Hardware-accelerated Evolutionary Hard Real-Time Task Mapping for Wormhole Network-on-Chip with Priority-Preemptive Arbitration

Yunfeng Ma

PhD

University of York
Computer Science

September, 2016

Abstract

Network-on-Chip (NoC) is an alternative on-chip interconnection paradigm to replace existing ones such as Point-to-Point and shared bus. NoCs designed for hard real-time systems need to guarantee the system timing performance, even in the worst-case scenario. A carefully planned task mapping which indicates how tasks are distributed on a NoC platform can improve or guarantee their timing performance. While existing offline mapping optimisations can satisfy timing requirements, this is obtained by sacrificing the flexibility of the system. In addition, the design exploration process will be prolonged with the continuous enlargement of the design space. Online mapping optimisations, by contrast, are affected by low success rates for remapping or a lack of guarantee of systems timing performance after remapping, especially in hard real-time systems. The existing limitations therefore motivate this research to concentrate on the mapping optimisation of real-time NoCs, and specifically dynamic task allocation in hard real-time systems.

Four techniques and implementations are proposed to address this issue. The first enhances the evaluation efficiency of a hard real-time evaluation method from a theoretical point of view. The second technique addresses the evaluation efficiency from a practical point of view to enable online hard real-time timing analysis. The third technique advocates a dynamic mapper to enhance the remapping success rate with the accelerated model and architecture. The final technique yields a dynamic mapping algorithm that can search schedulable task allocation for hard real-time NoCs at run time, while simultaneously reducing the task migration cost after remapping.

Contents

List of Tables

List of Figures

Acknowledgement

A Ph.D study is a long and tough journey with tears and smiles, pain but happiness with sense of accomplishments.

I would like to first express my sincere gratitude to my supervisor Dr. Leandro Soares Indrusiak and advisor Prof. Alan Burns for their guidance and support through my journey. Their insightful comments, encouragement, and the hard questions push me to widen my research from various perspectives.

I would especially like to thank my parents for their endless love and support; without them I would not have the opportunity to receive a higher-standard education. They always believe in me and cheer me up when I doubt myself.

The special thanks also go to my wife Mrs. Xiaoyuan Chen for supporting me spiritually throughout writing this thesis and my life in general. I also give my thanks to my lovely son Waylon Ma. Your birth brings me the endless happiness and let me know the responsbility for being a good father. You are the source to encourage me to move forward in difficulties.

Many thanks also go to my friends: Dr. Yuan Wang, Mr. Zhe Jiang, Mr. Hao Wei, Mrs. Guanlang Hu – for blowing my mind with the discussion of research questions and support me when I run to troubles.

Finally, I give my deepest thanks to all the people who helped me. Thank you and with you a good luck!

Declaration

I, Yunfeng Ma, declare that the research presented in this thesis is my original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. Some parts of this thesis have been presented at and published in the proceedings of various conferences, or journals. All sources are acknowledged as References.

1 Y. Ma, M. N. S. Mohd Sayuti and L. S. 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 Proceedings of the 9th International Symposium on Reconfigurable Communication-centric System-on-Chip (ReCoSoC), 2014.

2 Y. Ma and L. S. Indrusiak, Hardware-Accelerated Response Time Analysis for priority-preemptive Networks-on-Chip, 10th International Symposium on Reconfigurable and Communication-centric Systems-on-Chip (ReCoSoC), 2015.

3 Y. Ma and L. S. Indrusiak, Hardware-Accelerated Parallel Genetic Algorithm for Fitness Functions with Variable Execution Times, in Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), 2016.

4 Y. Ma and L. S. Indrusiak, Hardware-Accelerated Analysis of Real-Time Networks-on-Chip, in Microprocessors and Microsystems (MICPRO), 2017.

Chapter 1

Introduction

1.1 Background

Continuous development of technologies in the manufacture of semiconductors has led to significant shrinkage in the physical size of transistors in the deep sub-micron domain [121]. Billions of transistors can be placed on a single chip, thus allowing more powerful processing. An example of this is the Haswell, which is Intel's fourth generation processor, implemented on its 22nm trigate process technology [70]. Current CMOS technologies already support a Multi-Processor System on Chip (MPSoC) implementation integrating hundreds of cores [121]. This has made the current means of on-chip interconnections (shared buses or point-to-point) impractical due to a lack of scalability, predictability and reusability. In addition, the International Roadmap for Semiconductors (ITRS) forecasts that nodes implemented with 5nm technology will be achieved around 2019 [27]. This will enhance the need for an alternative on-chip interconnection paradigm. Consequently, Network on Chips (NoCs), which is inspired by general computer networks, has been proposed as an alternative on-chip interconnection architecture and attracted more and more attention from both academia and industry, for example Arteris and Sonics, two major vendors of NoC solutions (FlexNoC

and SonicsGN).

1.2 Timing Performance

As an alternative interconnection paradigm for MPSoCs, NoCs need to have the ability to support the concurrent execution of multiple IP (Intellectual Property) cores and the message exchange between them through the underlying communication infrastructure (shown in Figure 1.1). In real-time systems not only do the messages need to be generated and transferred correctly, but the message computation and communication also need to be finished within a given time bound (normally before a predefined deadline). This requirement is common in safety critical systems, such as the engine control of a vehicle and the fly control unit of a plane, because any message error or timing violation could cause the response time of one or more system functions to exceed the stipulated time period and further result in incorrect or late responses in practice. For example, if the reaction of the brake function in a car system is delayed, the braking distance could be longer than the predicted range and may cause an accident. This requirement is also known as the timing performance of NoCs and is used as an evaluation criterion.

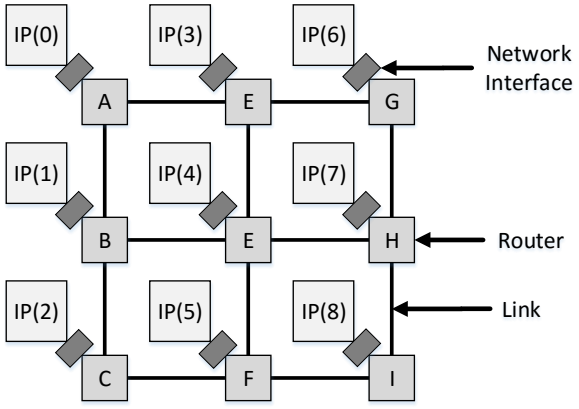

Figure 1.1: NoC Architecture Example.

1.2.1 NoC Design Flow

The timing performance of NoCs can be guaranteed by deploying tasks (sub-programs of application) to IPs with a carefully planned task allocation, which is a one-to-one, multiple-to-one or multiple-to-multiple mapping relation between tasks and IPs. A suitable mapping can be obtained through an optimisation loop that involves applications, predictable NoC architectures and performance evaluation methods (shown in Figure 1.2).

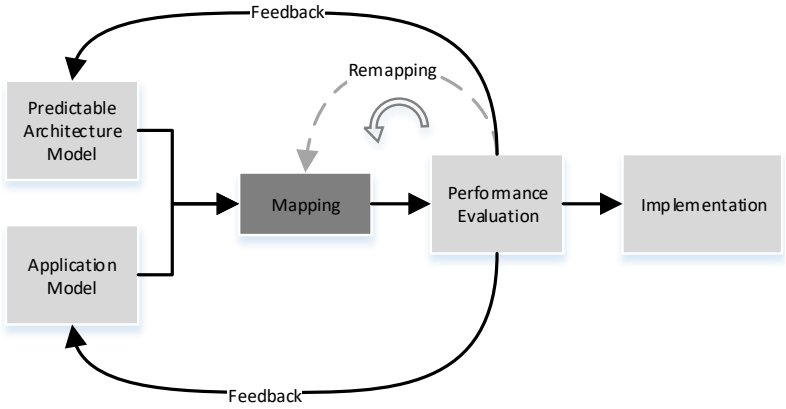

Figure 1.2: NoC Design Flow.

An optimisation loop starts with an initial attempt at task distribution. Thereafter, the performance estimation of the attempt will be compared with the design requirements. Iterative modifications will be made to the mapping, or even the application model and architecture model, until the design requirements can be satisfied.

1.3 Motivation and Goal

Through the optimisation loop, it can be seen that for a given architecture and application the real-time timing performance of a NoC can be affected by task allocation. This can be understood from two points of view.

First, task allocation can determine which IP a task can be executed on. The

order in which tasks are executed on an IP follows a priority ordering of tasks, and lower priority tasks could be disrupted by higher priority tasks. A compact task allocation will intensify the competition for occupying IPs among tasks and result in the response time for lower priority tasks on IPs increasing. Second, communication between tasks could also be affected by task allocation. This happens not only with respect to the length of the communication, but also in the competition to control the communication path on the physical layer. For example, if two related tasks are allocated at a distance from one another, communication length will be increased and cause a long communication response time. It can be seen that the original communication path in Figure 1.3a has been significantly reduced with a better mapping in Figure 1.3b. In addition, when a large number of tasks try to communicate with each other tasks using the same physical communication path (in other words, in a compact task allocation), the lower priority communications are paused by the higher ones, causing the lower priority communications to be delayed. This can be seen from the communication competition in the left end column in Figure 1.3a. Therefore, a carefully planned task allocation can reduce both the task computation time and the communication time.

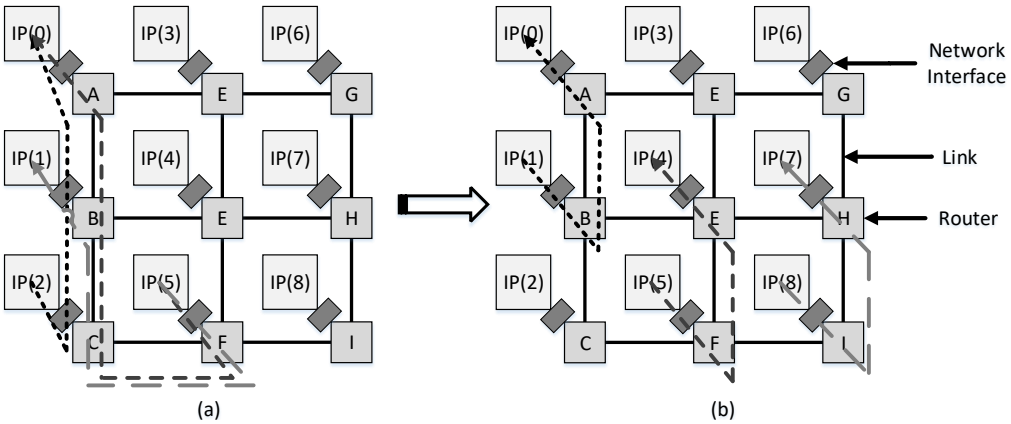

Figure 1.3: Example of Influence from Mapping: (a) compact task allocation, (b) suitable task allocation.

A NoC based real-time task mapping can be explored via either offline or online approaches. **Offline** (or static) algorithms can guarantee the system real-time

timing performance by obtaining a suitable mapping at design time, even for hard real-time systems in which there is no tolerance for timing violations in either task computation or communication. However, system flexibility is low and design exploration takes a long time. This phenomenon becomes even worse with an increase in the complexity of application or extension in architecture or both. In contrast, **online** (or dynamic) algorithms can provide high system flexibility and adaptability. However, the timing performance is not good. This can be understood as follows.

First, the focus of the state-of-the-art has primarily been on reducing the remapping overhead (the time taken to finish one remapping), with less attention paid to the system timing constrains. This is evidenced by the fact that there is no timing performance evaluation in a number of the existing algorithms (for example, Random Mapping (RM)), in soft real-time systems; a few timing violations can be allowed as long as the overall system performance requirements can be satisfied.

Second, some of the existing online methods can remap hard real-time NoCs, but they either do not take communication timing cost into account, for example, the deadline distribution strategy (reviewed in Section 2.3) or have low system flexibility and suffer a high remapping overhead at run time, or have a long design exploration time at design time. Therefore, in both static and dynamic approaches, a fast mapping algorithm that can guarantee NoCs real-time timing performance is necessary, especially for hard real-time systems.

In addition, the existing dynamic mappers are also limited by low success in the remapping rate. This can be explained by the following two points.

First, to achieve a lower remapping overhead, the existing mappers focus mainly on how to allocate a new task to a running NoC, without considering moving the tasks that are already on the NoC. For example, a task with a utilisation of 80% is waiting to be added onto a running homogeneous NoC, whose use of each IP has been illustrated in Figure 1.4a, (with just the

computation cost on IPs). It can be seen that none of the IPs can provide enough computational resources to accept the new task, if there is no task migration among the tasks already running on the NoC. However, if the tasks originally allocated on IP 2 can be moved to IP 6, a free IP (IP 2) can easily accept the new task, as shown in Figure 1.4b. Thus, a global remapping could increase the possibility of achieving a successful dynamic mapping for a real-time NoC.

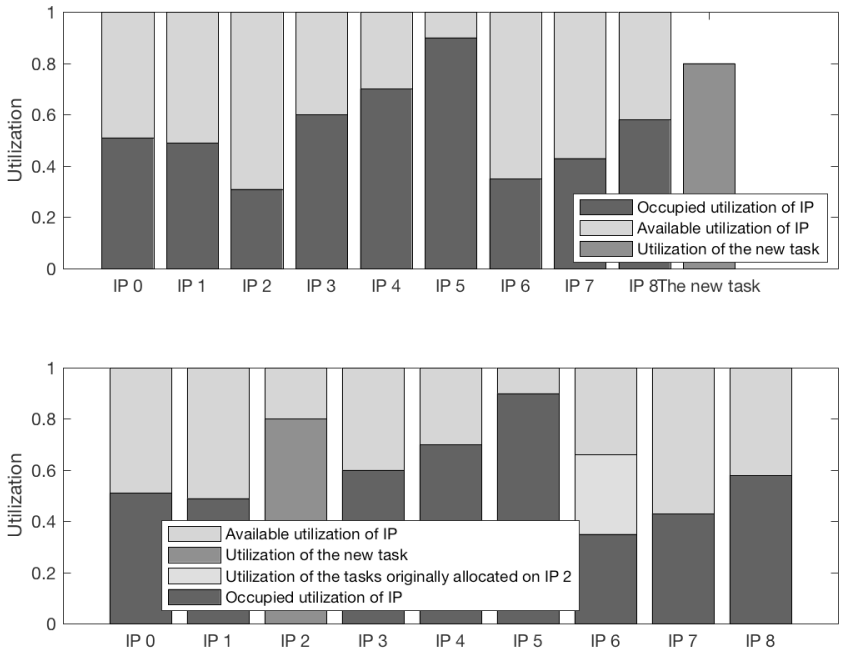

Figure 1.4: Existing Mapper vs Global Remapping Mapper

Second, research such as DSM [78] considers running task reallocation. Their mapping solutions are constructed using a fixed, predefined criterion. Although this measure could significantly reduce the search space and accelerate the remapping, it also imposes restrictions on the ability to generate good candidate task allocations and further results in a decrease in the remapping success rate.

Therefore, all of above existing limitations motivate this research to concentrate on the mapping optimisation of real-time NoCs, in particular dynamic task allocation in hard real-time systems. The research problem focused on in

this thesis is whether a schedulable task allocation can be found that is able dynamically and efficiently to meet the application's hard real-time timing requirements in an NoC based MPSoC, even in the worst-case scenario.

1.4 Novel Contributions

This thesis will achieve the above stated aims through a series of studies:

1. End-to-End Response Time Analysis (E2ERTA) can be used to evaluate whether a given task allocation can satisfy system hard real-time timing constraints on a specified priority pre-emptive arbitration NoC. However, calculation complexity is a barrier that prevents it from being applied in complex applications or large NoCs evaluation, in both static and dynamic mapping problems. The proposed Inexact E2ERTA can reduce this calculation complexity and improve evaluation efficiency.

2. Hardware accelerated Inexact E2ERTA (Inexact HW-E2ERTA) can further enhance the evaluation efficiency of E2ERTA. It can be used as a fast evaluation component to provide feedback to a given task allocation in mapping algorithms to facilitate the exploration of a large design space. It can also be applied as a fast and guaranteed deterministic admission controller to decide whether a given task can be added to a running system.

3. Parallel search is one advantage of search-based algorithms, such as Genetic Algorithm (GA). The proposed accelerated parallel GA can accelerate the search speed. It enhances the possibility for producing global task reallocation at run time on an NoC platform. In addition, the two accelerated GA operators in the proposed parallel GA can be extracted and applied in other architectures, since they are not mutually dependent on the proposed architecture.

4. The proposed dynamic mapping algorithm can be used in both static and dynamic mapping problems for fast search and evaluation task allocation for NoC-based MPSoCs. For static systems, it can facilitate optimisation over larger NoCs or more complex applications, or both. For dynamic systems, it can guarantee the system hard real-time timing requirements after remapping, while simultaneously reducing the task migration cost. It can reduce the resource cost, improve the system flexibility and enhance fault tolerance. In addition, by introducing more features, such as power, this dynamic mapping algorithm can easily be extended to consider and optimise more objectives simultaneously, in order to provide a comprehensive system optimisation.

1.5 Thesis Structure

The remaining chapters of this thesis are organised as follows:

- Chapter 2 reviews the state-of-the-art related to this research problem.
- Chapter 3 proposes a system model for this research problem and formulates the research hypothesis through an analysis of the problem. In addition, a problem breakdown is provided, which is used to guide the research direction.
- Chapter 4 analyses the factors that can affect the evaluation efficiency of End-to-End Response Time Analysis from a theoretical point of view and proposes an inexact analysis method to accelerate its evaluation efficiency.
- Chapter 5 analyses the efficiency of End-to-End Response Time Analysis from a practical point of view and suggests an implementation with inexact accelerated components to enhance its evaluation efficiency.
- Chapter 6 explores the search efficiency from the perspective of the searching algorithm and indicates which type of algorithm is most

suitable in this research. In addition, a modified architecture with accelerated components is also introduced.

- Chapter 7 combines the improvements achieved in Chapters 4, 5 and 6 to solve the research problem.
- Chapter 8 summarises the achievements of the research and identifies directions for future work.

Chapter 2

Literature Review

Dynamic mapping for hard real-time NoC based MPSoC is the topic considered in this research. This chapter will review the state-of-the-art related to this topic in the following order:

1. Section 2.1, gives an overview of NoC discussing the basic components of NoC and examples of predictable NoC architecture;
2. How the NoCs timing performance is evaluated based on these predictable architectures is reviewed in Section 2.2;
3. Recent NoC mapping algorithms are reviewed in Section 2.3;
4. Evolutionary algorithms, which can be used in NoC mapping problems, will be discussed in Section 2.4.

2.1 Network-on-Chip

The NoC architecture consists of three basic components (link, network interface and router). An example of NoC architecture with 3*3 topology

is shown in Figure 1.1. It can be seen that communication between IPs are completed by a set of routers which are linked together through physical links. The connections between IPs and routers are executed through Network Interfaces (NIs). These basic components will be reviewed in this section, followed by examples of predictable NoC architecture.

2.1.1 Links

In NoC, a link denotes the physical interconnection between two routers, and can be classified into one or more logical or physical channels, with each channel consisting of a set of wires [11]. The messages which need to be transmitted throughout the NoC are partitioned into fixed-length packets which are in turn separated into basic datagrams (or basic transfer units) called flits. A packet will be transferred in flit-by-flit style [35]. In most cases, a flit matches for a phit (physical unit refers to the minimum amount of data that can be transmitted in one link transaction). Moreover, the implementation of synchronization strategy of links can be accomplished by either a synchronization protocol, which can be implemented by dedicated wires or mixed-time FIFO [17] or globally asynchronous locally synchronous (GALS) [69] with local handshake protocols being assumed.

2.1.2 Network Interfaces

The Network Interface (NI) is set between the router and the local IP. It converts the IP views of communication to the router view. An example is shown in Figure 2.1. This conversion can be treated as a high level communication service which packetises the low level data to the high level packets used for transmission on the network at the originator end, and depacketises the packets back to data at the receiver end [11]. This function can be accomplished by various interface services, since disparate IPs may have different interface protocols. By providing these interface services, cores

can be integrated seamlessly with the NoC platform. In addition, this property also reduces the interdependence between IPs and the network, and offers abundant reusable IP blocks when doing SoC implementations.

