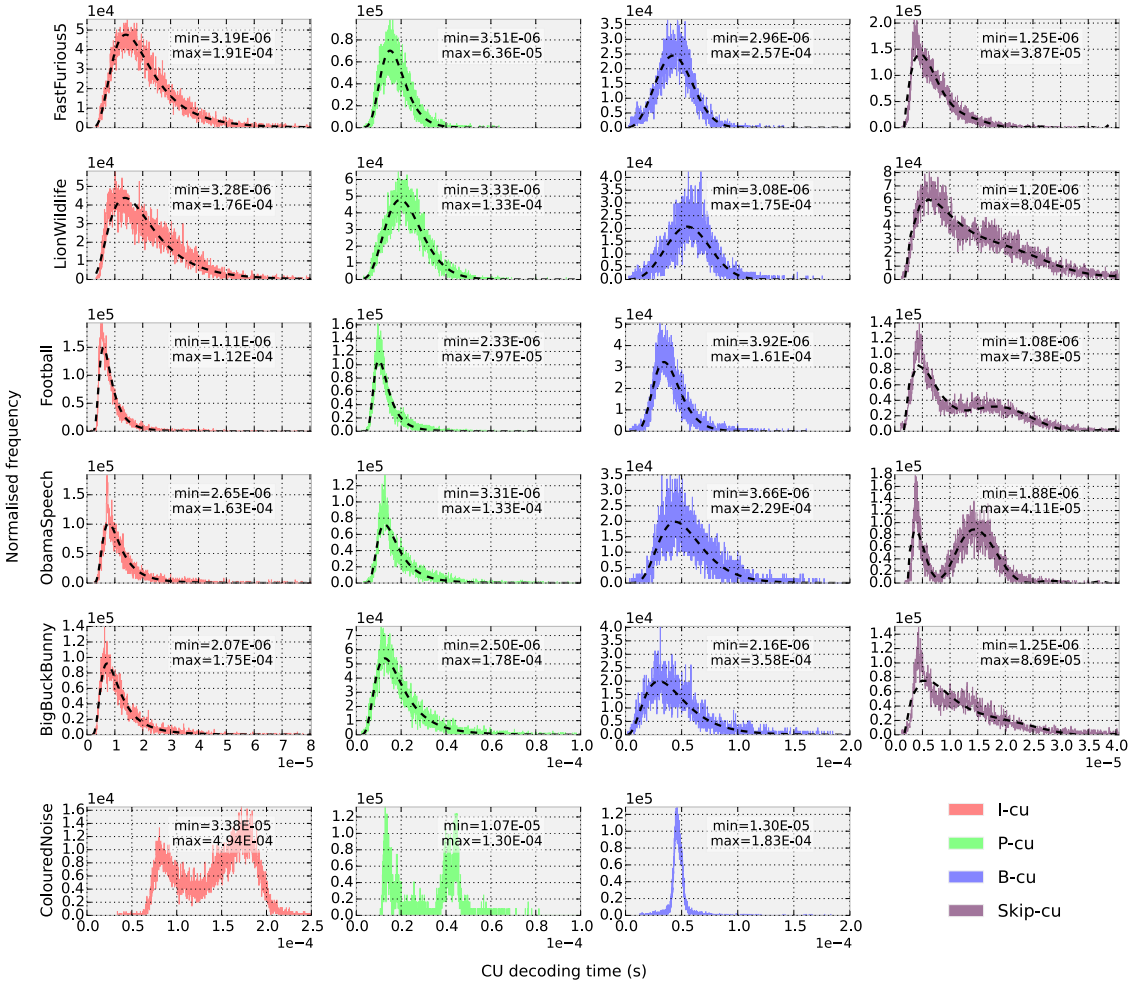


Figure 7.8: Normalised CU-level decoding time histogram per video sequence. I/P/B-CUs distributions fitted to an exp-Weibull PDF (dashed line; parameters given in Table C.1); polynomial function fitted to Skip-CUs (parameters given in Table C.2). (NB: Sub-figures use different scales, share axes and distribution tails are cropped in order to assist visualisation)

Furthermore, encoding *LionWildlife* resulted in a higher number of 16x16 to 64x64 sizes; hence, increased CU decoding times are seen. Larger CU-sizes could lead to higher decoding times due to bottlenecks at the memory subsystems [207].

Frame decoding time distributions for the video streams are given in Figure 7.9. The number of different CU sizes/types and their decoding costs contribute to the shape of the frame-level decoding time distribution. The primary factor for the frame decoding time as explained by Kim et al. [207], is the distribution of different CU sizes in a frame. Small CU sizes in a frame will result in a higher frame decoding time. This is evident in *Football*, where frame decoding times are relatively higher and it also has a high number of 8×8 and 16×16 CUs. LionWildlife also has a high frame decoding cost, which correlates with a high number of P and B-CUs as well as a higher B-CU decoding time. Overall in every video stream, $t_I > t_P > t_B$ where the terms t_I, t_P and t_B denote I,P,B frame decoding times. This is mainly due to the number of Skip-CUs

in a frame. For example, a B-frame primarily contains Skip-CUs (Figure 7.7b), and Skip-CUs decoding times are lower than other CU types, resulting in a relatively lower overall B-frame decoding time. It takes about 2-3 times longer to decode *ColouredNoise* when compared to the other streams, because about 99% of the stream consists of I-CUs and no Skip-CUs.

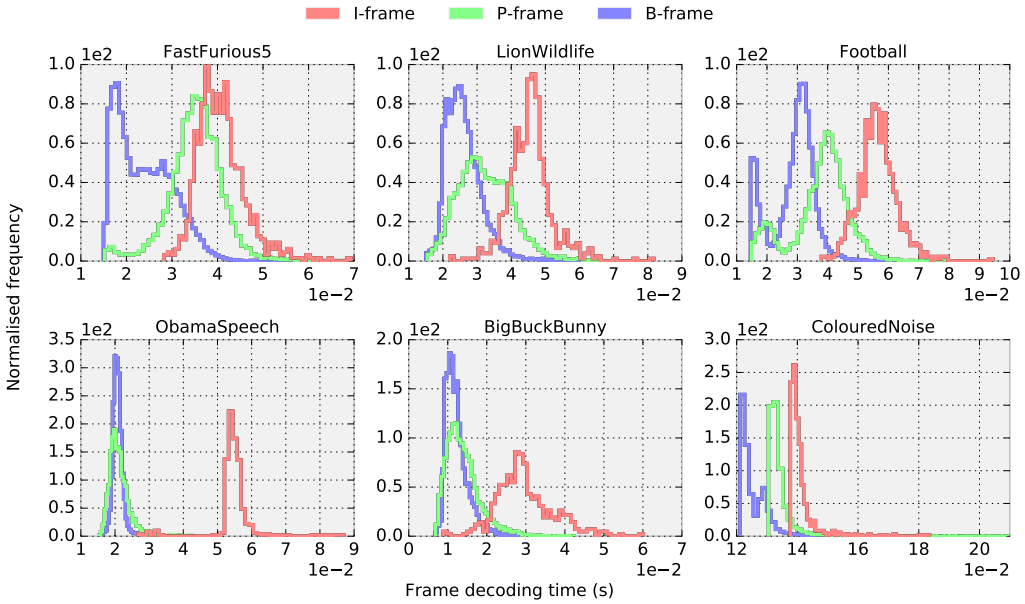


Figure 7.9: Normalised frame-level decoding time distribution histogram for each video sequence

7.6 Workload communication volume characteristics

In Section 7.4 the inter-frame dependency pattern in a GoP was analysed. This section investigates the *volume* of reference data required for inter-prediction, which is essentially the *weight* of each arc in the GoP-level TG application model. Apart from reference frame-data, the encoded frame also needs to be loaded from main memory in order to perform the decoding operations; this forms the memory read traffic in the application. The encoded frame sizes are also analysed to improve the memory read traffic payload model.

7.6.1 Reference data

The reference data is the pixel data of a decoded frame; the maximum amount of data a P-frame can reference is ($fr_{size} = (h \times w) \times bpp$), where h and w represents the frame dimensions and bpp is bytes per pixel. For B-frames, this upper bound is doubled due to bi-directional prediction, hence B-frames typically reference twice the volume of data than P-frames. Figure 7.10 shows the distribution of reference data categorised by direction of reference; for example $P \leftarrow I$ refers to the data referenced by a P-frame from an I-frame in the GoP. Considering the distributions and their sample sizes for each video stream, overall all inter-frames have a preferred reference probability of: $p_I > p_P > p_B$, where for example p_I denotes the probability of a frame obtaining data from an I-frame. This observation is true because B-frames have the lowest decoded frame data accuracy due to bi-directional prediction, and I-frames have the highest accuracy due to only using intra-prediction.

The reference data distributions are highly dependent on the RFD (Figure 7.6b), the number of contiguous B-frames (Figure 7.6a) and the distribution of frame types (Figure 7.5) in a GoP. This is also the reason for the wide variation of reference data in the $B \leftarrow P$ distribution. The data traffic variation seems to be higher in videos with high rate of scene-change (e.g. *FastFurious5*), while the mean amount of data volume is higher in largely static videos (e.g. *ObamaSpeech*). *ColouredNoise* has very low reference data because a majority of the CUs are I-CUs. *BigBuckBunny* has a lower amount of reference data due to its lower resolution.

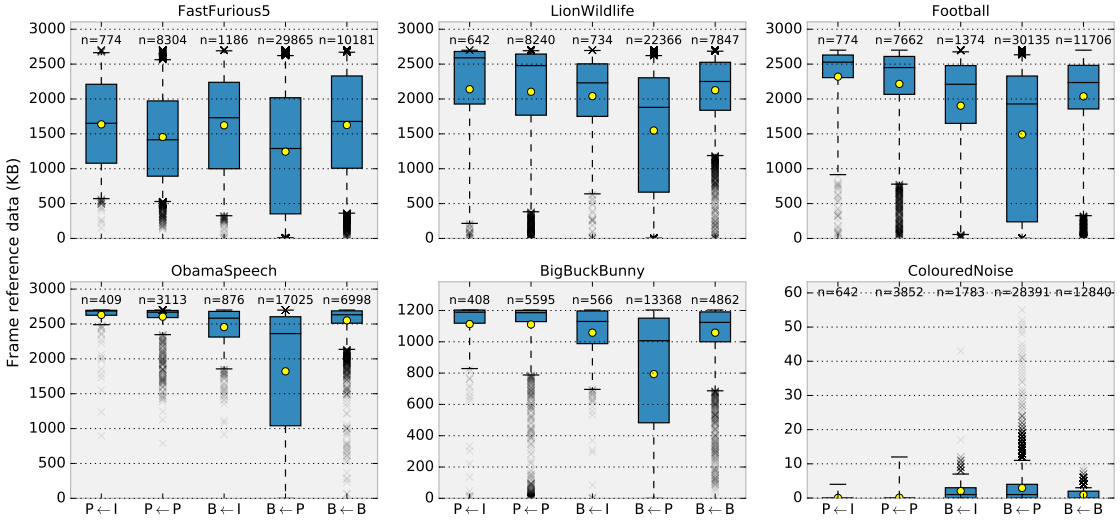


Figure 7.10: Distribution of frame reference data for all video sequences. E.g. $P \leftarrow I$ refers to the data referenced by a P-frame from an I-frame. Sample-size of the distribution given as n. Circular markers indicate mean value

7.6.2 Encoded frame sizes

As described in Section 5.1.1.1, the encoded frame size represents the memory read transaction payload in the frame-level application model. The model assumes the complete encoded frame data needs to be read from memory into the PE local memory before executing a frame decoding task. Figure 7.11 shows the distribution of encoded frame sizes as ratio of the uncompressed frame size (fr_{size}). The distributions are fitted with a exp-Weibull distribution (parameters given in Table C.3). In the previous application model, a fixed compression ratio for a memory read, was assumed as per Eq. (5.1). In the new HEVC application model, the compression ratio is sampled from an exp-Weibull distribution with the parameters obtained from the real distributions shown in Figure 7.11.

Overall, $s_I > s_P > s_B$, where s_I, s_P, s_B denote I/P/B encoded frame sizes. However, the long-tailed distributions tell us that there may be extreme-case scenarios where this may not always be true (further verified in [60]). It can be observed that the variation in the compression ratio distribution, is relative to the motion/scene-change rate of the video streams. Low-motion videos such as *ObamaSpeech* have a much lower encoded P/B frame size than high-motion videos. B-frame sizes have very long-tailed distributions compared to I or P-frames due to the variation in the number of Skip-CUs in the frame; because Skip-CUs do not contain a residual, the amount of data encoded is relatively small. In general I-frame sizes have less variation than inter-predicted frames.

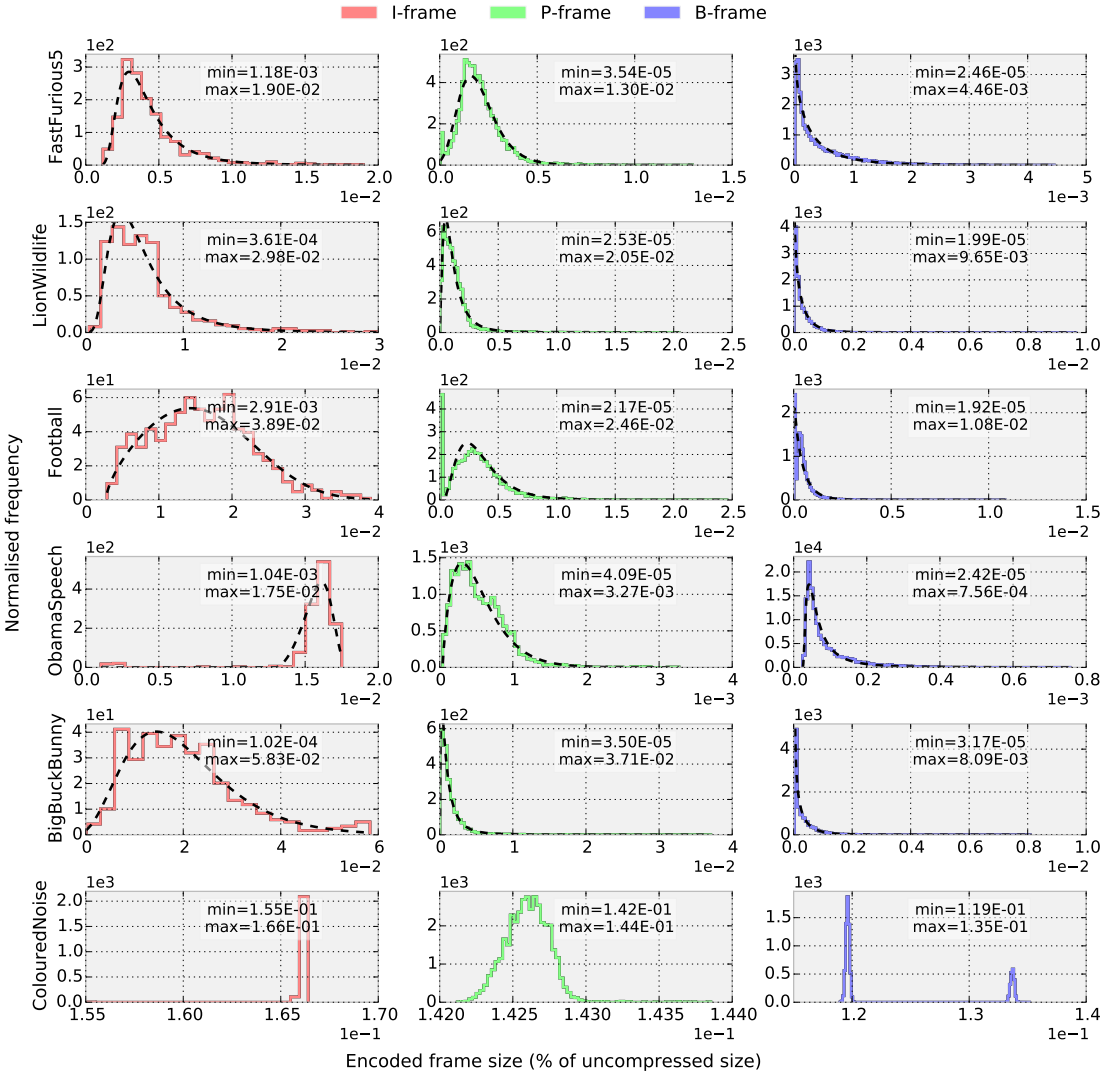


Figure 7.11: Distribution of frame compression ratio (encoded frame size as a % of the uncompressed frame size). Distribution fitted to Exp-Weibull PDF; parameters given in Table C.3. (NB: Sub-figures use different scales in order to assist visualisation)

7.7 CPU and memory usage breakdown

The CU and frame decoding time measurements presented in Section 7.5.2, actually include the time spent on different subsystems of the experimental platform. However, the application model used in this work assumes the frame decoding task *execution time* (x_i), is the pure *computation time* of the task excluding other temporal artefacts such as fetching data and code from external memory, processor stalls, cache misses etc. Note that, the memory transaction costs and inter-task communication costs are modelled separately in the application model. Modern processor architectures contain numerous complex design features such as instruction level parallelism, out-of-order execution, complex branch prediction and hardware prefetching units in order to keep their execution pipelines busy. Therefore, calculating the actual time spent only on computation is a challenging task. Furthermore, the task computation cost can vary significantly, depending on the platform architecture.

This section of the chapter investigates the breakdown of the CPU/memory usage of a video

decoding process using the *Intel VTune performance analyser/profiler* [208]. The profiler categorises the application execution time according to the following classification:

- *Front-end bound* : this is portion of time spent by the processor to fetch and decode instructions from cache/memory and translate them into micro-operations (μops).
- *Back-end bound* : the μops delivered from the front-end are fed into the back-end portion of the processor to schedule, execute and commit (retire) the instructions. This section is further split into the following two sub-categories:
  - *Memory-bound* : indicates time spent on fetching data from the memory subsystem (including L1/L2/L3 cache misses at all levels and fetching from off-chip DRAM memory).
  - *Core-bound* : indicates the time the execution pipeline was stalled. Long latency instructions (e.g. divide operations) and non-vectorised code can contribute to this section.
- *Retiring* : this category represents instructions that have finished execution and in a performance context indicates useful work done by the processor. It is reasonable to consider this is the portion of time purely spent on computation.
- *Bad speculation* : this is the time spent on bad branch predictions resulting in flushing the pipeline.

Figure 7.12a shows the breakdown of the decoding time spent in the above different platform components. Different resolutions of the *LionWildlife* video (2 GoPs) were used to obtain the measurements. Results show that on average, ≈45% of the time is spent on actual computation (retiring) and ≈30% of the time is spent on pipeline stalls and ≈15% on the memory subsystem. Result also indicate that the front-end bound issues are increased for low resolutions videos. Figure 7.12b shows a breakdown of the memory subsystem transactions. These measurements would depend on architectural characteristics such as cache and cache-line size and DRAM bandwidth. Results indicate that a decoder is mainly L1-cache bound which is not a main concern as L1 has the shortest latency. DRAM memory transactions can contribute to up to 44% of the memory subsystem latency.

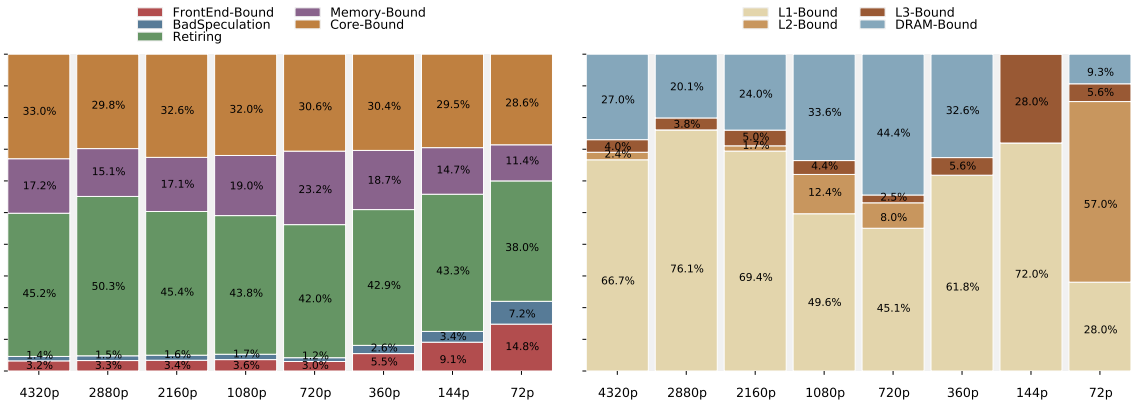


Figure 7.12: Decoding time breakdown according to the platform subsystems (for varying resolutions of the *LionWildlife* video): (a) High-level breakdown: front-end, back-end, retiring and bad speculation, (b) Breakdown of the memory-subsystem related decoding time

7.8 Synthetic HEVC stream decoding workload generation

This section presents algorithms to synthetically generate HEVC video decoding workloads using the GoP/CU/frame level characteristics of HEVC video streams, obtained via analysis in the previous sections (i.e. Sections 7.4, 7.5 and 7.6). The workload generation follows the hierarchical structure of the video streams as shown in Figure 7.1. Firstly, for each GoP in the video stream workload, the GoP structures need to be created, which defines the reference data and dependency patterns. Secondly, for each HEVC frame in a GoP, the CUs need to be generated in a bottom-up fashion. Likewise, any number of HEVC video streams can be synthetically generated and fed into an abstract system-level simulator.

The ratio-based relationships and analytical distribution parameters obtained for each video stream type in the stream analysis stage (i.e. Sections 7.4, 7.5 and 7.6) can be used as a basis for generating similar type of video sequences. For example, to generate a video with high motion and heavy scene-changes, the number of P-frames can be generated using the exp-Weibull PDF parameters similar to *FastFurious5*. If a slow motion video with high level of fine grain imagery is required, the contiguous B-frames ratios can be similar to *ObamaSpeech* and the CU sizes can be made similar to *Football*. Similarly, different types of videos can be generated by mixing/adapting the model parameters as needed.

7.8.1 Synthetic dynamic GoP structure generation

Construction of a synthetic HEVC dynamic/adaptive GoP structure can be done in two stages as per Algorithm 7.1. This algorithm is invoked to build each GoP in the video stream workload. In the first phase of the algorithm, the GoP sequence (in temporal decoding order) is generated, taking into account the exp-Weibull distributed P,B frames. The position of the B-frames within the GoP are uniformly distributed, but the selection of contiguous B-frame sizes are derived from the ratio relationships observed from the trace results (Figure 7.6a). Lines 9-16 generates B-frame groups (i.e. hierarchical B-frames) and inserts them at random positions in the GoP. In *Phase II* of the algorithm, each inter-predicted frame is assigned reference frames as per the analysis in Section 7.4. P-frames refer to the immediate previous I/P frame in the GoP, this gives a long task chain like property to the TG as shown in Figure 7.4. B-frames are assigned multiple reference frames (forward/backward) as per the RFD ratios seen in Figure 7.6b. Lines 19-29 derive possible and legal reference frames within the GoP. The RANDOM.CHOICE() function, generates a random sample from the given possible reference frame set, according to the specified *probabilities* derived from the analysis.

The parameter N in Algorithm 7.1 can be varied to obtain different GoP lengths. Typically, N is constant for all GoPs in the video, and only the frame sequence and dependency structure will vary. $nBmax$ and W_p can be varied to obtain different numbers of B and P frames within a GoP. The parameters rB^{fwd}_{max}, rB^{bwd}_{max} and rP^{fwd}_{max} can be used to provide more reference frames to P and B frames, this would in turn increase the number of edges in the TG. vs^{prob}_{rfd} is representative of the length of the edges in the TG and also gives a notion of the number of possible reference frames for a B-frame.

7.8.2 Synthetic HEVC frame generation

Once the GoP sequence and structure has been generated (Section 7.8.1) the frames within each GoP can then be created as per Algorithm 7.2. The algorithm builds up the frame-level task in

Algorithm 7.1: Pseudo-code for GoP structure generation

Input : N : GoP length ,
nB_{max} : max. sequential (contiguous) B-frames,
W_p : exp-Weibull PDF shape params. for number of P-frames in GoP (a,c,scale,loc)
B_{seq}^{prob} : contiguous B-frame probabilities, vs_{rfd}^{prob} : reference distance ratios
$rB_{max}^{fwd}, rB_{max}^{bwd}$: max. forward and backward B-frame references,
rP_{max}^{fwd} : max. forward P-frame references, r_d^{max}: max. reference distance
Output: GoP frame sequence (decoding order) and dependency structure for job J_i

1 **Phase I - Construct GoP sequence**

2 Validate parameters: **assert**$((N-1)\%(nB_{max}+1) == 0)$
3 Min/Max num. P-frames in the GoP: $(nP_{min}, nP_{max}) = \{(N-1)/(nB_{max}+1), (N-1)\}$
/* Derive exp-Weibull distributed number of P,B-frames */
4 $nP = \text{EXPWEIBULL}(W_p).\text{SAMPLE}(nP_{min}, nP_{max})$
5 $nB = (N-1) - nP$
6 Set of possible contiguous B-frame lengths: $nB_{sizes} = [1..nB_{max}]$
7 $B_{fr} = \{\}$ /* hash table of positions and B-frame numbers */
8 $GOP_{fr} = \text{“}I\text{”} + (\text{“}P\text{”} * nP)$
/* get contigous B-frame positions in the GoP, iteratively */
9 **while** $\sum B_{fr}.values() < nB$ **do**
10 $pos = \text{RANDOM.CHOICE}([1..nP], \text{prob='UNIFORM'})$
11 $tmpB_{fr} = \text{RANDOM.CHOICE}(nB_{size}, \text{prob=}B_{seq}^{prob})$
12 **if** $(B_{fr}[pos].value + tmpB_{fr}) <= nB_{max}$ **then**
13 $B_{fr}[pos] += tmpB_{fr}$
14 **end**
15 **end**
16 $GOP_{fr} \leftarrow B_{fr}$ /* Put B-frames into GoP */

17 **Phase II - Construct GoP frame references**

/* hash table of inter-frame references */
18 $fr_{refs} = \{\}$
19 **for** $fr_{ix}, fr \in GOP_{fr}$ **do**
/* P-frames depend on previous I/P frames */
20 **if** $fr == \text{“}P\text{”}$ **then**
21 $r_{POCs}^{all\_fwd} = GOP_{fr}[POC < fr_{ix} \cap (RFD \leq r_d^{max}) \cap \neg\text{“}B\text{”}]$
22 $refs_{fwd} = \text{RANDOM.CHOICE}(r_{POCs}^{all\_fwd}, \text{size} = rP_{max}^{fwd}, \text{prob=}vs_{rfd}^{prob})$
23 $fr_{refs}[fr_{ix}] = refs_{fwd}$
24 **else if** $fr == \text{“}B\text{”}$ **then**
/* B-frames depend on previous/future I/P/B frames */
25 $r_{POCs}^{all\_fwd} = GOP_{fr}[POC < fr_{ix} \cap (RFD \leq r_d^{max})]$; $r_{POCs}^{all\_bwd} = GOP_{fr}[POC > fr_{ix} \cap (RFD \leq r_d^{max})]$
26 $refs_{fwd} = \text{RANDOM.CHOICE}(r_{POCs}^{all\_fwd}, \text{size} = rB_{max}^{fwd}, \text{prob=}vs_{rfd}^{prob})$
27 $refs_{bwd} = \text{RANDOM.CHOICE}(r_{POCs}^{all\_bwd}, \text{size} = rB_{max}^{bwd}, \text{prob=}vs_{rfd}^{prob})$
28 $fr_{refs}[fr_{ix}] = \{refs_{fwd}, refs_{bwd}\}$
29 **end**

a hierarchical bottom-up manner, by first iterating through each CTU in the frame (line 3) and then constructing a set of CUs per CTU (lines 5-17). The number of CUs in a CTU is obtained based on the selection of specific CU sizes (line 6); the cumulative CU sizes must be equal to the CTU size (e.g. 64×64). For each CU a random CU type is selected using the probabilities given as parameters (line 9). The CU decoding time is sampled from an exp-Weibull distribution (lines 10-13). The frame decoding time is calculated as the summation of the CU decoding time (C_{dt}) of all CUs in the frame (line 20). Similarly, CU reference is selected from the given reference frame list fr_{ref} (line 14). The input parameters to the algorithm facilitates the selection of the CU size/type/decoding time and references; these parameters were derived in the previous stream analysis process. Videos of different resolutions can be generated by changing

the h, w parameters.

