Early Design Space Exploration of Hard Real-Time Embedded Networks-on-Chip

M. Norazizi Sham Mohd Sayuti

PhD

University of York
Computer Science

December 2014

To

My Mother

Abstract

Networks-On-Chip (NoC) is seen as a solution for addressing the limitation of the current bus-based communication in embedded systems. Some of these systems are designed for executing hard real-time services. In such systems, the services have to deliver output within strict timing constraints since the lateness in output delivery could cause severe consequences to human life. Task mapping is a crucial step for integrating an application and a hardware platform during system design. Existing schedulability analyses are available to evaluate the hard real-time performance of task mapping, but exploring the vast number of task mappings at the early design stage can be challenging due to several issues. These issues are caused by the influence of other design parameters on the hard real-time performance produced by task mapping, the existence of conflicting design objectives with the hard real-time system constraints, the restriction of the current hard real-time evaluation functions for searching alternative task mappings and the enormous evaluation of population-based search heuristics in the current task mapping techniques. This thesis proposes several design space exploration techniques to address these issues. The first technique is proposed for addressing the problem of optimising multiple design parameters while keeping all tasks and messages in the system fully schedulable. The second technique addresses the conflicting objectives problem using a multi-objective optimisation approach. The third technique yields a new metric that is useful for improving task mappings with unschedulable tasks and messages. Finally, the last technique is a new mapping algorithm for constructing a feasible task mapping rather than have to evaluate a population of task mappings to achieve the same objective.

Contents

List of Figures

List of Tables

Acknowledgements

In the name of Allah, most Gracious, most Compassionate. All praise be to Allah, with your blessings and mercy I managed to complete this thesis.

I would like to thank my supervisor, Dr. Leandro Soares Indrusiak, for his advice, guidance, encouragement and supports throughout the years of research until the submission of this thesis. His enthusiasm and precise view on research has made a deep impression on me and I have learned and experienced so much from him. I would also like to thank my assessor, Dr. Neil Audsley, for his advice on my work.

In addition, thanks to all the rest of the colleagues in the Real-Time Research Group for giving me unforgettable helps and for sharing fruitful discussions with me.

A special thanks to the person who without her I would not have finished this thesis, the one who supported me as much as she could, my wife, Suhailah Abu Bakar. My son Muhammad Ammar Safwan and my daughter Nurul Anis Syafiqah, your laughs and smiles give me the strength to pursue this study to the end.

I would like to express my sincere gratitude to my father, Mohd Sayuti Mardi, for his permanent love and Dua.

Finally, I would like to acknowledge the Universiti Sains Islam Malaysia and the Ministry of Education Malaysia for providing me with the financial supports to pursue this study.

Thank you.

Declaration

Herewith I declare the research as described in this thesis is my original work undertaking between October 2011 and December 2014 at University of York, United Kingdom. This research was undertaken under the supervision of my supervisor, Dr. Leandro Soares Indrusiak. Some parts of this thesis have been published in several conference papers.

- Chapter 3:
  M. N. S. Mohd Sayuti and L. S. Indrusiak, Simultaneous optimisation of task mapping and priority assignment for real-time embedded networks-on-chip, in 23rd Euromicro Conference on Parallel, Distributed and NetworkBased Processing, IEEE, 2014.
- Chapter 4:
  M. N. S. M. Sayuti and L. S. Indrusiak, Real-time low-power task mapping in networks-on-chip, in IEEE Computer Society Annual Symposium on VLSI, 2013.
  M. N. S. M. Sayuti, L. S. Indrusiak, and A. Garcia-Ortiz, An optimisation algorithm for minimising energy dissipation in noc-based hard real-time embedded systems, in Proceedings of the 21st International Conference on Real-Time Networks and Systems, 2013.

Chapter 1

Introduction

1.1 Overview

Modern embedded systems have advanced rapidly in recent decades, influencing every aspect of our daily life including communication, transportation and manufacturing. The systems are becoming more complex due to the demand from market forces for increasingly sophisticated features. One of the key technologies that drives the designs of such systems is System-on-Chip (SoC) [1], whereby some or all functionalities of a complete system can be integrated into single chips. This technology is recognised from the integration of several computation and data processing elements such as Intellectual Properties (IP blocks): a reusable pre-designed electronic blocks of logic circuits. Different IP blocks in the system are designed to perform specific functions and may require specific data, or to compute data that are required by other IP blocks. An on-chip communication infrastructure is therefore needed to support the delivery of data between IP blocks in the system.

Conventional SoC designs [2, 3] rely on Point-to-Point (P2P), for example MPEG2 Encoder [4] or bus communication such as the Philips Nexperia Digital Video Platform, as the network infrastructure that connects multiple IP blocks in the system. Generally, P2P communication supports data exchange between IP blocks through dedicated channels, whereas bus communication introduces a single or multiple channels that are shared between IP blocks. With bus com-

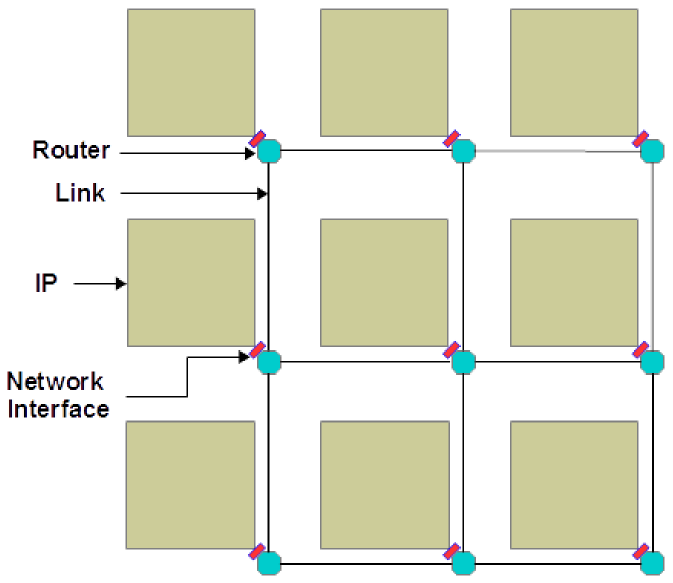


Figure 1.1: High-level NoC architecture

munication, an IP sends data at a time to one or more receiving IP blocks, and the others wait until the bus is free. However, each of them has its own disadvantages as the number of IP blocks increases to support complex functionalities. For example, P2P communication suffers from the underutilisation of wires when the connected IP blocks are idle and the increasing number of wires makes the problem even worse [5]. Bus communication has non-scalable architecture [2] and a network bottleneck could happen from congestion as more IP blocks are added to the same bus and share its bandwidth.

Some attempts have been made to provide an efficient and standardised communication infrastructure for connecting multiple computing resources on SoC. A new network paradigm with better scalability known as Networks-on-Chip (NoC) [6] has been proposed to overcome the scalability problems of P2P and bus communication architectures. Inspired by the success of computer networks (for example, local and wide area networks), NoC designs inherit some of the characteristics from the networks as shown in the high-level view of a typical NoC architecture in Figure 1.1. This is an example of a 3x3 Mesh NoC, which comprises three types of network component: router, physical link and network interface.

The details of each component are described in Chapter 2.1; this current paragraph gives only an overview of how the network operates. As can be seen in Figure 1.1, a total of nine IP blocks are connected indirectly to the physical links by routers. IP blocks have different forms, for example a Central Processing Unit (CPU), peripheral devices or memory controllers, which are responsible for processing and storing computational data. Network interfaces, which glue together the IP blocks and the routers, provide communication services to the IP blocks through the encapsulation of the network's low-level functions. Connection to the network allows communication to happen between the IP blocks and enabling them to send and receive data in the form of *messages*: the flows of data in the network. In the architecture of a NoC, routers are responsible for the transmission of messages, such as by determining the message routes and how the messages travel through them. Physical links, which connect the routers, channel the messages from a source to a destination. Although the number of links increases to support more IP blocks, the links' bandwidth is shared among messages. For example, when congestion occurs, the blocked messages can be re-routed through other routes or some messages can be redirected to idle links. With such routing mechanisms, links can be utilised more efficiently, and at the same time NoCs provide the provision of parallel communication.

1.1.1 Real-Time Service

NoC architecture such as that shown in Figure 1.1 connects multiple IP cores and may enable different services to run concurrently in the system. Among these services there exists a type in which its correctness is determined not only by its output but also by the time at which the output is available [7]. Any lateness of the service is intolerable because severe consequences to human life could result. For example, brakes will not be activated in time or actuators will use stale data. This service is known as a *hard real-time* service and it is common in automotive control and safety-critical systems [8].

The main characteristic of the service comes from the requirement to produce its output within strict timing constraints. Its correctness relies on whether the tasks and messages which run in the system to deliver the service can respond within the set timing constraints in any scenario. A timing constraint can be defined as the latest point in time (or a *deadline*) at which a task must execute or a message must arrive to produce the output. Before the computational data is processed by the IP cores, several messages may be received and sent between more than one task to transfer the data. Messages are sent from a sender to a recipient and the time interval of the transmission is called the network latency. The execution time of a task to compute and process the data summed with the latency yields the *end-to-end* response time of the task. From the difference between the response time *upper bound* of a task and its deadline, the task is *schedulable* if its deadline is not missed, otherwise it is *unschedulable*. In fact, the execution of a task depends on the arrival of data, and thus a reliable data exchange between IP cores is an integral part in the delivery of the service.

For a hard real-time system, the *predictability* of tasks' and messages' behaviours is a fundamental requirement. In the NoC platform, computation and communication resources are shared between tasks and messages, and the access to the resources must be controlled to ensure their behaviours are predictable. One way to control their access to the resources, a *priority pre-emptive* scheduling policy is usually employed. With this policy, each task and message is assigned with a *priority level*. Based on the priority levels of all tasks and messages, the order of access to the shared resources can be controlled by pre-empting some tasks and messages, especially those with low priority levels. This pre-emptive policy, however, causes *interference* to low-priority tasks and messages, consequently delaying their end-to-end response time. In the worst-case scenario, when delays can become enormous due to the pre-emption, their deadlines could be missed. If this situation occurs, the system is deemed unschedulable.

1.1.2 Task Mapping Process

Task mapping has been identified as a critical part of embedded system design [9]. At the system level, it is a necessary design step prior to the evaluation of a complete system. As depicted in Figure 1.2, the mapping process integrates an application and a hardware architecture to create a complete system, which is then followed by a performance analysis. Based on the feedback from the

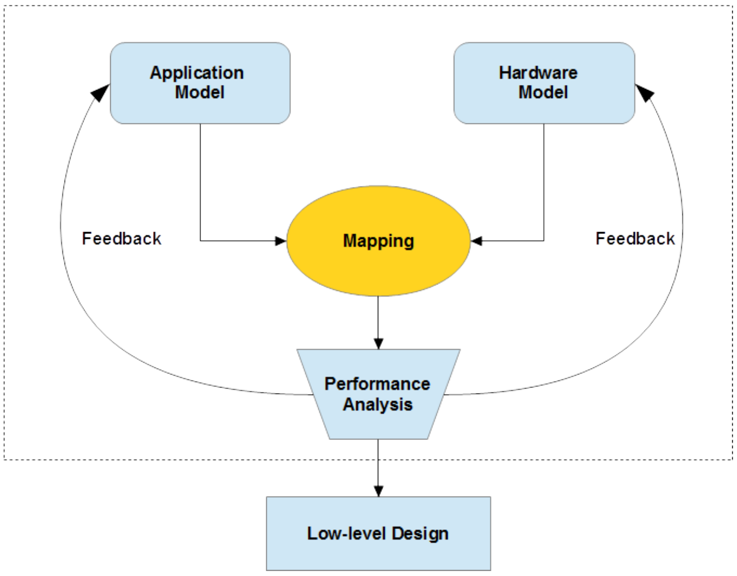
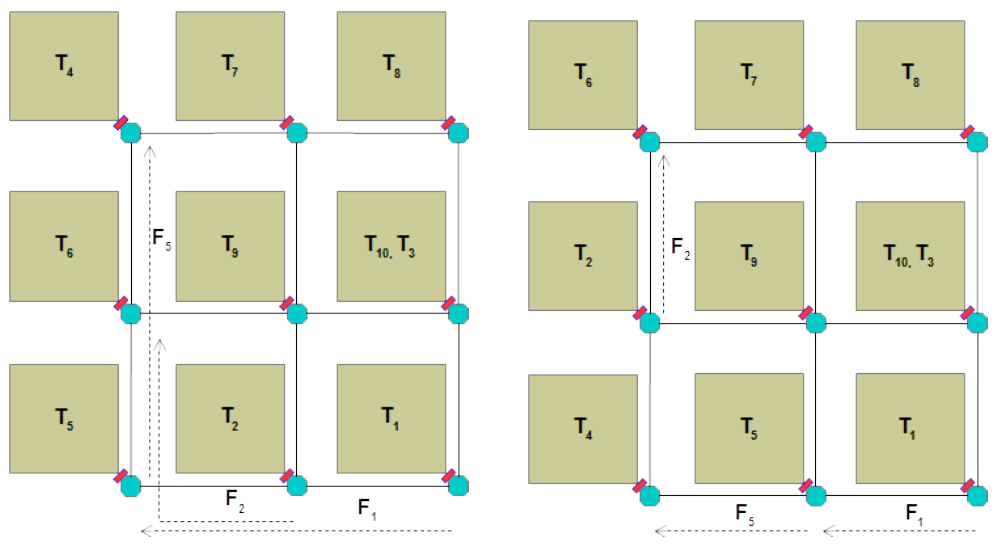


Figure 1.2: Task mapping process

performance analysis, appropriate modifications can be made to the application, to the hardware architecture or to the task mapping itself. Several iterations of the same process maybe needed before the system performance meets the specified design objectives. Once designs with the right performance are found, those designs are further refined at the low-level design.

A hard real-time application model contains a set of tasks and the task mapping process could allocate them differently on the NoC platform (see Figure 1.1). As shown in Figure 1.3, each IP or processing core can have one or more tasks to execute depending on how the task mapping process allocates the tasks. The end-to-end response time of the tasks could be affected depending on their locations and the interference experienced by them in the shared resources. If some computation or communication resources have more tasks or messages to execute, the low-priority tasks and messages will receive high interference from their high priority counterparts. Figure 1.3a shows a possible task mapping output from the mapping process. In this example, tasks T_1, T_2, and T_5 each send a message to its respective recipient at different processing cores. Message F_1 originating from task T_1 travels on the same link as message F_2. The latter message also shares the same link with message F_5 sent by task T_5. Assuming that the priority order of



(a) Before (b) After

Figure 1.3: Interference before and after changing task mapping

these tasks is $T_1 > T_2 > T_5$, and the messages follow the same order as the transmitting tasks, then message F_5 will receive interference directly from message F_2. At the same time, message F_2 receives interference from message F_1. This further delays the response time of message F_5 because it has to wait for message F_2 to completely arrive at its destination and to release the link. Based on the same application and NoC platform, a new task mapping is created as depicted in Figure 1.3b. Based on the new task mapping, it is possible to avoid interference between messages by mapping these tasks at different locations. This is a simple example of how different task mappings could impose varying influences on the schedulability of tasks and messages in the system.

One of the requirements of a hard real-time system is that a task or a message is deemed schedulable when its deadline is not missed in any scenario. Fully schedulable tasks and messages make the system predictable, and thus a task mapping that meets the objective of creating a fully schedulable system is most desirable. The mapping process could produce many task mappings and might use the number of schedulable tasks and messages by which to assess them. Figure 1.4 shows how each task mapping points to different areas in the graph based on the metric.

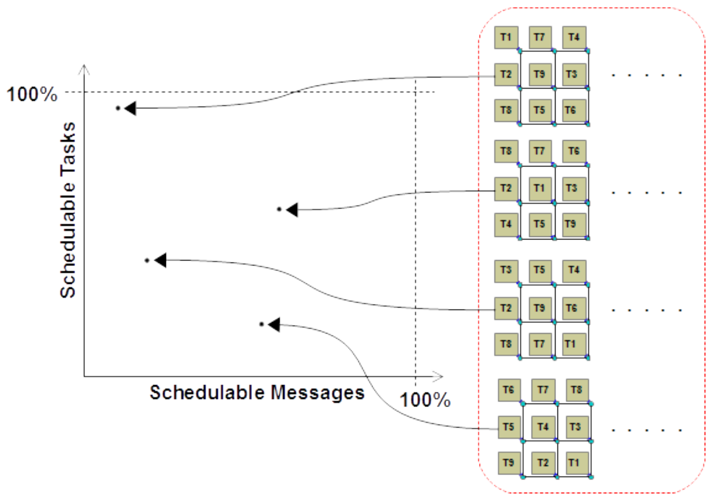


Figure 1.4: Different task mappings produce various performances

1.2 Motivation and Goal

Different task mappings produce varying performances as shown in Figure 1.4. It becomes a necessity to evaluate as many task mappings as possible to increase the probability of finding the best selection of designs for further refinement at the low level of abstraction. However, the number of task mappings grows exponentially with the size of the task set and the size of platform (that is, the number of IP cores). Within a limited time frame, exhaustive searching is prohibitive [10] and it is unlikely a schedulable task mapping will be found by arbitrarily mapping the tasks on the platform.

For hard real-time systems, the end-to-end response time of all tasks, which includes the latency of the sent messages, must not exceed the timing constraints of the systems. In order to ensure the systems are predictable, a priority pre-emptive scheduling policy is used to schedule tasks and messages in a way that gives high-priority tasks and messages guaranteed access to the computation and communication resources. However, if task mapping is inefficient, the interference suffered by low-priority tasks and messages can become enormous, affecting their end-to-end response times. For achieving a fully schedulable system, it is desirable to find a task mapping that allows all tasks and messages meet their deadline. How to lessen the interference suffered by low priority tasks and messages while enabling the high-priority tasks and messages to have guaranteed access to the resources is the key issue. It has been reported in [11] that a schedulable task mapping cannot be found when the platform contains limited computation and communication resources. Some researchers [12] have focused on priority scheduling but did not address the task mapping problem, and thus limited the ways in which task mapping can be optimised with their approaches. Others considered both task mapping and priority scheduling but their approaches were limited to systems with bus communication [13, 14, 15].

In practice, conflicting design objectives normally exist, and any change to the task mapping must consider all the objectives to reflect the true performance. NoC architecture is highly configurable but in a small silicon area its designs are constrained in some aspects such as power consumption [16]. In addition, for NoC-based embedded systems with hard real-time requirements, meeting the timing constraints of the requirements is essential since any lateness in the response times might cause one or more tasks miss their deadline. However, changing the task mapping solely to achieve low power consumption hides the impacts on the real-time performance and *vice versa*. For example, allocating tasks near to each other reduces the power dissipated by NoC, but can increase the contention for the network resources and this might lead to enormous delays exceeding the deadlines. Conversely, if those tasks are mapped far from each other, this might further increase power consumption due to the involvement of many network components (such as routers and links) for transmitting the messages. Therefore, the goal is to find a schedulable task mapping for the system while considering the other objective at the same time. The issue here is how to achieve a good trade-off between more than one conflicting objectives. Although multiple objectives in searching for task mapping have been considered before, one of these approaches caused enormous evaluation time [17] and others [18, 19, 20] did not address finding a task mapping for the hard real-time systems. A few have proposed finding task mapping for this kind of system, but the techniques lacked insight into power consumption[11, 21].

In the analysis of hard real-time systems, a quantitative schedulability metric such as the total number of unschedulable tasks and messages is a convenient fitness value for evaluating the feasibility of task mapping. A task mapping is assumed as unschedulable if the metric yields one or more unschedulable tasks and messages, otherwise it is assumed as schedulable. Applying this metric in task mapping optimisation helps minimising the number of unschedulable tasks and messages and converging to a fully schedulable task mapping. However, if a schedulable task mapping cannot be found by the optimisation algorithm, no further information can be applied by the algorithm to facilitate the search of the schedulable task mapping with a given platform. It has been reported that in some cases finding the feasible task mapping was unsuccessful[11, 21] with the existing schedulability metric. Therefore, it is useful if this metric can be improved to provide additional information for the algorithm to search for the schedulable task mapping from the design space. The key issue is how the schedulability metric can be improved to make the unschedulable task mapping schedulable.

The search-based optimisation approach such as genetic algorithms has the potential to explore and evaluate many task mappings in a single run to identify the best of them. However, the number of evaluation that has to be performed increases with the size of population used during optimisation. Depending on the runtime complexity of the evaluation function, the optimisation might take a significant amount of time to find a schedulable task mapping. Instead of depending on a large population to find a schedulable task mapping, an alternative algorithm that consume less optimisation runtime but could find a task mapping that is nearly as good as the genetic algorithms is desirable. One of the advantages in the reduction of optimisation runtime is that it could help system designers to reduce the amount of time to explore the design space. However, whether the new algorithm is capable to find a schedulable task mapping as good as the genetic algorithm in less time than the latter algorithm is the key issue.

The previous studies are discussed fully later in the thesis, but all these issues contributed to motivating our research works. This thesis addresses the issues, with the main goal is to find a feasible task mapping that can make a NoC based hard real-time system schedulable. The following proposition determined the central focus of the research works.

A schedulable task mapping can be found for NoC-based hard real-time embedded system

1.3 Summary of Contributions

The thesis proposition set out above will be achieved by a series of research works as follows:

1. A simultaneous optimisation approach, involving task mapping and priority assignment, is proposed to overcome the problem of meeting the timing constraints of low-priority tasks and messages due to the high interference experienced by them. The notion of changing the priority to reduce the interference of low-priority tasks and messages enables the approach to effectively find the feasible task mapping for the system better than the previous optimisation approaches that rely on the static priority pre-emptive scheduling policy and random priority assignment. In addition, the proposed approach facilitates the optimisation algorithm to converge faster than the previous optimisation algorithms.

2. Finding a task mapping in the presence of more than one conflicting objective requires consideration of the trade-off between the objectives. Single-objective optimisation is ineffective for the purpose since it focuses solely on one objective but ignores the others, causing the impact of the latter objectives to be hidden from the system design. Aggregating all objectives into one objective will work if conflicts do not exist between the objectives, but forcing this approach on the conflicting objectives will introduce bias to the solutions. A multi-objective optimisation algorithm is used to address the schedulability and NoC power dissipation optimisation problems. It finds a schedulable task mapping as effectively as the single-objective optimisation algorithm, but with lower power dissipation than the task mapping of the latter algorithm.

3. A quantitative schedulability metric, such as the total number of unschedulable tasks and messages, can be used to evaluate the feasibility of task mapping. However, this metric has a limitation when the system becomes unschedulable based on a given task mapping because it does not provide further information that can be used to improve the task mapping. To overcome this problem, a new fitness function is proposed to produce a

new metric called the *breakdown frequency* as the fitness value of every task mapping. The breakdown frequency is the minimal frequency that could make all tasks and messages schedulable in the system without changing the task mapping. With the new fitness function, the optimisation algorithm has the means to improve an unschedulable task mapping to make it schedulable. Another benefit of using the approach as part of the optimisation algorithm is that it enables further minimisation of the system's operating frequency as the optimisation progresses over time.

4. A GA-based optimisation algorithm is a good means of finding schedulable task mappings. It depends on the population size to provide diversity which is essential for exploring many solutions at the same time in the population. However, its benefit comes at the expense of increased evaluation time. As an alternative to the optimisation algorithm, a constructive mapping algorithm is proposed to construct a task mapping rather than to explore many task mappings simultaneously. Based on particular attributes such as the utilisation of task and message, task schedulability and the number of outgoing messages, the proposed algorithm could provide a schedulable task mapping nearly as effective as the previous algorithm, but with reduced evaluation time.

1.4 Thesis Outline

The remaining six chapters of this thesis are organised in the following structure. Chapter 2 reviews the latest techniques related to the early design space exploration of NoC-based hard real-time systems to provide the historical background to the research undertaken in this thesis. It includes the main branches of the subject, mainly on the exploration and evaluation aspects. Sections explaining the NoC and task mapping are also included in the chapter. The first technical chapter (Chapter 3) introduces the proposed design space exploration technique that addresses the problem of optimising multiple design parameters. Chapter 4 proposes a technique that addresses the optimisation problem of task mapping based on multiple objectives. Schedulability is a crucial requirement that must be met by the system, however, the exploration technique that depends on the schedulability metric is ineffective to address the problem when schedulable task mappings cannot be found. This problem is addressed by a new fitness function in Chapter 5. The final technical chapter, Chapter 6, proposes a constructive mapping algorithm as the alternative to the GA-based optimisation algorithms. With this proposed algorithm, the search for task mapping does not rely on a pool of task mappings and thus the process of finding the schedulable task mapping(s) is speeded up. All the contributions of the preceding chapters are concluded in Chapter 7, and the possible direction of potential future works in this field is suggested.

Chapter 2

Hard Real-Time NoC Design Space Exploration

Task mapping exploration for hard real-time systems based on NoC is the central topic of this thesis. This chapter reviews the state-of-the-art works related to the topic as the background to the research works presented in this thesis. The survey is divided into several sections. In section 2.1, an overview of the NoC architecture is provided to introduce different network components and policies such as router, network interface, link, routing protocol and flow control mechanism. Then, in section 2.2 we review existing works related to task mapping and some of the design parameters that could affect the schedulability of task mapping. Design space exploration is facilitated by two main components, the search component and the evaluation component. The search component is reviewed in section 2.3 and it includes several heuristics used to explore task mapping, Pareto-optimal concept for finding a trade-off between multiple objectives and a relation between search and decision making. The evaluation component is reviewed in section 2.4 and it includes different types of evaluation techniques that can be used to evaluate task mapping. In the same section, some of the schedulability analyses for evaluating the schedulability of tasks and messages are also reviewed. All reviews in the sections are conducted in respect of NoC-based hard real-time systems.

2.1 Networks-On-Chip

Networks-on-Chip is currently viewed as a potential solution for providing high performance on-chip communication with better scalability for systems with intensive parallel communication requirements [6]. It has many configurable parameters that can be tailored according to application requirements, offering a variety of possible implementations through different configurations. In order to understand the parameters of NoC, this section discusses the basic building blocks of a generic NoC architecture.

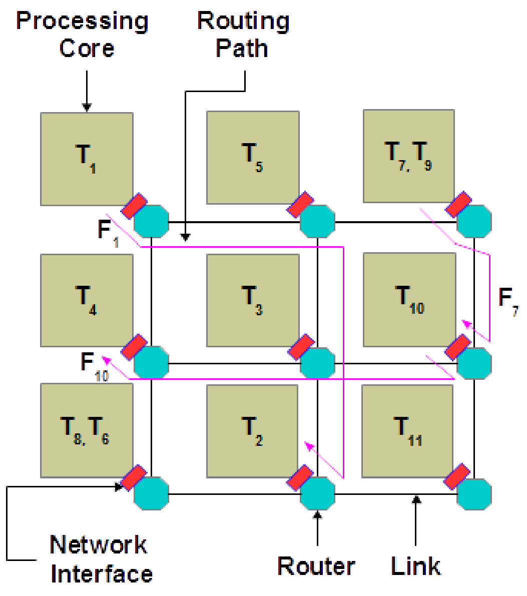


Figure 2.1: An example of task mapping and message routing in a 3x3 Mesh NoC

Generally, a NoC contains three types of network building blocks [22]: routers, physical links and Network Interfaces (NIs). Routers are responsible for forwarding packets from a source to a destination node along the specified routing path, which is determined by a network routing protocol. Routers' input and output ports are connected by links. A link is a physical entity containing a set of wires. Flow control and arbitration policies provide packet management to regulate how links are shared by the contending network packets. Typically, every router is connected to an Intellectual Property (IP) block (such as a processing element, a memory or a peripheral device) and each IP has a different communication protocol from the others. IP blocks require NIs to synchronise communication with routers so that data transmission through shared communication resources is possible. Figure 2.1 shows message routing paths in typical high-level view of a 3x3 mesh NoC structure.

2.1.1 Link

A link is a physical interconnection between two routers, or between a router and a core. It contains one or more physical or logical channels [23]. A *flit* [24] is a basic transfer unit at link level. Several flits may be forwarded through the channels in multiple cycles because of the physical channel width constraint, that is, a flit-by-flit transmission between routers.

Flit transmission is controlled by a flow control (or synchronisation protocol) by sending request/acknowledge signals between a sender and a receiver to regulate the transfer of flits. This synchronisation mechanism is essential to ensure the successful transfer of flits, for example by first checking the buffer space at the receiver side prior to any flit transfers. The synchronisation protocol can be implemented by dedicated wires, mixed-time FIFO (First In First Out) [25] in a multi-clock domain or as asynchronous circuit techniques [26].

2.1.2 Network Interface

A network interface provides high level communication services to the IP blocks by encapsulating the low-level network functions provided by NoCs. By having this layer, IP blocks with different communication protocols are able to integrate seamlessly with the NoC infrastructure. High-level encapsulation of low-level network functions facilitates less interdependence between the IP blocks and NoC, and also helps to ease the reuse of abundant IP blocks available to chip designers.

Communication services provided by the NI involve a point-to-point communication between cores and routers. In this type of communication, a traffic encapsulation service provides the packetisation and de-packetisation of packets. At the destination node, newly arrived packets are converted into signals that can be understood by the core's communication protocol. Conversely, signals from cores must be converted into network packets before routers can forward data throughout the network. In addition to this service, other services [23] such as global addressing, data and buffer managements are also offered by the NI.

As depicted in Figure 2.2 a generic network interface is composed of two parts [23]: the front end and the back end. The front end may be implemented by adhering to a socket protocol and several socket protocols exist, such as the commonly used Open Core Protocol (OCP) [27] or other standards such as Virtual Component Interface (VCI) [28] and Device Transaction Level (DTL) [29]. The OCP offers several properties including specific socket implementation to facilitate design reuse, as well as emphasising on how to simplify the system verification and test. One OCP compliant NI implementation was proposed by Bjerregaard [30], and Radulescu [31] implemented NI for AEthereal NoC based on the transaction-based protocol to allow backward compatibility to the existing communication protocols.

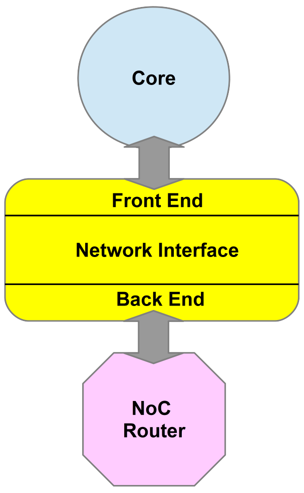


Figure 2.2: NoC network interface

2.1.3 Routers

A NoC's router is composed of several connection components [5, 32, 33] as depicted in Figure 2.3. Among these components are communication ports, including a local port connected to a core and a number of input and output ports connected to routers. The input and output ports are connected respectively to a single incoming physical link and a single outgoing physical link.

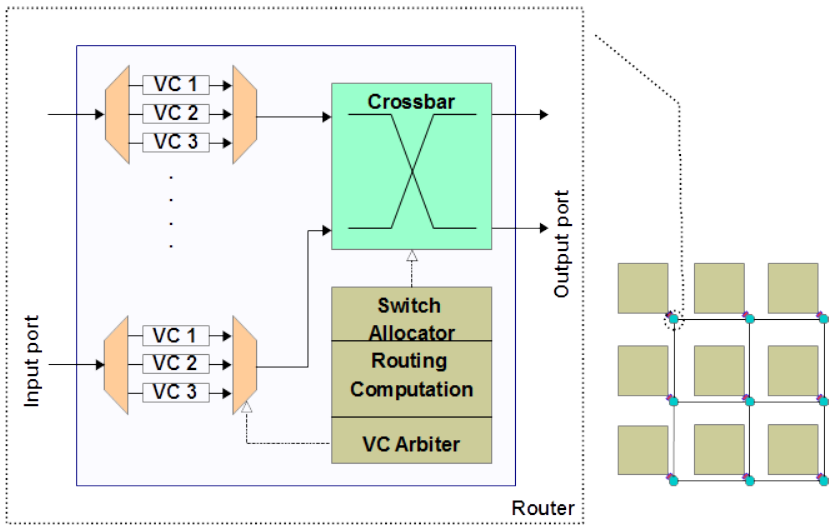


Figure 2.3: NoC router structure

Control logic inside the router is performed by four components: the Routing Computation (RC) unit, the Virtual Channel Arbitration (VA) unit, the Switch Allocation (SA) unit and the Crossbar unit. A packet contains several flits and these components operate at flit level. The routing of a packet is performed based on the destination address which is saved inside its header flit. Based on the information in the header flit, the RC unit directs the header flit to the appropriate output port.

Typically, NoC routers without virtual channels have a single buffer in every port. For some routers with virtual channels [34, 35, 36], each port is associated with multiple buffers. When incoming packets request access to the VCs of an input port, the VA unit checks their header flits and arbitrates between the packets to select which packets are assigned to the input VCs. The VCs from all input ports request access to the crossbar unit, and the component that decides the winner amongst the VCs is the SA unit. The SA unit arbitrates all the VCs because more than one VC may request access to the same output port. It also configures the crossbar unit appropriately by connecting the selected VC to the output port. Unlike RC and VA which both perform the logic operations on the header flit, the SA unit performs its operation on every flit. For

In NoC, buffers may be distributed either at the input or the output ports, or at both sides at the same time [37, 38, 39]. Input port buffers provide storage and enable queuing for arriving flits before reaching the crossbar, reducing the probability of packet loss due to insufficient bandwidth in the router. Normally implemented as a FIFO queue, input queuing may cause head-of-line blocking in routers. This phenomenon occurs due to the FIFO characteristic that only allows packets to proceed on a first-come-first-served basis. If two input ports at the same time contend for the same output port, the unselected input port will contain the current flits in the input buffer, blocking the rest of the flits along the path from arriving at the input port.

One of the solutions for overcoming this problem is to by distribute buffers between the crossbar and the output ports, creating output queuing instead of input. A group of outgoing flits from the crossbar to the output ports is stored in the output buffers while waiting for a transmission to the next router. Although the input ports can continuously receive flits, when contention occurs in the network the routing path of flits will be blocked, thus preventing the routers from progressing the flits forward. If the capacity of the output buffers is saturated at the receiving router, the outgoing flits may be discarded or lost during transmission. Retransmission of flits could cause communication overheads as well as increasing the amount of traffic in the network. By receiving acknowledgement on the buffer status from the receiving router prior to the flits' transmission, a reliability check is established to prevent the flits from being lost during transmission. On the other hand, input and output queuing inherits all the advantages mentioned previously. Network performance may become better due to the bandwidth increase in the router but at the expense of increased complexity. Increasing the number of buffers may have other drawbacks as well, such as excessive power consumption and high implementation cost.

Furthermore, the transmission of flits from routers is controlled by network policies such as a routing protocol and flow control. A routing protocol provides the routing paths for flits and a flow control manages the transmission timing of flits. Both network operations work in relation to each other to ensure smooth data transmission between the routers. The routing operation is responsible for connecting an input port to its corresponding output port to establish a communication path as computed by the routing algorithm (based on routing table look-up or source routing). On the other hand, the flow control mechanism provides appropriate timing for forwarding data between routers, that is, the synchronisation of data transfer between routers after the establishment of the communication path.

2.1.3.1 Routing Protocol

The basic function of a routing algorithm is to select the appropriate path for each packet upon arrival at one of the input ports by deciding which output port the packet will be forwarded to. The selection of a routing path is based on the routing information carried by the packet header. According to the taxonomy presented in [24], routing algorithms can be classified based on their characteristics. The two main categories which divide those algorithms are deterministic routing and adaptive routing.

In a network with deterministic routing, a packet is routed according to a pre-determined routing path between the sender and receiver. The routing path is computed prior to the packet transmission from the source router and remains static for the whole duration until the packet arrives at the destination router, hence the name *static routing*. Among the routing schemes that follow deterministic routing are source routing and XY routing. A source-routing scheme relies on the source node to provide a routing path for the packet prior to its transmission, which is then stored in the packet header. Intermediate routers use the routing information to reserve a path for the packet until it arrives at the destination router. In the XY routing scheme, a packet is forwarded along the row until it reaches an intermediate router where the destination node is perpendicular to it, then it is forwarded along the column until it arrives at the destination node. XY routing may refer to a routing table in order to determine the routing path. The NoC implementation proposed by Kavaldjiev [35] uses source routing, and NoCs which implement the XY routing algorithm include Dally [34], QNoC [40] and Hermes [36].

Adaptive routing determines routing paths for packets according to the traffic condition or link status, for example to avoid congested areas or perhaps faulty nodes in the network. The flexibility of this kind of routing algorithm allows packets to be routed through alternative paths when one of these events occurs.

2.1.3.2 Flow Control

Flow control is responsible for defining how data is transferred along a routing path [41]. A specific flow control mechanism exists at every layer of a network and operates on different type of datagram. For example, the message flow control synchronises fixed-length packet transmission at the network layer, whilst the physical channel flow control manages synchronisation of flits at the bits level.

A packet can be further split into several chunks called flits and a flit consists of several *phits*. Every flit is a fixed-length data transmitted over a physical link and the flits of a packet require several cycles before completely reaching the destination. A phit represents the number of bits transmitted in parallel over the physical wires of a link in a single cycle. In contrary to phits, which are true physical entities, packets and flits are considered as logical in representation.

The way a packet is forwarded varies depending on the forwarding strategy in the flow control policy. Most of the forwarding strategies utilise limited buffers at the input and output ports. This includes common forwarding strategies such as store-and-forward [24], virtual cut-through [42] and wormhole switching [43].

In wormhole switching, a packet is sent from a router to another router flit-by-flit. With this scheme, routers forward the next flits without waiting for the full packet to arrive. A header flit of a packet contains the necessary routing information to select the packet's routing path. Once the routing path has been established, subsequent flits follow the header flit along the same routing path. A flit is forwarded as soon as sufficient space becomes available at the next router, otherwise it remains in the current router's buffer. When the flit header is blocked due to congestion, subsequent flits are stalled along the routing path resembling a worm, which explains its name.

NoCs implementing wormhole switching require a smaller size of buffers in the input and output ports than virtual-cut-through and store-and-forward policies, as the transmission is made on a flit-by-flit basis. This avoids having to accumulate all the segments to build a complete packet before initiating transmission, which keeps end-to-end latency low because flits are transmitted as soon as the buffer at the next router becomes available. Several NoCs [36, 40, 44] have employed the wormhole switching technique. In Hermes NoC [44], the NoC was implemented with a wormhole switching technique combined with four types of routing algorithms including a deterministic XY routing and three partially adaptive routings including west first, north last and negative first. Kavaldjiev [35] implemented a wormhole switching technique with a source routing algorithm. The same implementation was also applied by AEthereal [45] to support Best Effort (BE) type packets.

The virtual-cut-through is similar to wormhole switching in terms of the forwarding mechanism, but operates at the packet level. With this technique, a packet can be forwarded to the next router as soon as sufficient buffer space is available. In other words, it does not necessarily have to wait for the entire packet to arrive at the current router before it starts forwarding it. However, larger buffer size is required than for wormhole switching because it must be able to accommodate the entire packet when the next router is not ready to accept due to insufficient buffer capacity. In addition, a packet may have to wait longer due to the time needed to free sufficient buffer capacity before the next router is ready to accept. If a packet is blocked when congestion occurs in the routing path, it stalls inside the current router but does not block the path as it does in wormhole switching.

Based on the store-and-forward policy, a packet must be in complete form before it can be forwarded to the next router. Similar to virtual-cut-through, the packet stalls in the current router if sufficient buffer space is not available in the next router. In terms of end-to-end latency, this forwarding strategy causes lower performance of the NoC than the previous forwarding strategies discussed above. An example of a NoC that employed a store-and-forward policy was introduced by Kumar [33].

2.1.3.3 Virtual Channels

The flit-by-flit transmission mechanism reduces end-to-end latency but several drawbacks such as low link utilisation and deadlock [46] are likely to occur in a NoC implementing wormhole switching policy. One way to alleviate this problem is to apply Virtual Channels (VCs) [34], whereby a single physical link is multiplexed into separate logical channels to allow access for other packets to proceed even when congestion is blocking the preceding packet in the same path.

Generally, an input port (or output port) is associated with a buffer that becomes a temporary storage for a packet until the next router has adequate space to receive it. When the next router is ready to receive, the input port is matched with the corresponding output port by configuring the switch crossbar accordingly. If two packets arrive at the same time at different input port but compete for the same output port, an arbitration unit must decide which packet will gain access to the physical channel. If the packets originated from BE traffic, fair distribution of resource can be achieved by allocating a similar amount of usage time for each packet. For Guaranteed Traffic (GT), some traffic flows can be guaranteed access on shared resources by assigning priority levels to packets, that is, higher-priority packets can pre-empt lower priority packets to gain access to the physical channels.

Indeed, buffers and physical channels are two important network resources which are not only shared but also concurred among packets. Although a NoC architecture with a single queue in input or output port is simple and less complex in implementation, when a large number of packets are competing for the same resources at the same time, its bandwidth is reduced considerably and the network will experience maximum throughput quickly. When this happens, head-of-line blocking can occur and the valuable network resources will be blocked for a long duration by a packet which prohibits other packets en route from utilising it. This phenomenon can be more clearly understood by referring to Figure 2.4. In this figure, each router node has five output ports (north, east, south, west and local) and each port is associated with a buffer. Packet X is transmitted from router 2 to router 3 but cannot proceed to router 5 due to a blockage. However, the flow of packet X is still alive in the physical channel between router 2 and router 3.