Figure 2.1: Network Interface Example Modified from [90].

2.1.3 Routers

The main component in a NoC architecture is the router (an example architecture in a mesh-based NoC is depicted in Figure 2.2). It is the medium which connects the local port to other neighbour routers. The router is responsible for switching the correct message from its input ports to the correct output ports at the correct time according to the message routing path, with the support of routing, switching, virtual channel and flow control, which are reviewed in this subsection.

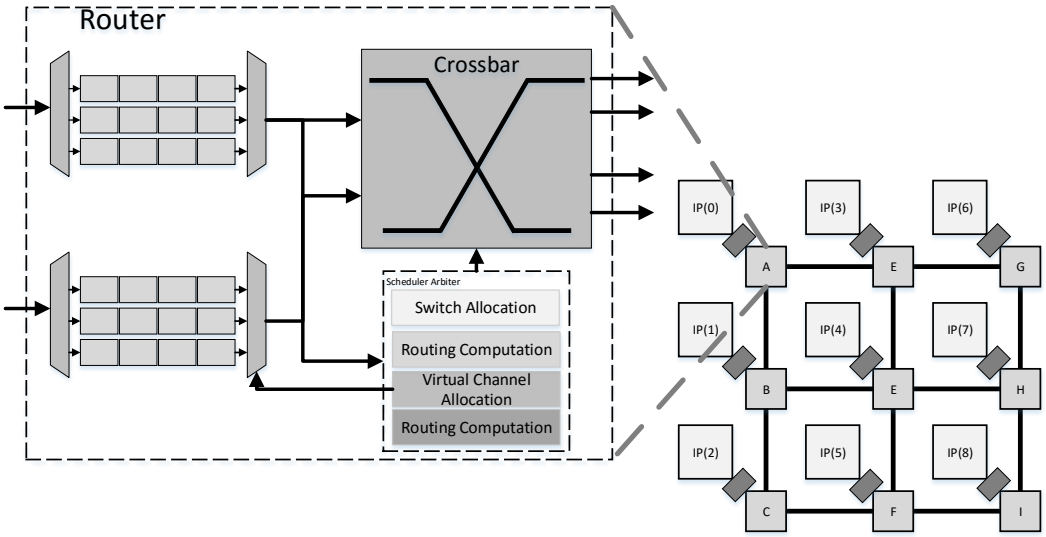

Figure 2.2: NoC Router Architecture.

Routing

Routing determines the path for each packet, from its initiator to its destination. It decides which output port channel/s the packet upon arrived will be forwarded to, within each intermediate router. Its algorithms can be classified using several criteria, such as where/when and adaptivity.

Where/When

Where/When refers to where or when a routing decision is made. On this basis, routing algorithms can be categorised into centralised routing, source routing, and distributed routing. Centralised routing can provide a better routing path, since the information considered includes not only the address of source and destination, but also the working situation of the current system. However, considerable computation time and power consumption will be required by introducing one extra control component. In contrast, in distributed routing, the routing decision is determined at each router, which only knows its neighbourhood as packets travel across the network. A header containing only the destination address is used to select output channel/s. Source routing algorithms will predetermine complete routing paths as a header on source nodes before injecting packets into the network. The switches of routers along a path will be configured accordingly by the header.

Adaptivity

Adaptivity of routing algorithms refers to whether information other than the address will be considered during the routing decision making. Routing algorithms can be classified into deterministic routing and adaptive routing. Deterministic routing is also known as static routing, because the same routing path will always be generated for a given pair of source and destination address. In source routing, a unique path will be produced without considering any system traffic situation. In distributed routing, a unique configuration will be produced in each intermediate router. Taking XY routing as an example, a

packet will first travel along the X axis (the row) until it reaches the node which is the perpendicular crosspoint of the source row and the destination column. It will then move forward to the column until it arrives at the receiver node. Deterministic routing is currently widely used on NoCs, because it is simple, fast, and easy to analyse. By contrast, although adaptive routing can provide flexible routing decision based on system working situation, it also has some disadvantages, for example, it is resource hungry, there are difficulties in implementation and analysis and it is slow in making decisions.

Switching

The switching strategy determines how a path will be built for packet propagation. One of its taxonomies can be seen in Figure 2.3.

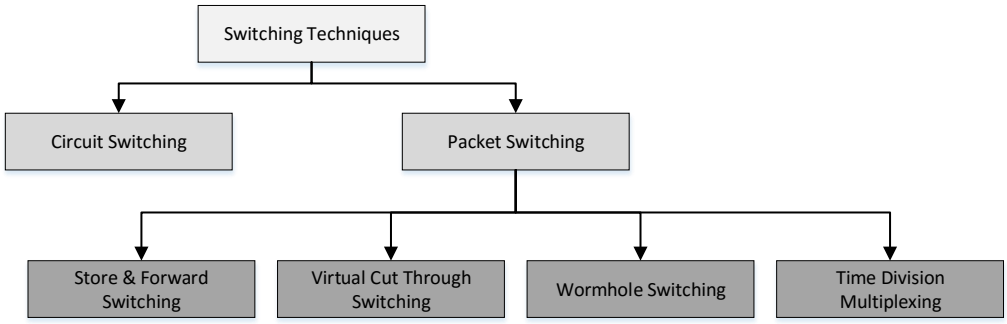

Figure 2.3: Taxonomy of Switching Techniques.

Circuit switching provides an end-to-end path which is reserved on each intermediate router prior to the data transition by injecting a routing probe and released by the destination or the last bits of data. Messages of any length can be propagated to the destination without interruption, after an acknowledged flit has been returned to sender. To enhance this technique, [62] advocated a new circuit switching mechanism with separated control and data transmission network, in order to reduce the average latency of circuit establishment.

Packet switching, however, does not reserve the entire channel. Packet switch-

ing can be classified as Store & Forward (SF), Virtual Cut Through (VCT) and Wormhole Switching (WH). **SF** [35], which is suitable for integrally transferring short and frequent packets, requires a buffer from both input and output to store an entire packet. Routing decisions are made on each intermediate router, as long as the whole packet has been buffered completely. The header flit cannot be forwarded to the next router if either the routing decision has not been made or the available buffer space in the downstream router is insufficient to store an entire packet. By contrast, instead of waiting for the whole packet, **VCT** allows the header flit to cut-through into the following router as soon as the routing decision has been made, and the remaining flits follow the same output channel as their predecessors to the destination. Transmission of different packets cannot be interleaved or multiplexed over one physical channel. It will store the entire packet on an intermediate router buffer and behave the same as SF if the next router is occupied. In **WH**, a header will be used to build a path for the following flits (belonging to the same packet) to snake with it to their destination in pipeline style, possibly spanning a number of routers. If the header cannot proceed, the wormhole chain will be stalled, occupying flit buffers in each router on the path constructed so far and possibly blocking other communications, or even creating a chain-blocking. This could result in a packet experiencing multi-blocking during its journey and cause a difficulty in analysing the timing behaviour [110]. WH offers low network latency and buffer cost. However, its level of congestion is high and very deadlock-prone without special measures such as Virtual Channel, which will be reviewed next.

Time Division Multiplexing (TDM) could be treated as an alternative switching method to pure circuit-switching with higher resource utilization [79]. TDM allocates the resources according to timetables which consist of a given number of time slots. Each slot is reserved for a special connection. The tables in all routers are synchronized by a global TDM schedule to guarantee virtual circuits for connection free scheme. Thus, some in-router components such as arbitration, and flow control can be removed. A summary of the comparison of these techniques is given in Table 2.1.

Table 2.1: Switching Technique Comparison.

Switching	Communication Entity	Path Reservation	Buffer Size	Resource Utilization	Comments
Circuit Switching	Flit	Yes	Small	Low	Requires setup, Acknowledgement and path tear down phases
Store & Forward	Packet	No	Large	High	Header must wait for entire packet before processing to next roter
Virtual Cut Through	Packet	No	Large	High	Header can be forwarded to next router before tail arrives at current node
Wormhole	Flit	Yes	Small	Moderate	Header blocking reduces efficiency of link bandwidth

Virtual Channel

A Virtual Channel (VC) [29] is used to enhance the network throughput by applying a number of shallow buffers to decouple the network resources, which substitute for the implementation of a single deep buffer at input/output ports. It is able to produce enhancements of between 20% and 50% [28]. An example is shown in Figure 2.4. On NoCs without VC, a packet has to be stalled and stored in local buffers if the target router it is trying to access is already occupied by other packets. In addition, this phenomenon can be worse if an NoC is under heavy traffic flow, especially in wormhole switching based NoCs, because of the block chain. In contrast, by introducing a VC the other unblocked packets held in the VC can virtually bypass the blocked one and access the next router. Which packet gets to use the physical channels is decided by a priority competition according to the arbitration policy. That is why a higher priority packet would take precedence over a lower priority packet in a priority pre-emptive arbitration NoC. The higher network throughput and use of physical channel bandwidth are the advantages provided by a VC. In addition, combining a VC with wormhole switching offers several benefits, such as being deadlock free, making more efficient use of network channels and supporting different service levels [12]. Although the switching complexity remains moderate, this combination has became prevalent and is advocated by several NoC architectures [64], [94].

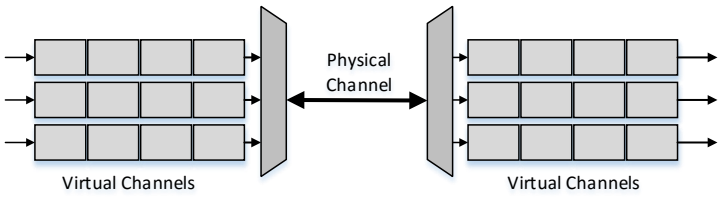

Figure 2.4: Virtual Channel Example.

Flow Control

Flow Control (FC) refers to the allocation of network resources to a packet traversing NoCs, such as buffer capacity, control state and channel bandwidth. A taxonomy of current popular FC techniques is shown in Figure 2.5. Bufferless flow control is mainly used for circuit switched networks, and provides a dedicated end-to-end transmission path. However, buffered flow control focuses on packet switched networks.

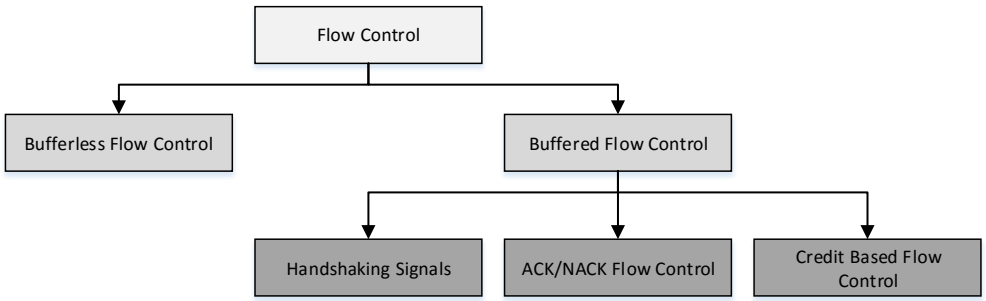

Figure 2.5: Flow Control Taxonomy [1].

Handshake is accomplished by using a valid signal, which is sent whenever a sender transmits any flit, and an acknowledge (ACK) signal which is returned by the receiver when the validation has been successfully acquired. Low cost implementation and low link utilisation are its advantage and disadvantage. Examples of NoCs using handshake as flow control can be seen in [127] and [105]. **ACK/NACK** requires to copy a data flit in current router until an ACK signal has been received. Otherwise, if a NACK signal has been detected, the flit will be retransmitted. An example can be seen in [7]. **Credit based Flow Control** requires the upstream router to keep counting the available

space in the downstream router as credits which will be decreased if a flit is sent and increased if the flit is accepted by the receiver. The integrity of the packet can be guaranteed. Examples of NoCs which are based on this technique include Ætheral [46], [89], [34], SPIN [49], QNoC [13] and [42].

2.1.4 Predictable NoC Architecture

By applying the techniques reviewed in previous subsections, a predictable NoC architecture, which is the basis for predicting or evaluating NoC performance with a given application, can be provided. According to its customisation and parametrisation capabilities, the NoC can generally be classified into heterogeneous architecture and homogeneous architecture. Heterogeneous NoCs can provide more efficient design compared with homogeneous ones in terms of area, power and timing performance, since its architectures can be customised following any application requirements. An example can be seen in XPIPES [7] and [8]. However, these advantages also make these kind of NoCs as application oriented architectures. Homogeneous NoCs, in contrast, can reduce the development time, by using a generic architecture. Their instantiation space is depicted in Figure 2.6, according to their customisability and granularity, which refers to the level at which the NoC or NoC components are described.

According to the switching technique applied, predictable NoC architectures can be classified in to circuit switching, packet switching with priorities (PSwPri), TDM and hybrid. SoCBUS [122] is the first circuit-switched NoC. Its hard real-time system requirements can be fitted by applying a pre-runtime static scheduling phase. However, its inefficient use of resources, the cost of non-scalability and the delay for setting pathes mean that neither it and its improved versions such as [75] are not selected as the main approach.

The techniques applied by most existing real-time NoC architecture are dominated by two branches, PSwPri and TDM. PSwPri NoCs such as [104], [13] and [112] allow some contentions happen in both computation and

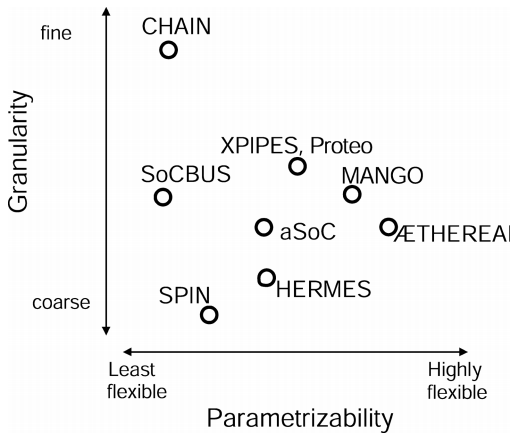

Figure 2.6: NoC Instantiation Space [11].

communication, but using various priority-arbitration policies and schedule strategies to ensure some tasks to be served first (normally higher priority ones), at the same time taking care of the others (normally lower priority ones) and guarantee all tasks are finished within their deadlines. Thus, they are also known as connectionless NoCs. The high throughput and low buffering requirements, compared with circuit switching NoCs, are provided by applying wormhole-switching with credit-based flow-control and virtual channels. Meanwhile, some real-time evaluation methods such as [58] also increase the confidence of researchers in using this kind of NoCs as the solutions.

By contrast, all contentions are avoided by applying resource reservation in TDM switched NoCs. Once packets are injected into the network, they will be transfered through dedicated channels which are reserved by their associated time slots to their destination, without any interrupt. Although, TDM NoCs are challenged by the complex time slot generation process and global synchronization among all TDM slot tables, they still attract considerable attention from both researches and engineers as their simple and efficiency routers. Æthereal [46] and Nostrum [82] are two pioneering NoC examples based on TDM. In addition, there are improved versions based on TDM such as Argo [39] and dÆlite [88].

The last category is hybrid NoCs which focus on improving the resources utilization by combing TDM and PSwPri. A typical example can be seen in MANGO NoC [12], in which the GS router and BE router are integrated. The BE router supports connectionless communication by applying the packet switching technique, while the GS router is used to enhance the connection free communication. A comparable architecture can be seen in [50] using a dynamic time slots reservation technique.

Among the existing real-time architectures, it could be seen that homogeneous NoC should be selected for general application research. Moreover, the TDM and PSwPri NoCs are two main solution branches which should be considered. Æthereal NoC and priority pre-emptive arbitration based NoC are two pioneers in these two branches. Although some successes have been reported in higher performance by adjusting some techniques [39], [88], these two NoCs still can be used as the typical examples to represent these two branches.

Ætheral NoC

The Ætheral NoC [46] applies the techniques of pipelined Time-Division-Multiplexed circuit switching (TDM) and packet switching techniques to enhance the system Quality of Service (QoS), for example, hard real-time and soft real-time, which are supported by Guaranteed Services (GS) and Best-Effort services (BE) respectively. The architecture of Ætheral NoC can be categorised into router and NI, with multiple links between them.

Router

The router in Æthereal NoC takes the responsibility of contention-free routing and transferring flits to their destinations by placing two routers (a contention-free GS router and a BE router) in parallel to support GS and BE services separately. The review of Ætheral router will start with contention-free routing, and followed by the router architectures of each individual service.

Contention-Free Routing is accomplished by using a slot table mechanism which uses a slot table stored in each router to configure the cross bar at each time slot. An example is shown in Figure 2.7. The column of the slot table represents the configuration of each output and the row indicates the configuration at each time slot. The data flits can be transferred to their destination with SF switching in pipeline style. This mechanism requires a global synchronisation of the whole system, which can be implemented in two ways. One way is using a combination of a single, centralised synchronous clock line and various techniques such as waterfall clock distribution and synchronous latency insensitive design. The other is a distributed approach with a Synchronous-Data-Flow (SDF) model. The NoC can be synchronised by forcing each router to synchronise with its neighbours, although, this will result in the whole system only running as fast as its slowest router. This mechanism can entirely avoid network contention and guarantee the performance of GS service. However, the large slot table is resource hungry and its size will be increased along with the complexity of applications.

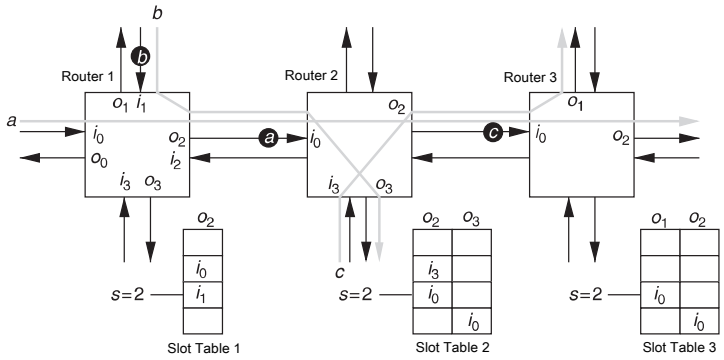

Figure 2.7: Æthereal Contention-Free Routing modified from [46].

GS and BE Router GS and BE routers in Ætheral are in charge of data channelling by using the slot table to configure their cross bar. **GS router** is relatively simple compared with BE router. Its architecture consists of a set of queues used to temporarily store data flits, simple connections between queues, and a reconfigurable switch. The flow control for GS router can be removed, since GS block will be served at the next clock cycle [46]. The **BE router**,

in contrast, cannot guarantee the best-effort block served at the next clock cycle. A credit based flow control mechanism should be introduced to ensure that no flit is transferred beyond the receiver's capacity. In addition, it also introduces wormhole switching, source routing and round-robin arbitration, since best-effort services have equal priority.

Network Interface

The NI of Ætheral NoC can generally be divided into NI shell and NI kernel. The **NI shell** is responsible for offering various communication protocols such as AXI and OCP to fill the gap between NoCs and IPs [89]. **NI kernel** communicates with the NI shell through point-to-point ports which are also buffered with FIFO (First-In-First-Out) for supporting clock domain crossing. It packetises the packet received from NI shell and schedules them to the router according to priority and packet type (GS packet can cut-through directly, BE packet will be scheduled by round-robin arbitration).

Dynamic Reconfiguration on Ætheral NoC

The dynamic reconfiguration of Ætheral NoC refers to BE services, since GS services are predefined at design time [46]. It can be achieved by both centralised and distributed models. The configuration of both is accomplished by reconfiguring the slot tables in each router. The centralised model can modify the slot tables through GS packets along their route to the destination, which guarantees the reconfiguration. However, the distributed model uses BE packets (setup, teardown and acksetup packets) to program the resources along the routing path with a source routing strategy. An acksetup packet will be returned to sender when the setup packet has arrives at the destination. Otherwise, a teardown packet will be received and the pre-build path will be released. This reconfiguration is not guaranteed and is highly depended on the traffic load on the NoC.