Algorithm 7.2: Pseudo-code for generation of HEVC frames using CU-level properties

Input : h, w : video resolution height and width CTU^{px}_{max} : CTU size (generally 64×64),
$CU_{sizes} = \{64, 32, 16, 8, 4\}$
$CU_{types} = \{I\text{-}CU, P\text{-}CU, B\text{-}CU, Skip\text{-}CU\}$,
CU^{p}_{size} : CU size probabilities per frame type - Figure 7.7a,
CU^{p}_{types} : CU type probabilities - Figure 7.7b
$W^{I}_{p}, W^{P}_{p}, W^{B}_{p}$: CU-decoding time exp-Weibull parameters (a,c,scale,loc)
p^{skip}_{c} : Skip-CU decoding time polynomial coefficients
$dI^{lim}_{CU}, dP^{lim}_{CU}, dB^{lim}_{CU}, dSkip^{lim}_{CU}$: min/max decoding time range for CU types
fr_{ref} : list of reference frames for current frame
PB_{rf} : probability list of current frame type referencing an I/P/B frame
CU_{dsf} : CU-decoding scale factors for different CU types

Output: An HEVC frame decoding task with CU-level information (i.e. decoding time, type, size and reference data for each CU in frame)

Number of CTUs per frame: $N_{CTU} = (h \times w)/CTU^{px}_{max}$
/* Construct each CTU in the frame */
$fr_{CTUs} = \{\}$ /* empty CTU list in frame */
for *each_CTU* $\in [0..N_{CTU}]$ **do**
  $px = 0; CTU_{CUs} = \{\}$ /* init. data struct. */
  /* get CU-level information for each CU in the frame */
  while $px < CTU^{px}_{max}$ **do**
    /* randomly select CU size */
    $C_s = \text{RANDOM.CHOICE}(CU_{sizes}, \text{prob}=C^{p}_{s})$
    if $(px + C_s) \leq CTU^{px}_{max}$ **then**
      $px += (C_s)^2$
      /* randomly select CU type */
      $C_t = \text{RANDOM.CHOICE}(CU_{types}, \text{prob}=C^{p}_{t})$
      /* get decoding time per CU type, randomly sampled from an analytical PDF */
      if $C_t == I\text{-}CU$ **then** $C_{dt} = \text{EXPWEIBULL}(W^{I}_{p}).\text{SAMPLE}(dI^{lim}_{CU}) \times CU_{dsf}[C_t]$
      else if $C_t == P\text{-}CU$ **then** $C_{dt} = \text{EXPWEIBULL}(W^{P}_{p}).\text{SAMPLE}(dP^{lim}_{CU}) \times CU_{dsf}[C_t]$
      else if $C_t == B\text{-}CU$ **then** $C_{dt} = \text{EXPWEIBULL}(W^{B}_{p}).\text{SAMPLE}(dB^{lim}_{CU}) \times CU_{dsf}[C_t]$
      else if $C_t == Skip\text{-}CU$ **then** $C_{dt} = \text{POLYNOMIAL}(p^{skip}_{c}).\text{SAMPLE}(dSkip^{lim}_{CU}) \times CU_{dsf}[C_t]$
      /* pick CU reference from reference frame list */
      $C_{rf} = \text{RANDOM.CHOICE}(fr_{ref}, \text{prob}=PB_{rf})$
      /* append CU info. to CU list */
      $CU_{info} = \{C_s, C_t, C_{dt}, C_{rf}\}; CTU_{CUs} \leftarrow CU_{info}$
    end
  end
  /* append to CTU list */
  $fr_{CTUs} \leftarrow CTU_{CUs}$
end
/* The frame decoding time and reference data can be obtained as follows: */
Frame decoding time: $x_i(\tau_i) = \sum_{\forall CTU_i \in fr_{CTUs}} \sum_{\forall CU_i \in CTU_i} C_{dt}$
Frame ref. data (bytes) from parents : $fr_{ref}[j] = \sum_{\forall CTU_i \in fr_{CTUs}} \sum_{\forall CU_i \in CTU_i} \begin{cases} (C_s)^2 \times 8bpp, & \text{if } C_{rf} = j \\ 0, & \text{otherwise} \end{cases}$

7.8.3 Limitations of the proposed workload generator

The size and type characteristics of the CUs obtained during the stream analysis (Section 7.5.1) are cumulative values of all frames in the video stream. Similarly, the CU decoding time distributions analysed in Section 7.5.2, represent the decoding time for each CU type in a complete video stream. Therefore, frame-level variations are not accurately captured in this model. However, consecutive frames in the same video stream would have similar CU properties due to high spatial/temporal correlation, but frames in different GoPs would have less correlation. In order

to introduce a degree of variation in the CU types/sizes between different frames each probability parameter (e.g. CU size probabilities) are varied by a percentage value according to a normal distribution. For example *FastFurious5* has 31% Skip-CUs in P-frames (Figure 7.7b) but when generating frames this probability is varied slightly, according to a normal distribution with $\mu = 0.31$ and $\sigma = 0.05$. The σ value needs to be high enough to add a certain level of variation between different frames of the same video stream, but not too high such that the original proportions will be masked. In a real video stream however, the variation between frames would be complex and would not fit a theoretical distribution. A deeper analysis into the distribution of the CU types and sizes of a distribution of frames will need to be analysed, in order to infer the variations between frames.

In Algorithm 7.1 (lines 9-16), B-frame groups are created with given probabilities of each group size, such that the final sum of B-frames $nB = (N-1) - nP$. This problem can be considered a variation of the coin-change/subset-sum problem which is considered to be NP-Complete [209], and therefore requires a dynamic programming approach. The technique given in the algorithm is a simple iterative technique which may not always retain the final B-frame grouping probabilities after creation. Therefore, the probabilities may require minor adjustments during the GoP creation process to maintain the required contiguous B-frame probabilities at the stream-level.

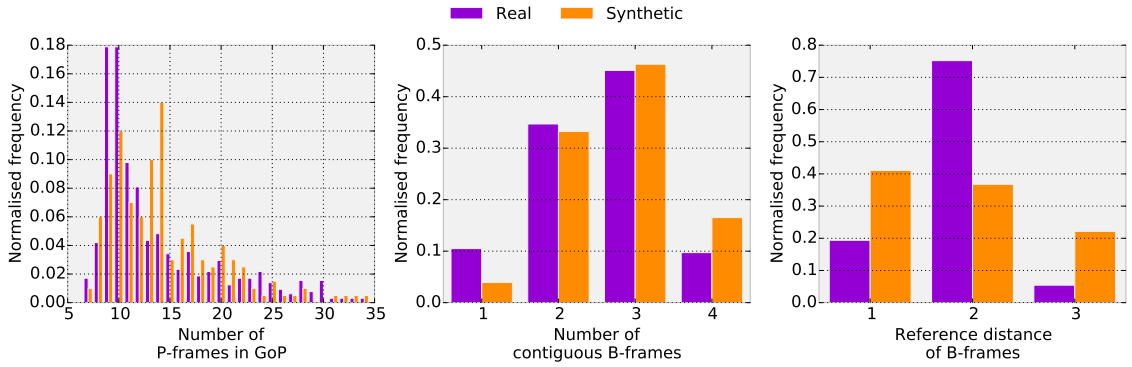
The profiled CU decoding time results do not contain the stream parsing (CTU/CU header parsing) delay which needs to be integrated into the CU computation cost model. During evaluation of the workload generator, it was noticed that the CU decoding costs needs to be scaled up to account for the parsing delay. The scale factors used during evaluation are presented in Table C.4. Furthermore, as discussed in Section 7.7, the memory subsystem overheads are included in the profiled CU decoding time measurements. Memory transaction related timing need to be omitted from the decoding time to obtain a pure computation cost model. Therefore, when simulating HEVC workloads the CU decoding time is multiplied by a factor of 0.6, to account for the $\approx 40\%$ DRAM transaction delay within the execution time. These scale factors need to be tuned appropriately to suit a platform and decoder with different memory and CPU architecture characteristics. For the evaluation of the workload generator the scale factors shown in Table C.4 gave satisfactory results.

7.9 Evaluation

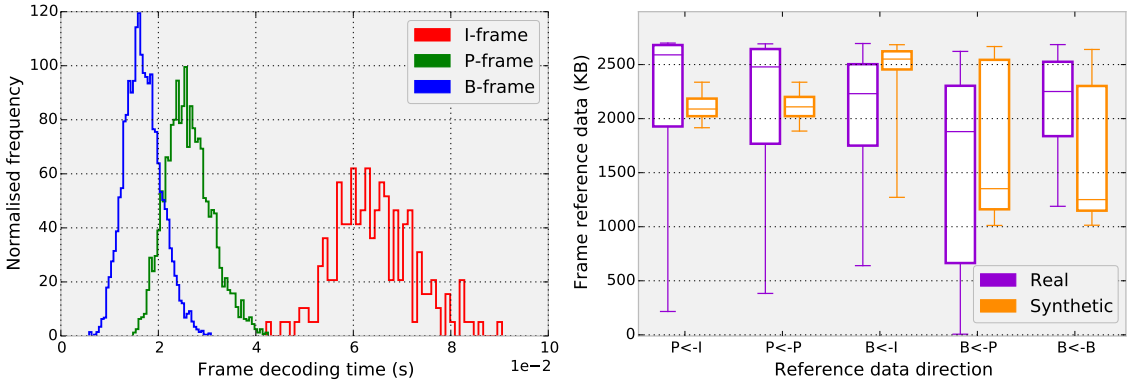
To evaluate the proposed HEVC video decoding workload generator, 200 synthetic HEVC coded dynamic GoPs (35 frames each) were generated with parameters suited to generate the *Lion-Wildlife* 720p video. The synthetic HEVC video stream characteristics were then compared with the real video stream. All parameters used for the workload generation were derived from the GoP/frame/CU level analysis and can be found in Appendix C.3.

7.9.1 Results discussion

Figure 7.13a shows the GoP structure characteristics of the real and synthetically generated HEVC video stream decoding workload. The number of P-frames have similar distribution shape (exp-Weibull) and the contiguous B-frame groups of sizes of 1,2 and 3 are similar to the real stream. However, the minor mismatch in the P-frame distribution introduces minor mismatch errors, in the contiguous B-frame group sizes and RFDs, as seen in the evaluation results.



(a) GoP structure comparison: *Left* - Number of P-frames in a GoP; *Center* - Proportions of contiguous B-frames; *Right* - Proportion of B-frame reference distances.



(b) Frame characteristics comparison: *Left* - Frame decoding time per I/P/B frame type; *Right* - Distribution of reference data

Figure 7.13: Comparison of GoP and frame-level characteristics of a real vs. synthetically generated HEVC video stream

Figure7.13b(left) shows the decoding time distributions of the synthetically generated HEVC frames; these need to be evaluated against the data given in Figure 7.9(*LionWildlife*). Due to inaccurate representation of the frame-level variations (Section 7.8.3), the distribution of the frame decoding times generated do not exactly follow the same shape as the real video stream. However, it can be seen that the decoding times are approximately in the same region for P and B-frames (i.e between $1.0-4.0\times10^{-2}s$), which is majority of the video stream; and a slightly larger I-frame decoding time ($4.0-8.5\times10^{-2}s$) can be seen. Similar to the real video, P-frame decoding times are overall larger than B-frames. The generated frame decoding times show a narrower spread of values than the real video stream.

The reference data volume distribution comparison is shown in Figure 7.13b(right). The inter-quartiles of the real and synthetically generated HEVC video distributions overlap, especially in the case of $B \leftarrow P$. However, the $P \leftarrow I$ and $P \leftarrow P$ show a narrower spread and the $B \leftarrow B$ show a slightly lower distribution range than the real video. These discrepancies in the distributions are due to the inaccurate view of the frame-by-frame CU variations as explained in Section 7.8.3.

7.10 Summary and novel contributions

The previous chapters consider MPEG-2 workloads with fixed GoP structures. This chapter extends the application model to consider HEVC decoding workloads with adaptive GoP structures. The following novel contributions are made to extend the state-of-the-art in video decoding workload characterisation:

- *Characterisation of adaptive GoP structures:* existing research in video decoding characterisation lacks analysis regarding several important GoP-level properties. GoP-dependency structures, patterns of B-frame groupings (i.e. hierarchical B-frames), reference frame distances and reference data volume analysis have been presented for real video streams. This work quantitatively shows that the inter-frame dependency pattern of the GoPs are highly correlated with the level of activity or motion in the video stream. Statistical distributions have been used to model the number of P/B-frames in a GoP.

- *HEVC frame decoding model derived using CU-level analysis:* this chapter contains in-depth analysis of video streams at the CU-level (CU sizes, types and decoding times). An HEVC frame decoding model is then derived in a bottom-up fashion using fine-grain CU-level information. This level of detail also facilitates modelling of HEVC Tile or WPP task models.

- *Algorithms to synthetically generate HEVC decoding workloads:* the video stream analysis provided several statistical properties of the adaptive GoP structures and the CU-level characteristics. These properties were then introduced to two workload generation algorithms to synthetically generate DAG-based workloads that have similar statistical characteristics of real HEVC video stream decoding. The presented generators can be used to create any number of video streams with different spatial/temporal behaviours (e.g. high-/low motion, coarse/fine level detail imagery etc.).

The evaluation of the proposed technique showed that the workload generators do not fully capture the extreme variations between real video stream frames due to several limitations outlined in Section 7.8.3. However, the properties such as reference frame dependencies, communication data patterns and decoding execution time bounds are comparable.

Many existing works in design-space exploration assume Gaussian/uniform distributed random workload patterns, which is not sufficient to model complex workloads as shown in the analysis presented in this chapter. The inter-task communication patterns (i.e. inter-frame reference characteristics) are shown to be highly dynamic and therefore, need to be accurately modelled when evaluating performance of communication-centric resource management policies. This extended application model is used in the following chapter to explore resource allocation techniques for HEVC encoded video streams.

Chapter 8

Task mapping for parallel HEVC tile decoding

The previous technical chapters of this thesis focused on the resource allocation problem for multiple video streams encoded using MPEG-2 video compression standard. In Chapter 7, the dynamic nature of complex, modern video coding standards such as HEVC was discussed in detail. HEVC was designed to efficiently compress video streams with ultra-high resolution (4K/3840×2160 and above) and makes use of several complex coding tools such as adaptive, hierarchical B-frame structures and different frame block structures (coding units). HEVC also introduces several native data-parallel decoding features such as tile-level partitioned decoding, where a frame is spatially partitioned by the encoder, such that each tile can be decoded in parallel to improve throughput at the decoder (Section 2.1.4.1). These new coding techniques make the workload highly dynamic and poses several resource management challenges as discussed in this chapter.

This chapter will explore several task mapping schemes for multiple soft real-time, HEVC stream decoding. This work will particularly focus on the decoding tile-parallel HEVC streams with adaptive GoP structures with dynamic frame dependency patterns, which are unknown a priori. The mapping heuristics also have to be robust enough to manage video streams with large variations in resolutions (i.e. a low 288p video and 2160p UHD video) as well as variations in their CCR due to the varying spatial and temporal video characteristics (e.g. slow/fast motion and/or high/low detail imagery as discussed in Chapter 7). In the case of UHD video decoding, the application model will have large computation and communication costs. In this case, it is necessary to exploit tile-level parallel decoding to reduce the latency of decoding a large frame. However, a higher level of parallelism increases the NoC usage leading to higher communication energy consumption.

To address the above issues, two runtime task-to-PE mapping techniques are presented in this chapter. They consider various factors such as application characteristics, blocking and task clustering/grouping in order to balance parallelism, reduce decoding lateness and NoC communication cost. The work in this chapter also shows that memory traffic latencies eventually become a severe bottleneck and cannot be addressed by task mapping alone. To this extent, several task-to-main memory controller port (MMCP) selection schemes are investigated to reduce memory traffic contention and further reduce the overall job response times.

The remainder of the chapter is organised as follows. Section 8.1 introduces the refined system model which explains the tile-level application model and job CCR derivation. Section 8.2 details the resource allocation problem this chapter is attempting to address. The proposed two runtime task mapping schemes are presented in Section 8.3, along with several of their variants. Section 8.4 introduces different MMCP selection schemes. In Section 8.5 the proposed

techniques are evaluated under different types of workloads.

8.1 System model refinement

A platform model similar to the one described in Section 5.1.2 is used in this chapter, where an open-loop centralised resource management scheme is used. The memory traffic is taken into account in the application and platform models, similar to Chapter 5. As described in Section 6.1.3, the resource manager in this work performs dynamic task mapping at the *job-level*, where each job in a video stream is given a different task-to-PE mapping configuration. Stream-level runtime mapping cannot be performed as the job dependency patterns are not known beforehand. Task priorities are assigned as per Section 5.1.1.2.

8.1.1 Application model refinements

The application model extensions defined in Section 7.1 form the basis of the application model in this chapter. The number of tasks within a job will not change in a video stream, but the task dependency structure will vary, based on the temporal correlation in the video sequence (Section 7.4). The model assumes video streams are encoded using hierarchical B-frame structures with a maximum of 4 contiguous B-frames and a maximum of 4 reference frames per B-frame. CU-level characteristics as described in Section 7.5 are used to derive the task execution cost model.

The application model assumes the video stream has tile-level partitioning enabled and the number of partitions per frame is determined by the encoder. The model assumes all frames in a stream have equal number of tiles and high resolution videos having a higher number of tiles per frame than lower resolution videos. Therefore, each frame-level decoding task τ_i consists of tile-level decoding *sub-tasks*, denoted as τ_i^j. The new tile-level job structure is then denoted as J_i^T. The tile decoding sub-tasks follow the same notation as the frame decoding tasks (e.g. x_i^j is the computation cost of the j^{th} tile of task τ_i). Tiles will inherit the real-time properties such as priorities, periods, sub-task deadlines from their frame-level task.

As described in Section 7.5 and Section 7.8.2, the HEVC frame computation cost and reference data volumes are characterised at the CU-level. Similarly, the tile-level computation costs (x_i^j) and reference data volumes can be easily derived by accumulating the computation cost and reference data for each CU in a tile. As per the standard [13], the model assumes there are approximately equal number of CTUs per tile. Different regions of a real video frame will have varying spatial detail and temporal correlations, leading to unbalanced computation and reference data dependencies amongst tiles in a frame. Similarly, the actual execution cost of the tiles in the frame model will vary, based on the size and type of CUs it has. The WCET of a tile c_i^j can be estimated via Eq. (8.1) as a ratio of the frames WCET; where $\tau_i^j(h) \times \tau_i^j(w)$ represents the tile dimensions and $\tau_i(h) \times \tau_i(w)$ is the frame dimensions. The worst-case payload for a reference frame can be calculated similar to the previous model as per Eq. (3.1) and the worst-case payload for a tile can be estimated as Eq. (8.2) where N_T denotes the number of tiles per frame.

$$\text{Tile WCET: } c_i^j = c_i \times \frac{\tau_i^j(h) \times \tau_i^j(w)}{\tau_i(h) \times \tau_i(w)} \tag{8.1}$$

$$\text{Tile reference data payload: } PL_{Msg_i^j} = \frac{PL_{Msg_i}}{(N_T)^2} \tag{8.2}$$

During frame to tile partitioning, the precedence constraints of the TG need to be unaffected. Hence, the number of edges and nodes in the TG will scale proportionally to the number of tiles (N_T) per frame decoding task (number of new edges = $|edges| \times (N_T)^2$). In the example given in Figure 8.1, a TG with 4 nodes and 3 edges is transformed into a *tile-level*, with 8 nodes and 12 edges. Note that the memory traffic flows are not displayed in Figure 8.1 but essentially each τ_i^j will have a memory read and memory write flow. The payload of the memory read will be the encoded size of a tile (modelled as per Section 7.6.2) and the memory write will be the decoded tile size, in bytes (i.e $\tau_i^j(h) \times \tau_i^j(w) \times bpp$). For the sake of clarity, henceforth, *tile-level* subtasks and their respective data flows will be commonly referred to as a *tasks* and *flows* respectively. Frames will be referred to as *frame-level tasks*.

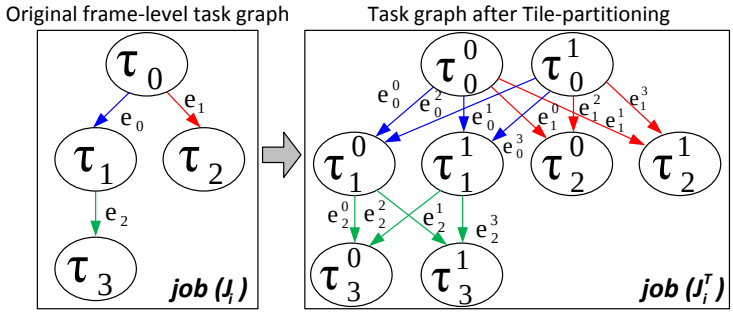


Figure 8.1: Illustration of original HEVC frame-level task graph before and after tile partitioning (assuming $N_T = 2$). Bottom example shows tile partitioning of a 15 frame GoP encoded using Hierarchical B-frames

8.1.1.1 Deriving the analytical CCR of a job

The resource allocation heuristics presented in this chapter considers the CCR of the jobs as a metric for deciding on task placement. The analytical CCR of a frame-level job (denoted $CCR(J_i)$) is calculated via Eq. (5.13). In order to calculate $CCR(J_i)$ the RM would require the following information from the video stream: the number of nodes and edges in a job, the task WCET and the maximum payload of a reference data flow (assumed as $res(VS_i) \times bpp$). The actual reference data flow payload and the actual task execution cost is unknown to the open-loop RM. Hence, the $CCR(J_i)$ calculation can be considered as an upper-limit of the job CCR. The exact reference data payload can be obtained by parsing each of the CTU/CU headers in the video bit-stream, but this adds additional runtime computational overhead and hence is avoided in this design. In the previous application model, the $CCR(J_i)$ does not vary between jobs due to fixed job task-dependency patterns. In this model, adaptive dependency structures are considered, hence the $CCR(J_i)$ will also vary between jobs of the same video stream.

The upper and lower bounds for $CCR(J_i)$ can be estimated if the video streams are encoded using hierarchical B-frames and the maximum contiguous B-frames (nB_{max}) and the maximum forward/backward reference frames ($rP_{max}^{fwd}, rB_{max}^{fwd}, rB_{max}^{bwd}$) are known. These parameters are set by the encoder at the video stream-level and typically do not vary between GoPs. The maximum

and minimum number of edges in a TG can be estimated via Eq. (8.5), where the maximum number of P and B frames in a GoP can be calculated via Eq. (8.3) and Eq. (8.4) respectively.

$$\text{Max/min P-frames in GoP: } nP_{MAX/MIN} = \{N-1, \frac{N-1}{BMAX+1}\} \text{ ,where N=GoP length} \tag{8.3}$$

$$\text{Max/min B-frames in GoP: } nB_{MAX/MIN} = \{(N-1) - nP_{MIN}, (N-1) - nP_{MAX}\} \tag{8.4}$$

$$\begin{aligned}\text{Max/min edges in TG} = \{&(nP_{MIN} \times rP^{fwd}_{max}) + (nB_{MAX} \times (rB^{fwd}_{max}, rB^{bwd}_{max})) \\ &, (nP_{MAX} \times 1) + (nB_{MIN} \times 2)\}\end{aligned} \tag{8.5}$$

8.2 Problem statement

The application model refinements introduced in the previous section bring upon several challenges to the tile-parallel HEVC video stream decoding, resource allocation problem. In this section the following issues are discussed:

- Two conflicting mapping objectives: job lateness reduction and NoC usage (i.e. inter-task communication cost) reduction.
- Varying job dependency patterns due to adaptive GoP structures and increase in task graph scale due to tile partitioning.
- Varying video stream computation and communication requirements.
- Memory traffic bottlenecks.

8.2.1 Balancing predictability and energy consumption

Improving predictability of SRT video stream decoding in this work, comes in the form of reducing the variability, mean and maximum job lateness. Energy consumption reduction can be obtained in two ways. Firstly by reducing the communication cost, i.e by limiting the NoC usage. Secondly by exploiting variability in utilisation of the PEs across the platform, such that dynamic power management techniques can be employed efficiently. In certain SRT/best-effort video decoding use-cases, reducing energy consumption maybe more crucial than video stream latency. For example, playback delays up to 30s for long video sequences have shown to be tolerable by users [65], but inefficient battery usage (e.g. in a mobile device) is not acceptable. As predictability and energy consumption are most often conflicting goals, in this work several dynamic mapping techniques are explored to *balance* the two metrics rather than optimise on one or both. This subsection will detail the difficulties in optimising job lateness and communication cost reduction.

Job lateness can be reduced when the mapping configuration takes into account task and flow contention. Higher amounts of interference directly increase the job lateness. Note that such interference can be from tasks/flows of the same job (commonly referred to as intra-interference) or tasks/flows from other jobs of different streams (commonly referred to as inter-interference). Communication costs (i.e. cumulative flow basic latencies) are typically reduced by task clustering/grouping, where multiple communicating tasks are placed on the same PE in order to reduce the number of flows injected into the NoC. Placing tasks closer together results in shorter routes, and also helps to reduce communication costs. The clustering can be

tight/loose depending on the level of task dispersion and the size of the region occupied by communicating tasks. However, if the clustering is too tight the amount of inter/intra-interference can increase and the system can start to experience contention hot-spots; even though the communication cost can be largely reduced.

The workload CCR should also be considered when deciding on the level of task clustering. Low CCR (i.e computationally heavy) workloads may require more parallelism (i.e. loose clustering) in order to meet deadlines than high CCR workloads. Therefore, the same trade-off made between job lateness and communication cost reduction for high CCR workloads may not be achievable for low CCR workloads. For high CCR workloads, tighter task clustering may result in both lateness and communication cost reduction.

Unlike static mapping techniques, dynamic mapping heuristics need to have low mapping execution overhead and have limited application and platform knowledge. Intelligent runtime heuristics that take into account different factors such as task/flow interference, degree of clustering, workload CCR etc. can be effective in balancing the mapping objectives, but can also be computationally expensive. Therefore, designing a lightweight dynamic mapping heuristic that takes into account all the factors discussed above is a challenging research problem.

8.2.2 Dynamic GoP-structures and task graph scale

The adaptive GoP nature of video streams poses a new challenge to the mapping problem previously not considered. As job dependency patterns and the number of P/B frames in a job are not fixed, the task mappers need to be invoked at the arrival of each job. LWCRS and IPC runtime mappers proposed in Chapter 5, are not applicable in this case. LWCRS performs tight temporal packing of tasks to reduce the worst-case job response time and increase the D-AC's admission rate. The IPC mapper relies on a fixed dependency structure to reduce the response time of the longest path of the job, which would not be efficient if the job dependency structure is dynamic (specifically in the case of high number of contiguous B-frames).

Consider the hierarchical B-frame encoded, 15 frame GoP structure shown in Figure 8.2. After tile partitioning, the tile-level TG will have significantly higher number of nodes and edges. As another example, a job with 31 frame-level tasks, 54 edges and 10 tiles per frame, can result in a tile-level TG representation with 310 tasks and 5400 edges. This large increase in the number of tasks and especially communication flows, induces more interference to lower priority tasks and flows already existing in the system. Therefore, the task mapping should take into account the blocking when selecting a PE to map a task to. Considering the inter-task communication distance also becomes challenging, as tasks in the tile-level TG may have a larger number of parent tasks with varying reference data volumes. Grouping communicating tasks may not be trivial due to interference issues mentioned before. Attempting to allocate resources for a large number of tasks and flows can lead to high mapping overheads, if the mapping heuristic is very sophisticated. For example, taking into account multiple factors such as flow/task blocking, communication distance, task clustering etc. can yield better mapping quality, but can also lead to high mapping execution overhead.