Some flits of packet X are stored in buffer at the port south of router 3 and the rest are stored in buffer at the port east of router 2 until the blockage is cleared. At the same time, router 1 is ready to transmit packet Y but it is blocked by packet X from using the same physical channel. During this time, the physical channel between router 3 and router 4 remains idle because it cannot be utilised by packet Y due to the blockage by packet X. As a consequence, the blockage creates low throughput at the output ports and consequently causes inefficient utilisation of the physical channel bandwidth.

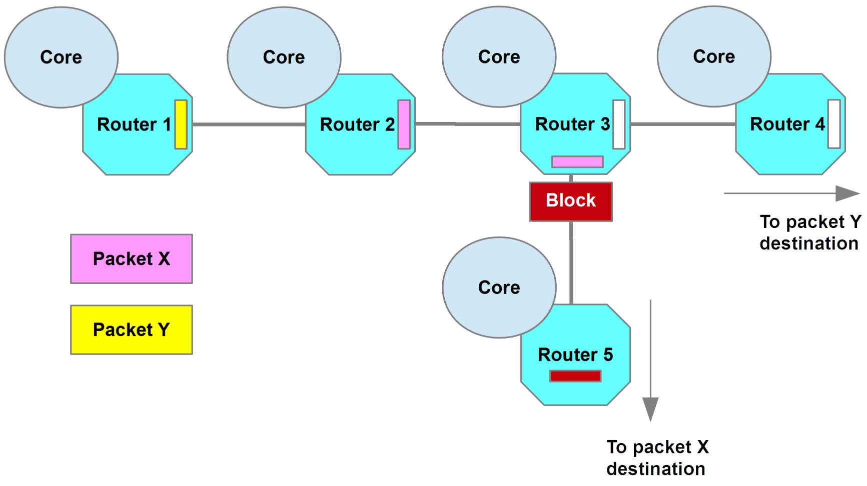


Figure 2.4: Head-of-line blocking

Achieving higher network throughput and efficient utilisation of physical channel bandwidth is difficult when network resources are tightly coupled with single buffer designs in ports. The percentage of network throughput that can be achieved is between 20%-50% of overall network capacity [47]. Dally proposed a concept called *virtual channels* [34], which decouple the network resources by associating the input port with more than one shallow depth buffer as substitutes for a single deep-length buffer implementation. Figure 2.5 shows a NoC with each router assigned with two virtual channels at each input port. With the same situation as shown in Figure 2.4, packet Y arrives at routers 2, 3 and 4 without being blocked by packet X, even though both packets share the same physical channel. In Figure 2.5, the physical channel utilisation is better than in the single queue implementation (see Figure 2.4) due to the existence of different buffers to store packet X and packet Y separately. Therefore, when packet X is blocked at a port south of router 3, the same physical channel can be re-assigned to packet Y to allow it to proceed to the next router. The decoupling of packets' queuing through virtual channels increases the throughput and maximises the utilisation of physical channels.

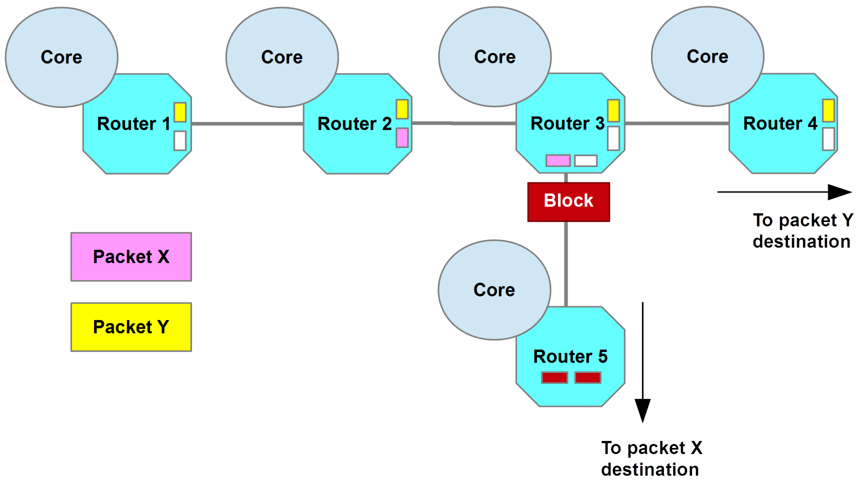


Figure 2.5: Head-of-line blocking avoidance with Virtual Channels

Virtual Channel (VC) flow control requires that a physical channel is split into multiple virtual channels and that each virtual channel has its own associated buffer queue. Figure 2.6 shows a simple FIFO buffer structure of a virtual channel. A specified number of k-flit FIFO buffers are contained within an input port of a NoC router and each FIFO buffer corresponds to a VC. In a conventional NoC router, the number of VCs per input port is fixed. With the VC technique, if a packet which is currently using the physical channel is blocked by congestion in its path, other packets in the other virtual channels will compete for the physical channel and the chosen packet will bypass the blocked packet. Some NoC implementations such as ANoC [48] consists of fixed priority arbitration scheme which can be implemented using a VC technique. With this scheme, each VC is assigned a priority level to accommodate packets with the same priority level. A request coming from a packet with priority level i is served by no other VCs except a VC with the same priority level. A direct one-to-one relation between a packet and its corresponding VC based on the same priority level enables a virtual path for the packet by reserving a series of VCs along the packet route.

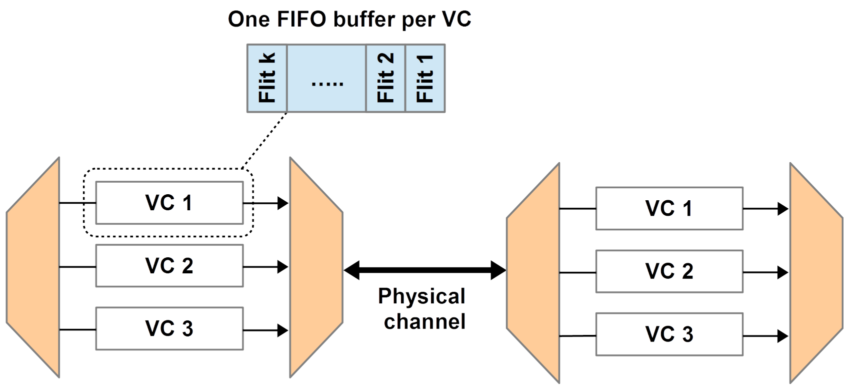


Figure 2.6: Virtual Channels

2.1.4 Summary

NoC contains several network components such as routers, link and NI. It also depends on routing policies and flow control to deliver messages. In this section, these components, policies and control mechanisms were explained to give a snapshot on how NoC works.

2.2 Task Mapping

Task mapping can be explored by changing the allocation of tasks on the multiprocessor platform, producing different types of mapping with varying performances. However, task mapping is a NP-hard problem [49]; its time complexity expands with the number of tasks and cores and hence it is impossible to find the optimal solution in polynomial time unless the right decision is made every time a task is mapped onto a core. Search heuristics such as Genetic Algorithm (GA) is well-known to address this kind of problems although it cannot be guaranteed that the best solution it finds is optimal. GA working principles are based on natural selection of a population. A group of individuals in a population are evolved to create better individuals that represents useful solutions or task mappings. Individuals are evolved by GA through manipulation of each individual chromosome through several evolution steps. Chromosome is a string of information in which task mapping can be easily encoded to create different task mappings.

A few researchers have proposed several approaches to find the mapping of IP blocks or cores onto the NoC architecture based on single-objective optimisation such as the minimisation of system delay or total communication energy. Lei *et al.* [50] proposed a two-step optimisation approach using multiple GAs to map IP blocks onto a NoC architecture. The objective was to improve the system performance by minimising system delay, which is defined as the summation of all tasks' execution times and communication delays. The aim of the first step is to search for the appropriate type of IP for each task, whilst the second step is aimed to find the best mapping of IP blocks on a NoC platform. In Lei *et al.*'s evaluation model, a communication delay is estimated based on the average distance in number of hops between any two nodes. Murali *et al.* [51] addressed the problem of mapping processing cores onto the NoC architecture by taking into consideration the bandwidth constraints of the links. By finding a mapping that could reduce the total communication cost, the desired message transfers can be supported by the links. In order to achieve this, the minimisation of the communication cost was performed by splitting traffics across multiple paths between source and destination. The total communication cost is the cumulative product of bandwidth requirement and the hop distance of all traffic flows. Hu *et al.* [52] presented a mapping approach based on a Branch-and-Bound algorithm for mapping IP blocks/cores onto a NoC architecture. Their main objective was to minimise the total communication energy while ensuring that all communication flows have enough bandwidth to travel through the communication paths. With their proposed algorithm, the search for the best mapping with the least energy is performed alternately between branching and bounding steps. In the former step, creating new nodes (mappings) follows the form of a search tree, in which a node is branched out from its parent node. The decision whether to create a new node depends on the energy cost (must be less than that currently found) and the conditions (which includes meeting the bandwidth limits) that must be met by the current node. Further expansion of a node is stopped if any of these requirements is not met. A similarity between these approaches is the dependence on the hop distance to calculate the communication delay and bandwidth requirement. However, this ignores the fact that communication delay can also be influenced by the contention between the traffic flows. Furthermore, in single-objective optimisation, the exploration of task mapping is directed towards achieving a single objective, consequently task mappings become inefficient due to lack of performance in other aspects.

Some researchers [17] have focused on multi-objective optimisation of task mapping, with other parameters assumed to be fixed. Ascia *et al.* [17] used a search-based heuristic called SPEA2 [53] to explore IP mapping, based on the minimisation of delay and the average power consumption. Using a cycle-accurate simulation technique, evaluation was performed on each mapping using various dynamic behaviours from synthesised traffics and a real application. Although task mappings produced by this heuristic are better than random mappings, the use of simulation as an evaluation technique in design space exploration has a few drawbacks. The time cost of detailed simulation is very high to evaluate every mapping, since every simulated application has a number of simulations which have to be performed in order to gain representative performance results [54]. If a detailed simulation model is used to achieve a precise measurement, the amount of time escalates even further to propagate events from each of the components in the model during simulation. SPEA2, like any GA, depends on the size of the population to explore the design space and with a larger population, the diversity of the task mappings is high, hence providing many alternative task mappings to explore. The high time cost of the evaluation technique can be a factor that could prevent the idea of using a larger population. Resorting to a smaller population, however, has its own pitfall, because it leads to early convergence due to reduced diversity in the population. Early convergence too has its own consequence: either the feasible solutions may be hard to find, or the feasible solutions may probably not be the best solutions found so far. The preparation of a detailed simulation model and its verification that follows afterwards must also be taken into account since system designers are normally given tight schedules to choose good designs.

Similar to [17], other researchers have proposed another multi-objective optimisation [18], but using a different type of GA called NSGA [55]. According to the thermal model [52], heat dissipated by an IP can be transferred to other nearby IP blocks if they are positioned closer to each other, causing a rise in temperature at that concentrated spot. Increasing the average hop distance could dissipate heat further from the spot and create a good thermal balance, but conversely it means increasing the communication delay between them. Based on this notion, they proposed an approach to find task mappings with good trade-offs between the IP thermal balances and hop distance. Different kinds of mapping exist to address the communication synthesis problem as well as the computational synthesis problem. The two-step optimisation approach proposed by Jena [19] applied a GA known as NSGA-II [56] to explore solutions for both problems based on multiple objectives. The purpose of the first step is to find the feasible mapping of tasks onto IP blocks in a way that minimises the computation power consumption and the total cost of resources. Following the first step is the second step that maps IP blocks on a NoC platform based on multiple objectives: the minimisation of the number of switches and the maximisation of link bandwidth. A similar energy model to [18] was applied, but modified to include the computational power consumption for the first step's optimisation. Nedjah *et al.* [20] addressed the same synthesis problems as [19] but used more than two optimisation objectives. Two different GAs were applied in their approach, the NSGA-II [56] and micro-GA [57], to explore mappings in terms of power, area and execution time. Unlike previous approaches that relied on the hop distance to calculate the communication delay, contention in shared communication channels was considered when calculating the total execution time of computation and communication. Contention was modelled as a time penalty imposed on every flit that would be transmitted when the contention occurs in shared communication channels. The time penalty was a product between the number of flits and the time required to transmit a flit through a communication channel, however it was rendered less accurate by assuming that each flit has an equivalent amount of delay since some flits would be more delayed due to pre-emption.

None of the approaches reviewed above, either facilitated by simulation or using an analytical method as the evaluation technique, are suitable for addressing the optimisation problem of mapping a hard real-time application onto a

NoC-based platform. For this type of systems, the feasibility of task mapping is determined by how many tasks and messages are schedulable. This requires rigorous analysis not only of the response time of tasks but also of the latency of messages. However, applying the number of hops [18, 50, 52] when calculating the communication delays is insufficient as it lacks any insight into the amount of interference that some of the messages may experience when contention occurs, whilst the use of the time penalty [20] does not calculate accurately how long some messages have suffered from delays.

Some researchers applied well-known techniques in machine learning to address the application mapping problems on NoC-based platforms. Sepulveda [58] based her technique on artificial immune algorithm to find mapping for several applications running on the same SoC, which meets multiple objectives of power and average latency minimisation. Other mapping technique was based on Ant Colony Optimisation (ACO) as proposed by Wang in his work [59]. His approach was aimed at minimising the NoC link bandwidth, so that an efficient NoC design with a low link operating frequency and small link width can be achieved. In both approaches, the researchers did not consider hard real-time application as what the Benyamina conducted in his work [60]. Although Particle Swarm Optimisation (PSO) used in Benyamina's approach [60] is a good optimisation technique for finding a task mapping with minimised execution time and energy consumption under specific hardware constraints, the evaluation model that they used yields pessimistic results due to the lack of necessary analysis on the worst-case execution time of task.

In a hard real-time system, meeting the deadline of every task and message in any scenario is necessary. One or more unschedulable tasks or messages that fail to meet their deadlines have undesirable effects which reduce the predictability of the system. Since task mapping could affect the response time of tasks and the latency of messages, a few researchers [11, 21, 61] have proposed GA-based mapping approaches for this type of system. For evaluating the schedulability of task mappings, real-time analyses were applied to all tasks and messages. The task mapping approach proposed by [11] focuses on single-objective optimisation, which minimises the number of unschedulable messages to find the task mapping. However, evaluation of the schedulability of tasks was omitted from their approach. It is therefore not known how many tasks can be schedulable with the found task mappings. Excluding the schedulability of tasks from the evaluation and focusing only on the timing constraints of all messages is insufficient and most likely makes the system prone to failure using the task mapping. In fact, message transmissions are initiated following task computations, hence delays on the tasks side significantly affect the response time of messages.

Similarly, Gan *et al.* [61] covered the computation aspect of the system in the evaluation model of their multi-objective optimisation approach with the schedulability analysis of tasks, but excluded the schedulability analysis of messages. At the early design phase, some application requirements can be uncertain and new functionalities may be added during the design to meet the market demands. Taking these aspects into account, Gan *et al.* [61] proposed a mapping approach to find a task mapping for hard real-time applications by looking into the uncertainties in the worst-case execution time and future tasks that represent new functionalities. Their aims were to find a mapping with a high probability of a task set being schedulable and a high flexibility of accommodating future tasks. In order to achieve these aims, the optimisation approach was applied to maximise the robustness of the task mapping and the flexibility at the same time. However, excluding the schedulability analysis of messages from the evaluation model hinders the system's overall schedulability, since uncertainties in the worst-case execution time, although they influence the response time of tasks, also affect message transmission.

Realising the importance of both computation and communication in assessing the overall schedulability of the system, Racu *et al.* [21] presented an approach for task mapping, which includes schedulability analyses of both tasks and messages. Using a single objective, mainly to minimise the number of unschedulable tasks and messages, they showed that it is possible to find a feasible task mapping using a heuristic such as GA. In reality, however, designing a system is usually constrained by limited resources and energy consumption, so system designers are obliged to meet multiple objectives.

Design parameters in a NoC-based hard real-time embedded system, if correctly configured, can achieve the desired system performance as defined by the design objectives. However, considering all design parameters is a complex problem and exploration can easily become intractable [10]. A slight change on the configuration might produce a different instance of the system with varying performance. Different types of design parameter exist, and in the following subsections previous works related to specific parameters used in this study are reviewed with respect to task mapping for hard real-time embedded systems.

2.2.1 Priority Assignment

In section 2.2, previous works [11, 21, 61] related to the task mapping of hard real-time tasks with fixed priorities were reviewed. Those researchers proposed several approaches for mapping this kind of task, whose priorities are assigned according to a priority assignment policy [62], or by random assignment. With a good heuristic such as GA, they have shown that by changing the task mapping itself to meet the single objective [11], it is possible to find a feasible task mapping. However, in some cases where a feasible task mapping cannot be found, the algorithm converged below 100% schedulability towards the end of optimisation, as shown in Figure 2.7.

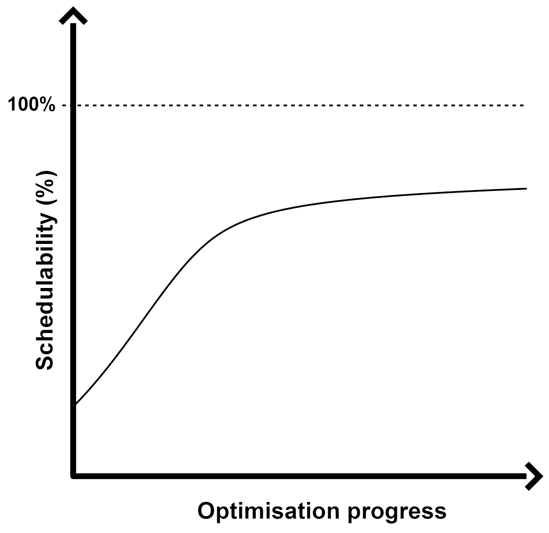


Figure 2.7: Optimisation algorithm converged below 100% schedulability

The result is a manifestation of the restriction imposed on the design space exploration of a system which focuses only on task mapping. This restriction renders the algorithm inapplicable for effectively finding a feasible task mapping in those cases. In this kind of system, shared resource usage is controlled among tasks and messages according to a priority assignment policy. High-priority tasks and messages are given access to the resources by pre-empting their low-priority counterparts. This has its side effect in that any low-priority task or message that fails to meet the deadline as a consequence of prolonged delays will easily become unschedulable. The importance of both steps in the multi-processor real-time scheduling of hard real-time systems is highlighted in [63], which divided the multi-processor scheduling into two types of problem: the allocation of tasks and the assignment of priority levels.

Multi-processor scheduling algorithms have been developed in an attempt to solve these problems and can be categorised into two different types of scheduling: partitioning scheduling and global scheduling. In the partitioning approach, the multi-processor scheduling problem is addressed as a set of single processors that are independent from the others. Every processor is viewed as having a separate priority-ordered queue in which a set of tasks is allocated. A task may require one or more job execution towards its completion. Once a set of tasks has been allocated in a processor, all jobs coming from the tasks must be executed only on the corresponding processor, that is inter-processor job migration is prohibited. In the global approach, the multi-processor system has a single priority-ordered queue for storing all eligible tasks. The global scheduler then selects tasks based on their priority so that the highest-priority one has the highest chance of being executed first in the available processors. Job migration is allowed in this approach.

For both the partitioning and the global approaches, tasks are scheduled in an order given by a priority assignment policy for determining the sequence of job execution in several processors. The priority assignment policies can be grouped either as static or dynamic. A static (fixed) priority assignment policy assigns a unique priority level for each task; therefore all jobs associated with the task have the same priority. The Rate-Monotonic (RM) approach [64] is an example of this type of scheduling policy. In spite of having static priorities, a dynamic priority assignment policy assigns the jobs of a task with different priorities all the time during execution, for example Earliest Deadline First (EDF) [64], or with the priority of each job able to change at any time, for example Least Laxity First (LLF) [65]. These priority assignment policies enforce pre-emptive scheduling to ensure the predictability of the system.

The interference suffered by the pre-empted tasks is related to the way priorities are assigned to them. For example, with the RM priority assignment policy [62], priorities are assigned to tasks according to their minimum inter-arrival interval or period. Whilst this is optimal for a single processor system [64] in that it can provide a schedulable ordering whenever such an ordering exists, it limits the NoC-based multi-processor system from becoming fully schedulable. The complexity is much greater in such a system, and contention is likely to occur anywhere in shared computation and communication resources. Therefore, the execution of tasks in one processor can directly or indirectly affect the response time of other tasks in different processors as well as the messages transmitted throughout the NoC. A task with the longest period which shares resources with those that have shorter period suffers the most interference.

Similarly, messages can be assigned with priorities in the same way as tasks. Based on RM, Mutka *et al.* [66] proposed a priority assignment approach to address the priority assignment problem for messages. Due to the non-optimality of the approach, Shi *et al.* [12] proposed an approach based on the Branch-and-Bound search algorithm to find a feasible priority ordering for making a set of traffic flows schedulable. Since the focus is on the schedulability of traffic flows, they excluded the priority ordering for a set of mapped tasks. This is insufficient, considering that the message transmissions of low-priority tasks are prone to delays caused by a lengthy pre-emption from high-priority tasks on the same processor. Therefore, the applicability of that approach is limited to the schedulability of traffic flows. As far as the overall schedulability of the system is concerned, a priority assignment approach should take into account the end-to-end response time tasks. Racu *et al.* [21] considered the end-to-end response time schedulability analysis as the underlying evaluation of their approach. However, they addressed a different kind of problem, which was to find a suitable setting for the single-objective optimisation algorithm. Improving the schedulability of the system by relaxing the restriction on the design space exploration through priority assignment and task mapping was not the aim of their work.

2.2.2 Operating Frequency

State-of-the-art approaches to hard real-time task mapping use the number of unschedulable tasks as the metric to evaluate the fitness of every mapping [11, 21]. Figure 2.8 shows the outcome of the approaches when using GA to optimise task mapping.

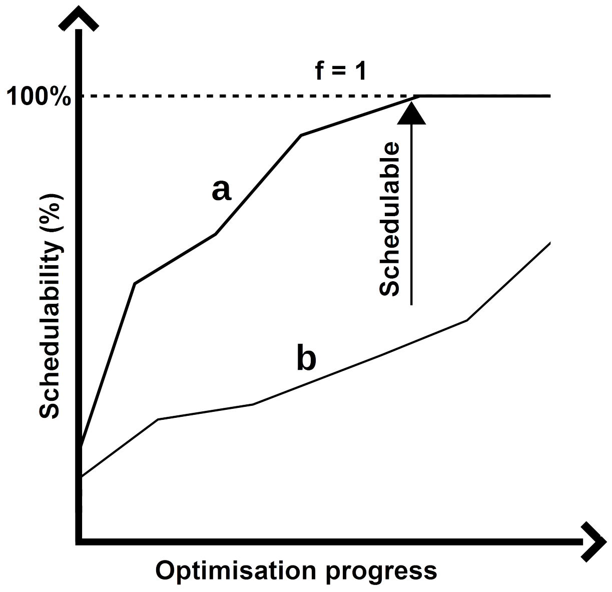


Figure 2.8: Schedulability convergences of the optimisation algorithm

Axis x represents each point of time (or can be shown as the number of generations) the best task mapping was found by the algorithm while axis y represents the level of schedulability in percentage achieved by the task mapping at each point of time. Line *a* is an example of a successful convergence to a fully schedulable task mapping. This result is similar to the experimental work presented in [11, 21], showing that the GA could converge to a fully schedulable task mapping using the metric. However, the same authors also reported results similar as shown by line *b*. Line *b* shows that in some cases the algorithm failed to converge to a fully schedulable task mapping with the same metric. The heuristic depends on the stochastic nature of its characteristic to perform the exploration on the design space, so it requires a useful metric to direct its search towards the optimisation objective. Without a metric that could give an additional insight into the availability of feasible task mapping, it is hard for the heuristic to improve its search in the direction desired by the system designers.

An abundance of real-time analyses [67] has been developed to support the schedulability evaluation of hard real-time systems. The quantitative metrics used in [11, 21] are based on this type of analysis. In these analyses, particular worst-case timing attributes such as the worst-case execution time of each task is assumed to be known in advance, which is possible by using a specific technique such as that proposed in [68]. Under an operating system frequency (or nominal frequency), the worst-case execution time of each task running on its own without any interference from other tasks is determinable. So, it is safe to assume that the evaluation of each task to yield the metric for a given task mapping is valid at that frequency. Based on this concept, single-objective task mapping optimisation can be performed to minimise the metric [11, 21]. Another study [61] considered multiple objectives such as maximising the robustness and flexibility of task mappings to increase the probability of a task set being schedulable and also to accommodate future scenarios.

Those works were primarily aimed at finding feasible task mappings for hard real-time systems, but they lacked further work on optimisation involving frequency scaling. One of the purposes of frequency scaling is to find a minimum frequency at which a task set can be executed with power efficiency in place. For hard real-time embedded systems, changing the frequency has significant effects on the response time of tasks, affecting the ability of those tasks to meet their deadline. Some researchers have proposed single-processor frequency scaling for real-time tasks; [69] and [70] presented an approach that keeps the corresponding frequency constant at runtime, whilst [71] proposed finding multiple speeds for a single processor. Although not many works have been published on frequency-aware task mapping optimisation for NoC-based hard real-time embedded systems, Shin *et al.* [72] utilised the GA algorithm to assign the lowest operating speed to links by reducing slack time. In that approach, the communication load on each link was used to determine the worst-case communication delays. However, in priority pre-emptive hard real-time systems, it is insufficient to consider just communication loads due to the extra delays caused by pre-emption by high-priority messages, which was not considered in their model. This limits the application of the approach for minimising frequency for this type of system.

2.2.3 Summary

Many researchers have proposed different techniques to optimise task mapping. In this section, we reviewed the suitability of the techniques for optimising task mapping intended for NoC-based hard real-time systems. Most of the techniques, especially those that were targeted for finding task mappings in average case, are not suitable for addressing the hard real-time task mapping optimisation problems. Finding task mappings for this kind of systems require analysis of every task and message to determine their schedulability in the systems. Other design parameters which can influence the fitness of task mapping such as priority assignment and operating frequency were also reviewed.

2.3 Early Design Space Exploration

A NoC-based hard real-time embedded system contains numerous design parameters which can be configured to meet specific design objectives. Finding a feasible configuration that meets those objectives, however, is a challenging task due to the vast design space that embodies many potential configurations. The design space expands to even larger multi-dimensional space as more parameters are involved in the design. Consider task mapping and priority assignment as an example: the number of task mappings that can be explored from n tasks and m number of cores is m^n, while the number of ways priority assignment can be changed is $n!$. Combining both parameters together yields a larger design space than exploring them separately; the total number of permutations the parameters can be explored becomes $m^n \times n!$. Therefore, pinpointing the exact location of a feasible configuration becomes harder because any configuration from the vast design space has a potential to become feasible. Since the exact location is not easily determined, exploring as many potential configurations as possible becomes a necessity until the desired configuration is found, or the search algorithm reaches its maximum search limit.

At the low-level abstraction of design, exploring as many configurations as possible is prohibitive due to the complexity arising from high amount of detail in the system architecture. This imposes several constraints on system designers for deciding suitable design choices early in the design stage and for producing the first prototype quickly under a tight design schedule. The impracticality of exploring many possible configurations urges system designers to shift the paradigm from low-level to system-level abstraction. At the system-level of abstraction, separation between functionalities and the architecture of the system reduces complexity, simplifies verification and at same time allows more alternative designs to be explored [73]. The exploration step at this stage henceforward is called *early design space exploration*.

Early design space exploration is often applied to deal with high-level synthesis problems [10] such as resources allocation (for example, task mapping) and for the scheduling of operations under a variety of constraints. Task mapping is a necessary design step for assessing the performance and the cost of the whole system against the allocated resources. In the task mapping process as depicted in Figure 1.2, decoupled models of application and architecture are integrated according to the Y-chart approach [74]. Decoupling eases the refinement of each model, but through the complete model (mapped architecture) the analysis of each task's and message's response time can become possible, as far as the hard real-time embedded system is concerned.

In fact, task mapping is one of the main problems alongside priority assignment in the multi-processor scheduling of hard real-time systems [67]. Both problems have such a high impact on the system's schedulability that scheduling algorithms were developed to address them. Normally, bin-packing algorithms are applied to address the problem since allocating tasks to multi-processors is analogous to bin-packing problems [67]. The bin-packing algorithms' common working principle is deterministic, filling a set of elements into one or more containers according to specified rules and hence producing fewer alternatives of task mapping. On the other hand, DSE techniques could explore as many task mappings as possible to address the multi-processor scheduling problem, providing more design options for system designers.

The success of a DSE technique depends on its ability to efficiently explore either in the problem space or in the objective space, or in both at the same time [10]. In the problem space, a system designer uses a search algorithm to explore design parameters. As depicted in Figure 2.9, the dimension and size of the problem space is defined by the intersection of the axes which represent the parameters. The specification of parameters is part of the problem description, for example in the context of finding task mapping the size of a platform is given by the number of cores that it contains and is represented as a series of integer indexes.

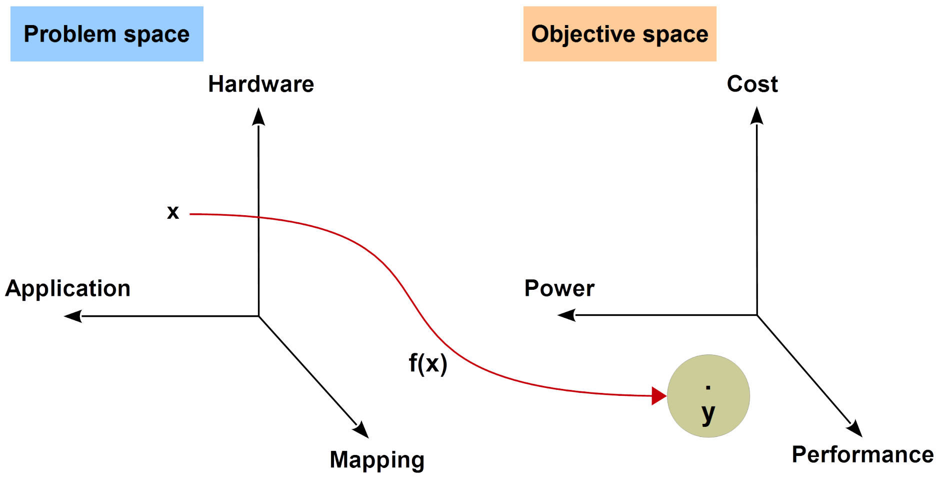


Figure 2.9: Design space exploration perspectives

On the other hand, the exploration of parameter values is guided by one or more objectives that exist in the objective space and objectives are often defined to optimise the system in terms of cost, power or speed, or all three at the same time. With a single objective, exploration is restrained in one direction towards the objective. The exploration becomes harder and more complicated with multiple objectives because it must now be restrained towards multiple directions, in other words, the exploration of fulfilling one objective should also support the fulfilment of the other objectives. Very often design objectives conflict with each other, making simultaneous optimisation of multiple objectives complicated and hardly realisable [75].

Exploration in the objective space is facilitated by one or more fitness values of every configuration found by the algorithm in the problem space. Given a group of fitness values of each configuration, its whereabouts can be located in the objective space. The algorithm utilises these values to compare and then selects the configurations that are closest to the objectives. Fitness values of a newly-found configuration are yielded by one or more fitness function. To help choose the best configurations, these values need to be as accurate as possible, or within acceptable estimations. At the higher level of abstraction, based on coarse-grained models, fitness estimation is used to achieve fast evaluation. As a result, this leads to a slight deviation from the accurate plotting point, as shown by the blue region of the circle surrounding point y in Figure 2.9. As long as the radius of the circle is within the acceptable range from the true point, the search algorithm will be less affected by the influence of the deviation. Otherwise, the location of every configuration will be easily misrepresented in the objective space, causing false comparisons and leading to unfair selections. An efficient fitness function will facilitate the selection of the best configurations by allowing the algorithm to evaluate as many as alternative configurations as it can. Nevertheless, a good fitness function will provide a necessary metric to give an insight on the quality of each configuration.

2.3.1 SW/HW Co-design Limitation

In the traditional SW/HW co-design [76] top-down approach, a complete specification that specifies the functionalities and the details of implementation is required. Therefore, during the design of system architecture, both the functional and the non-functional requirements of the system are taken into consideration. At this stage, system designers decide which parts of the architecture will become the software or hardware blocks. The development and refinement of each block can be performed separately, followed by an integration to create a complete design for evaluation as a whole system. For a successful integration, interfaces must be well-defined during the decomposition of the two blocks. Without well-defined interfaces, the integration will be hard to achieve due to the incompatibility between the two blocks.

With this approach, each block has a tendency to rely closely on the others, which might increase the development time significantly [76, 77]. For example, during the refinement process, software engineers must wait for the hardware to execute the source codes. Software implementation can also be affected by the complexity of hardware details when design flaws exist on the hardware side, causing a software problem that requires more debugging time. Furthermore, hardware description is still based on a Hardware Description Language (HDL), for example VHSIC Hardware Description Language (VHDL) or Verilog, to represent hardware models. Evaluation of these models can be executed through simulation techniques with cycle-accurate precision. However, highly complex implementation details take a long time to evaluate, hence reducing the ability to explore wide areas in the design space.

Design methodology has recently shifted above the RTL level to achieve a better understanding of the system. This high level of abstraction is known as system level or Electronic System Level (ESL) [78] design. System level design is still in its infancy and a definition that gives an appropriate meaning of it is yet to be derived, but the closest interpretation is that given by Bailey [78]:

> *"The utilisation of appropriate abstractions in order to increase comprehension about a system, and to enhance the probability of a successful implementation of functionality in a cost-effective manner, while meeting necessary constraints"*.

At system level, complex system architecture can be modelled by reducing some degree of detail. One way to accomplish this is by focusing first on the behaviour (functionality) of the system rather than how it is implemented at accurate timing [79]. Since the model will be simplified, appropriate evaluation techniques can evaluate it faster through estimation. This will lead to successful implementation, as vast exploration can be performed to give system designers a pool of choices to select from at the early stage of design. It is more cost-effective to know which choice works best at the early stage, since further refinement can now be performed based on the best choices made by system designers.

2.3.2 Pareto-optimal Concept

In a single objective optimisation problem, selecting the best solution is a matter of determining which solution is at the front of an ordered list of solutions.

Depending on the optimisation objective, it can be a solution with the greatest minimum or maximum fitness value. As far as a single objective is concerned, deducing the best solution is relatively easy as all the potential solutions can be fully ordered in the solution space.

Unlike single-objective optimisation, determining the best solution in a multi-objective optimisation is harder because the algorithm used to explore the design space has to meet different objectives at the same time. Furthermore, the objectives have a tendency to be in conflicting directions, complicating the process of selecting the optimal solution for the problem in hand. Due to these conflicting objectives, a solution may have a good fitness in one objective but be worse in other objectives, thus trying to produce an optimal solution that meets all the objectives is too hard to achieve. Instead of trying to search for the optimal solutions by trying to become excellent in all objectives, the search algorithm seek a set of solutions that have good trade-offs between the objectives. In this way, the difficulty of determining the preferred solution can be reduced and system designers have multiple choices to decide which solution best meets the requirement of the system. If the algorithm is effective, achieving a solution with a good trade-off between the objectives is possible and definitely more efficient than some random-picked solutions.

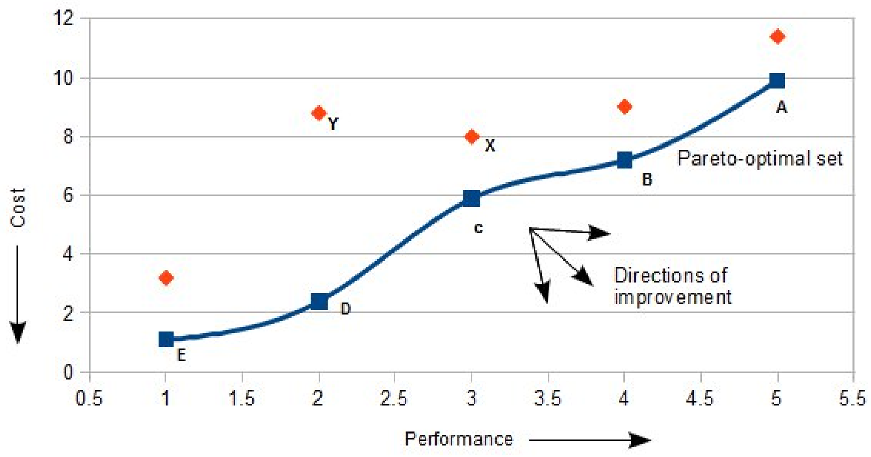


Figure 2.10: Pareto-optimal set

A set of solutions with superiority above the others in the solution space is a dominant set. According to the Pareto concept of optimality [80], this set is known as the Pareto-optimal set. To be in this set, a solution must not be dominated by other solutions, that is, the solution must be better at least in one objective and not least in other objectives compared with its counterparts. The number of solutions is not limited to one; any solution that meets the criterion is a possible member of this set. All the solutions in this set are equal, in the sense that none can be considered better than the others, because their trade-offs are incomparable. Based on Pareto's concept, using the example shown in Figure 2.10 the explicit definitions of dominance and optimal solution are described as follows:

Definition 1(Pareto criterion for dominance): The Pareto criterion for a solution to become dominant in respect to the others is that the solution must be at least good in one objective but without becoming worse in other objectives. For example, given two solutions **X** and **Y** in the solution space, **X** dominates **Y** because the former provides better performance and lower cost than the latter.

Definition 2 (Pareto-optimal solution): A solution is known as the Pareto-optimal solution if *Definition 1* is met when further improvement stops. All solutions in the solution space that meet *Definition 2* are contained within a non-dominated set (also known as a Pareto-optimal set) as illustrated by the blue line in Figure 2.10. The non-dominated solutions are incomparable with each other. For example, neither solution **A** nor solution **B** dominates the other; **A** offers better performance but **B** has lower cost, although the latter has lower performance than the former.

2.3.3 Search and Decision Making

In multi-objective optimisation problems, the complexity is aggravated by conflicting objectives. If conflict does not exist, each objective can then be individually targeted by the optimisation algorithm. By aggregating all objectives into a single objective, each solution will be treated similarly to a single-objective optimisation problem, by which they can be fully-ordered and the optimal solution will be easily decided. However, this is not always realistic as in the real world conflicting objectives commonly occur. For example, performance and power are often conflicting; high performance consumes more energy, but releases a lot of heat and dissipates more power too. The device may have less operating time and its lifetime might also be reduced.

Conflict between objectives prevents optimising the solutions towards each objective simultaneously, that is, meeting all the objectives at the same time. Instead, solutions are produced with different trade-offs between all the objectives, which subsequently raises a problem in the multi-objective decision-making. Input from decision makers is necessary to rank solutions and decide which solution is the most appropriate for the optimisation problem. Furthermore, trying to satisfy all the decision makers places them in a difficult situation as each of them might have different preferences over the objectives and therefore different orderings of the solutions.

Search and multi-criteria decision-making are related to each other. For example, search before making a decision allows the elimination of all dominated solutions and facilitates the selection of the non-dominated set. On the other hand, decision-making can also be made prior to a search by multiplying the objectives with different weights according to the order favoured by the decision makers. This early decision-making is suitable in a situation where the targeted market highly favours a particular criterion over the others. The importance of considering both aspects in multi-objective optimisation was described by Horn [81] in his three-category ordering for search and multi-criteria decision-making.

1. Making multi-criteria decisions before search

   This ordering is the most common approach to handling multi-objective optimisation. Generally, it involves the aggregation of multiple objectives into a single objective. Based on the objective, total solutions can be fully ordered and the solution that has a fitness value nearest to the objective is the preferred one. A few classic methods as used in [55] for implementing this kind of ordering are objective weighting and distance-to-target functions.

The objective weighting method is a linear combination of multiple objective functions into one objective function such as:

$$U(A) = w_0 b_0 + w_1 b_1 + w_2 b_2 \ldots w_{k-1} b_{k-1}$$

Each objective is given a weight or constant coefficient so it reflects the preferences of the designer on specific objectives. If all the objectives are given $w_i = 1$, an overall objective function is formulated with less conflict between objectives.

In the distance-target method, a target vector is selected as an ideal solution for each objective and each obtained solution is measured according to how far it is from this target. Based on this notion, the aggregation of all the objectives into a single objective can be defined as:

$$U_p(A, B) = (\sum_{i=0}^{k-1} |a_i - b_i|^p)^{\frac{1}{p}} \qquad p \geq 1$$

This single objective function is determined by two main factors; the selection of the target vector and the actual formula used for the metric (distance). The target vector must be carefully selected for each objective before the metric can be calculated. Arbitrary selection must be avoided as it might lead to a non-optimal solution. The actual formula too can affect the relation between solution points and their orderings. For example, *p=1* derives a linear relation while *p=2* adds non-linearity to the relation. Nevertheless, in both types of ordering, a decision maker must decide those related factors (such as weight, target vector) accurately before deriving the objective function.

2. Search before making multi-criteria decisions

   Search prior to multi-criteria decision-making has been proposed as a method to overcome the simplistic nature of multi-objective aggregation. Based on the Pareto-optimality concept, a decision maker has clear definitions of the criterion of a dominant solution and the non-dominated set of solutions.

The author refers interested readers to subsection 2.3.2 for a detailed explanation of this concept. Based on this concept, results from exploration in the problem space are interpolated in the solution space, which is embodied by multiple objectives. Figure 2.10 is an example of a solution space from which non-dominated solutions (a Pareto-optimal set) are determined based on their trade-offs between objectives. The objectives are defined according to the system requirements and do not require any weighting assignment beforehand. Previous work [17] applied this type of search and multi-criteria decision making to find task mapping for a NoC-based system.