Priority Pre-Emptive Arbitration Based NoC

The priority pre-emptive arbitration based NoC is widely researched in academic contexts. Its architecture (shown in Figure 2.8) is based on mesh topology NoCs and offers bidirectional links with uniform bandwidth between two routers. In this NoC, priority based wormhole switching is adopted with the XY routing protocol. In addition, credit based flow control technique is also introduced to ensure no more flits are transferred than the receiver can accept. Furthermore, in order to overcome the deadlock problem of wormhole switching, a virtual channel mechanism is added and implemented. When a communication flow arrives at a router, it will be served according to its priority level, which is inherited from the task that initiates the flow, and the flows that have been received so far by the router. The path which the observed flow should be forwarded to is determined by XY routing instead of being predefined in Ætheral NoC. Although its router area is big, because of its complex scheduler, its behaviour is much more flexible than Ætheral NoC.

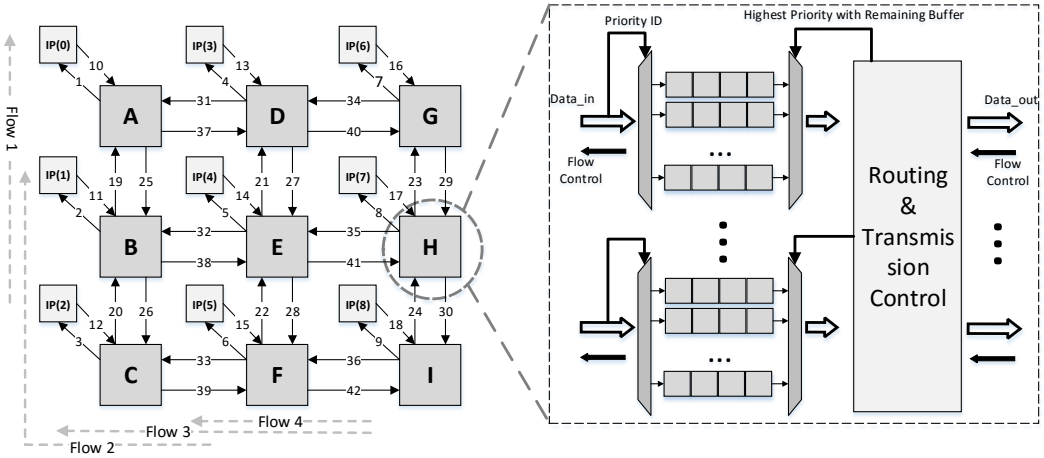

Figure 2.8: Priority Pre-Emptive Arbitration Based NoC Architecture.

*The numbers on black arrow are the index of links.

Comparison

The Æthereal NoC and the Priority Pre-Emptive Arbitration Based NoC are pioneers of two most popular solution branches (TDM and packet switching) respectively. Æthereal can reduce the area of routers, simplify the scheduler and provide strong predictability for guaranteed services. However, it increases the implementation complexity of NI. Guaranteed services may be over-reserved and lead to lower resource use. Although the BE services are introduced to improve the average performance, the worst-case performance is difficult to estimate. In addition, the dynamic reconfiguration speed is low, since all slot tables need to be modified, and the reconfiguration cannot be guaranteed by applying a distributed model. This may impose costs of both resources and time. By contrast, the Priority Pre-Emptive Arbitration Based NoC can provide guaranteed timing performance with a careful task allocation and system evaluation. Although, its router is big because of its complex scheduler, its dynamic reconfiguration is simple and fast, as it can be simply finished by moving tasks to other IPs, rather than requiring the configuring of slot tables. A table of comparisons is listed in Table 2.2 to show the difference between them, and this is used in the section 3.1.2 of the following chapter for NoC platform selection.

Table 2.2: Æthereal vs Priority Pre-Emptive Arbitration Based NoC.

NoC Architecture	Ætheral NoC	Priority Pre-Emptive Arbitration Based NoC
Router area	small	large
NI area	large	small
Routing	slot table, contention free	deadlock free, XY routing
Comfigure speed	low	high
Configure cost	cost	low
Felxiablity	low	high
Timing guarantee	high	high
Switching	SF for GS, Wormhole BE	Wormhole
Complex applications	larger slot table	no change

2.2 NoC Performance Evaluation

Exploring or evaluating the performance of an NoC implementation is an indispensable step in achieving the design goal, which can be either minimum level of performance with lowest cost or highest performance with a given cost [65]. The tools used in performance evaluation can be categorised as simulation models and analytical models.

Simulation models provide flexible and accurate methods for researchers to tackle performance estimation. However, when the targets are complex MPSoCs, what can reasonably be simulated, and how to select the hardware for the NoC or how to program it according to the simulation results, is restricted [66]. Moreover, the difficulty of predicting a finite set of test scenarios and the slow simulation speed with high computation costs are also barriers to the application of fast evaluation, especially in dynamic optimisation problem.

However, formal mathematical analytical models are popular since they can provide not only a fast performance analysis of the worst-case scenario at an early design phase, but also be invoked as feedback in any NoC optimisation process. Therefore, an analytical model, such as Synchronous Data Flow Analysis (SDF) [71], End-to-End Response Time Analysis (E2ERTA) [100] and Queueing Theory (QT) [67], could be suitable for this research, which explores dynamic mapping problems. SDF and E2ERTA will be reviewed in this section, as examples to support the two predictable architectures reviewed in Section 2.1.4.

2.2.1 Synchronous Data Flow Analysis

Synchronous Data Flow (SDF) is a special case of data flow [71]. It restricts the general data flow graph in order to test efficiently whether or not a finite static schedule exists in a given set of nodes, and if so, to find it. The

assumption made in SDF is that the number of tokens consumed or produced at each node is fixed and known in advance [71]. A synchronous data flow (SDF) graph is illustrated in Figure 2.9. The number on each arrow (not in square) describes the number of tokens consumed or produced at each node. Any node can fire (execute) whenever the input data are available on its incoming ports. A node which has no incoming arrows can fire at any time. As a result, many nodes can fire concurrently.

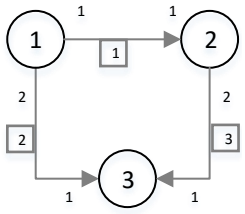

Figure 2.9: A Synchronous Data Flow Graph [71].

The process of finding a finite static schedule for a given graph can be divided into the necessary test and sufficient test. To find the schedule for the example in Figure 2.9, the graph can be abstracted as a topology matrix as shown in Equation 2.1 by numbering each node and arrow (the index number is the box) and setting a row to each arrow and a column to each node. For example, the left column in Equation 2.1 is [1 2 0] representing the relationship in producing or consuming tokens between node 1 and the three arrows. Node 1 produces one and two tokens to arrow 1 and 2 respectively, but is not related with arrow 3. After abstracting the topology matrix, the necessary test can be used and may further lead to the sufficient test, if the necessary condition (the rank of this matrix should be equal to the number of nodes minus one) has been satisfied. Otherwise, a fine static scheduling does not exist.

$$\Gamma = rank(\begin{bmatrix} 1 & -1 & 0 \\ 2 & 0 & -1 \\ 0 & 2 & -1 \end{bmatrix}) = number\ of\ nodes - 1 \tag{2.1}$$

The sufficient test can be undertaken by Equation 2.2, where J is any positive

integer and q describes the number of times each node should be invoked in one cycle of a periodic schedule. Each row of the q refers to a node.

$$\Gamma q = 0 \Longrightarrow \begin{bmatrix} 1 & -1 & 0 \\ 2 & 0 & -1 \\ 0 & 2 & -1 \end{bmatrix} q = 0 \Longrightarrow q = J \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \tag{2.2}$$

2.2.2 End-to-End Response Time Analysis

End-to-End Response Time Analysis (E2ERTA) [58] is a formal mathematical evaluation model used to explore the timing performance of a hard real-time system on priority pre-emptive arbitration based NoC (an example is shown in Figure 2.8). Its results can indicate the value of the end-to-end response time of a task which starts at the time point the task is released on the initial IP, and lasts until the last flit of the packet (the task generated) is received by the destination IP under the worst-case scenario. Thus, it can be affected by the response time of both task computation and flow communication. This phenomenon can be seen in Figure 2.10 which follows the example of Figure 2.8 and considers the deadlines of all tasks as being same and equal to period.

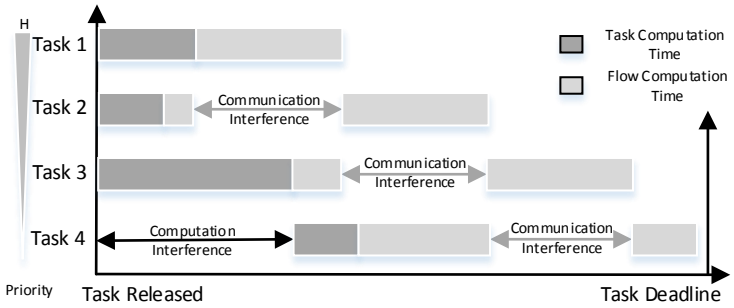

Figure 2.10: E2ERTA Example.

On each IP, the tasks are released by priority order. Hence, higher priority tasks can easily preempt lower priority tasks. $Task_3$ and $Task_4$ are released at the same time on IP(8). However, as $Task_3$ has higher priority than $Task_4$, it can directly take the node and preempt the release of $Task_4$. Similarly,

communication interference also exists, since it is undertaken by physical channels (such as routers and links) which are shared by multiple flows. It can be seen from that $Flow_2$ (initiated by $Task_2$) is directly interrupted by $Flow_1$ (initiated by $Task_1$). Therefore, to calculate the E2ERTA, the response time of both tasks and flows need to be computed.

Response Time Analysis for Tasks

In order to know whether a given task set can be scheduled on a single IP, Liu and Layland [74] define the Rate Monotonic approach with static priority pre-emptive scheduler. Audsley et al. [4] then extended the response time analysis to include release jitter. Using response time analysis, the times that the higher priority tasks will be released can be calculated, during the response time of the lower priority task under the worst-case situation. At the same time, the exact worst-case response time of the lower priority tasks can also be obtained. The result can be calculated by Equation 2.3, where r_i, c_i, B_i, t_i and $hp(i)$ represent the response time, worst-case computation time, blocking time, period of $task_i$ and the set of tasks with higher priority than $task_i$ respectively. The calculation can be terminated by either $r_i > d_i$ (d_i is the deadline of $task_i$) or $r_i^{n+1} = r_i^n$ (the response time of $task_i$ is not increased and the $r_i^0 = c_i$).

$$r_i^{n+1} = c_i + B_i + \sum_{\forall j \in hp(i)} \lceil \frac{r_i^n}{t_j} \rceil c_j \tag{2.3}$$

Response Time Analysis for Flows

In [104], the authors models the links and flows as shared processors and tasks respectively and extended the response time analysis to obtain the communication delay of each flow. Different from the relationship among tasks on a single IP, the relationship among flows is complex. A lower priority

flow ($Flow_i$) may suffer both direct and indirect interference from higher priority flows which are from its direct interference set (S_id) and indirect interference set (S_ii) respectively (formal defination can be seen in Chapter 3). The flows in these two sets can affect the worst-case response time of $Flow_i$ by pausing the communication of $Flow_i$. Their definitions are based on the relationship between $Flow_i$ and higher priority flows and are listed as follow:

- The flows in the direct interference set:
  - having higher priority than $Flow_i$;
  - sharing at least one link with $Flow_i$.
- The flows in the indirect interference set:
  - having higher priority than $Flow_i$;
  - having no shared link with $Flow_i$;
  - interfering with the flows in the direct interference set of $Flow_i$.

Figure 2.8 shows an example of a traffic flows relationship, where the priority of flows follows the increase of the index number of flows, a lower number refers to a higher priority. In this example, the $Task_3$ *and* $Task_4$ are allocated on IP(8); $Task_2$ and $Task_1$ are allocated on IP(5) and IP(2) respectively. The direct interference set and indirect interference set for each $Flow_i$ are listed in Table 2.3.

Table 2.3: Traffic Flow Example

$Flow_i$	Direct interference set	Indirect interference set
$Flow_1$	$\{\phi\}$	$\{\phi\}$
$Flow_2$	$\{Flow_1\}$	$\{\phi\}$
$Flow_3$	$\{Flow_2\}$	$\{Flow_1\}$
$Flow_4$	$\{Flow_3\}$	$\{Flow_2\}$

*The flows follow the examples in Figure 2.8.

By considering the relationships among flows, the communication performance of an NoC can be explored. The response time of flows can be calculated using Equations 2.4a and 2.4b, where R_i, C_i and T_i are used to represent the response time, basic latency, and period of $Flow_i$; J_j^R and J_j^I indicate the release jitter and interference jitter of $Flow_j$; S_id shows the direct interference set of $Flow_i$.

$$R_i^{n+1} = C_i + \sum_{\forall j \in S_{id}} \lceil \frac{R_i^n + J_j^R + J_j^I}{T_j} \rceil C_j \tag{2.4a}$$

$$J_j^I = R_j - C_j \tag{2.4b}$$

$$R_i^0 = C_i \tag{2.4c}$$

However, this is only the response time of the communication part. The computation part is not included in Equation 2.4a. Therefore, [58] assumes that the release jitter of a traffic flow can be replaced by the worst-case response time of the initial task of the flow (that is, $J_i^R = r_i$) and rewrote the Equation 2.4a to Equation 2.5 to provide the end-to-end response time for a flow, where the equation determins when either $R_i^{n+1} = R_i^n$ or $R_i^{n+1} > D_i$ (deadline of $Flow_i$).

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

Accelerated Methods for Response Time Analysis

From Equations 2.3, 2.4a and 2.5, it can be seen that the calculation of response time analysis is based on an iterative calculation. Since this iterative calculation needs an indefinite number of iterations to compute the final results, the response time analysis is inefficient. In addition, it will also be affected by increases in the complexity of applications and the size of NoC. Therefore, it is worth consideriong the efficiency improvement of response time analysis.

Bini and Baruah [10] present an upper bound estimation mechanism by analysing the workload to find the upper bound of the task response time, in order to avoid the need for exact result computation. This is shown in Equation 2.6, where r_i^{up} is response time up bound of $Task_i$ and $hp(i)$ is the set of tasks with higher priority than $Task_i$. However, it can only be used as a sufficient test, since it cannot guarantee the final result.

$$r_i^{ub} = \frac{c_i + \sum_{\forall j \in hp(i)} c_j(1 - u_j)}{1 - \sum_{\forall j \in hp(i)} u_j} \tag{2.6}$$

[31], explores this problem from a different view, pointing out a lower bound of response time of a task. This lower bound can be found by using Equations 2.7a, 2.7b and 2.7c. The $I_j(r_{i-1})$ denotes the worst-case interference due to $Task_j \in hp(i)$ occurring during the response time of $Task_{i-1}$, B_i and u_i are the maximum block time and utilisation $(\frac{c_i}{t_i})$ of $Task_i$, J_i and r_i^{lb} indicate the release jitter and lower bound of response time of $Task_i$ respectively. Their results suggest that by applying this technique, the number of iterations needed when executing the worst-case response time analysis can be reduced up to 33.3% (average number of celling operations). Although these two techniques are primarily proposed to improve tasks response time analysis on single IP, they could still be applied in the response time analysis of flows on NoCs, since flows analysis is inherited from tasks analysis. Moreover, they

could also benefit the E2ERTA.

$$I_j(r_{(i-1)}) = \left\lceil \frac{r_{(i-1)} + J_j}{t_j} \right\rceil c_j \tag{2.7a}$$

$$\begin{aligned} r_i^{lb}(k) = & \frac{B_i + c_i + \sum_{\forall j \in lep(k) \cap hp(i)} I_j(r_{i-1})}{1 - \sum_{\forall j \in hp(k)} u_j} \\ & + \frac{\sum_{\forall j \in hp(k)} J_j u_j}{1 - \sum_{\forall j \in hp(k)} u_j} \end{aligned} \tag{2.7b}$$

$$r_i^{lb} = \max_{\forall k=1 \ldots i} r_i^{lb}(k) \tag{2.7c}$$

2.2.3 Summary

In this section, NoC performance evaluation methods were reviewed. From the perspective of running time system state estimation and simulation speed, simulation models are difficult and slow. This results in simulation models being unsuitable for fast performance evaluation, especially for dynamic optimisation problems. Two examples of analytical models were also reviewed. End-to-End Response Time Analysis (E2ERTA) may be able to undertake fast optimisations, although the existing E2ERTA cannot be used as a solution directly, since it may cause low evaluation efficiency. However, it has the potential to demonstrate a suitable performance and can be used as a feedback function in a fast optimisation loop with modification.

2.3 Application Mapping Algorithm

For a given NoC architecture and application, the performance of a NoC can be improved by an optimised task allocation, since different mapping can directly affect not only the task computation time on IPs, but also the flows communication time with various level of network congestion. The state-of-the-art mapping methods can generally be classified into static mapping and dynamic mapping, according to when a mapping decision is made. A static/offline mapping is predefined at design time. It is only used for deployment at the beginning and remains thereafter. Normally, it has enough time and resources to process design space exploration and thus the best performance with a given resource can be obtained. To achieve this, it requires the system information (application and working environment) can be fully known at design time and guaranteed not to change at run time. Dynamic/online mapping, in contrast, distributes tasks to NoC along with the application execution. Since a system working situation is considered, dynamic mapping can provide a better solution if an NoC working environment has dynamic behaviours (such as battery management, fault tolerance and user behaviours). Thus, at design time the information required about the application or working environment is less than that for static mapping, but a remapping overhead (time used to finish one remapping) will be involved whenever the existing task allocation is changed. Modified from the taxonomy in [92] and [107], the classification of each type of mapping algorithm can be further divided into several sub-categories as shown in Figure 2.11. This section will start with a review of static mapping and then move to dynamic mapping.

2.3.1 Static Mapping Strategy

The static mapping strategy can broadly be divided into exact mapping and search-based mapping, according to how a mapping solution is generated.

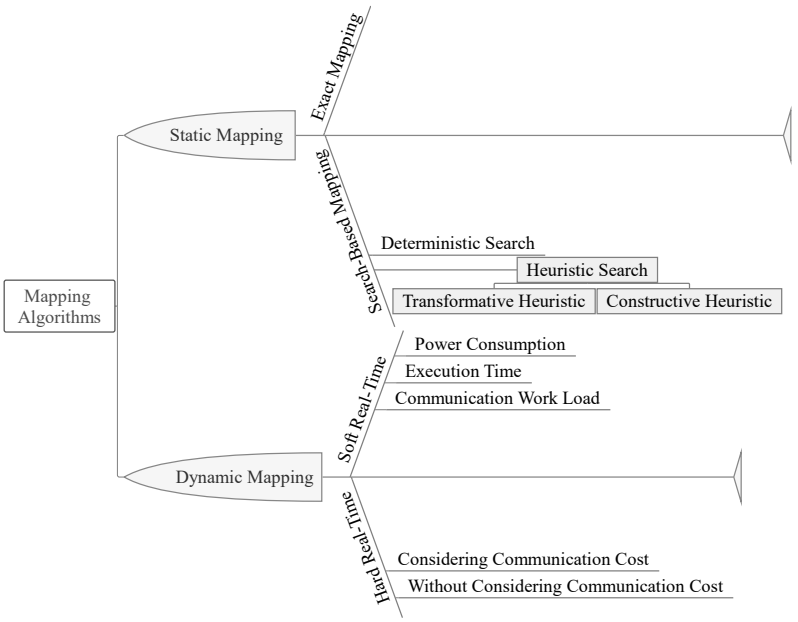

Figure 2.11: Mapping Taxonomy modified from [92] and [107].

Exact Mapping Algorithm

The algorithms in the exact mapping category adopt mathematical programming, for example Integer Linear Programming to optimise the performance of NoC in respect of factors like execution of processors, communication cost [5], architecture [83], Symmetric Multi-Processing (SMP) [85], power [111], [86], and contention [20]. Although the optimal solution can be provided after these optimisations, the complex calculation and the long computation time in mathematical programming are the main drawbacks. In addition, these become worse with increases in the complexity of application and the size of NoC.

Search-Based Mapping Algorithm

Following the taxonomy in Figure 2.11, the search-based mapping algorithms can be further sub-categorised into deterministic search and heuristic search.