8.2.3 Varying video stream CCRs

Due to the variation in the GoP structures, different jobs of the same video can have varying CCR levels. Furthermore, the CCR can also vary based on the stream type. From the real video stream

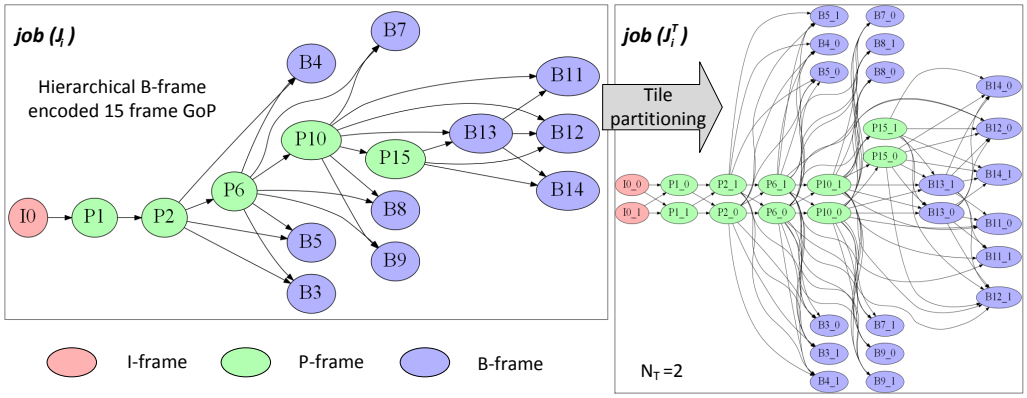


Figure 8.2: Example of tile partitioning of a 15 frame GoP encoded using Hierarchical B-frames

analysis in Chapter 7, it was seen that high motion videos such as *LionWildlife/FastFurious5* have high decoding execution cost and low data communication properties (e.g. reference data volume, number of B-frames and contiguous B-frame sizes), leading to low CCR conditions. The inverse is true for low motion video sequences such as in *ObamaSpeech*, where higher data communication and lower amount of computation cost was seen; these type of videos display high CCR levels.

Figure 8.3 illustrates the job CCR $CCR(J_i)$ variation of different synthetically generated HEVC video streams (parameters obtained from Chapter 7 stream analysis). The $CCR(J_i)$ distributions are categorised by resolutions and sub-categorised by *video type* (genre) and GoP lengths (N). In the figure, the video types represent the streams that were analysed in Chapter 7. ACTION, DOCUMENTARY (DOC), SPORT, ANIMATION (ANIM) and SPEECH represent the FastFurious5, LionWildlife, Football, BigBuckBunny and ObamaSpeech videos respectively. A breakdown of the CCRs in terms of the individual computation and communication costs can be found in Appendix D.1. The line plot in Figure 8.3 denotes the upper/lower bound CCR calculation as per Section 8.1.1.1. Whilst different resolutions do not vary much in CCR, low motion videos (e.g. SPEECH) can have CCRs twice that of high motion videos (e.g. ANIM/ACTION). GoPs with higher number of frames can also show a slight increase in CCR values than GoPs with lower number of frames.

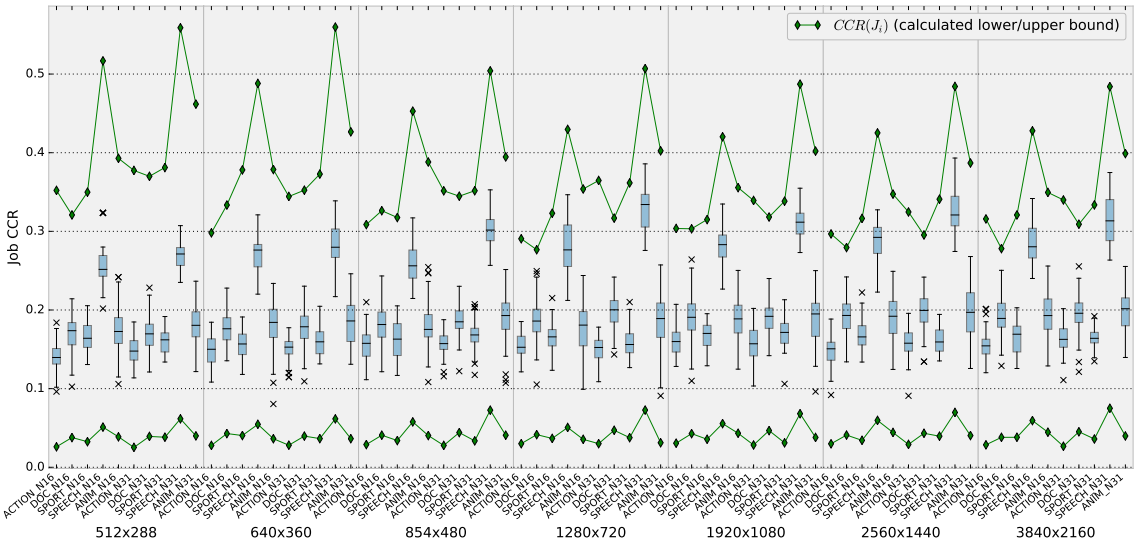


Figure 8.3: Varying job CCR ($CCR(J_i)$) for different video streams. Varying resolutions, GoP lengths *(N=16,31)* and video types *(ACTION, DOC, SPORT, SPEECH, ANIM)*. (80 GoPs per video))

Higher CCR workloads can give rise to communication bottlenecks and hence tasks require dense clustering to reduce the usage of the NoC (thereby reducing congestion). On the other hand, workloads with a low CCR need to be parallelised (i.e. sparsely distributed) in order to achieve required throughput. For low CCR jobs the mapper can make use of HEVC tiles to decode a single frame on multiple PEs in parallel. Deciding the degree of parallelism for a given CCR and the low/high ranges of CCR for an arbitrary workload is non-trivial, specially at runtime.

This chapter also explores the performance of the mappers, when the CCR level of the overall video stream decoding workload is increased (independent to stream characteristics). An example where the *workload CCR* may increase is when the computation cost of decoding a frame is significantly decreased. For example, when the computationally intensive functional components of the decoder are either optimised or implemented in hardware rather than software (e.g. [210], [211]). In these instances the computation cost of the frame decoding tasks will decrease whilst not affecting the communication cost. Processor and NoC bandwidth differences could also give rise to different CCRs.

When the computation cost is reduced (i.e. CCR is increased), the rate flow injection into the NoC will also increase; resulting in higher NoC contention. This can increase the end-to-end job response time even though the computation cost has been reduced, and effectively the flow latency becomes a bottleneck. The mapping techniques should be designed in such a way, that it is either easy to adapt the level of clustering based on the workload CCR or it should dynamically adapt to the CCR of the workload at runtime. The latter implies more sophisticated heuristics, which may result in higher mapping overhead.

8.2.4 Efficient memory traffic management

Based on the task to PE mapping configuration the level of contention and congestion on NoC links will vary. An example arbitrary mapping of 2 jobs on a NoC platform is shown in Figure 8.4. In this illustration, the links are colour coded according to the amount of congestion (i.e. with respect to the number of flows and total payload). In the figure, RT_i denotes the routers and $MMCP_i$ denotes the MMCPs. Note than in the platform model each MMC has 2 ports. In the figure, for convenience, the MMC ports are displayed but not the MMCs themselves (e.g. $MMCP_0$ and $MMCP_1$ is connected to the same north-MMC). It can be seen that in majority of the cases, the local links (i.e. $PE_i \leftrightarrows RT_i$) and the MMCP links (i.e $RT_i \leftrightarrows MMCP_i$) are heavily congested. Especially the $RT_i \rightarrow MMCP_i$ links are heavily congested as memory write flows have a much larger payload than memory read or data flows.

The local-link congestion occurs, when the clustered tasks are placed on the same PE and the each of the task's incoming/outgoing flows interfere with each other. This issue, along with general data traffic NoC congestion can be alleviated by mapping approaches which manage/-control the level of task clustering efficiently. The congestion issue due to memory traffic however may not always be solved by task mapping. For example in Figure 8.4, placing the $B_{1,3}$ task on PE_6 instead of PE_2 can help more evenly balance the NoC data traffic congestion, but the $RT_2 \rightarrow MMCP_1$ link may still be congested as $B_{1,3}$ would still communicate with $MMCP_1$ due to close proximity. However, if task $B_{0,2}$ communicated with $MMCP_2$ the $RT_2 \rightarrow MMCP_1$ link congestion can be reduced. Similarly, if some of the tasks in PE_1 communicated with $MMCP_7$ instead of $MMCP_0$, then the congestion on link $RT_1 \rightarrow MMCP_0$ can also be reduced. The reduction in congestion will in turn lead to lower flow and job response-times. Therefore, the resource allocation

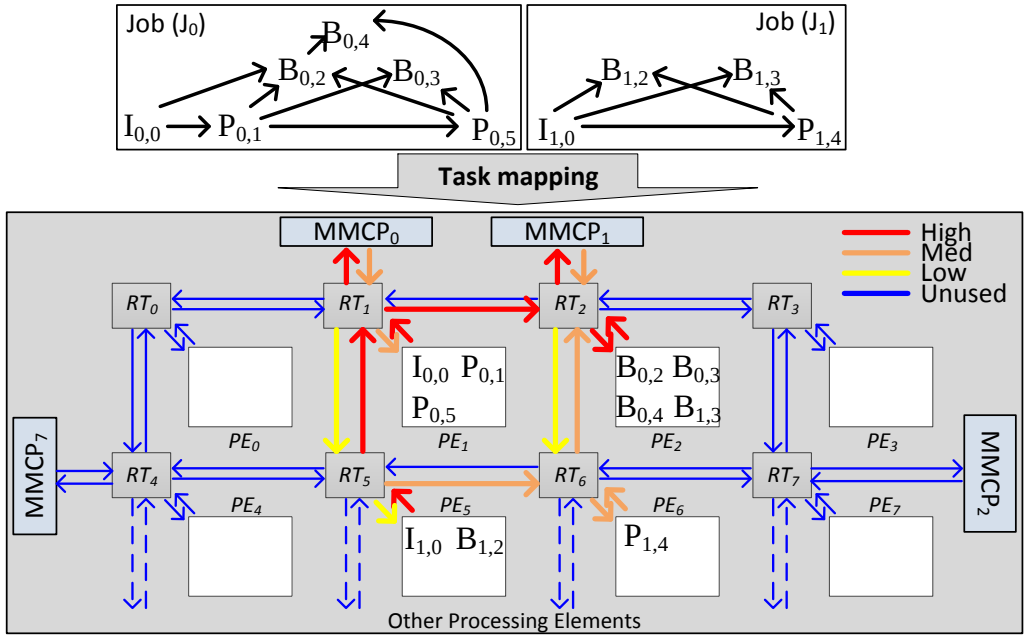


Figure 8.4: Example showing NoC link congestion. Assume the 2 Jobs (J_0 and J_1 are from different video streams).

protocol should also consider efficient task-to-MMCP mapping schemes (also simply referred to as *MMCP selection schemes*). As these MMCP selection heuristics will add further overhead to the task mapping process, it is crucial they are lightweight.

8.3 Tile mapping schemes for HEVC stream decoding

This section introduces several runtime tile-level task mapping heuristics which attempt to address the problems outlined in Section 8.2. The terms *tile mapping* and *task mapping* are used interchangeably in this chapter. The primary concerns of the mapping heuristics can be categorised as follows:

- Determine which tasks to cluster/group together to reduce communication cost.
- Determine the level of task clustering (sparsely/densely) to balance parallelism and communication cost.
- Adapt the level of clustering based on the CCR of the video stream jobs.
- Determine the distance between the child and parent tasks as well as the distance between tiles of the same frame-level task.
- Take into account task interference induced to already active/mapped lower priority tasks when selecting a PE.

To this extent, two primary tile mapping heuristics are introduced. The first heuristic does not make any assumptions regarding the length/structure of the GoP and hence, can be used to map complex, random dependency patterns. The second heuristic, is fine-tuned to target video streams encoded using adaptive, hierarchical B-frames. Hence, it is relatively less robust than the first but can be used to give better mapping results for the aforementioned type of videos. Fast variations of these heuristics are also presented for comparison.

Similar to Chapter 5, the mapping schemes proposed in this chapter follows an open-loop resource management technique. Therefore, the runtime task mapping table TMT is used to track previous mappings and approximately determine the worst-case utilisation of the PEs at runtime.

8.3.1 Generic longest-path aware tile task mapping (CL)

The first proposed task clustering-based mapping scheme (denoted ***CL***) groups the tasks based on the TG's longest-path. Algorithm 8.1 presents the pseudo-code of the CL algorithm. The primary level of clustering is achieved by grouping the frame-level tasks that lie on the *longest-path* (denoted $Lpath(J_i)$) of the TG (line 1). The longest path of a topologically sorted TG with no weights can be obtained in linear time [198]. In Algorithm 8.1, τ_0^0 denotes the first tile of the first frame (usually an I-frame) in the job; τ_i^0 denotes the first tile of tasks $\tau_i^j \notin Lpath(J_i)$.

Algorithm 8.1: CL: CCR and blocking-aware clustered tile mapping pseudo-code

Input : J_i = topologically sorted frame-level task graph of i_{th} job,
J_i^T = topologically sorted tile-level task graph of i_{th} job,
TMT : the runtime task mapping table,
PE_list : list of PEs in the platform
Output: Updated TMT with new tile to PE mapping ($\tau_i \rightarrow PE_i$)

```
/* Find the frames-level task subset that is in the longest path of the job */
1  Lpath(J_i) = getLongestPath(J_i)
/* Determine the CCR specific hop count parameters */
2  Calculate analytical CCR of the job : CCR(J_i)
3  {NH_T, NH_CT, NH_CH} = getCCRspecificHopCount(CCR(J_i))
/* map each tile task iteratively */
4  foreach τ_i^j ∈ J_i^T do
       /* manage task clustering density based on if task is part of Lpath(J_i) or not */
5      if τ_i^j ∈ Lpath(J_i) then
           /* place first tile of first task (τ_0^0) on PE with minimum lower priority tasks; place other
              tiles close to τ_0^0 */
6          if τ_i^j == τ_0^0 then
7              PE_i = getPEwithMinLPTasks(TMT, τ_0^0, PE_list) // constrained PE search
8          else
9              sPE_N == getPENeighbours(PE(τ_0^0), NH_CT)
10             PE_i = getPEwithMinLPTasks(TMT, τ_i^j, sPE_N) // constrained PE search
11         end
12     else
           /* place first tile of non Lpath(J_i) tasks (τ_i^0) closest to its parent with most probable
              data dependency. Other tiles mapped closer to τ_i^0 */
13         if τ_i^j == τ_i^0 then
14             P_τ_i^j = parent task with largest probable data dependency
15             sPE_N = getPENeighbours(PE(P_τ_i^j), NH_CH )
16             PE_i = getPEwithMinLPTasks(TMT, τ_i^j, sPE_N ) // constrained PE search
17         else
18             sPE_N = getPENeighbours(PE(τ_i^0), NH_T )
19             PE_i = getPEwithMinLPTasks(TMT, τ_i^j, sPE_N ) // constrained PE search
20         end
21     end
22     Map τ_i^j → PE_i ; Update TMT{PE_i, τ_i^j}
23 end
```

The algorithm, iteratively assigns tiles to PEs in topological order (similar to frame decoding order). At each iteration a constrained set of PEs (sPE_N) are obtained according to the maximum hop-distance specified in Table 8.1. The hop distances control the level of task clustering and are chosen according to the $CCR(J_i)$. Higher number of hops will result in larger number

of PEs to be considered (more dispersion). As shown in Table 8.1, the algorithm can be configured to increase the hop parameters slightly, when the cumulative cost of the TG longest-path is higher than the job end-to-end deadline (i.e. $cost(Lpath(J_i)) > D_{e2e}$). If this condition is true, it indicates the video stream requires more parallelism; often in the case of UHD video streams with very high computational and communication requirements. sPE_N is then searched to obtain the PE with the minimum number of mapped lower-priority tasks with respect to the target task τ_i^j; such that interference to already mapped tasks can be minimised. This is done via the *getPEwithMinLPTasks()* helper function (lines 6, 9, 15 and 18).

Table 8.1: CL task clustering hop count parameters based on CCR of a job $CCR(J_i)$

CCR range	**IF** $[cost(Lpath(J_i)) \leq D_{e2e}]$, $\{NH_{CT}, NH_T, NH_{CH}\}$	**IF** $[cost(Lpath(J_i)) > D_{e2e}]$, $\{NH_{CT}, NH_T, NH_{CH}\}$
Low ($CCR(J_i) < 0.18$)	{Max hops, Max hops, Max hops}	{Max hops, Max hops, Max hops}
Med ($0.18 \leq CCR(J_i) \leq 0.23$)	$\{2, \max(\frac{NOC_W}{2}, N_T), \max(\frac{NOC_W}{2}, N_T)\}$	$\{\frac{N_T}{2}, N_T, \frac{N_T}{2}\}$
High ($CCR(J_i) > 0.23$)	{2, 2, 2}	$\{\frac{N_T}{2}, \frac{N_T}{2}, \frac{N_T}{2}\}$

Figure 8.5 has been provided to illustrate the NH_T, NH_{CH} and NH_{CT} hop distance parameters. The figure shows 5 frame-level tasks with $N_T = 3$ tiles per frame. NH_T denotes the maximum hop distance between $\tau_i^j \notin Lpath(J_i)$ and τ_i^0 (the initial tile of the respective frame-task τ_i). NH_{CT} is the maximum hop distance between the $\tau_i^j \in Lpath(J_i)$ and τ_0^0. NH_{CH} denotes the maximum hop distance from a $\tau_i^0 \notin Lpath(J_i)$ and their parent task with the highest probable data dependency ($P\_\tau_i^j$). The RM estimates $P\_\tau_i^j$ intuitively using the reference data volume analysis in Section 7.6.1 (i.e. $p_I > p_P > p_B$).

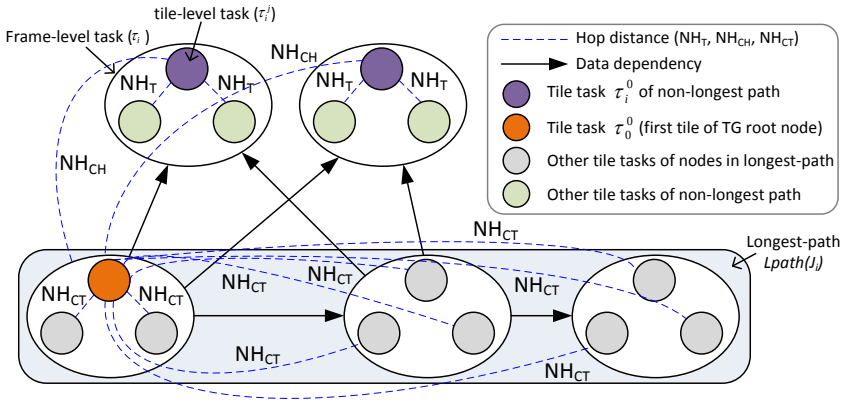


Figure 8.5: Illustration of CL clustered mapper hop parameters

Sparse clustering (i.e larger hop parameters) is performed for low-CCR jobs, in order to increase data-parallelism. For high-CCR (e.g. communication-heavy) jobs, the clustering density is increased (i.e. small hop counts) to reduce communication-energy and network interference. The low/medium/high CCR ranges as shown in Table 8.1 are estimated from the analytical CCR calculations as per Section 8.1.1.1 and Figure 8.3. These ranges, and associated hop counts are parameters that can be adjusted to achieve the required balance between communication cost and job lateness reduction. The hop counts shown in Table 8.1 gave a reasonable trade-off between the two metrics. The number of CCR ranges can also be changed if more knowledge about the workload scenarios are known a priori.

Figure 8.6 shows an example of tile mapping on a 4x4 NoC using the proposed CL mapper and the least utilised (LU) mapper. Both mapping schemes, scatter the low CCR job's tasks over

the PEs. The CL mapper has constrained about 1/3 of the tasks of the medium CCR job to PE(0,0) and the rest have been placed in other regions of the NoC in order to balance clustering vs. spreading. The high CCR job has been allocated mainly to 5 PEs in the system - PE(0,1), PE(1,1), PE(0,2),PE(1,0) and PE(2,0); tighter groupings using less number of PEs would increase the contention in the system, hence is avoided by CL. LU produces an even load distribution, regardless of the CCR of the stream, causing increased NoC usage.

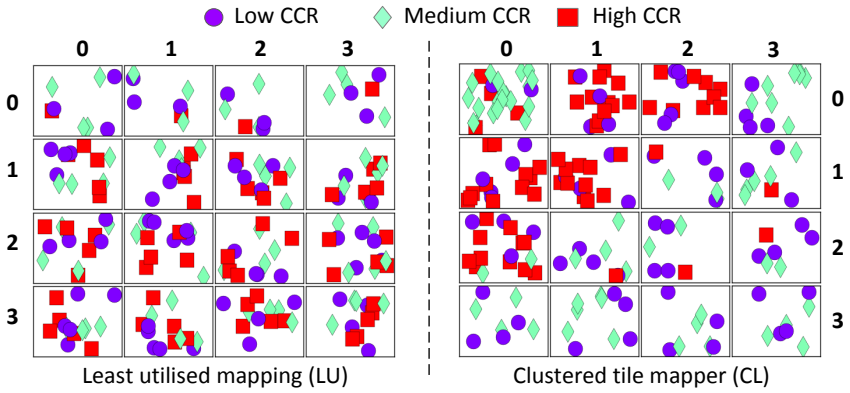


Figure 8.6: Task allocation example of least-utilised PE mapper (LU) vs the proposed cluster-based mapper (CL). 4x4 NoC, 3 video streams, 1 GoP each. Markers represent individual tasks

8.3.1.1 CL mapping complexity analysis

The CL mapper algorithm complexity is dependent on the number of tasks in a job $|J_i^T|$, number of PEs $|PE|$ and number of tasks in a task queue $|TQ|$. The worst-case complexity would be $O(|J_i^T| \times |PE| \times |TQ|) \equiv O(n^3)$. Realistically $|TQ| \ll |PE|$, even under very high workloads, as the admission controller would detect the system is overloaded and start to reject new jobs/video streams. Therefore, the worst-case scenario can be considered unrealistic.

8.3.1.2 Variants of the CL tile mapper

Three variants of the CL mapper are presented in order to reduce its execution overhead and to evaluate the usefulness of CCR based task clustering.

The first variant, termed ***CL-F*** is a faster $O(n^2)$ implementation of CL, where the PEs are first sorted in increasing order of number of lowest priority mapped tasks. This fast version is more suited to map streams with shorter GoP lengths (i.e. lower inter-job arrival time). The algorithm pseudo-code of CL-F is shown in Algorithm 8.2. The sorted PE list is used in each iteration of the main loop (lines 5-26). Unlike in CL, CL-F does not invoke *getPEwithMinLPTasks()* at each iteration, to reduce the complexity of the mapping, but as a result loses mapping accuracy. In lines 10 and 18 of Algorithm 8.2, the sorted PE list $sorted(PE\_list)$ is used and the speed-ups are gained when selecting a PE for $\tau_i^j \in Lpath(J_i)$ (line 12) and $\tau_i^j \notin Lpath(J_i)$ (line 22).

The second variant of CL, termed ***CL-NoCCR*** is exactly similar to Algorithm 8.1, but the hop distance parameters (i.e. NH_T, NH_{CH}, NH_{CT}) are fixed rather than varying them based on the job CCR. This results in the distribution of the tasks on the PEs to be independent of the CCR of the job. As a preliminary investigation the parameters are fixed at $NH_T = N_T$, $NH_{CT} = 1$ and $NH_{CH} = 1$. A comparison of CL vs. CL-NoCCR will allow us to evaluate the benefit of adaptive, CCR-aware clustering over a fixed task clustering approach.

Algorithm 8.2: CL-F: Fast first-fit increasing variant of the CL mapper

Input : J_i : topologically sorted frame-level task graph of i_{th} job,
J_i^T : topologically sorted tile-level task graph of i_{th} job,
TMT : the runtime task mapping table,
PE_list : list of PEs in the platform
Output: Updated TMT with new tile to PE mapping ($\tau_i \rightarrow PE_i$)

/* Find the frames-level tasks in the longest path of the job */
$Lpath(J_i)$ = $getLongestPath(J_i)$;
Sort PE_list inc. order of num. of low-pri. tasks w.r.t τ_0^0 and store in $sorted(PE\_list)$
/* Determine the CCR specific hop count parameters */
Calculate analytical CCR of the job : $CCR(J_i)$
$\{NH_T, NH_{CT}, NH_{CH}\} = getCCRspecificHopCount(CCR(J_i))$
/* map each tile task iteratively */
foreach $\tau_i^j \in J_i^T$ **do**
  /* manage task clustering density based on if task is part of $Lpath(J_i)$ or not */
  if $\tau_i^j \in Lpath(J_i)$ **then**
    /* place first tile of first task (τ_0^0) on PE with minimum lower priority tasks; place other tiles close to τ_0^0 */
    if $\tau_i^j == \tau_0^0$ **then**
      $PE_i = sorted(PE\_list)[0]$ // get first element
      sPE_N = $getPENeighbours(PE_i, NH_{CT})$
      Sort sPE_N according to $sorted(PE\_list)$; store in circular list : $sorted\{sPE\_N(\tau_0^0)\}$
    else
      $PE_i = sorted\{sPE\_N(\tau_0^0)\}.next()$ // get next element in circular list
    end
  else
    /* place first tile of non $Lpath(J_i)$ tasks (τ_i^0) closest to its parent with most dependent data. Other tiles mapped closer to τ_i^0 */
    if $\tau_i^j == \tau_i^0$ **then**
      $P\_\tau_i^j$ = parent task with largest probable data dependency
      sPE_N = $getPENeighbours(PE(P\_\tau_i^j), NH_{CH})$
      PE_i = sort sPE_N according to $sorted(PE\_list)$ and get first element
      ot_sPE_N = $getPENeighbours(PE_i, NH_T)$, (circular list)
    else
      PE_i = $ot\_sPE\_N.next()$ // get next element in circular list
    end
  end
  Map $\tau_i^j \rightarrow PE_i$; Update $TMT\{PE_i, \tau_i^j\}$
end

The third variant of the CL mapper is similar to the IPC mapper presented in Chapter 5, where the longest-path frame subset ($Lpath(J_i)$) is assumed to be the clustering of I and P frames. This variant of CL is termed ***CL-IPC*** and it avoids calculation of the longest-path of the TG. Apart from the $Lpath(J_i)$ selection criteria, all other algorithmic characteristics of CL-IPC is similar to CL.

8.3.1.3 Limitations of the CL tile mapper

A drawback of the CL mapping heuristic is the number of parameters that requires tuning to obtain good performance. The 3 hop parameters for 3 CCR ranges which gives $(MAX\_HOPS)^9$ permutations where MAX_HOPS denotes the maximum hop distance in the NoC. The number of parameters are doubled if a separate set of parameters are chosen for the $cost(Lpath(J_i)) > D_{e2e}$ case. Hence, finding parameter combinations that balance communication cost and lateness may be cumbersome. Generally, better results were seen when NH_{CT} was lower than NH_T or NH_{CH} (i.e. the $Lpath(J_i)$ tasks are densely clustered than others). The robustness and generic nature of the CL mapper also gives rise to some limitations. The clustering can be too dense for

video stream GoPs with a high number of P-frames all lying in the job's longest path. In dense clustering scenarios task-contention can become more severe than flow-contention.

8.3.2 B-frame grouping-aware tile task mapping (CL-BG)

The second tile mapping heuristic proposed in this chapter attempts to target the native B-frame grouping property seen in videos encoded using adaptive, hierarchical B-frames. This clustered tile mapping heuristic is termed ***CL-BG***. The hierarchical B-frame grouping structure seen in GoPs were discussed in Section 7.4, and examples illustrated in Figure 7.4. In these structures, B-frames can be used as reference by other B-frames (generalised B-frames). In Chapter 7, it was observed that in these GoP structures, groups of B-frames are separated by P-frames and due to high temporal correlation between frames the B-frame groups only refer to frames within the group. Similarly, P-frames tend to only refer to other I/P frames immediately before it. Note that these observations are typically true for videos encoded with lower number of reference frames targeted towards systems with low-memory and low-decoder complexity (e.g. smart phones).