3. Integrate both search and multi-criteria decision making

   In this category, both search and multi-criteria decision-making are integrated into a hybrid approach. Essentially, this involves a preliminary step of searching the possible trade-off points. At this step, a multi-criteria decision is made to limit the search space by selecting a trade-off point as a focused target for searching the solutions. Then, subsequently finding the solutions is based on the target as a reference point to direct the exploration. Fonsesca *et al.* [82] implemented a hybrid approach by applying MOGA [83] in the preliminary step to find the target trade-off point as well as in the subsequent search to find the solutions.

Multi-objective aggregation methods limits the number of solution to one in a single search. Furthermore, it is extremely sensitive to the accuracy of information that formulates it. A decision maker must have prior in-depth knowledge of all objectives to be able to determine the weight factor; in order to correctly reflect the degree of conflicts between those objectives. In case information is not available or is uncertain, the fixation on each factor will be based on a hunch, which could lead to an inaccurate estimation of each solution's appropriateness. On the other hand, search before making multi-criteria decisions through the single large population search to find the non-dominated set offers greater potential for implicitly parallel search than multiple independent searches of the former category. It would seem that non-dominated search is superior than the aggregative search, but in case when the actual non-dominated set is difficult to find, since the aggregative search concentrates in one direction in the search space, it might find one solution that is desired. Combination of both search and multi-criteria decision making inherits both advantages and disadvantages of the former two categories. It depends on a trade-off point as the focused target to limit the search space, but the reference point is not guaranteed to be the best direction for conducting the subsequent searches, unless the decision makers have prior knowledge of the direction.

2.3.4 Search Heuristics

In the real world, conflicting objectives are normally unavoidable, for example a system that is highly optimised for speed might consume more power and therefore cause higher heat dissipation. The algorithms commonly applied in combinatorial optimisation, such as Simulated Annealing (SA) [84], Tabu Search (TB) [85, 86] and Branch-and-Bound (BB) [87], could be used for the multi-objective optimisation problem if objectives are aggregated into a single criterion [81]. However, the aggregation of multiple objectives into a single-objective function imposes biases towards particular objectives. In fact, with conflicting objectives, a single best solution is rare in the solution space, since several incomparable solutions with a variety of trade-offs exist. For this reason, search heuristics that only depend on one search agent such as SA are rarely used in multi-objective optimisation. Since SA lacks the ability to provide simultaneous exploration of many solutions at the same time in one iteration cycle and only one best solution is produced at the end of every execution, the algorithm will be unlikely to produce solutions with different trade-offs unless multiple executions are performed. The Pareto-optimal set can be determined once a pool of solutions has been generated after completing many executions.

It is important to understand that in Pareto's concept of optimality, the search algorithm to be used is not explicitly specified. GA has the ability to provide unbiased search in partially-ordered spaces [81] and is able to optimise the solutions in the presence of conflicting objectives. The concept of GA was proposed between 1960 and 1970 by Holland [88] and his associates. This concept adopted the idea from evolutionary theory which explains how species evolve in nature. During species evolution, strong species have better opportunities to pass on their genes to create a new generation and weak species are faced with extinction. In the long period of time through several generations, strong species with good genes become dominant in the population. Multi-objective GAs [53, 56, 89] that could facilitate improvement of the members of the Pareto-optimal set are highly applicable in multi-objective optimisation.

In terms of efficiency in determining the Pareto-optimal set, GA is better than SA [90] because what can be produced by several executions of SA can be accomplished by one iteration cycle of a GA's execution. This is due to GA's ability to generate a large population of solutions to conduct simultaneous searches in the design space. Empirical comparison in [91] has shown that for multi-objective optimisation problems, SA had a tendency to find solutions in the Pareto-optimal set compared with GA when the size of the problem was small. However, when the size of the problem became larger, GA outperformed SA. As reported in [92], even for single-objective optimisation, GA was able to find approximately the same solutions in the Pareto-optimal set as were found by the SA and the Branch-and-Bound heuristic. Multi-objective GAs have many variations and among them the state-of-the-art GAs for multi-objective optimisation problem are NSGA-II [56], SPEA2 [53] and PAES [89]. Systematic comparison between multi-objective algorithms was conducted by Zitzler in his empirical study [93].

Although, GA has the ability to find multiple non-dominated solutions by searching different regions in the search space, it suffers from one main disadvantage due to the lack of information in determining the optimality of the best solution which it finds. Thus, it is hard to know when to stop searching. GA uses the number of generations to perform iterative operations and it stops at the specified number of generations. Furthermore, the concept of evolution underlying GA's main operations requires a population with a number of individuals in it to produce a better population than the previous generations. As a consequence, the algorithm consumes a large proportion of its execution time evaluating every individual in the population.

2.3.5 Summary

In this section, different aspects related to the search component of design space exploration were reviewed. This includes Pareto-optimal concept used in multi-objective optimisation to find good trade-offs between conflicting objectives, search and decision making criteria, and several search heuristics used in single and multi-objective optimisation.

2.4 Evaluation Techniques

Optimisation requires at least one fitness function to provide the necessary metric for evaluating prospects from the problem space, which have the potential to become the preferred solutions for the optimisation problem in hand. Based on the metric, the value of every prospect in the objective space can be determined, facilitating the selection of the best solutions that meet or come near to meeting the specified objectives. The two types of evaluation techniques in DSE; high-level simulation and analytical methods are explained in the following sections.

2.4.1 Simulation Techniques

The simulation technique is a popular and widely used tool for evaluating the performance of embedded systems at design stage. According to the standard definition by the IEEE [94], a simulation is

> *"A model that behaves or operates like a given system when provided with a set of controlled inputs".*

From this definition, a given system may refer to a real thing that has its own behaviours. A behaviour represents how the specific execution paths are carried out in the system and a simulation relies on an execution model to represent the behaviour of interest. Therefore, different models are required to simulate different types of behaviour. Given the right controlled inputs or stimulus [95], these models might execute hopefully as the intended typical working modes. However, if the stimulus is not correctly defined, the simulation's results may lead to wrong interpretation of the system performance. A simulation can be used for various objectives with different type of stimulus, for example, system designers can use the simulation technique to observe the characteristics of the system under normal or highly dynamic effects.

Highly complex embedded systems such as a flight control system are difficult to evaluate through a simulation due to the high complexity of its behaviours. This complexity is more manageable if a different simulation is used to represent a subset of behaviours in the system. In other words, the execution model of a simulator is defined according to a subset of the system which contains that part of the components which executes the functionality under investigation. For this purpose, a wide variety of Models of Computation (MoC) [96] has been developed to provide a thorough understanding of the whole performance of the system. A model of computation defines the underlying behaviour of how the components in the system work and their interaction with each other. By assuming that the system is constrained to a MoC, the required computation and communication resources can be efficiently evaluated and optimised during design. For example, an untimed MoC such as Synchronous Data Flow Graphs (SDF) [97] contains a set of processes that can be constructed according to an order but with the absence of timing properties. This type of MoC can be applied for the modelling and analysis of the data path and finding the optimal buffer sizes. A detailed discussion of different types of MoC can be found in [96] and the comparison between the features of several MoCs is discussed in [98].

Simulation techniques are widely used to support performance evaluation at mixed level of abstraction in funnel design [10]. As a system passes through the refinement process from high level to low level of abstraction, different simulation types are used to evaluate the system performance at each layer. However, the common pitfall in the simulation technique is the trade-off between simulation time and execution details [54]. This classic problem is very well known and has attracted attention from the research community [99, 100]. When the design process shifts to low level towards the implementation stage, the accuracy of performance evaluation is improved by adding implementation details. However, the execution of the simulation will takes longer to complete. A long execution time in simulation is not effective for early exploration in the design space at the high level of abstraction and therefore the application of low-level simulation is not desirable. On the other hand, reducing the implementation detail can decrease the time taken for the simulation but at a cost of low accuracy in the results. At the high level of abstraction, simulators with coarse-grained execution models providing faster evaluation with acceptable accuracy are desirable. These simulators can provide early insight on many design alternatives that are difficult to find at the late design stage. In the following subsections, the system level simulation techniques above Registers Transfer Level (RTL) for evaluating NoC's performance will be discussed.

2.4.2 Instruction Set Simulator

An Instruction Set Simulator (ISS) is a software tool which mimics a program running on the target hardware architecture, for example a processor. This tool is able to read the processor instructions and simulate their execution as closely as possible to the target machine. It provides virtual prototyping capability as a testing platform for software designed for the targeted processor that may not be available during the early stage of design. ISS simulation results provide information such as timing information (for example, the clock cycle that is useful for validating the software response time) and the internal values of the processor (for example, registers and memory for examining the execution of instructions). However, ISS suffers from overheads in terms of instruction decoding, functional operation and instruction scheduling. Several works [101, 102, 103] have suggested ways to improve ISS simulation time to cater for the increasing complexity of the architecture and the pressure time to market. A commonly used ISS tool such as SimpleScalar [104] supports Instruction Set Architectures (ISA) such as the ARM and PowerPC.

ISS is one of the main components in hardware/software co-simulation [105] and it plays an important role in the development of heterogeneous platforms such as MPSoC. Early integration [106, 107, 108, 109] of ISS with a SystemC simulation model facilitated validation of overall system-level performance of MPSoC. In those works [106, 107] SimpleScalar ISS was integrated into the MPSoC model with bus-based communication. Both use the same approach which utilises the

GNU-Debugger-Interface for the communication between the ISS and the SystemC simulation model. Wieferink *et al.* [108] proposed ISS integration with the NoC simulation framework. Their work was an extension of previous work proposed by [110] whose developed NoC simulation framework based on the TLM concept and implementation was carried out with the SystemC library to measure the network latency and the throughput performance of NoC. This integration makes ISS as a good substitute for abstract processor modules to further evaluate the various impacts which the processor components have on NoC, for example various traffic patterns caused by instruction caches in the processor. In this approach, ISS is generated by modelling the customised processor with LISA [111], an Architecture Description Language (ADL) which can describe the processor architecture itself and also automatically generate software development tools (such as compiler and linker) as well as HDL description. In [109], SimpleScalar ISS was used similar to [106, 107] but integration with the SystemC simulation model was implemented with shared memory and Memory Mapped IO (MMIO). This approach offered greater improvement than [108] by enabling the usage of different routing algorithms.

ISS-based simulation is useful if the final application software is available and the target architecture is customisable. Development tools such as compiler and linker specific to the processor are also required to generate object code from the application in order to develop the ISS simulation model. Even if these tools are available, the application may requires recompilation to generate new object code after bug fixing. This can be prohibitive for early design space exploration if significant amounts of time and modelling efforts are required to design a working ISS simulation model instead of exploring alternative solutions.

NoC itself has many parameters which might have various impacts on the performance of MPSoC. Integration with ISS is relevant to investigate more deeply the further impact of the processor components on the NoC architecture when preceded by early exploration of various alternative solutions of NoC at the system level. This exploration can be performed through the separation of concerns whereby network performance is measured in the absence of high computation details and even without the presence of low-level communication signals. Possible alternatives can be explored more quickly, which provides the ability to reveal many potential solutions for NoC. This type of simulation is called abstract performance simulation [10].

2.4.3 Abstract Performance Simulation

Enabling faster simulation execution with reasonable accuracy of results is one of the reasons why abstract performance simulation was developed to evaluate systems at the high level of abstraction. In the context of communication, the Transaction-Level Modelling (TLM) [79] concept offers system designers a way to realise this type of simulation. Through this concept, low-level on-chip communication operations are abstracted using high-level read/write function calls. This is relevant at the early design stage since low-level communication operations are unnecessary and often not available. Several benefits can be reaped from this concept not only in terms of fast simulation speed but also the ease of programming, which allows quick development of simulation models as well as reducing modelling efforts significantly. Many previous works [110, 112, 113] utilised this concept in SystemC or Java to implement this type of simulation.

Kogel *et al.* [110] proposed TLM based network simulation for systems on-chip with complex and heterogeneous communication schemes such as point-to-point, bus and crossbar. The simulator supports cycle-accurate network performance evaluation and timed simulation is implemented by annotating execution delays to the network infrastructure. The simulation speed is further improved by increasing data abstraction to Abstract Data Type (ADT) level. ADT is a data granularity representation of the sets of functionally associated data. The executable model on which the simulation runs consists of a NoC channel, master module(s), slave modules and network engines. Various types of interconnect can be plugged in to the NoC channel as network engine modules. During a simulation run, the master module initiates transactions by calling the communication services of the NoC channel. Several processes in the NoC channel are responsible for the processing and delivery of the received transactions to the destination.

Both throughput and average latency are popular metrics widely used for NoC performance analysis. Area is another important metric for MPSoC not only for achieving a sleek and portable design for small devices, but also for reducing the amount of power needed, which can be gained by minimising the number of gates. Pestana *et al.* [112] proposed a simulation technique for providing analysis in terms of throughput, average latency and area which allowed investigation of the impact of the cost-performance trade-off of a NoC design. As in [110], the simulator was developed with SystemC based on the TLM principles. The XML files used in the simulation framework allowed the parameterisation of several NoC parameters such as the topology, mapping and connection. The application was modelled with several traffic generators connected to the NI as the master modules in order to generate synthetic workload for simulating temporal and spatial traffic distributions. The analysis of NoC's area assumed that the total area was comprised of only the router and the NI, and wire was assumed to have zero-cost.

The design of NoC architecture may vary according to different configurations of parameters such as topology, routing algorithm, flow control and virtual channels. Various schemes of NoC architecture creating different levels of complexity emerged from the combination of these parameters. One of these schemes combined an XY routing, a Mesh topology and wormhole switching as the widely adopted a NoC architecture. An example of a NoC implementation which adopts this scheme is HERMES [36]. Based on this implementation model, a TLM-based simulation technique was proposed by [113]. The applied executable model underlying this simulation contains several cores which are connected to a single channel. At any time, cores can be either a sender or a receiver of packets. For every packet header received by the channel, a transaction is created and kept alive until it has arrived at the target core. The total latency represents the transaction lifetime of the transmitted packet, which is calculated by also considering the concurrence of several flows competing for the same network resource, hence improving the accuracy of the performance value. In addition, the simulation benefits in terms of reduced execution time from the underlying algorithm. The algorithm was developed based on wait-for-event, that is, the system is simulated only when events occur, for example the arrival of a new packet, and not as frequently as every clock cycle. However, when the number of hops exceeds the total count of flits, the simulation result may become too pessimistic. In other words, the next packet can transmit only when the previous packet occupying the route has completely arrived at the target destination. This excludes the true wormhole switching implementation whereby flits of a flow undergo 'shrinking' and 'growing' phases. The analysis may compute a latency figure that is too pessimistic, as the consequence of the situation not happening in reality is considered.

A TLM-based simulation in SystemC is not the only the way to develop abstract performance simulation. In [114], the author proposed a Java-based cycle-accurate simulation approach which operates at flit-level. The approach was based on the assumption that the NoC infrastructure is comprised of two main components: the switch and the link. Switch implementation was based on a pipeline model which consists of three stages which are input buffering, routing/arbitration and output buffering. In each operation, flits queuing inside the communication queues and flit transmission through the output ports as well as the link both have a cost associated with them. By annotating performance cost to the IP model and NoC infrastructure, end-to-end average network performance can be calculated upon the arrival of flits at the input port of destination IP. However, in the simulation model, flit transmissions work on the first-come-first-served basis, and complex interactions such as interference between flows are lacking in the model. In systems in which such interference exists due to a pre-emptive scheduling policy, the use of shared resources is prioritised to provide particular traffic flows with guaranteed accesses to shared resources regardless of the arrival sequences of those flows. Furthermore, full propagation of the data stream through all stages in the pipeline can lead to unnecessary increases in latency as not all flows must perform all the stages, that is, higher priority flows are not pre-empted and immediately after a full path has been assigned flits can be transmitted instantaneously from the output port. The lack of support in the modelling of contention between flows makes this kind of systems too optimistic, and thus limits its use for systems with guaranteed services.

The author in [115] proposed a high-level simulator implementation similar to [114]. This simulator accepts a NoC configuration as input which contains network-related information such as topology, connected modules, modules' type and their logical position in NoC. Modules can be defined as routers, interfaces or links with correspondence cost unit such as delay, power consumption and area. Network-related behaviour is modelled by the packet injection rate configuration of processors. Then, performance metrics are calculated by summation of cost when packets traverse the network. However, the accuracy of the average performance results depend on the stability of the average value defined for each module.

As proposed in [116], faster simulation time can be achieved by reducing the details of the architectural and behavioural aspects yet within the acceptable range of accuracy to maintain its usefulness. In this simulation model, routers were implemented to avoid sending flit-by-flit data transmission, by using a payload abstraction technique to send data between consumers and producers. The payload abstraction technique only requires packet modelling in two structures: the header and the trailer. During packet transmission, data payload is included inside the header whereas the trailer only contains information such as evaluation parameters. Therefore, the transmissions along the hops only involve the releasing and receiving of headers and trailers only; flit-by-flit transmission is omitted. Based on wormhole packet switching, the trailer follows the header along the reserved transmission path for the packet. The release timing of the trailer is calculated from an analytical method that relies on the header release time and the trailer is released as soon as the calculated release time of the trailer arrives. However, in wormhole packet switching, a flow can be blocked by other flows with higher priorities and due to this the release time of the flow's header can be delayed significantly. Despite a gain in faster simulation time, the model of computation used by the simulator is susceptible to blocking scenarios, causing inaccurate evaluation results due to the long delay of a header's release. This is because the distance between a header and its trailer becomes closer when such scenarios occur, which is less reflective of the real wormhole packet switching NoCs.

The abstract performance simulation technique is suitable for estimating the average performance of NoC-based systems and its accuracy relies on specific features such as the stability of the approximated costs defined in the NoC infrastructures [110], the network operations [114] or the payload abstraction [116]. However, the difficulty in producing traffic patterns that trigger the worst-case scenario of NoC-based hard real-time systems imposes a limitation on the use of the technique as a performance evaluation tool. In this case, analytical methods are usually preferred as the evaluation approach since the worst-case scenario must be considered to calculate the upper-bound performance for analysing the schedulability of the systems.

2.4.4 Analytical Methods

In general, analytical methods are derived from constrained models by assuming that particular conditions are true but relaxing or disregarding others. In these idealised conditions, the complexity of the details can be managed properly, simplifying the derivation of evaluation models or formulas for evaluation purposes. Many analytical methods for NoC are targeted for evaluating average cases, utilising frequently-used performance metrics such as the average latency of packets, network throughput or bandwidth. It is a common approach to use simplified assumptions such as the number of hops to derive the performance metrics [50, 51], although other conditions may also add further delays to the latency such as the concurrence between traffic flows on the same network resources. For example, between two flows with different priority levels, the low-priority flow is pre-empted according to a priority pre-emptive arbitration policy to give the high-priority flow guaranteed access to the shared resources. As a result, the low-priority flow normally suffers enormous delays which affect its overall response time to meet the deadline. Based on the hop count, the evaluation model may be less complex to apply easily in optimisation, but it suffers from simplistic assumptions which might affect the accuracy of the evaluation.

Other analytical methods take into consideration other aspects of the network conditions, such as queuing delays in NoC routers [117]. The router model is simple; it contains four input port channels each with a buffer and a crossbar switch interconnects them together. Packets arriving at a channel are stored in the channel's buffer and processed on first-come-first-served basis. The router model depends on the Poisson process to compute the occurrence of the packet header arrival (not including the packet body) within a specific time length. This process requires an average arrival rate of header flits at the channels for computing random header arrival events following exponential distribution. Based on this router model, the average number of packets at each buffer can be calculated, supporting thorough performance analysis including average buffer utilisation, overall average packet latency and network throughput. Incoming packets are easily delayed by the existing packets in the buffer as well as by other packets in other buffers which arrived before the incoming packets, hence the approach is suitable for average cases when a low or medium traffic condition is present. However, the accuracy of the average packet latency decreases when the network throughput reaches saturation point due to the degradation of the Poisson process.

Using average metrics based on the number of hops such as average packet latency and network throughput is insufficient to evaluate the schedulability of the systems in any scenario. Instead, the worst-case upper bounds of task response time and message network latency are needed for this purpose. In hard real-time analyses, one of the ways to analyse the schedulability of systems is by comparison between the upper bounds and the deadlines of task and messages; in this way the percentage of tasks and messages that are schedulable is determinable.

2.4.5 Real-Time Analysis

Hard real-time system requirements dictate that stringent timing constraints must be met for delivering guaranteed services. Average case evaluation techniques yielding commonly-used performance metrics are inadequate for validating the timing constraints of the systems. Metrics of this kind, although useful for best-effort systems, are inapplicable when the schedulability of every task and message is concerned and must be analysed to ensure that the requirements are met. The upper bounds of the response times of each task and message are needed for an accurate evaluation of the systems; hence evaluation techniques based on real-time analysis are more suitable for this purpose.

In hard real-time multi-processor systems with priority pre-emptive scheduling, two main problems exist: task allocation and priority assignment. The availability of the well-known scheduling techniques and analyses of single processors is an advantage for multi-processor scheduling which follows the partitioning approach. In partitioned multi-processor scheduling, each core is assumed to have its own queue to schedule tasks in that core, hence task or job migration between cores can be avoided. The RM scheduling algorithm has been proved by Liu and Layland [64] to be the optimal priority assignment policy for synchronous periodic and sporadic task sets running on a single processor, with every task having its period equivalent to the deadline. For similar task sets but with a deadline less than or equal to the period, Deadline Monotonic (DM) [118] becomes the optimal priority assignment policy in a single processor. However, RM and DM optimality fails for a task set when the deadline of each task exceeds its period [119]. If such an assignment that can schedule all tasks with an arbitrary deadline exists, the priority assignment algorithm proposed by [120] is known to be able to find it.

Task allocation problem in multi-processor scheduling is known to be NP-hard [49] because tasks can be allocated to processors in a number of ways and the number can grow exponentially depending on the size of task set and the number of processors. Given the total number of tasks is T and the number of processors is P, the total number of ways task allocation can be performed is P^T. Consider an example of a 2D-Mesh NoC in which has 16 processors and a task set with 33 tasks, the number of ways those tasks can be allocated to the processors is $5.44x10^39$. Therefore, it is challenging to find a feasible task mapping in polynomial time through an exhaustive search.

Early works on partitioned multi-processor scheduling with fixed-priority task sets considered RM to be the priority assignment [121, 122], whilst task allocation is performed using bin-packing algorithms such as First Fit (FF), Best Fit (BF) or Worst Fit (WF). This is because task allocation problem is synonym with a bin-packing problem: allocating tasks to a processor until it reaches its full utilisation is similar with stacking items into a bin until it is full. Others used search heuristics such as Simulated Annealing [49] and Branch-and-Bound [123] to allocate tasks onto processors. Hereafter, a fixed-priority task set with a deadline equal to period is known as an implicit-deadline task set.

Several tests associated with partitioned multi-processor scheduling for implicit-deadline task sets have been used to validate the improvement made by those algorithms. A classic schedulability metric known as the approximation ratio computes a ratio between the number of processors required to schedule a task set with a given scheduling algorithm and the number of processors required by the optimal scheduling algorithm. The lower the approximation ratio of an algorithm, the better it is than the existing algorithms. For example, the Rate Monotonic Matching (RMM) algorithm [124] with a 3/2 ratio is better than the previous best approximation ratio of 7/4 of the Rate Monotonic General Task (RMGT) algorithm [125]. However, the applicability of this metric as a schedulability test is severely limited for several reasons. First, because finding the minimum number of processors in the optimal case is an NP-hard problem and obviously it becomes harder with a growing number of ways the task set can be allocated as the number of processors is increased to accommodate the larger size of task set. Second, the approximation ratio test only holds if the number of processors in the optimal case grows towards infinity. Third, the utilisation bound of the algorithms tends to become too pessimistic [126].

Utilisation bounds are metrics for addressing the difficulty confronted by partitioned multi-processor scheduling algorithms when scheduling an implicit-deadline task set with large utilisation. The worst-case utilisation bound of a scheduling algorithm is defined as the minimum utilisation of a task set that is just schedulable according to the scheduling algorithm. Any utilisation that a task set has, as long as it is below or equivalent to the bound will be schedulable. The importance of utilisation bounds for scheduling algorithms has seen a variety of utilisation functions proposed since [127], for example targeting specific types of task allocated to the same processor [125] or targeting scheduling algorithms that are based on bin-packing heuristics such as BF, FF or WF [128, 129]. An utilisation bound of a scheduling algorithm provides a simple sufficient test to determine whether a task set is schedulable under the scheduling of the algorithm. It is sufficient to imply that the task set cannot be unschedulable depending on the scheduling algorithm if its utilisation is below or equivalent to the utilisation bound. Conversely, the task set may have utilisation above the utilisation bound, but it is not necessarily unschedulable according to the other scheduling algorithms. Therefore, it is categorised as a sufficient but not necessary test.

As an alternative to approximation ratio and utilisation bounds, a metric based on the number of tasks that are deemed schedulable provides a comparative measure for determining the effectiveness of scheduling algorithms. Typically, this number is compared with the total number of tasks in the task set. Ideally, an unbiased task set generation algorithm is desirable to generate the task sets.

In multi-processor systems, dependency between tasks demands a reliable network to communicate data. Similar to processors, a network is a shared resource and contention between messages will typically occur. Consequently, message transmissions will experience network delays when the network bandwidth gradually becomes saturated. The effects appear as a severe increase in message latency, as well as in the response time of the dependent tasks. In partitioned multi-processor scheduling algorithms, the communication aspect is excluded to reduce the complexity of the analysis. The importance of addressing the interdependent relationship between real-time tasks and messages caused Tindel *et al.* [130] to propose an 'holistic' schedulability analysis, which included integrated scheduling of both processor and communication under the assumption that messages are not pre-emptive processes executed on a Time Division Multiple Access (TDMA) supported bus. The holistic approach [130] was derived from the classic scheduling theory of distributed systems, enabling end-to-end response time analysis of multi-processor system. However, the complexity of the analysis grows proportionally with the system size and the number of dependencies between components. In different works, researchers have considered other communication protocols to bound communication delays in their schedulability analyses, such as the Time-Triggered Protocol (TTP) [131], Asynchronous Transfer Mode (ATM) [132] and the Controller Area Network (CAN) protocol [133]. This provides a variety of analyses to choose which suits the specific type of bus communication protocol. NoC has a different network infrastructure from bus, for example routing algorithm, flow control mechanism, switch crossbar and others, hence those schedulability analyses may not be suitable for hard real-time multi-processor systems that use NoC as the communication network.

End-to-end timing analysis based on the schedulability analysis technique known as SymTA/s [134] adopted the compositional method as the underlying approach. The difference between the compositional method and holistic analysis is that the former approach is well structured with respect to the architecture, which improves the understanding of the complex dependencies in the system. Based on the event stream models that describe the possible I/O timing of tasks in IP components, an output event stream of one component becomes the input event stream of the other component, enabling the individual timing of components to be calculated. After this, component-wise local analysis can be performed from the existing schedulability analysis techniques using the properties of the event streams such as period or jitter. However, this technique is intended for systems which use a bus communication and the communication paradigm in a bus is vastly different from NoC, which contains various configurable parameters such as topology, routing and packet arbitration control, hence the approach is less suitable for NoC performance analysis.

Meeting the timing constraints of communication is crucial to the reliability of NoC-based hard real-time systems. With interdependent tasks, non-delay execution on processors is half of the timing constraints that must be met; the rest is relying on NoC to deliver the messages without delays. Several works have suggested providing schedulability analyses to bound the network latency of packets in NoC. Shi *et al.* analysis [135] targeted priority pre-emptive wormhole switching in the NoC architecture. The notion used in this analysis is similar to an analogy of tasks running on shared processors, but with tasks replaced by traffic flows running on NoC's shared routing paths. Arbitration control of the system is regulated by the priority pre-emptive policy, creating interference that affects the latency of particular flows, especially those with lower priorities. To provide tight upper-bound latency, timing characteristics such as release jitter is also considered in the analysis. The analysis is still an early work for hard real-time NoC; hence it is constrained by specific assumptions in its ability to ease the complexity of the NoC model. For example, the complete routing path allocated to a traffic flow is blocked for the whole duration until all of its flits have arrived at the destination. In the real concept of wormhole switching NoC, a flow undergoes shrinking and expanding phases, hence part of the links in the routing path will be unblocked as soon as the previous flits have been transmitted to allow other flows to utilise them. As a result, the upper bound latency yielded by the analysis may be a little pessimistic due to the longer blocking time of the previous flows. Nevertheless, the analysis provides an opportunity to explore the design space of NoC from the corner-case point of view and is a good starting point for providing total schedulability analysis of the systems. Recently, the analysis has been applied as a fitness function to find mapping solutions [11].

Several extensions to this work have been proposed to include upper-bounds for buffers [136] and end-to-end response time of tasks [137, 138]. The analysis in [136] provides tight bounds, but they assume that buffers are arbitrarily large to avoid the back-pressure problem (that is, when a virtual channel has the highest priority over a link, but has no credits because the buffer down the line is full), hence it may lead to high area and energy dissipation due to large buffers. To avoid the assumption, the schedulability analysis proposed by Indrusiak [138] took the previous work [135] as the foundation to create end-to-end response time analysis for tasks in NoC-interconnected multi-processor hard real-time systems.

2.4.6 Summary

Evaluation techniques are one of the main components in early design space exploration. During task mapping exploration, task mappings must be evaluated to determine their feasibility for the system in design. Exploring as many as task mappings during early design space exploration requires efficient evaluation techniques, without such techniques significant amount of time will be consumed to evaluate many solutions. In this section, different types of evaluation techniques and their suitability for evaluating the fitness of task mapping were reviewed. Real-time analysis is a suitable approach for analysing the schedulability of tasks and messages for a given task mapping compared with simulation techniques or analytical methods that yield average metrics.

2.5 Summary

This chapter reviews previous research works related to NoC, task mapping, design space exploration and evaluation techniques. The scope of the review focuses on task mapping exploration of hard real-time systems with NoC as the interconnection. From the previous works, state-of-the-art techniques lack the support to find task mapping for this kind of systems. Most of the techniques rely on average metrics such as average latency and bandwidth constraints to optimise task mapping in average case. This kind of metrics is not sufficient to analyse the schedulability of every task and message in worst case. Some researchers have performed task mapping for NoC-based hard real-time systems but only consider single-objective optimisation and messages schedulability. This motivates the proposal of new techniques to address different aspects of task mapping optimisation problems. In the next chapter, a new technique is introduced to address the interference experienced by low-priority tasks and messages in finding schedulable task mapping for NoC-based hard real-time systems with priority pre-emptive scheduling.

Chapter 3

Task Mapping and Priority Assignment Optimisation

At system level design, task mapping is a process of combining an application and a platform together to create a complete system. It determines the allocation of tasks on the processing cores of the platform. Furthermore, it also affects how messages are routed over the NoC employing static routing policy. In a hard real-time system based on NoC, applying a fixed priority pre-emptive scheduling policy is one way to resolve contention between tasks and messages, making their behaviour more predictable for analysis. This scheduling policy enables pre-emption on certain tasks and messages based on the priority levels assigned to them. As the result, the response times of low-priority tasks and messages are delayed to allow access for their high-priority counterparts.

Given a task mapping, one or more tasks may be allocated to the same core, and if these tasks need to communicate with the other task at a different core, the outgoing messages are routed through the same links between the cores. With this task mapping, low-priority tasks and messages may experience high interference, which delays their response times. Substantial delays in the response time leads to missed deadlines in the system, and the result of this is potentially severe consequences to human life. To address this problem, this chapter introduces an approach that allows simultaneous optimisation of both task mapping and priority assignment. The main idea is to explore a priority assignment which could lessen the interference of low-priority tasks and messages of a given task mapping while guaranteeing access to shared resources for their high-priority counterparts.

In this chapter, section 3.2 explains the system model. This is followed by section 3.3 which explains the schedulability analysis used for evaluating every task and message. The algorithm used for the optimisation process is described in section 3.4. Section 3.8 discusses the results of the experiment and this chapter concludes with a summary of the proposed approach in section 3.9.

3.1 Interference on Low-Priority Messages

One possible reason why a NoC-based hard real-time embedded system with a fixed priority pre-emptive policy is unschedulable is the overwhelming delays which cause the response times of tasks and messages to exceed their deadlines. Every task mapping allocates tasks differently on a platform. Sometimes, several high-priority tasks are mapped on the same computing resource and this creates high interference which low priority tasks cannot accept. Again, messages that are sent by the latter tasks may experience the same interference from the high-priority messages, since both type of messages share the same links near the resource. The response times of the tasks and messages will be easily affected by the interference; in a worst-case scenario their deadlines could be missed.

The following example illustrates how a given task mapping (see Figure 3.1) failed to make all tasks schedulable in a system that employed a fixed priority scheduling policy such as Rate Monotonic (RM) scheduling [62]. With RM scheduling, tasks are assigned with priorities according to their period. Based on the properties shown in Table 3.1, assuming the priority ordering is $\tau_1 > \tau_2 > \tau_3$, task τ_3 with the longest period receives the lowest priority among the tasks. In this example, it is assumed that the same ordering applies to the messages as well. Since each task is separately allocated to a different core, the latency of messages has more direct influence on the end-to-end response time of tasks. Based on the task mapping in Figure 3.1, message F_3 receives interference directly from message F_2. Since message F_2 blocks the link it shares with message F_3 while waiting for message F_1 to completely arrive at its destination, message F_1 is said to have indirect interference on message F_3.

Table 3.1: Task and message properties

Task	WCET[1]	Message	Basic latency[2]	Period (deadline)[3]
τ_1	1	F_1	1	3.2
τ_2	1.25	F_2	1.15	3.5
τ_3	1.50	F_3	1.25	3.8
τ_4	1	-	-	3.0

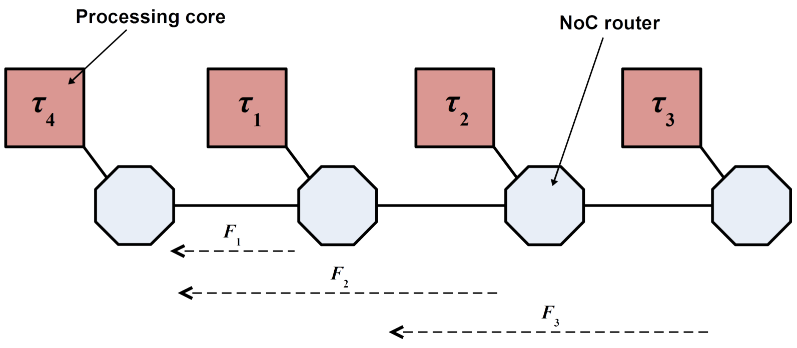
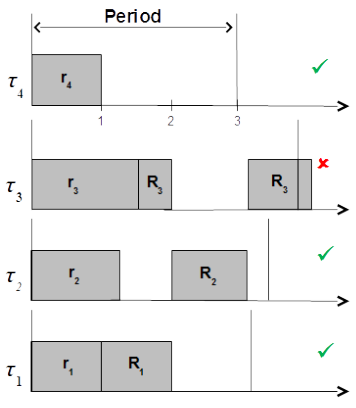
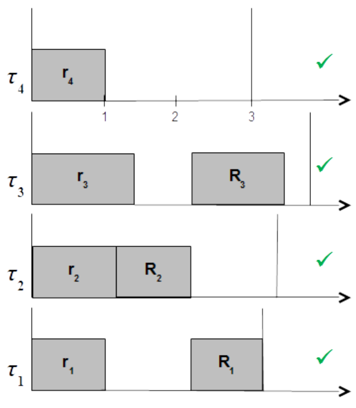


Figure 3.1: Task mapping of a NoC based multicore system



(a) $\tau_4 > \tau_1 > \tau_2 > \tau_3$



(b) $\tau_4 > \tau_2 > \tau_1 > \tau_3$

Figure 3.2: Task i response time (r_i) and its message latency (R_i) with two different priority orderings

[1]Worst-case execution time of task.
[2]Maximum latency of a message travelling via a route on its own.
[3]Assume deadline equivalent to the period of task.

If the scheduling policy [62] is referred to as optimal, a scheduling that makes all the tasks in the system schedulable will always be found whenever such a scheduling exists. However, as shown by the example in Figure 3.2a, the resulting priority assignment makes the system unschedulable, for example, task τ_3 has failed to meet its deadline. If a different priority ordering is used such as $\tau_2 > \tau_1 > \tau_3$ on the same task mapping, not a single task misses its deadline, as depicted in Figure 3.2b. By changing the priority ordering, the amount of interference received by low-priority tasks can be reduced and at the same time high priority tasks are allowed to have access to the resources. From the illustration discussed above, it can be understood that the restriction imposed by the fixed priority pre-emptive scheduling policy limits the potential of the task mapping to become feasible for the system.

3.2 System Model

In the design methodology based on Y-chart [74] shown in Figure 1.2, the mapping step combines separated aspects of design [73] between a set of application functions and a hardware architecture. The application functions impose specific computation and communication loads on the system, which can be described in an application model as design-time characteristics. For the hardware architecture, its characteristics are described in a hardware model.

The application comprises a set of n hard real-time tasks $\Gamma = \{\tau_1, \tau_2, \tau_3, ..., \tau_n\}$. It is assumed that a task is periodically (or sporadically) activated and independent of other tasks. With this assumption, a task may send a message to other task, but the execution of the receiving task is not dependent on the arrival of the message. We assume that the end-to-end deadline of a task is equivalent to its period. In detail, the deadline is the timing requirement for the task to complete its execution and for sending a message if the task has a connection with other task on a different core.

For analysing the end-to-end response time of a task, its Worst-Case Execution Time (WCET) c_i [68] must be determined in advance. Applying techniques such as path analysis and profiling can determine the WCET of every task, but the specifics are beyond the scope of this current study. It is therefore assumed that the quantitative analysis of this property has been completed and is available prior to the task mapping step. The operating frequency of processor producing the WCET is known as the nominal frequency.

A task may be shared with other tasks in a single core depending on a given task mapping which allocates the tasks on the hardware platform. The execution of the tasks on the core can be scheduled in a predictable way according to a fixed priority pre-emptive scheduling policy[139]. This kind of policies rely on the priority level of each task to distinguish between the tasks which one should be executed first and which one should be pre-empted if these tasks shared the same resources. The priority assignment can be performed statically with either a random assignment or according to specific policy such as the RM priority assignment [62]. This tuple lists the properties related to task i.
$\tau_i = \{c_i, t_i, p_i, d_i\}$

c_i	worst-case execution time of task i
t_i	period of task i
p_i	priority level of task i
d_i	end-to-end deadline of task i

Two tasks, each is allocated at a different core, communicate with each other by exchanging messages. For meeting the hard real-time requirements, the delivery of a message must be completed before or at the end-to-end deadline of the source task. It is assumed that the message Msg_i is transferred as soon as the execution of the source task is complete. Based on this assumption, the message is only released after the data that needs to be delivered is available in a complete form and not half way during the execution of source task. Therefore, we ignore the overhead of NI for processing the data as the processing cores are directly connected to the routers. In reality, however, a message can be sent as soon as data is available for transmission and the processing performed by NI may also delay the end-to-end response time. As a result of these simplification, our system model is considered as pessimistic.

Before transmission, every message is packetised, producing one or several packets depending on the size of its payload. A series of packets sent over the

NoC from a source to a destination creates traffic flows. A message with a large number of packets requires more than one traffic flow to complete data transfer. A traffic flow is released periodically (or sporadically) and independent of other flows; it is transmitted subsequently after the arrival of the preceding traffic flow at the destination. Several traffic flows from different sources share the same communication resources (such as routers and links) if a part of or all their routes are similar. In order to prioritise the traffic flows in shared communication resources, we assume that the priority level of the corresponding traffic flow is inherited from the sending task. The maximum packet size for each message is assumed has been determined before mapping.

Each traffic flow has basic latency [135]: the latency of a flow when no other interference exists on its path between the source and destination. For most NoCs, basic latency can be deterministically found and it is a function of the number of hops, the number of flits of the packet, the time needed for a flit to traverse a link and the time needed for a router to route and arbitrate packet headers. Each flow's basic latency is measured based on the nominal operating frequency at which the NoC is operated.

All the related properties for each flow i can be summarised in the following tuple.

$Msg_i = \{So_i, De_i, P_i, L_i, C_i\}$

So_i	source task
De_i	destination task
P_i	priority level of message i
L_i	maximum packet size of message i
C_i	basic latency of message i

The platform is modelled to represent a 2D Mesh architecture of an on-chip multicore system, which uses a NoC as the interconnection. Although, our evaluation model does not restrict the use of other types of topology, the 2D Mesh topology is chosen as our platform model because it is a typical topology used to model a NoC platform due to its simplicity. Therefore, it helps reducing the complexity of the hardware platform when the NoC is implemented with the static routing policy such as X-Y routing.

It is assumed that the platform is homogeneous and contains a number of processing cores. As in most multicore systems, tasks share the same computation resources, and hence a processing core could have one or more tasks run on it. Based on partitioned scheduling of multi-processor [63], each processor is assumed as having its own run queue rather than a single global queue [67]. This enforces each task to run only on a single processor, avoiding additional communication loads as the result of resuming a task's job from one processor to another. Since a task is independent and runs only on a processor, it can only be delayed by the other tasks on the same processor.