Deterministic Search

The deterministic search, which mainly use the Branch-and-Branch (BB) approach, is an exhaustive search. It can systematically explore the search area with a tree topology and find a mapping solution by searching each tree branch while at the same time bounding inadmissible solutions [92]. [56] and [57] optimised the energy performance of a tile-based regular NoC through BB search, but some hotspots are generated. To overcome hotspots, [73] introduced the traffic balanced IP mapping (TBMAP) criterion; however, this results in some irregular routers, various network interfaces (such as single-router to single-IP, single-router to multiple-IP, double-router to single-IP) and some long data paths. To some extent, the deterministic mapping algorithm can find the suitable solution, but the high memory depth demand and long computation time are the main drawbacks. The search time will increase along with increases in the search space. This limits the BB searching algorithm application to small problems, since more complex problems mean larger design area.

Heuristic Search

According to whether an optimisation starts from existing mapping/s, the heuristic search can be divided into transformative heuristics and constructive heuristics.

Transformative Heuristics obtain better solution from improving existing mapping/s, by applying search-based algorithms. [124] compares various search-based algorithms and finds that Evolutionary Algorithms (EA)s show the better performance than others in solving search problems. Therefore, here we mainly review the research using EAs such as Genetic Algorithm (GA), Particle Swarm Optimisation (PSO), Ant Colony Optimisation (ACO) and so on, for mapping searching of NoCs.

GA based transformative heuristics

GA is a search-based algorithm. It can be used to explore the near-optimal or good-enough solution for complex optimisation problems which have a

large searching space and cannot adequately be addressed by mathematical analysis or exhaustive search. Details will be reviewed in Section 2.4.

The work in [72] (two-step optimisation) and [128] tries to minimise the overall system delay on homogeneous NoC architectures by using single objective GA. An architecture aware analytic mapping algorithm (A3MAP) in [60] can be used for both homogeneous and heterogeneous cores on regular and irregular mesh or custom based NoCs to reduce the amount of traffic. The task graph and NoC architecture topology are abstracted as two characteristic matrices as inputs for Mixed Integer Quadratic Programming (MIQP) to generate the initial population of the following GA optimisation loop. In addition, [24] and [25] are also proposed for customised NoC in power consumption optimisation.

Apart from single objective GA, Multi-Objective GAs (MOGAs), such as NSGA-ii (an improved version of Non-Dominating Sorting Algorithm) [32], which is a fast and elitist multi-objective GA using pareto optimal as selection criteria to generate offspring population, are also adopted in NoC mapping problems. The research in [61] which is a two-step optimisation with NSGA-ii is used to minimise the energy consumption for both computational and communicational areas, while also reducing the maximum link bandwidth. After evolving a task graph to core communication graph (CCG), the elitism set is used as the initial population of the second step optimisation for mapping CCGs to NoC. [9] also focuses on reducing the energy and bandwidth by using NSGA-ii. In addition to these methods, researchers also combine the GA with biology conspectus such as Multi-objective Adaptive Immune Algorithm (MAIA) [99], to optimise the system energy.

PSO and ACO based transformative heuristics

Particle Swarm Optimisation (PSO) [87] is a population based stochastic search technique which is inspired by simplified social behaviours such as bird flocking and fish schooling. The candidate solutions in PSO could be affected by the experience of other candidates. The local best and global best solution can guide the evolution of the next generation. Work using PSO to improve

the NoC performance can be seen in [120] and [118], which are both two-step algorithms focusing on power consumption and latency respectively. In [6], a hybrid multi-objective PSO with pareto selection strategy is presented to enhance both the execution time and energy. The comparison of results in [6] show that the genetic approach is better than the PSO approach in terms of efficiency and accuracy.

Ant Colony Optimisation (ACO) [26] is inspired by the biological behaviour of ants searching for a path between their colony and the source of food. The results shown in [117] suggest that ACO is better than a random mapping in NoC optimisation. The main drawback of ACO is the uncertain time to convergence, but theoretical analysis for ACO is also difficult [97].

Constructive Heuristics provide a mapping solution through step-by-step distribution of tasks to IPs according to predefined criteria [92]. They can be divided into two groups by considering whether an improvement step follows. In **Without iterative improvement**, [91] presents an improvement of the execution time and communication energy by mapping in a spiral style from centre to boundaries. [102], iteratively pairs the two most communicated IPs together using IP ranking, merging IP set and refreshing IP set, in order to reduce the cost of hardware of the NoC. IP ranking calculates the communication bandwidth (sum of the bandwidth from it to other IPs and form other IPs to it) for each IP, and then sorts them. The merging IP process merges the most communicated IP sets with two-by-two style, iteratively, according to the IP ranking. Thereafter, the merged IP sets are refreshed by treating as an individual IP. A tree model IP grouping is introduced in [125].

In contrast, **with iterative improvement**, the final mapping solution can be improved. An example is [84] which maps a core graph to mesh based NoC with three steps. Step one uses a predefined criteria to construct an initial mapping, allocating the core with maximum communication demand to the node with maximum neighbours; then find the core with most communication demand with the cores already mapped, and allocating it to a node with minimum communication cost (hop-count bandwidth) with the mapped cores. In

the second step, a minimum path will be calculate with respecting bandwidth constraints. Then, final step improves the initial mapping iteratively by pair-wise swapping of mapped cores according to the second step. Obviously, the performance of these kinds of methods are fully dependent on the predefined criteria, for example, core selection, node selection and swapping strategy. The system performance cannot be guaranteed if the criteria have limitations, since a suitable criterion is hard to define for a complex system.

Summary

In this subsection, the static mapping algorithms are reviewed. They require the application information and system working environment to be known in advance and maintained during the application execution. They have enough time and resources to compute an optimised mapping solution at design time. However, this requirement may not always be satisfied in light of increasing complexity of application or increasingly variable working environment (such as considering user behaviours). In addition, the conflict between extended design space and limited resources will directly prolong the optimisation execution time. Thus, it is worth considering either moving to dynamic task allocation or introducing some accelerators for a fast optimisation.

2.3.2 Dynamic Mapping Strategy

Different from the static mapping strategy, which requires task allocation to be adjusted at design stage, the dynamic mapping strategy can assign tasks to an NoC at run time. It can improve the system adaptability to enhance performance in variable working environments. A taxonomy modified from the classification in [107] is presented and shown in Figure2.11.

Soft Real-Time Dynamic Mapping

The dynamic mapping algorithms used in soft real-time systems can place tasks onto a NoC with predefined criteria to improve the system performance. Unlike hard real-time systems, there is no hard deadline for task execution. Thus, the design aim is mainly focused on how to improve the throughput and reduce the remapping overhead, rather than guaranteeing the system performance. According to the aspects focused on, the algorithms in this category can be divided into several groups shown in Figure 2.11.

Power Consumption

Power consumption seems like the most popular topic in soft real-time dynamic mapping problems. [14] and [16] focus on heterogeneous NoCs. They apply First Free (FF), Nearest Neighbour (NN), Minimum Maximum Channel Load (MMC), Minimum Average Channel Load (MAC), and Path Load (PL) as the mapping algorithms. Similar work can be seen in [123]. [19] and [23] provide dynamic mapping for NoC architecture which supports multiple voltage levels. They both introduced a region selection step before task allocation. In [23], the task allocation is undertaken by one of the following methods, Best Case (BC), Worst Case (WC), Euclidean Minimum (EM), Fixed Centre (FC), Random Frontier (RF) and Neighbour-aware Frontier (NF). BC refers to the optimal solution. It is a kind of exhaustive search. Thus, it is only suitable for small problems. [21] and [22] consider the influence from user behaviours. [77] and [76] try to find suitable allocation using NN in a spiral route from centre to boundaries. In addition, the work in [78] also improves the energy performance by searching mapping in a spiral route, but it also considers the execution time.

Execution Time

In addition to power consumption, [78] has taken execution time into account by reducing the mapping time, reconfiguration time and task migration time. Its mapping solution is searched with a spiral route (from centre to boundaries)

by placing two communicated tasks close to each other. The work in [116] and [18] propose two mapping algorithms which are modified from GA to find an appropriate approach to trigger task migration and reduce the migration cost. They claim that triggering based on packets sent by a single node performs well. Beyond that, [33] attempts to use Integer Linear Programming (ILP) to improve the execution time and communication cost. However, the high complexity with ILP restricts it to application as a small problem optimisation tool.

Communication Work Load

Communication work load is another aspect that has occupied the interest of researchers. In [15], the authors select the method used in [14] and [16] to reduce the channel load, congestion and packet latency. In [41], [108] and [109], an agent-based mapping algorithm is proposed. It has Global Agents and Cluster Agents to handle dynamic mapping hierarchically. In [63], the most communicated task will be first packeted into a single IP according to the task graph to reduce the communication overhead. Thereafter, the requested IP will be mapped to the NoC by considering the minimum route distance with its master.

Work-Stealing

As well as direct mapping task to cores, dynamic mapping can also be achieved by scheduling among multiple cores for soft real-time tasks such as Work-Stealing (WS). A classical WS scheduler requires each core have a double-ended queue to store tasks and dequeue a task from the head of this queue, execute it and continue with the next task with in Late-In-First-Out, unless the queue is empty. Whenever a core finds its double-ended queue is empty, it attempts to randomly select a victim core to steal a task from the tail of the victim's queue. Another victim will be selected if the queue of the current victim is empty as well. This process may repeat forever or terminate when all cores have been checked. It can be seen that the computation workload can be balanced by WS, however, the stealing operation may cause

difficulty for the victim core to track back its lost task, if there is data tacking between the victim core and its lost task. At the same time, frequent stealing operations increase the task migration which will affect the network communication workload. Moreover, the WS may not always improve the stealing tasks response time, since the response time will be added by both stealing process and task migration. Therefore, only few research, such as [81] and [45] selected as the solution for NoC dynamic mapping.

Hard Real-Time Dynamic Mapping

The dynamic mapping algorithms used in hard real-time systems can be treated as the task admission controller. It decides whether a task can be allocated on a working NoC, taking into consideration the timing performance for both the new added task and existing tasks on NoC after remapping. Once permission has been obtained, an allocation will be processed. Otherwise, the task will be rejected. According to whether the response time of communication (traffic flows) on NoC can be guaranteed, these algorithms can be grouped into two sub-groups.

Considering Communication Cost

Based on when the performance analysis is addressed, this group can be further divided into online analysis and offline check online call. Examples of **Online analysis** can be seen in [80] and [37]. They attempt to use schedulability analysis as the performance inspector, to check whether a task can be allocated on a multi-core or many-core system such as avionics and medical devices. The results show that the systems hard real-time performance can be guaranteed if a successful task allocation can be made; however, the new task will suffer a relatively long admission time. The authors in [107] claimed that even if a joining request of a task has been rejected, the time used to manage this request is not wasted, and the lack of admittance can be used as feedback to make alternative operations. But the low passing rate may decrease resource utilisation, since only tasks which meet both the

computation and communication requirements can be allocated. In addition, the time-consuming evaluation method restricts these methods to only being suitable for small task sets. The **offline check online call**, by contrast, will have small dynamic remapping overhead. This is because, according to working requirements, a system will be divided into several states which will be well optimised by a static mapping strategy at design time and stored in the system at run time. The system can change its mode and load a suitable state according to its working environment. Thus, only a lightweight dynamic remapping strategy will be required. Recent work in this vein can be seen from both [36] and [106]. These methods can guarantee the system performance with a small remapping overhead. However, the predefined system stages reduce the system flexibility, as the information used to predefine the system working environment cannot always be obtained, especially if user behaviours are considered.

Without Considering Communication Cost

The methods in this subgroup mainly consider deadline distribution. The deadline of an application is distributed as the local deadline of each task. The resource manager will allocate these tasks at run time and ensure that their deadlines are satisfied. These methods can be seen in [54]. However, the communication among tasks, which is common in complex applications, is not considered.

2.3.3 Summary

In this section, the current mapping algorithms have been classified and reviewed. Although the static mapping strategy can guarantee system performance, its requirements may not always be satisfied, especially for some complex applications or dynamic working environments such as user behaviours. In addition, the conflict between extended design space and limited resources will result in a long optimisation time. The dynamic mapping

strategy can improve system flexibility with less a priori knowledge of system working environment and application by allocating tasks at run time. However the state-of-the-art cannot guarantee the system performance for soft real-time mapping problems, since they either do not have performance feedback or the feedback does not cover the worst-case scenario. In addition, although a global remapping strategy has been considered, the search area is limited, since the mapping construction has to follow a predefined criteria. For hard real-time dynamic problems, the existing algorithms will either be affected by long evaluation period (online or offline) or fail to consider the feedback on communications response time. These can result in a high remapping overhead, lack of flexibility or unpredictable communication timing performance. Therefore, the current challenge for hard real-time dynamic mapping problems is how to find a mapping to satisfy end-to-end timing performance (include both computation and communication part) with reduced mapping overhead.

Moreover, from the mapping method point of view, there are various current dynamic mapping methods. There is no one class which can be considered as popular, due to disadvantages such as lacking of feedback and low mapping constructive ability. By contrast, the static mapping methods are dominated by search-based algorithms which are further categorized into heuristic and constructive. Between them, the heuristic ones (one representative of which is GA, which is one sub class of Evolutionary Algorithms) are more attractive, since they are user-friendly and have strong search ability. That shows the EA is a good method for solving static mapping problem and may be considered for extension to dynamic mapping problem field. Therefore, in the following section, we will focus on the review of Evolutionary Algorithms.

2.4 Evolutionary Computing

Evolutionary Computing (EC) is also defined as a set of Evolutionary Algorithms (EAs) which are Evolutionary Programming [43], Genetic Algorithms

(GAs) [52], Evolution Strategy (ES) [115] and Genetic Programming (GP) [68]. It is a kind of heuristic search-based algorithm and can be considered as automatic problem solver or optimisation methods for complex problems [48], such as NoC mapping optimisation with both static and dynamic strategies (seen in Section 2.3). In this section the EA will be reviewed through with basic EA concepts, then GA and finally GA improvement.

2.4.1 EA Basic

The inspiration of EAs comes from the theory of Darwinian evolution [30]. EAs can solve or optimise problems by imitating the process of natural evolution. A population concept has been introduced in these algorithms. It consists of a number of individuals which represent potential solutions to a target problem. The living environment is imitated by a cost function which is used to indicate how well an individual can fit the environment by a numerical value. At the same time, it also represents the fitness of a potential solution to the target problem. In the natural environment, individuals breed and produce offspring, involving variations which are carried out by genetic operators such as mutation and recommendation (crossover). During the evolution, individuals with high fitness value will survive; others will die out. A generic flow of EAs is presented in Figure 2.12.

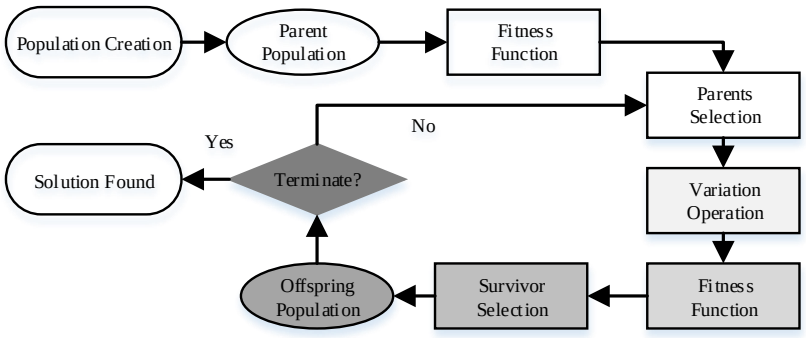

Figure 2.12: Generic Flow of EAs Modified From [119].

EAs start from an initial population generation, which normally follows a

random strategy and passes the environment/fitness evaluation. Thereafter, in the evolutionary iterations, suitable individuals will be selected as parents to breed offspring through variation operations. A fitness value will also be assigned to each offspring. The new generation involved in the next evolutionary loop, is generated by selecting the highly fitted individuals and eliminating others with a survivor selection mechanism which imitates natural selection. The evolution will be stopped with the fittest individuals whenever the termination condition has been met. These individuals can be treated as the 'near-optimal' or 'good-enough' solutions to the target problem. Several concepts or components used in this evolutionary loop will be reviewed as follows.

Representation

To apply EAs to solve or optimise an actual problem, a method is necessary to abstract a problem as a special data structure used in EAs. This abstracting and its inverting process are defined as representation and translation respectively. Normally, a solution is abstracted as a chromosome/s in an individual. Each chromosome consists of a number of genes which reflects the parameters of the problem. How a problem will be represented in EAs is determined by the problem characteristic and genetic variations strategy. The common representation methods are binary strings, integers, real numbers, graphs or hybrids. An example can be seen in Figure 2.13, which shows how the NoC mapping problem is represented in GA. The gene index and gene value of a chromosome can be treated as the task index and IP index respectively to indicate which IP a task will be allocated to.

Genetic Variation

Genetic variation can be understood as recombination (crossover) and mutation. These processes create offspring by applying small random changes

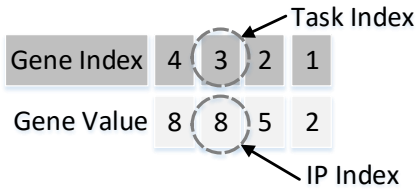

Figure 2.13: NoC Mpping Problem Representation in GA.

Note: Follows the example in Figure. 2.8.

on existing individuals. They are the primary power that makes EAs evolve towards the optimal.

1. Crossover

   Crossover produces offspring by recombining (partial swapping) the chromosomes from parents following a predefined crossover strategy such as single-point, two-point and uniform crossover [113]. The number of times the crossover can happen in a generation is indicated by a predefined probability which is also known as the crossover rate. Whenever this probability is satisfied, the selected segment or gene will be swapped between parents. An example can be seen in Figure 2.14.

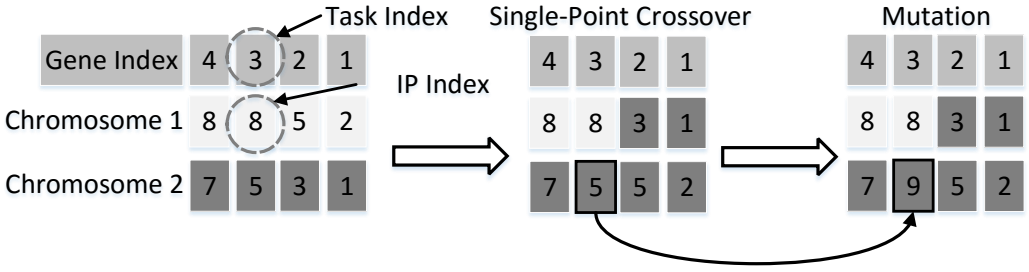

Figure 2.14: Crossover and Mutation Example.

Note: Flows the example in Figure. 2.13.

2. Mutation

   Mutation is another operation to provide genetic variation during evolution iterations. It generates the new gene value for each gene of a chromosome and replaces the old one if a mutation probability has

been satisfied (in a manner similar to crossover rate). One example is illustrated in Figure 2.14.

Evaluation

In the EA evolution process, how well an individual behaves in the living environment is evaluated as a numerical score by a cost function or functions in multi-objective optimisation. The score/s indicate how close the potential solution represented by an individual is to the optimal solution of the target problem. These results are used as guidances to lead the evolution towards the optimum by applying a selection strategy which imitates natural selection to distinguish which one survives and produces offspring in next generation (as discussed below).

Selection

The imitation of natural selection is undertaken by selection strategies in EAs. Selection exploits the potential solutions explored by crossover and/or mutation and extracts elites to force the EAs to search in a relatively small but effective search area, while at the same time continuously driving the search area towards an optimal solution. The commonly used selection strategies are uniform, fitness proportionate, fitness ranking and tournament selections. By applying a selection mechanism, EAs can achieve a rapid convergence to the optimal solution of a target problem.

Termination Condition

When and how an EA evolution should be terminated can be indicated by some commonly used criteria:

1. low convergence: if the obtained solutions cannot be further improved

within a number of generations;

2. number of generations: if a predefined number of generations has been achieved;

3. good-enough result: if a good-enough solution has been found, it may not be the optimal one;

4. optimal result: if the known best solution has been found.