To illustrate the B-frame grouping property, consider the example hierarchical B-frame GoP structure shown in Figure 8.2. In the frame group P_{10} to P_{15}, there are 6 frames (2 P-frames and 4 B-frames). The B_{11} to B_{14} group of B-frames lie between P_{10} and P_{15}. A high number of in/out edges exist between these 6 frames. A similar grouping behaviour can be seen in frames P_2 to P_6. In hierarchical B-frame GoPs, a significant amount of communication occurs between the B-frame groups and the P-frames surrounding the B-groups. This inherent grouping property in hierarchical B-frames, is exploited in this specialised mapping heuristic, to form task clusters surrounding these B-frame groups.

Figure 8.7 *(top)* shows how multiple task clusters can be formed at I/P-frame boundaries with each cluster having a maximum of 1 P-frame. In all instances the first P-frame (e.g. P_1) and the subsequent B-frame group (e.g. B_2 to B_4) will be clustered together, whilst the following P-frame (e.g. P_5) will be placed in the next cluster. In this preliminary version of CL-BG, isolated P-frames (e.g. P_{13}, P_{14}, P_{15}) are clustered individually as single element clusters; however, future work can explore merging consecutive P-frames. As shown in Figure 8.7 *(top)*, each cluster has a *cluster primary frame-level task* which is either the I or P frame in the cluster. In the example, $I_0, P_1, P_5..P_{15}$ are primary frame-level tasks of each cluster; their corresponding tile-level primary tasks (e.g. the first tile of the primary task : I_0^0, P_1^0 etc.) are denoted as cl_{pt}. It can be observed that the cluster primary task has a high number of out-going edges to its child tasks. Algorithm 8.3 shows the pseudo-code to perform this clustering, which sequentially places each frame in the GoP into a separate cluster.

Once the clusters have been formed they are mapped onto the NoC as shown in Figure 8.7 *(bottom)*. Two parameters are used in the mapping phase of the algorithm. The cluster region hop distance parameter (denoted as NH_{GT}) controls the distance between the cl_{pt} and the other cluster tiles. The inter-cluster hop distance parameter (denoted as NH_{GP}) controls the distance between primary tasks of different clusters (i.e. inter-cluster distance). Distance between clusters must be kept short in order to reduce long communication routes but clusters must reduce overlapped regions to avoid inter-cluster interference (i.e. $NH_{GP} > NH_{GT}$). For this reason, NH_{GP} is set as $(NH_{GP} = NH_{GT} + 1)$, thus reducing the complexity of the parameter tuning process.

Algorithm 8.4 shows the pseudo-code of the CL-BG mapping technique. In line 1 of the algorithm the cluster hop parameters are obtained according to the $CCR(J_i)$. For this work, the

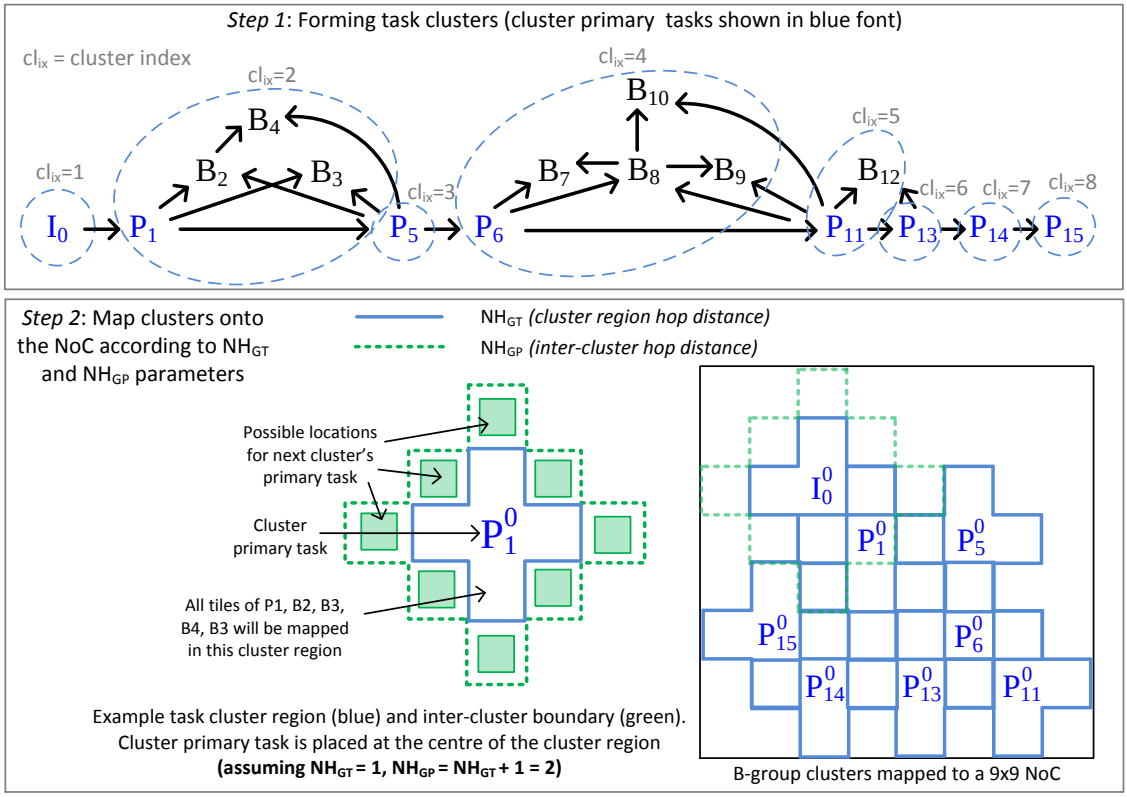


Figure 8.7: Illustration of hierarchical B-frame grouping-aware task clustering. *Top:* Virtual clustering of frame GoPs, *Bottom:* Mapping of clusters onto a 9x9 NoC

parameters given in Table 8.2 gave the best compromise between job lateness and communication cost reduction. Similar to CL, the hop counts are increased by 1 if $cost(Lpath(J_i)) > D_{e2e}$ is true, where in CL-BG, the $Lpath(J_i)$ is assumed to be the I and P-frame combined task chain. A comprehensive tuning of the NH_{GT} parameter can be found in Appendix D.2. The cl_{pt} of the first cluster is mapped on the PE with the minimum number of low-priority tasks (line 6), and the rest of the tasks in the first cluster is mapped according to the PE utilisation and at NH_{GT} hop distance away from the cl_{pt}. The algorithm attempts to map each cluster in regions which have the least occupancy of tasks, by ranking each possible PE with respect to the unused number of neighbouring PEs (lines 17-27). Non-primary tasks of the clusters are mapped similar to the first cluster (lines 28-31).

8.3.2.1 CL-BG mapping complexity analysis

The cluster formation in CL-BG (Algorithm 8.3) mainly consists of a single loop and several conditional statements, thus its runtime complexity is $O(|J_i|)$. The CL-BG mapping algorithm (Algorithm 8.4) is similar to CL, where it has $O(|J_i^T| \times |PE| \times |TQ|) \equiv O(n^3)$ worst-case complexity. Note that the NH_{GT} parameter limits the PE search space, and is usually kept small to balance communication and lateness; hence, the average case complexity of CL-BG can be considered as $O(|J_i^T| \times |TQ|) \equiv O(n^2)$ and the worst-case complexity is impractical.

8.3.2.2 Fast variant of CL-BG

Similar to the CL mapper, a fast variant of CL-BG termed ***(CL-BG-F)*** is also explored. CL-BG-F is similar to CL-F (Algorithm 8.2), where the PEs are initially sorted by their mapped number of

Algorithm 8.3: CL-BG: clustering frames according to hierarchical B-frame groupings

Input : J_i : topologically sorted frame-level task graph of i_{th} job
Output: $clustered(J_i)$: clustered GoP frames

$tmp\_c = \{\}$ // temporary single cluster list
$clustered(J_i) = \{\}$ // all clusters 2D list
/* Group frames iteratively */
foreach $\tau_i \in J_i$ **do**
  if $[(f_t(\tau_i) == I)$ **or** $((f_t(\tau_i) == P)$ **and** $(|tmp\_c| == 0))]$ **then**
    Insert $\tau_i \rightarrow tmp\_c$
  else if $[(f_t(\tau_i) == P)$ **and** $(|tmp\_c| > 1)$ **and** $(fr_{ix}(\tau_i) == |J_i| - 1)]$ **or** $[(f_t(\tau_i) == P)$ **and** $(|tmp\_c| == 1)$ **and** $(fr_{ix}(\tau_i) == |J_i| - 1)]$ **then**
    Insert $\{tmp\_c, \{\tau_i\}\} \rightarrow clustered(J_i)$ // insert τ_i as a single element set
  else if $[(f_t(\tau_i) == P)$ **and** $(|tmp\_c| > 0)]$ **then**
    Insert $tmp\_c \rightarrow clustered(J_i)$; $tmp\_c = \{\}$ // reset
    Insert $\tau_i \rightarrow tmp\_c$
  else if $[(f_t(\tau_i) == B)$ **and** $(|tmp\_c| > 0)$ **and** $(fr_{ix}(\tau_i) == |J_i| - 1)]$ **then**
    Insert $\tau_i \rightarrow tmp\_c$
    Insert $tmp\_c \rightarrow clustered(J_i)$
  else if $[(f_t(\tau_i) == B)$ **and** $(|tmp\_c| > 0)]$ **then**
    Insert $\tau_i \rightarrow tmp\_c$
end
Return $clustered(J_i)$

Table 8.2: CL-BG task clustering hop count parameters based on CCR of a job $CCR(J_i)$

CCR range	NH_{GT}	$NH_{GP} = NH_{GP} + 1$
Low ($CCR(J_i) < 0.18$)	4	5
Med ($0.18 \leq CCR(J_i) \leq 0.23$)	1	2
High ($CCR(J_i) > 0.23$)	1	2

low-priority tasks, and then sequentially selected for each mapped tile task. Therefore, CL-BG-F does not call the *getPEwithMinLPTasks()* and *getLowestUtil()* at each iteration in the algorithm, thus greatly reducing the execution overhead.

8.4 Smart memory controller selection

This section introduces several heuristics that can be used for MMCP selection, in order to reduce the memory traffic congestion problem as described in Section 8.2.4. A task's main memory read/write transactions would go via the NoC through the assigned MMCP. Reducing memory traffic contention is very challenging as there are only 8 MMCPs in the platform, and potentially hundreds or thousands of flows that can utilise these MMCPs simultaneously. The tasks in a job are first mapped to PEs and then they are each assigned a MMCP at runtime. The following MMCP selection heuristics are explored in this chapter.

- Closest MMCP selection **(MMCP-Dist)**:

  In this heuristic, the MMC with the shortest route between source ($src = \tau_i^j$) and destination ($dst = MMCP_i$) is selected. This is the default MMC port selection scheme used in the current system model and only considers the distance (i.e. number of hops/links) between task location and MMCP.

- Least utilised route selection **(MMCP-LU)**:

  In this heuristic, for each task τ_i^j, the utilisation of the route (i.e. the cumulative payload of the flows using the route as per Eq. (8.6)) between $\tau_i^j \rightarrow MMCP_i$ is considered. In Eq. (8.6), the total worst-case payload ($w.c.PL(Msg_i)$) and period T_i of all flows on all links in the route

Algorithm 8.4: CL-BG mapping algorithm pseudo-code

Input : J_i : topologically sorted frame-level task graph of i_{th} job,
J_i^T = topologically sorted tile-level task graph of i_{th} job,
TMT : the runtime task mapping table,
PE_list : list of PEs in the platform,
$clustered(J_i)$: clustered frames using Algorithm 8.3
Output: Updated TMT with new tile to PE mapping ($\tau_i \rightarrow PE_i$)

$\{NH_{GT}, NH_{GP}\} = getCCRspecificHopCount\_CLBG(CCR(J_i))$
/* Iteratively, map tiles to PEs according to clusters */
foreach $\tau_i^j \in J_i^T$ **do**
  Get cluster index cl_{ix} for τ_i^j; Get cluster primary task cl_{pt} for cl_{ix}
  /* first cluster */
  if $cl_{ix} == 0$ **then**
    /* if task is a primary task of cluster */
    if $\tau_i^j == cl_{pt}$ **then**
      $PE_i = getPEwithMinLPTasks(TMT, \tau_i^j, PE\_list)$// constrained PE search
    else
      /* other tasks of first group */
      $PE_i = getLowestUtil(TMT, getPENeighbours(PE(cl_{pt}), NH_{GT}))$// constrained PE search
    end
  else
    /* primary task of other groups */
    if $\tau_i^j == cl_{pt}$ **then**
      /* Attempt to map new clusters in unsed regions of the NoC */
      Get mapped PE of previous group's primary task : $PE(cl_{pt}(cl_{ix}-1))$
      $sPE\_N = getPENeighbours(PE(cl_{pt}(cl_{ix}-1)), NH_{GP})$
      Get PEs used by all and prev. other clusters (via TMT) : $\{all\_cl\_sPE, prev\_cl\_sPE\}$
      $tmp\_PE\_ranks = \{\}$ // track PE ranks
      foreach $pe_j \in sPE\_N$ **do**
        $ssPE\_N = getPENeighbours(pe_j, NH_{GT})$
        $unsed\_PEs\_all = ssPE\_N \setminus all\_cl\_sPE$ // set difference
        $unsed\_PEs\_prev = ssPE\_N \setminus prev\_cl\_sPE$ // set difference
        if $|unsed\_PEs\_all| > 0$ **then**
          $tmp\_PE\_ranks[pe_j] = |unsed\_PEs\_all|/|ssPE\_N|$
        else
          $tmp\_PE\_ranks[pe_j] = |unsed\_PEs\_prev|/|ssPE\_N|$
        end
      end
      $PE_i = getPEwithMinLPTasks(TMT, \tau_i^j, \max(tmp\_PE\_ranks))$// constrained PE search
    else
      /* other tasks of other groups */
      $PE_i = getLowestUtil(TMT, getPENeighbours(PE(cl_{pt}), NH_{GT}))$ // constrained PE search
    end
  end
  Map $\tau_i^j \rightarrow PE_i$; Update $TMT\{PE_i, \tau_i^j\}$
end

are taken into account. Route utilisation evaluation has to be performed for each $MMCP_i$. The structure of the algorithm is similar to the LU mapping algorithm, where each task $\tau_i \in J_i$ is iteratively assigned the least utilised MMCP. The utilisation change made by an MMCP assignment for one task is carried over into the next iteration, when selecting the MMCP for the next task in the job.

$$U(route) = \sum_{\forall links \in route\{src \rightarrow dst\}} \sum_{\forall Msg_i \in link} \left\lceil \frac{w.c.PL_{Msg_i}}{T_i} \right\rceil \tag{8.6}$$

- Shortest least utilised route selection **(MMCP-DistLU)**:

This heuristic, combines the above MMCP-Dist and MMCP-LU techniques to take into account both the distance of the MMCP and the utilisation of the route to the MMCP. Each $MMCP_i$ is weighted according Eq. (8.7) and the one with the minimum weight is selected. Load and distance metrics were also used by Awasti et al. [94] to assign threads to MMCs.

$$\text{MMCP-DistLU weighting}: w(MMCP_i) = norm(U(route)) + norm(|route|) \tag{8.7}$$

- Proportionate blocking-aware MMCP selection **(MMCP-LBP)**:

  In this scheme, the worst-case blocking of the flows are taken into account. Firstly, the number of higher priority flows already mapped to each MMCP, with respect to a target priority (priority of τ_0^0), is calculated. The overall mean number of high priority flows across all MMCPs is also calculated (referred to as $\mu(|hp\_flows|)$). A list of MMCPs (of length $|J_i^T|$) is then created; referred to as $prop\_MMCP\_list$. MMCPs that have a lower number of mapped high priority flows than $\mu(|hp\_flows|)$ are included twice as more in the list. Each task in J_i^T is then assigned sequentially to each MMCP in $prop\_MMCP\_list$. A proportionately scaled list is formed to avoid over-utilising a single MMCP.

- Fair distribution of tasks to MMCPs **(MMCP-Fair)**:

  In this heuristic, each MMCP will have a uniform number of tasks assigned to them. When a new job is received, the MMCP is sorted (increasing order) according to the number of tasks already allocated to them and the new tasks are assigned iteratively.

- Select MMCP different to parent task **(MMCP-InvPar)**:

  This scheme attempts to select MMCPs not already selected by its parent tasks. Tasks write back to the main memory as soon as they complete execution. If both parent and child tasks are mapped on the same PE or in close proximity and the same MMCP is selected for both parent and child, under high CCR conditions the MMCP selected and its respective links will be heavily congested by multiple MEM_WR flows. The objective is to reduce the impact of MEM_WR flow congestion on the same set of links, by diverting the child task MEM_WR flows to a different MMCP. The pseudo-code of MMCP-InvPar is shown in Algorithm 8.5. Tasks are iteratively assigned an MMCP and at each iteration the parents MMCPs are omitted from the selection range (lines 5-12). If the parents have already been assigned a higher number of MMCPs than N_T, then already used MMCPs are included when assigning MMCPs to the children. Therefore, if N_T and/or number of parent tasks are high, this scheme behaves similar to MMCP-Fair. However, unlike in MMCP-Fair, multiple children of the same parent can have a similar MMCP assignment.

- Random MMCP selection **(MMCP-Random)**: In this simple heuristic, an MMCP is selected at random for each tile task τ_i^j in the job.

Some of the MMCP selection schemes introduced above require keeping track of task-to-MMCP assignments of tasks already admitted and active in the system from previous jobs. This is mainly a requirement for the MMCP-LU, MMCP-DistLU and MMCP-LBP heuristics. It is assumed the RM keeps track of the MMCP assignment similar to the runtime mapping table, without requiring monitoring feedback from the application/platform. The RM combines the

MMCP assignment and the information within the runtime task mapping table (TMT) to derive a set of flows for a given network route.

Under certain workload conditions the MMCP-LU assignment can lead to uneven number of tasks assigned to each MMCP. For example, consider the scenario where a 31 frame job of a 2160p video arrives into the system and due to the nature of the MMCP-LU heuristic, it assigns 4 tasks to the first 7 MMCPs and only 3 tasks to the 8th MMCP. Next, a job of a low resolution 288p, 16 frames per job, video arrives into the system. Due to the large difference in the route utilisation induced by high/low resolution videos, the 8th MMCP will now get a large number of the low-res video's tasks assigned to it.

Algorithm 8.5: MMCP-InvPar main memory controller port selection algorithm pseudo-code

Input : J_i^T : topologically sorted tile-level task graph of i_{th} job,
N_T : number of tiles per frame TMT : the runtime task mapping table,
$MMCP\_clist$: list of MMCPs in the platform as circular list,

Output: New tile task to MMCP mapping $(\tau_i^j \rightarrow MMCP_i) : MMCPT$

```
/* Iteratively, map tiles to MMCPs, but try to avoid same MMCP as parents                    */
foreach τ_i^j ∈ J_i^T do
    /* first task : (i.e. I_j^0) of the job - root node                                       */
    if τ_i^j == τ_0^j then
        MMCP_i = MMCP_clist.next()
    else
        /* if first tile of a task                                                           */
        if τ_i^j == τ_i^0 then
            s_pMMCP : get all MMCPs assigned to parent tasks of τ_i^j
            s_uMMCP = MMCP_clist \ s_pMMCP // find MMCPs unused by parents (set diff.)
            if |s_uMMCP| < N_T then
                Extend s_uMMCP by obtaining more MMCPs from MMCP_clist
            end
            MMCP_i = s_uMMCP.next()
        else
            MMCP_i = s_uMMCP.next()
        end
    end
    Map τ_i^j → MMCP_i ; Update MMCPT{MMCP_i, τ_i^j}
end
```

8.5 Evaluation

This evaluation section conducts two primary experiments as follows:

- Experiment A ***(ExpA)***: Evaluates the predictability, performance and overhead of the proposed cluster-based tile mapping techniques (including their variants) against non-clustered and existing task mapping baseline mappers. The behaviour of the tile mappers are evaluated under different workload CCR levels.

  Two hypotheses are being tested in this experiment: (a) the proposed CL mapper can offer lower communication costs than the baseline mappers; (b) the proposed CL-BG mapper can offer comparable or lower maximum job lateness levels but with lower communication cost than the baseline mappers, thereby balancing predictability and communication energy savings.

- Experiment A ***(ExpB)***: Evaluates the predictability, performance and overhead of the different MMCP selection heuristics discussed in the previous section, combined with the primary clustered/non-clustered tile mappers.

  The primary hypothesis being tested in this experiment is that the MMCP-InvPar selection scheme can reduce the maximum job lateness levels compared to the other baseline MMCP selection schemes.

8.5.1 Experimental design

The application and platform models described in Section 8.1 are used for all experiments in this section. All experiments assume an 8×8 NoC platform. The workload is generated using the parameters and analysis derived from Chapter 7. Therefore, it can be assumed that the PEs have an operating frequency of 3GHz (corresponding to the HEVC decoder platform in Section 7.3.1), with necessary scale adjustments made to compensate for DRAM latencies (as explained in Section 7.8.3). Message flow header routing cost is assumed to be 7 clock-cycles, NoC frequency is set at 100MHz and the link width is set to 16 bytes. As explained in Section 3.3, a lower NoC frequency (i.e. low bandwidth) is assumed, in order to induce a reasonable amount of network utilisation/congestion. A platform with 4 memory controllers (2 ports each) on either side of the NoC was considered for all experiments in this evaluation. In ExpA, the task-to-MMCP selection scheme is fixed as MMCP-Dist for each mapping type.

The workload for the experiments are 24 parallel video streams ($WL = 6.13 \times 10^7$) with varying video resolutions (low-res 288p to UHD 2160p). Each video in the workload has 5 GoPs, 16 or 31 frames per GoP (randomly selected) and different video type (randomly selected out of those shown in Figure 8.3). In the previous chapters all video streams had a fixed frame rate of 25fps and 12 frames per GoP, which effectively gave a job end-to-end deadline of $D_{e2e} = 0.48s$. However, to reflect modern video streaming workloads, low/med resolution videos (e.g. 288p-720p) were randomly assigned a frame rate of either 30fps or 60fps. Videos with very high resolution (e.g. 1080p-2160p) were assigned a fixed frame rate of 60fps. Hence, different video streams will have a variable D_{e2e} with respect to its frame rate and number of frames per GoP. Simulations were carried out for 30 unique seeds for every experiment, which results in varying video stream characteristics such as arrival patterns, task computation costs, reference data payloads, job dependency structures etc. The proposed tile mappers are compared against the LU and PP mappers, both introduced in Section 5.6.1.1 and a random mapper (termed RND) which assigns tasks to PEs randomly. Table 8.3 summarises the experimental conditions, parameters and metrics used to explore the aforementioned evaluation objectives.

For the ExpA experiment the tile mappers are evaluated under two workload CCR levels: a *normal workload CCR* level as shown in Figure 8.3 and a *higher workload CCR* level. To induce a high workload CCR condition the computation cost of all frames have been scaled down by a factor 0.1 and the communication cost is unaltered. This effectively scales up the job CCR ranges shown in Figure 8.3. ExpB is only evaluated under the high workload CCR condition. Hereafter, the normal workload CCR profile will be denoted as CCR_NORMAL and the higher workload CCR profile will be denoted as CCR_HIGH.

The primary metrics measured in both experiments are the job lateness, cumulative basic latency of all flows (commonly referred to as communication cost) and mapping execution overhead. The job lateness primary metric can be further broken down into the data/memory flow response time and the task turn-around time, in order to give more insight into the causes of

Table 8.3: Summary of experimental evaluation design parameters

Eval. obj.	Workload configuration	Independent variables	Response variables
ExpA	Normal workload CCR High workload CCR (comp. cost $\times 0.1$)	*Mapping techniques:* CL, CL-F, CL-NoCCR, CL-IPC, CL-BG, CL-BG-F, PP, LU, LU-F, RND	*Primary:* Job lateness, communication cost (cumulative flow basic latency), mapping exec. overhead, data/memory flow response time and task turn-around time. *Secondary:* Communication cost breakdown (number of flows, route lengths and payload sizes) and NoC PE busy time distribution
ExpB	High workload CCR (comp. cost $\times 0.1$)	*Mapping techniques:* CL, CL-BG, LU, PP *MMCP selection schemes:* MMCP-Dist, MMCP-LU, MMCP-DistLU, MMCP-LBP, MMCP-Fair, MMCP-InvPar, MMCP-Random	

lateness. Recall from Section 3.2.1 that the task turn-around time does not include the flow response-time but includes the blocking time due to interference from higher priority tasks. Lower variability and mean/maximum job lateness values are desirable in terms of predictability (Section 3.2.1) whilst lower communication costs directly reduce NoC energy consumption (Section 3.2.3). The secondary metrics include the NoC PE busy time distribution and a breakdown of the communication cost (number of flows, total payload and flow route length). As explained in Section 3.2.3, a higher mean PE busy time variation in the NoC cores is desirable, in order to facilitate dynamic PE power saving techniques.

Note that the hop parameters of the cluster based mappers and their variants (i.e. CL, CL-BG etc.) have not been changed for the different CCR specific workload profiles. However, the $CCR(J_i)$ ranges (Table 8.1, Table 8.2) which are used to select the fixed hop parameters for CL and CL-BG, have been scaled up by 10 to proportionally match the reduction of the task computation cost in CCR_HIGH.

8.5.2 Results discussion

8.5.3 ExpA - Investigating cluster mappers under different workload CCRs

The results of ExpA under the CCR_NORMAL workload scenario is shown in Figure 8.8. The top row displays the primary metrics (job lateness, communication cost and mapping execution overhead) and the bottom row displays the secondary metrics - the data/memory flow response time and task turn-around time. Due to the large magnitude of the sample size of the secondary metrics (e.g. 4.8×10^6 total number of flows per mapping type for all seeds), only the mean, minimum (Min.) and maximum (Max.) of each seeded simulation run are shown in plots. A similar set of plots are shown in Figure 8.9 for the CCR_HIGH workload scenario. First a discussion of the mapping results under CCR_NORMAL is presented followed by a discussion of the CCR_HIGH results.

8.5.3.1 Primary metric evaluation

All evaluated mappers have a high amount of outliers in the job lateness distributions, representing jobs that encountered severe blocking and delay by higher priority jobs. All mappers have comparable mean and inter-quartile (IQR) job lateness levels but varying outlier lateness

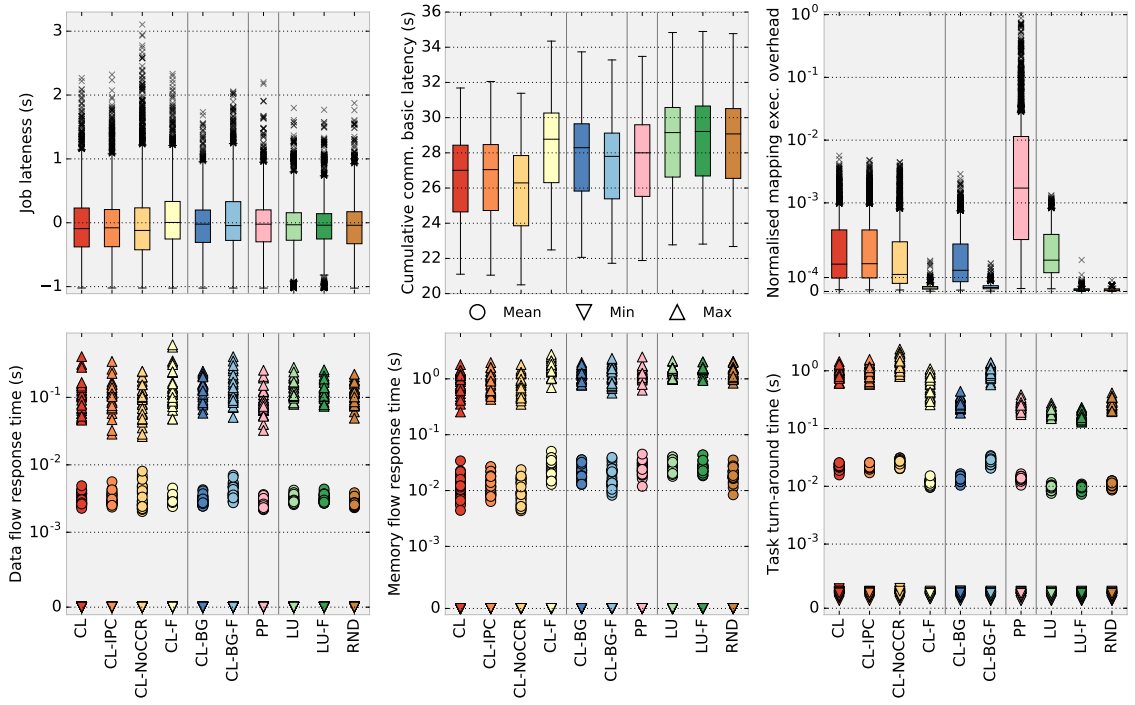


Figure 8.8: Evaluation results of the tile mapping schemes - normal workload CCR (CCR_NORMAL). *Top row:* job lateness, communication cost (cumulative basic latency), normalised mapping execution overhead. *Bottom row:* data flow response time, memory flow response time, task turn-around time. Bottom row results indicate mean/max/min of each seeded simulation run. Note that the top-right and bottom-row plots use a logarithmic scale in their y-axis.

levels. Hence, the job lateness of the mappers can be primarily evaluated based on their outliers (maximum job lateness) and the variability of the job lateness distributions.