The NoC contains different types of components and the interested reader is directed to section 2.1 for a more detailed explanation. Likewise tasks, messages share the same communication resources, that is, a NoC's link is used to route several packets from different routers. Links between nodes are bidirectional and contain uniform bandwidth. It is assumed that each traffic flow has an exclusive virtual channel assigned to it at each port of a router. Therefore, the number of virtual channels at each port must be sufficient to support all priority levels, so that blocking due to unavailable virtual channels does not happen.

Following a deterministic routing scheme (for example, XY routing), the NoC's routers enforce static routes between sources and destinations for forwarding packets. Packets are forwarded between routers based on wormhole packet switching, which requires a packet to be split into smaller communication data called flits. The transmission of a packet flits begins with the header flit which is then followed by the payload, one after another until all flits have been transmitted. The priority pre-emptive arbitration unit arbitrates the access of flows on shared links in accordance with their priority levels. At any time, if several packets contending for a link, only the highest priority packet among them will be forwarded through the link if the buffer in the downstream router is enough to accommodate the packet's flit.

3.3 End-to-end Response Time Analysis

The main metric that we used for evaluating the schedulability of hard real-time system is the number of unschedulable tasks and messages. This metric can be calculated by comparing the response time upper bound of each task and message with their deadlines. Since the 1960s many RTAs have been proposed, providing an abundance of techniques for analysing the schedulability of hard real-time systems. For our proposed approaches, the analysis of the end-to-end response time upper bound of task, in which includes the worst case response time of task and the worst-case latency of message, is needed to calculate the metric as the fitness of task mapping. For this purpose, an extension is proposed based on the previous works [135, 140]. The end-to-end response time upper bound is defined as the time since the task is released until the last packet that it sends arrives at the receiving task in a worst-case scenario. It is assumed that a task is schedulable if its response time upper bound does not exceed its deadline.

The proposed analysis uses the timing properties listed in both tuples in section 3.2. Before the end-to-end response time analysis is derived, the related variables for the analysis are described in further details as follows:

- c_i : The worst-case execution time required by task τ_i on each of its releases. It is the maximum time that the task can take to finish execution when running on its own over a processing core.
- t_i : The minimum inter-arrival interval between two consecutive releases of the task τ_i , hereafter called the period of the task.
- d_i : The deadline requirement of task τ_i to complete its execution since a release of the task.
- $hp(i)$: The set of high-priority tasks that share a computing resource with task τ_i and could pre-empt it.
- r_i : The worst-case response time of task τ_i calculated since a release of the task.

- C_i : The basic latency required by traffic flow F_i on each of its releases. It is the time taken by the flow to arrive when no other interference exists on its path between the source and destination.
- T_i : The minimum inter-arrival interval of flow F_i, hereafter called the period of the flow, is the time between two consecutive releases of packet i.
- Q_i : The deadline requirement of flow F_i to complete its transmission since a release of the flow.
- S_i^D : The set in which contains the high-priority traffic flows that share one or more links with the observed flow F_i and could pre-empt it, causing direct interference to the flow.
- J_j^R : The release jitter of the observed flow F_i is the maximum time the flow can wait for release after arrival.
- J_j^I : The interference jitter of the observed flow F_i is the indirect interference experienced by the flow, as the result from the direct interference imposed by the high-priority flows on flow F_j, where $F_j \in S_i^D$.
- R_i : The worst-case latency of the observed flow F_i calculated since the release of the flow.

The hard real-time system based on NoC applies priority pre-emptive scheduling, and hence high-priority tasks are given guaranteed access to the shared computing resources over their low-priority counterparts. However, pre-emptive actions create interference and add delays to the response time of tasks with low priority levels. The interference may contribute largely to the end-to-end response time delays, and this factor is taken into account in the proposed analysis.

First, we analyse the worst-case response time of a task in a single processor. From the classic schedulability analysis [140], equation (3.1) calculates the worst-case response time (r_i) of task i. If for each task i of a task set, $r_i \leq d_i$, then the task set is deemed schedulable on the processor, where r_i and d_i are the response time and deadline of task i respectively. The first term of this equation refers to the worst-case execution time (c_i) of task i. The second term refers to the maximum interference experienced by task i, which is coming from high-priority tasks in the interference set ($hp(i)$), that is, tasks which share the same resource as task i.

$$r_i^{n+1} = c_i + \sum_{\forall Task_j \in hp(i)} \left\lceil \frac{r_i^n}{t_j} \right\rceil c_j \tag{3.1}$$

Similar to tasks, in a hard real-time embedded NoC, traffic flows are assigned with fixed priorities and packets which belong to a traffic flow inherit the same priority as the flow. Priority pre-emptive scheduling imposes direct and indirect interference on some packets [135], especially those with low priority levels, as the result of providing the high-priority flows with guaranteed access to links.

The definition of direct interference is rather straightforward; it is the interference imposed by a traffic flow with higher priority than the other flow with lower priority, and together they share at least one physical link. From Figure 3.3, two flows F_i and F_j is in a direct competing relationship if both flows meet the condition $P_i > P_j$ and has at least one common physical link. For F_j, a direct interference set S_j^D is defined as the group of high priority flows that meet these conditions, $S_j^D = \{F_i\}$. In the example in Figure 3.1, traffic flows F_1, F_2, F_3 meet the condition $P_1 > P_2 > P_3$. Traffic flows F_2 and F_3 are in a direct competing relationship but flow F_1 is excluded since it does not share a link with flow F_1, thus $S_3^D = \{F_2\}$.

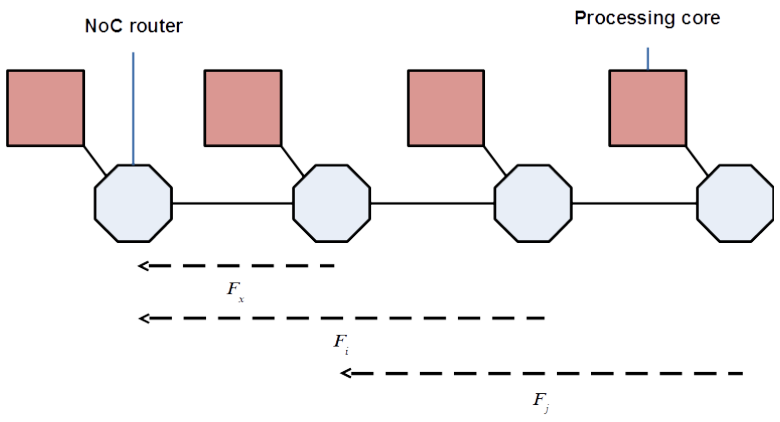


Figure 3.3: Direct and indirect relationship in a NoC based multicore system

From Figure 3.3, although the flows F_j and F_x do not share links together, the indirect interference emerges from an indirect competing relationship between them. In this relationship, the flow F_x imposes an indirect interference on the flow F_j. For this relationship to establish, the intervening flow F_i must exist between the two flows and shares links with them, and the three flows meet the condition $P_x > P_i > P_j$. For flow F_i, an indirect interference set S_j^I is defined as the group of traffic-flows that imposes indirect interference on it. Based on the same example in Figure 3.1, the traffic flows F_1 and F_3 are in an indirect competing relationship, and the flow F_1 imposes an indirect interference on the flow F_3. Therefore, the indirect interference set for the flow F_3 is $S_3^I = \{F_1\}$.

The basic latency C can be calculated with equation 3.2, where H is the number of hops, V is the time needed for a flit to traverse a link, B is the time needed for a router to route a flit and arbitrate packet headers and L is the packet size in number of flits.

$$C = H \times V + L \times B \tag{3.2}$$

With these conditions, a tighter bound on the worst-case latency (R_i) of a flow in equation (3.3) is provided as proposed in [135]. The first term of equation (3.3) is the observed flow's basic latency, and the second term refers to the maximum latency caused by the direct and indirect interference of the high-priority flows in the interference set (S_i^D). Given a set of traffic flows in which each has a period T, packets are transmitted consecutively after each minimum interval. This set is deemed schedulable if for each flow i in this set, $R_i \leq Q_i$, where Q_i is the deadline of flow i.

$$R_i^{n+1} = C_i + \sum_{\forall Flow_j \in S_i^D} \left\lceil \frac{R_i^n + J_j^R + J_j^I}{T_j} \right\rceil C_j \tag{3.3}$$

We extend the real-time analysis in [140] and [135] to support the end-to-end response time analysis of tasks. The analysis could help to determine whether task and flow sets are schedulable in the system with a given task mapping. We define the end-to-end response time of a task as the time taken from its release until the last packet it sends is received by the receiving task. In order to extend these equations, the following assumptions are considered:

- The release of a traffic flow occurs only after the sending task has finished its execution
- The overhead of NI is assumed as very low and hence negligible.

In real-time analysis only the maximum response time of a task is concerned and if the task is able to meet its deadline with the maximum execution time in a worst-case scenario, then the task is deemed schedulable in any scenario. Therefore, the minimum execution time of the task is assumed as null in our model. As shown in Figure 3.4, a task i is deemed schedulable if $r_i \leq d_i$. From the same figure, consider the maximum deviation of a flow i from its release period (denoted as J_i^R), the delay of the flow to reach its destination is $J_i^R + R_i$. Based on the assumptions listed above, a packet is not transmitted as soon as it is generated but it is hold until the sending task finishes its execution. Since the overhead of NI is assumed as negligible, then it is immediate that the release jitter of the flow can be deduced as the worst-case response time ($J_i^R = r_i$) of its sending task. Consider the release jitter of flow i, the flow is deemed schedulable if $J_i^R + R_i \leq d_i$ or after substitution $r_i + R_i \leq d_i$.

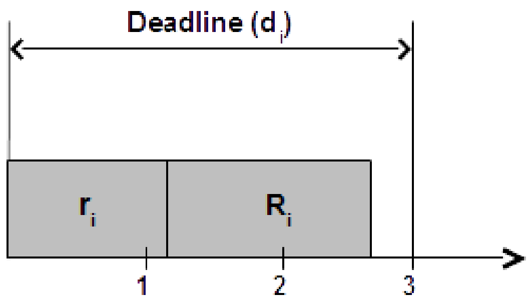


Figure 3.4: Time window of task and message

From equation (3.3), J_j^I is referred to as the interference jitter of flow i and it is caused by the indirect interference as a result of pre-emption by the high-priority flows which shared the same path as flow j but not as the observed flow i. It is defined as the maximum deviation between two successive packet release and can be yielded from calculating the difference between its maximum and minimum value. For example, consider a situation where no higher priority packet is sent in a period, and hence the minimum start transmission time of the high-priority packet becomes zero, then the upper bound of interference jitter for the traffic flow i is $J_j^I \leq R_j - C_j$. Therefore:

$$R_i^{n+1} = C_i + \sum_{\forall Flow_j \in S_i^D} \left\lceil \frac{R_i^n + r_j + J_j^I}{T_j} \right\rceil C_j \tag{3.4}$$

The worst-case response time of a task and the worst-case latency of a flow can be calculated by equations (3.1) and (3.4) respectively. To reduce the complexity of the analysis, we assume the deadline of a task to be equivalent to its period, that is, $d = t$, and that a traffic flow shares the same deadline as its sending task, hence $Q = t$. By validating the end-to-end response time of a task against its deadline, the schedulability of the task can be determined. A task which is allocated to a core is schedulable if it meets its deadline. However, packets sent by the task to another task at a different core may not necessarily be schedulable if they fail to meet the deadline of their sending task (for example, due to the interference from other packets sharing the same path or to the delay in execution of the task itself). Given that Ut_i and Uf_i are the numbers of unschedulable tasks and flows respectively, the following comparisons determine whether a task set and its respective packet flows are schedulable in the system.

$$Task_i : if\ r_i > d_i \Rightarrow Ut_i = 1 \tag{3.5}$$

$$Flow_i : if\ r_i + R_i > d_i \Rightarrow Uf_i = 1 \tag{3.6}$$

The metric representing the schedulability of the system is the total number of unschedulable tasks and flows. If $S = 0$, the system is deemed to be fully schedulable.

$$S = \sum_{i=1}^{k} Ut_i + \sum_{i=1}^{l} Uf_i \tag{3.7}$$

3.4 Optimisation Process

In general, the proposed optimisation approach uses GA as the optimisation algorithm to explore task mappings and selects the best that meet or are close to the optimisation objectives. It relies on the fitness function (see equation 3.7) to evaluate the schedulability of every task mapping. The valuable feedback of the function is used to further optimise the task mappings.

GA was selected as the optimisation algorithm in our approach because of its well-known performance in addressing optimisation problems. Based on evolutionary principles, the algorithm produces a group of individuals and improves them over several evolution cycles (or generations) to create a better population. This improvement is a manifestation of an evolutionary process that manipulates individual chromosomes. The chromosome is a repository in every living being and contains information about an individual, and some of this information is passed through generations. The thread-like structure of the chromosome used by GA provides a practical repository to represent multiple design parameters. With these characteristics, GA becomes an effective optimisation algorithm for exploring multiple parameters simultaneously.

As shown in Figure 3.5, first the optimisation algorithm creates several individuals in a parent population and over generations it evolves them with support from its operators. During this process, other populations such as offspring and combined populations are created as temporary populations, which later are merged into the parent population. The algorithm can run continuously until it reaches its maximum generation, or the algorithm can be allowed to run until the optimisation objective is met. The maximum generation is also referred to as the termination condition of the algorithm.

In each generation, the parent population undergoes several steps performed by a set of GA operators: the selection, crossover and mutation operators. As depicted in Figure 3.6, the selection operator selects two individuals from the population to become parents for producing new offspring. The selection of parents is based on the binary tournament procedure that selects two random individuals and compares them according to their fitness values. An individual with better fitness value is the winner and then becomes a parent. The same procedure is

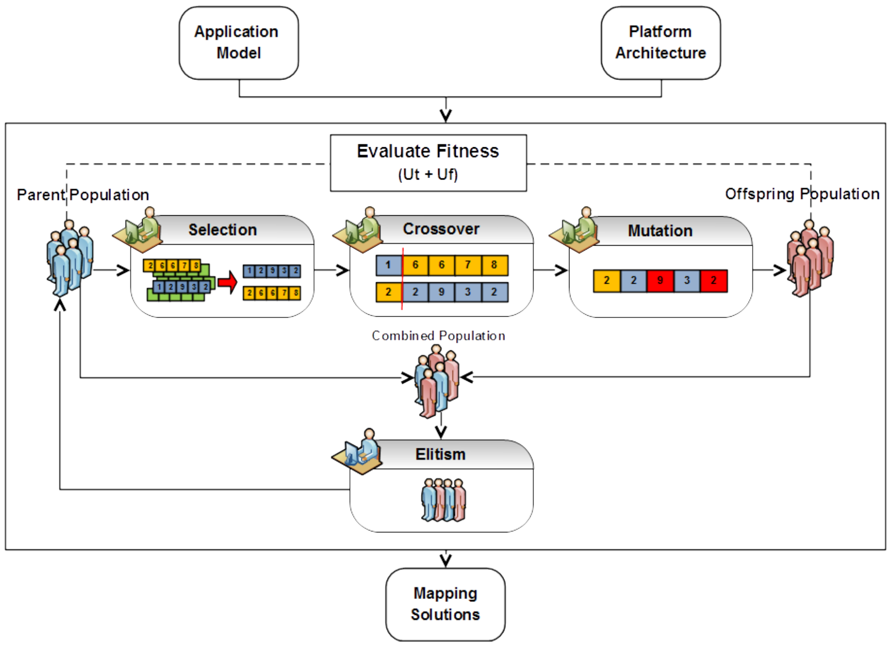


Figure 3.5: Single objective optimisation process

repeated again to choose its mate.

In a mating process (or crossover), genes from both parents are exchanged to produce new offspring. The number of genes that are exchanged is decided by the crossover point (the red line) drawn across both chromosomes of the parents as displayed in Figure 3.7. Only genes on the right side of the crossover point are exchanged, whereas the genes on the left side remain the same. However, only some parents are involved in the mating process; the others will be passed on to the offspring population as themselves. In the former situation, the offspring are passed to the offspring population. This probability is decided by the crossover rate of the mating process.

After crossover, a proportion of genes in an offspring's chromosome closely resembles its parents' genes. If all individuals in the offspring population resemble their parents, the population will have less diversity between individuals. Diversity is a catalyst to widen the exploration in the design space; hence diversification of individuals is an important step to avoid early convergence. Gene mutation is one way to increase the population diversity. The mutation is conducted on selected genes and the number of genes that will be mutated is determined by a

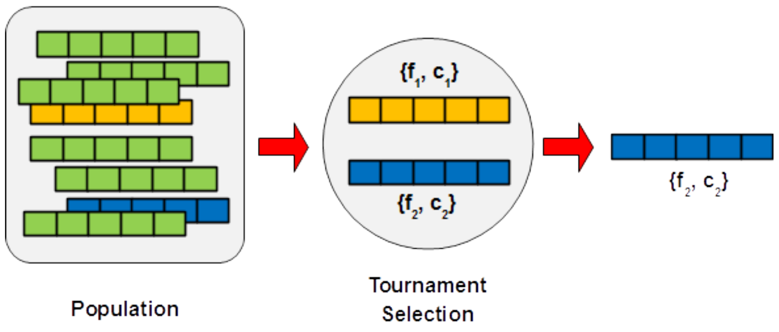


Figure 3.6: Binary tournament selection

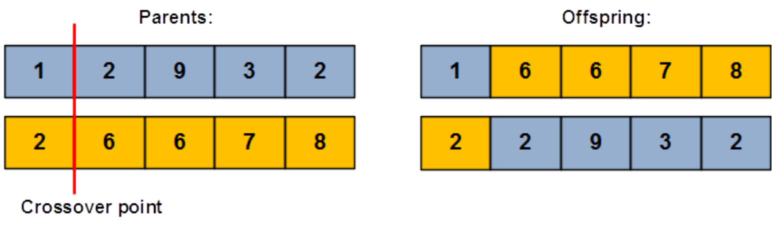


Figure 3.7: Single-point crossover

mutation rate. This rate must be carefully chosen, otherwise it will bring other side effects to the algorithm's performance. Setting a mutation rate to a high value will have more mutated genes but it eliminates the inherited genes from parents. Any good characteristic from parents will vanish and offspring will become completely different individuals from their parents. As a consequence, the search of the task mapping will be diverted to unpromising directions away from the intended direction that leads to the objective. Figure 3.8 shows an example of gene mutation; given a mutation rate of 0.01, each gene's mutation value (a random number between 0 and 1) is compared with the rate. Any gene with a mutation value less than the rate will be mutated by replacing the existing value in the gene with a new value, for example, for task mapping the new value is one of the core indexes.

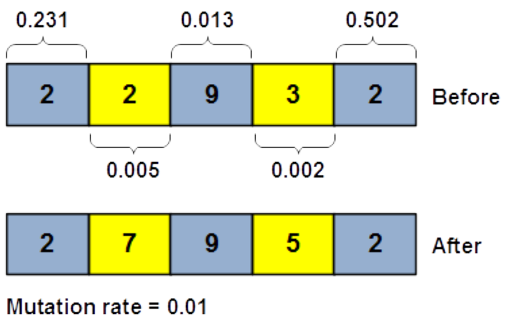


Figure 3.8: Mutation

3.5 Task Mapping and Priority Assignment Configuration

In a priority pre-emptive hard real-time system based on NoC, every task and flow is assigned with a priority level to determine the order of precedence for accessing the computing and communication resources. Priority assignment of a set of tasks and flows could be based on a priority pre-emptive scheduling or a random assignment. In a single processor platform, the rate monotonic scheduling policy is optimal [64], because if the priority scheduling that can make the system schedulable exists, it can always be found by the policy. However, this is not always the case in a multi-processor system since tasks can be mapped anywhere in the platform and the strict assignment policy that bounds the scheduling of tasks does not provide low-priority tasks with any option to avoid interference from high-priority tasks. The impact of this inflexibility causes delays to the end-to-end response time of low-priority tasks leading to them becoming unschedulable. The simultaneous optimisation of task mapping and priority assignment is proposed to insert flexibility into the scheduling of tasks in a way that the low-priority tasks can become schedulable.

The optimisation process combines task mapping and priority assignment as a configuration for the system. The information within each configuration specifies the location of tasks in the platform and the priority level of each task. A chromosome is a thread-like structure used by GA and represents an individual within a population. In order to use GA as the optimisation algorithm, the information within the configuration must be encoded in the chromosome. A successful encoding of information depends on how the chromosome is structured to make every configuration as an individual in the population. This will allow the GA operators to simultaneously configure both task mapping and priority assignment by evolving the individuals.

A chromosome is built upon a set of small units called genes and each gene represents a variable of specific type (such as integer or binary). As Figure 3.9 shows, a chromosome can be segregated to form several groups of genes with the same type, allowing different design parameters to be structured into the same chromosome. In Figure 3.9, the chromosome is divided into two parts. The first part (green) represents information on task mapping and the other half (orange) represents the priority assignment of all tasks. Decoding the chromosome will reveal a configuration on how to map all tasks onto processing cores and how to assign priorities to the mapped tasks.

The first half of the chromosome (green) contains a group of genes for encoding a task mapping. This group represents a set of tasks to be mapped on the system. A task accommodates a gene and it is defined as an integer variable to store a processing core index, onto which the task will be mapped in the system. In a multi-processor system, several tasks can share a processing core; hence the same

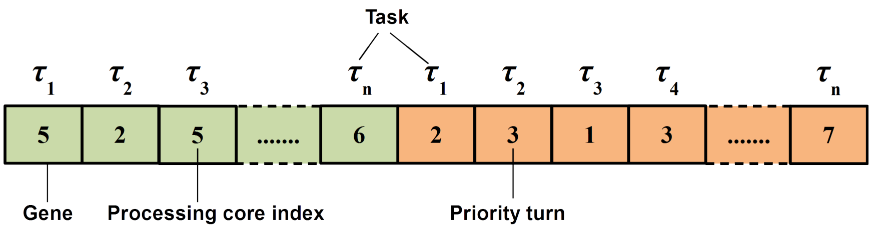

Task
τ_1 τ_2 τ_3 τ_n τ_1 τ_2 τ_3 τ_4 τ_n
5 2 5 .... 6 2 3 1 3 7
Gene
Processing core index
Priority turn

Figure 3.9: Task mapping and priority assignment chromosome structure

index (of a processing core) can appear multiple times at different genes. For example, task τ_1 (1st gene) and task τ_3 (3rd gene) are mapped onto the same processing core with index 5. Figure 3.10 shows the task mapping on a 3x3 platform after decoding the chromosome.

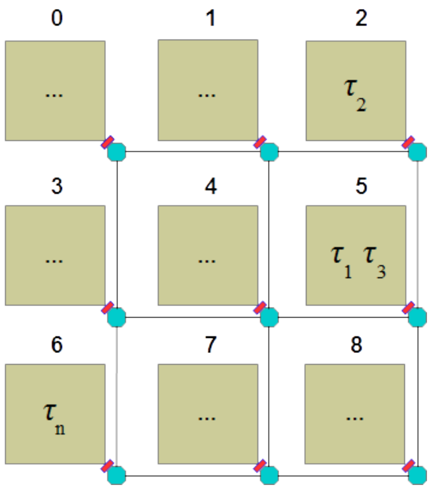
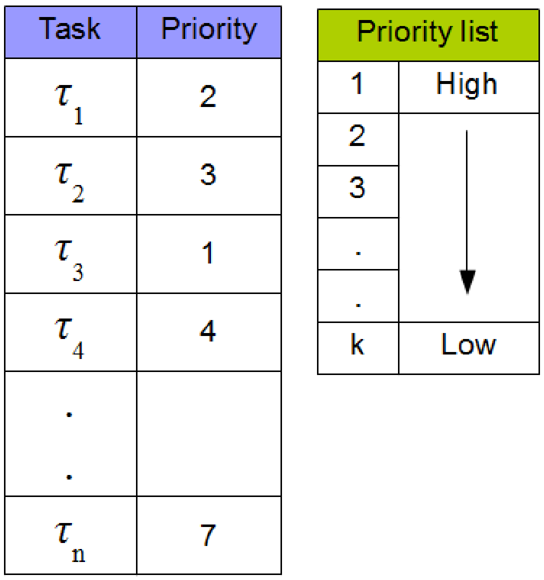


Figure 3.10: Corresponding task mapping after decoding chromosome

Priority assignment refers to the information encoded in the rest of the chromosome: the group of genes shown in orange. Similar to the former group, this group represents the same set of tasks and the genes are defined as integer variables. It is commonly assumed that the priority of every task is unique in a fixed priority pre-emptive scheduling policy, in other words, the same priority cannot be assigned to more than one task. This is to ensure predictable execution of tasks in shared resources and further explanation can be referred from section 3.2. Therefore, every gene in the group is configured in a way that gives every task a unique priority. In order to configure in this way, every task is given a priority in turn. A separate priority set is created and the value of every gene actually refers to the turn of the task in getting a priority from the priority set. A benefit from this priority encoding is to reduce the complexity of having to avoid the same priority being assigned twice by the algorithm's operators if priority levels are defined directly in the chromosome.



Task	Priority
τ_1	2
τ_2	3
τ_3	1
τ_4	4
. .	
τ_n	7

Priority list	
1	High
2	
3	
.	
.	
k	Low

Figure 3.11: Corresponding priority assignment after decoding chromosome

The priority set is a sequence of integer numbers and sorted in a descending order (from high to low priority). Since every task will receive a different priority level, the size of the priority set is equivalent to the number of tasks. It is worth noting that this is a design choice; a system designer can choose a different order sequence, for example, by sorting in ascending order. A task receives its priority from the priority set in turn. For example, task τ_3 has value one, which means it gets the first turn, hence task τ_3 receives the highest priority over the rest of the tasks. Subsequently, task τ_1 with value two gets second turn, and so on, followed by the rest of the tasks. In a case where different tasks get the same value, the order of precedence is decided from the left side of the chromosome. For example, task τ_2 and τ_4 share the same value (three), hence task τ_2 receives priority first, followed by task τ_4, that is, the priority of task τ_2 is higher than that of task τ_4. The priority level of each task after decoding the chromosome is shown in Figure 3.11.

3.6 Schedulability Objective

A hard real-time embedded system is deemed schedulable if the response time of all the tasks and messages in the system finishes before their deadlines. Meeting this requirement is crucial for the system, and hence configuring the values of the design parameters must align with the objective. The objective is minimising the number of unschedulable tasks and flows in the system. If the schedulability metric is equal to zero, then it can be assumed that the configuration has achieved the specified objective. The objective function is derived as equation (3.8), where S is the number of unschedulable tasks and flows and can be calculated with equation (3.7).

$$Obj_1 = \min(S) \tag{3.8}$$

3.7 RTA Integration with GA Framework

Generally, GAs are meta-heuristics and can be adapted to become optimisation algorithms for addressing different kinds of optimisation problem. The basic working principles of GAs are based on the notion of genetic evolution. During evolution, several steps occur which change the chromosomes of organisms into becoming better or worse. These steps including the mating of parents, the mutation of individual chromosomes and the selection of the best individuals from a population; all of these are performed by several GA operators. The implementation of these operators may vary, depending on the chromosome structure. Nonetheless, optimisation objectives are the deciding factor for the selection of fitness functions and the design of chromosomes, making each GA distinct from the others.

A software developer can develop a GA in several ways. Starting from scratch is possible, but requires more time to implement and test the algorithm. Existing GA templates have already been implemented in several GA frameworks. The adoption of a framework could facilitate faster development of our proposed optimisation techniques. Among the well-known GA frameworks are jMetal [141], ECJ [142] and JGAP [143]. Each of these has specific features, but jMetal is aimed at multi-objective optimisation. However, it can easily be adapted for single objective optimisation, which is used to address the optimisation problems in the following chapters.

The jMetal framework provides a set of classes based on object-oriented principles which can be easily adapted for creating either an extension or a wholly new optimisation algorithm. These classes represent the basic building blocks of a GA such as the parent classes for genetic operators and the optimisation problems. By taking advantage of the classes using code-reusing principles, the implementation of new classes of the proposed optimisation algorithms is seamless. Once these classes are fully implemented within the framework, the optimisation process can take advantage of other features such as the function which compares performance between multiple algorithms or the function which filters Pareto-optimal solutions. The integration of the framework with the design space exploration environment is not a straightforward process. A customisation has been made in the framework to integrate with the system model.

Our proposed optimisation algorithm is known as SCGA and this algorithm was created by implementing the chromosome structure (see Figure 3.9) in GA and integrating the schedulability fitness function which consists of the end-to-end response time analysis (section 3.3). The algorithm was configured according to the optimisation objective explained in section 3.6.

3.8 Evaluation

3.8.1 Test Benches

A test bench defines a set of tasks and messages of an application. For studying the performance of the proposed approach a test bench based on the realistic application called Autonomous Vehicle Application (AVA) (see Appendix A) was used in the evaluation. AVA contains 33 tasks for processing data from various sensors and 38 traffic flows to traverse input data from sensors and output data to different actuators in the autonomous vehicle system. Due to the certain restrictions imposed on our evaluation model, multiple dependencies between tasks were not supported and thus AVA has been simplified to ensure compatibility with the evaluation model.

In addition, a group of synthetic test benches with a larger number of tasks and messages than the AVA were generated for further evaluating the proposed approach. By increasing the number of tasks and messages, the amount of interference between them is likely to escalate. This will allow us to investigate how much interference can be reduced by the proposed approach compared with the baselines. In this evaluation, we increased the number of tasks and messages based on the platforms size (that is, the size of task and flow set for the 10x10 platform is larger than for the 5x5 platform).

A random number of tasks between a given range, was chosen and the same way was applied for generating flows. The previous study [11] used a synthetic application containing 50 tasks and 50 messages to map hard real-time tasks onto a 4x4 and 5x5 NoC platforms. In this experiment, we added a larger platform (10x10) besides the 4x4 and 5x5 platforms and we increased the number of test benches by varying the utilisation levels to provide better insights on the performance of the proposed approach. To provide a wider set of evaluations than [11], multiple range of minimum and maximum number tasks and messages were considered according to the size of platforms as shown in Table 3.2.

Table 3.2: The number of tasks and messages for generating synthetic test benches

Range	**Minimum**	**Maximum**	**platform**
1	40	50	4x4
2	50	60	5x5
3	100	110	10x10

Then, random pairs between source and destination tasks were created. Each task was characterised with a worst case execution time c, a period t and a priority level p, while each flow is characterised with a basic latency C, a packet size L and a priority level P. Priorities are assigned to tasks according to the Rate Monotonic priority assignment policy [62] (referred to as RM) or random assignment (referred to as RAN). A flow is given the same priority level as the source task. Given a range between a maximum and a minimum values, a random period t is generated according to the uniform distribution. Based on function (6.1), c can be calculated with a given utilisation of the task. For flows, the packet size (in the number of flits) is chosen between the range [3, 28000] according to the uniform probability distribution. The period of a flow is equivalent to the period of the source task. With the assumptions of a flit travel through a link in one cycle and a router takes one cycle to arbitrate packet headers and route a flit, the basic latency C can be calculated from equation 3.2.

The design space of task mapping grows exponentially with the number of tasks and cores, as explained in 2.4.5. An algorithm that is capable of addressing this NP-hard problem in polynomial time as yet is not exist. Exploring many task mappings using GA requires a significant number of individuals for ensuring the diversity of the population is maintained to produce good task mappings. However, this increases the number of evaluation because every task mapping must be evaluated for selection process, consequently delaying the runtime of the task mapping optimisation process. Furthermore, not only the number of evaluation is increased, every task and message in the test bench must also be analysed for each task mapping. To cover as many as task mappings for every test bench (with different size of task and message sets and hence utilisation) in many runs is challenging given the limited time of this study.

A balance between the number of GA runs and the number of results that we intended to achieve to prove the hypothesis must be found for this experiment. Our target was to focus on the range of utilisation at which the improvement of the proposed approach can be shown, starting from where it can find a schedulable task mapping until it becomes unschedulable. From the results published in [11], a fully schedulable task mapping on the 5x5 platform can be found for the synthetic test bench with 60% core utilisation. The reported results are based on the number of unschedulable messages as the metric for determining the schedulability of the task mapping, but it provides us a clue to select the starting point of which utilisation level to choose. For this purpose, we selected a level of utilisation from 45% percent, and then gradually increased it to investigate the improvement. The chosen percentage is lower than 60% because we took into account not only the schedulability of messages but also tasks, which is not taken into account in [11].

The test benches were numbered consecutively proportional to their utilisation levels, for example, TB1 has a lower utilisation level than TB5. These test benches can be divided into two groups as listed in Table 3.3. Each test bench in the first group consists of a set of tasks with a range of utilisation between 46% and 73% of 5x5 core utilisation, and a set of messages with less than 1% of 5x5 link utilisation[4]. In the second group each test bench contains a set of tasks with a range of utilisation between 46% and 55% of 10x10 core utilisation, and a set of messages with less than 1% utilisation of 10x10 link utilisation[4]. Synthetic test benches generation based on these settings provided us with a harder set of application than the realistic application AVA to measure how much improvement that the proposed approach can introduce.

Table 3.3: Synthetic test benches for 4x4, 5x5 and 10x10 platforms

Group	**Test benches**	**Task utilisation**[5]	**No. of tasks**	**No. of messages**	**Mapping platform**
1	TB1 - TB20	$0.46 \leq util \leq 0.73$	54	54	4x4, 5x5
2	TB21 - TB30	$0.46 \leq util \leq 0.55$	104	104	10x10

In this experimental work, we investigated the performance of the proposed approach with varied task utilisation, but message utilisation was constant. This allowed us to closely observe the impact of changing the task mapping and priority on sets of task with different task utilisation levels. Furthermore, this way also allowed us to keep the number of tests relevant as the GA-based optimisation algorithms (SCGA and baselines) consumes an amount of time to perform optimisation until the maximum generation. In fact, the increase in the number of tasks and messages also adds to the duration of optimisation due to the number of interference which needs to be calculated for analysing the schedulability of

[4]Total links on a 5x5 platform is 130 and 10x10 platform is 560.

[5]Assuming each core can support 100% utilisation, total utilisation of 5x5 and 10x10 are 2500% and 10000% respectively.

the system.

3.8.2 Baselines

Several baselines were selected and compared with SCGA. One of the baselines was GA, which only configures task mapping and depends on either RM or RAN to assign a priority to each task and message in the test benches, hereafter are known as GA(RM) and GA(RAN). SCGA and GA were based on the same optimisation process (see Figure 3.5) to meet a common objective, which is finding a fully schedulable task mapping for NoC-based hard real-time systems.

SCGA and GA were configured with the same settings [144] to perform evaluation under the same condition. The researchers [144] have performed a parametric analysis for multi-objective GA based on the selection of settings published earlier in [21]. Based on the analysis, we chose the best GA settings as displayed in Table 3.4 for the purpose of evaluation.

Table 3.4: A set of GA settings used in evaluation

Population	**Crossover rate**	**Mutation rate**	**Generation**
100	0.5	0.01	500

One of the differences between the algorithms is the way a solution is encoded in their chromosomes. As previously mentioned, the GA relied on RM or RAN for priority assignment, hence its chromosome only contains information on task mapping similar to the first half of SCGA's chromosome (see Figure 3.9). On other hand, SCGA's chromosome was configured as explained in section 3.5.

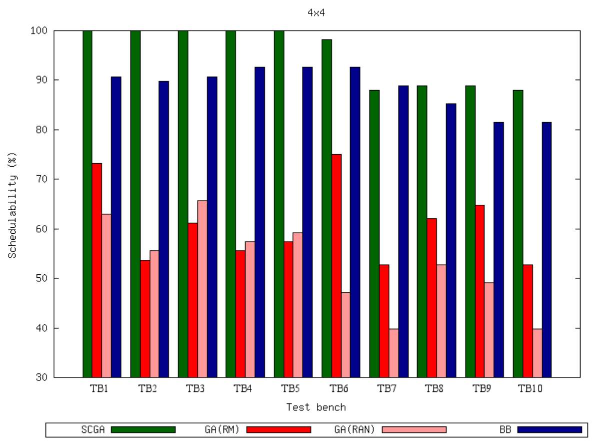
Previous work [12] addressing the priority assignment problem in the NoC-based system was based on a Branch-and-Bound (BB) heuristic to find a suitable priority assignment for a set of traffic flows. In order to compare SCGA with that approach, the same technique was developed but we modified it to include the function for assigning priorities to a set of tasks. This is because BB only configures priority assignments based on a given task mapping, and hence we used a random task mapping as its input.

In addition to the above baselines, a Nearest Neighbour (NN) and a set of bin-packing mapping heuristics such as First Fit (FF) and Best Fit (BF) were also used as baselines for SCGA. NN allocates a set of tasks onto a platform by first allocating a task at a random processing core and then repeatedly allocates the remaining tasks to the nearest processing core until all tasks have been allocated. Based on the utilisation of each task, FF allocates a task to the first processing core that fits the task, that is, the full utilisation of the core is not exceeded. BF also refers to the utilisation of each task similar to FF, but allocates a task to the processing core that has the least utilisation.

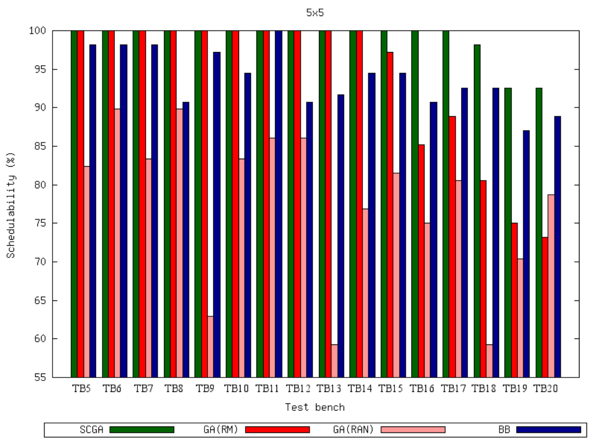
3.8.3 Results

An experiment was performed for the purpose of studying the effects of the proposed approach to improving the schedulability of the system. The hypothesis of this experimental work states that SCGA is better than the baselines in finding the schedulable task mapping for the system. Before discussing the results further, it is worth mentioning that Figures 3.12, 3.13 and 3.13 shows the results based on a single GA run per test bench. This experiment was conducted after we took into consideration the time overhead in running the GA multiple times for the same test bench. In every run, the GA must evaluate each task and traffic flow for every task mapping produced until it terminates. Consider an example with 33 tasks and 38 flows, and the GA is configured with 100 populations for running until 500 generations in every run. The total number of evaluations is approximated around 3,550,000 evaluations (= 100 populations 500 generations (33 tasks + (38) flows)). Furthermore, the number of evaluation increases for large size of task and flow sets and since statistical evidence require a significant set of samples, running the GA multiple times for the same test bench under limited time and resources is impractical.

The best schedulable task mappings found by each of the baselines (GA(RAN), GA(RM) and BB) and SCGA in the populations at the end of optimisation, are shown respectively in Figures 3.12a and 3.12b for both the 4x4 and 5x5 platforms. From the results shown in Figure 3.12a, SCGA has outperformed GA and BB with its fully schedulable task mappings from TB1 until TB5. From TB6 onwards, none of the algorithms has succeeded in finding a fully schedulable task mapping, although the schedulability percentage of SCGA is higher than



(a) 4x4 platform



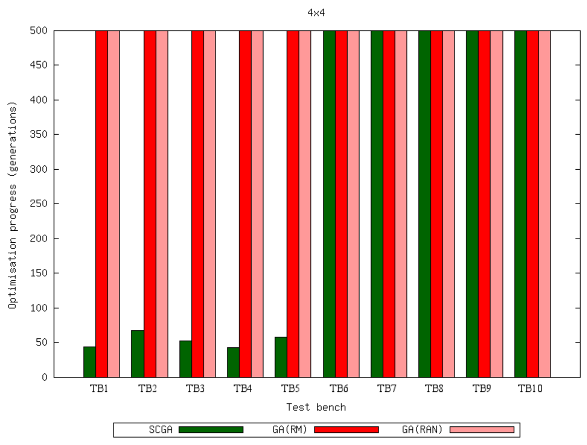
(b) 5x5 platform

Figure 3.12: Schedulability of the best task mapping found by SCGA, GA and BB for every test bench mapped onto 4x4 and 5x5 platforms

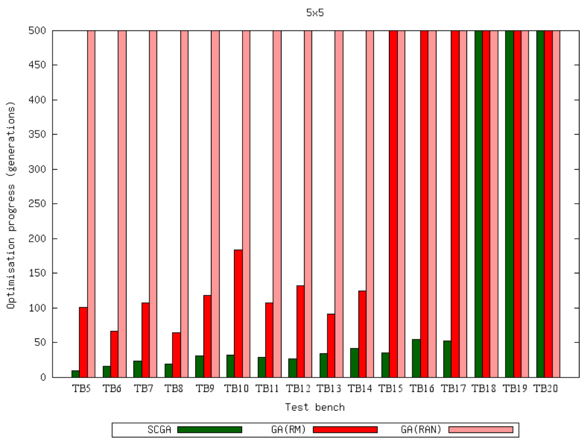
the baselines. GA(RM) and GA(RAN) have failed to find any schedulable task mapping for all the test benches. Similar to them, none of the task mappings that were given to BB can become fully schedulable.