2.4.2 Genetic Algorithm with NoC Mapping Problem

As mentioned at the beginning of this section (2.4), EC is a set that contains a series of evolutionary algorithms. One of these is the Genetic Algorithm (GA). GA is a robust problem solving and optimisation search tool proposed in [53]. One of its outstanding advantages is that its framework is sectional. It can be benefited by selecting various representation, crossover, mutation and selection strategies. Thus, almost all of the components and features of EAs which have been discussed above can be reflected onto GA. Because of this, it is popular in many complex problem solving or optimisation contexts. One of these is the NoC mapping problem.

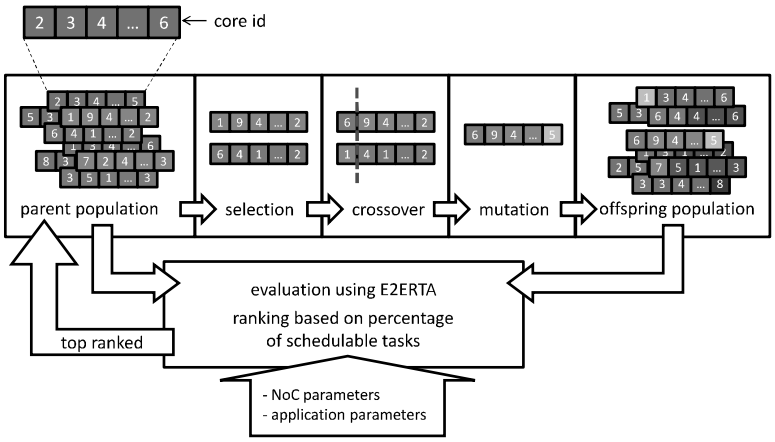

Figure 2.15: NoC's Mapping Problem with GA.

The popularity of the GA is in NoC mapping problems can be seen from the research in both static mapping and dynamic mapping that have been reviewed in Section 2.3. Similar to the generic flow of EAs (in Figure 2.12), a possible GA optimisation pipeline (shown in Figure 2.15) also starts from the creation of an initial parent population which contains a number of randomly generated chromosomes (i.e. randomly selecting the value of each gene of each chromosome). Its offspring population is bred by operating over the parent population with mutations and crossovers. How well the candidate solutions fit the problem to be solved is evaluated by a fitness function. In a case of hard real-time timing performance optimisation, it will cover both tasks computation time on IPs and flows communication time on an NoC. The example in this figure adopted the E2ERTA. The fitness value is how many tasks and flows cannot be scheduled based on current candidate mapping. The values will be further used to rank all chromosomes of the combined population and thus define which of them will be allowed into the next generation. The process is then repeated a fixed number of times or until a mapping without unschedulable tasks or flows is found.

The advantages of using GA to optimise NoCs performance can be understood as follows. First, unlike other heuristic algorithms which require specific knowledge of the target problem, a GA is a model-free heuristic algorithm and can be used as a general tool. This reduces the entry level for more researchers to explore this area. Compared with other model-free heuristic methods such as random search, local search, tabu search and simulated annealing, GA takes into account the fitness landscape [114] (which can indicate the guiding ability of a fitness function), and by providing a concept of population, the GA not only explores solution space in multiple directions at once but also prevents infinite resource cost. Secondly, GAs can easily be extended to consider many parameters simultaneously in order to support multi-objective optimisation problems which are common in reality. One example could be guaranteeing the timing performance of NoC after remapping, while at the same time reducing remapping overheard by reducing the number of tasks which need to be migrated. Thirdly, GAs are intrinsically parallel. Parallel

computing is easy to be implemented on GA to release its computational intensities. In addition, multi-factorial architectures can further enhance the optimisation ability of a GA to solve multiple independent problems in parallel. Since its framework is modularised, each component can also be tested individually and reused.

However, GAs are not problem-free. The shortcomings can be listed as follows. First, GAs cannot guarantee the final result after experiencing a number of generations. Second, since multi-individuals are introduced in a population, fitness assignment time will be longer than other conventional approaches, especially when a complex fitness function is applied. In addition, this aspect will become worse if multiple objectives are considered. Third, the GAs configuration, problem representation and calibrated parameters all affect their performance. Their designs usually require very careful consideration.

2.4.3 GA Improvement

The three disadvantages of GAs discussed in previous subsection 2.4.2 can affect GAs performance in dealing with NoCs mapping problems. The first two shortcomings are more critical than the last one, since the third shortcoming could be overcome by a number of experiments, extra self-adaptation mechanisms, or experienced designers. The first two shortcomings are related and can be combined into one problem, which is search efficiency. This is because, by improving the search efficiency of a GA, a larger design space can be explored. The larger space means more potential solutions and a higher optimisation success rate. The efficiency of the GA can be improved from two directions, architecture and implementation, which will be reviewed in this subsection.

Parallel GA

Parallel GAs (PGAs) not only have the advantages of Serial GAs (SGAs), but also high search efficiency and less prone to the sub-optimal problem. The taxonomy of PGAs is shown in Figure 2.16.

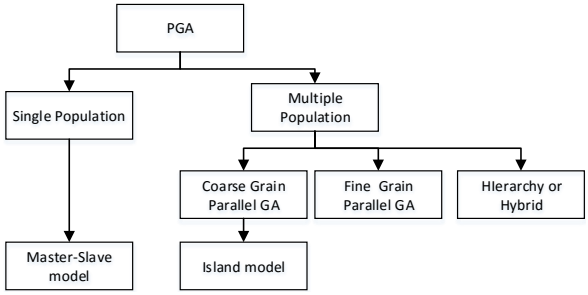

Figure 2.16: PGA Taxonomy modified from [2].

The Master-Slave model behaves like an SGA but with fast evaluation. It is proposed in [47] and further modified as a semi-synchronous and a distributed asynchronous concurrent model. Two recent examples that consider asynchronisation can be seen in [93] and [59]. Other PGAs have multiple populations. On the basis of the ratio between their computation and communication, these PGAs can be classified as either coarse grain parallel GA (cgpGA) with a high ratio or fine grain parallel GA (fgpGA) with a low ratio. The hierarchy or hybrid is a group which combines the cgpGA and fgpGA. Another common GA model (Island Model) is classified into cgpGA. The different architectures are shown in Figure 2.17. Since the Master-Slave model can fit our requirements (advantages of GA and high efficiency search) without introducing other communication mechanisms, it is reviewed with an example.

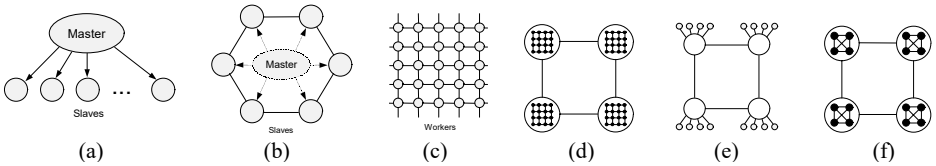

Figure 2.17: PGA Architectures [2]: (a) master-slave model (b) coarse grain, (c) fine grain; (d), (e) and (f) hybrid.

In a Master-Slave GA, the master is in charge of breeding offspring, survival selection and slave loading; the slaves undertake the individual evaluation. The loading and reloading of fitness functions for a Master-Slave GA with a population size of 4 and 2 fitness functions is shown in Figure 2.18. Whenever the master is ready (all individuals have already been generated and all fitness functions are in idle state), the master will assign two individuals to fitness function 0 and 1 respectively and launch them simultaneously. The second round release will only be started when all results have been collected by the master in order and all fitness functions have returned to idle state again.

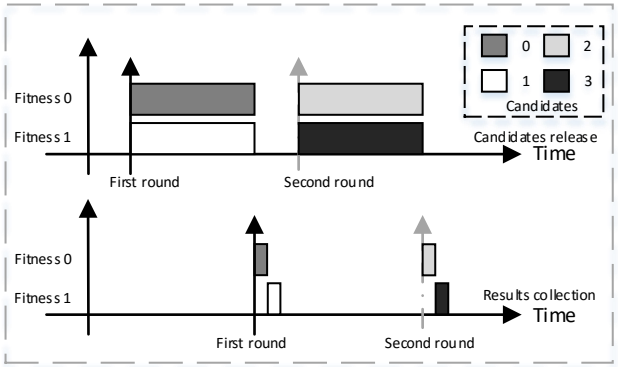

Figure 2.18: Master-Slave GA's Fitness Functions' Loading and Reloading.
Note: Population sieze is 4; number of fitness functions is 2.

Hardware GA

Apart from parallel architectures, researche also tries to implement GAs on various platforms to increase search efficiency. The authors of [95] present a hardware implementation of a sequential GA and further refine it in [96]. Although they applied a parallel parent selection, the performance improvement is not very significant and its memory interference component is rather difficult to implement. In addition, the research also proposes various GA operators to reduce the search time of GAs. [3] introduces an implementation of GA operators for a compact GA, which is suitable for a binary coding style. In [55], [98] and [40], the authors present implementations for either

both crossover and mutation or only crossover to produce improvements via hardware.

2.5 Summary

This chapter reviews the techniques related to NoC from the standard part of architecture, evaluation and optimisation. It can be seen that mapping can significantly affect an NoC performance with a given architecture and application. A static mapping strategy can guarantee system performance, but its requirements may not always be satisfied at design time. In addition, current mapping methods tend to suffer from a long design period, if the design space is enlarged. Dynamic mapping can improve system flexibility; however, pooling system performance feedback or low mapping constructive ability are the problems it faces in soft real-time systems, high remapping overheads or lack of considering communication evaluations are the drawbacks faced by hard real-time systems. Genetic Algorithms (GAs) which are one of the Evolutionary Algorithms (EAs), as one representative of the search-based mapping method, dominate the static mapping field and could be considered for being extended to the dynamic mapping problem, although current implementations suffer a long search time, which can also be treated as low search efficiency.

Chapter 3

Problem Analysis

This chapter provides a description of the research system model and further suggests the hypothesis to be tested. Based on this hypothesis, a breakdown of the problem is discussed which is then used as a guideline for the following chapters.

3.1 System Model

From the literature review (Chapter 2), we note that a single NoC architecture may not be able to support different kinds of applications. Therefore, it is crucial to select an appropriate architecture according to the requirements of a given application at design stage. The selected architecture will further affect which analysis methods is applied. For example, the timing performance of Ætheral can be evaluated using Synchronous Data Flow. However, a priority pre-emptive NoC can be assessed using End-to-End Response Time Analysis. A suitable mapping strategy should also be adopted based on application requirements. The chosen strategy directly determines when the mapping decision will be made and whether the mapping is done statically or dynamically. In addition, the decision affects the system performance

after mapping. Therefore, it is necessary to define a system model before we can further analyse the research problem. The system model used for conducting this research consists of the application model, NoC platform, mapping evaluation and mapping algorithm.

3.1.1 Application Model

The applications considered in this research can be divided into several functions which consist of one or a series of real-time tasks ($\Gamma = \{Task_1,\ Task_2,\ Task_3, ...,\ Task_n\}$). For example, whenever the brake function in a car application is enabled, the mechanical brake system and the brake light will react at the same time. Not all functions are required all the time. Some of them will only be enabled dynamically according to time, working environment or user behaviour. The details of the assumptions of the application are listed as follows:

- all tasks can be launched periodically or sporadically and executed independently. In other words, a task can be released without receiving messages such as data or a start signal from other tasks;
- each task can be represented as $Task_i$ and described by the following parameters:
  - c_i is the Worst-Case Execution Time (WCET) of $Task_i$,
  - t_i is the period of $Task_i$,
  - p_i is the priority of $Task_i$,
  - d_i is the deadline of $Task_i$,
  - r_i is the response time of $Task_i$,
  - m_i is the working state of $Task_i$ to show whether $Task_i$ has been enabled,
  - b_i is the maximum blocking time of $Task_i$,

  - $lep(k)$ is the set of tasks with the priority lower than or equal to $Task_i$,
  - $hp(i)$ is the set of tasks with higher priority than $Task_i$,
  - u_i is the utilisation of $Task_i$, it equals to $\frac{c_i}{t_i}$;

- the basic parameters ($c_i,\ t_i,\ p_i,\ d_i$) can be obtained at design stage;
- not all tasks have to be launched permanently, some system functionalities are driven by time, working environment or user behaviour;
- system can behave like soft real-time during remapping or mode changing, but the hard real-time timing performance should be guaranteed after remapping;
- the system is hard to be abstracted as finite states which describe a number of tasks should be executed, since various working environment or user behaviour or there are not enough resources to store pre-designed static task allocation for all states.

3.1.2 NoC Platform

Expected NoC Platform

The application considered in this research is dynamic (it cannot be fully predicted in advance) and requires that the expected NoC platform to be able to dynamically accept and map one or more tasks which are enabled by the behaviours of system or user. This dynamic task allocation process can simply accept tasks and directly distribute them to available IPs (processors). It can also be complex if global remapping (reallocate both the new accepted tasks and the existing running tasks) is considered. In other words, a task should be moved to any the IPs if they are available. In addition, the communication between IPs is facilitated by direct message passing instead of a shared memory. This is because a shared memory requires the support

from a memory controller and results in the memory controller not able to be physically reallocated. The memories used in the proposed NoC platform are applied locally to support task processing on each IP. Other parameters, such as number of tasks and IP cores, latency and bandwidth can be justified by the worst case of running application. This is because although we cannot fully predict the system runtime working stages, we can know it worst case, such as all tasks being enabled.

Technique Selection

Following the requirements of the NoC platform, we can select techniques as follow. From Chapter 2, we know that TDM and packet switching with priorities are two most popular solution branches for real-time NoC architectures. TDM routers are simple and area efficiency, however their dynamic performances are not as strong as PSwPri NoCs. This is because each communication path in TDM NoCs has to be pre-designed and stored in slot tables at each router for hard real-time services. Any change of current task allocation could result in path changes among many communications. In other words, the data in each slot table has to be regenerated and reconfigured. This directly results in a very costly reconfiguration process after remapping. Furthermore, for hardware resources, more sources have to be reserved for slot table in TDM NoCs, in order to support its application dynamic character and this becomes worse with the incrementation of application complexity. In contrast, PSwPri NoCs typically apply wormhole switching with credit-based flow control, routing algorithm and arbitration policy [51]. These techniques are low cost for remapping, since the communication path is generated and organised automatically, regardless of whether the task allocation has been changed or not. In addition, it is not necessary to reserve extra slot table, as long as the computation and communication requirements can be guaranteed. Therefore, we select PSwPri NoCs architecture as our basic platform and configure it as follow.

Topology determines how routers are interconnected in a NoC, such as mesh, torus, ring, butterfly, octagon, spidergon, star and so on. The research evaluates the performance of various NoC topologies, e.g. [1], [44] and [38], and claims that 2D mesh are is one of the most common type, since its regular structure and grid type shapes are easily extended for large NoCs and also best suited for the 2D layout on a chip. Thus, for general purpose (expecting our dynamic mapping algorithm to be widely accepted), the 2D mesh topology is selected.

Routing and arbitration determine how a packet will be transmitted on an NoC and how the NoC allocates resources for each packet. Compared with adaptive routing, deterministic routing is considered to be simple, fast and easy to evaluate. We select XY routing as it is the most popular in existing real-time NoCs [51]. Similarly, we decided to use Fixed-Priority arbitration scheduler [103], because it has a complete analysis for evaluating the NoC end-to-end response time [58]. Last but not the least, we introduced a virtual channel to compensate for the disadvantages of wormhole switching.

Selected NoC Platform

Following the discussion above, the NoC considered in this research is a pre-emptive arbitration NoC which can be described as the listed parameters:

- mesh topology;
- XY routing algorithm;
- virtual channels and credit-based flow control;
- fixed-priority arbitration;
- wormhole switching.

3.1.3 Mapping Evaluation

On the NoC platform, whether a task allocation satisfies the requirement criteria of the system can be estimated by End-to-End Response Time Analysis (E2ERTA) [58]. The calculated end-to-end response time by E2ERTA is derived from the response time of both computation on IP and the communication (traffic flow or package transmission over the NoC). Therefore, for a better understanding, we model these two resources of response time as task model and flow model respectively, as shown below:

1. Task Model follows the model made for the application. Each task can be represented as $Task_i = \{c_i,\ t_i,\ p_i,\ d_i,\ r_i,\ m_i,\ B_i,\ lep(k),\ hp(i),\ u_i\}$.

2. Flow Model follows the schedulability analysis in [103]. The traffic flow set can be described as $F = \{Flow_1,\ Flow_2,\ Flow_3, ...,\ Flow_n\}$ and each flow can be presented as $Flow_i = \{C_i,\ T_i,\ P_i,\ D_i,\ J_i^R,\ J_i^I,\ R_i,\ S_{id},\ S_{ii}\}$.

   - C_i is the basic latency of $Flow_i$;
   - T_i is the period of $Flow_i$;
   - P_i is the priority of $Flow_i$;
   - D_i is the deadline of $Flow_i$;
   - J_i^R is the release jitter of $Flow_i$;
   - J_i^I is the interference jitter of $Flow_i$;
   - R_i is the response time of $Flow_i$;
   - S_{id} is the direct interference set of $Flow_i$;
   - S_{ii} is the indirect interference set of $Flow_i$;
   - L_i is used to calculate C_i, if C_i is not given;
   - U_i is the utilisation of $Flow_i$, it equals to $\frac{C_i}{T_i}$.

The flows are expected to inherit the priority from its initial $Task_i$, without considering multiple broadcast for easy evaluation. S_{id} and S_{ii} respectively present the direct and indirect interference set of $Flow_i$. The flows in these two sets affect the worst-case response time of $Flow_i$ by pausing the commnucation of $Flow_i$. The definitions of these can be seen in Section 2.2.2.

3.1.4 Mapping Algorithm

The application focused on in this research is a hard real-time dynamic application. It requires tasks enabled by the system or users to be accepted and allocated dynamically on the expected NoC platform. The mapping process involves remapping existing tasks or global rearrangement when necessary. Thus, dynamic mapping methods are our natural choice. However, according to the background review, most current dynamic mapping solutions pay too much attention to allocate new added tasks for reducing mapping cost and less focus on remapping or the evaluation of system timing performance. Although, some researches have been made to improve the current drawbacks, they are failed in low remapping success rate or low system flexibility or high resource cost. Therefore, there is no one solution can be accepted by most people among various dynamic mapping solutions and they are not suitable for this research. Thus, it is worth to consider to transplant and improve the solutions used in the most similar application scenarios and adapt them to the dynamic system, instead of struggling in optimizing the existing dynamic mapping methods.

Static mapping is the most similar scenario to dynamic mapping problem which can be treated as several fast static mapping process (discussed in the following section). Therefore, we could focus on the excellent candidates in static mapping and consider the possibility of applying them to dynamic mapping field. From the review, the most common static mapping is search based mapping solution which can be further classified into heuristic and constructive method. Between them, the heuristic one which can be represented

by GA attracts more attentions because of its strong searching ability and low entering requirements. Thus, we tentatively select GA as the mapping solution of this research and discuss the potential advantages of doing so.

The benefits of applying GA can be listed as follow: firstly, the increment of NoC size and complexity of application directly enlarges the design explore area and indirectly causes the impossibility of searching mapping exhaustively and dynamically. Although the optimal solution cannot be guaranteed, a GA could provide near-optimal or good-enough solutions to satisfy the system requirements. At the same time, it reduces the entering barriers by transforming the difficult process of listing all possible solutions into the relatively simple and fast process of searching.

Second, the search of GA is stochastic without any predefined limitation, thus, it can maximise the coverage of search area and increase the remapping success rate. In addition, the concept of population not only provides multiple alternative solutions at the same time, but can also easily be improved by use of a parallel architecture such as Master-Slave GA. All these are helpful means to generate more candidate solutions and provide a higher possibility of obtaining optimal or good-enough solutions. Moreover, it is easy to manipulate multi-objective in search-based algorithms. This provides the ability to adjust the trade-off between timing performance and remapping cost (number of tasks in migration), in order to reduce the remapping overhead.