Discussion of CL, CL-BG and variants:

Under the CCR_NORMAL workload scenario, the CL mapper and its variants (CL-IPC and CL-NoCCR) show a higher maximum job lateness than the other evaluated techniques; however, they have a clear advantage in communication cost reduction ($\approx 3s$ lower cumulative communication basic latency reduction). Both CL and CL-IPC show similar results in terms of lateness and communication cost but CL-IPC has a slightly lower mapping execution overhead. CL-NoCCR has the worst job lateness, which indicates that using a dynamic CCR-aware hop parameter technique (as in CL), is more beneficial than having fixed hop parameters. CL-F shows a similar maximum job lateness level to CL but has slightly increased mean job lateness and higher communication cost. Note however that CL-F has 1-2 orders of magnitude lower mapping overhead than the other mappers due to its inaccurate but fast mapping technique.

The proposed CL-BG mapper (and variants) shows a lower job lateness than CL but at the cost of higher communication cost. Compared to the LU, LU-F and RND mappers, CL-BG has comparable maximum job lateness but lower communication cost. Unlike CL, the CL-BG technique has fewer parameters, therefore it can be easily tuned (as shown in Appendix D.2) to balance communication cost and lateness reduction or to optimise a single metric depending on the system requirements. For example, the parameters $NH_{GT} = \{2,4,6\}$ for the low/med/high $CCR(J_i)$ ranges can give low job lateness results but at the cost of increased communication cost. The chosen parameters of CL-BG (Table 8.2) gave the best trade-off for both workload CCR levels. The CL-BG-F variant shows higher lateness than CL-BG but has an order of magnitude lower mapping overhead. CL-BG has lower mapping overhead than CL; on the evaluation

platform (Python 2.7 based simulator executed on a 2.8GHz AMD Opteron 6386 CPU, 256GB RAM platform) the CL-BG mapper has a mean absolute mapping execution overhead of 0.07s and maximum overhead of 0.8s.

Discussion of baseline mapper results:

The LU mapper distributes the tasks evenly across the PEs, hence it incurs higher communication cost. LU incurs lower task contention therefore leading to lower overall job lateness levels than the cluster-based mappers under CCR_NORMAL workload conditions. The LU mapper has a lower mapping overhead than the clustering-based mappers due to its simple heuristic and the fast variant (LU-F) has lower mapping overhead to LU.

In large complex workload scenarios, the RND mappers mapping quality is similar to LU (i.e. sparse distribution of tasks), therefore shows similar lateness and communication cost results. Due to its simplicity, the RND mapper has the lowest mapping execution overhead. The PP tile mapper shows comparable job lateness and communication cost to CL-BG but its mapping execution overhead is 2-3 orders of magnitude higher than the other mappers. This is primarily due to the computationally intensive task-graph operations carried out by the PP algorithm (Algorithm 2.1) such as searching for nodes and edges with max/min weights and node merging. Note the high variability in execution overhead of the PP mapper, which relates to the variation in the number of nodes and edges in the TG. For very high number of nodes or edges (e.g. in the order of 100 or 1000) the PP mapper performs rather poorly.

Breakdown of the job lateness:

The bottom row of Figure 8.8 gives an indication of the timing bottlenecks of the mapping techniques. For all mappers, under the CCR_NORMAL workload scenario, the memory flow response times are much higher than the data flow response times due to the larger memory write traffic payloads. Memory traffic also typically encounters higher amounts of contention because only 8 MMCPs are available in the platform. Note than in ExpA, all mappers use the same MMCP-Dist memory controller port selection scheme, hence all mappers show a similar maximum memory flow response time. The overall task turn-around times are also higher than the data flow response times. This is especially true in the case of the CL type mappers, where the results indicate the task contention is the bottleneck for the CL mappers due to tight clustering. LU-F has the lowest and CL-NoCCR has the highest task turn-around time.

Discussion of the high CCR workload results:

Figure 8.9 shows the results of the tile mappers under the CCR_HIGH workload condition. In this scenario, as the computation costs are scaled down by an order of magnitude, the flow response time (especially the memory traffic) is clearly a dominant factor for all mappers. Memory flow response-time are about 4-5 times higher than data flow response-times and more than an order of magnitude higher than task turn-around times. This means, a slight increase in the mean/maximum memory latencies can severely impact the job lateness.

The results show that unlike in CCR_NORMAL, under the CCR_HIGH condition the CL-NoCCR mapper variant has the lowest maximum job lateness as well as the lowest communication cost due to its tight clustering. Note that its task turn-around time is higher than the others, but this does not impact the overall job lateness significantly, as the mean memory response time is lower. The job lateness gap between the CL and LU mappers have also decreased. When CCR is increased the job lateness of LU, LU-F and RND mappers increase whilst CL's job lateness

reduces. When the workload CCR is higher, the CL-BG mapper has a lower job lateness distribution (IQR and maximum) than the LU mapper. The CL-F and CL-BG-F mapper variants show a much worse lateness levels than in CCR_NORMAL mainly due to their higher maximum memory flow response times. PP has a lower IQR job lateness than CL-BG but as in CCR_NORMAL, its mapping execution overhead is still much larger.

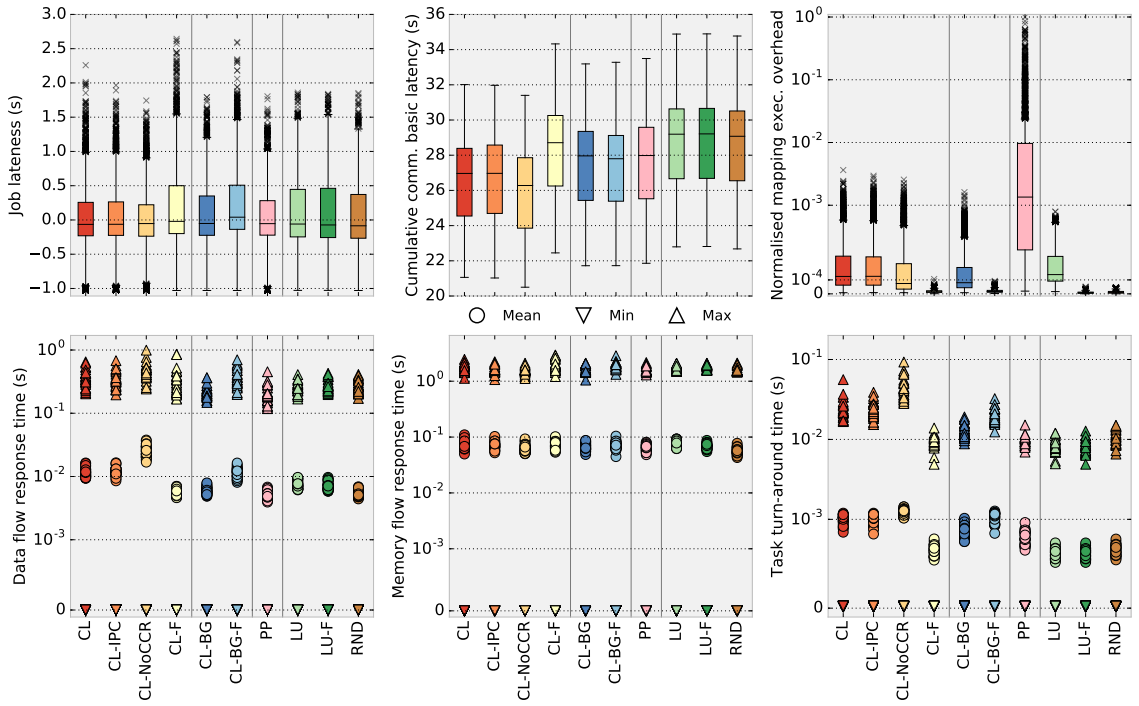


Figure 8.9: Evaluation results of the tile mapping schemes - high workload CCR (CCR_HIGH). *Top row:* job lateness, communication cost (cumulative basic latency), normalised mapping execution overhead. *Bottom row:* data flow response time, memory flow response time, task turn-around time. Bottom row results indicate mean/max/min of each seeded simulation run. Note that the top-right and bottom-row plots use a logarithmic scale in their y-axis.

8.5.3.2 Secondary metric evaluation

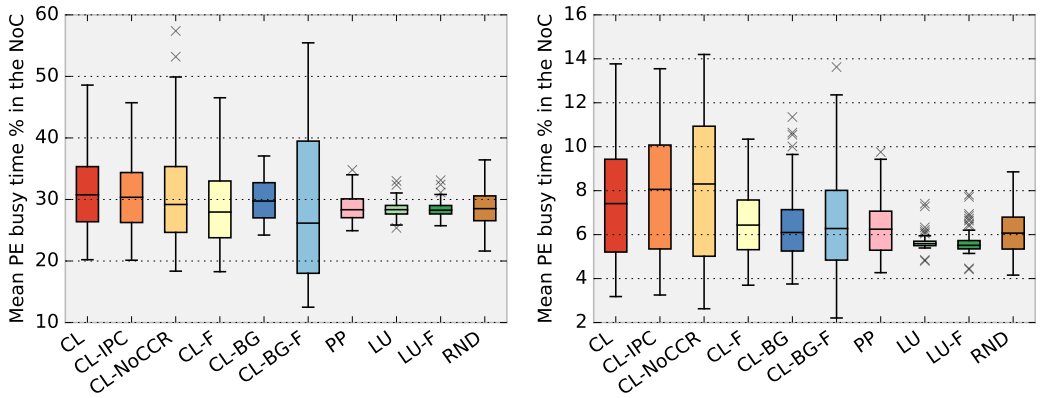
The communication cost breakdown for the different mapping techniques are presented in Table 8.4. The results for both CCR_NORMAL and CCR_HIGH conditions are similar hence, only the CCR_NORMAL results are shown. Higher number of flows and route lengths are the causes of higher communication cost in LU, LU-F and RND compared with the cluster-based tile mapping techniques. CL-BG has slightly lower mean route length than CL but injects higher number of flows into the NoC. PP shows a high number of flows and total payload compared with CL but has the lowest mean route length out of all the evaluated mappers. RND has the highest mean route length but lower number of total flows and total payload than LU due to arbitrary grouping of tasks.

The variation of the mean PE busy time across the different PEs in the NoC are shown in Figure 8.10. In the distributions, each data point represents a PE's mean busy time percentage. Essentially these plots represent the distribution of the load across the PEs. A large variation is seen in the cluster-based mappers due to the nature of the task dispersion. CL (and variants) have a larger workload distribution spread than CL-BG or PP out of the cluster mappers. The variation is higher in the CCR_NORMAL condition ($\approx$ 30% min-max variation seen in CL) than

Table 8.4: Communication cost breakdown for all tile mappers - CCR_NORMAL workload scenario

Mapping type	Total payload (GB)	Mean route length	Mean # flows $\times 10^4$	Total # flows $\times 10^5$
CL	1187	5.42	13.46	40.37
CL-IPC	1191	5.45	13.51	40.54
CL-NoCCR	1159	5.34	13.10	39.30
CL-F	1267	5.47	14.98	44.93
CL-BG	1246	5.20	14.47	43.42
CL-BG-F	1227	4.83	14.21	42.62
PP	1237	4.70	14.36	43.07
LU	1284	6.66	15.26	45.78
LU-FFI	1286	6.67	15.32	45.96
RND	1280	6.72	15.21	45.63

in the CCR_HIGH condition. CL-BG-F has the highest variation and LU has the lowest variation in the CCR_NORMAL condition. A similar trend can be seen in the CCR_HIGH scenario although here, the CL-NoCCR mapper has the largest variation. Overall, a lower PE busy time can be seen in the LU/LU-F non-clustered tile mapping schemes, as more time is spent on NoC communication than the cluster-based mappers. As LU and LU-F evenly spreads the workload across the NoC its PE busy time variation is smaller than RND.



(a) CCR_NORMAL workload condition **(b)** CCR_HIGH workload condition

Figure 8.10: Distribution of PE busy times for different mapping types. Each data point in each distribution represents the mean busy time for a PE in the NoC

8.5.4 ExpB - Investigating different MMCP selection schemes

The primary metric results of experiment ExpB is shown in Figure 8.11. The x-axis of each sub-plot is first categorised into the different mapping types (CL, CL-BG, PP and LU) and within each mapping category, the results of using different MMCP selection schemes are presented. In the ExpB experiment there are two treatments - the task mapping technique and the MMCP selection scheme. The presentation of the results in Figure 8.11, are similar to Figure 8.8 and Figure 8.9; the top row shows the job lateness, the cumulative basic latency (communication cost) and the normalized mapping execution overhead, the bottom row shows the data/memory flow response time and the task turn-around time. Note that the y-axis scales for each sub-plot are different. There are negligible differences in communication cost by using different MMCP selection schemes, as the amount of total memory traffic flows and total payload do not change;

only the mean communication route length will vary slightly. Similarly, the task turn-around times also does not vary significantly based on the MMCP selection type. Hence, this results discussion will mainly focus on the other 4 primary metrics.

The MMCP-LU and MMCP-Dist schemes show the highest amount of job lateness distribution out of the evaluated MMCP selection techniques. MMCP-LU and MMCP-Dist schemes produce highly uneven assignment of tasks to MMCPs as explained in Section 8.4. This leads to higher memory traffic contention for certain MMCPs in the NoC leading to larger memory flow response times and in turn higher overall job lateness results. Therefore, under every mapping type, using the MMCP-LU heuristic can result in a high maximum lateness level. A higher job lateness IQR variability is seen in the MMCP-LU and MMCP-Dist schemes when compared with the other MMCP selection techniques. The combination of the MMCP-Dist and MMCP-LU schemes, termed MMCP-DistLU, shows a relatively lower lateness distribution especially when used in conjunction with the CL, CL-BG and PP mappers.

The job lateness of the MMCP-LBP varies based on the mapping type; it has slightly lower maximum job lateness in CL, than compared to MMCP-Rand but performs poorly under the LU mapper. The MMCP-Rand heuristic show results comparable to MMCP-LBP under the CL-BG and PP mappers but has slightly higher maximum lateness under the CL mapper. MMCP-Fair and MMCP-InvPar heuristics show the lowest job lateness out of all the evaluated MMCP selection techniques. Using the MMCP-InvPar heuristic, the mean job lateness of the CL mapper can be brought down by 0.21s compared with the default MMCP-Dist heuristic, which is ≈20% of a job's D_{e2e} (assuming 31 frames per GoP and 30fps) and the maximum job lateness can be reduced by 1.5s. The lateness reductions are due to the significant decrease in the memory flow response times as shown in Figure 8.8 (bottom-right). When compared with the MMCP-Dist heuristic, the MMCP-InvPar and MMCP-Fair have higher mean/max data flow response times; however, these increases do not impact the job lateness as the memory traffic response times are about an order of magnitude higher than the data traffic response times. MMCP-InvPar shows slightly lower number of total late jobs than MMCP-Fair under the CL-BG, PP and LU mappers (Table 8.5), even though its maximum job lateness is comparable.

All MMCP selection schemes adds further overhead into the runtime resource allocation procedure in the system. The mapping execution time results of the different MMCP selection schemes are shown in Figure 8.8(top-right). These measurements include the overhead incurred by both the mapping technique and the MMCP selection heuristic. The MMCP-LU and MMCP-DistLU techniques have about 2-orders of magnitude higher execution overhead than the other heuristics, as the utilisation change made by an MMCP assignment for one task is taken into account when selecting a MMCP for the next task. The MMCP-InvPar scheme has a slightly lower maximum and mean mapping execution overhead than the MMCP-Fair scheme under all mapping types. Note that the MMCP selection schemes under the PP mapper has large overheads primarily due to the overhead of the PP mapper itself.

Table 8.5: Number of late jobs per MMCP selection scheme (columns) and mapping type (rows)

	MMCP-Dist	MMCP-LU	MMCP-DistLU	MMCP-LBP	MMCP-Rand	MMCP-Fair	MMCP-InvPar
CL	1415	1311	1015	792	795	615	620
CL-BG	1491	1485	1016	719	683	507	503
PP	1495	1436	985	647	639	460	455
LU	1518	1398	1086	1416	768	517	515

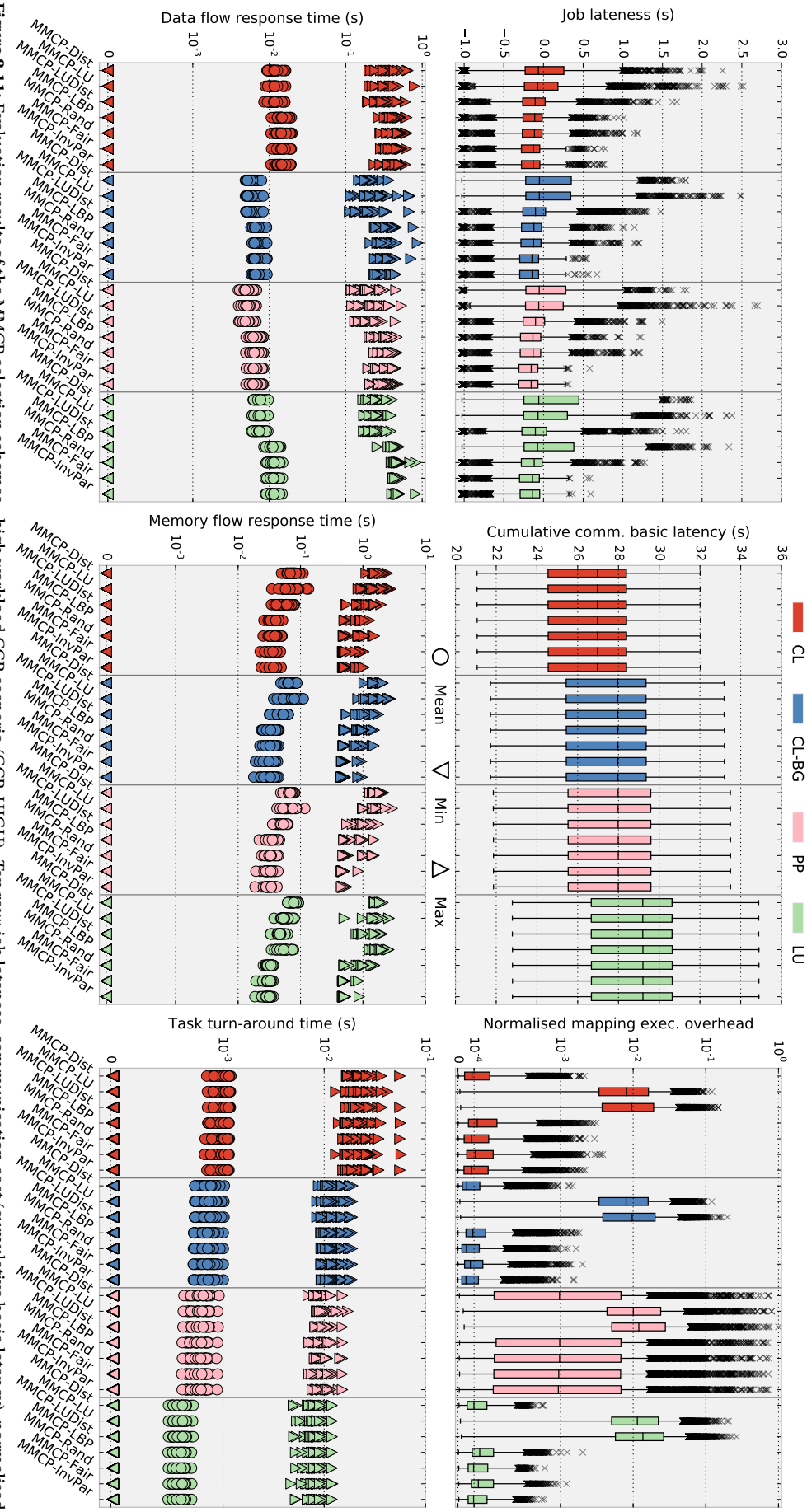


Figure 8.11: Evaluation results of the MMCP selection schemes - high workload CCR scenario (CCR_HIGH). *Top row:* job lateness, communication cost (cumulative basic latency), normalised mapping execution overhead. *Bottom row:* data flow response time, memory flow response time, task turn-around time. Bottom row results indicate mean/max/min of each seeded simulation run. Note that the top-right and bottom-row plots use a logarithmic scale in their y-axis.

8.5.5 Results summary

The results from *ExpA* confirms the hypothesis that the proposed tile clustering-based mappers, can be used to lower the communication cost of HEVC tile decoding at the expense of reasonable job lateness degradation (in the case of CL) or at the same job lateness level (in the case of CL-BG).

The proposed CL mapper and its variants showed significantly lower communications costs over the other evaluated mapping types. CL can be used to obtain an average communication saving of $\approx 10\%$ over a uniform or random workload distribution method (LU and RND). However, this saving comes at the cost of $\approx 0.2s - 0.5s$ higher maximum job lateness increase compared with the LU mapper. For certain SRT video streaming applications, where buffering can be used, this lateness may be acceptable. The CL mapper also has an advantage over the others in terms of PE busy time distribution. Its uneven distribution of workload can be exploited to facilitate further power saving techniques such as putting idle/unused PEs to low-power mode.

The proposed CL-BG mapper can be used to obtain slightly lower/comparable job lateness to LU and RND but with $\approx 4-6\%$ lower average communication cost. Most importantly, unlike the other evaluated techniques, the CL-BG mapper shows a good *balance* between job lateness (i.e. predictability improvement) and communication cost reduction (i.e. NoC energy cost reduction) under both normal and higher CCR workload conditions. Note that both CL and CL-BG have parameters that can be re-adjusted to improve one metric over the other.

The communication cost saving of the proposed mappers are mainly due to the low number of flows injected into the NoC, as well as short message flow route lengths. Uniform (LU) and random (RND) mapping can lead to long message routes and higher number of flows due to inefficient use of the network. The PP mapper clusters tasks together to efficiently balance computation and communication and it shows slightly lower mean/max job lateness levels to CL-BG and LU under the CCR_HIGH workload condition but its mapping execution overhead is 2-3 orders of magnitude higher than the other mappers, which makes it unsuitable for runtime mapping of large and complex task-sets.

The evaluation results from *ExpB*, partially confirms the experimental hypothesis stated earlier. The MMCP-InvPar MMCP selection heuristic can reduce the maximum job lateness, lower than all baseline except the MMCP-Fair heuristic. However, MMCP-InvPar has slightly lower number of late jobs compared with MMCP-Fair when combined with the CL-BG, PP and LU mappers. MMCP-InvPar heuristic can provide a maximum job lateness that is $\approx 75\%$ lower than the default MMCP-Dist heuristic and $\approx 25-50\%$ lower compared to MMCP-DistLU and a simple random assignment (MMCP-Rand). The maximum overhead of the MMCP-InvPar technique is also lower than MMCP-Fair and comparable to MMCP-Dist. The experiments also showed that a non-uniform distribution of number of tasks to MMCPs (as seen in MMCP-Dist or MMCP-LU) was not efficient as MMCP contention can be uneven, leading to larger memory flow response-times. The results also showed that in certain MMCP selections techniques may have varying lateness results depending on the type of task mapping profile used (e.g. in the case of MMCP-LBP).

8.6 Summary and novel contributions

To summarise, this chapter presented the following contributions to existing work on tile-parallel HEVC video decoding on many-core platforms:

- Tile parallel HEVC video *encoding* on NoC-based multi-core platforms have been explored in the past by [187] and *decoding* has been explored in [25, 26]. However, they assume a thread-based programming model which uses shared memory communication, which is known to have performance bottlenecks as the platform size and number of threads increase. To the best of the author's knowledge, this is the first work to explore, open-loop, tile-parallel, multiple HEVC video stream decoding on a distributed memory model (commonly seen in NoC based many-cores), where inter-task communication is performed directly via the on-chip interconnect.

- This work introduced several application specific challenges in tile-level parallel HEVC video decoding. Dynamic GoP structures with complex dependency patterns, task graph scale increase due to tile-partitioned parallelisation and varying video stream CCRs due to different video types were explored. This work also presented the problem of balancing two conflicting mapping objectives - lateness and NoC communication cost reduction.

- Two runtime HEVC tile mapping techniques were presented. The first proposed tile mapper (CL) is a generic technique which takes into account the job's longest path and combines it with a blocking and CCR-aware clustering heuristic. The second proposed mapper (CL-BG) is a stream specific task clustering technique targeted at streams encoded using hierarchical B-frame structures. These two mappers are used to improve the job lateness reduction (i.e. predictability) and reduce the NoC communication cost (i.e. communication energy) in the platform whilst having a low mapping overhead. Faster, variants of these mappers were also introduced, although they offer less predictability.

- Several main memory control port selection heuristics were presented and their predictability, performance and overhead evaluated. The proposed MMCP-InvPar heuristic attempts to reduce memory contention by trying to assign tasks different MMCPs than their parent tasks.

The proposed mappers and MMCP selection techniques were evaluated in terms predictability (job lateness), NoC energy savings (communication cost) and overheads (mapping execution overhead). The job lateness and communication cost metrics were further broken down to analyse the bottlenecks in finer granularity. Evaluations of the proposed tile mappers were carried out in both normal and high CCR workload conditions.

The CL mapper is mainly aimed at reducing the NoC communication cost whilst maintaining reasonable decoding lateness. The predictability of the CL-BG mapper is slightly better than the CL mapper in both CCR conditions evaluated and its communication cost saving is higher than a uniform utilisation-based runtime mapper. Both proposed mappers have parameters that can be tuned to control the level of task clustering, in order to trade-off or balance predictability and communication cost in the platform. CL-BG has fewer parameters than CL, making it simpler to customise. However, unlike CL, CL-BG is constrained to video streams encoded using hierarchical B-frames which is a common but optional encoder setting. CL-BG illustrates that task clustering can be customised with some application domain knowledge in order to balance the level of task/flow contention and the two conflicting metrics. The evaluation also showed that existing dynamic task mappers such as PP can be very computationally inefficient when handling large and complex workloads. The LU mapper can be used when saving NoC energy is not a concern and workloads have a low CCR.

Section 8.2.1 and Section 8.2.4 discusses in detail the strengths and weaknesses of task clustering and the effect it has on contention, communicating cost saving and lateness reduction. With limited knowledge of certain application characteristics CL-BG can be efficiently used to balance the level of task/flow contention and the two conflicting metrics. The parameters of the proposed techniques can also be adjusted based on the job CCR level which is necessary when the video stream CCR can change dynamically due to the level of motion in the video.