Once again, SCGA performed better than GA(RM), GA(RAN) and BB on the 5x5 platform. As shown in Figure 3.12b, all SCGA's task mappings from TB5 until TB17 are schedulable. Although a performance drop is seen from TB18, it still maintained a higher schedulability than any of the baselines. GA(RM) showed a similar performance to SCGA, but its performance fell below 100% from TB15 onwards. Both GA(RAN) and BB were unable to find any schedulable task mapping for all the test benches on the 5x5 platform, except for TB11 for which BB found a fully schedulable task mapping. Nevertheless, SCGA outperformed all the baselines on the 4x4 and 5x5 platforms by enabling the task mapping to become fully schedulable by changing the priority assignment of tasks. For some test benches on which a fully schedulable task mapping was hard to find, SCGA still maintained its performance by producing task mappings with better schedulability than the baselines.

SCGA, GA(RM) and GA(RAN) each took a particular amount of progress to find the first schedulable task mapping during optimisation. Figure 3.13 shows a comparison between the algorithms based on the number of generations to



(a) 4x4 platform



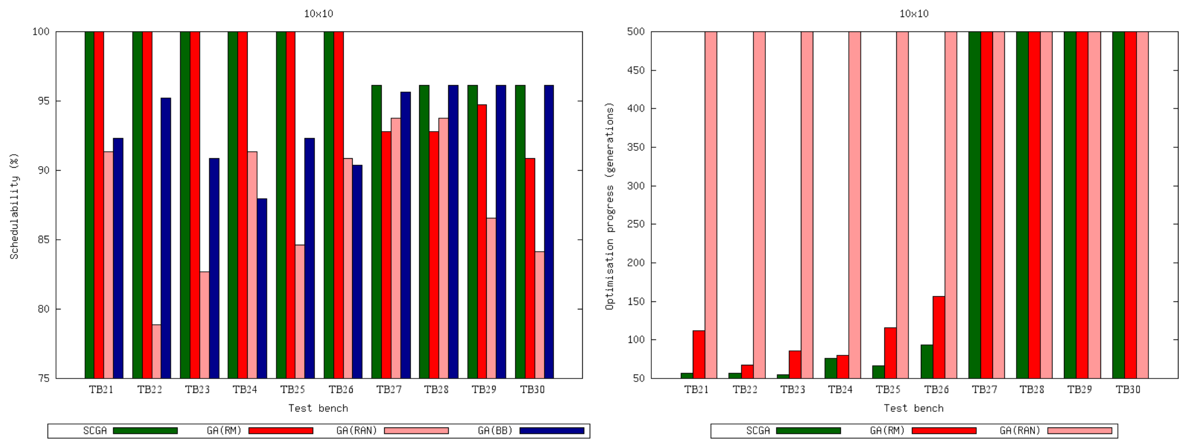
(b) 5x5 platform

Figure 3.13: Optimisation progress taken by SCGA and GA to find the first schedulable task mapping for each test bench on 4x4 and 5x5 platforms

produce the first schedulable task mapping for every test bench. It is worth noting that at the maximum generation (500), a schedulable task mapping may or may not have been found. Figure 3.12a shows that GA(RM) and GA(RAN) were unable to produce a schedulable task mapping for all the test benches on the 4x4 platform. Both algorithms continued up to a particular level of schedulability but failed to achieve a fully schedulable task mapping before the optimisation ended at 500 generations[6]. This is the reason both algorithms are plotted with the maximum generations for all the test benches in Figure 3.13a. Meanwhile, SCGA converged better than GA by taking fewer than 100 generations to find a schedulable task mapping for all the test benches except from TB6 to TB10, when it could not find any schedulable task mapping. SCGA performed similarly on a 5x5 platform, as shown in Figure 3.13b; it took fewer generations to produce a schedulable task mapping than GA(RM) and GA(RAN) from TB5 until TB17. From the results, SCGA not only found fully schedulable task mappings for the test benches but also it found them in a lower number of generations than the baselines.

It is worth noting that with the same test bench, finding a schedulable task

[6]Approximately 300 seconds. This duration can become longer for mapping a large task set onto a small platform due to the increase in the volume of interference.



(a) Schedulability (b) Optimisation progress

Figure 3.14: Based on a 10x10 platform; (a) schedulability of the best task mapping, (b) optimisation progress to discover the first schedulable task mapping.

mapping on a smaller platform was harder than on the larger platform. Increasing the size of platform provides more resources to which tasks can be allocated and reduces the contention between them. For example, SCGA required fewer generations to find a schedulable task mapping for TB5 on the 5x5 platform as compared with the 4x4 platform.

So far, SCGA has shown good performance for mapping a set of test benches onto smaller platforms such as 4x4 and 5x5. A set of different test benches (TB21-TB30), with larger utilisation than the synthetic test benches in group 1, were used for a larger platform (10x10). As depicted in Figure 3.14a, SCGA and GA(RM) each found a fully schedulable task mapping from TB21 until TB26. Nonetheless, in Figure 3.14b, SCGA showed better convergence than GA(RM) and GA(RAN). This performance is consistent with the previous mapping on the smaller platforms. Although none of the algorithms was able to find any schedulable task mapping from TB27 onwards, once again SCGA's task mapping was better in schedulability compared with GA(RM) and GA(RAN). Furthermore, GA(RAN) was unable to find any schedulable task mapping and BB failed to find any priority assignment that could make the task mapping schedulable for all the test benches on the 10x10 platform.

We have so far discussed, the feasibility of SCGA in improving the task map-

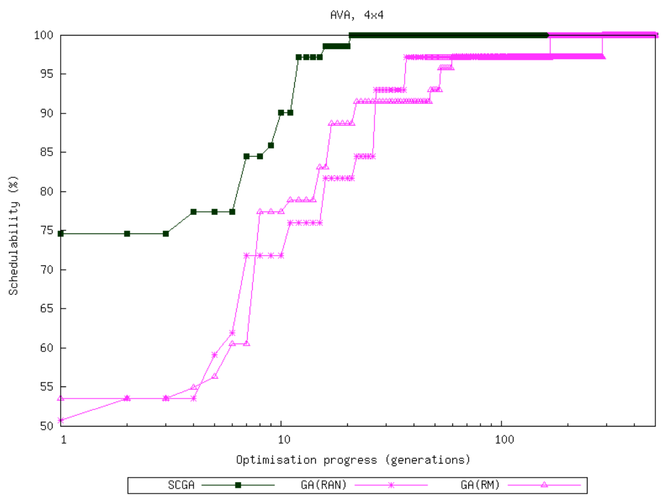


Figure 3.15: Schedulability convergence of SCGA and baselines when mapping AVA onto a 4x4 platform

ping and priority assignment based on a set of synthetic applications. To further demonstrate the feasibility of SCGA, we studied the convergence of it when mapping a realistic test bench (AVA) onto the 4x4 platform. As depicted in Figure 3.15, SCGA, GA(RM) and GA(RAN) successfully converged to a schedulable task mapping. Although all of them found the task mapping, it can also be seen from Figure 3.15 that SCGA's convergence rate was faster than that of its counterparts. SCGA took less than 30 generations to converge, compared with GA(RM) and GA(RAN) which took more than 100 generations. One generation after the beginning of optimisation (at first generation), SCGA produced a task mapping with almost 75% schedulability. Afterwards, SCGA gradually improved the task mapping at every generation and finally it converged to a schedulable task mapping, whereas its counterparts still struggled to achieve a task mapping with schedulability between 85% and 90%. In another study, several mapping heuristics such as NN, FF and BF were used to map the same application and platform. The comparison of the heuristics and SCGA is depicted in Figure 3.16, which shows that these heuristics failed to find any schedulable task mapping, in contrast with SCGA which successfully found a schedulable task mapping. A reason why the NN and the bin-packing heuristics failed to achieve a fully schedulable task mapping are due to their fixed way of mapping the test bench. As a consequence, these heuristics produced no alternative task mappings and the lack of changes causes less opportunity for low priority tasks and messages to evade interference from their high priority counterparts.

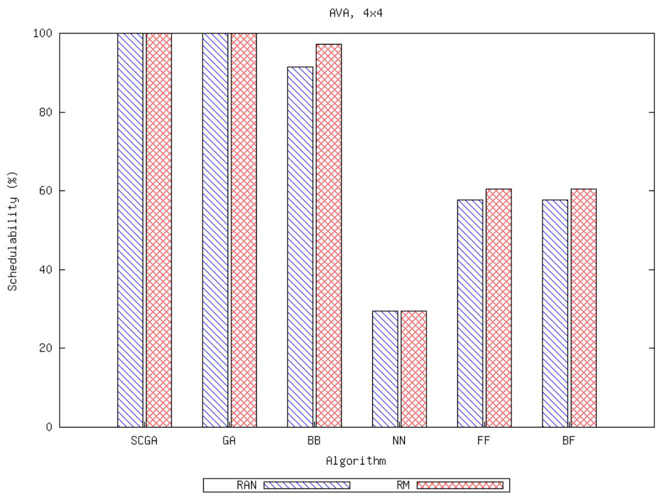


Figure 3.16: Schedulability of task mappings found by SCGA and bin-packing heuristics when mapping AVA onto a 4x4 platform

Based on the consistent performance shown by SCGA in all the case studies, it can be inferred that an optimisation algorithm integrated with our proposed approach could perform better than the baselines. Better schedulability and faster convergence are some of improvements made by allowing the algorithm to simultaneously configure task mapping and priority assignment. In spite of the additional dimension over the existing task mapping configuration, SCGA was still able to find the schedulable task mapping and converged faster than the baselines. The convergence was measured in number of generations the GA took to find the first schedulable task mapping and SCGA was able to find the task mapping in fewer generations than the baselines. Although some of the baselines were able to find a schedulable task mapping as SCGA, the faster convergence of SCGA is a significant improvement over the baselines. The results presented in this section suggest that our experimental hypothesis is valid for the chosen benchmarks. Based on these results, SCGA can be introduced as an efficient exploration tool to find the feasible task mapping that could make the NoC-based hard real-time system schedulable.

3.9 Summary

A task mapping is not feasible for a hard real-time embedded system if it leads to the system becoming unschedulable because of the interference suffered by low-priority tasks. Reducing the interference is possible by changing the priority assignment of the mapped tasks to lessen the effects of pre-emption when sharing the same resources. Based on this notion, we proposed an approach which simultaneously configured task mapping and priority assignment to find a possible configuration that can make the system fully schedulable. Finding the configuration considers the overall schedulability of every task and message by taking into account the end-to-end response time of all mapped tasks. The results from experiments based on different types of test benches mapped onto small and large platforms, we have shown that the proposed approach was able to find better configurations than the baselines. In addition, simultaneous configuration of task mapping and priority also improved the convergence of the optimisation algorithm in fewer generations than the baselines.

Chapter 4

Multi-objective Task Mapping Optimisation

Task mapping determines where tasks shall be allocated on a platform. It influences the performance of a system and its power dissipation at the same time. The conflicting nature of the objectives requires an optimisation technique to consider the trade-off between the two attributes. However, without appropriate fitness functions to facilitate the optimisation technique, finding a task mapping with a good trade-off between the real-time performance and power dissipation of a NoC-based hard real-time system is challenging. In order to address this problem, this chapter introduces a multi-objective optimisation technique for finding task mappings based on multiple objectives.

This chapter is organised as follows. Section 4.2 introduces the proposed multi-objective optimisation algorithm. The power macromodel for calculating NoC power dissipation is explained in section 4.4. Task mapping representation in the GA chromosome is described in section 4.5 and the objectives that guide the optimisation process are explained in section 4.6. A discussion of the results obtained from the evaluation of the proposed approach is included in section 4.7. Finally, a summary concludes the contribution of this chapter in section 4.8.

4.1 Conflicting Optimisation Objectives

A system dissipates an amount of power to produce specific performances. The two attributes are normally conflicting with each other: faster performance requires more power than slower performance. It is possible to reduce NoC power dissipation by mapping communicating tasks closer to each other and hence reducing the number of routers and links to deliver messages over the NoC. With this approach tasks are concentrated in the same area, but their end-to-end response times can be affected from contention in the area. For a hard real-time system, the end-to-end response time of each task must not exceed its deadline including in the worst scenario, otherwise the system will become unschedulable. State-of-the-art techniques [11, 21] are based on single-objective optimisation to find a schedulable task mapping. However, such techniques focus solely on the hard real-time performance and lack any insight into NoC power dissipation. Since power dissipation was not considered during optimisation, how much NoC power dissipation involves in the message deliveries was unknown; the NoC might not be efficient in power dissipation.

On the other hand, some energy-aware task mapping approaches [18, 19, 20] have been proposed for addressing the multi-objective optimisation problems of NoC. Mostly targeted for average cases, the approaches estimate average energy dissipation based on the number of hops packets take to arrive at their destinations. Although sufficient for best-effort systems, those approaches lack the calculation for the end-to-end response time upper bound of each task, which is required for evaluating the schedulability of the tasks in a worst case scenario. In the scenario, maximum interference will be likely to happen from contention between tasks or between traffic flows, delaying their end-to-end response times. Since the analysis of schedulability is excluded from the optimisation process, searching for a schedulable and low power task mapping is challenging using these approaches.

Therefore, finding a task mapping for the NoC-based hard real-time embedded systems must take into account the real-time performance alongside other objectives such as minimising power dissipation. Focusing solely on one objective will only bring good in one aspect, but at the disadvantages of the other objectives. Without a proper multi-objective optimisation technique, a conflict between hard real-time performance and power is difficult to resolve and the task mapping will not be optimised accordingly to meet the requirements.

4.2 Multi-Objective Optimisation Algorithm

A multi-objective optimisation algorithm (MOGA) is proposed to find task mappings with good trade-offs between the conflicting objectives: the minimisation of total unschedulable tasks and flows, and the minimisation of NoC power dissipation. In order to achieve this goal, the algorithm evaluates the schedulability of the system and calculates NoC power dissipation as the fitness values of every task mapping, and then selects the best task mappings based on the fitness values.

It is worth noting that the proposed algorithm originated from a GA, which is a meta-heuristic that can be adapted to address task mapping optimisation problems. A new instance of the meta-heuristic can be implemented by integrating relevant fitness functions to evaluate task mappings, by formatting its chromosome structure to represent a task mapping or by introducing new operators to manipulate the chromosomes. An example of such meta-heuristics is NSGA-II [56]. In addition to its configurable properties, it was chosen as the proposed approach because it provides a non-dominated sorting, which is essential for selecting task mappings based on the trade-off between objectives. With its configurable properties and the non-dominated sorting, it is amenable to problems with multiple objectives, as well as its well-known reputation for solving multi-objective optimisation problems.

As shown in Figure 4.1, MOGA starts with a parent population initially populated by randomly created individuals. Every individual in the population represents a task mapping. The evolution of the population over generations produces better individuals through three steps: crossover, mutation and selection. More than one fitness value can be assigned to each individual to facilitate individual ranking and selection of the best individuals for the offspring population. Once the offspring population is completed, it replaces its parent population. Repetition of this process over time gradually improves the quality of the individuals to become better than their predecessors. The evolution steps are similar to the single objective optimisation algorithm; interested readers are referred to the previous chapter, section 3.4 for detailed explanation of the process.

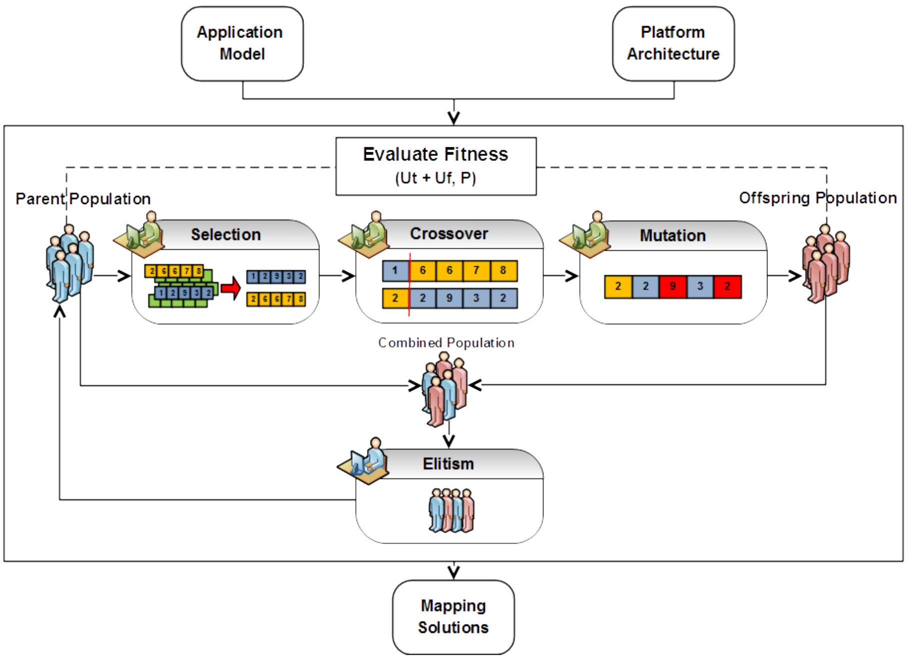


Figure 4.1: Multi-objective optimisation process

The main difference between the single and multi-objective optimisation algorithms is at the final stage of the optimisation, where the latter algorithm's non-dominated sorting plays an important role. A fully ordered list for selecting the best task mapping is less effective to be applied within MOGA due to the existence of conflicting objectives. Instead, the selection of individuals is made based on the Pareto-optimal concept, which is implemented as the non-dominated sorting in the algorithm. With this concept, task mappings that exist in the non-dominated set are regarded as having the best trade-offs in the population. Figure 4.1 shows how individuals from parent and offspring populations are combined to select individuals with the best trade-offs between objectives.

Normally, the number of members in the set is less than the total number of individuals in the population. Therefore, one non-dominated set is not sufficient and more individuals are needed to fill a population. Different levels of non-

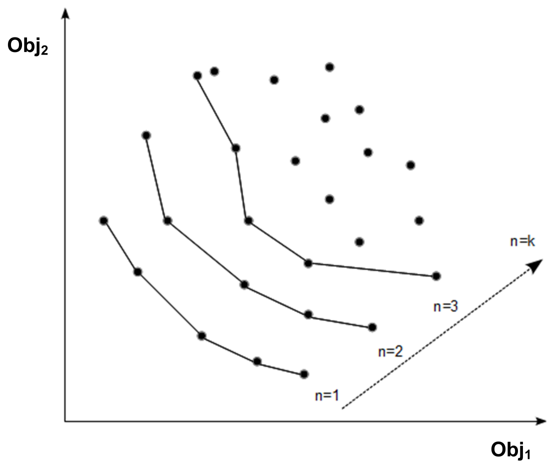


Figure 4.2: Non-domination levels of a population

dominated sets as shown in Figure 4.2 are created to fill an offspring population with the best individuals in the current generation. The first non-dominated set with level 1 is the best non-dominated set, in which its individuals are not dominated by others in the population. Several non-dominated sets after level 1 are created until the offspring population is filled with individuals from these sets. Once the offspring population is filled with the best individuals, it becomes a new parent population to be evolved in the next generation.

4.3 Schedulability Analysis

NoCs are generally scalable and flexible, but analysing their performance guarantees and power dissipation in optimisation is a complex task. For addressing the task mapping optimisation problem, both metrics are needed to determine the trade-off between the objectives. In Chapter 3 an end-to-end real-time analysis was introduced as the fitness function for evaluating the performance guarantees of the system. The schedulability metric that MOGA uses is similar to that applied by the single-objective optimisation algorithm, so equation 3.7 can be reused to calculate the number of schedulable tasks and flows, as one of the fitness values of every task mapping found by MOGA. Interested readers are referred to section 3.3 for details of the fitness function.

4.4 Power Estimation Macromodel

From the results discussed in chapter 3, the task mapping optimisation process has successfully found a feasible task mapping for the NoC-based hard real-time system. Although all tasks and messages are schedulable with the task mapping, it is unknown how much power is dissipated by NoC. This is due to the selection of task mapping made by the GA is solely based on the schedulability metric. Since the GA lacks the function to estimate how much power will be dissipated by NoC, it is difficult to imply the fitness of every task mapping in terms of power dissipation. As a consequence, the schedulable task mapping can be found but it may not be good in power dissipation. This condition could lead to selection of inefficient task mapping. In the optimisation process, the minimisation of the number of unschedulable tasks and messages can be shown as a convergence to a fully schedulable task mapping. If the objective function for minimising the power dissipation is not supported, the graph may show inconsistent power dissipation of the task mapping, as shown in Figure 4.3.

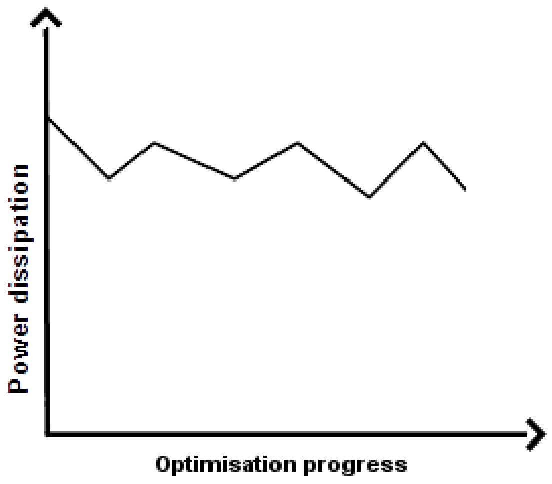


Figure 4.3: Inconsistent power dissipation

A NoC dissipates power when messages traverse through it. According to the general power model proposed by [145], each NoC network component (router, link and NI) dissipates power when transmitting a packet along a route as shown in Figure 4.4. The rationale behind the chosen power model as part of our approach is because their NoC model has similarities with our NoC model. For example, both NoC take advantage from the pipeline feature of wormhole switching to traverse all flits between routers.

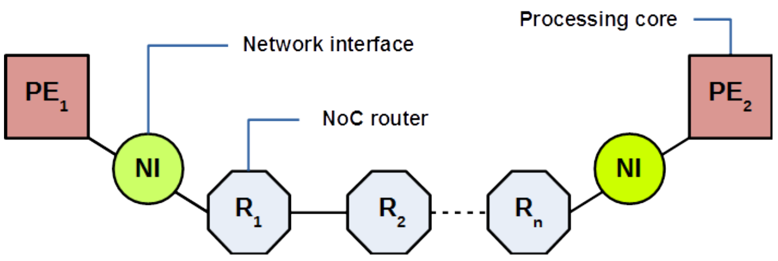


Figure 4.4: General model of the power macromodel

Given that f is the flit size[7] of a packet and h is the hop count[8] of the packet's route, the number of routers through which the packet travels is $h + 1$ and the total flit size of a packet including its header is $f + 1$. The value for f varies depending on the size of bits used in the application model. We assume the application model is based on 16 or 32 bits. Given that P_r is the power dissipated by a router to transmit a flit (assume that the power dissipated by a router to transmit a header flit and data flit is the same), the amount of power dissipated by the routers can be defined as

$$P_{router} = (h + 1)(f + 1)P_r.$$

Before packets are ready to be transmitted over the NoC, a payload coming from an IP core is packetised first into several packets in the NI of the sending router. Upon arrival, the packets are depacketised by the NI of the receiving router before the payload is forwarded to the destination IP core. Different protocols and mechanisms used in the NI affects its power dissipation. For this power model we assume that the NI has minimum buffering and supports OCP 2 and AHB protocols [146]. Assuming the power dissipated by an NI to process a flit is P_n and since a packet is processed twice by these NIs, the amount of power

[7]The number of flits in a packet.
[8]A hop is the distance between two directly connected routers in a NoC.

dissipated by both NIs is

$$P_{ni} = 2(f + 1)P_n.$$

A packet traverses through a number of links, which is equivalent to the hop count (h) between the source and destination. The power dissipated by a link can be affected by its size. For our power model, we assume the size of the link is 32 bits. Given the power dissipated by a link to transmit a flit is P_l, the amount of power dissipated by all links can be expressed as

$$P_{link} = h(f + 1)P_l.$$

Then, the summation of power dissipated by all the network components is defined as

$$P_m = P_{router} + P_{ni} + P_{link}.$$

After substitution

$$P_m = (h + 1)(f + 1)P_r + 2(f + 1)P_n + h(f + 1)P_l. \tag{4.1}$$

4.5 Task Mapping Configuration

Generally, to find feasible task mappings with the multi-objective optimisation algorithm, the individuals of the population are evolved in a way that improves the trade-off between the schedulability and power dissipation objectives. For this purpose, the multi-objective optimisation algorithm requires a uniform solution representation to encode every task mapping as an individual. The values in a chromosome define an individual and all individuals in the population share the same chromosome structure comprising a group of genes.

An individual's chromosome used in the proposed approach of this chapter only represents a task mapping. The chromosome contains information related to a task mapping and it is similar with the first part of chromosome used in section 3.5. However, it does not include the second part of the chromosome (priority assignment). This is due to our main focus to address the task mapping optimisation problem of conflicting objectives. Figure 4.5 is included to show one possible way to encode the task mapping.

Every gene in the chromosome represents a task (such as τ_1 or τ_2) of the application to be mapped on the multicore platform. The length of the chromosome in terms of the number of genes is equivalent to the total number of tasks (n). Each task is mapped only once onto a core, but each core can be shared by multiple tasks. Every core is defined as an index from 0 to $k-1$, where k is the total number of cores in the platform. Then, it is straightforward that each gene contains an index of a processing core onto which the task that it represents will be mapped. For example, task τ_3 will be mapped onto the processing core with index 9. A core can be shared by multiple tasks, for example the processing core with index 2 appears twice, in the genes that represent task τ_1 and task τ_2.

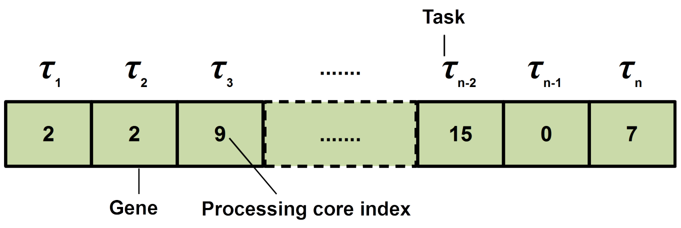


Figure 4.5: Task mapping chromosome structure

4.6 Schedulability and Power Objectives

The focus of the proposed multi-objective optimisation technique is to address the optimisation problem of finding task mappings when multiple objectives exist. These objectives are

- **First objective** (Obj_1) : minimising the number of unschedulable tasks and flows
- **Second objective** (Obj_2) : minimising NoC power dissipation

Since the first objective of the multi-objective optimisation algorithm is similar to the schedulability objective discussed in section 3.6, the same objective function in equation (3.8) can be applied. Further details of the function are explained in section 3.6.

$$Obj_1 = \min(S)$$

The second objective is the minimisation of NoC power dissipation, for transmitting packets over its network components. The equation of the second objective is shown in equation (4.2), where P_m is the estimation of total power dissipated by each component of NoC when transmitting packet m. This metric is calculated using equation (4.1).

$$Obj_2 = \min(\sum_{m=1}^{l} P_m) \tag{4.2}$$

The proposed optimisation process does not take into account the energy dissipated by the execution of tasks in each individual core, as that metric does not contribute to the ranking of alternative mappings in terms of the overall energy dissipation of the system (that is, in communication all cores will dissipate roughly the same amount of energy to execute a particular task). This situation would be of course different if our optimisation were to also include thermal balance as one of its objectives, but this is left as future work.

4.7 Evaluation

4.7.1 Test benches and Baselines

The main purpose of this evaluation is to compare between a single-objective and a multi-objective optimisation approaches. To provide a fair comparison, we selected similar test benches used to evaluate the previous single-objective optimisation algorithm [11] for mapping hard real-time task sets on NoC platforms. The first test bench is the Autonomous Vehicle Application (see Appendix A). The second test bench is the Synthetic Application (SAP). SAP consists of 50 real-time tasks and 50 real-time messages and has shorter task and message inter-arrival intervals (periods), so that it becomes more intense in communication than AVA and harder for the optimisation algorithm to map onto the platforms. Both test benches were mapped onto the 4x4 and 5x5 2D-mesh NoC platforms to find a schedulable mapping, which were similar with the platforms used in the previous work [11].

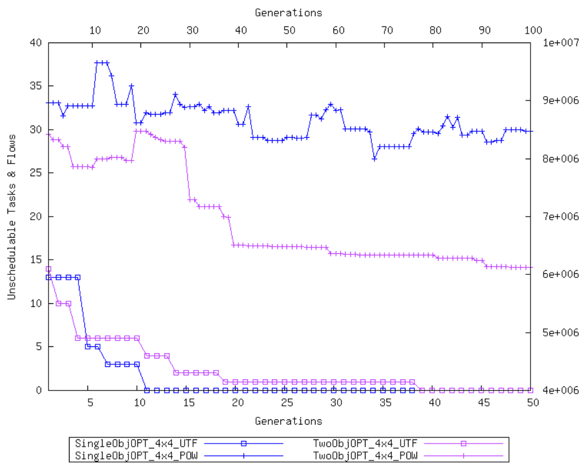
A set of mapping algorithms including the Single Objective Genetic Algorithm (SOGA) proposed in [11], a random mapper and a Nearest Neighbour (NN) mapper were used as baselines. NN mapper is similar as the baseline (NN) used in the experimental work of section 3.8.2. The random mapper randomly allocates tasks onto the two platforms.

Equation 4.1 calculates the NoC power dissipation by assuming the values for P_{router}, P_{link} and P_{ni} are given. For this experiment work, we selected the values of these parameters based on the results published in [145]. The power dissipated by a router is 8% higher than that dissipated by an NI and the power dissipation ratio between a router and a link is equal to one. These values were captured from the power analysis based on a 5x5 NoC with 32-bit router and 4-flit input FIFO buffers, and an NI with minimum buffering and supports the OCP 2 and AHB protocols. The NoC was designed in Verilog HDL at the RTL level, synthesised with Synopsys Design Compiler and mapped onto an UMC 65 nm technology.

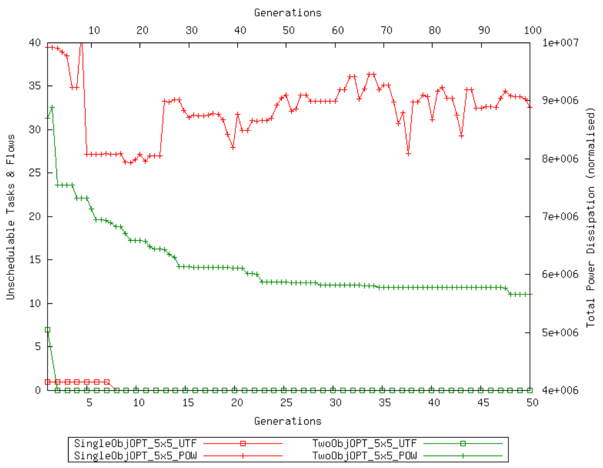
4.7.2 Results

The hypothesis for this experiment states that the mapping solutions found by MOGA will be as good as or better than the solutions produced by SOGA in meeting hard real-time timing constraints, and always better in power dissipation. It is expected that the latter part of the hypothesis would be easy to demonstrate because SOGA does not optimise power dissipation. The challenging part is to show how MOGA can quickly converge towards fully schedulable solutions, which is not straightforward as it also has to keep many low-power task mappings within the population at every generation. Following the evidence that supports the experimental hypothesis, another experimental study was carried out with the aim of showing that GA-based task mapping optimisation can produce mappings that are far better than the random and NN mappers.

Figures 4.6 and 4.7 shows the convergences of the best task mapping found by

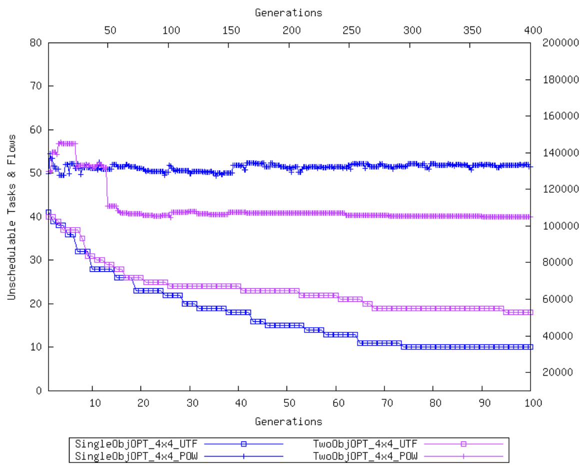


(a) AVA, 4x4

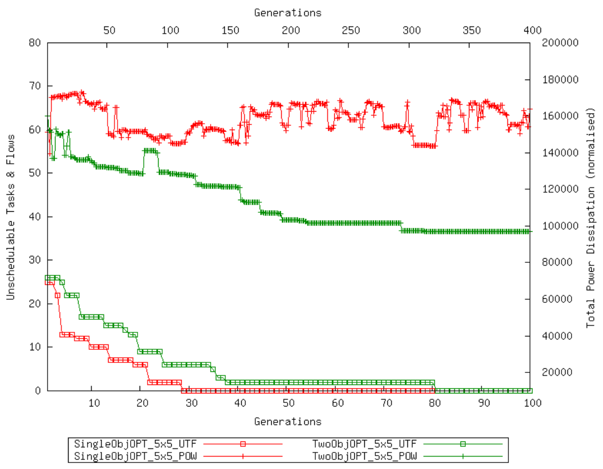


(b) AVA, 5x5

Figure 4.6: Task mapping convergence over generations between MOGA and SOGA using AVA mapped onto 4x4 and 5x5 platforms



(a) SAP, 4x4



(b) SAP, 5x5

Figure 4.7: Task mapping convergence over generations between MOGA and SOGA using SAP mapped onto 4x4 and 5x5 platforms

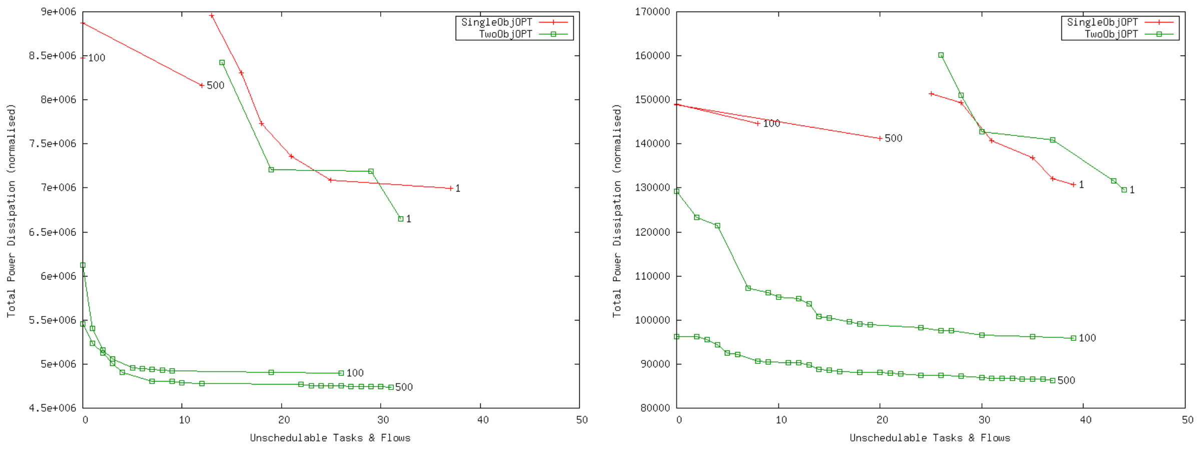
SOGA and MOGA over generations when mapping AVA and SAP each onto 4x4 and 5x5 platforms. The test benches were mapped onto the platforms to show the improvement in both metrics: the schedulability of the system given as the total number of unschedulable tasks and flows, and NoC energy dissipation. Two vertical axes shown on each graph plot the number of unschedulable tasks and flows (on the left axis, labelled *UTF*) and the total NoC energy dissipation (on the right axis, labelled *POW*) respectively. The latter metric is normalised by the power dissipated by a single flit over a single hop. To show the improvement in both metrics over generations, the horizontal axes of the graphs represents the scale between first and last generation.

In order to plot the graphs, the best task mapping must be chosen from among the solutions of the population. For single-objective optimisation, task mappings can be fully ordered, hence the best task mapping is evident. For example, the best task mapping is a task mapping with zero or the fewest unschedulable tasks and flows. The best task mapping is then evaluated in terms of power dissipation to gain its second fitness value. In multi-objective optimisation, selection was done differently because a single best solution may not exist. In a non-dominated set, normally more than one solution exists and all members of that set can be considered as best solutions. Therefore, a task mapping with zero or the lowest number of unschedulable tasks and flows among them was selected. If more than one task mapping in the set has the same fitness value, a task mapping with the lowest power dissipation is preferred.

Both optimisation algorithms, MOGA (labelled *TwoObjOPT*) and SOGA (labelled *SingleObjOpt*), converged to a fully schedulable system for AVA on both 4x4 and 5x5 platforms. As shown in Figures 4.6a and 4.6b, in fewer than 50 generations both algorithms were able to find a task mapping that could produce a schedulable system. In terms of convergence rate, SOGA converged faster than MOGA on the 4x4 platform, but contrarily on the 5x5 platform, in that MOGA converged slightly faster than SOGA. In terms of power dissipation, SOGA's performance was inconsistent. Although some reductions were observed, the trend was not maintained in the long run. Its inconsistent performance is depicted in both graphs for all the platforms where AVA was mapped. Conversely, MOGA obtained mappings which met all real-time constraints and at the same time consistently improved the total power dissipation, better than SOGA.

In chapter 3, a comparison between our proposed approach (SCGA) and the GA(RAN) was conducted, and both algorithms optimised task mapping based on the single objective. From the results shown in Figure 3.15, we implied that the former algorithm has better convergence than the latter algorithm based on the mapping of AVA test bench onto the 4x4 platform. With the same test bench and platform, in this chapter, the convergence of a single-objective GA (SOGA) and another proposed approach known as MOGA, which is based on multi-objective optimisation, was compared and the results is shown in Figure 4.6. It is interesting to analyse the performance of MOGA and SCGA because the latter approach is based on the single-objective optimisation but performed better than the single-objective GA baseline, although it manipulated another parameter (priority assignment) in its chromosome for improving the schedulability of the task mapping. One similarity in the comparisons are the GA that we used as the baseline: GA(RAN) and SOGA performed a single-objective optimisation and optimised task mapping. Based on the similar baselines, we analysed the performance of SCGA and MOGA based on the results depicted in both figures. From Figure 4.6, MOGA converged after SOGA has found a schedulable task mapping but SCGA outperformed GA(RAN) as shown in Figure 3.15. Therefore, we imply that SCGA has better optimisation runtime (in number of convergence) than MOGA based on the mapping of AVA onto the 4x4 platform.

On the other hand, MOGA showed better performance than SOGA when mapping SAP onto 4x4 and 5x5 platforms, outperforming the latter algorithm in terms of power dissipation. It is worth noting that SAP was synthetically created with specific timing constraints, which makes it harder to find a schedulable mapping on a 4x4 platform. This is the reason behind the unsuccessful mapping by both algorithms to achieve a schedulable system with SAP on the 4x4 platform. In spite of this result, MOGA was still able to reduce the power dissipation in NoC below what was achieved by SOGA, as shown in Figure 4.7a. Both algorithms produced improved performances when mapping SAP onto the 5x5 platform. Although, MOGA and SOGA converged to a schedulable system with their best task mapping in fewer than 100 generations, the latter algorithm showed a faster convergence than MOGA. For all cases including SAP mapped onto the 5x5 platform, as depicted in Figure 4.7b, MOGA's performance in terms of power dissipation was better than that of SOGA. Based on the chosen benchmarks, our experimental hypothesis has been validated by the MOGA's performances, which produced schedulable task mappings similar to those produced by SOGA, but always better in power dissipation.



(a) AVA, 4x4 (b) SAP, 5x5

Figure 4.8: Non-dominated sets at certain generations (1, 100 and 500), using AVA with a 4x4 platform and SAP with a 5x5 platform

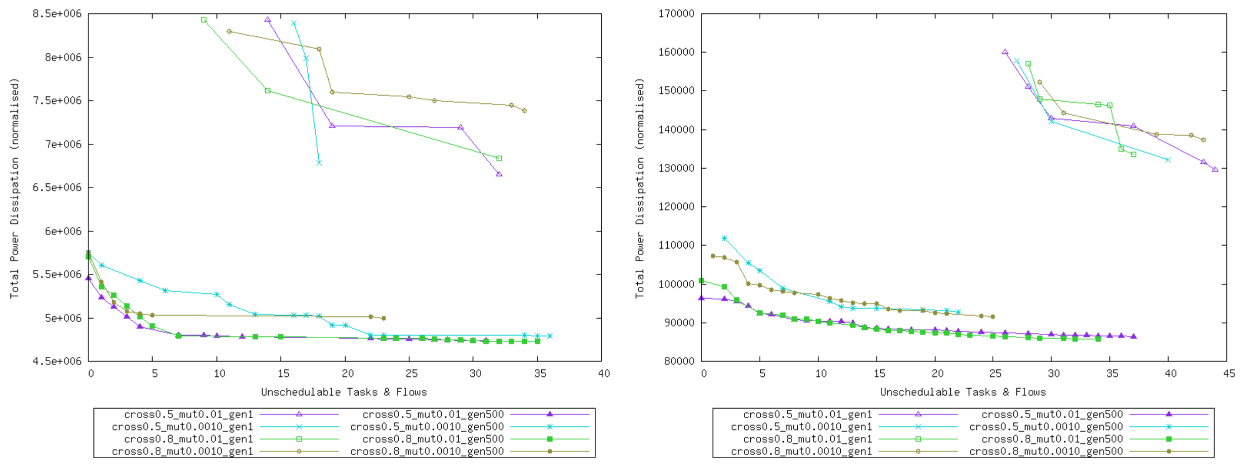
The plot in Figure 4.8 shows the level 1 non-dominated set selected at specific generation (1, 100 and 500) for both MOGA and SOGA. To enable the comparison, task mappings found by SOGA were filtered in a similar way to that applied by MOGA to find the non-dominated solutions. This is possible once the power dissipation fitness value is yielded for every task mapping before the offspring population becomes the new parent population in the next generation. With both fitness values, the Pareto-optimal concept can be applied to the population for determining the first level of the non-dominated set.