Third, GA not resource-hungry. This is of great importance in dynamic task allocation search because dynamic optimisation is normally executed with limited resources. No matter it is using the computation abilities of IPs to execute the searching algorithm, or introducing a dedicated component, the limitation of resources is the barrier which cannot be avoided. The GA can maintain its resources cost during its whole searching process. Therefore, GA could be a good choice for this research. However, this choice is not problem free. The challenges are discussed in the next section.

3.2 Problem Analysis and Thesis Hypothesis

The distinctions between dynamic and static mapping can also be seen from two other aspects. The first is whether all tasks should be enabled at the beginning and kept activated forever. (In a dynamic system, which task should be enabled is determined by the current system state, which can be affected by system behaviours or user behaviours). The second is whether there is a process for tasks migration. For example, Figure 1.4, shows the tasks originally allocated on IP3 are migrated to IP6, in order to vacate enough computation resources to accept the new added task. Both the existing and new added tasks may require a migration. Therefore, the process of a dynamic task allocation for NoC can be described as shown in Figure 3.1. It covers both the mapping algorithm steps and two extra steps (gathering related tasks and task migration).

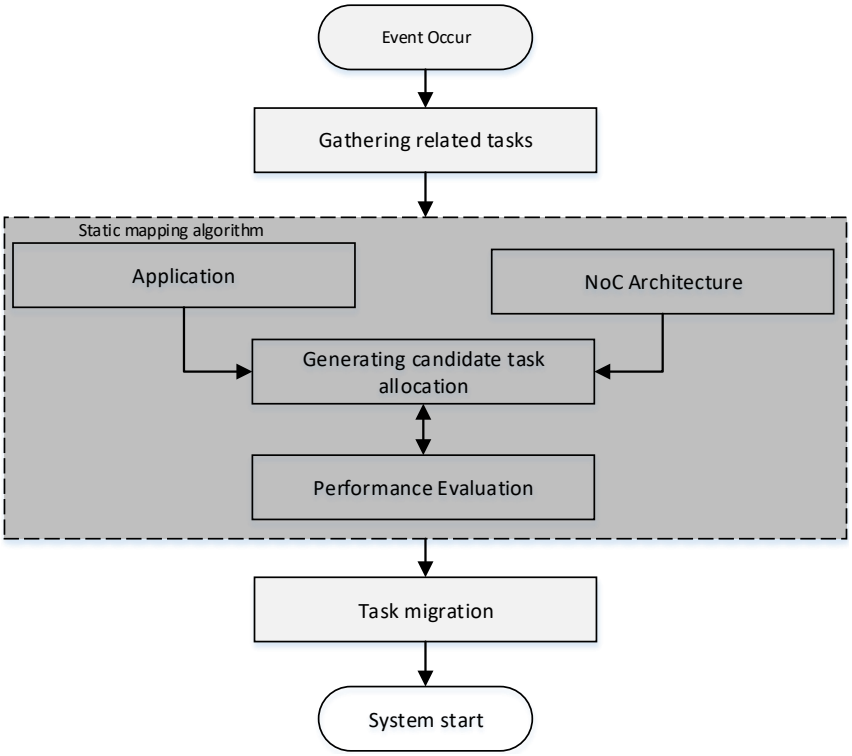

Figure 3.1: NoC Dynamic Mapping Process.

If we use an equation to represent the overall execution time for a dynamic task allocation, the equation can be written as Equation 3.1, where ET is the execution time. $ET_{dm},\ ET_{grt},\ ET_{ma}$ and ET_{tm} represent the execution time used by overall dynamic mapping, gathering related tasks, mapping algorithm and task migration respectively.

$$ET_{dm} = ET_{grt} + ET_{ma} + ET_{tm} \tag{3.1}$$

Since we have assumed that the basic task parameters ($c_i,\ t_i,\ p_i,\ d_i$) can be known at the design stage, the related tasks for a specific event can be predicted. Therefore, ET_{grt} can be treated as zero and Equation 3.1 can be modified as Equation 3.2.

$$ET_{dm} = ET_{ma} + ET_{tm} \tag{3.2}$$

3.2.1 Problem Analysis

Based on Equation 3.2, we can analyse the research problem from two steps, depending on whether the ET_{tm} factor is considered.

Without Considering Task Migration Time

If we assumed that no time is required for migrating tasks over the NoC, the ET_{dm} will be determined by ET_{ma} only. Then, the time allocation of such a dynamic system can be presented as in Figure 3.2. Although a system like this is dynamic, during the period between the occurrence of two events (e.g. from time point 1 to time point 2), the system behaves statically. This is because in this period before the new event arrives, which task should be activated has already been determined by the previous system state. In other words, before the new event arrives, the task allocation could be treated as a

static mapping process. Therefore, a dynamic task allocation process can be treated as a series of fast static mapping processes.

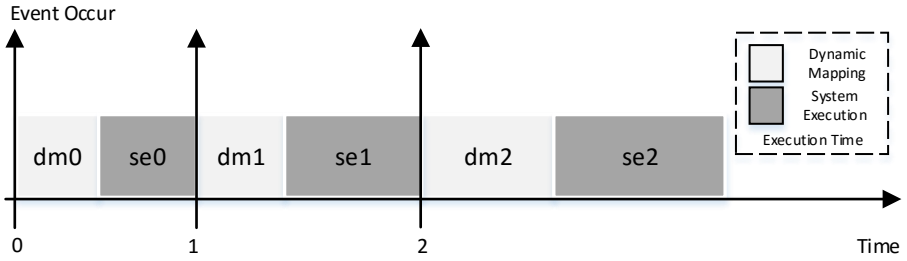

Figure 3.2: Example of Dynamic Mapping Time Allocation.

With Task Migration Time

If we take task migration time into account, a time slot between dynamic mapping and system execution would have to be added for transferring the information (e.g. code, states, data and so on) of tasks over the NoC. Task migration is a complex problem. In this research, although we do not calculate the exact task migration time, we can still potentially reduce ET_{tm}. This is because the more tasks migrated the more time is required. Thus, cutting down the number of migrated tasks can reduce the ET_{tm} and optimise the remapping cost.

3.2.2 Thesis Hypothesis

Based on the problem analysis, we can formulate our hypothesis:

An schedulable task allocation can be found dynamically and efficiently to meet the application's hard real-time timing requirements and reduce task migration cost in an NoC based Multi-Processor System-on-Chip.

Considered in light of the system model and problem analysis we have discussed, this hypothesis can be divided into two steps:

- first step
  - *without considering task migration time, by using an accelerated task dynamic mapper (which can be implemented as a search-based algorithm such as Genetic Algorithm) for wormhole NoC with priority-preemptive arbitration, the global remapping overhead (ET_{dm}) can be significantly reduced and the system hard real-time performance can be guaranteed after remapping*;
- second step
  - *taking task migration time into account, with respect to a hard real-time system timing performance, by minimising the difference between the new and old task allocations, the number of tasks migrated in the following task migration process can be reduced, thus reducing the task migration time cost.*

3.3 Problem Breakdown

The main problem we need to alleviate is to efficiently find an acceptable task mapping to meet the applications hard real-time timing requirements, even in a worst-case scenario. This can be broken down into the two aspects timing and efficiency. In our system model, we select the E2ERTA as our evaluation method, which is a worst-case timing analysis. If any mapping is able to pass this evaluation, the timing performance can be guaranteed.

However, finding a suitable mapping efficiently is difficult and presents the main challenge in this research. This is because both exact hard real-time analysis (e.g. E2ERTA) and dynamic mappers (e.g. search-based algorithm) are all time-consuming algorithms from the perspective of the state-of-the-art. An example is shown in [100]. It is a Java implementation of a GA with E2ERTA as fitness function for NoC static task allocation by using a multiple core desktop computer. Its searching time can be hours or days for one

mapping search. Therefore, to alleviate this problem, we have to identify the processes that cause the searching process to be time-consuming, and then improve them.

According to our hypothesis, when we do not consider ET_{tm}, ET_{dm} is only determined by ET_{ma}. In other words, the efficiency is directly related to the ET_{ma}. Since we intend to apply exact hard real-time analysis as the fitness function with search-based algorithm as dynamic mapper to search, the dynamic mapping searching process could be written as Equation 3.3a and 3.3b.

$$ET_{ma} = ET_{dynamic\ mapper} + ET_{Fitness} \tag{3.3a}$$

$$ET_{ma} = ET_{searching\ operation} + ET_{Fitness} \tag{3.3b}$$

As a result, to improve the efficiency of dynamic task allocation, we should consider making improvements to both the fitness function and search-based algorithm respectively. The Chapter 4 and Chapter 5 will attempt to improve the fitness from both theoretical and practical points of view. Chapter 6 will focus on the improvement of the dynamic mapper. After that, Chapter 7 will combine the improvements together to achieve the research goal.

3.4 Summary

In this chapter, we specified a system model to bound the type of dynamic application we focused on and selected the NoC platform, evaluation model and mapping algorithm. Based on our system model, we formulated hypothesis and analysed it from two steps. We also further discussed the research problem breakdown, which will be used to guide subsequent chapters.

Chapter 4

Performance Improved Inexact End-to-End Response Time Analysis

As stated in the research problem and its breakdown which were discussed in Section 1.3 and 3.2, the efficiency of dynamic task allocation using search-based algorithms for hard real-time applications can be affected by both the efficiency of the search algorithm and the evaluation efficiency (hard real-time timing analysis method). However, the influence from evaluation methods is more serious form two reasons. The first is that, compared to search operations, fitness functions are more complex in most situations and have longer execution time. The second one is that, as an evaluation method, the fitness function has to assess each candidate solution and has to be loaded a great number of times during the search process. So, even a tiny improvement in fitness functions execution could lead to a significant improvement to the overall search time. Therefore, to improve the efficiency of dynamic task allocation with hard real-time constraints, we can begin by considering optimisation of the hard real-time timing evaluation method.

However, it is difficult to optimise a hard real-time timing evaluation method,

such as End-to-End Response Time Analysis (E2ERTA). E2ERTA is based on a complex iterative calculation. Its execution time is difficult to predict. In addition, increases in the number of cores in NoCs and in the complexity of applications (i.e. increasing number of tasks and communication flows) make E2ERTA calculation significantly more difficult and also imply a high computation cost. This cost is not very critical in static task allocation problems, since in most time there are enough computing resources and time for search for a suitable mapping. In contrast, it is critical in dynamic task mapping or admission controllers whose working resources and time to respond are limited. This is because long time analysis directly increases waiting time before an admission decision can be made. Consequently, it may result in system errors or even crashes. Therefore, whether the computation time of E2ERTA can be reduced, and the magnitude of reduction, are important issues for increasing the efficiency of our dynamic task allocation.

In this chapter, we will analyse how the complexity of E2ERTA arises and introduce two techniques to modify E2ERTA to a less tight analytical model (Inexact End-to-End Response Time Analysis). Its performance is evaluated by experiments on a software platform.

4.1 Complexity of End-to-End Response Time Analysis

The efficiency of E2ERTA is directly related to its complexity of calculation. High complexity of calculation would lead to long execution time and further result in low efficiency in dynamic task allocation search. As reviewed in Section 2.2.2, the end-to-end response time of a task can be divided into computation and communication response time.

For calculating the exact value of the end-to-end response time of tasks on NoCs, the author in [58] combined Equation 2.3 (page 46, computing the

response time of tasks on IPs) and Equation 2.4 (page 48, calculating the response time of flows on NoCs) by assuming that the release jitter of a traffic flow can be replaced by the worst-case response time of the initial task of the flow (that is, $J_i^R = r_i$). By observing the rewritten E2ERTA equation (Equation 2.5, page 48), we can conclude that the computation of E2ERTA is based on an iterative calculation. To process this iterative calculation, a number of intermediate results are needed before we can obtain the final result. The more intermediate results are required, the longer will be the computation time. According to the requirement of E2ERTA, the termination condition of this iterative calculation is either $R_i^{n+1} = R_i^n$ *or* $R_i^{n+1} > D_i$. This means that for each calculation, the number of iterations is not fixed and consequently the number of intermediate results is not a constant. We can make the assumption that we only consider the termination condition as $R_i^{n+1} = R_i^n$ and ignore $R_i^{n+1} > D_i$, since smaller D can terminate the iterative calculation early. Under this assumption, the lower priority a task has, the greater number of intermediate results and more computation time it will suffer. Thus, the complexity of calculation of E2ERTA will be increased along with increases in the size of the task set. In other words, the execution time of calculating E2ERTA is caused by its dependence on iterative calculations. Therefore, to improve its efficiency, we need to alleviate the workload of its iterative calculations.

4.2 Inexact End-to-End Response Time Analysis

As discussed above, the particular characteristics of E2ERTA (and mainly its dependence on iterative calculation) is a barrier to improving its efficiency. More iterations mean a longer execution time. Therefore, reduction in the number of iterations required becomes a crucial consideration. In this section, we will analyse this problem from two perspectives and propose two possible techniques (Pre-Check and New Lower Bound) to alleviate this problem,

according to whether the iterative calculation can be avoided. After that, these two techniques will be assembled with various analysis composites in order to explore the performance and coverage trade-off of an Inexact End-to-End Response Time Analysis.

4.2.1 Pre-Check

This subsection will organise and discuss the following questions: what is Pre-Check; how to calculate the potential boundaries; which pair boundary should be selected; and what are the limitations of Pre-Check?

Pre-Check Definition

As an exact calculation, E2ERTA can provide accurate worst-case response time for tasks and flows. However, the cost of this accuracy is a longer execution time. In reality, the result we are focus on is whether a task or flow can be scheduled, rather than the exact value of response time. If a method can identify the schedulability without computing the exact response time value, the execution time of E2ERTA may be reduced. A possible solution is to identify a range of the final response time, and this is presented in this section. It is named Pre-Check (PRE). An example is shown in Figure 4.1. In this example, the deadline is higher than the upper bound of response time, so the observed task or flow can be always scheduled.

Potential Boundaries

There are several methods that could be used to identify the boundary of the final response time. The first one is inspired by Equation 2.5 (page 48). It can be observed that the calculation of E2ERTA includes a ceiling function ($\lceil x \rceil$) which returns the minimum following integer number. In mathematics, a ceiling function can be replaced by inequalities such as Equation 4.1.

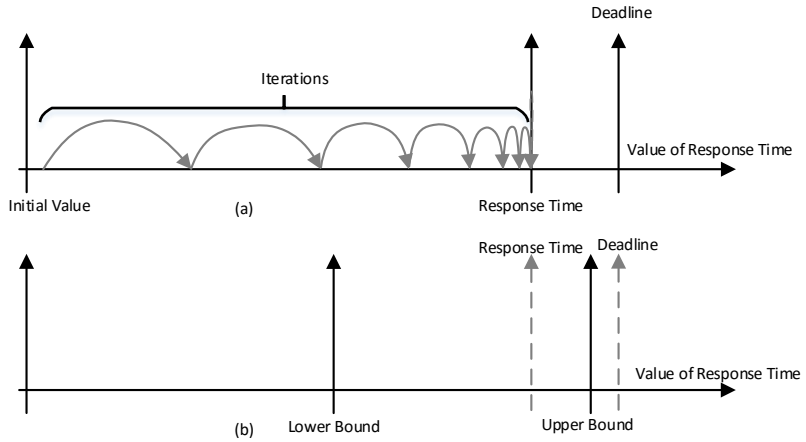

Figure 4.1: (a) Example of Original E2ERTA Iterative Calculation, (b) Example of PRE

$$x \leq \lceil x \rceil \leq x + 1,\ x \in R \tag{4.1}$$

Therefore, if this ceiling function could be replaced by applying inequalities, the upper and lower bounds of the response time of a task or flow can be found. Taking the flow part as an example, the Equation 2.5 can be modified as below. X is either the lower bound or upper bound of R_i.

$$X \geq C_i + \sum_{\forall j \in S_{id}} \left\lceil \frac{R_i + r_j + J_j^I}{T_j} \right\rceil C_j \tag{4.2a}$$

$$X \leq C_i + \sum_{\forall j \in S_{id}} \left\lceil \frac{R_i + r_j + J_j^I}{T_j} + 1 \right\rceil C_j \tag{4.2b}$$

Since the R_i can take any value in the range $[R_i^{lb}, R_i^{up}]$, we can use R_i to replace X to calculate the boundary situation. Then R_i exists on both sides of these two equations and could be rearranged to one side and further obtain Equation 4.3a and 4.3b. Because $\frac{C_j}{T_j}$ equals to U_j and $1 - \sum_{\forall j \in S_{id}} \frac{C_j}{T_j}$ is always positive, Equation 4.3a and 4.3b can be further modified to obtain Equation

4.4a and 4.4b.

$$(1 - \sum_{\forall j \in S_{id}} \frac{C_j}{T_j}) R_i \geq C_i + \sum_{\forall j \in S_{id}} \left\lceil \frac{r_j + J_j^I}{T_j} \right\rceil C_j \tag{4.3a}$$

$$(1 - \sum_{\forall j \in S_{id}} \frac{C_j}{T_j}) R_i \leq C_i + \sum_{\forall j \in S_{id}} \left\lceil \frac{r_j + J_j^I}{T_j} + 1 \right\rceil C_j \tag{4.3b}$$

$$\Downarrow$$

$$R_i^{lb} \geq \frac{C_i + \sum_{\forall j \in S_{id}} \left(r_j + J_j^I\right) U_j}{1 - \sum_{\forall j \in S_{id}} U_j} \tag{4.4a}$$

$$R_i^{ub} \leq \frac{C_i + \sum_{\forall j \in S_{id}} \left[\left(r_j + J_j^I\right) U_j + C_j\right]}{1 - \sum_{\forall j \in S_{id}} U_j} \tag{4.4b}$$

Apart from only using inequalities to replace a ceiling function, researchers also tried to apply other methods to find the range of response time. Here are two techniques. The authors in [10] also tried to use the workload to find an upper bound of task's response time, as shown in Equation 2.6 (page 49). This can be used as a sufficient test for the schedulability test of task. In [31], the authors explored this problem from considering the lower bound of tasks' response time. They pointed out that the lower bound can be found by using Equation 2.7a, 2.7b and 2.7c (page 50). The $I_j(R_{i-1})$ denotes the worst-case interference due to $Task_j \in hp(i)$ occurring during the response time of $Task_{i-1}$.

Boundary Selection

According to the discussion before, there are four potential boundaries (two for upper bound and two for lower bound) that could be used. For different calculations (task or flow), it is necessary to identify which boundary can be used. In a task's response time analysis, r_j *and* J_j^I do not exist and can be

set as zero. Compared to Equation 2.6 (page 49), Equation 4.4b is pessimistic after setting r_j *and* J_j^I to zero. Therefore, Equation 2.6 will be selected as the upper bound of task's response time. Moving to the lower bound, the proposed lower bound Equation 4.4a may be less tight than Equation 2.7c (page 50). This is because Equation 2.7c selects the maximum from a series of lower bounds. The result of Equation 4.4a is one candidate in this series and may not be the maximum one. Thus, Equation 2.7c has been selected as the lower bound of task's response time. The boundary of tasks' response time can be obtained by Equation 4.5a and 4.5b.

$$r_i^{lb} = \max_{\forall k=1...i} r_i^{lb}(k) \tag{4.5a}$$

$$r_i^{ub} = \frac{c_i + \sum_{\forall j \in hp(j)} c_j(1-u_j)}{1 - \sum_{\forall j \in hp(j)} u_j} \tag{4.5b}$$

In a flow's response time analysis, r_j *and* J_j^I are present. Therefore, Equation 2.6 (page 49) cannot be directly selected as the upper bound of flow's response time. In addition, considering the calculation complexity, Equation 4.4a and 4.4b are similar. Partial components among them can be reused. This can further reduce the execution time. Therefore, Equation 4.4a and 4.4b are selected as the lower and upper bound of flow's response time respectively.

Here, what we need to notice is that the boundary obtained by Equation 4.4a and 4.4b is only the response of communication part instead of end-to-end response time of a flow. The actual end-to-end response time of a flow should include both computation and communication, in other words, it should be $r_i + R_i$. Thus, the using of the flow boundary should consider the related r_i.