Lastly, the memory contention bottlenecks identified during preliminary evaluation of the proposed mappers were addressed, by exploring several MMCP selection heuristics. Experimental results showed that a task distance/locality-based heuristic can cause high levels of memory contention and the benefits that a good mapper can bring are limited by inefficient MMCP selection schemes. Instead, a uniform MMCP to task assignment scheme with parent/child task MMCP selection awareness can help to significantly reduce memory traffic contention and improve predictability over a distance-based or utilisation-based selection scheme.

Chapter 9

Conclusions and future work

Large-scale, complex, video decoding on many-core systems brings upon many new challenges to the runtime resource allocation problem. This thesis proposed possible approaches that can be used in the context of admission control, resource management organisation and resource allocation for hard and soft real-time multiple video decoding on NoC based many-cores. This conclusion chapter contains a summary of the key findings and contributions made in this thesis. Several potential future research ideas to extend this work are also presented.

9.1 Review of contributions in this thesis

The work presented in this thesis was based upon the following two thesis hypotheses made in Chapter 1:

- *Application and blocking-aware, runtime mapping heuristics combined with a deterministic admission controller can be used to guarantee timing requirements and improve system utilisation for HRT video streams encoded using classical video codecs; a low-overhead, distributed, remapping technique, can be further used to reduce the lateness, of SRT video streams.*
- *Application-specific task clustering and mapping combined with better memory controller selection heuristics, can be used to balance communication cost and lateness reduction of SRT decoding of complex video streams, encoded using modern video codecs*

The above hypotheses have been confirmed by a series of experiments carried out in Chapters 4-8 in this thesis. The primary theme in this research work has been to improve predictability and performance in the system when only limited knowledge of the workload is available at runtime. To this extent, several novel dynamic task allocation and resource management policies have been proposed and evaluated against existing techniques under varying workload conditions and platform sizes. The following sub-sections will summarise the contents and main contributions made in this thesis, along with a critical assessment of each proposed technique.

9.1.1 A data-parallel multiple video stream decoding application model

A gap exists between the multicore embedded multimedia systems design research community and the parallel video coding research community. The former often adopts a generic *task-parallel* application model with direct core-to-core communication via message passing. The latter almost always considers a multi-threading based *data-parallel* model using shared memory communication.

Chapter 3 introduced a DAG based application model for frame parallel stream decoding on NoC-based distributed memory architectures. The proposed application model follows a generic hierarchical structure, sufficient to describe decoding of time-varying number of parallel streams and reference frame data dependencies transmitted between cores, via a NoC interconnect. This type of model enables system designers to explore existing multicore scheduling and resource allocation techniques on data-parallel video decoding workloads, thus helping to reduce the gap between the two above mentioned research communities. Several of the constraints and assumptions made in the model are subsequently lifted in Chapters 5,7 and 8.

Summary of novelty: A hierarchical, data-parallel multi-stream video decoding application model, that is targeted towards modern communication-centric NoC-based many-core platforms.

9.1.2 Deterministic and heuristic-based admission control strategies

In Chapter 4, several admission control techniques were presented which can be used in the context of real-time video decoding systems. A hard real-time deterministic admission controller (D-AC) was introduced which performs end-to-end video stream schedulability tests, to determine if any stream deadlines will be missed by admitting a new stream. A stream-level runtime mapping procedure and certain constraints to the stream encoding enables the D-AC to perform worst-case response time analysis on the video streams. The platform model assumes priority-preemptive task and message flow scheduling and arbitration.

The second admission controller relies on a closed-loop, heuristic-based strategy, to determine the admission decision, and hence aimed at soft real-time time workloads. Here, the lateness metric of the tasks within the system buffers/task queues, is used within the closed-loop admission control heuristic (Heu-AC). Individual subtask deadlines were estimated as a ratio of the end-to-end job deadline.

Evaluation results showed that the D-AC provides a high level of predictability with none of the admitted video streams being late. The drawback in the D-AC approach is that the conservative nature of the schedulability analysis enforces strict admission control, thereby significantly under-utilising the system resources. This issue is subsequently addressed by the work in Chapter 5. The Heu-AC approach on the other hand has parameters that can be tuned to achieve a reasonable balance between predictability and utilisation, which makes it suitable for soft-real time video decoding. However, the Heu-AC relies on a closed-loop system, where the admission controller queries the status of the PE task queues to make an admission decision.

Summary of novelty and new insights: Two novel admission control strategies are proposed. Firstly, a deterministic admission controller that uses end-to-end schedulability analysis to provide hard real-time timing guarantees to admitted multi-stream video decoding workloads. Secondly, a heuristic-based admission controller that uses video stream lateness as a tunable decision metric, that can be used to serve soft real-time video streams to balance predictability and utilisation. Experimental evaluation provided a new insight that, task and message flow contention should be minimised or appropriately taken into account in the D-AC analysis in order to improve the admission rates.

9.1.3 Runtime task mapping for hard real-time video stream decoding

The distance of communicating tasks and the level of contention in the system are primary factors that can affect the worst-case response time of tasks. These properties can vary depending on the task to PE mapping configuration. Therefore, certain task allocations can lead to increased end-to-end job worst-case response times, resulting in the D-AC to reject a higher amount of video streams, over other more efficient task placements. To this extent, Chapter 5 presented two dynamic task mapping techniques, to efficiently map the video decoding tasks on the NoC, in order to improve the admission rate and utilisation of the D-AC.

The first proposed technique (LWCRS) can be used to map generic DAG based applications with fixed task priorities. The worst-case remaining slack metric is used to decide on which PE to map each task in a job. LWCRS attempts to ensure balanced-blocking, such that tasks are temporally packed tightly as long as their deadlines are not missed. The second technique (IPC) is application-specific and uses known properties of the GoP dependency structure to make mapping decisions. IPC groups the I and P frame decoding tasks together thus reducing the contention and response time of the overall job. Both techniques are facilitated by an open-loop resource manager which does not incur system monitoring overhead unlike existing techniques in the literature.

The proposed heuristics were compared against several dynamic and static baseline mapping techniques existing in the literature. The proposed techniques showed higher utilisation and D-AC admission rates than the baseline runtime mappers. This shows that taking into account the blocking, the inter-task communication distance and limited knowledge of the application characteristics can help to improve the quality of the mapping. The experiments also showed that certain mappers may behave differently based on the communication to computation ratio (CCR) of the workloads evaluated.

Summary of novelty and new insights: Two novel dynamic task mapping techniques are proposed. The first technique (i.e. LWCRS) takes into account both the blocking incurred by target task and existing tasks in the system. A second runtime mapper (i.e. IPC) introduces minimal application-knowledge into the mapping heuristic to further reduce network contention. Both approaches, can be used in conjunction with a deterministic admission-controller to improve admission rates; evaluations showed improvement over existing runtime mappers. Furthermore, certain experimental results showed that the workload communication-to-computation ratio should be considered in the mapping heuristic to account for diverse workloads

9.1.4 Bio-inspired, distributed task remapping for NoCs

Centralised resource management techniques suffer from scalability and reliability issues (Section 2.3.2.1). The state-of-the-art cluster-based/hierarchical resource management techniques can incur high communication protocol cost (Section 2.3.2.2). The few fully distributed NoC management techniques existing in the literature have complex protocols, rely on custom hardware or rely on code duplication (Section 2.3.2.3). Thus, in Chapter 6 the possibility of using a bio-inspired, distributed task remapping technique (termed PSRM) for NoCs is investigated. The proposed remapping technique attempts to reduce the job lateness, whilst maintaining low resource management communication overhead. Each PE autonomously makes remapping decisions using a balanced-blocking heuristic and an existing swarm-inspired, load-balancing protocol. Chapter 6 also introduces several significant adaptations to an existing cluster-based

resource management approach.

PSRM is evaluated against the adapted cluster-based remapper as well as centralised and random remapping baselines. Due to lack of global view of network traffic at runtime, certain reallocations can cause positive or negative job lateness improvement. Overall, PSRM showed marginally better predictability than the baselines but with a much lower communication overhead. PSRM showed comparable workload distribution to the cluster-based remapping approach. Furthermore, unlike the cluster-based or centralised approaches, PSRM provides a higher degree of reliability as the remapping decisions are completely decentralised.

The relatively high number of PSRM's parameters and their sensitivity, makes the parameter search process challenging. A simple random parameter search showed marginal predictability improvements over the baseline remappers; however, more efficient tuning strategies such as in [201], can be used to obtain better results. Furthermore, if more information regarding the workload is known a priori (e.g. periodicity, number of streams and their resolutions), the parameters tuning could be more focused and can yield better timing improvements.

Summary of novelty and new insights: A decentralised, bio-inspired task remapping technique for NoCs is proposed. The technique has significantly lower communication overhead than a state-of-the-art cluster-based/hierarchical management protocol. The proposed technique can be used to further reduce job lateness and perform load-balancing. Experimental evaluation provided insight that whilst the proposed scheme can provide several scalable benefits over existing management approaches, properly tuning such an algorithm to obtain good performance can be challenging.

9.1.5 Workload characterisation of HEVC video stream decoding

The application model used in the technical Chapters 4-6 represent video streams encoded using classical codecs such as MPEG-2. In Chapter 7, the application model was extended to capture the complex properties of HEVC video streams with adaptive hierarchical B-frame GoP structures. GoP-level and coding-unit level properties of real HEVC video stream trace data was analysed. Tractable workload generation algorithms were introduced in Chapter 7, which use the statistical properties obtained from the stream analysis stage, to generate realistic, abstract DAG-based HEVC decoding workloads. These algorithms are robust enough to create any amount of synthetic video stream workloads with varying spatial and temporal properties, which is useful in high-level design space exploration.

The evaluation of these synthetically generated video decoding workloads showed that the variation of the different frames were not accurately captured using this model. This was due to several video stream characteristics being analysed at the stream-level, rather than on a frame-by-frame basis. However, properties such as frame execution time bounds, data dependency patterns and communication volumes of the synthetically generated workloads matched the real video stream.

Summary of novelty and new insights: DAG-based, abstract, HEVC video stream decoding workload generation models were proposed. These models were derived using CU/GoP-level statistical analysis that have not been investigated before. Evaluation revealed that the CCR of a stream is not completely random and has direct relationship with the level of spatial/temporal redundancies in the video and the encoder options used.

9.1.6 Runtime task clustering and mapping for Tile-parallel HEVC decoding

Tile-parallel HEVC stream decoding improves the decoding throughput by enabling the decoding of several portions (i.e. tiles) of a frame simultaneously. In Chapter 8, the complexities of decoding tile-parallel HEVC video streams with adaptive, hierarchical B-frame GoP structures is explored. Several application specific problems such as dynamic job dependency structures, variable video stream CCRs and HEVC tile parallelisation challenges are addressed. The work in Chapter 8 primarily attempts to balance the conflicting objectives, job lateness (predictability) and NoC communication cost reduction (energy).

Chapter 8, introduced two tile-level task clustering and mapping techniques to address the above mentioned issues. The first heuristic, termed CL, clusters the tile tasks in the longest path of the job differently than the rest of the tiles. It also considers the task blocking and the inter-task communication distance. The second heuristic, termed CL-BG, is stream-specific, such that it explicitly performs clustering based on hierarchical B-frame structures. Both mapping techniques use job CCR related parameters to control the level of task clustering (sparse/dense). Chapter 8 also explored several main memory controller (MMCP) to task selection schemes in order to reduce memory traffic bottlenecks especially seen in high CCR workloads. An MMCP selection scheme that attempts to assign different MMCPs to child tasks compared to their parent tasks is investigated (termed MMCP-InvPar). MMCP selection heuristics based on hop distance, link utilisation, uniform and random selection were also explored.

Evaluations were carried out against a load-balancing uniform mapper (LU), an existing computation and communication load balancer (PP) and variants of CL and CL-BG. CL showed superior communication cost reduction but it had a higher job lateness than the other mappers. LU showed lower job lateness levels but at the cost of high NoC usage. The CL-BG mapper, showed a good balance in communication cost and lateness reduction in both high/normal CCR conditions. The high mapping execution overhead of PP, makes it unsuitable for larger task-sets. MMCP evaluation showed that both MMCP-InvPar and uniform selection performed best in terms of job lateness reduction. Furthermore, the results showed that certain MMCP selection schemes can have different job lateness results based on the type of mapper used, indicating a relationship/dependency between them.

The experiments showed that it is infeasible to derive a lightweight runtime mapping heuristic that optimises on both target metrics under all workload conditions. Nevertheless, using a certain degree of application knowledge, a heuristic can be formed to provide a satisfactory balance between the two conflicting metrics. The primary challenge in MMCP selection is that there are only a small set of MMCPs to select from and a large number of concurrent memory transactions. More intelligent MMCP selection schemes can be employed which takes into account the overall memory traffic interference patterns when assigning MMCPs to tasks, however this would incur a large resource allocation overhead.

Summary of novelty and new insights: Tile-parallel HEVC decoding introduces new challenges to the task allocation problem as discussed above. Two, novel runtime task clustering and mapping techniques are presented, specifically for multi-stream tile-level parallel HEVC decoders, to decrease lateness and communication cost. The first mapper is general-purpose and attempts to reduce the latency of the tasks in the jobs longest path. The second is stream-specific and takes into account the popular B-frame grouping encoding option of video streams to form task clusters. Preliminary evaluations gave new insight to bottlenecks in memory traffic management, which lead to investigation of several task-to-MMCP selection schemes.

9.2 Future research directions

This section presents several research directions that can be taken to extend/build-upon the work carried out in this thesis.

- **Hardware prototyping:** The findings in this thesis were produced via simulation, albeit using abstract workloads with a high degree of realism. Having shown that the proposed techniques show promising results in simulation, future work can explore implementing them on an actual many-core hardware platform. To begin with, a wormhole switched NoC, with priority-preemptive arbitration would need to be implemented. The resource manager can either be a PE within the NoC or an individual entity outside main PE network. The distributed algorithms proposed in this work can be implemented as a middleware layer on top the PEs. However, hardware prototypes would have to address implementation specific issues, such as develop methods for multiple video stream workload input, data collection and tracking capabilities. Existing video decoders developed for shared memory, multi-threaded systems would need to be ported to a message passing programming model.

- **Investigating priority assignment schemes:** The resource management techniques proposed in this thesis purely consider the optimisation of task placement, and assume the task/flow priority assignment scheme is provided by the designers or users. Hence, the optimisation of task priority assignment is not explored. However, it is evident from the literature that jointly optimising both mapping and priority assignment can lead to better predictability in a system and therefore needs to be explored as part of future work. Dynamic priority assignment can also be investigated; for example, dynamically adjusting the priority with respect to the lateness/deadline of the task. The schedulability analysis would also need to be changed to support dynamic priorities. Independent priority assignment heuristics for tasks and flows can also be investigated. It would also be interesting to explore the behaviour of priority assignments at different levels of stream granularity i.e. coarse grain - stream level or fine-grain - tile/block-level. Similarly, periodicity-dependent priority assignment needs to be explored when video streams with varying frame rates are considered.

- **Fully distributed task dispatching:** In the bio-inspired task remapper proposed in Chapter 6, each PE autonomously arrived at remapping decisions, but notified a centralised dispatcher. A distributed task dispatching mechanism can be explored; this eliminates the need for a dispatcher notification, which only introduces very low communication overhead, but improves reliability further. One approach would be to send a single, short message around the network notifying each PE sequentially (e.g. zig-zag order), that a new job has arrived. PE's with high amounts of slack (i.e. Queen Nodes from the PSRM algorithm), can decide to take on additional load from the new job, and label the task(s) as *taken* and continue to pass the message to the next PE in the network. Each PE can also load task data from memory without the assistance from a central dispatcher/resource manager; this would enhance the reliability of the system further.

- **Application model adaptations:** There are several ways the application model can be further extended. Lower stream granularity (e.g. CTU/CU level) parallel decoding can be explored, which would be suited for systems with constrained local memory. The limitations

mentioned in the existing HEVC workload generator (Chapter 7) can also be addressed - to capture the frame-by-frame variations within the stream. The application model can also be adapted to represent streams with different encoding settings - e.g. intra-only streams or I/P-frame or I/B-frame only streams, different bit-depths etc. Realistic video stream, job arrival patterns can also be integrated into the application model. More realistic internet/network traffic models can be used such as Poisson or Pareto distributed models which capture the bursty nature of streaming video.

- **Parallelisation granularity vs. local memory utilisation and network contention:** The work in this thesis explores parallel video decoding only at the frame and Tile granularities. Coarser-grain parallelism such as at the GoP-level, requires much larger local PE memory requirements but no inter-task communication and memory transactions with large payloads. On the other hand, finer-grain parallelism such as at the CTU/CU-level, can have very small computation and memory costs, but introduces a higher amount of small payload data and memory communication transactions. System models at different granularity levels can behave differently in terms of network contention, energy consumption and latency. It would be interesting to carry out a design space exploration on hybrid stream parallelisation techniques. A decoder can dynamically combine different levels of granularity at runtime, in order to optimise the response time (when needed), whilst constrained by a local memory capacity and/or power budget.

- **Jointly optimise memory controller port selection and PE mapping**: In this work, all tasks in a job are first mapped onto PEs and then they are assigned MMCPs. Thus, the MMCP selection is a secondary phase of the resource allocation process. Future work can explore heuristics where *both* task-to-PE mapping and MMCP selection can jointly be performed for each task in the job within a single heuristic. For example, isolate consecutive P-frame/tile decoding tasks to trade-off communication cost reduction to reduce memory traffic contention. These type of sophisticated heuristics would be mainly required when managing high CCR workloads.

Appendices

Appendix A

Heu-AC parameter combination evaluation

A range of different Heu-AC parameters (IBL_α, TQL_α) were tested under the high workload (16 worklfows) condition and results are given in Figure A.1. A higher rejection rate, zero late streams and low PE utilisation is seen for very low values of IBL_α. Higher IBL_α, TQL_α values cause more admissions and late streams and therefore more resource utilisation.

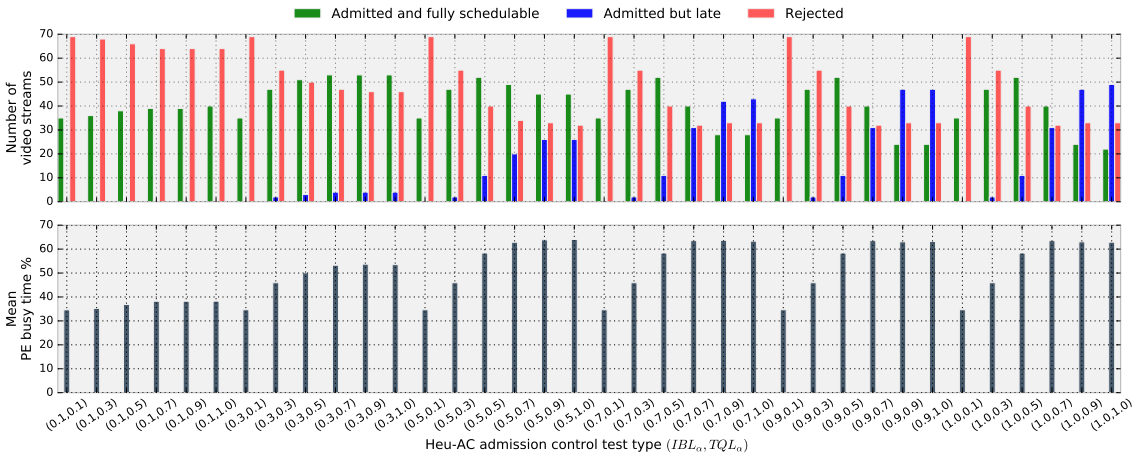


Figure A.1: Heu-AC ratio parameter (IBL_α, TQL_α) combinations evaluation results for admission rate vs. mean PE busy time

Appendix B

PSAlgo algorithm supplementary information

B.1 PSAlgo algorithm events

This appendix section contains the pheromone propagation (Algorithm B.1) and decay cycles (Algorithm B.2) as given in [104].

Algorithm B.1: PSAlgo - Propagation Cycle *(PSPropagation)*

Input : $threshold_{hc}$ - propagation threshold,
K_{hdecay} - hop decay factor,
h_i - local pheromone level
Output: hd - updated pheromone dose

if hd *received* **then**
  if $hd[0] < threshold_{hc}$ **then**
    $h_i = h_i + hd[2]$
    broadcast $hd = \{hd[0]+1, hd[1] \times K_{hdecay}, hd[2], hd[3]\}$
  else
    drop hd
  end
end

Algorithm B.2: PSAlgo - Decay Cycle *PSDecay*

Input : T_{DECAY} - decay cycle period,
K_{tdecay} - decay factor,
h_i - local pheromone level
Output: h_i - updated local pheromone level

while *true* **do**
  $h_i = h_i \times K_{tdecay}$
  wait for T_{DECAY}
end

B.2 PSRM parameter tuning results

The results of the random parameter search of the PSRM remapping technique is given in Figure B.1. The left y-axis (related to the box-plots) represent the distribution of mean job lateness improvement (per seeds) and the right y-axis (related to the line plot) represents the cumulative job lateness improvement (mean of all seeds). The x-axis indicates the respective parameter combination values evaluated; definitions of the notation used are given below:

- T_{QN} : QN differentiation cycle period

- T_{DECAY} : decay cycle period
- HC : pheromone propagation range (hop count)
- T_{RM} : task remapping cycle period
- H_{QN} : initial pheromone dosage propagated by the QN
- Q_{TH}^{α} : QN threshold increment factor
- Q_{TH}^{β} : QN threshold decrement factor
- K_{HDECAY} : hormone decay per hop
- K_{TDECAY} : rate at which hormone will decay at each decay cycle

The best parameter combination is one which gives a higher job lateness improvement. It can be noticed that certain combinations can result in a larger range of lateness improvement distributions than others.

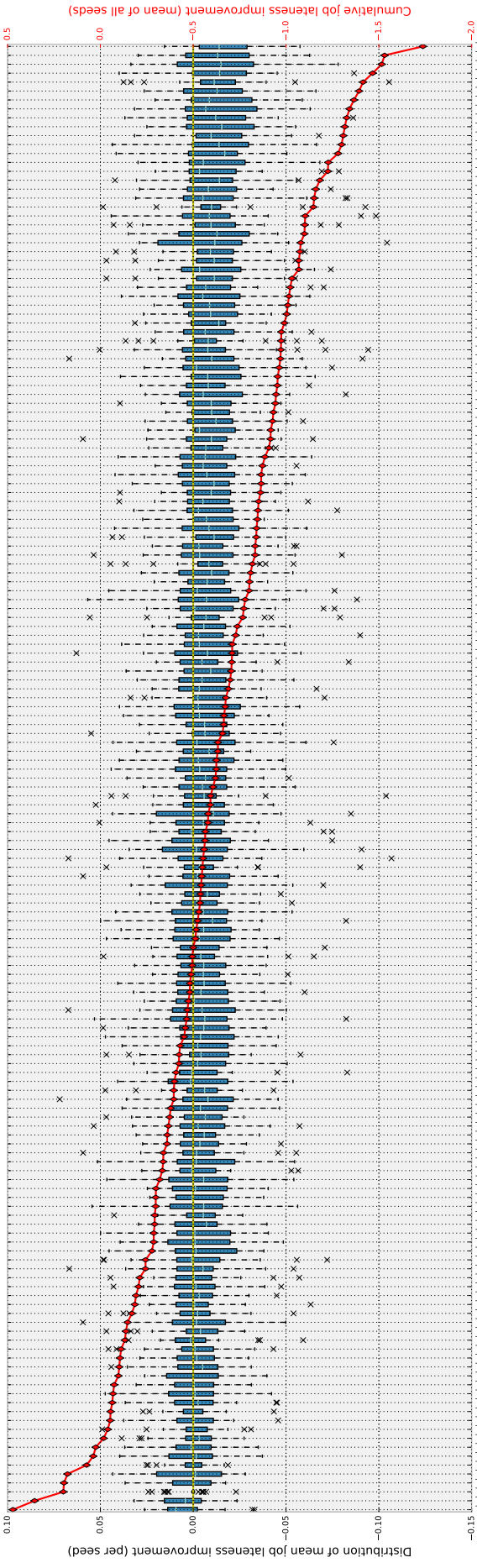


Figure B.1: Results of random search based parameter tuning of PSRM

Appendix C

HEVC workload model - supplementary information

This appendix section includes certain settings, parameters and statistical properties extracted from the video streams analysed in Chapter 7. This supplementary material will also be beneficial for the reproducibility of the analysis and experiments performed in Chapter 7.

C.1 Encoder and decoder settings

This section provides the encoder and decoder settings (Listing C.1 and Listing C.2 respectively), used during the creation and playback of the video sequences tested.

Listing C.1: x265 encoder command-line arguments

```
x265 --input video_raw.yuv -o video_enc.bin -I 35 --b-adapt 2 --bframes 4 --no-weightp
--no-weightb --no-open-gop --b-intra --ref 2 --csv video_stats.csv --csv-log-level 2
--log-level 4
```

Listing C.2: OpenHEVC decoder command-line arguments

```
ohevc -i video_enc.bin -f 1 -o video_dec.yuv -p 1 -n
```

C.2 Distribution fitting parameters

The following tables contain the workload parameters obtained via stream analysis/evaluation:

- Parameters of the exp-Weibull distribution fitted to the CU decoding time (I-CU, P-CU and B-CU) - Table C.1
- Coefficients of the polynomial fit to the Skip-CU decoding time - Table C.2
- Parameters of the exp-Weibull distribution fitted to the encoded frame size - Table C.3
- CU-decoding time scale factors obtained through evaluation - Table C.4

C.3 Parameters used for evaluation

The workload generator was used to synthetically generate video stream GoPs that represented the *LionWildlife* video characteristics. Table C.5 shows the workload generator parameters used.

Table C.1: I/P/B-CU decoding time distribution, exp-Weibull fit shape parameters

CU type	a	c	scale
FastFurious5			
I-CU	1.77E+01	6.56E-01	3.05E-06
P-CU	7.47	1.28	8.42E-06
B-CU	1.38	2.55	4.51E-05
LionWildlife			
I-CU	9.24	7.43E-01	5.06E-06
P-CU	3.29	1.57	1.58E-05
B-CU	1.38	2.78	5.78E-05
Football			
I-CU	2.87E+03	3.21E-01	1.02E-08
P-CU	9.12E+02	4.46E-01	1.44E-07
B-CU	7.13	1.35	1.99E-05
ObamaSpeech			
I-CU	1.23E+03	3.25E-01	2.09E-08
P-CU	8.07E+02	3.93E-01	1.08E-07
B-CU	6.54	1.14	2.49E-05
BigBuckBunny			
I-CU	6.56E+02	3.08E-01	1.93E-08
P-CU	3.47E+01	5.62E-01	1.49E-06
B-CU	4.74	9.47E-01	1.91E-05

Table C.2: Skip-CU decoding time distribution, polynomial fit coefficients

FastFurious5
$f(x) = 2.34\times10^{53} - 4.62\times10^{49}x + 3.87\times10^{45}x^2 - 1.78\times10^{41}x^3 + 4.84\times10^{36}x^4 - 7.63\times10^{31}x^5 + 5.82\times10^{26}x^6 + 4.03\times10^{20}x^7 - 4.09\times10^{16}x^8 + 2.45\times10^{11}x^9 - 2.99\times10^{5}x^{10}$
LionWildlife
$f(x) = -8.75\times10^{49} + 3.92\times10^{46}x - 7.60\times10^{42}x^2 + 8.35\times10^{38}x^3 - 5.71\times10^{34}x^4 + 2.52\times10^{30}x^5 - 7.21\times10^{25}x^6 + 1.30\times10^{21}x^7 - 1.38\times10^{16}x^8 + 7.40\times10^{10}x^9 - 9.15\times10^{4}x^{10}$
Football
$f(x) = -8.54\times10^{50} + 3.41\times10^{47}x - 5.87\times10^{43}x^2 + 5.71\times10^{39}x^3 - 3.43\times10^{35}x^4 + 1.32\times10^{31}x^5 - 3.24\times10^{26}x^6 + 4.90\times10^{21}x^7 - 4.22\times10^{16}x^8 + 1.75\times10^{11}x^9 - 1.82\times10^{5}x^{10}$
ObamaSpeech
$f(x) = 5.45\times10^{52} - 4.99\times10^{48}x - 3.55\times10^{44}x^2 + 7.47\times10^{40}x^3 - 4.81\times10^{36}x^4 + 1.65\times10^{32}x^5 - 3.32\times10^{27}x^6 + 3.93\times10^{22}x^7 - 2.61\times10^{17}x^8 + 8.51\times10^{11}x^9 - 9.65\times10^{5}x^{10}$
BigBuckBunny
$f(x) = -1.13\times10^{50} + 5.22\times10^{46}x - 1.04\times10^{43}x^2 + 1.16\times10^{39}x^3 - 8.05\times10^{34}x^4 + 3.58\times10^{30}x^5 - 1.03\times10^{26}x^6 + 1.84\times10^{21}x^7 - 1.92\times10^{16}x^8 + 9.95\times10^{10}x^9 - 1.16\times10^{5}x^{10}$

Table C.3: Encoded frame size distributions, exp-Weibull fit shape parameters.