The non-dominated sets produced by MOGA dominate all the sets produced by SOGA. As shown in Figures 4.8a and 4.8b, at 100 and 500 generations MOGA has improved the trade-off of its task mappings by minimising both metrics lower than the SOGA's task mappings. In addition, MOGA showed better convergence towards the optimal region of the solution space (the lower-left corner), where fully schedulable low-power solutions were found. A schedulable mapping (that is, touching the vertical axis) was found at 100 and 500 generations, but again the mappings found by MOGA had much lower power dissipation. As shown in the plots, there is a significant difference in power dissipation, which highlights how much quality improvement can be made when both objectives are taken into account.

MOGA has several parameters that can be configured to achieve specific performances. An additional experiment was conducted in order to show the influence of the parameters. This was achieved by careful parametric analysis, which included the ranges shown in Table 4.1. MOGA was executed with each configuration to perform mapping on 4x4 and 5x5 platforms using AVA and SAP test benches.

Table 4.1: List of GA parameter values used in parametric analysis

GA Parameters	**Values**
Population Size	100
Crossover Rate	0.5, 0.8
Mutation Rate	0.01, 0.001
Max Generations	500



(a) AVA, 4x4 (b) SAP, 5x5

Figure 4.9: Non-dominated sets produced with different GA configurations at first and last generations

The results are plotted in Figures 4.9a and 4.9b, in which the configurations are compared against each other using the level 1 non-dominated set at first and last generations. For both test benches, MOGA performed better when configured with 0.5 crossover rate, 0.01 mutation rate, 100 population size and allowed to run for 500 generations. Furthermore, MOGA was configured with the best setting and was compared with SOGA and other baselines (NN and random mappers) in both metrics. Figure 4.10 shows the outstanding performance of MOGA in finding a fully schedulable task mapping with lower power dissipation than all the baselines, which validates once again our experimental hypothesis.

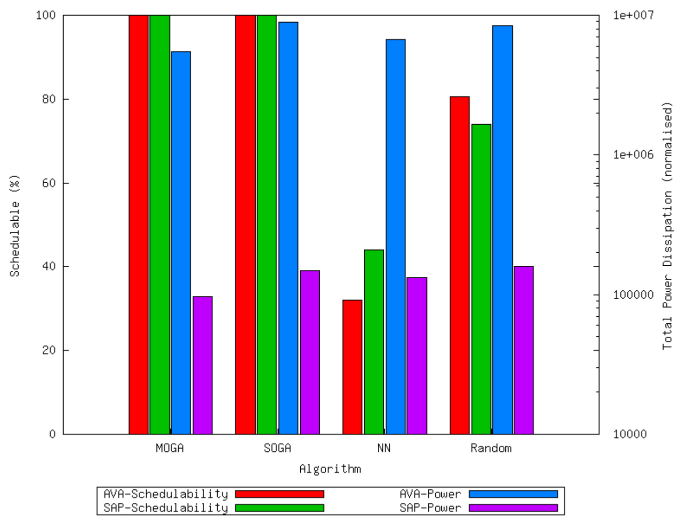


Figure 4.10: Overall comparison against several baselines

4.8 Summary

The main contribution of this chapter is a multi-objective optimisation technique which could find a schedulable task mapping whilst minimising the energy dissipation of a NoC-based hard real-time embedded system. This was achieved by the integration of analytical fitness functions into the algorithm, which consisted of an energy macromodel to estimate the energy dissipated by NoC and an extended end-to-end schedulability analysis that can validate the schedulability of any task or traffic flow in the system. The algorithm is feasible for the early design space exploration; it is able to explore many task mappings in the existence of more than one conflicting objectives. We have illustrated its feasibility with two case studies and we have shown that the algorithm could find better trade-offs between both objectives than the single-objective GA and some baselines.

Chapter 5

Breakdown Frequency Metric for Task Mapping Evaluation

In the previous chapters, the proposed single and multi-objective optimisation techniques have produced schedulable task mappings based on the end-to-end response time analysis. From this analysis, the numbers of unschedulable tasks and messages are calculated as a fitness value of every task mapping. This is a convenient metric that facilitates the optimisation techniques to find task mappings which keep all tasks and flows schedulable. However, it has also been reported that in some cases those task mappings cannot be found and the system would never become fully schedulable. The metric only suggests that if all tasks and messages are schedulable then the task mapping is feasible for the system, otherwise it is not. This limits the ability of the optimisation techniques due to the limited information that the metric can provide to facilitate exploration of alternative task mappings. As the result, system designers may have limited choice of task mappings and this could lead to designs that have unnecessary increases in complexity and cost. A fitness function is proposed to overcome this problem, one which allows the optimisation algorithm to explore task mappings more effectively.

In this chapter, section 5.2 explains the proposed fitness function. The optimisation algorithm and its objective are described in section 5.3. Then, section 5.4 discusses the evaluation results and finally section 5.5 summarises the proposed technique.

5.1 Limitation of the Schedulability Metric

The schedulability of a hard real-time system could be determined by applying a Real-Time Analysis (RTA) to analyse the response time. With this analysis, a task's end-to-end response time upper bound can be calculated and validated against its deadline. A schedulability metric based on the number of unschedulable tasks and messages can then be yielded and used as a convenient metric for determining the schedulability of every task mapping. The analysis serves as the underlying evaluation mechanism that facilitates task mapping optimisation algorithms to find a schedulable task mapping. A well-known optimisation algorithm such as GA made it possible to optimise several task mappings at the same time. At the early optimisation process, task mappings are unschedulable, but as shown in Figure 2.8 (see line a) further optimisation improves the task mappings to become fully schedulable.

Although, the function may seem to be useful in bringing the algorithm to a successful convergence, it has also been reported that in some cases finding the feasible task mapping was unsuccessful [11, 21]. This could possibly occur if some of the shared resources are exhausted by a large number of tasks and flows to the extent that enormous interference results because of the inefficient task mapping. As a result, the plot in Figure 2.8 would show a line (see line b) that never touches the 100% schedulability level. The current schedulability fitness function provides a metric which limits the potential of a task mapping optimisation algorithm to explore alternative task mappings. This may hinder system designers' insight on other task mappings, which could be better than previous task mappings. Limited choice of task mappings may lead system designers to redesign the system based on the understanding that the current platform is unable to achieve a fully schedulable system. For example, the number of cores can be increased to support hard real-time tasks and messages. The system may be easily schedulable, but at the expense of increased complexity, area and cost.

Furthermore, in order to use the schedulability fitness function it is assumed that the WCET of every task is known in advance, which is determinable at the nominal operating frequency using several well-known techniques such as [68]. It is sufficient with the fitness function to evaluate the schedulability of the system if the designer's concern is about finding feasible task mappings for the system running at that frequency only. In terms of reducing energy dissipation, it is always beneficial to create a system with lower operating frequency. Unfortunately, with the current fitness function the potential of finding the task mappings that could make the system schedulable is limited. Certainly, increasing the operating frequency will make the system schedulable, but without a guarantee that the frequency values are minimal.

5.2 Breakdown Frequency Scaling

A task is deemed unschedulable if its deadline is missed, as depicted in Figure 5.1. This could happen when, for example, a task has to wait for the execution of higher priority tasks, messages have to travel a long path over the NoC, or they are delayed by congestion. It may be possible to improve this condition by increasing the operating frequency (f') to speed up the execution time of the task and the transmission time of its messages. Consequently, the end-to-end response time of the task is reduced, enabling it to meet its deadline as shown in Figure 5.2.

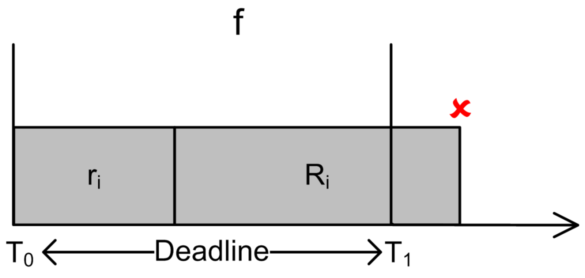


Figure 5.1: Response time (r_i) of task i and latency (R_i) of messages at the nominal frequency (f)

Initially, this seems like an unacceptable trade-off, since increasing the operating frequency could lead to high power consumption. Therefore, in this chapter a new fitness function is proposed with the aim to find the minimal frequency

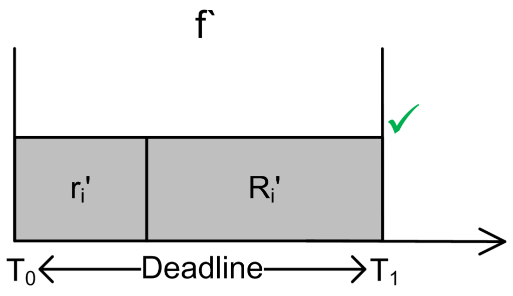


Figure 5.2: Response time (r_i) of task i and latency (R_i) of messages at the increased frequency f'

that makes the system schedulable. The minimal frequency is referred to as the *breakdown frequency* and can be applied as a property for every task mapping for facilitating the search for the schedulable task mappings. For example, the plot in Figure 5.3 shows the breakdown frequency of the best task mapping found by the optimisation process at each point in time (normalised to the nominal frequency of the system). During the early optimisation process, the breakdown frequency of the task mapping is likely to be much higher than the nominal frequency, but nonetheless with its breakdown frequency the system becomes schedulable (unless there are starving tasks or messages, or the breakdown frequency is higher than what the system can achieve). The optimisation process then tries to minimise the breakdown frequency so that the system can become schedulable at a lower operating frequency. The point where the plotted curve touches the horizontal line ($f = 1$) is equivalent to the point where the plotted line a touches the 100% schedulability in Figure 2.8, that is, the optimisation has found a fully schedulable mapping at the nominal frequency. From that point onwards, the proposed fitness function allows the optimisation process to improve the mapping, lower its breakdown frequency even further and at the same time maintains the schedulability.

The breakdown frequency is defined as the minimal frequency value at which every task and flow executes on the system without missing its deadline. This value represents a group of frequencies: one each for the processors and the NoC. In other words, the processors and the NoC could run at different frequencies, but we let the search algorithm to scale both frequencies at the same time with

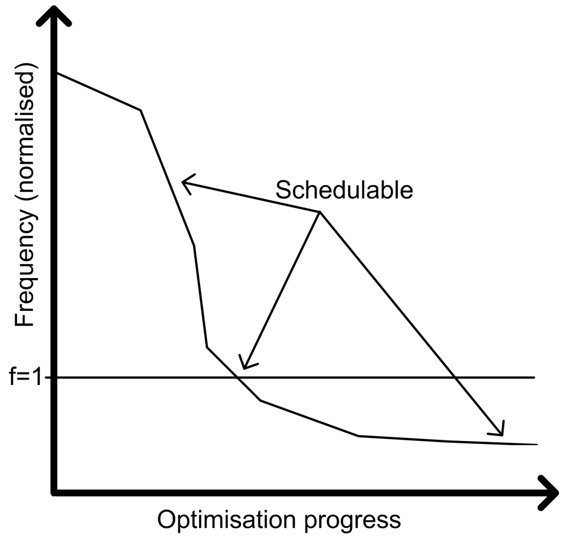


Figure 5.3: Advantages of using the breakdown frequency for improving task mappings

the same multiplicative factor. We assume that the execution of tasks and flows scales linearly with the breakdown frequency and with this assumption we ignore the effects of memory latency and bandwidth on the processor performance [147].

The breakdown frequency is a frequency bound permitted in the system for tasks and flows to run without missing their deadlines. Below this frequency, some of the tasks and flows will become unschedulable in the system. On the other hand, as mentioned in section 3.2, the worst-case execution time of a task set and the basic latency of flows are calculated under the nominal frequency. The breakdown frequency value could be lower, higher or equal to the nominal frequency depending on how it scales. If the breakdown frequency is higher than the nominal frequency, tasks and flows will perform faster than the execution at the nominal frequency. If some of the tasks or flows in the system cannot be schedulable at the nominal frequency, the system could make them schedulable by running at the breakdown frequency. Conversely, the system will benefit from low operating frequency if the breakdown frequency is below the nominal frequency. Although, the tasks and flows execute slower than at the nominal frequency, all tasks and flows are guaranteed to be schedulable at the breakdown frequency.

Similar to the schedulability metric (see equation 3.7), the breakdown frequency can be applied as a metric for every task mapping found during optimisation. With this metric, a search-based optimisation algorithm will be able to determine which task mapping has the potential to produce a schedulable system with the minimal operating frequency. Usually the algorithm searches for a group of task mappings, and needs to choose only the best task mapping from the group. With the breakdown frequency as one of task mapping properties, the algorithm can sequentially rank the task mappings from low to high frequency. Then, the selection of the best task mapping is by choosing the one that meets or comes nearest to the optimisation objective. As the optimisation progresses over time, the task mappings will be improved, further lowering the breakdown frequency.

It is worth noting that it is still useful for the optimisation algorithm to determine the breakdown frequency of a given mapping even if that frequency is infeasible for the processors or the NoC. During optimisation it facilitates the algorithm in the task mapping selection process. In practice it helps designers to choose the operating frequencies for the processors and the NoC, as long as the breakdown frequency is lower than at least one of the feasible frequencies.

Taking a task mapping as input, two main steps relate to each other in finding the breakdown frequency that will become one of its properties. These steps are frequency scaling and schedulability analysis. Frequency scaling plays the role of choosing a candidate frequency value from a given range of settings: a finite set of possible frequency settings defined for the system. Following the choice of the frequency on the system, an analysis that serves as the second step determines the schedulability of all tasks and flows. A fitness function hereafter known as the Breakdown Frequency Fitness Function (BFF) integrates both steps to calculate the breakdown frequency.

Specifically, the function's frequency-scaling step gradually scales a finite set of frequencies until no further frequency setting is available for evaluation. The selection of frequencies is defined between a minimum and a maximum frequency setting, which is normally used in embedded systems. It is assumed that the selected setting represents a set of frequencies of the processing cores and NoC. The frequency range can be expanded or decreased accordingly prior to the optimisation, but remains fixed for the duration of the whole optimisation. It is worth noting that the function does not limit the choices of scaling algorithms, hence any suitable algorithm capable of exploring the frequency set can also be applied.

A binary search algorithm was chosen as the scaling algorithm in this work not only due to its simple implementation, but also because it can halve the number of items to check in the finding of the breakdown frequency. This reduces the number of frequency settings to test before the breakdown frequency is found. The frequency range is sorted in an ascending array and the scaling algorithm starts selecting a frequency from the middle index of the array. If the selected frequency does not make all the mapped tasks and flows schedulable then it selects a new frequency from the sub-array on the right of the middle element (increase). If they are schedulable, then it selects from the sub-array on the left of the middle element (decrease). The scope of selection from the array is reduced by shifting to the left or right sub-array. When no more scaling can be performed, that is, the selection of the middle index returns null, the function will return the breakdown frequency for the task mapping.

The breakdown frequency (F') is defined as a property (fitness) of a given task mapping. For every task mapping, BFF calculates the breakdown frequency of a given task mapping according to the process flow shown in Figure 5.4. A task mapping defined as x becomes the input parameter of the function. Equation 5.1 yields the breakdown frequency as the fitness value of the task mapping.

$$F' = f(x) \tag{5.1}$$

Schedulability evaluation of all tasks and flows, based on the method described in subsection 3.3, is a repeating process performed for every selected frequency applied to the system. The output is the number of unschedulable tasks and flows, which indicates how schedulable the system is under the selected frequency. This metric is useful when determining the breakdown frequency of the task mapping. For example, if the number of unschedulable tasks and flows is zero, this indicates that the system is schedulable and the algorithm will reduce the frequency further. Otherwise, it will increase the frequency. If the algorithm has no further frequency setting to evaluate, then the lowest frequency that makes the system schedulable

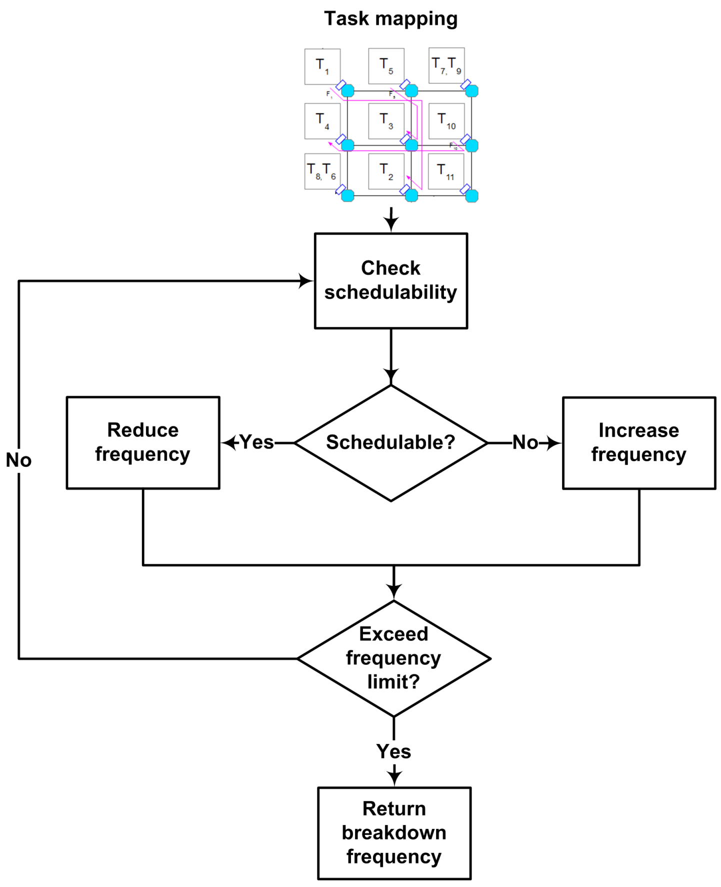


Figure 5.4: Process flow of the breakdown frequency fitness function

is returned as the breakdown frequency for the task mapping. In a case where the breakdown frequency cannot be found (for example, because it needs a larger setting than the available values in the frequency range), the task mapping will be tagged to indicate that the breakdown frequency is not available from within the specified frequency settings. Interested readers are referred to Appendix B for the pseudo-code of the breakdown frequency fitness function.

5.3 Optimisation Objective and Solution

The optimisation process is similar to the process described in section 3.4 except that the fitness function, the optimisation objective and the metric used to evaluate each task mapping are different. Therefore, in this section only the differences are explained in detail. Readers are referred to section 3.4 for a detailed explanation of every step in the optimisation process.

The breakdown frequency fitness function does not limit the configuration of other design parameters, but in the proposed approach only the task mapping is considered as input. This reduces the complexity at the function side while allowing close observation of the impact on the task mapping improvement made by the proposed approach.

Using GA as the optimisation algorithm produces a population containing several individuals. Evolution of these individuals happens in a number of generations. As before, the maximum number of generations is the termination condition of the algorithm, that is, the improvement of the population stops at this point.

As shown in Figure 3.5, individuals are refined by a set of operators over several generations. Each operator has a specific role in the process, identical to the evolution of living organisms in the real world. A uniform chromosome structure as depicted in Figure 4.5 is a working unit for all the operators. With the chromosome structure, representation of all individuals as task mapping is described in a similar way. As in nature, a chromosome is built upon a set of small units called genes. Each gene represents a task and contains a processing core index (given as an integer number) to which a task should be allocated in the system. Decoding the chromosome reveals an instruction on how to map all tasks onto the processing cores of the system. Interested readers are referred to section 4.5 for further details of the structure of a chromosome.

In general, the optimisation algorithm explores the design space of the system by selecting the best task mappings that have fitness values which match with the optimisation objective, or at least come near to it. The optimisation objective is to minimise the breakdown frequency of the system. As shown in equation 5.2, F' is the breakdown frequency of a given task mapping, therefore the optimisation objective is given by

$$Obj_1 = \min(F') \tag{5.2}$$

5.4 Evaluation

5.4.1 Test benches and Baselines

In this experiment work we selected ten test benches including one realistic and several synthetic applications to study the feasibility of the proposed approach in addressing the task mapping optimisation problem. The realistic test bench used for this experiment was AVA (see Appendix A), and the interested readers are referred to section 3.8.1 for further explanation on the test bench. The rest of the test benches were synthetically created and numbered from TB-1 to TB-9 according to an increasing order of task set utilisations. Various utilisation levels were achieved by varying the number of tasks and flows, and the execution time of tasks. Specifically, TB-1 consisted of a task set with 625% utilisation, followed by TB-2 (724%), TB-3 (975%), TB-4 (1125%), TB-5 (1875%), TB-6 (3850%), TB-7 (5500%), TB-8 (6500%) and TB-9 (7500%). For TB-1 to TB-5, each contained a set of messages with 1075% communication utilisation, whilst TB-6 to TB-9 were 8488% each. Every task had a unique priority and a flow inherited the same priority as its sending task. The test benches were mapped onto three different sizes of platform, 4x4, 5x5 and 10x10 mesh NoCs.

The runtime of the proposed approach was expected to be longer than the baselines due to the iterative evaluation for finding the breakdown frequency. Based on the approach, every task mapping in a GA population must be assigned with a breakdown frequency fitness but depending on the size of population the overall runtime of the optimisation process might take a significant amount of time to complete. In order to keep the runtime of the optimisation reasonable, only certain synthetic test benches with specific workloads were selected for the evaluation purpose. The selection of workloads for these test benches (from TB-1 until TB-9) were based on the performance shown by the GA used in the previous chapters, that is in difficulty of finding a schedulable task mapping in a 4x4, 5x5 and 10x10. The main reason is to observe how much improvement the proposed approach can provide under these workloads. Take an example of TB-4 test bench with total task utilisation of 1125%, which takes around 70% of a 4x4 platform, it was difficult to find a schedulable task mapping with the previous GA-based optimisation algorithm. We were interested to see how much the proposed approach could improve the task mapping with the same test bench.

In order to show the improvement achieved by the proposed fitness function, the Schedulability Fitness Function (SCF) from previous work [11] was implemented to yield the schedulability metric as the property of a task mapping. An optimisation algorithm that depends on SCF as the fitness function optimises task mapping in a way that reduces the number of unschedulable tasks and flows at the nominal frequency. The comparison between both optimisation algorithms determines which of the two fitness functions provides a more useful fitness value for task mappings: BFF which finds the breakdown frequency for a task mapping to become schedulable or SCF which calculates the number of unschedulable tasks and flows at the nominal frequency.

It should be noted that the difference between the two fitness functions can be described in terms of the fitness value yielded by each of them as follows:

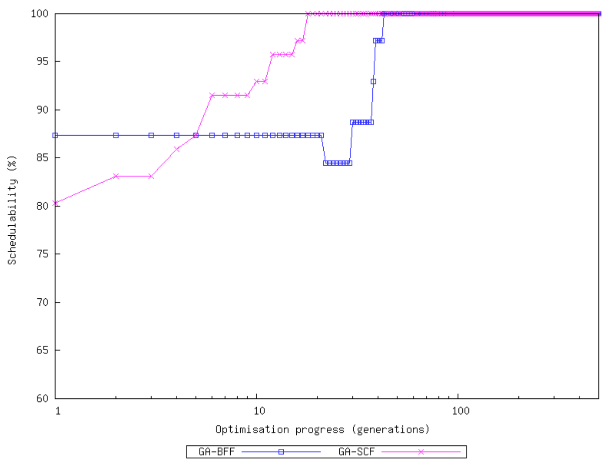
- The Breakdown Frequency Fitness Function (BFF) yields the breakdown frequency as the property of a task mapping;
- The Schedulability Fitness Function (SCF) yields the number of unschedulable tasks and flows as the property of a task mapping.

The optimisation algorithm used for this experiment is based on the same single-objective GA baseline used in section 3.8.2. GA is a meta-heuristic and, based on the fitness functions, two different instances of the algorithm were created, referred to hereafter as GA-BFF and GA-SCF. The main difference between GA-SCF and GA-BFF is their optimisation objectives, although both have the same main aim of producing a schedulable task mapping for the system. The objective of each algorithm is derived from the metric of its fitness function. For example, the objective of GA-SCF is to minimise the total number of unschedulable tasks and flows, whilst the objective of GA-BFF is to minimise the breakdown frequency of the system. GA's basic operations are based on evolutionary principles such as the crossover of parents' chromosomes and the mutation of genes to create diversity between individuals. GA's operations demand a good setting to perform efficiently. The same GA settings as shown in Table 3.4 were used to configure both GAs.

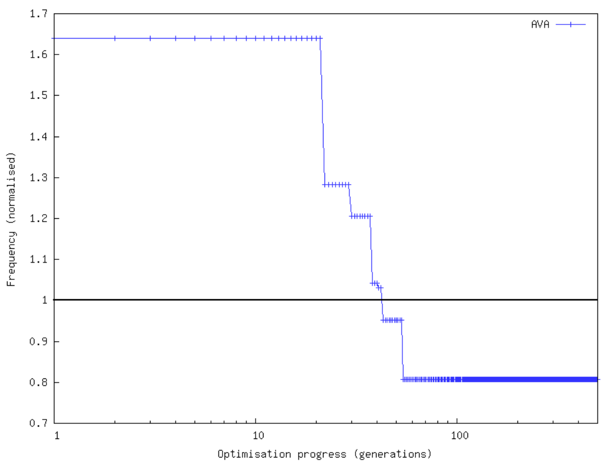
5.4.2 Results

The purpose of this experiment was to demonstrate the feasibility of BFF as a fitness function for task mapping optimisation. With the fitness function, the optimisation algorithm will find the breakdown frequency of a given task mapping which makes all tasks and flows fully schedulable. The experimental hypothesis states that GA-BFF is better in finding the schedulable task mapping than GA-SCF.

On a 4x4 platform, as depicted in Figure 5.5a, both GA-BFF and GA-SCF could find a schedulable task mapping for AVA at the nominal frequency. Although both algorithms successfully converged to 100% schedulability between 10 and 100 generations, GA-SCF's performance was faster than that of GA-BFF. The faster convergence shown by GA-SCF could be from the direct minimisation of the real-time quantitative metric, which is the number of unschedulable tasks and flows. Therefore, optimising task mappings at the nominal frequency is rather straightforward. On the other hand, GA-BFF has a different optimisation objective from GA-SCF, which is to improve task mappings based on the minimisation of breakdown frequency. At the beginning of optimisation, the breakdown frequency is usually higher than the nominal frequency. To reach the nominal level, GA-BFF has to scale the breakdown frequency and at the same time maintain



(a) Schedulability



(b) Breakdown frequency

Figure 5.5: Based on the mapping of AVA onto a 4x4 platform in a single run; (a) the schedulability convergence of GA-BFF and GA-SCF at the nominal frequency, (b) the breakdown frequency convergence of GA-BFF

the schedulability of tasks mappings in many iterations. Nonetheless, both are comparably good at finding the feasible task mapping at the nominal frequency and are able to converge in fewer than 100 generations. Moreover, Figure 5.5b shows the GA-BFF convergence below the nominal level, which is a significant improvement over GA-SCF, which could find the schedulable task mapping only at the nominal level (see Figure 5.5a).

GA-BFF could find a schedulable task mapping if a breakdown frequency exists. Furthermore, it facilitates the optimisation process to find the schedulable task mapping at the nominal frequency early during the optimisation, as indicated by point 'K' on the graph depicted in Figure 5.6. This graph shows the cumulative time difference between GA-BFF and GA-SCF to complete task mapping optimisation in 500 generations. It is worth noting that the point 'K' was the time taken by GA-BFF to find a schedulable task mapping at the nominal frequency. The time taken was between 1000 sec and 1500 sec, above the GA-SCF' time (less than 1000 sec), but the difference between the two times was less significant. However, the time overhead of the optimisation increased as the GA-BFF continued frequency scaling for each task mapping. This is due to the

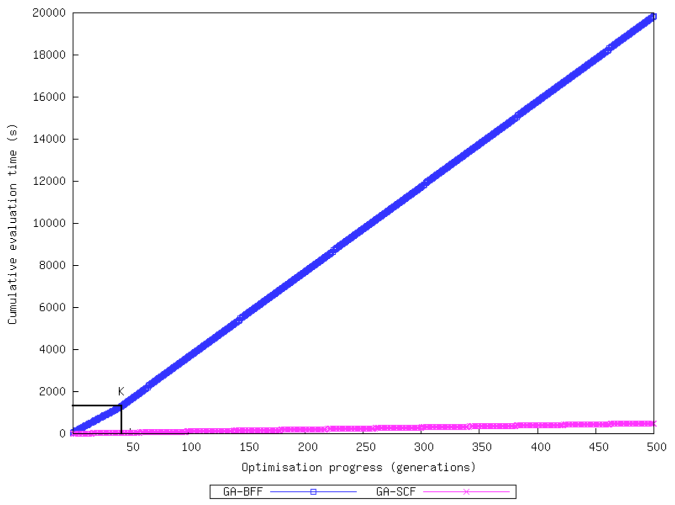
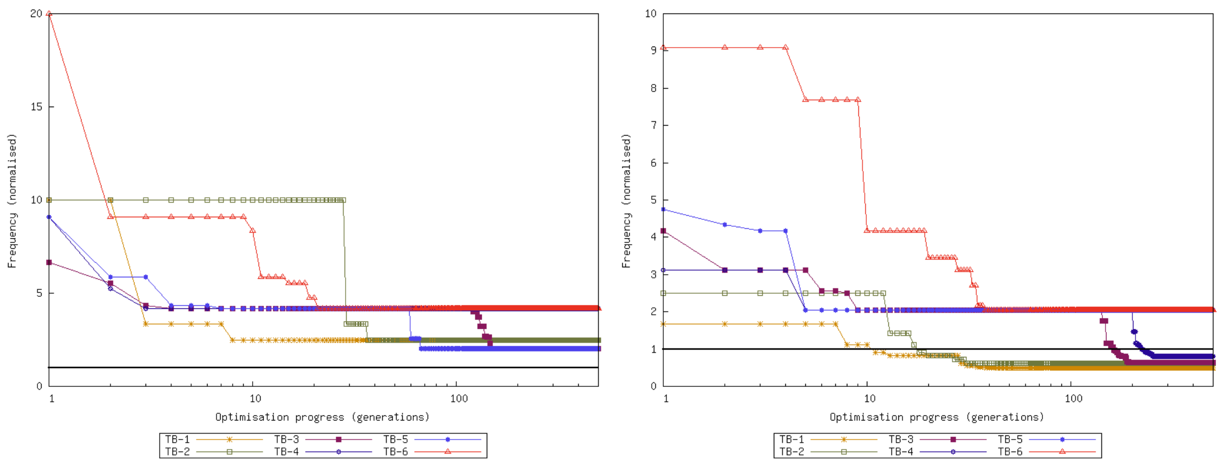


Figure 5.6: Cumulative evaluation time between GA-BFF and GA-SCF and point 'K' indicates the time when GA-BFF found the schedulable task mapping at the nominal frequency, based on AVA and a 4x4 platform

iterative evaluation process of all tasks and messages during frequency scaling until the breakdown frequency of the task mapping can be found (if it exists). This evaluation is necessary for each selected frequency value to determine if it could make all tasks and messages in the system schedulable. As shown in Figure 5.6, GA-BFF consumed more time than GA-SCF and the difference of cumulative time to complete 500 generations between the two GAs is significant. However, the rest of the time beyond point 'K' was spent reducing the operating frequency further below the nominal level, which is an improvement over GA-SCF.

In spite of the time overhead, GA-BFF improved task mappings by finding their breakdown frequency to make all tasks and flows schedulable. By mapping six test benches (TB-1 to TB-6) onto 4x4 and 5x5 platforms, the lowest breakdown frequency of a given task mapping was recorded at each point in time (from the 1st to the 500th generation). The improvement is shown in Figure 5.7a and Figure 5.7b: in each of the graphs a horizontal black line ($y = 1$) is drawn to represent the nominal frequency level. As shown in Figure 5.7a, GA-BFF has successfully found the breakdown frequency for at least one task mapping in every generation. In other words, it found the schedulable task mapping even during the early stage of optimisation and the algorithm maintained its schedulability and gradually



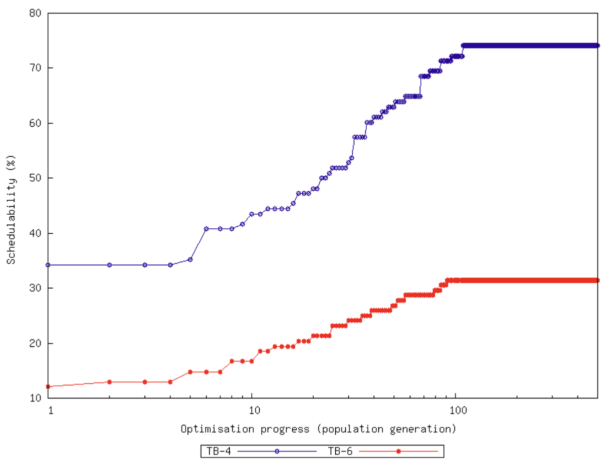
(a) 4x4 platform (b) 5x5 platform

Figure 5.7: Breakdown frequency convergence of GA-BFF based on the mapping of synthetic test benches onto 4x4 and 5x5 platforms in a single run respectively

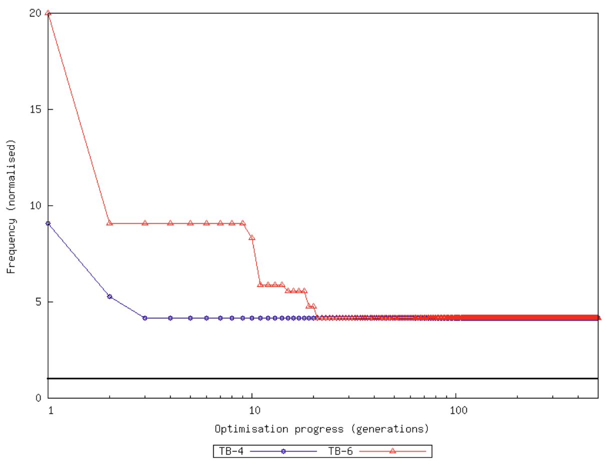
scaled down the breakdown frequency towards the nominal level.

Furthermore, when the platform size was increased to 5x5 and the same test benches were mapped, GA-BFF successfully converged below the nominal frequency as depicted in Figure 5.7b, except for TB-5 and TB-6. With a larger platform, it was easier for GA-SCF to find the schedulable task mappings, but its task mappings limited the schedulability of the system to the nominal frequency only. GA-BFF offers more advantages with the breakdown frequency; all tasks and flows of all the test benches could become schedulable while at the same time allowing the system to execute below the nominal frequency. Similar to the 4x4 platform at the beginning of optimisation, the breakdown frequency was higher than the nominal level, but as the optimisation progressed, the algorithm converged to the nominal level and subsequently fell below it. Although TB-5 and TB-6 had higher utilisation than the other test benches, a schedulable task mapping could still be found by GA-BFF using the breakdown frequency. This improvement shows that by using BFF as the fitness function, GA-BFF could serve as an alternative optimisation algorithm in a case where GA-SCF has failed to find any schedulable task mapping at the nominal frequency.

In some cases, such as when a task set with large utilisation exists, mapping

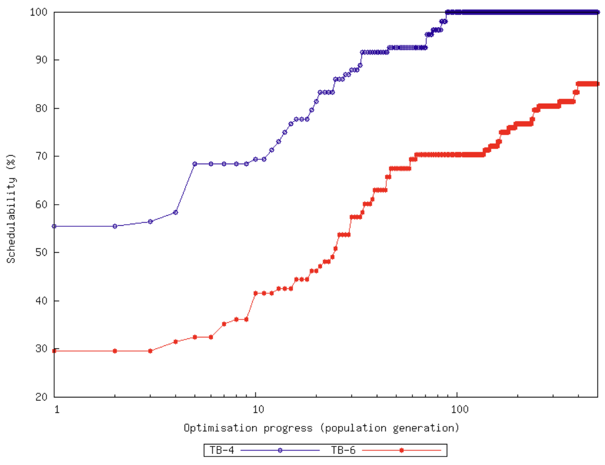


(a) GA-SCF

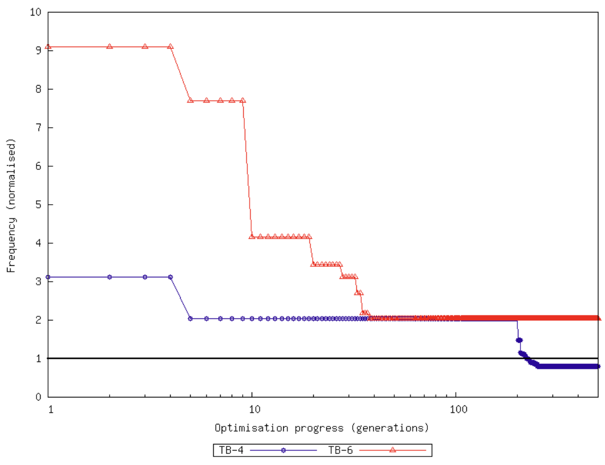


(b) GA-BFF

Figure 5.8: Based on two synthetic applications and a 4x4 platform; (a) GA-SCF schedulability convergence, (b) GA-BFF breakdown frequency convergence



(a) GA-SCF



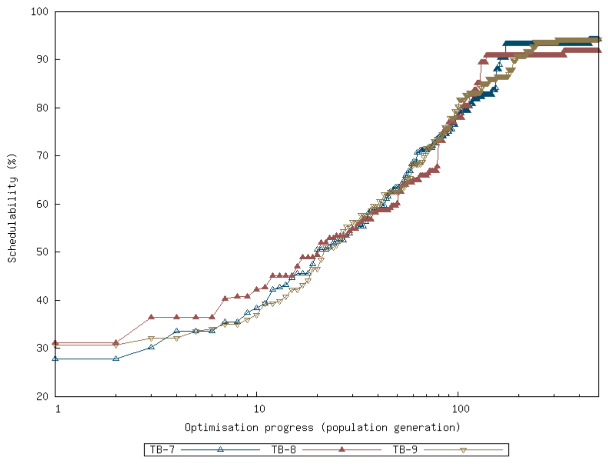
(b) GA-BFF

Figure 5.9: Based on two synthetic applications and a 5x5 platform; (a) GA-SCF schedulability convergence, (b) GA-BFF breakdown frequency convergence

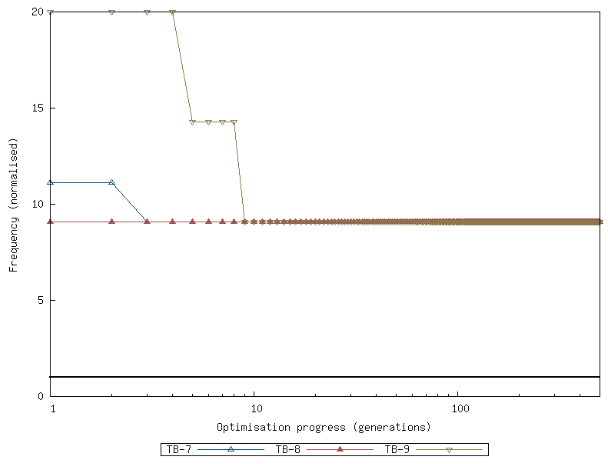
it to smaller platforms using GA-SCF makes the system hardly schedulable. As an example, this can be shown using two synthetic test benches (TB-4 and TB-6), with each mapped onto a 4x4 or 5x5 platform respectively. The results in Figure 5.8a and Figure 5.9a show how many tasks and flows were schedulable with GA-SCF. In Figure 5.8a, GA-SCF mapped the TB-4 task set with less than 80% schedulability and the TB-6 task set with less than 35% at the nominal frequency. For a hard real-time system, achieving 100% schedulable tasks and flows is essential to ensure that the predictability of the system is maintained, but in this case GA-SCF's task mappings are considered infeasible for the system. On the other hand, GA-BFF provides a better way of improving the task mapping optimisation process through the minimisation of the breakdown frequency. If the breakdown frequency can be found, GA-BFF will be able to maintain the schedulability of the task mapping at the same time as gradually reducing the frequency as the optimisation progresses to the last generation. The plot depicted in Figure 5.8b shows how GA-BFF converged; although above the nominal level, its task mapping at each point in time is feasible for the system.

It should be noted that a 5x5 platform provides a larger number of processing elements than a 4x4 platform, reducing the number of shares in a single processor and at the same time increasing the probability of a low-priority task evading interference. As the result, GA-SCF successfully converged to 100% schedulability for the TB-4 test bench as depicted in Figure 5.9a. With a larger platform, the mapping was less difficult for GA-SCF, but its task mappings were only usable at the nominal frequency. In fact, GA-SCF failed to find any feasible task mapping for TB-6. GA-BFF addressed this optimisation problem by finding the breakdown frequency to improve the task mappings and make them feasible for the system. As shown in Figure 5.9b, the algorithm converged below the nominal level for TB-4 and slightly above the level for TB-6.