Limitation

Although PRE can provide much simpler calculation than the original E2ERTA, it has one limitation that reduces the efficiency of applying it to replace E2ERTA. This is because whether PRE can identify the schedulability depends on the deadline distribution. This phenomenon can be explained by Figure 4.2. A confident result of schedulability can be obtained when a deadline is allocated such as in case a or c. In case a, the deadline is always lower than the lower bound, and the observed task or flow is always unschedulable. In case c, on the contrary, the candidate task or flow can be always scheduled. However, if the deadline is allocated between the lower bound and upper bound as seen in case b, the schedulability test is failed. As a result, the PRE can only be used as a sufficient test of E2ERTA. Therefore, another method that can compensate the limitation of PRE is needed and will be discussed in the following section.

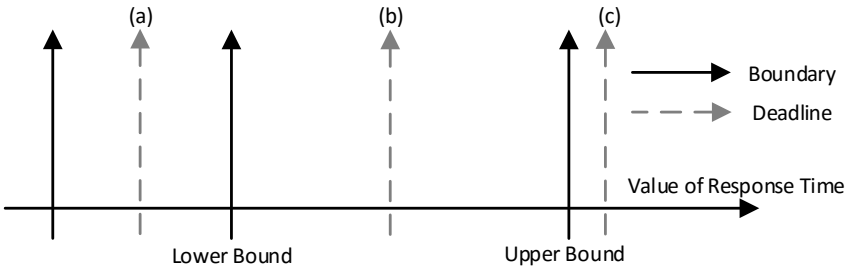

Figure 4.2: Example the Limitation of PRE

4.2.2 New Lower Bound

This subsection will discuss a possible technique, New Lower Bound, to reduce the number of iterations required during E2ERTA calculation.

New Lower Bound Definition

Normally, the calculation of E2ERTA starts from an initial value and requires multiple iterations, which are represented by the solid lines in Figure 4.3. Through these iterations, the real value of response time is approached constantly until the real value is obtained (and cannot be increased any more) as long as the value is not higher than the deadline. Although only the final result can represent the response time, these intermediate iterations have to be calculated. The calculation of intermediate iterations is the only way to get the final result, which may be difficult and take a long time. But this does not mean there is no possibility of reducing the number of iterations and maintaining the accuracy of calculation at the same time. One possible solution is using a larger initial value to replace the original ones (c_i for task and C_i for flow). Here, this possible solution is named New Lower Bound (NLB). An example is shown in Figure 4.3 with dashed lines.

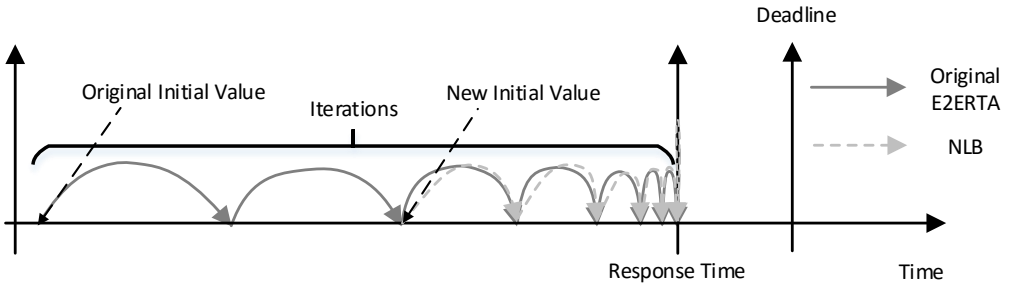

Figure 4.3: Example of NLB

Lower Bound Calculation

In [31], the authors applied a lower bound for a task's response time as the new initial value to replace the original one (c_i) to start response time analysis. The results show that fewer iterations, and hence shorter computation time, are required compared to the original response time analysis. This provides the possibility to select this technique to optimise the task part calculation of E2ERTA. In addition, although this technique is mainly focused on the tasks'

aspect, it could still be introduced to flows. This is because the flows' response time analysis is inherited from tasks'. They have the same characteristics (iterative calculation). Therefore, Equations 4.5a and 4.4a could be directly selected for calculating tasks' and flows' new initial values respectively.

Limitation

The main limitation is that the improvement in computation time may not be stable. In order to explain this situation, we can assume that:

- the total execution time of NLB is ET_{NLB},
- the execution time used to calculate the lower bound is ET_{clb},
- the execution time used to compute the following exact calculation is $ET_{NLB.E2ERTA}$,
- the execution time used by original E2ERTA is $ET_{O.E2ERTA}$.

Since the total execution time of NLB consists not only of the time for executing exact calculation, the total value will be the sum of ET_{clb} and $ET_{NLB.E2ERTA}$. In extreme cases, a remarkable improvement in $ET_{NLB.E2ERTA}$ can be achieved; however, the time spent for ET_{clb} may be very long. In other words, $ET_{NLB.E2ERTA} \leq ET_{O.E2ERTA}$ can be guaranteed, but $ET_{clb} + T_{NLB.E2ERTA} \leq ET_{O.E2ERTA}$ cannot. Thus, this drawback may cause ET_{NLB} to be similar or even worse than $ET_{O.E2ERTA}$ in some extreme cases.

We can also understand this limitation from another point of view. We could get a lower bound of response time of a task or a flow through process NLB. What can be guaranteed is that the result is not smaller than the original initial value. That means the lower bound we got from NLB may be the exact original initial value or slightly bigger than the original initial value (not big enough to reduce the number of iterations). Thus, when this situation happens, the performance of NLB may be worse than the original E2ERTA.

Table 4.1: Analysis Composites List

Index	Schemes	Details
1	PRE	Only using PRE to improve E2ERTA
2	NLB	Only using NLB to improve E2ERTA
3	PRE+ E2ERTA	Using PRE to pre-check schedulability. If there are no exact results, the E2ERTA will be used to do exact test.
4	PRE+NLB	Using PRE to pre-check schedulability. If there are no exact results, PRE will be abandoned and the NLB will be used to do exact test.
5	PRE+conditional started NLB	Using PRE to reduce searching space. If PRE cannot further reduce searching area in five generations, then NLB provides exact test to guide the rest of the searching.

4.2.3 Analysis Composites

As discussed before, if we only apply PRE or NLB to optimise E2ERTA, the optimisation effect may not be outstanding (either the final results may not be guaranteed or the execution time may be similar or even worse than original E2ERTA) in some extreme cases. It seems possible, however, that the disadvantages of these two techniques could be compensated to a certain degree if they are assembled with some combination schemes. To explore performance and coverage trade-off by combining analyses, we propose a series of schemes which are listed in Table 4.1.

As a sufficient test, PRE cannot guarantee the calculation of response time analysis. It has to co-operate with an exact test, which could be E2ERTA or NLB. For example, the PRE could be executed first. If the result has been found then the following exact test could be ignored; otherwise, the exact result will be calculated by the following test. Therefore, PRE could be assembled with either E2ERTA or NLB (scheme 3 and 4) to alleviate its limitations.

Apart from being used as a sufficient test, PRE could also be used as a fast

evaluation in a search-based algorithm. Normally, the search area of an NoC mapping problem is large. Most of the search space does not contain suitable solutions. In this situation, a graded fitness function which can provide multiple levels of execution speed and precision could be selected. The fast but inexact fitness can be applied to quickly reduce the search area. After that, focusing on a small search space, slow but exact fitness can be applied for a deep search. PRE is exactly this kind of fast but inexact fitness function. In addition, the dynamic mapper intended for selection is a search-based algorithm. Therefore, it is worth considering an assembly scheme such as this fifth scheme in this research.

In the fifth scheme, the PRE will be executed first to reduce the search area. If the best candidate cannot be further improved within five generations, it will be considered that the PRE can no longer guide the optimisation. Then an exact evaluation (NLB) will be enabled. Any number of generations can be tolerated, depending on how much confidence we have about PRE. Five is selected here simply as an example. Certainly, there may be other assembly schemes. The five we selected is already enough to show the performance of PRE and NLB. Other schemes could be explored in the future work.

4.2.4 Experiment and Results Analysis

This subsection will propose an experiment for exploring performance and coverage trade-off of various analysis composites. It is organised as follows: experiment platform; experiment configuration; and results analysis.

Experiment Platform

The experiment platform established is designed with a search-based algorithm. There are two reasons why a search-based algorithm has been selected. First, as a graded fitness function, the fifth scheme is ideally proposed for a search-based algorithm, since it can provide fast and deep search ability for both large

and small search area. Second, the limitations of PRE and NLB do not arise in every instance, but are present in a few extreme cases only, such as a bad deadline distribution (between the lower bound and upper bound of response time) and a long execution for obtaining new initial value. The more cases have been checked, the greater the coverage and higher the accuracy of the evaluation will be. Therefore, the experiment platform should provide various cases for these analysis composites. Search-based algorithm will generate many different candidates during its search process. Thus, it can be used as a cases provider on this platform.

Table 4.2: Chromosome Representation

Gene index (Task index)	1	2	3	4	...	7	8	9
Gene value (Processor number)	5	7	5	9	...	8	1	3

The search-based algorithm applied on this platform is GA, whose optimisation pipeline has been shown in Figure 2.15 (page 66). It works by manipulating chromosomes which represent individual solutions to the problem we are trying to optimise. In this experiment, a chromosome must represent a specific case which is also a mapping of tasks to cores over an NoC. An example is shown in Table 4.2, where each gene of the chromosome represents a task. The content of each gene indicates to which processing element the current indexed task will be allocated. Therefore, the number of genes on a chromosome is the number of application tasks we are trying to map.

Experiment Configuration

To measure the performance of our proposed analysis composites in various situations, we configure our experiments as follows:

- Computer Platform:
  - Intel(R) Core(TM) i7-3770 CPU @ 3.4GHZ,

  - Windows 7 64-bit Operating System,
  - Java based implementation,
  - compiler version: Java 7 version 51,
  - Real-time priority (setting the priority of application to real time in windows task manager for faster response);
- NoC platform configuration:
  - the size of NoC is from small to large, 4*4, 6*6, 9*9 and 10*10,
  - 10*10 is the largest NoC used in the baseline [101];
- Benchmark configuration:
  - Autonomous Vehicle (TB1, 38 tasks, average task utilization is 19.15%),
  - Synthetic (TB2, 50 tasks, the average task utilization is 40%) [101],
  - Extended Synthetic applications (TB3, 100 tasks, average task utilization is 41.30%) in Table A.1 and A.2,
  - summary of benchmark is shown in Table A.5;
- GA configuration:
  - follows the suggested GA setting in [101],
    - the probability rate of crossover is 0.5%,
    - the probability rate of mutation rate is 0.01%,
    - the size of population is 100,
    - the number of generations is 50.

For a given NoC platform and test bench, we test these five analysis composites and E2ERTA with the same initial population 100 times respectively, and use the average results to draw the curves. This approach is determined by two factors: first, because the random created initial population of GA will cause the evolution to start from unfair stages and further affect the evolution; and second, since GA is a stochastic search, one time testing cannot illustrate the difference among all analysis composites and E2ERTA.

Results Analysis

Among various experiment configurations, some of their results are similar. Here only five cases have been selected as examples and are shown in Figure 4.4 to 4.8. More details to show the results distribution of these figures are shown in 8.1. The similarities among all configurations are listed in Table 4.3.

Table 4.3: Results Similarity

NoC Size	Test bench	Results
4*4	TB1	Figure 4.4.
	TB2	Figure 4.5.
	TB3	No mapping result can be found.
5*5	TB1	Similar to Figure 4.4 with less search time.
	TB2	Hardly find mapping solution.
	TB3	No mapping result can be found.
6*6	TB1	Similar to Figure 4.4 with less search time.
	TB2	Figure 4.6.
	TB3	No mapping result can be found.
7*7	TB1	Similar to Figure 4.4 with less search time.
	TB2	Similar to Figure 4.6 with less search time.
	TB3	No mapping result can be found.
8*8	TB1	Similar to Figure 4.4 with less search time.
	TB2	Similar to Figure 4.6 with less search time.
	TB3	No mapping result can be found.
9*9	TB1	Similar to Figure 4.4 with less search time.
	TB2	Similar to Figure 4.6 with less search time.
	TB3	Figure 4.7.
10*10	TB1	Similar to Figure 4.4 with less search time.
	TB2	Similar to Figure 4.6 with less search time.
	TB3	Figure 4.8.

*Note: "No mapping result can be found" means no mapping solutions can ensure all tasks and flows being scheduled in 50 generations; "Hardly find mapping solution" means the mapping solutions can be found or cannot (if found, the generations will be very close to 50).

Table 4.4 illustrates the main information from experiment results. The number of generations required by PRE is greater than that needed by E2ERTA. This means the guiding performance of PRE as a fitness function

is worse than E2ERTA's. In addition, using PRE alone may cause the optimisation to get blocked at 20 (number of unshedulable tasks and flows), which can be seen in the TB2 with 4*4 NoC and Figure 4.5. With the number of unschedulable tasks and flows decreased, the guiding ability of PRE as a fitness function is reduced. As it cannot identify the difference among various candidates, the optimisation can hardly be proceeded further. Therefore, as discussed in the part on the limitations of PRE, only using PRE as a fitness function in search-based algorithms is not appropriate.

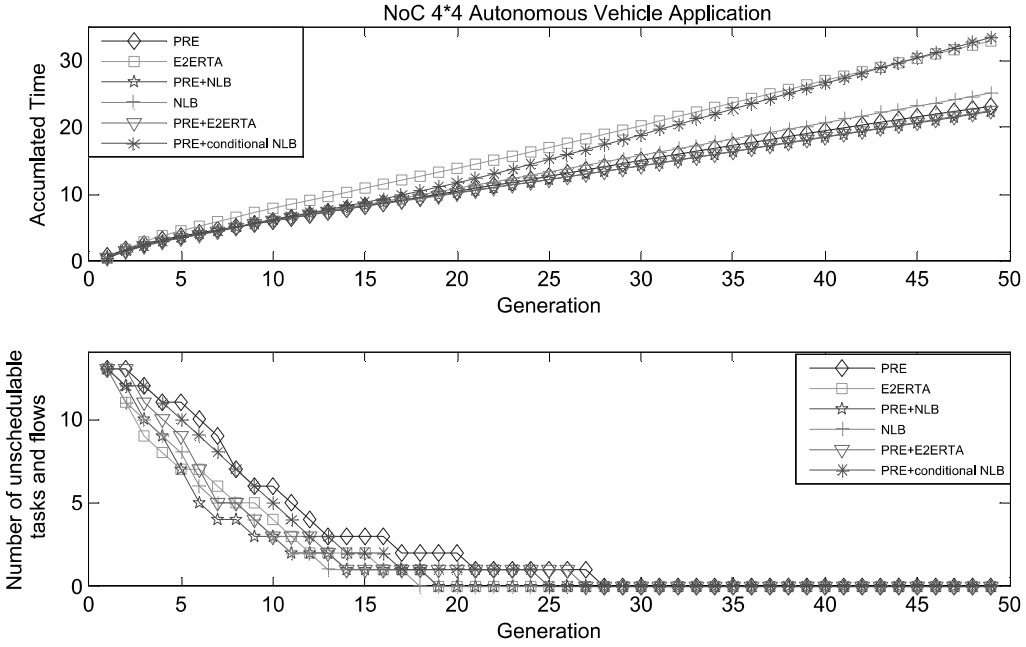

Figure 4.4: Autonomous Vehicle application on 4*4 NoC.

For other analysis composites, the number of generations required to reach the optimal is almost the same as with E2ERTA, which means they can provide the same guiding ability as E2ERTA. But, as the Acc column in Table 4.4 illustrates, they require significantly less search time than E2ERTA. This indicates that they have better timing performance. There is an exception in TB3 with 10*10 NoC shown in Figure 4.8. The PRE+E2ERTA seems to spend more time and generations. The reason for this is that PRE cannot guarantee the exact evaluation results. When this happens, the E2ERTA has to be triggered for an exact evaluation. We can notice that this situation happens

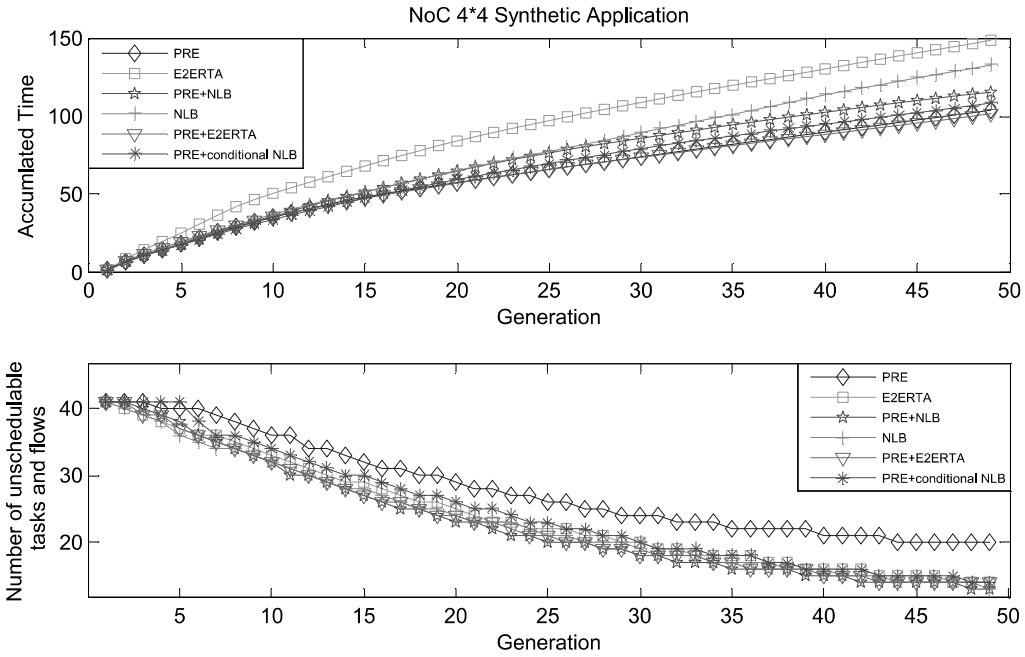

Figure 4.5: Synthetic application on 4*4 NoC.

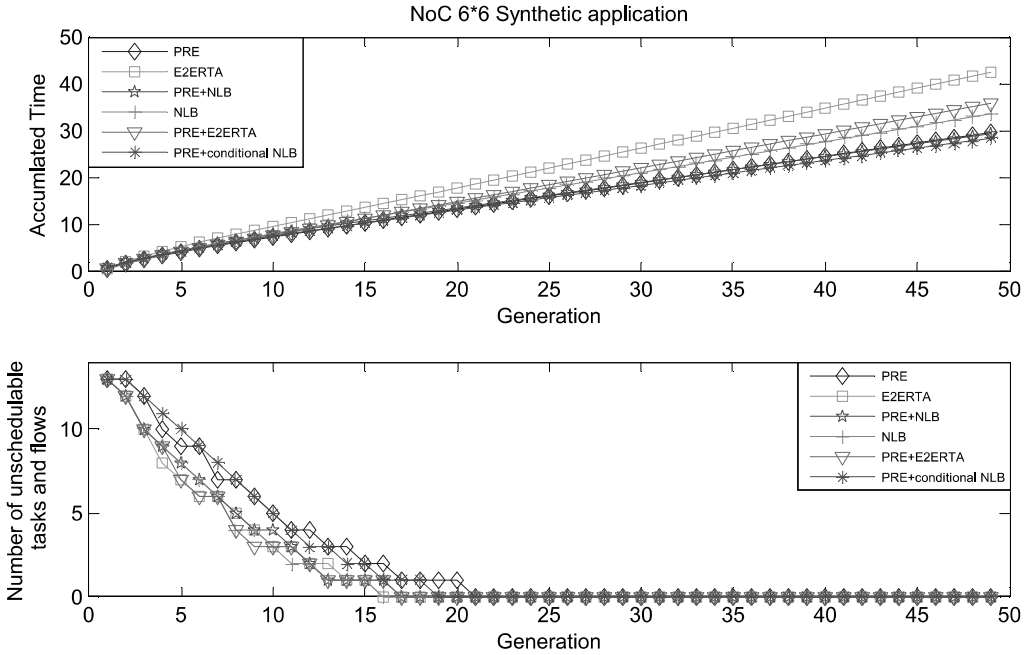

Figure 4.6: Synthetic application on 6*6 NoC.