Fr. type	a	c	loc	scale.
FastFurious5				
I-Fr	4.76E+01	5.15E-01	5.81E-04	1.93E-04
P-Fr	1.57E+01	7.90E-01	2.71E-05	4.41E-04
B-Fr	1.29	7.38E-01	2.46E-05	3.08E-04
LionWildlife				
I-Fr	1.31E+02	4.50E-01	0.00	1.52E-04
P-Fr	1.20E+01	5.07E-01	0.00	1.27E-04
B-Fr	1.17	7.79E-01	1.99E-05	2.47E-04
Football				
I-Fr	6.05E-01	2.74E+00	2.74E-03	1.81E-02
P-Fr	4.15	9.49E-01	1.45E-04	1.56E-03
B-Fr	1.16	8.88E-01	1.92E-05	4.15E-04
ObamaSpeech				
I-Fr	4.56E-01	6.21E+00	1.29E-02	3.69E-03
P-Fr	1.60	1.18E+00	3.79E-05	4.17E-04
B-Fr	8.45E+01	2.66E-01	2.31E-05	1.10E-07
BigBuckBunny				
I-Fr	3.36	1.23	0.00	1.39E-02
P-Fr	5.51	4.59E-1	2.23E-05	1.70E-04
B-Fr	9.80E-01	7.34E-01	3.17E-05	2.57E-04

Table C.4: CU-decoding time scale factors

CU-type	Scale factor
I-CU	random.uniform(1.4, 1.5)
P-CU	random.uniform(1.4, 1.9)
B-CU	random.uniform(1.0, 1.0)
Skip-CU	random.uniform(1.0, 1.01)

NB. Further scaling by 0.6 is applied when accounting for memory transactions.

Table C.5: Workload generator parameters used for evaluation (GoP size : N=36, 200 GoPs)

Contiguous B-frame probabilities B-frame groups of size 1, 2, 3, 4 = {0.005, 0.49, 0.505, 0.001}
Reference distance probabilities RFD: {1, 2, 3} = {0.194, 0.752, 0.054}
Number of P-frames exp-Weibull PDF (a, c, scale, loc) = {68.03, 0.72, 1.5, 0.0}
CU type probabilities (I/P/B/Skip-CU) I-fr = {1.0, 0.0, 0.0, 0.0} P-fr = {0.22, 0.404, 0.0, 0.377} B-fr = {0.041, 0.179, 0.094, 0.686}
CU size probabilities ($64\times64, 32\times32, 16\times16, 8\times8, 4\times4$): Intra-frame = {0.098, 0.262, 0.404, 0.183, 0.053} Inter-frame = {0.27, 0.31, 0.27, 0.15, 0.0}
I/P/B-CU decoding exp-Weibull parameters (a, c, scale, loc): I-CU = {9.24, 7.43E-01, 5.06E-06, 0.0} P-CU = {3.29, 1.57, 1.58E-05, 0.0} B-CU = {1.38, 2.78, 5.78E-05, 0.0} min/max timing ranges : I-CU: {3.28E-06 ,1.76E-04} P-CU: {3.33E-06 ,1.33E-04} B-CU: {3.08E-06 ,1.75E-04}
Skip-CU decoding polynomial parameters $f(x) = -8.75\times10^{49} + 3.92\times10^{46}x - 7.60\times10^{42}x^2 + 8.35\times10^{38}x^3 - 5.71\times10^{34}x^4 + 2.52\times10^{30}x^5 - 7.21\times10^{25}x^6 + 1.30\times10^{21}x^7 - 1.38\times10^{16}x^8 + 7.40\times10^{10}x^9 - 9.15\times10^{4}x^{10}$ min/max timing ranges: {1.20E-06, 8.04E-05}
Reference data frame selection probabilities Target frame : Reference frame P-fr : {I-fr: 0.55, P-fr: 0.45, B-fr: 0.0}, B-fr : {I-fr: 0.60, P-fr: 0.30, B-fr: 0.10}

Appendix D

HEVC tile mapping - supplementary information

D.1 Computation and communication requirements of synthetically generated HEVC streams

The figures contained in this section show the computation and communication requirements of several synthetically generated HEVC streams (as per Chapter 7). The workloads have been generated using the parameters obtained from real video stream analysis carried out in Chapter 7. In the figure, the video types represent the streams that were analysed in Chapter 7. ACTION, DOCUMENTARY (DOC), SPORT, ANIMATION (ANIM) and SPEECH represent the FastFurious5, LionWildlife, Football, BigBuckBunny and ObamaSpeech videos respectively. The communication cost of the jobs assume a 100MHz NoC and the computation costs assume the CU-level decoding time characteristics as per Chapter 7.

Figure D.1 shows the B-frame computation cost for increasing resolution levels and varying video genres. The I and P-frame computation costs follow a similar trend. The varying computation costs arises from the different CU types in a frame based on the temporal and spatial dependency in the video sequence. Note that, to omit the memory latencies within the decoding computation cost measurements, the CU-level decoding times have been scaled down by a factor of 0.6 (further discussed in Section 7.8.3). 2160p frame decoding cost is over an order of magnitude higher than 288p videos. A similar rate of increase can be seen in reference frame data payloads for varying resolution levels as shown in Figure D.2. Figure D.3 shows the number of edges in a GoP/job for different video resolutions, GoP lengths and video genres.

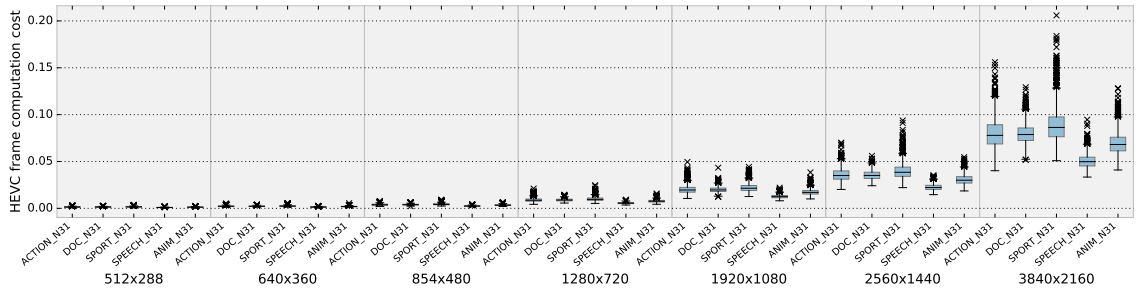


Figure D.1: Synthetic HEVC B-frame computation cost for varying video stream resolutions and types (GoP length $N = 31$)

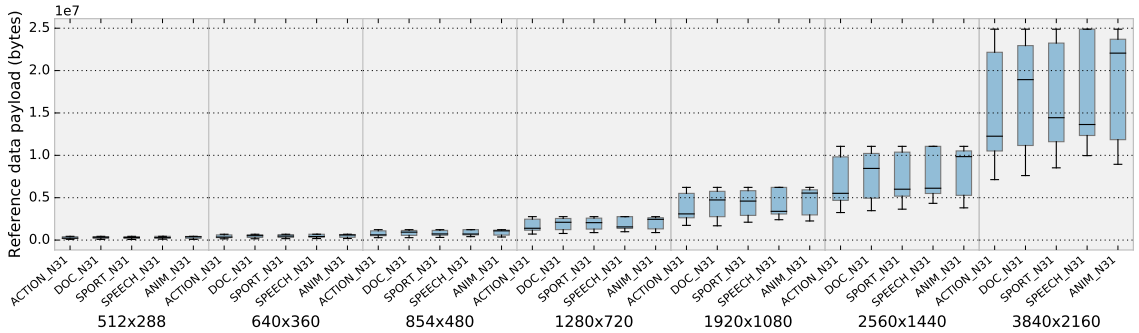


Figure D.2: Reference data payloads per synthetically generated HEVC frame, for varying video stream resolutions and types (GoP length $N = 31$)

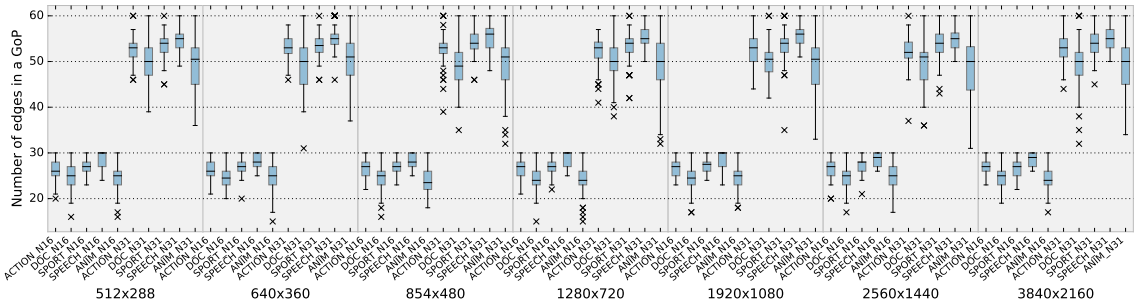


Figure D.3: Number of edges in synthetically generated GoPs with max. 4 contiguous B-frames and max. 2 reference frames per fwd/bwd direction. For varying video stream resolutions and types (GoP length $N = 16, 31$)

D.2 CL-BG HEVC tile mapper parameter tuning results

Figure D.4 shows the NH_{GT} hop count parameter being varied between 1-6 for all 3 (low/med/high) CCR ranges defined in Table 8.2. The metrics presented in the figure are the job lateness (for 3 seeded experimental runs) and the cumulative communication basic latency of the message flows. $NH_{GT} = 1$ for med/high CCR ranges gave very low communication costs. Relatively, larger NH_{GT} values gave a low maximum job lateness but at the cost of increased communication cost (e.g. 2,4,6, 6,6,3).

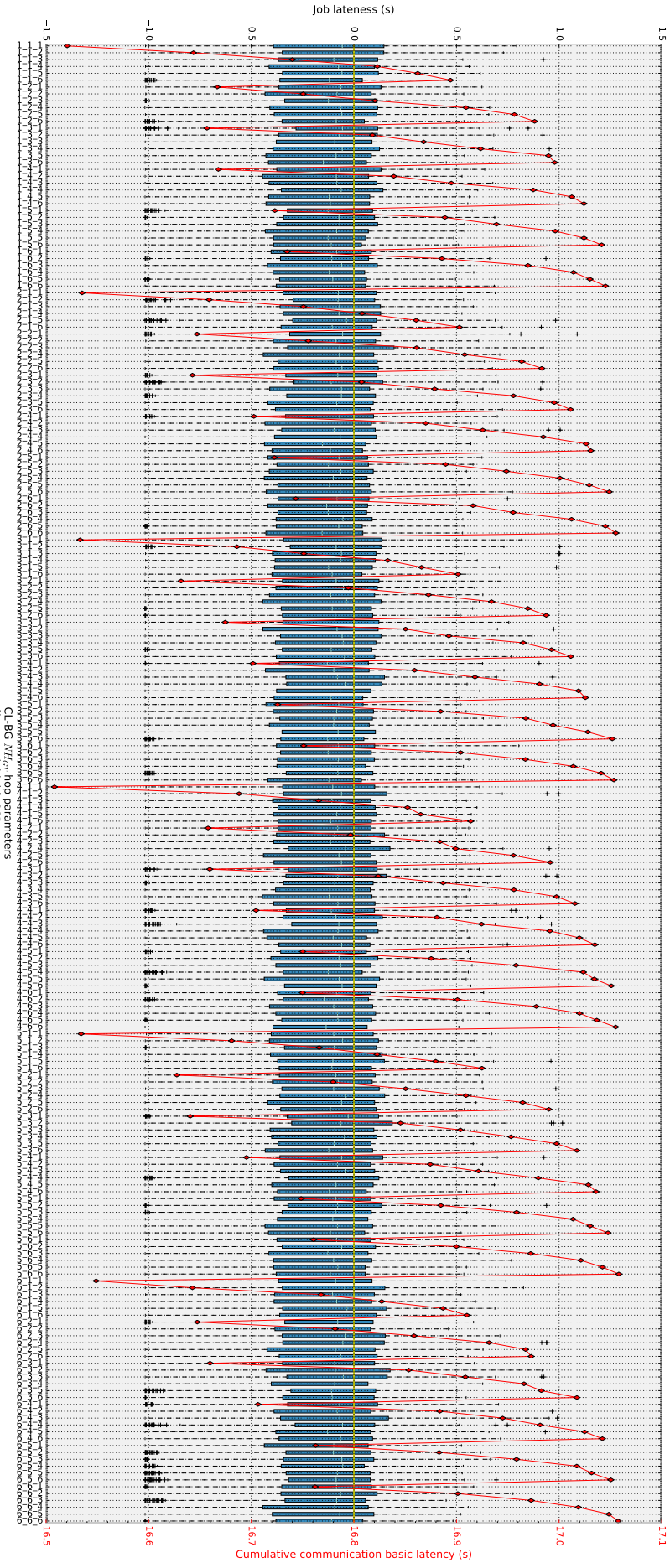


Figure D.4: CL-BG tile mapper NH_{GT} hop parameter tuning results

Abbreviations

AC	Admission Control
AVC	Advanced Video Coding
BN	Best Neighbour
CCPRM	Castilho's Cluster based Protocol based ReMapping
CCR	Communication to Computation Ratio
CL	Clustering based
CL-BG	Clustering based on B- frame Grouping
CPU	Central Processing Unit
CTU/CU	Coding Tree Unit/Coding Unit
D-AC	Deterministic Admission Controller
DAG	Directed Acyclic Graphs
DPB	Dependency Picture Buffer
DRAM	Dynamic Random Access Memory
DSE	Design Space Exploration
DVFS	Dynamic Voltage and Frequency Scaling
E2ERTA	End-to-End Response Time Analysis
EQF	EQual Flexibility
FCFS	First Come First Served
FPGA	Field Programmable Gate Array
GA/GA-MP	Genetic Algorithm/Genetic Algorithm based Mapper
GoP	Group of Pictures
H-AC	Heuristic-based Admission Controller
HEVC	High Efficiency Video Coding
HRT/SRT	Hard/Soft Real Time
IBL/TQL	Input Buffer Lateness/Task Queue Lateness
IDCT	Inverse Discrete Cosine Transform
IPC	I and P frames Combined
IQR	Inter-Quartile Range
LM/LU	Least Mapped/Least Utilised
LMP/GMP	Local cluster Manager/Global manager
LWCRS	Least Worst Case Remaining Slack

MMC/MMCP	Main Memory Controller/Main Memory Controller Port
MP&PR	Mapping and Priority assignment
MPEG	Motion Picture Experts Group
MPSoC	Multiprocessor System-on-Chip
NoC	Network-on-chip
PDF	Probability Density Function
PE	Processing Element
PP	Preprocessing Procedure
PSRM	Pheromone Signalling based ReMapping
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RFD	Reference Distance
RM	Rate Monotonic
RTA	Response Time Analysis
RTS	Real Time System
SD/HD	standard/high definition
TDM	Time Division Multiplexing
TG	Task Graph
TMT	Task Mapping Table
TQ	Task Queue
UHD	Ultra-high definition
VBR	Variable Bit Rate
VLC	Variable Length Coding
WCET	Worst Case Execution Time
WCRT	Worst Case Response Time
WN/QN	Worker Node/Queen Node
WPP	Wavefront Parallel Processing

Bibliography

[1] R. R. Schaller, "Moore's law: past, present and future," *IEEE Spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[2] S. Borkar, "Thousand core chips: a technology perspective," in *Design Automation Conference (DAC)*. ACM, 2007, pp. 746–749.

[3] Kalray, "MPPA-256 processor product brief, (Feb 2014)," URL: http://www.kalrayinc.com/IMG/pdf/FLYER_MPPA_MANYCORE.pdf, visited on 2016-06-21.

[4] EZchip, "Tile-gx72 processor product brief, pb041-rel. 4.0 (17 april 2015)," URL: http://www.tilera.com/files/drim__TILE-Gx8072_PB041-04_WEB_7683.pdf, visited on 2016-01-13.

[5] Samsung, "Exynos 7 Octa (7420) 8-core mobile processor," URL: http://www.samsung.com/semiconductor/products/exynos-solution/application-processor/EXYNOS-7-OCTA-7420, visited on 2016-06-22.

[6] MediaTek, "MediaTek Helio X20 - 10 core mobile processor," URL: http://mediatek-helio.com/x20/, visited on 2016-06-22.

[7] ITRS, "International technology roadmap for semiconductors," URL: http://www.itrs.net/, ITRS, Tech. Rep., 2012.

[8] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[9] BBC, "Online article: The barriers to video on the move," URL: http://news.bbc.co.uk/1/hi/entertainment/4338508.stm, Oct 2005, visited on 2016-06-23.

[10] Qualcomm, "Enabling the full 4k mobile experience: System leadership," URL: https://www.qualcomm.com/documents/enabling-full-4k-mobile-experience-system-leadership, visited on 2016-06-21.

[11] M. Alvarez, E. Salamí, A. Ramirez, and M. Valero, "A performance characterization of high definition digital video decoding using H.264/AVC," in *IEEE International on Workload Characterization Symposium*, 2005, pp. 24–33.

[12] A. Bilas, J. Fritts, and J. P. Singh, "Real-time parallel MPEG-2 decoding in software," in *International Parallel Processing Symposium*. IEEE, 1997, pp. 197–203.

[13] G. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[14] ITU Telecommunication Standardization, "H.265 - 04/2015 - Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS infrastructure of audiovisual services - Coding of moving video - High efficiency video coding," ITU-T, Tech. Rep., Apr 2015.

[15] G.-M. Su, X. Su, Y. Bai, M. Wang, A. V. Vasilakos, and H. Wang, "QoE in video streaming over wireless networks: perspectives and research challenges," *Journal on Wireless Networks*, pp. 1–23, Aug 2015.

[16] J. Marescaux, J. Leroy, M. Gagner, F. Rubino, D. Mutter, M. Vix, S. E. Butner, and M. K. Smith, "Transatlantic robot-assisted telesurgery," *Nature*, vol. 413, no. 6854, p. 379380, Sep 2001.

[17] B. Hannaford, J. Rosen, D. W. Friedman, H. King, P. Roan, L. Cheng, D. Glozman, J. Ma, S. N. Kosari, and L. White, "Raven-II: An open platform for surgical robotics research," *IEEE Transactions on Biomedical Engineering*, vol. 60, no. 4, pp. 954–959, Apr 2013.

[18] G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, and V. Zaccaria, "An industrial design space exploration framework for supporting run-time resource management on multi-core systems," in *International Conference and Exhibition on Design, Automation Test in Europe (DATE)*, 2010, pp. 196–201.

[19] W.-Y. Shieh and J.-C. Huang, "Falling-incident detection and throughput enhancement in a multi-camera video-surveillance system," *Medical Engineering & Physics*, vol. 34, no. 7, pp. 954–963, Sep 2012.

[20] M. McGill, J. H. Williamson, and S. A. Brewster, "A review of collocated multi-user TV," *Journal on Personal and Ubiquitous Computing*, vol. 19, no. 56, pp. 743–759, Jun 2015.

[21] S. Medic, N. Spiric, S. Tanackovic, M. Vidakovic, and N. Kuzmanovic, "Proposition for mosaic video playback on android based DTV devices," in *IEEE International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, 2013, pp. 1–3.

[22] Mividi, "Broadcast TV multiviewer monitoring system," URL: http://mocomsoft.com/en-US/products/MultiViewMonitoring.aspx, visited on 2016-06-23.

[23] A. Eleftheriadis, M. R. Civanlar, and O. Shapiro, "Multipoint videoconferencing with scalable video coding," *Journal of Zhejiang University SCIENCE A*, vol. 7, pp. 696–705, May 2006.

[24] Q. Li, Y. Andreopoulos, and M. van der Schaar, "Streaming-viability analysis and packet scheduling for video over in-vehicle wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 56, no. 6, pp. 3533–3549, Nov 2007.

[25] G. Georgakarakos, L. Tsiopoulos, J. Lillius, J. Haldin, and U. Falk, "Performance evaluation of parallel HEVC strategies," in *Euromicro Conference on Parallel Distributed and Network-Based Processing*, 2015, pp. 137–144.

[26] C. C. Chi, M. Alvarez-Mesa, J. Lucas, B. Juurlink, and T. Schierl, "Parallel HEVC Decoding on Multi- and Many-core Architectures: A Power and Performance Analysis," *Journal of Signal Processing Systems*, vol. 71, pp. 247–260, 2013.

[27] M. Mandelli, L. Ost, G. Sassatelli, and F. Moraes, "Trading-off system load and communication in mapping heuristics for improving NoC-based MPSoCs reliability," in *International Symposium on Quality Electronic Design*, 2015, pp. 392–396.

[28] A. M. Rahmani, M. H. Haghbayan, A. Kanduri, A. Y. Weldezion, P. Liljeberg, J. Plosila, A. Jantsch, and H. Tenhunen, "Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach," in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2015, pp. 219–224.

[29] C. Meenderinck, A. Azevedo, B. Juurlink, M. A. Mesa, and A. Ramirez, "Parallel scalability of video decoders," *Journal of Signal Processing Systems*, vol. 57, pp. 173–194, Nov 2009.

[30] 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 (DAC)*, 2013, pp. 1–10.

[31] W. Quan and A. D. Pimentel, "A hybrid task mapping algorithm for heterogeneous MPSoCs," *ACM Transactions on Embedded Computer Systems*, vol. 14, no. 1, pp. 14:1–14:25, Jan 2015.

[32] 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.

[33] S. Kaushik, A. Singh, and T. Srikanthan, "Computation and communication aware runtime mapping for NoC-based MPSoC platforms," in *System-on-Chip Conference (SOCC)*, 2011, pp. 185–190.

[34] O. Moreira, J.-D. Mol, M. Bekooij, and J. Van Meerbergen, "Multiprocessor resource allocation for hard-real-time streaming with a dynamic job-mix," in *IEEE Real Time and Embedded Technology and Applications Symp. (RTAS)*, 2005, pp. 332–341.

[35] 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 (DAC)*, 2008, pp. 760–765.

[36] S. Kobbe, L. Bauer, D. Lohmann, W. Schröder-Preikschat, and J. Henkel, "DistRM: distributed resource management for on-chip many-core systems," in *International Conference on Hardware/software codesign and system synthesis (CODES+ISSS)*, 2011, pp. 119–128.

[37] G. Castilhos, M. Mandelli, L. Ost, and F. G. Moraes, "Hierarchical energy monitoring for task mapping in many-core systems," *Journal of Systems Architecture*, vol. 63, pp. 80–92, Feb 2016.

[38] L. Ost, M. Mandelli, G. M. Almeida, L. Moller, L. S. Indrusiak, G. Sassatelli, P. Benoit, M. Glesner, M. Robert, and F. Moraes, "Power-aware dynamic mapping heuristics for NoC-based MPSoCs using a unified model-based approach," *ACM Transactions on Embedded Computer Systems*, vol. 12, no. 3, pp. 75:1–75:22, Apr 2013.

[39] J. Huang, A. Raabe, C. Buckl, and A. Knoll, "A workflow for runtime adaptive task allocation on heterogeneous MPSoCs," in *International Conference and Exhibition on Design, Automation and Test in Europe (DATE)*, 2011, pp. 1–6.

[40] B. Kienhuis, E. F. Deprettere, P. Van Der Wolf, and K. Vissers, *A methodology to design programmable embedded systems*. Springer Berlin Heidelberg, 2002.

[41] J. Watkinson, *The MPEG Handbook*, 2nd ed. Focal Press, Sep 2004.

[42] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 560–576, Jul 2003.

[43] European Telecommunications Standards Institute (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," (ETSI), Tech. Rep., Jun 2011.

[44] B. Zatt, M. Porto, J. Scharcanski, and S. Bampi, "GOP structure adaptive to the video content for efficient H. 264/AVC encoding," in *IEEE International Conference on Image Processing (ICIP)*, 2010, pp. 3053–3056.

[45] S. Sowmyayani, J. Rani, and P. Arockia, "Adaptive GOP structure to H.264/AVC based on scene change," *ICTACT Journal on image and video processing: special issue on video processing for multimedia systems*, vol. 5, no. 1, pp. 868–872, 2014.

[46] I. E. Richardson, *The H.264 Advanced Video Compression Standard*, 2nd ed. Wiley-Blackwell, Apr 2010.

[47] V. Sze, M. Budagavi, and G. J. Sullivan, *High Efficiency Video Coding (HEVC), Algorithms and Architectures*. Springer, 2014.

[48] J. Ohm, G. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards; including High Efficiency Video Coding (HEVC)," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1669–1684, Dec 2012.

[49] M. Holliman and Y. K. Chen, "MPEG decoding workload characterization," in *Workshop on Computer Architecture Evaluation using Commercial Workloads*, 2003, pp. 23–34.

[50] S. Saponara, K. Denolf, G. Lafruit, C. Blanch, and J. Bormans, "Performance and complexity co-evaluation of the advanced video coding standard for cost-effective multimedia communications," *EURASIP Journal on Applied Signal Processing*, vol. 2004, pp. 220–235, Jan 2004.

[51] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC complexity and implementation analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1685–1696, Dec 2012.

[52] J. Brandenburg and B. Stabernack, "Exploring the concurrent execution of HEVC intra encoding algorithms for heterogeneous multi core architectures," in *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Sep 2015, pp. 1–8.

[53] G. Cebrin-Mrquez, J. L. Hernndez-Losada, J. L. Martnez, P. Cuenca, M. Tang, and J. Wen, "Accelerating HEVC using heterogeneous platforms," *The Journal of Supercomputing*, vol. 71, no. 2, pp. 613–628, Feb 2015.

[54] Y.-i. Kim, J.-T. Kim, S. Bae, H. Baik, and H. J. Song, "H.264/AVC decoder parallelization and optimization on asymetric multicore platform using dynamic load balancing," in *IEEE International Conference on Multimedia and Expo (ICME)*, Jun 2008, pp. 1001–1004.

[55] B. Bross, M. Alvarez-Mesa, V. George, C. C. Chi, T. Mayer, B. Juurlink, and T. Schierl, "HEVC real-time decoding," *SPIE Journal, Applications of Digital Image Processing XXXVI*, vol. 8856, 2013.

[56] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel Scalability and Efficiency of HEVC Parallelization Approaches," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1827–1838, 2012.

[57] H. Jo, D. Sim, and B. Jeon, "Hybrid parallelization for HEVC decoder," in *International Congress on Image and Signal Processing (CISP)*. IEEE, 2013, pp. 170–175.

[58] Y. Duan, J. Sun, L. Yan, K. Chen, and Z. Guo, "Novel efficient HEVC decoding solution on general-purpose processors," *IEEE Transactions on Multimedia*, vol. 16, no. 7, pp. 1915–1928, Nov 2014.

[59] C. C. Chi, M. Alvarez-Mesa, B. Bross, B. Juurlink, and T. Schierl, "SIMD acceleration for HEVC decoding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 5, pp. 841–855, May 2015.

[60] 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 Systems (ECRTS)*, 2003, pp. 73–82.

[61] N. Mastronarde, K. Kanoun, D. Atienza, P. Frossard, and M. van der Schaar, "Markov decision process based energy-efficient on-line scheduling for slice-parallel video decoders on multicore systems," *IEEE Transactions on Multimedia*, vol. 15, no. 2, pp. 268–278, Feb 2013.