GA-BFF's performance is consistent when mapping a group of test benches (TB-7, TB-8 and TB-9) with high utilisation onto a 10x10 platform. The results depicted in Figure 5.10a show that GA-SCF failed to find any schedulable task mapping for the system, even on a platform with a greater number of processing elements than on 4x4 and 5x5 platforms. Conversely, GA-BFF showed better performance, outperforming GA-SCF, as depicted in Figure 5.10b. Based on the



(a) GA-SCF



(b) GA-BFF

Figure 5.10: Based on synthetic test benches and a 10x10 platform; (a) GA-SCF schedulability convergence, (b) GA-BFF breakdown frequency convergence

breakdown frequency, the system can become schedulable with the given task mapping, even though the frequency is above the nominal level. The results presented in this section suggest that GA-BFF is better in finding schedulable task mappings than GA-SCF for the given test benches, and this validates the experimental hypothesis stated earlier in this chapter.

5.5 Summary

Infeasible task mappings from an optimisation process can be improved if the breakdown frequency that makes all tasks and messages schedulable is found. Scaling the frequency with the intention of finding the frequency, however, is not a simple matter due to the fact that different task mappings introduce various interference patterns, affecting the response time of each task and message. Without an appropriate frequency-scaling technique suitable for hard real-time task mapping optimisation, merely lowering the frequency will only increase the execution time of tasks and the latency of messages. This may delay the end-to-end response time and consequently shift the system into becoming unschedulable. Conversely, increasing the frequency could easily make the system schedulable, but only, at the expense of higher energy dissipation. The minimal frequency at which the system could become schedulable is essential for the system to avoid unnecessary any increment of the operating frequency. For this purpose, a new fitness function based on the notion of breakdown frequency is proposed to address the task mapping optimisation problem. The breakdown frequency can be used as a new metric for hard real-time task mapping optimisation to search for the schedulable task mappings. If the breakdown frequency can be found for a given task mapping, the system will definitely be schedulable at the early stage of optimisation. This will save exploration time and help early design decisions, especially if the optimisation algorithm successfully converges below the nominal frequency at the early stage of optimisation. Conversely, if a feasible task mapping cannot be found at the nominal frequency, the task mapping can be improved by applying the breakdown frequency to make all tasks and messages schedulable. The minimal frequency is further reduced as the optimisation progresses, providing a potential for reducing power consumption. Although task mapping optimisation could reap the benefits offered by the new approach, the process of finding the breakdown frequency is still an iterative operation. Every time a new frequency is applied, all tasks and messages need to be evaluated to determine their schedulability, and this increases the run-time of the optimisation.

Chapter 6

Constructive Algorithm for Mapping Hard Real-Time Tasks

From the results presented in the previous chapters, GA-based optimisation algorithms have been proven to be efficient techniques for finding a schedulable task mapping for a NoC-based hard real-time system. The algorithm's adaptive feature enables the simultaneous configuration of multiple design parameters and the integration of several fitness functions for addressing multi-objective optimisation problems. Devising different exploration strategies is an effective way of addressing the optimisation problems in the designs of such systems. However, GA's characteristic which gives the advantage of exploring a large design space depends on the size of the population. A large population provides better diversity among individuals. It helps the search for a schedulable task mapping to become more effective, but in return increases the number of evaluations that need to be performed during optimisation. As a result, the algorithm needs significant time to find the schedulable task mapping. In this chapter, a new constructive mapping algorithm is presented to find the task mapping. Unlike GA which depends on a large number of individuals to search for the task mapping, it constructs a task mapping based on specific design properties. This approach avoids evaluating many solutions in a single run, and thus reduces the amount of time needed to find the task mapping.

This chapter is organised as follows: section 6.1 explains the motivation of the proposed approach and section 6.2 describes in detail how the constructive algorithm performs task mapping. Section 6.3 discusses some of the results from the experimental work. Finally, section 6.4 summarises the proposed approach.

6.1 Motivation

In the previous chapters (Chapters 3, 4 and 5) GA-based optimisation algorithms have been used to address the single and multi-objective optimisation problems in finding a fully schedulable task mapping for the hard real-time embedded system based on NoC. The algorithms rely on the end-to-end schedulability analysis to evaluate the schedulability of all tasks and messages in the system. The results depicted in Figure 4.6a are one example of how the algorithms found fully schedulable task mappings with low power dissipation from the design space of the system.

The evolution of individual genetics has been the underlying notion that supports the implementation of the evolution process in the algorithms; enabling the exploration of task mapping in a similar way to the evolution of a population. In order to enable exploration in a wide area of the design space, the algorithms rely on the diversity of individuals in the population. A large population increases the diversity, but at the expense of long optimisation run-time since the algorithm has to evaluate a large number of individuals. Conversely, the population becomes less diverse with a decreased population size and the tendency of converging to the unschedulable task mapping is high.

To support the evaluation of a large number of solutions, a fitness function that requires less computation time to yield a single fitness value is desirable. Without this kind of fitness function, the optimisation run-time of the algorithm will easily escalate. However, creating a fitness function that requires less time complexity is not a simple matter as the sizes of task and message set also affect the evaluation time. For example, to validate the system as deemed to be schedulable requires that every task and message is explicitly analysed by the function and thus the number of iterations increases with the number of tasks and messages. The time complexity of the function is exacerbated by the amount of direct and indirect interference that must also be calculated to yield the worst-case end-to-end response time. If the same analysis is applied together with other equations to calculate a different metric, it will further increase the evaluation time of the fitness function. Figure 5.6 shows how much difference there was in the times of two different fitness functions which depended on the same schedulability analysis when they were used within the same optimisation algorithm.

6.2 Constructive Task Mapping Algorithm

This section introduces the Constructive Mapping Algorithm (CoA) by giving an insight on how it constructs a task mapping for the NoC-based hard real-time system. Based on the listing shown in Figure 6.1, the flow of the algorithm can be divided in three main steps as follows:

- **First step** : mapping a set of tasks to containers
- **Second step** : mapping containers onto NoC platform
- **Third step** : evaluate schedulability of all tasks and messages

Before introducing all the steps, it is worth explaining what a container means in this section. We assume a container as a virtual processor and has a total size equivalent to a fully utilised processor. It can contain one or more items depending on the size of the items. A task consumes a portion of processor computation time, similar as an item filling a space of a container. Therefore, stacking items into containers in a sense analogous to mapping tasks onto processors. By representing processors as containers gives a degree of freedom for task allocation and also for rearranging containers after the allocation. For example, based on the processor utilisation, the current task with the highest utilisation level is allocated to the container with the least utilisation level. Then, all the containers are rearranged in ascending order to determine the new container with the least utilisation level. In addition, by doing this also allows the analysis of schedulability for all tasks prior of mapping the containers onto the NoC platform.

In the first step, the tasks of a given task set are mapped onto a set of containers. A container is selected for a task based on the container utilisation

```
1: procedure SchedulabilityTest($Application\ application, Platform\ platform$)
2:     $Integer\ Ut = 0$
3:     $Integer\ Uf = 0$
4:     for each $task_i$ in $application$ do
5:         if $PassSchedulabilityTest(task_i,\ platform) != true$ then $Ut = Ut + 1$
6:         end if
7:     end for
8:     for each $msg_i$ in $application$ do
9:         if $PassSchedulabilityTest(msg_i,\ platform) != true$ then $Uf = Uf + 1$
10:        end if
11:    end for
12:    return $Ut + Uf$
13: end procedure
14: procedure MappingToContainer($Application\ application, Platform\ platform$)
15:    $ArrayList\ taskList = CreateTaskList(application)$
16:    $ArrayList\ containerList = CreateContainer(platform)$
17:    $SortTaskUsingUtilisation(taskList)$
18:    for each $task_i$ in $taskList$ do
19:        $AllocateLeastUtilisedContainer(task_i,\ containerList)$
20:        $SortContainer(containerList)$
21:    end for
22:    return $containerList$
23: end procedure
24: procedure MappingToPlatform($ArrayList\ containerList, Platform\ platform$)
25:    $SortContainerUsingOutgoingMessages(containerList)$
26:    for each $container_i$ in $containerList$ do
27:        $AllocateLongestEuclideanDistanceNode(container_i,\ platform)$
28:    end for
29:    return $platform$
30: end procedure
```

Figure 6.1: Constructive mapping algorithm pseudo code

and the schedulability of all the tasks, including the one that will be mapped and the existing tasks in the container. In order to do this, the utilisation of each task must be calculated before mapping. If the task interconnects with other tasks, the utilisation of message on a link is required. Once the two values available for each task and its corresponding message, the summation of both values yield the total utilisation of the task. Based on this attribute, the task set can be sorted in descending order, with the top in the set is a task with the highest total utilisation. Our intention is to select a task from the highest to the lowest utilisation. This way a task that has a high computation and communication volume is allocated first to a container. If tasks with low utilisation are allocated first, then the containers will have less space by the time the turn of tasks with high utilisation arrives.

A selected task from the ordered set is allocated to a container with the least utilisation. One of the reasons is to give it a high chance of allocation to the container, that is, if the utilisation of the selected container is already high then the container probably cannot accommodate the task, especially if the task has high utilisation. Another reason why tasks are allocated this way is to avoid so many tasks allocated to one container, otherwise, it is likely that tasks which are already in the container will experience high interference. For example, a container may have high utilisation because it is packed with a lot of tasks and these tasks, depending on what priority levels they have, will receive interference from high priority tasks. On the other hand, a container might be full because it contains a task with high utilisation, but by first allocating tasks with high utilisation the interference in the container could be lessen. However, having a container with only a task is less efficient, considering it can accommodate more tasks if it has enough space. Since interference will affect the schedulability of every task in a container, their schedulability is tested along with the given task as if they have been all allocated to the container.

In the second step, tasks in all the containers are mapped onto the platform. All the tasks from the same container are mapped onto the same core. The allocation of container on the platform depends on another attribute known as the Euclidean distance: the distance between two cores in the NoC platform. Before any allocation is performed by the latter process, the containers are sorted according to their number of outgoing messages. At this stage, whether a task will need to send a message can be determined, that is, if two interconnected tasks are mapped into the same container, messages will not be sent between the two tasks over the network. Based on the number of messages, containers are sorted in descending order; the container with the highest number of messages becomes the top in the list of containers. The purpose of sorting in this order is to allow the mapping of containers, which sends high number of messages, away from each other. Starting with the container at the top of the list, this process maps the tasks of the container onto the farthest core in the platform from the recent mapped core, consecutively until the last container. The benefit of mapping the containers this way is to avoid tasks that are mapped onto different cores but are not communicating with each other from being closely mapped in the same area, the idea is to lessen the interference in the shared network resources. This kind of mapping is targeted for cores with less or no communication between them, but need to send messages through the network resources. In future, we will improve the mapping of cores with high communication volume between them.

In the last step, the schedulability of all tasks and messages are evaluated to determine the fitness of the task mapping. At this research stage, the purpose of CoA is to construct a task mapping in a single run based on the properties explained in the following sections. Several repetition of the constructive mapping process by using different ways of mapping the tasks, if a schedulable task mapping cannot be found, is possible. However, this require integration of other rules and properties with the flow of the algorithm to create different kind of mappings. In future work, this kind of techniques will be considered.

6.2.1 *Main* Function

This section explains the entry point of the constructive mapping algorithm, as shown in Figure 6.2.

The main function receives two inputs: an application object consisting a set of tasks and a platform object consisting a NoC platform. In the function, a task mapping is constructed in two steps and then tested to determine its schedulability.

```
1: procedure MAIN($Application\ app, Platform\ plat$)
2:     $ArrayList\ containerList = MappingToContainer(app,\ plat)$
3:     $Platform\ mappedPlatform = MappingToPlatform(containerList,\ plat)$
4:     $Integer\ totalUnschedulable = SchedulabilityTest(app,\ mappedPlatform)$
5: end procedure
```

Figure 6.2: Constructive mapping algorithm example

The *MappingToContainer* function (line 2, Figure 6.2) sorts the task set according to the utilisation order before each task is pushed into a container. A list of containers is created to represent all the processing nodes in the platform and thus the number of containers in the set is equivalent to the number of processing nodes. The notion behind the first step is finding an allocation of tasks in the containers, starting from a task with the highest utilisation down to that with the lowest utilisation, without exceeding the specified utilisation of each container. After all tasks have been allocated to containers, the function returns the container list.

The *MappingToPlatform* function (line 3, Figure 6.2) receives the container list returned by function *MappingToContainer* and performs the second step by first sorting the container list in ascending order according to the volume of outgoing messages sent by the tasks in every container. Second, it maps all tasks in every container onto the processing nodes of the NoC platform starting from the container with the highest number of outgoing messages to the farthest node from the latest mapped node.

Once all tasks have been mapped onto the platform, the *SchedulabilityTest* function (line 4, Figure 6.2) will analyse all tasks and messages of the application to determine their schedulability. This function calculates the number of unschedulable tasks and messages as the fitness of the task mapping.

In the following sections, the details of each function are explained in more detail.

6.2.2 *SchedulabilityTest* Function

The *SchedulabilityTest* function (line 1, Figure 6.1) receives an application and a platform objects as inputs and evaluates the schedulability of every task and message. After evaluating all tasks and messages in the application, the number of unschedulable tasks and messages is calculated and returned as the function's output. This return value is the result that helps to determine how schedulable the system is with the task mapping.

In the *PassSchedulabilityTest* procedure, the worst-case response time of task i is calculated using equation 3.1 and then, with equation 3.5, its schedulability is analysed. If the task is unschedulable, the procedure returns false and the Ut variable is increased by one. Similar to the evaluation of tasks, every message is analysed by calculating its worst-case latency based on equation 3.4 and then comparing this with its deadline using equation 3.6. If the message is unschedulable, the Uf variable is increased by one. Then, the cumulative value of all unschedulable tasks and flows is given by the summation of the variables and returned as the function's output (line 12).

6.2.3 *MappingToContainer* Function

The *MappingToContainer* function determines the allocation of tasks in a set of containers based on task utilisation. For this purpose, a task list and a container list must be created respectively from the application and platform objects which it receives as inputs. The task list contains all the tasks extracted from the application object (line 15, Figure 6.1). Based on the number of processing cores in the platform object, the same number of containers is created at line 16.

At line 17, the task list is sorted by the heuristic sorting procedure called *SortTaskUsingUtilisation*. This procedure sorts the task list according to task and message utilisation. The utilisation of task i is computed using equation 6.1, where c_i and t_i are the worst-case execution time and the period of task i respectively. At this stage, the full routes of messages are yet to be determined. It should be noted that the routes of all messages can only be determined after all tasks have been mapped on the platform, and therefore the utilisation of each message on all links (or complete path) cannot be calculated by this function. Instead, the utilisation of each message is computed as a single-hop utilisation based on equation 6.2. Given the size of packet i as $PacketSize_i$ and its period T_i, the utilisation of message i is given by equation 6.2. Then, the summation of both utilisations can be calculated with equation 6.3, which is the property used by the sorting algorithm to sort all tasks in the list. The list is sorted in descending order. At the top of the list is the task with the largest utilisation followed by the next task with the second largest utilisation and so on until the last task with the least utilisation comes at the end of the list.

$$Util\_t_i = \frac{c_i}{t_i} \tag{6.1}$$

$$Util\_f_i = \frac{PacketSize_i}{T_i} \tag{6.2}$$

$$Util\_Total_i = Util\_t_i + Util\_f_i \tag{6.3}$$

$$Util\_Total_i + Util\_Container_j \leq Util\_Max \tag{6.4}$$

After all tasks have been sorted in the list, a task is selected from the top of it to become a candidate for one of the containers (line 19). This operation is performed by the procedure *AllocateLeastUtilisedContainer*, which receives the task and the list of containers as inputs. All the containers are checked according to the conditions for every task that is given as input and a container that meets the conditions will become the container for the task. The *first condition*, as shown by equation 6.4, is true if the total utilisation of the container and the task does not exceed the maximum utilisation of the container. Any container that does not meet this condition is excluded from the selection.

Following the first condition, the procedure checks the remaining containers according to the second condition. The *second condition* states that the least utilised container with the highest schedulability percentage will be selected as the container of the given task. In other words, the second condition refers to the schedulability percentage first before deciding which container has the least utilisation among them. For example, in Figure 6.3, container $C2$ is selected for task $T1$ rather than container $C1$, even though the latter container has the lowest utilisation. In order to follow the second condition, each of the remaining containers is evaluated by analysing the schedulability of all the existing tasks in the container and the task that is going to be pushed into it. It should be noted that the maximum utilisation of each container can be defined as 100% (full utilisation) or any reasonable percentages, for example 80% or 90%. Once all tasks have been allocated to the containers, the container list is returned as the output of the *MappingToContainer* function.

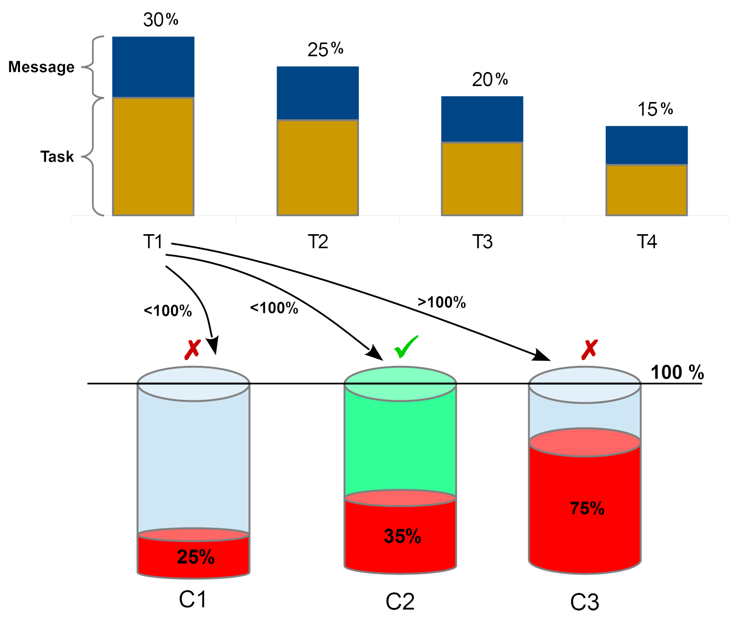


Figure 6.3: Mapping tasks to containers based on task utilisation and the schedulability inside the containers

6.2.4 *MappingToPlatform* Function

Given a list of packed containers and the NoC platform as inputs, the function *MappingToPlatform* allocates the tasks in the containers onto the processing nodes of the platform. First, at line 25 Figure 6.1, the procedure *SortContainerUsingOutgoingMessages* sorts the container list according to the number of outgoing messages from each container. The container with the highest number of outgoing messages leads the container list, followed by the second container and so on until the last container with the lowest number of outgoing messages.

In the next step, once the sorting has been completed, tasks from the first container in the list are mapped onto the first processing node in the platform. Unlike the first container, the mapping of the rest in the list is determined based on the Euclidean distance from the recent mapped node. Given the location of a source node (x_s, y_s) and a destination node (x_d, y_d), the Euclidean distance between the two nodes can be calculated using function 6.5. A processing node is restricted to one container only; once a container has been mapped it will be excluded from the list to prevent it from being reused.

$$Euclidean\_distance_{s,d} = \sqrt{(x_d - x_s)^2 + (y_d - y_s)^2} \tag{6.5}$$

The mapping step is performed by the procedure *AllocateLongestEuclidean-DistanceNode* at line 27. For every container except the first one in the container list, the function calculates the Euclidean distance from the recent mapped node to another node which is still available in the platform, as shown in Figure 6.4. From the figure, the Euclidean distance between the recent mapped node (1, 1) and the available node (3, 3) is 2.82. Given a container, the Euclidean distances to all available nodes are calculated from the recent mapped node, excluding any node that has already been allocated to a container.

A set of free nodes with varying Euclidean distances may be available for a container at the same time. It is the aim of the algorithm to disperse the interference in NoC by mapping containers with the highest number of outgoing messages far away from each other. Therefore, a node with the longest Euclidean distance will be selected as the node where the tasks from the container will be mapped. If more than one free nodes with the same Euclidean distance exist at the same time, the function selects the first node that it finds with the longest distance. The function returns once all tasks from the containers have been mapped onto the platform.

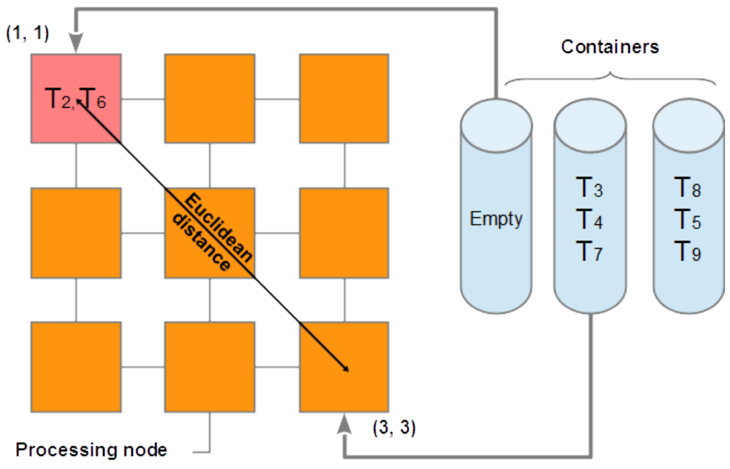


Figure 6.4: Mapping tasks from containers to NoC platform based on the number of outgoing messages and Euclidean distance between two nodes

6.3 Evaluation

6.3.1 Test benches and Baselines

Several sets of synthetic test benches and three different platform sizes, 4x4, 5x5 and 10x10, were used in this study. For every platform, a set of test benches was generated in the same way as explained in section 3.8.1. Some of the synthetic test benches in the sets were similar with the synthetic test benches used in section 3.8.1, but each set contained a greater number of test benches than the test bench sets used in that section because the experimental work in this chapter required more utilisation range to prove the hypothesis. The last test bench in every set contained the maximum utilisation equivalent to the maximum utilisation of all cores in the platform. For example, on a 4x4 platform, the maximum utilisation of all cores was 1600%. In addition to the synthetic test benches, a test bench known as AVA (see Appendix A), similar as the realistic test bench explained in section 3.8.1 was again used in this study.

For this study, a single-objective GA and a Random Mapper (RN) were used as baselines. The multi-objective GA such as MOGA was not selected because our aim was to study how the CoA can produce a schedulable task mapping (because it was designed for schedulability and not power dissipation) and the suitable baseline for comparison is the single-objective GA. The GA used in this study is the same baseline algorithm as what we used in section 3.8.2 and the RN is the same algorithm explained in section 4.7.1. GA is classified as a meta-heuristic and thus the general time complexity that applies to all the genetic algorithms is difficult to find. Instead, it is common to use the convergence rate to measure how fast it finds the solution. For example, in the results section of the previous chapters, the number of generations is used to show the algorithm's convergence to a fully schedulable task mapping. Conversely, CoA is a deterministic algorithm which iterates once in every execution, thus it is different from GA which performs in several iterations (or generations). In order to compare the run-time performance of both algorithms in performing task mapping, the cumulative time of finding the best task mapping was calculated. For GA, it is the total time to find the first schedulable task mapping or, if the mapping cannot be found, it is the time taken to perform every generation. Since CoA only found a task mapping in each execution, it is the time taken to produce the task mapping itself.

6.3.2 Results

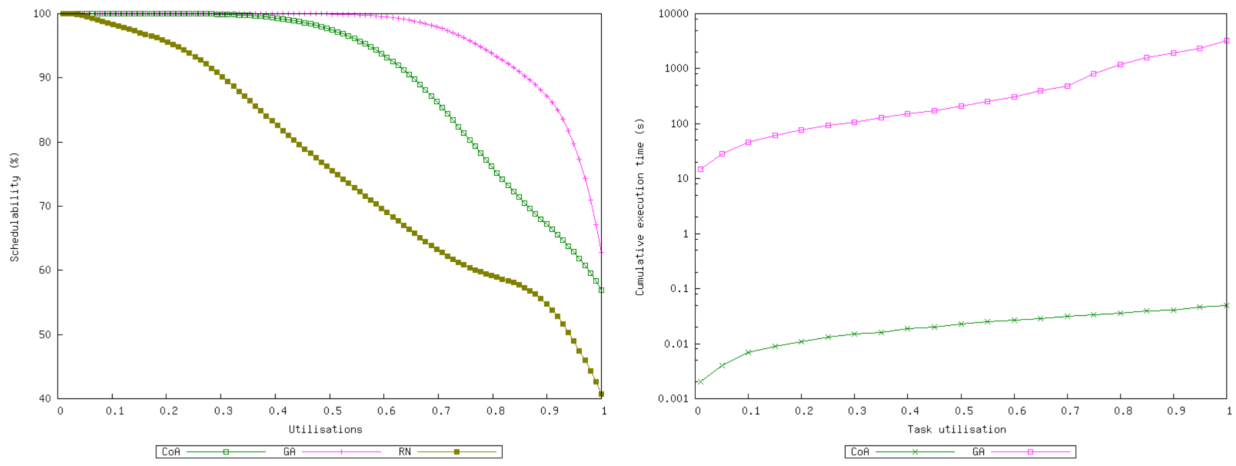
CoA is proposed to serve as an alternative heuristic for producing a schedulable task mapping but without requiring a population of individuals as GA does to explore task mappings. As mentioned in the motivation section of this chapter, the optimisation run-time can be affected by the number of evaluations performed by GA, depending on how many individuals are contained in the population. The problem is further exacerbated if the fitness function which evaluates every individual consumes a large amount of time to yield a single fitness value, increasing the total time taken to optimise a single parameter.

It should be noted that the natures of the two algorithms are different, influencing the way of finding a schedulable task mapping. For example, GA finds the task mapping by evolving many individuals in the population over generations: an ability that helps to simultaneously explore many alternative solutions. Unlike GA, CoA is less dependent on many solutions, instead it constructs a task mapping based on the specific properties of a task set, reducing the exploration effort and thus offering the potential to apply a computation-intensive fitness function.

As an optimisation algorithm, it has been proven that GA is a good option for addressing the design space exploration problems and finding the task mappings effectively. At the same time, CoA could become an alternative approach if the search effort depends on computation-intensive fitness functions that require more time to evaluate, which is a hurdle for optimisation techniques based on GA which depend on many individuals. This experiment was conducted to study the performance of CoA: if it is not similar, to what extent can it produce a schedulable task mapping compared with GA.

Figure 6.5, 6.6 and 6.7 shows the schedulability percentages achieved by CoA and the baselines. The figures also display the cumulative execution time taken by CoA for creating a task mapping and by GA for finding the first schedulable task mapping. If a schedulable task mapping is not found until the last generation, the cumulative time of GA includes the total time to complete all the generations. It is worth noting that CoA is our early version of the constructive mapping algorithm. It builds a task mapping based on certain properties in a single run, unlike GA that has established and can explore many task mappings, the way of the task mapping can be changed by CoA is limited. With the early version of CoA we expected that achieving the same level of performance as GA is difficult. Therefore, we aimed for a task mapping with better schedulability than a random task mapping, but the difference in performance as compared with GA must not less than 50%. The main advantage of CoA is its fast execution time because it does not need to evaluate many task mappings in a single run. If the difference between the execution time of CoA and the GA is significant, it could be implied that CoA has a margin for improvement in the next version (for example, by adding evaluation functions to improve mapping).

Figure 6.5a displays the mapping results on a 4x4 platform. At each utilisation level, the graph shows the schedulability percentage of the system based on the task mapping produced by CoA and the baselines (RN and GA). For GA, the selected task mapping displayed on the graph was the first schedulable task mapping found during optimisation, whereas CoA and RN only produced a single task mapping each in every execution. The level of schedulability of CoA started to drop below 100% at 35% utilisation, whereas GA recorded a drop at 60% util-



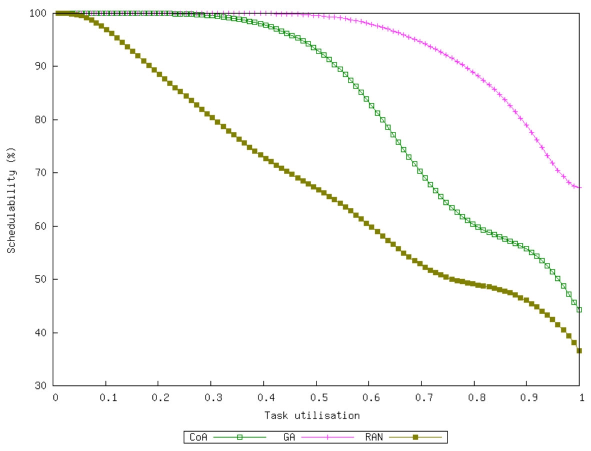
(a) Schedulability (b) Cumulative time

Figure 6.5: Based on a 4x4 platform; (a) schedulability convergence of CoA and the baselines, (b) the cumulative time to perform task mapping

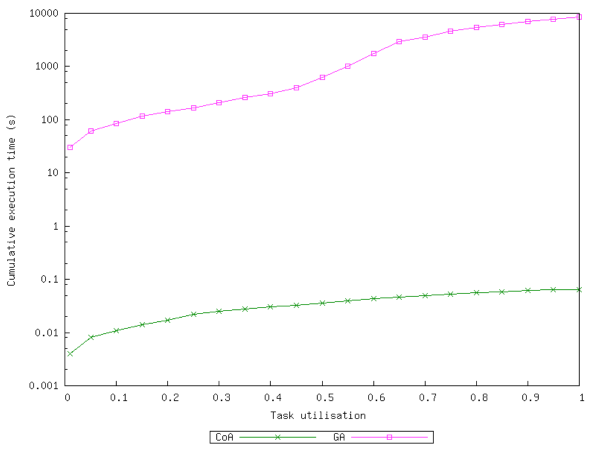
isation. Although CoA performed less than GA when the level of utilisation was above 35% utilisation, at levels below 35% utilisation both algorithms produced similar performance, that is, a fully schedulable task mapping was found. The value of 50% from the recorded GA performance (60% utilisation) is 30%, and hence at 35% utilisation CoA has achieved the target we set earlier. In addition, Figure 6.5b shows that CoA took less execution time than GA to produce a schedulable task mapping.

On a 5x5 platform, CoA showed a similar performance pattern to that on the 4x4 platform. As depicted in Figure 6.6a, the performance of CoA started to drop after reaching 30% utilisation compared with GA at 50% utilisation, but was better than the random mapper. At 30% utilisation, CoA has achieved the target we set earlier, more than half of 50% utilisation of GA. As shown in Figure 6.6b, CoA maintained the same performance as for the 4x4 platform by executing in less time than the GA.

Again, CoA showed a drop in schedulability at 20% utilisation on a 10x10 platform compared with GA at 30% utilisation. These results are depicted in Figure 6.7a, which also displays the worst performance by RN compared with CoA. Similar as in the 4x4 and 5x5 platforms, CoA achieved the target we set

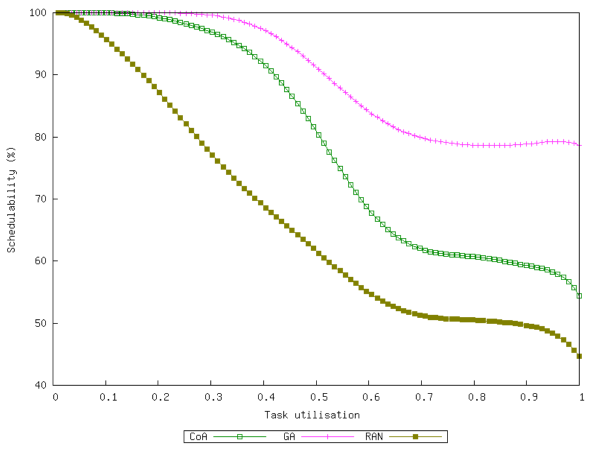


(a) Schedulability

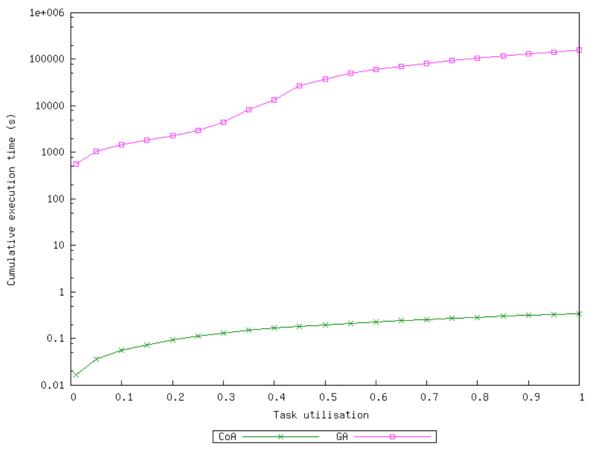


(b) Cumulative time

Figure 6.6: Based on a 5x5 platform; (a) schedulability convergence of CoA and the baselines, (b) the cumulative time to perform task mapping



(a) Schedulability

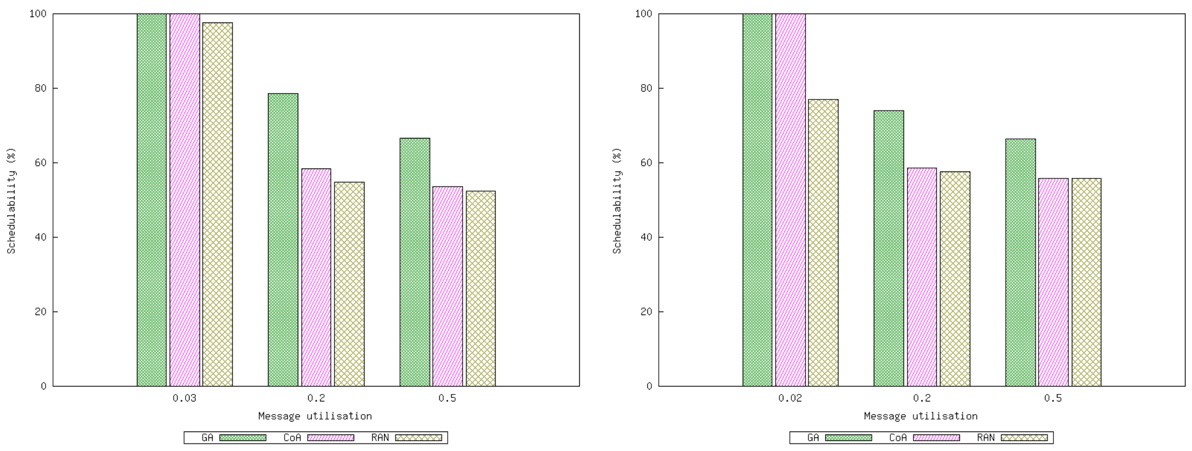


(b) Cumulative time

Figure 6.7: Based on a 10x10 platform; (a) schedulability convergence of CoA and the baselines, (b) the cumulative time to perform task mapping

earlier at 20% utilisation, which is more than 15% difference in utilisation (50% out of 30% utilisation). Figure 6.7b depicts the cumulative time of both CoA and GA, and it shows that CoA's cumulative execution time was lower than that of GA.

The previous results depicted in Figures 6.5a was based on the increase of task utilisation, but message utilisation was constant at 3% utilisation. Therefore, further investigation was conducted to study the impact of increased message utilisation. From 3%, message utilisation was gradually increased to 20% and 50% utilisation. For this investigation, task utilisation was fixed at 20% of a 4x4 platform. Figure 6.8a shows the results of mapping for CoA and the baselines (GA and random mapping). From the results, CoA and GA both found a schedulable task mapping at 3% message utilisation. With increased message utilisation at 20% and 50%, both algorithms failed to find any schedulable task mapping, although GA's task mapping had better schedulability than CoA's task mapping. In all the utilisation levels, RAN produced task mappings with lower schedulability than CoA.



(a) 4x4 platform

(b) 5x5 platform

Figure 6.8: Task mapping results on 4x4 and 5x5 platforms based on different levels of message utilisation

Figure 6.6a shows results of task mapping on the 5x5 platform with message utilisation was fixed at 2% utilisation. Starting with 2% message utilisation, Similar as the 4x4 platform, task utilisation was fixed at 20% and the message utilisation was gradually increased to 20% and 50% utilisation. The mapping results are shown in Figure 6.8b for CoA and the baselines. At 2% utilisation, both CoA and GA were able to find a schedulable task mapping. However, at higher utilisation levels (20% and 50%) neither algorithm found any schedulable task mapping, although GA produced task mappings with better schedulability than CoA. CoA outperformed RAN at all utilisation levels except at the 50% utilisation, where the performances of both algorithms were recorded as similar.

The graph depicted in Figure 6.7a shows the results of task mapping on a 10x10 platform, by gradually increasing task utilisation while message utilisation was constant at 1%. Similar as the 4x4 and 5x5 platforms, further investigation on the 10x10 platform was conducted to study the effects of increased message utilisation on task mapping. From 1%, message utilisation was increased to 15% and 50%. For the 10x10 platform, the amount of task utilisation was fixed at 15% utilisation. Figure 6.9 shows the results of mapping on the 10x10 platform. CoA and GA each produced a schedulable task mapping at 1% utilisation. However, at the 15% and 50% utilisations, both failed to find any schedulable task mapping, although GA produced a task mapping with higher schedulability than CoA. CoA outperformed RAN, which failed to find any schedulable task mapping at all utilisation levels.

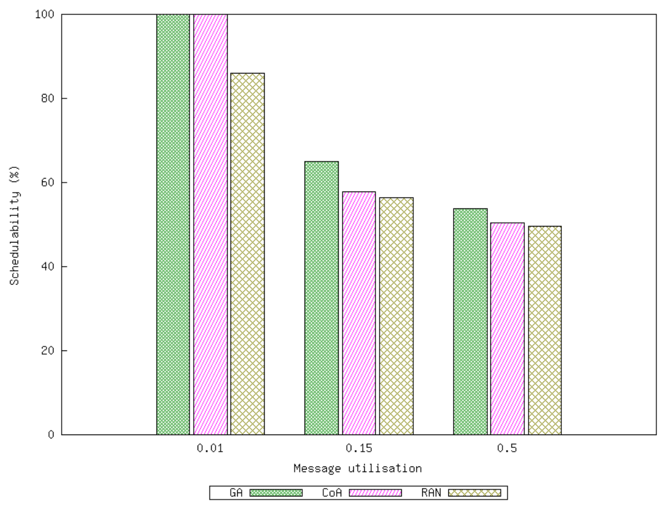


Figure 6.9: Task mapping results on a 10x10 platform based on different levels of message utilisation

Table 6.1: Selected percentages of utilisation for studying the effects of varying message utilisation on task mapping

Platform	Task utilisation[9]	Message utilisation[10]
4×4	20%	3% 20% 50%
5×5	20%	2% 20% 50%
10×10	15%	1% 15% 50%

The study of the effects on task mapping based on the increases in the message utilisation was limited to certain percentages due to the time constraint of this study. Constant task utilisation at 20% for each of the 4x4 and 5x5 platforms and 15% for the 10x10 platform were selected based on the results depicted in Figure 6.5a, 6.6a and 6.7a. From the figures, the selected percentages were the point where CoA could find a schedulable task mapping right before the schedulability started to drop. The study began from the corresponding levels of message utilisation (3% for the 4x4 platform, 2% for the 5x5 platform and 1% for the 10x10 platform) of the selected task utilisation, followed by the intermediate (15% and 20%) and the high (50%) percentages of message utilisation. The intermediate message utilisation (20% for the 4x4 and 5x5 platforms, and 15% for the 10x10 platform) were for studying the effects on task mapping when tasks and messages had equal utilisation. Table 6.1 displays the summary of the selected task and message utilisation for the study.

In addition to the synthetic test benches, the AVA test bench as used in the previous experiment was used again to study the performance of CoA in mapping a real application. CoA showed equal performance to GA, since both algorithms found a fully schedulable task mapping. This is clearly depicted in Figure 6.10, which also shows that CoA outperformed RN; the latter algorithm only achieved

[9]Task utilisation were selected from results depicted in Figure 6.5a, 6.6a and 6.7a

[10]3%, 2% and 1% were the corresponding message utilisation of the selected task utilisation for the 4x4, 5x5 and 10x10 platforms respectively

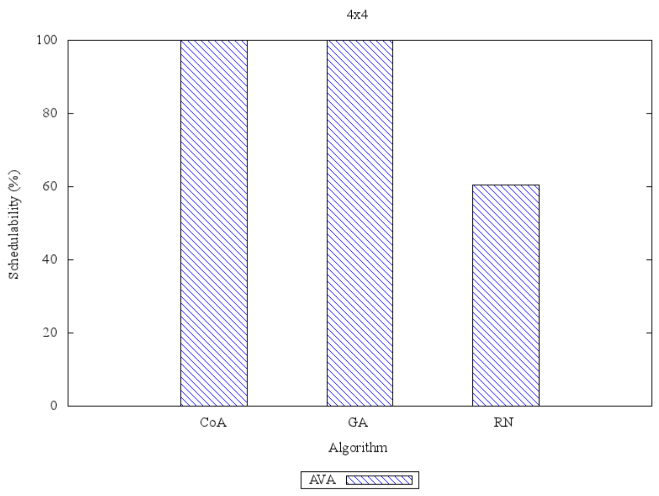


Figure 6.10: Schedulability of AVA on a 4x4 platform with CoA, GA and RN task mappings

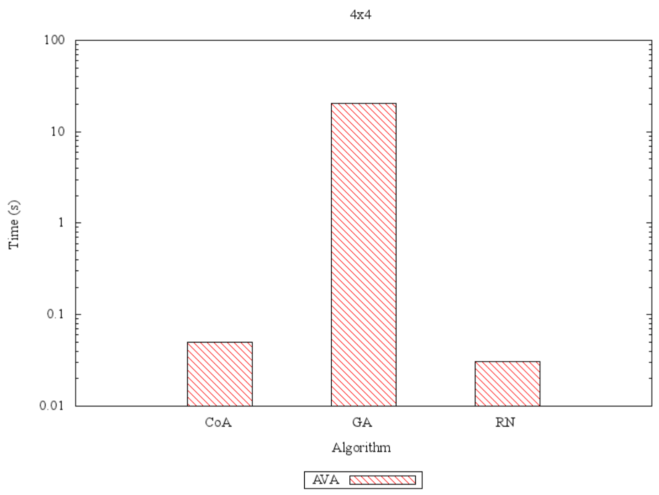


Figure 6.11: Time taken by CoA, GA and RN to find a schedulable task mapping

60% schedulable tasks and messages in the system. Although both CoA and GA exhibited the same performance, the former algorithm took less time to perform task mapping compared with GA. As depicted in Figure 6.11, the difference in time between the two algorithms is significant, which shows how much time can be saved to achieve the same result.