[62] D. Dixon, *Adobe Encore DVD: In the Studio*. OReilly Media, Inc., 2004.

[63] S. Tanwir and H. Perros, "A survey of VBR video traffic models," *IEEE Communications Surveys Tutorials*, pp. 1778–1802, 2013.

[64] O. Hohlfeld, E. Pujol, F. Ciucu, A. Feldmann, and P. Barford, "BufferBloat - how relevant? a QoE perspective on buffer sizing," Technische Universitat Berlin, Tech. Rep., 2012.

[65] S. Egger, T. Hossfeld, R. Schatz, and M. Fiedler, "Waiting times in quality of experience for web based services," in *International Workshop on Quality of Multimedia Experience (QoMEX)*. IEEE, 2012, pp. 86–6.

[66] W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast, and Z. Wang, "A NoC traffic suite based on real applications," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2011, pp. 66–71.

[67] H. Jung, C. Lee, S.-H. Kang, S. Kim, H. Oh, and S. Ha, "Dynamic behavior specification and dynamic mapping for real-time embedded systems: HOPES approach," *ACM Transactions on Embeded Computer Systems*, vol. 13, no. 4s, pp. 135:1–135:26, 2014.

[68] A. S. Mashat, "VBR MPEG traffic: characterisation, modelling and support over ATM networks," Ph.D. dissertation, University of Leeds, UK, 1999.

[69] A. C. Bavier, A. B. Montz, and L. L. Peterson, "Predicting MPEG execution times," in *ACM SIGMETRICS Performance Evaluation Review*. ACM, 1998, pp. 131–140.

[70] Y. Tan, P. Malani, Q. Qiu, and Q. Wu, "Workload prediction and dynamic voltage scaling for MPEG decoding," in *Asia and South Pacific Design Automation Conference (ASPDAC)*. IEEE Press, 2006, pp. 911–916.

[71] F. H. Seitner, R. M. Schreier, M. Bleyer, and M. Gelautz, "A macroblock-level analysis on the dynamic behaviour of an H.264 decoder," in *International Symposium on Consumer Electronics*, 2007, pp. 1–5.

[72] Z. Ma, H. Hu, and Y. Wang, "On complexity modeling of H.264/AVC video decoding and its application for energy efficient decoding," *IEEE Transactions on Multimedia*, vol. 13, no. 6, pp. 1240–1255, Dec 2011.

[73] G. Chiruvolu, T. K. Das, R. Sankar, and N. Ranganathan, "A scene-based generalized markov chain model for VBR video traffic," in *IEEE International Conference on Communications (ICC)*. IEEE, 1998, pp. 554–558.

[74] Intel, "Intel Xeon Phi product family," URL: http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html, 2015, visited on 2016-01-16.

[75] Xilinx, "Zynq UltraScale+ MPSoC," URL: http://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html, 2015, visited on 2016-01-16.

[76] S. Dutta, R. Jensen, and A. Rieckmann, "Viper: A multiprocessor SOC for advanced set-top box and digital TV systems," *IEEE Design Test of Computers*, vol. 18, no. 5, pp. 21–31, Oct 2001.

[77] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, no. 1, 2006.

[78] W. Jerraya, Ahmed Amine; Wolf, *Multiprocessor Systems-On-Chips*. Elsevier, 2005.

[79] R. A. Shafik, P. Rosinger, and B. M. Al-Hashimi, "Mpeg-based performance comparison between network-on-chip and amba mpsoc," in *2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, Apr 2008, p. 16.

[80] L. S. Indrusiak, "End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration," *Journal of Systems Architecture*, vol. 60, pp. 553–561, 2014.

[81] F. Moraes, N. Calazans, A. Mello, L. Mller, and L. Ost, "HERMES: an infrastructure for low area overhead packet-switching networks on chip," *Integration, The VLSI Journal*, vol. 38, no. 1, pp. 69–93, Oct 2004.

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

[83] M. Li, Q.-A. Zeng, and W.-B. Jone, "DyXY: A proximity congestion-aware deadlock-free dynamic routing method for network on chip," in *Design Automation Conference (DAC)*. ACM, 2006, pp. 849–852.

[84] A. Agarwal, C. Iskander, and R. Shankar, "Survey of network on chip (NoC) architectures & contributions," *Journal of Engineering, Computing and Architecture*, vol. 3, no. 1, pp. 21–27, 2009.

[85] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal network on chip: concepts, architectures, and implementations," *IEEE Design Test of Computers*, vol. 22, no. 5, pp. 414–421, 2005.

[86] S. Liu, A. Jantsch, and Z. Lu, "Analysis and evaluation of circuit switched NoC and packet switched NoC," in *Euromicro Conference on Digital System Design (DSD)*, 2013, pp. 21–28.

[87] D. Wu, B. M. Al-Hashimi, and M. T. Schmitz, "Improving routing efficiency for network-on-chip through contention-aware input selection," in *Asia and South Pacific Design Automation Conference (ASPDAC)*, 2006, p. 3641.

[88] B. Nikolic, P. Meumeu Yomsi, and S. M. Petters, "Worst-case memory traffic analysis for many-cores using a limited migrative model," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2013, pp. 42–51.

[89] A. Vajda, *Multi-core and Many-core Processor Architectures*. Springer US, 2011.

[90] M. M. K. Martin, M. D. Hill, and D. J. Sorin, "Why on-chip cache coherence is here to stay," *ACM Communications*, vol. 55, no. 7, pp. 78–89, Jul 2012.

[91] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of scheduling techniques for addressing shared resources in multicore processors," *ACM Computing Surveys*, vol. 45, pp. 4:1–4:28, Dec 2012.

[92] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, and et al., "T-CREST: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 449–471, Oct 2015.

[93] D. Abts, N. D. Enright Jerger, J. Kim, D. Gibson, and M. H. Lipasti, "Achieving predictable performance through better memory controller placement in many-core CMPs," in *International Symposium on Computer Architecture*. ACM, 2009, pp. 451–461.

[94] M. Awasthi, D. W. Nellans, K. Sudan, R. Balasubramonian, and A. Davis, "Handling the problems and opportunities posed by multiple on-chip memory controllers," in *International conference on Parallel architectures and compilation techniques (PACT)*. ACM, 2010, pp. 319–330.

[95] A. Azevedo, C. Meenderinck, B. Juurlink, A. Terechko, J. Hoogerbrugge, M. Alvarez, and A. Ramirez, *Parallel H.264 decoding on an embedded multicore processor*. Springer, 2009.

[96] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing*, 2nd ed. Addison Wesley, Feb 2003.

[97] D. Kranz, K. Johnson, A. Agarwal, J. Kubiatowicz, and B.-H. Lim, "Integrating message-passing and shared-memory: early experience," in *ACM SIGPLAN symposium on Principles and practice of parallel programming (PPOPP)*. ACM, 1993, pp. 54–63.

[98] C. Zimmer and F. Mueller, "Nocmsg: Scalable noc-based message passing," in *Cluster, Cloud and Grid Computing (CCGrid), IEEE/ACM International Symposium on*. IEEE, 2014, pp. 186–195.

[99] M. R. Casu, M. R. Roch, S. V. Tota, and M. Zamboni, "A NoC-based hybrid message-passing/shared-memory approach to CMP design," *Journal on Microprocessors and Microsystems*, vol. 35, no. 2, pp. 261–273, Mar 2011.

[100] N. Ma, Z. Lu, and L. Zheng, "System design of full HD MVC decoding on mesh-based multicore NoCs," *Journal on Microprocessors and Microsystems*, vol. 35, no. 2, pp. 217–229, Mar 2011.

[101] H. Migalln, V. Galiano, P. Piol, O. Lpez-Granado, and M. P. Malumbres, "Distributed memory parallel approaches for HEVC encoder," *The Journal of Supercomputing*, pp. 1–12, Feb 2016.

[102] K. Sigdel, M. Thompson, C. Galuzzi, A. D. Pimentel, and K. Bertels, "rSesame - a generic system-level runtime simulation framework for reconfigurable architectures," in *International Conference on Field-Programmable Technology FPT*. IEEE, 2009, pp. 460–464.

[103] Y. Ben-Itzhak, I. Cidon, and A. Kolodny, "Delay analysis of wormhole based heterogeneous NoC," in *IEEE/ACM International Symposium on Networks on Chip (NoCS)*, May 2011, pp. 161–168.

[104] 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," *International Journal of Distributed Sensor Networks*, vol. 2013, pp. 1–14, 2013.

[105] M. Hosseinabady and J. l. Nunez-Yanez, "Fast and low overhead architectural transaction level modelling for large-scale network-on-chip simulation," *IET Computers Digital Techniques*, vol. 6, no. 6, pp. 384–395, Nov 2012.

[106] L. Indrusiak, J. Harbin, and O. M. Dos Santos, "Fast simulation of networks-on-chip with priority-preemptive arbitration," *ACM Transactions on Design Automation of Electronic Systems*, vol. 20, no. 4, pp. 1–22, Sep. 2015.

[107] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim, "A detailed and flexible cycle-accurate network-on-chip simulator," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 86–96.

[108] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2015, pp. 162–163.

[109] L. Jain, B. M. Al-Hashimi, M. S. Gaur, V. Laxmi, and A. Narayanan, "Nirgam: a simulator for noc interconnect routing and application modeling," in *International Conference and Exhibition on Design, Automation and Test in Europe (DATE)*, 2007, p. 1620.

[110] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, and et al., "The Gem5 simulator," *SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, Aug 2011.

[111] Imperas, "Open virtual platforms (ovp)," URL: http://www.ovpworld.org/technology_ovpsim, visited on 2016-06-28.

[112] J. A. Stankovic and K. Ramamritham, "What is predictability for real-time systems?" *Journal on Real-Time Systems*, vol. 2, no. 4, pp. 247–254, Nov 1990.

[113] G. C. Buttazzo, *Hard Real-Time Computing Systems*, ser. Real-Time Systems Series. Springer US, 2011, vol. 24.

[114] P. A. Burns and P. A. Wellings, *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*, 4th ed. Addison Wesley, May 2009.

[115] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, Jan 1973.

[116] S. Altmeyer, R. I. Davis, and C. Maiza, "Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," *Real-Time Systems Journal*, vol. 48, no. 5, pp. 499–526, Jun 2012.

[117] G. C. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems. a survey," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 3–15, Feb 2013.

[118] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys*, vol. 43, no. 4, pp. 35:1–35:44, 2011.

[119] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," Technical Report - Department of Computer Science, University of York, UK, Tech. Rep., 1991.

[120] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamritham, J. Stankovic, and L. Strigini, "The meaning and role of value in scheduling flexible real-time systems," *Journal of Systems Architecture*, vol. 46, no. 4, pp. 305–325, Jan 2000.

[121] P. Holman and J. H. Anderson, "Adapting pfair scheduling for symmetric multiprocessors," *Journal of Embedded Computing*, vol. 1, no. 4, pp. 543–564, 2005.

[122] N. W. Fisher, "The multiprocessor real-time scheduling of general task systems," Ph.D. dissertation, University of North Carolina, USA, 2007.

[123] I. Shin, A. Easwaran, and I. Lee, "Hierarchical scheduling framework for virtual clustering of multiprocessors," in *Euromicro Conference on Real-Time Systems ECRTS*, 2008, pp. 181–190.

[124] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, and et al., "The worst-case execution-time problem;overview of methods and survey of tools," *ACM Transactions on Embedded Computer Systems*, vol. 7, no. 3, pp. 36:1–36:53, May 2008.

[125] G. C. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo, *Soft Real-Time Systems: Predictability vs. Efficiency*, 1st ed. Springer US, Jul 2006.

[126] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, pp. 284–292, 1993.

[127] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Real-Time Systems Symposium RTSS*, 2007, pp. 149–160.

[128] Z. Shi, A. Burns, and L. S. Indrusiak, "Schedulability analysis for real time on-chip communication with wormhole switching:," *International Journal of Embedded and Real-Time Communication Systems*, vol. 1, no. 2, pp. 1–22, 2010.

[129] H. Kashif, S. Gholamian, R. Pellizzoni, H. D. Patel, and S. Fischmeister, "ORTAP: An offset-based response time analysis for a pipelined communication resource model," in *Real-Time and Embedded Technology and Applications Symposium, IEEE*, 2013, pp. 247–258.

[130] R. Zimmermann and K. Fu, "Comprehensive statistical admission control for streaming media servers," in *International Conference on Multimedia (MULTIMEDIA)*. ACM, 2003, pp. 75–85.

[131] J. Dengler, C. Bernhardt, and E. Biersack, *Deterministic admission control strategies in video servers with variable bit rate streams*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1996.

[132] T.-W. Kuo, L.-P. Chang, Y.-H. Liu, and K.-J. Lin, "Efficient online schedulability tests for real-time systems," *IEEE Transactions on Software Engineering*, vol. 29, no. 8, pp. 734–751, Aug 2003.

[133] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Springer Science & Business Media, Dec 2012.

[134] P. Dziurzanski, A. K. Singh, and L. S. Indrusiak, "Feedback-based admission control for hard real-time task allocation under dynamic workload on many-core systems," in *Conference on Architecture of Computing Systems (ARCS)*. Springer International Publishing, 2016, pp. 157–169.

[135] C. Lu, X. Wang, and X. Koutsoukos, "Feedback utilization control in distributed real-time systems with end-to-end tasks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 6, pp. 550–561, Jun 2005.

[136] M. Di Natale and J. Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Real-Time Systems Symposium (RTSS)*, Dec 1994, pp. 216–227.

[137] B. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," *IEEE Trans. on Parallel and Distributed Systems*, vol. 8, no. 12, pp. 1268–1274, Dec. 1997.

[138] H. A. Ghazzawi, "Feedback admission control for workflow management systems," Ph.D. dissertation, University of York, UK, 2015.

[139] H. Vin, P. Goyal, and A. Goyal, "A statistical admission control algorithm for multimedia servers," in *ACM International Conference on Multimedia (MULTIMEDIA)*. ACM, 1994, pp. 33–40.

[140] C. J. Hamann, M. Roitzsch, L. Reuther, J. Wolter, and H. Hartig, "Probabilistic admission control to govern real-time systems under overload," in *Euromicro Conference on Real-Time Systems ECRTS*, 2007, pp. 211–222.

[141] M. Ditze, P. Altenbernd, and C. Loeser, "Improving resource utilization for MPEG decoding in embedded end-devices," in *Australasian Conference on Computer Science - Vol. 26*, ser. ACSC '04. Australian Computer Society, Inc., 2004, pp. 133–142.

[142] J. Silvestre-Blanes, L. Almeida, R. Marau, and P. Pedreiras, "Online qos management for multimedia real-time transmission in industrial networks," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 3, pp. 1061–1071, Mar 2011.

[143] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: operating system support for multimedia applications," in *International Conference on Multimedia Computing and Systems*, 1994, pp. 90–99.

[144] J. Stankovic, C. Lu, S. Son, and G. Tao, "The case for feedback control real-time scheduling," in *Euromicro Conference on Real-Time Systems*, 1999, pp. 11–20.

[145] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao, "Feedback control real-time scheduling: Framework, modeling, and algorithms*," *Real-Time Systems*, vol. 23, no. 1-2, pp. 85–126, Jul 2002.

[146] M. U. K. Khan, M. Shafique, and J. Henkel, "Hierarchical power budgeting for dark silicon chips," in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2015, pp. 213–218.

[147] D. Isovic and G. Fohler, "Quality aware MPEG-2 stream adaptation in resource constrained systems," in *Euromicro Conf. on Real-Time Systems ECRTS*, 2004, pp. 23–32.

[148] N. H. Mastronarde and M. v. d. Schaar, "A queuing-theoretic approach to task scheduling and processor selection for video-decoding applications," *IEEE Transactions on Multimedia*, vol. 9, no. 7, pp. 1493–1507, Nov 2007.

[149] C. Blanch, R. Baert, P. Coene, M. D'Hondt, Z. Ma, and R. Wuyts, "Runtime scheduling for video decoding on heterogeneous architectures." in *International Conference on Real-Time Networks and Systems (RTNS)*, 2011, pp. 195–204.

[150] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE Transactions on Computers*, vol. 44, no. 12, pp. 1443–1451, Dec 1995.

[151] M. A. Bamakhrama and T. P. Stefanov, "On the hard-real-time scheduling of embedded streaming applications," *Design Automation for Embedded Systems*, vol. 17, pp. 221–249, 2013.

[152] Y.-H. Wei, C.-Y. Yang, T.-W. Kuo, S.-H. Hung, and Y.-H. Chu, "Energy-efficient real-time scheduling of multimedia tasks on multi-core processors," in *ACM Symposium on Applied Computing (SAC)*. ACM, 2010, pp. 258–262.

[153] V. Nollet, T. Marescaux, P. Avasare, D. Verkest, and J.-Y. Mignolet, "Centralized run-time resource management in a network-on-chip containing reconfigurable hardware tiles," in *International Conference and Exhibition on Design, Automation and Test in Europe (DATE)*, 2005, pp. 234–239.

[154] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang, "Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms," *Journal of Systems Architecture*, vol. 56, pp. 242–255, 2010.

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

[156] A. Bartzas, P. Bellasi, I. Anagnostopoulos, C. Silvano, W. Fornaciari, D. Soudris, D. Melpignano, and C. Ykman-Couvreur, "Runtime resource management techniques for many-core architectures: The 2PARMA approach," in *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA11), Invited Paper*, 2011.

[157] 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 Symposium on VLSI*, 2013, pp. 153–158.

[158] M. Ruaro, G. Madalozzo, and F. G. Moraes, "A hierarchical lst-based task scheduler for noc-based mpsocs with slack-time monitoring support," in *IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2015, pp. 308–311.

[159] I. Anagnostopoulos, V. Tsoutsouras, A. Bartzas, and D. Soudris, "Distributed run-time resource management for malleable applications on many-core platforms," in *Design Automation Conference (DAC)*, 2013, pp. 168:1–168:6.

[160] A. Rodriguez, A. Gonzalez, and M. P. Malumbres, "Hierarchical parallelization of an h.264/avc video encoder," in *International Symposium on Parallel Computing in Electrical Engineering (PARELEC)*, 2006, pp. 363–368.

[161] P. Zipf, G. Sassatelli, N. Utlu, N. Saint-Jean, P. Benoit, and M. Glesner, "A decentralised task mapping approach for homogeneous multiprocessor network-on-chips," *International Journal on Reconfigurable Computing*, vol. 2009, pp. 3:1–3:14, Jan 2009.

[162] J. Heisswolf, A. Zaib, A. Weichslgartner, M. Karle, M. Singh, T. Wild, J. Teich, A. Herkersdorf, and J. Becker, "The Invasive Network on Chip - a multi-objective many-core communication infrastructure," in *International Conference on Architecture of Computing Systems (ARCS)*, 2014, pp. 1–8.

[163] A. Weichslgartner, S. Wildermann, and J. Teich, "Dynamic decentralized mapping of tree-structured applications on NoC architectures," in *IEEE/ACM International Symposium on Networks on Chip (NoCS)*, 2011, pp. 201–208.

[164] U. Brinkschulte, M. Pacher, and A. Von Renteln, *Towards an artificial hormone system for self-organizing real-time task allocation.* Springer, 2007.

[165] B. Betting, U. Brinkschulte, and M. Pacher, "Evaluation and superiority analysis of a decentralized task control mechanism for dependable real-time SoC architectures," in *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), IEEE International Symposium on.* IEEE, 2013, pp. 1–8.

[166] 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, pp. 353–360.

[167] M. Bakhouya, "Towards a bio-inspired architecture for autonomic network-on-chip," in *High Performance Computing and Simulation (HPCS), International Conference on*, 2010, pp. 491–497.

[168] P. K. Sahu and S. Chattopadhyay, "A survey on application mapping strategies for network-on-chip design," *Journal of Systems Architecture*, vol. 59, no. 1, pp. 60–76, Jan 2013.

[169] T. Maqsood, S. Ali, S. U. Malik, and S. A. Madani, "Dynamic task mapping for network-on-chip based systems," *Journal of Systems Architecture*, vol. 61, no. 7, pp. 293–306, Aug 2015.

[170] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, Apr 1979.

[171] Y. Yi, W. Han, X. Zhao, A. T. Erdogan, and T. Arslan, "An ILP formulation for task mapping and scheduling on multi-core architectures," in *International Conference and Exhibition on Design, Automation and Test in Europe (DATE)*, 2009, pp. 33–38.

[172] M. Sayuti and L. Indrusiak, "Real-time low-power task mapping in Networks-on-Chip," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2013, pp. 14–19.

[173] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems," in *International Conference and Exhibition on Design, Automation and Test in Europe (DATE)*, 2002, p. 514521.

[174] J. Hu and R. Marculescu, "Energy and performance-aware mapping for regular NoC architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 4, pp. 551–562, Apr 2005.

[175] U. Brinkschulte, A. von Renteln, and M. Pacher, "Measuring the quality of an artificial hormone system based task mapping," in *International Conference on Autonomic Computing and Communication Systems*, 2008, pp. 32:1–32:8.

[176] C.-L. Chou and R. Marculescu, "User-aware dynamic task allocation in networks-on-chip," in *International Conference and Exhibition on Design, Automation and Test in Europe (DATE)*, Mar 2008, pp. 1232–1237.

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

[178] M.-H. Haghbayan, A. Kanduri, A.-M. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen, "MapPro: Proactive runtime mapping for dynamic workloads by quantifying ripple effect of applications on networks-on-chip," in *International Symposium on Networks-on-Chip NoCs*. ACM, 2015, pp. 26:1–26:8.

[179] I. Tcarenko, M. Fattah, P. Liljeberg, J. Plosila, and H. Tenhunen, "Multi rectangle modeling approach for application mapping on a many-core system," in *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2014, pp. 452–457.

[180] M. Mandelli, L. Ost, E. Carara, G. Guindani, T. Gouvea, G. Medeiros, and F. Moraes, "Energy-aware dynamic task mapping for NoC-based MPSoCs," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2011, pp. 1676–1679.

[181] M. Palesi, G. Ascia, F. Fazzino, and V. Catania, "Data encoding schemes in networks on chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 5, pp. 774–786, May 2011.

[182] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems," in *Design Automation Conference (DAC)*, 2014, p. 170:1170:6.

[183] H. I. A. A. Ali, L. M. Pinho, and B. Akesson, "Critical-path-first based allocation of real-time streaming applications on 2D mesh-type multi-cores." in *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2013, pp. 201–208.

[184] FFMPEG, "FFmpeg:open source codec project," URL: https://ffmpeg.org/, visited on 2016-01-16.

[185] J. Jeong, J. Choi, and S. Ha, "Parallelization and performance prediction for HEVC UHD real-time software decoding," in *IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia)*, 2014, pp. 40–49.

[186] R. K. Pal, I. Shanaya, K. Paul, and S. Prasad, "Dynamic core allocation for energy efficient video decoding in homogeneous and heterogeneous multicore architectures," *Future Generation Computer Systems*, vol. 56, pp. 247–261, Mar 2016.

[187] M. Shafique, M. U. K. Khan, and J. Henkel, "Power efficient and workload balanced tiling for parallelized high efficiency video coding," in *IEEE International Conference on Image Processing (ICIP)*, 2014, pp. 1253–1257.

[188] A. K. Singh, A. Kumar, and T. Srikanthan, "A hybrid strategy for mapping multiple throughput-constrained applications on MPSoCs," in *International conference on Compilers, architectures and synthesis for embedded systems (CASES)*. ACM, 2011, pp. 175–184.

[189] A. Weichslgartner, D. Gangadharan, S. Wildermann, M. Gla, and J. Teich, "DAARM: Design-time application analysis and run-time mapping for predictable execution in many-core systems," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2014, pp. 1–10.

[190] Y. Ge, P. Malani, and Q. Qiu, "Distributed task migration for thermal management in many-core systems," in *Design Automation Conference (DAC)*. ACM, 2010, pp. 579–584.

[191] O. Derin, D. Kabakci, and L. Fiorin, "Online task remapping strategies for fault-tolerant Network-on-Chip multiprocessors," in *IEEE International Symposium on Networks-on-Chip (NOCS)*, 2011, pp. 129–136.

[192] S. Holmbacka, M. Fattah, W. Lund, A.-M. Rahmani, S. Lafond, and J. Lilius, "A task migration mechanism for distributed many-core operating systems," *The Journal of Supercomputing*, vol. 68, no. 3, pp. 1141–1162, Mar 2014.

[193] J. Harbin and L. Indrusiak, "Dynamic task remapping for power and latency performance improvement in priority-based non-preemptive Networks On Chip," in *International*

Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2013, pp. 1–7.

[194] 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 Systems*, no. 1, pp. 1–15, 2007.

[195] F. G. Moraes, G. A. Madalozzo, G. M. Castilhos, and E. A. Carara, "Proposal and evaluation of a task migration protocol for noc-based mpsocs," in *IEEE International Symposium on Circuits and Systems*, 2012, pp. 644–647.

[196] B. Betting and U. Brinkschulte, "Analyzing the overhead of self-optimization through task migration within a decentralized task control mechanism for dependable system-on-chip architectures," in *IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, 2014, pp. 84–91.

[197] H. Tajik, H. Homayoun, and N. Dutt, "VAWOM: Temperature and process variation aware wearout management in 3D multicore architecture," in *Design Automation Conference (DAC)*. ACM, 2013, pp. 178:1–178:8.

[198] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed. Addison Wesley, Apr 2011.

[199] Y. Ma, M. Sayuti, and L. Indrusiak, "Inexact end-to-end response time analysis as fitness function in search-based task allocation heuristics for hard real-time network-on-chips," in *International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, 2014, pp. 1–8.

[200] Y. Ma and L. S. Indrusiak, "Hardware-accelerated response time analysis for priority-preemptive networks-on-chip," in *International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2015, pp. 1–8.

[201] I. Caliskanelli and L. Indrusiak, "Search-Based Parameter Tuning on Application-Level Load Balancing for Distributed Embedded Systems," in *IEEE International Conference on High Perferformance Computing and Communication on Embedded and Ubiquitous Computing (HPCC_EUC)*, 2013, pp. 2050–2057.

[202] I. Caliskanelli, "A bio-inspired load balancing technique for wireless sensor networks," Ph.D. dissertation, University of York, UK, Mar 2014.

[203] Multicoreware, "x265 HEVC encoder/H.265 video codec," URL: http://x265.org/, 2015, [Online; accessed 26-October-2015].

[204] M. Naccari, R. Weerakkody, J. Funnell, and M. Mrak, "Enabling ultra high definition television services with the HEVC standard: The thira project," in *IEEE International Conference on Multimedia and Expo (ICMEW)*, 2015, pp. 1–4.

[205] W. Hamidouche, M. Raulet, and O. Deforges, "Real time shvc decoder: Implementation and complexity analysis," in *IEEE International Conference on Image Processing (ICIP)*, 2014, pp. 2125–2129.

[206] G. S. Mudholkar, D. K. Srivastava, and M. Freimer, "The exponentiated Weibull family: a reanalysis of the bus-motor-failure data," *Technometrics*, vol. 37, pp. 436–445, 1995.

[207] I.-K. Kim, J. Min, T. Lee, W.-J. Han, and J. Park, "Block partitioning structure in the HEVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1697–1706, 2012.

[208] Intel, "Intel VTune Amplifier XE - Performance Profiler," URL: https://software.intel.com/en-us/intel-vtune-amplifier-xe, 2016, visited on 2016-08-29.

[209] S. Bcker and Z. Liptk, *The Money Changing Problem Revisited: Computing the Frobenius Number in Time O(ka 1)*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Aug 2005.

[210] Y. Kun, Z. Chun, D. Guoze, X. Jiangxiang, and W. Zhihua, "A hardware-software co-design for H.264/AVC decoder," in *Asian Solid-State Circuits Conference ASSCC*, Nov 2006, pp. 119–122.

[211] M. Abeydeera, M. Karunaratne, G. Karunaratne, K. D. Silva, and A. Pasqual, "4K real-time HEVC decoder on an FPGA," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 1, pp. 236–249, Jan 2016.