6.4 Summary

CoA is proposed for creating a schedulable task mapping without requiring an extensive amount of evaluation effort. Its approach is different from that of GA, which depends on a group of individuals to explore and improve task mappings. With CoA, a task mapping is constructed rather than explored, which allows it to avoid evaluating so many solutions. Based on specific attributes such as the utilisation of a task set, task schedulability and the number of outgoing messages, finding a schedulable task mapping without consuming a lot of computation time is possible. In particular cases, finding the schedulable task mapping with CoA is difficult, such as when the utilisation of a task set is high. Although its performance in this respect was less than what has been achieved by well-established GA, we imply that it has the potential for further improvement in the future. This is based on our finding from the mapping of AVA test bench. CoA showed that it could find a schedulable task mapping as the GA, but in less time than what was taken by the latter algorithm. Further evaluation with the synthetic test benches revealed its true potential by successfully achieving our target for the early version. Since CoA did not consume a lot of computation time, it could be extended with additional evaluation functions to improve the quality of task mapping that it produces.

Chapter 7

Conclusions

NoC is seen as a reliable communication infrastructure for hard real-time embedded systems. A hard real-time embedded system based on NoC has stringent timing constraints that must be met in any scenario, otherwise the system cannot be guaranteed to be schedulable. Task mapping determines how tasks are mapped on the NoC platform and could affect the schedulability of the system. Exploring task mappings which keep tasks and messages schedulable is challenging because other parameters have influences on the schedulability. In addition, finding schedulable task mappings must also consider other constraints that are crucial to the system performance. In this thesis, the task mapping optimisation problems have been successfully addressed based on different approaches. This chapter concludes the research findings and provides some suggestions for possible future works.

7.1 Review of Research Findings

In Chapter 1 the following proposition was stated:

> *A schedulable task mapping can be found for NoC-based hard real-time embedded system*

This proposition has been proven by a series of experimental results developed through Chapters 3, 4, 5 and 6.

Feasible task mappings for NoC-based hard real-time embedded systems are hard to find if low-priority tasks and messages receive high interference which leads to them becoming unschedulable. In a system with a fixed priority pre-emptive policy, low-priority tasks and messages are pre-empted to make sure that their high-priority counterparts have guaranteed access to shared resources. This scheduling policy is essential to ensure the predictable behaviour of the system, but it imposes interference on low-priority tasks and messages. The hard real-time requirements dictate that the system is deemed schedulable only when all tasks and messages meet their timing constraints in any scenario. In order to meet the requirements, the interference imposed on low-priority tasks and messages should be reduced, but without sacrificing the level of services received by their high-priority counterparts. To address this problem, the simultaneous configuration approach has been proposed in Chapter 3 based on the notion that changing the priority assignment could lessen the interference suffered by the low-priority tasks and messages of a given task mapping. With the approach, task mapping and priority assignment are simultaneously configured during the optimisation process. From the results of the experiments carried out, it has been proved that the approach addresses the problem effectively and offers other advantages as well. Briefly, it produced feasible configurations which enabled the system to meet the timing constraints of all tasks and messages, as opposed to the baselines which failed to achieve the same result using similar test benches. In addition, it improved the convergence of the optimisation algorithm to become faster than the baseline heuristics.

In Chapter 4, a multi-objective optimisation technique was proposed to address the problem of conflicting objectives in NoC-based hard real-time embedded system designs. The power dissipation problem affects the design of such systems. For example, mapping tasks closer to each other might reduce the energy footprint in the network, but it will also be likely to increase the interference between tasks, delay their response time and lead to the system becoming unschedulable. These constraints are made as a list of objectives that must be achieved in designs. Focusing on one design objective but ignoring the others hides their impacts from system design, whereas simply aggregating both objectives into one objective imposes bias on the solutions. The conflict between these objectives made them challenging to address through single objective optimisation techniques. In the multi-objective optimisation technique, the Pareto-optimal concept underlying the optimisation process provided an effective means of addressing the conflicting objectives by considering the trade-off between them. With the multi-objective optimisation algorithm, task mappings were improved better than the task mappings of the single objective optimisation algorithm. The results suggest that the proposed multi-objective optimisation technique is more effective than the single-objective optimisation technique at finding fully schedulable task mappings with low power dissipation.

In some cases, a task mapping that makes the system schedulable is hard to find. Improving the task mapping is challenging because of the limited information provided by the schedulability metric. As a result, this limits the potential of the optimisation algorithm to find alternative task mappings that might be schedulable. Based on the idea of increasing the frequency to speed-up the computation of tasks and the communication of messages, a fitness function was proposed in Chapter 5 to calculate a new metric called the breakdown frequency, which can be applied as a property of a task mapping. The breakdown frequency, if it exists, is the minimal frequency at which all tasks and messages in the system can become schedulable. Finding that frequency, however, is not a simple matter. For example, increasing the frequency will make the system easily schedulable, but it may not be the minimal frequency for the system. Instead, the proposed fitness function finds the minimal frequency by analysing the schedulability of all tasks and messages for every frequency value selected for the task mapping. The effectiveness of this approach helped the optimisation algorithm to improve the quality of the task mappings. As the optimisation progressed over time, the breakdown frequency gradually decreased, leading to a lower frequency below the nominal level. The only drawback of this approach is its immense evaluation time due to necessary evaluation of the schedulability of all tasks and messages for every selected frequency value.

The optimisation techniques proposed in Chapters 3, 4 and 5 depend on a GA to perform the task mapping optimisation. Its effectiveness in addressing the optimisation problems comes from manipulating a group of individuals in a population. Each individual represents a task mapping and its fitness must be evaluated by one or more fitness functions. The number of evaluations performed by the algorithm depends on the population size and a large size has a negative implication on the optimisation run-time. In order to avoid evaluating many solutions in a single optimisation run, a constructive algorithm was proposed in Chapter 6 to construct one task mapping rather than explore many task mappings at the same time. This provided a fast means of finding a schedulable task mapping. From the results based on synthetic test benches, although they were less than the achievement made by GA, the constructive mapping algorithm was able to find the schedulable task mapping up to specific utilisation levels. For the realistic test bench, the algorithm was able to find a schedulable task mapping as effectively as GA could. Based on these results, CoA has the potential to find the schedulable task mapping in less than the time taken by GA.

7.2 Future Works

The optimisation problems discussed in this thesis have been effectively addressed with the proposed DSE approaches, but some issues related to this subject are still open for future study.

Simultaneous configuration of task mapping and priority assignment improves the schedulability of the system and also the DSE process. However, the influence on the system performance is not limited to these parameters only; other parameters such as routing may have similar impacts on the schedulability of tasks and messages. In the system model, static XY routing was used to route messages. However, routing selection has a potential to reduce the interference in the network by redirecting high-priority messages away from low-priority messages. If this parameter can be manipulated accordingly, it might improve the schedulability of the system.

The breakdown frequency fitness function provides a new metric that can be applied as the fitness value of a given task mapping. The metric provides information on how to make the task mapping schedulable at the minimal frequency. Since frequency scaling affects the power consumption of the system, it is important to measure how much power will be consumed by the system if the breakdown frequency is applied. This provides another direction for future work, in which the function can be further improved by including a new power macro-model which is related to the frequency in the calculation of power consumption.

In the end-to-end schedulability analysis, the iterative method used to calculate the worst-case response time has the characteristic of pseudo-polynomial complexity. With an increase in processing and communication volume, the analysis may impose considerable delays on the optimisation process. In practice, system designers face time pressure to find approximate solutions quickly, thus at the early stage of design, analysing every task and message may not be a preferred way to accomplish this task. Therefore, an approximate approach with lower computational complexity in the fitness function is desired [148].

Appendix A

Table 1: Autonomous vehicle application tasks

Task	**Task description**	**Computation time**	**Period**
TPMS	Tyre pressure monitoring system	0.005	0.5
VIBS	Vibration sensor	0.005	0.1
SPES	Speed sensor	0.005	0.1
POSI	Position sensor interface	0.005	0.5
USOS	Ultrasonic sensor	0.005	0.1
FBU1	Frame buffer - Left camera, upper-left quadrant	0.01	0.4
FBU2	Frame buffer - Left camera, upper-right quadrant	0.01	0.4
FBU3	Frame buffer - Left camera, lower-left quadrant	0.01	0.4
FBU4	Frame buffer - Left camera, lower-right quadrant	0.01	0.4
FBU5	Frame buffer - Right camera, upper-left quadrant	0.01	0.4
FBU6	Frame buffer - Right camera, upper-right quadrant	0.01	0.4
FBU7	Frame buffer - Right camera, lower-left quadrant	0.01	0.4
FBU8	Frame buffer - Right camera, lower-right quadrant	0.01	0.4
STAC	Stability control	0.01	1
TPRC	Tyre pressure control	0.001	0.01
DIRC	Direction control	0.001	0.01
OBDB	Obstacle database	0.15	0.5
BFE1	Background estimation and feature extraction 1	0.02	0.04
BFE2	Background estimation and feature extraction 2	0.02	0.04
BFE3	Background estimation and feature extraction 3	0.02	0.04
BFE4	Background estimation and feature extraction 4	0.02	0.04
BFE5	Background estimation and feature extraction 5	0.02	0.04
BFE6	Background estimation and feature extraction 6	0.02	0.04

Task	**Task description**	**Computation time**	**Period**
BFE7	Background estimation and feature extraction 7	0.01	0.04
BFE8	Background estimation and feature extraction 8	0.01	0.04
FDF1	Feature data fusion 1	0.01	0.4
FDF2	Feature data fusion 2	0.01	0.4
STPH	Stereo photogrammetry	0.03	0.04
THRC	Throttle control	0.001	0.01
VOD1	Visual odometry 1	0.02	0.04
VOD2	Visual odometry 2	0.02	0.04
OBMG	Obstacle database manager	0.02	1
NAVC	Navigation control	0.01	0.5

Table 2: Traffic flows between tasks

Flow	Source	Destination	Flits	Period
1	POSI	NAVC	1024	0.5
2	NAVC	OBDB	2048	0.5
3	OBDB	NAVC	16384	0.5
4	OBDB	OBMG	32768	1
5	NAVC	DIRC	512	0.1
6	SPES	NAVC	512	0.1
7	NAVC	THRC	1024	0.1
8	FBU3	VOD1	38400	0.04
9	FBU8	VOD2	38400	0.04
10	VOD1	NAVC	512	0.04
11	VOD2	NAVC	512	0.04
12	FBU1	BFE1	38400	0.04
13	FBU2	BFE2	38400	0.04
14	FBU3	BFE3	38400	0.04
15	FBU4	BFE4	38400	0.04
16	FBU5	BFE5	38400	0.04
17	FBU6	BFE6	38400	0.04
18	FBU7	BFE7	38400	0.04
19	FBU8	BFE8	38400	0.04
20	BFE1	FDF1	2048	0.04
21	BFE2	FDF1	2048	0.04
22	BFE3	FDF1	2048	0.04
23	BFE4	FDF1	2048	0.04
24	BFE5	FDF2	2048	0.04
25	BFE6	FDF2	2048	0.04
26	BFE7	FDF2	2048	0.04
27	BFE8	FDF2	2048	0.04
28	FDF1	STPH	8192	0.04
29	FDF2	STPH	8192	0.04
30	STPH	OBMG	4096	0.04
31	POSI	OBMG	1024	0.5
32	USOS	OBMG	1024	0.1
33	OBMG	OBDB	4096	1
34	TPMS	STAC	2048	0.5
35	VIBS	STAC	512	0.1
36	STAC	TPRC	2048	1
37	SPES	STAC	1024	0.1
38	STAC	THRC	1024	0.1

Appendix B

```
procedure SchedulabilityTest($TaskMapping$)
    for each $task_i$ in $TaskMapping$ do
        if $PassSchedulabilityTest(task_i)! = true$ then return $false$
        end if
    end for
    for each $flow_i$ in $TaskMapping$ do
        if $PassSchedulabilityTest(flow_i)! = true$ then return $false$
        end if
    end for
    return $true$
end procedure
procedure ScaleFrequency($Schedulable, CurrentFrequency$)
    if $Schedulable == true$ then
        $NewFrequency = DecreaseFrequency(CurrentFrequency)$
    else
        $NewFrequency = IncreaseFrequency(CurrentFrequency)$
    end if
    return $NewFrequency$
end procedure
procedure CalculateBreakdownFrequency($TaskMapping$)
    $Frequency = ScaleFrequency(SchedulabilityTest(TaskMapping), 1)$
    while $Frequency! = null$ do
        $SetFrequency(Frequency)$
        $Schedulability = SchedulabilityTest(TaskMapping)$
        $Frequency = ScaleFrequency(Schedulability, Frequency)$
    end while
    return $Frequency$
end procedure
```

Figure 1: Breakdown frequency fitness function

Glossary

Roman Symbols

ACO	Ant Colony Optimisation
ADL	Architecture Description Language
ADT	Abstract Data Type
ATM	Asynchronous Transfer Mode
AVA	Autonomous Vehicle Application
AXI	Advanced Extensible Interface
BE	Best Effort
BF	Best Fit
BFF	Breakdown Frequency Fitness Function
CAN	Controller Area Network
CoA	Constructive Mapping Algorithm
CPU	Central Processing Unit
DM	Deadline Monotonic
DSE	Design Space Exploration
DTL	Device Transaction Level

EDF	Earliest Deadline First
ESL	Electronic System Level
FF	First Fit
FIFO	First In First Out
GA	Genetic Algorithm
GALS	Globally-Asynchronous Locally-Synchronous Systems
GT	Guaranteed Traffic
HDL	Hardware Description Language
IP	Intellectual Property
ISA	Instruction Set Architectures
ISS	Instruction Set Simulator
LLF	Least Laxity First
MMIO	Memory Mapped IO
MoC	Models of Computation
MOGA	Multi-Objective Genetic Algorithm
MPSoC	Multiprocessor System-on-Chip
NI	Network Interface
NN	Nearest Neighbour
NoC	Networks-On-Chip
OCP	Open Core Protocol
P2P	Point-to-Point
PSO	Particle Swarm Optimisation

RC	Routing Computation
RM	Rate-Monotonic
RMGT	Rate Monotonic General Task
RMM	Rate Monotonic Matching
RN	Random Mapper
RTA	Real-Time Analysis
RTL	Registers Transfer Level
SA	Simulated Annealing
SA	Switch Allocation
SAP	Synthetic Application
SCF	Schedulability Fitness Function
SDF	Synchronous Data Flow Graphs
SoC	System-On-Chip
SOGA	Single Objective Genetic Algorithm
TB	Tabu Search
TDMA	Time Division Multiple Access
TLM	Transaction-Level Modelling
TTM	Time-to-Market
TTP	Time-Triggered Protocol
VA	Virtual Channel Arbitration
VC	Virtual Channel
VCI	Virtual Component Interface

$VHDL$	VHSIC Hardware Description Language
$WCET$	Worst-Case Execution Time
WF	Worst Fit

References

[1] W. Wolf, A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (mpsoc) technology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, pp. 1701–1713, 2008. 3

[2] P. Aldworth, "System-on-a-chip bus architecture for embedded applications," in *International Conference on Computer Design*, pp. 297–298, 1999. 3, 4

[3] B. Cordan, "An efficient bus architecture for system-on-chip design," in *Proceedings of the Custom Integrated Circuits*, pp. 623–626, 1999. 3

[4] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, pp. 23:1–23:20, May 2008. 3

[5] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Design Automation Conference*, pp. 684–689, 2001. 4, 18

[6] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, pp. 70–78, 2002. 4, 16

[7] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*. Addison-Wesley Educational Publishers Inc, 2009. 5

[8] J. C. Knight, "Safety critical systems: challenges and directions," in *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, pp. 547–550, IEEE, 2002. 5

[9] R. Marculescu, U. Ogras, L.-S. Peh, N. Jerger, and Y. Hoskote, "Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2009. 6

[10] Matthias and Gries, "Methods for evaluating and covering the design space during early design development," *The VLSI Journal Integration*, vol. 38, pp. 131–183, 2004. 9, 33, 39, 40, 51, 54

[11] P. Mesidis and L. S. Indrusiak, "Genetic mapping of hard real-time applications onto NoC-based MPSoCs – A first approach," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2011. 10, 11, 31, 33, 36, 37, 63, 88, 89, 90, 100, 108, 109, 118, 127

[12] Z. Shi and A. Burns, "Priority assignment for real-time wormhole communication in on-chip networks," in *Real-Time Systems Symposium*, IEEE, 2008. 10, 35, 91

[13] M. Richard, P. Richard, and F. Cottet, "Allocating and scheduling tasks in multiple fieldbus real-time systems," in *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA'03. IEEE Conference*, 2003. 10

[14] P.-E. Hladik, H. Cambazard, A.-M. Déplanche, and N. Jussien, "Solving a real-time allocation problem with constraint programming," *Journal of Systems and Software*, 2008. 10

[15] A. Mehiaoui, E. Wozniak, S. Tucci-Piergiovanni, C. Mraidha, M. Di Natale, H. Zeng, J.-P. Babau, L. Lemarchand, and S. Gerard, "A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems," *ACM SIGPLAN Notices*, 2013. 10

[16] L. Benini and G. De Micheli, "Powering networks on chips: Energy-efficient and reliable interconnect design for socs," in *Proceedings of the 14th Inter-*

national Symposium on Systems Synthesis, ISSS '01, (New York, NY, USA), pp. 33–38, ACM, 2001. 10

[17] G. Ascia, V. Catania, and M. Palesi, "Multi-objective mapping for mesh-based NoC architectures," in *CODES+ISSS*, 2004. 10, 29, 47

[18] W. Zhou, Y. Zhang, and Z. Mao, "Pareto based multi-objective mapping ip cores onto NoC architectures," in *APCCAS*, 2006. 10, 30, 31, 100

[19] R. K. Jena and G. K. Sharma, "A multi-objective evolutionary algorithm based optimization model for network-on-chip synthesis," in *ITNG*, 2007. 10, 30, 100

[20] N. Nedjah, M. Silva, and L. Mourelle, "Customized computer-aided application mapping on NoC infrastructure using multi-objective optimization," *JSA*, vol. 57, pp. 79–94, 2011. 10, 30, 31, 100

[21] A. Racu and L. Indrusiak, "Using genetic algorithms to map hard real-time on NoC-based systems," in *ReCoSoC*, 2012. 10, 11, 31, 32, 33, 35, 36, 37, 91, 100, 118

[22] L. Benini and D. Bertozzi, "Network-on-chip architectures and design methods," *IEEE Proceedings on Computers and Digital Techniques*, vol. 152, pp. 261–272, 2005. 16

[23] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Comput. Surv.*, vol. 38, 2006. 17, 18

[24] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection networks: An engineering approach*. Morgan Kaufmann, 2003. 17, 21, 22

[25] T. Chelcea and S. M. Nowick, "Robust interfaces for mixed-timing systems with application to latency-insensitive protocols," in *Proceedings of the 38th annual Design Automation Conference*, pp. 21–26, 2001. 17

[26] S. Furber, *Principles of asynchronous circuit design: a systems perspective*. Springer, 2001. 17

[27] OCPIP, "Open core protocol (ocp) specification, release 2.0.," 2003. 18

[28] V. Alliance, "Virtual component interface standard version 2," 2000. 18

[29] P. Semiconductors, "Device transaction level (dtl) protocol specication, version 2.2," 2002. 18

[30] T. Bjerregaard, S. Mahadevan, R. Olsen, and J. Sparsoe, "An ocp compliant network adapter for gals-based soc design using the mango network-on-chip," in *International Symposium on System-on-Chip*, pp. 171–174, 2005. 18

[31] A. Radulescu, J. Dielissen, K. Goossens, E. Rijpkema, and P. Wielage, "An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network configuration," in *Design, Automation and Test in Europe Conference and Exhibition*, vol. 2, pp. 878–883, 2004. 18

[32] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proceedings of the conference on Design, Automation and Test in Europe*, pp. 250–256, 2000. 18

[33] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *IEEE Computer Society Annual Symposium on VLSI*, pp. 105–112, 2002. 18, 23

[34] W. Dally, "Virtual-channel flow control," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 3, pp. 194–205, 1992. 19, 21, 24, 25

[35] N. Kavaldjiev, G. Smit, and P. Jansen, "A virtual channel router for on-chip networks," in *IEEE International SOC Conference*, pp. 289–293, 2004. 19, 21, 23

[36] F. Moraes, N. Calazans, A. Mello, L. Mller, and L. Ost, "Hermes: an infrastructure for low area overhead packet-switching networks on chip," *The VLSI Journal Integration*, vol. 38, pp. 69–93, 2004. 19, 21, 23, 55

[37] A. Radulescu and K. Goossens, "Communication services for networks on chip," *Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation*, pp. 193–213, 2004. 20

[38] V. Soteriou, R. Ramanujam, B. Lin, and L.-S. Peh, "A high-throughput distributed shared-buffer noc router," *Computer Architecture Letters*, vol. 8, pp. 21–24, 2009. 20

[39] R. Ramanujam, V. Soteriou, B. Lin, and L.-S. Peh, "Design of a high-throughput distributed shared-buffer noc router," in *International Symposium on Networks-on-Chip*, pp. 69–78, 2010. 20

[40] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Qnoc: Qos architecture and design process for network on chip," *Journal of Systems Architecture*, vol. 50, pp. 105–128, 2004. 21, 23

[41] L.-S. Peh and W. Dally, "A delay model for router microarchitectures," *Micro, IEEE*, vol. 21, pp. 26–34, 2001. 22

[42] P. Kermani and L. Kleinrock, "Virtual cut-through: a new computer communication switching technique," *Computer Networks*, vol. 3, pp. 267–286, 1979. 22

[43] W. J. Dally and C. L. Seitz, "The torus routing chip," *Distributed Computing*, vol. 1, pp. 187–196, 1986. 22

[44] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Virtual channels in networks on chip: Implementation and evaluation on hermes noc," in *18th Symposium on Integrated Circuits and Systems Design*, pp. 178–183, 2005. 23

[45] K. Goossens, J. Dielissen, and A. Radulescu, "Aethereal network on chip: concepts, architectures, and implementations," *Design Test of Computers*, vol. 22, pp. 414–421, 2005. 23

[46] W. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, pp. 547–553, 1987. 24

[47] W. Dally, “Performance analysis of k-ary n-cube interconnection networks,” *IEEE Transactions on Computers*, vol. 39, pp. 775–785, 1990. 25

[48] E. Beigné, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, “An asynchronous noc architecture providing low latency service and its multi-level design framework,” in *Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings. 11th IEEE International Symposium on*, pp. 54–63, IEEE, 2005. 26

[49] K. W. Tindell, A. Burns, and A. J. Wellings, “Allocating hard real-time tasks: An np-hard problem made easy,” *Real-Time Systems*, vol. 4, pp. 145–165, 1992. 27, 60

[50] T. Lei and S. Kumar, “A two-step genetic algorithm for mapping task graphs to a network on chip architecture,” in *Euromicro Symposium on Digital System Design*, pp. 180–187, 2003. 28, 31, 58

[51] S. Murali and G. De Micheli, “Bandwidth-constrained mapping of cores onto noc architectures,” in *The conference on Design, Automation and Test in Europe*, 2004. 28, 58

[52] J. Hu and R. Marculescu, “Energy-aware mapping for tile-based noc architectures under performance constraints,” in *Design Automation Conference*, pp. 233–239, 2003. 28, 30, 31

[53] E. Zitzler, M. Laumanns, and L. Thiele, “SPEA2: Improving the Strength Pareto Evolutionary Algorithm,” tech. rep., Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK), 2001. 29, 49

[54] K. Z. Ibrahim, “Correlation between detailed and simplified simulations in studying multiprocessor architecture,” in *International Conference on Computer Design*, pp. 387–392, 2005. 29, 51

[55] N. Srinivas and K. Deb, “Multiobjective optimization using nondominated sorting in genetic algorithms,” *Evolutionary computation*, vol. 2, pp. 221–248, 1994. 30, 45

[56] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, 2002. 30, 49, 101

[57] C. A. Coello Coello and G. Toscano Pulido, "A Micro-Genetic Algorithm for Multiobjective Optimization," in *First International Conference on Evolutionary Multi-Criterion Optimization* (E. Zitzler, K. Deb, L. Thiele, C. A. C. Coello, and D. Corne, eds.), pp. 126–140, Springer-Verlag. Lecture Notes in Computer Science, 2001. 30

[58] M. J. Sepúlveda, W. J. Chau, G. Gogniat, and M. Strum, "A multi-objective adaptive immune algorithm for multi-application noc mapping," *Analog Integrated Circuits and Signal Processing*, vol. 73, no. 3, pp. 851–860, 2012. 31

[59] J. Wang, Y. Li, S. CHAI, and Q. Peng, "Bandwidth-aware application mapping for noc-based mpsocs," *Journal of Computational Information Systems*, vol. 7, no. 1, pp. 152–159, 2011. 31

[60] A. E. H. Benyamina, P. Boulet, A. Aroui, S. Eltar, and K. Dellal, "Mapping Real Time Applications on NoC Architecture with Hybrid Multi-objective Algorithm," in *META'10 Intenational Conference on Metaheuristics and Nature Inspired Computing*, (Djerba Island, Tunisia), Oct. 2010. 31

[61] J. Gan, P. Pop, F. Gruian, and J. Madsen, "Robust and flexible mapping for real-time distributed applications during the early design phases," in *Conference on Design, Automation and Test in Europe*, 2012. 31, 32, 33, 37

[62] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Real Time Systems Symposium*, 1989. 33, 35, 67, 69, 70, 89

[63] J. Garcia and M. Harbour, "Optimized priority assignment for tasks and messages in distributed hard real-time systems," in *Third Workshop on Parallel and Distributed Real-Time Systems*, 1995. 34, 72

[64] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *J. ACM*, 1973. 34, 35, 60, 82

[65] A. Ka and L. Mok, *Fundamental design problems of distributed systems for the hard-real-time environment.* MIT Thesis, 1983. 35

[66] M. W. Mutka, “Using rate monotonic scheduling technology for real-time communications in a wormhole network,” in *Second Workshop on Parallel and Distributed Real-Time Systems*, 1994. 35

[67] R. I. Davis and A. Burns, “A survey of hard real-time scheduling for multiprocessor systems,” *ACM Comput. Surv.*, vol. 43, pp. 1–44, 2011. 37, 39, 72

[68] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, “The worst-case execution-time problem–Overview of methods and survey of tools,” *ACM Trans. Embed. Comput. Syst.*, 2008. 37, 69, 119

[69] S. Saewong and R. Rajkumar, “Practical voltage-scaling for fixed-priority RT-systems,” in *Real-Time and Embedded Technology and Applications Symposium*, 2003. 37

[70] E. Bini, G. Buttazzo, and G. Lipari, “Speed modulation in energy-aware real-time systems,” in *ECRTS*, 2005. 37

[71] P. Mejia Alvarez, E. Levner, and D. Mossé, “Adaptive scheduling server for power-aware real-time tasks,” *ACM Trans. Embed. Comput. Syst.*, 2004. 37

[72] D. Shin and J. Kim, “Power-aware communication optimization for networks-on-chips with voltage scalable links,” in *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2004. 37

[73] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, “System-level design: orthogonalization of concerns and platform-based de-

sign," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, pp. 1523–1543, 2000. 39, 69

[74] B. Kienhuis, E. Deprettere, K. Vissers, and P. Van Der Wolf, "An approach for quantitative analysis of application-specific dataflow architectures," in *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 338–349, 1997. 39, 69

[75] M. Eisenring, L. Thiele, and E. Zitzler, "Conflicting Criteria in Embedded System Design," *IEEE Design and Test*, vol. 17, pp. 51–59, 2000. 40

[76] W. Wolf, "Hardware-software co-design of embedded systems [and prolog]," *Proceedings of the IEEE*, vol. 82, pp. 967–989, 1994. 41

[77] G. De Michell and R. Gupta, "Hardware/software co-design," *Proceedings of the IEEE*, vol. 85, pp. 349–365, 1997. 41

[78] B. Bailey, *ESL Design and Verification: A Prescription for Electronic System Level Methodology.* San Diego, CA: Elsevier, 2007. 42

[79] L. Cai and D. Gajski, "Transaction level modeling: an overview," in *International Conference on Hardware/Software Codesign and System Synthesis*, pp. 19–24, 2003. 42, 54

[80] V. Pareto, *Cours d'economie politique.* Librairie Droz, 1964. 44

[81] J. Horn, "Multicriterion Decision Making," in *Handbook of Evolutionary Computation* (T. Bäck, D. Fogel, and Z. Michalewicz, eds.), vol. 1, pp. F1.9:1 – F1.9:15, IOP Publishing Ltd. and Oxford University Press, 1997. 45, 48

[82] C. M. Fonseca and P. J. Fleming, "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization," in *Proceedings of the Fifth International Conference on Genetic Algorithms* (S. Forrest, ed.), (San Mateo, California), pp. 416–423, University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers, 1993. 47

[83] C. Fonseca and P. Fleming, “Multiobjective Genetic Algorithms,” in *IEEE Colloquium on Genetic Algorithms for Control Systems Engineering*, pp. 1–5, 1993. 47

[84] S. Kirkpatrick, C. Gelatt Jr, and M. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, pp. 671–680, 1983. 48

[85] F. Glover *et al.*, “Tabu search-part i,” *ORSA Journal on computing*, vol. 1, pp. 190–206, 1989. 48

[86] F. Glover, “Tabu search-part ii,” *ORSA Journal on computing*, vol. 2, pp. 4–32, 1990. 48

[87] E. L. Lawler and D. E. Wood, “Branch-and-bound methods: A survey,” *Operations Research*, vol. 14, pp. 699–719, 1966. 48

[88] J. Holland, “Adaptation in natural and artificial systems,” *Ann Arbor: University of Michigan Press*, vol. 2, 1975. 48

[89] J. D. Knowles and D. W. Corne, “The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Multiobjective Optimisation,” in *Congress on Evolutionary Computation*, pp. 98–105, 1999. 49

[90] S. Cieniawski, J. Eheart, and S. Ranjithan, “Using genetic algorithms to solve a multiobjective groundwater monitoring problem,” *Water Resources Research*, vol. 31, pp. 399–409, 1995. 49

[91] D. Nam and C. Park, “Multiobjective simulated annealing: A comparative study to evolutionary algorithms,” *International Journal of Fuzzy Systems*, vol. 2, pp. 87–97, 2000. 49

[92] S. Cieniawski, *An Investigation of the Ability of Genetic Algorithms to Generate the Tradeoff Curve of a Multi-objective Groundwater Monitoring Problem*. University of Illinois at Urbana-Champaign, 1993. 49

[93] E. Zitzler, K. Deb, and L. Thiele, “Comparison of Multiobjective Evolutionary Algorithms: Empirical Results,” *Evolutionary Computation*, vol. 8, pp. 173–195, 2000. 49

[94] A. Geraci, F. Katki, L. McMonegal, B. Meyer, J. Lane, P. Wilson, J. Radatz, M. Yee, H. Porteous, and F. Springsteel, *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries.* IEEE Press, 1991. 50

[95] N. Kitchen and A. Kuehlmann, "Stimulus generation for constrained random simulation," in *IEEE/ACM international conference on Computer-aided design*, pp. 258–265, 2007. 50

[96] A. Jantsch, *Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation.* Morgan Kaufmann, 2004. 51

[97] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, pp. 1235–1245, 1987. 51

[98] E. Lee and A. Sangiovanni-Vincentelli, "A framework for comparing models of computation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 1217–1229, 1998. 51

[99] L. Eeckhout, R. H. Bell Jr., B. Stougie, K. D. Bosschere, and L. K. John, "Control flow modeling in statistical simulation for accurate and efficient processor design studies," *SIGARCH Comput. Archit. News*, vol. 32, p. 350, 2004. 51

[100] S. Nussbaum and J. Smith, "Statistical simulation of symmetric multiprocessor systems," in *35th Annual Simulation Symposium*, pp. 89–97, 2002. 51

[101] J. Zhu and D. Gajski, "A retargetable, ultra-fast instruction set simulator," in *Design, Automation and Test in Europe Conference*, pp. 298–302, 1999. 52

[102] A. Nohl, G. Braun, O. Schliebusch, R. Leupers, H. Meyr, and A. Hoffmann, "A universal technique for fast and flexible instruction-set architecture simulation," in *Proceedings of the 39th annual Design Automation Conference*, pp. 22–27, 2002. 52

[103] M. Reshadi, P. Mishra, and N. Dutt, “Instruction set compiled simulation: a technique for fast and flexible instruction set simulation,” in *Design Automation Conference*, pp. 758–763, 2003. 52

[104] T. Austin, E. Larson, and D. Ernst, “Simplescalar: an infrastructure for computer system modeling,” *Computer*, vol. 35, pp. 59–67, 2002. 52

[105] J. Rowson, “Hardware/software co-simulation,” in *Design Automation Conference*, pp. 439–440, 1994. 52

[106] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino, “SystemC cosimulation and emulation of multiprocessor soc designs,” *Computer*, vol. 36, pp. 53–59, 2003. 52, 53

[107] F. Fummi, S. Martini, G. Perbellini, and M. Poncino, “Native iss-systemC integration for the Co-simulation of multi-processor SoC,” in *Design, Automation and Test in Europe*, 2004. 52, 53

[108] A. Wieferink, M. Doerper, T. Kogel, R. Leupers, G. Ascheid, and H. Meyr, “Early iss integration into network-on-chip designs,” *Computer Systems: Architectures, Modeling, and Simulation*, pp. 443–452, 2004. 52, 53

[109] T. Schonwald, J. Zimmermann, O. Bringmann, and W. Rosenstiel, “Network-on-chip architecture exploration framework,” in *12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, pp. 375–382, 2009. 52, 53

[110] T. Kogel, M. Doerper, A. Wieferink, R. Leupers, G. Ascheid, H. Meyr, and S. Goossens, “A modular simulation framework for architectural exploration of on-chip interconnection networks,” in *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 7–12, 2003. 53, 54, 55, 57

[111] A. Hoffmann, T. Kogel, A. Nohl, G. Braun, O. Schliebusch, O. Wahlen, A. Wieferink, and H. Meyr, “A novel methodology for the design of

application-specific instruction-set processors (asips) using a machine description language," *Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, pp. 1338–1354, 2001. 53

[112] S. Pestana, E. Rijpkema, A. Radulescu, K. Goossens, and O. Gangwal, "Cost-performance trade-offs in networks on chip: a simulation-based approach," in *Design, Automation and Test in Europe Conference*, vol. 2, pp. 764–769, 2004. 54, 55

[113] L. Indrusiak and O. dos Santos, "Fast and accurate transaction-level model of a wormhole network-on-chip with priority preemptive virtual channel arbitration," in *Design, Automation Test in Europe Conference*, pp. 1–6, 2011. 54, 55

[114] C. Grecu, A. Ivanov, R. Saleh, C. Rusu, L. Anghel, P. Pande, and V. Nuca, "A flexible network-on-chip simulator for early design space exploration," in *Microsystems and Nanoelectronics Research Conference, 2008. MNRC 2008. 1st*, pp. 33–36, 2008. 56, 57

[115] C. Cornelius, F. Sill, and D. Timmermann, "High-level simulations of on-chip networks," *Proc. of the 9th DSD, Cavtat, Croatia*, 2006. 56

[116] L. Ost, F. G. Moraes, L. Möller, L. S. Indrusiak, M. Glesner, S. Määttä, and J. Nurmi, "A simplified executable model to evaluate latency and throughput of networks-on-chip," in *Proceedings of the 21st annual symposium on Integrated circuits and system design*, pp. 170–175, 2008. 57

[117] U. Ogras, P. Bogdan, and R. Marculescu, "An analytical approach for network-on-chip performance analysis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, pp. 2001–2013, 2010. 58

[118] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, pp. 237–250, 1982. 60

[119] J. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Real-Time Systems Symposium*, pp. 201–209, 1990. 60

[120] N. Audsley, "On priority assignment in fixed priority scheduling," *Information Processing Letters*, vol. 79, pp. 39–44, 2001. 60

[121] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem," *Operations Research*, vol. 26, pp. 127–140, 1978. 60

[122] Y. Oh and S. H. Son, "Tight performance bounds of heuristics for a real-time scheduling problem," -, 1993. 60

[123] D.-T. Peng, K. Shin, and T. Abdelzaher, "Assignment and scheduling communicating periodic tasks in distributed real-time systems," *IEEE Transactions on Software Engineering*, vol. 23, pp. 745–758, 1997. 60

[124] T. Rothvoss, "On the computational complexity of periodic scheduling," 2009. 61

[125] A. Burchard, J. Liebeherr, Y. Oh, and S. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *Computers, IEEE Transactions on*, vol. 44, pp. 1429–1442, 1995. 61

[126] D.-I. Oh and T. Bakker, "Utilization bounds for n-processor rate monotone scheduling with static processor assignment," *Real-Time Systems*, vol. 15, pp. 183–192, 1998. 61

[127] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *Real-Time Systems Symposium*, pp. 193–202, 2001. 61

[128] J. M. Lpez, M. Garca, J. L. Daz, and D. F. Garca, "Utilization bounds for multiprocessor rate-monotonic scheduling," *Real-Time Systems*, vol. 24, pp. 5–28, 2003. 61

[129] J. Lopez, J. Diaz, and D. Garcia, "Minimum and maximum utilization bounds for multiprocessor rate monotonic scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, pp. 642–653, 2004. 61

[130] K. Tindell and J. Clark, “Holistic schedulability analysis for distributed hard real-time systems,” *Microprocessing and Microprogramming*, vol. 40, pp. 117–134, 1994. 62

[131] P. Pop, P. Eles, and Z. Peng, “Schedulability-driven communication synthesis for time triggered embedded systems,” *Real-Time Systems*, vol. 26, pp. 297–325, 2004. 62

[132] A. Ermedahl, H. Hansson, and M. Sjodin, “Response-time guarantees in atm networks,” in *Real-Time Systems Symposium*, pp. 274–284, 1997. 62

[133] K. Tindell, A. Burns, and A. Wellings, “Calculating controller area network (can) message response times,” *Control Engineering Practice*, vol. 3, pp. 1163–1169, 1995. 62

[134] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, “System level performance analysis - the symta/s approach,” *IEEE Proceedings on Computers and Digital Techniques*, vol. 152, pp. 148–166, 2005. 62

[135] Z. Shi, A. Burns, and L. Indrusiak, “Schedulability analysis for real time on-chip communication with wormhole switching,” *IJERTCS*, 2010. 63, 64, 71, 73, 75, 76

[136] H. Kashif and H. D. Patel, “Bounding buffer space requirements for real-time priority-aware networks,” in *ASP-DAC*, 2014. 64

[137] M. N. S. M. Sayuti, L. S. Indrusiak, and A. Garcia-Ortiz, “An optimisation algorithm for minimising energy dissipation in noc-based hard real-time embedded systems,” in *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, 2013. 64

[138] 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*, 2014. 64

[139] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings, “Fixed priority pre-emptive scheduling: An historical perspective,” *Real-Time Systems*, 1995. 70

[140] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, 1993. 73, 74, 76

[141] J. J. Durillo and A. J. Nebro, "jmetal: A java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, pp. 760–771, 2011. 87

[142] S. Luke, L. Panait, Z. Skolicki, J. Bassett, R. Hubley, and A. Chircop, "Ecj: A java-based evolutionary computation and genetic programming research system," *http://www.cs.umd.edu/projetc/plus/ec/ecj/*, 2001. 87

[143] K. Meffert, N. Rotstan, C. Knowles, and U. Sangiorgi, "Jgap-java genetic algorithms and genetic programming package," *http://jgap.sf.net*, 2008. 87

[144] M. N. S. M. Sayuti and L. S. Indrusiak, "Real-time low-power task mapping in networks-on-chip," in *IEEE Computer Society Annual Symposium on VLSI*, 2013. 91

[145] M. Palesi, G. Ascia, F. Fazzino, and V. Catania, "Data encoding schemes in networks on chip," *IEEE TCAD*, vol. 30, pp. 774–786, 2011. 104, 109

[146] D. Bertozzi and L. Benini, "Xpipes: a network-on-chip architecture for gigascale systems-on-chip," *Circuits and Systems Magazine, IEEE*, vol. 4, no. 2, pp. 18 – 31, 2004. 105

[147] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *Mobile Computing* (T. Imielinski and H. Korth, eds.), vol. 353 of *The Kluwer International Series in Engineering and Computer Science*, pp. 449–471, Springer US, 1996. 121

[148] Y. Ma, M. N. S. Mohd 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 *9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pp. 1–8, May 2014. 161