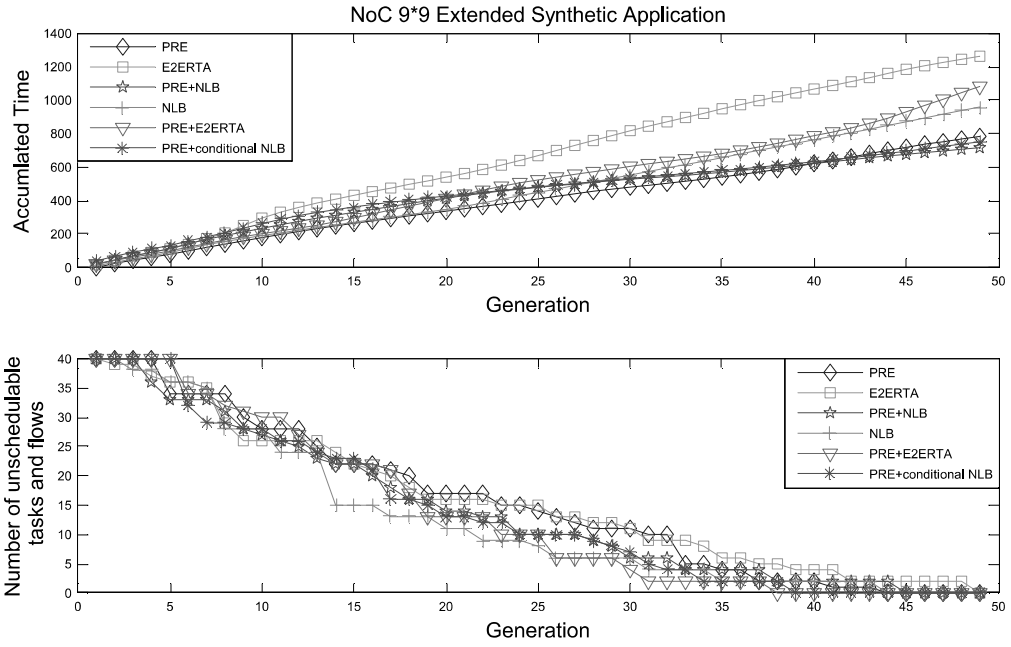

Figure 4.7: Extended Synthetic application on 9*9 NoC.

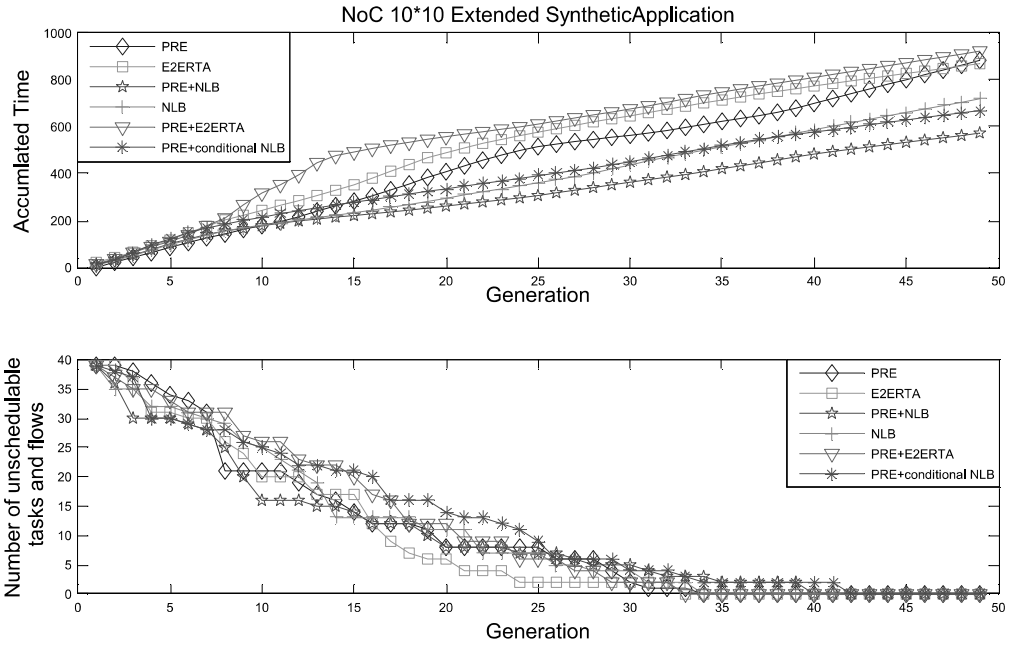

Figure 4.8: Extended Synthetic application on 10*10 NoC.

Table 4.4: Results Comparison

Schemes	NoC	TB	obj	Acc(s)	Gens	Per(%)
E2ERTA	4*4	TB1	0	13.27	19	0.00
PRE	4*4	TB1	0	14.09	28	27.95
NLB	4*4	TB1	0	9.85	18	21.65
PRE+E2ERTA	4*4	TB1	0	12.99	27	31.11
PRE+NLB	4*4	TB1	0	9.82	19	26.00
PRE+conditional NLB	4*4	TB1	0	15.17	25	13.12
E2ERTA	4*4	TB2	14	148.55	50	0.00
PRE	4*4	TB2	20	103.82	50	30.11
NLB	4*4	TB2	13	132.71	50	10.66
PRE+E2ERTA	4*4	TB2	14	101.58	50	31.62
PRE+NLB	4*4	TB2	13	115.39	50	23.32
PRE+conditional NLB	4*4	TB2	14	108.43	50	27.01
E2ERTA	6*6	TB2	0	14.47	16	0.00
PRE	6*6	TB2	0	13.83	21	27.18
NLB	6*6	TB2	0	11.98	16	17.21
PRE+E2ERTA	6*6	TB2	0	12.72	17	17.26
PRE+NLB	6*6	TB2	0	11.87	17	22.79
PRE+conditional NLB	6*6	TB2	0	12.53	19	27.08
E2ERTA	9*9	TB3	0	1261.65	49	0.00
PRE	9*9	TB3	0	699.87	44	38.22
NLB	9*9	TB3	0	848.96	44	25.06
PRE+E2ERTA	9*9	TB3	0	740.73	38	24.29
PRE+NLB	9*9	TB3	0	405.46	44	64.21
PRE+conditional NLB	9*9	TB3	0	620.26	39	38.23
E2ERTA	10*10	TB3	0	688.38	33	0.00
PRE	10*10	TB3	0	610.90	34	13.87
NLB	10*10	TB3	0	500.50	34	29.43
PRE+E2ERTA	10*10	TB3	0	719.34	33	-4.50
PRE+NLB	10*10	TB3	0	483.86	40	42.01
PRE+conditional NLB	10*10	TB3	0	598.71	42	31.66

*"TB": test bench; "obj": number of unschedulable tasks and flows; "Acc": accumulated time after the first generation; "Gens": number of generations used to achieve the corresponding "obj"; "Per": improved percentage of each generation compared with E2ERTA.

from generation 8 to 15, in which the accumulated time of PRE+E2ERTA was increased rapidly. Although the following execution speed is faster than E2ERTA, as can be seen from the drop of the curve, the overall performance is still affected by that situation.

Apart from PRE+E2ERTA, PRE+conditional NLB also costs more generations than E2ERTA in some cases, such as Figure 4.4, 4.6 and 4.8. The reason is that PRE+conditional NLB can tolerate five generations to PRE without any further optimisation before it starts NLB. Let us assume that there is still no suitable solution that can be found after five generations have already been tolerated. Then the NLB will be enabled. This may cause either number of generation or search time, or even both, to be more than E2ERTA. However, from the average time used by each generation, the PRE+conditional NLB is still more efficient than E2ERTA. To summarise, apart from PRE, all the other analysis composites require fewer generations or less execution time than E2ERTA in evaluating a mapping solution.

Furthermore, considering the Per column which is calculated by Equation 4.6, a higher percentage means less time is used in each generation by the analysis composites. It can be seen that the percentage of all five analysis composites is higher than that of E2ERTA. This shows that all these five analysis composites are faster than E2ERTA.

This phenomenon is significantly important in this research, which concerns a problem of dynamic task allocation for hard-real time NoC. Unlike the state-of-the-art, this research intends to apply a search-based algorithm as the dynamic mapper for a global task reallocation. As a fitness function in a search-based algorithm, any optimisation in the execution of E2ERTA will lead to a remarkable timing advantage to the overall search speed, since the fitness function will be loaded many times for evaluating each candidate. This is directly beneficial in the efficiency of dynamic mapping. In addition, a faster search means more candidates can be evaluated. It will provide a higher probability in finding a suitable mapping solution within a given time period.

$$
\begin{aligned}
Per =& \frac{t_{E2ERTA} - t_{scheme}}{t_{E2ERTA}} \\
=& 1 - \frac{Gens_{E2ERTA} * Acc_{scheme}}{Gens_{scheme} * Acc_{E2ERTA}}
\end{aligned}
\tag{4.6}
$$

- $Gens_{E2ERTA}$ is the number of generations required by E2ERTA;
- Acc_{scheme} is the time spent by the five schemes;
- $Gens_{scheme}$ is the number of generations required by the five schemes;
- Acc_{E2ERTA} is the time spent by E2ERTA;

4.3 Summary

In this chapter, we explored the complexity of E2ERTA, analysed the reasons for this complexity, and tried to alleviate this problem from an inexact view. Two improving techniques have been proposed and assembled with E2ERTA in five analysis composites to investigate the performance and coverage trade-off of an Inexact E2ERTA. Although PRE may have potential deficiency when only using it as a fitness function in a search-based algorithm, it provides good performance in search speed and in reducing the search area. The other four analysis composites considered have the same guiding ability as E2ERTA but use significantly less search time. This improvement is directly beneficial in both the search speed and the success possibility of optimisation, and indirectly in favour of the efficiency aspect of the research problem. However, the techniques discussed in this section only focus on the complex iterative calculation of E2ERTA, and improve it from a theoretical direction. Other possible directions may exist, and one of these will be discussed in the following chapter 5.

Chapter 5

Hardware Accelerated Inexact End-to-End Response Time Analysis

Following the discussion in Chapter 4, it is clear that the efficiency of fitness functions can directly affect the search efficiency of applying a search-based algorithm for dynamic task allocation problems in hard real-time systems. In Chapter 4, an inexact E2ERTA was proposed to alleviate the complexity of calculation in E2ERTA from a theoretical point of view, in order to make it a suitable fitness function in fast mapping searches. However, the question remains as to whether there are any other issues affecting the efficiency of E2ERTA. One possible issue is the current implementation method. This is discussed in this chapter and improved by analysis and accelerator designs. The chapter begins with an analysis of the existing implementation limitations and move on to an accelerated architecture implementation. Thereafter, this architecture will also be further improved by two accelerators, which inherit the techniques from the previously discussed inexact E2ERTA.

5.1 Implementation Methods

The efficiency of a start-of-the-art E2ERTA can suffer limitations from the implementation method. Currently, most E2ERTAs are implemented on a software platform (SW-E2ERTA) on which the E2ERTA is executed sequentially (such as [101]). The flow chart of a software implementation of an E2ERTA is shown in Algorithm 1

Algorithm 1 Software Version E2ERTA Working Process

1: **procedure top** ▷ E2ERTA
Require: Tasks' allocation, Tasks' information, Flows' information;
Ensure: Number of Unscheduled Tasks and Flows.
2: Task Response Time Analysis
3: Flow Response Time Analysis
4: Normalizing Results
5: ———————————————————————————
6: **sub procedure 1** ▷ Task Response Time Analysis
Require: Tasks' allocation, Tasks' information;
Ensure: Tasks' Response Time.
7: Get Task Interference
8: Get Task Response Time
9: **sub procedure 2** ▷ Flow Response Time Analysis
Require: Tasks' Response Time, Flows' information;
Ensure: Flows's Response Time.
10: Flow Routing
11: Get Flow Basic Latency
12: Get Direct Interference Set
13: Get Indirect Interference Set
14: Get Flow Response Time
15: **sub procedure 3** ▷ Normalizing Results
Require: Tasks' Response Time, Flows' Response Time;
Ensure: Number of Unscheduled Tasks and Flows.
16: **repeat**
17: **if** $r_i + R_i > D_i$ **then**
18: $Number_{\ Unscheduled\ Tasks\ and\ Flows}$ is increased.
19: **end if**
20: **until** all tasks have been normalized.

Note: Assuming each task only have one flow and $d_i = D_i$.

From the working process, it can be seen that generally the E2ERTA is executed in a sequential order. Before the final E2ERTA result of a given task set can be normalised, it is necessary to analyse the response time for tasks and flows. In other words, line 2 and 3 are necessary. The two sub-procedures are related as well, but this relationship is not always close. The result of sub-procedure 1 is only used in the last step (line 14) in sub-procedure 2. In other steps they are not related to each other. That is to say, theoretically, sub-procedure 1 and 2 could be partially executed in two parallel arms, but the existing E2ERTA software implementation architectures are not designed to support this. The next computing block cannot start until the previous block has finished. Hence, the existing software implementation architectures result in a low efficiency of E2ERTA.

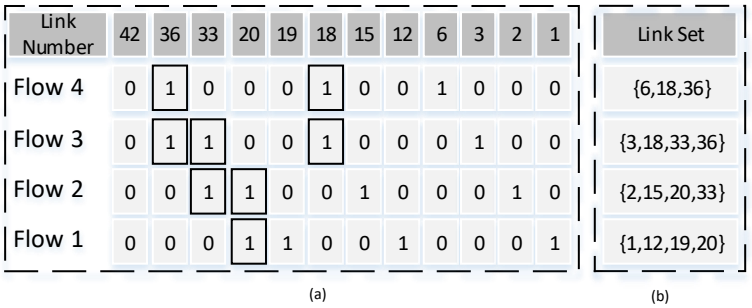

Figure 5.1: (a) Binary Coding Example, (b) Integer Coding Example

Moreover, the efficiency of operating vectors (bit vector comparison and logic operations such as 'and' and 'or') in software is low. For example, if we use the binary coding style to encode the results of the Routing Algorithm, the results could be similar to those in Figure 5.1a which follows the example in Figure 2.8, page 41. The hidden links which are not used are set to 0. To identify the relationship between $Flow_3$ and $Flow_4$, the SW-E2ERTA is required to compare these two flows bit by bit. Normally this comparison procedure will take multiple clock cycles, and the number of clock cycles will increase along with the size of the NoC. Because a larger size implies more links, it will result in more computation time. Even if integer coding is used (an example is presented in Figure 5.1b), the computation time would not be reduced significantly. A similar phenomenon can also be found in Get

Indirect Interference Set (line 13) and Get Flow Basic Latency (line 11).

In software field, some programs can be improved by applying advanced supports such as parallel computing, Single Instruction Multiple Data (SIMD) and Graphics Processing Unit (GPU). E2ERTA, however, is not one of such kind of programs. The reason is determined by the characteristics of E2ERTA and its target (NoCs), and can be understood as follow. E2ERTA is mainly based on an iterative calculation. Only partial processes of it can be parallelized. So, fully parallel computing is not realistic for E2ERTA. SIMD and GPU can be alternative solutions for improving the efficiency of software, however, they may be inappropriate or over expensive for E2ERTA. The data size of SIMD or GPU is determined by manufacturers. They are not fully flexible for users. But the number of links of a NoC can be very variable. This phenomenon becomes a problem for designers to find a general architecture for meeting variable size of NoCs. Besides, using SIMD or GPU to enhance E2ERTA is too expensive. For example, although the logic operations of vectors can be well finished by SIMD or GPU, the hardware costs of SIMD or GPU are much more than several logic gates which can also optimise E2ERTA. Therefore, the existing software implementation and optimisation would lead to a negative effect on the efficiency of E2ERTA.

5.2 Hardware E2ERTA

In order to alleviate the implementation limitations of E2ERTA, we discuss using hardware based methods to improve E2ERTA, introducing a hardware architecture named HW-E2ERTA. In this section, the implementation details are discussed, followed by the experiment and performance evaluation.

5.2.1 Architecture of HW-E2ERTA

In the E2ERTA calculation process, not all computation processes have to be launched sequentially. As discussed in Subsection 5.1, the Task Response Time Analysis and partial steps of Flow Response Time Analysis can be loaded simultaneously. Thus, a hardware implemented architecture is a possible solution for parallelisation of E2ERTA, which is shown in Figure 5.2. In this Figure, the Task and Flow Response Time Analysis can be released at the same time. In the Flow Response Time Analysis process, the two steps (Get Direct Interference Set and Get Flow Basic Latency) can be executed simultaneously, as soon as they receive the results from the Routing Algorithm. The Get Flow Response Time component will be launched when the Task Response Time Analysis, Get Indirect Interference and Get Flow Basic Latency components are finished. Its results and those from Task Response Time Analysis can then be gathered and organised.

5.2.2 Hardware Accelerated Vector Operator

As mentioned previously, vector operation is also a bottle-neck that needs to be resolved to make a breakthrough. One possible method is using logic gates to accelerate the vector operation. An example is shown in Figure 5.3.

In Figure 5.3a, the routing results follow the results in Figure 5.1. The width of the interference vector is 4. The right end of the interference vector represents $Flow_1$. The flow with value '1' ($Flow_3$) refers to the fact that this flow can interrupt the observed flow ($Flow_4$).

For Get Direct Interference Set step, the 'and' gate is used to identify the direct relationship between two flows (shown in Figure 5.3a). The logic 'and' operation is applied between the routing results of $Flow_4$ and $Flow_3$. If these two flows have shared links as labeled in block rectangles, the result is not all zeros. The relevant bit position is set to '1' in $Flow_4$'s Direct Interference

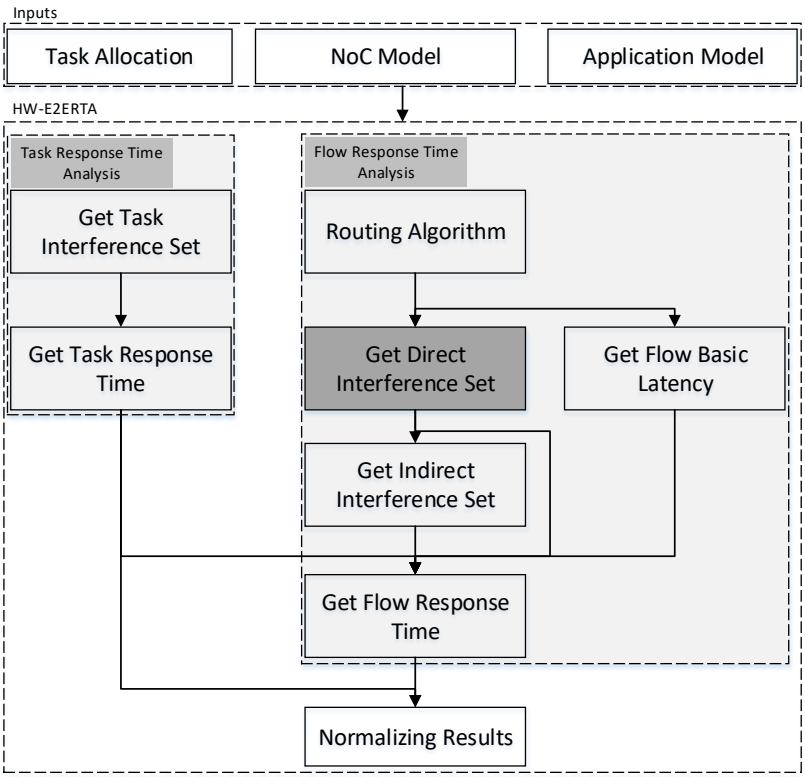

Figure 5.2: HW-Architecture of E2ERTA.

Set. Otherwise, the result is all zeros and the relevant bit is set to '0'.

When identifying the indirect relationship between two flows, the logic 'or' gate and the logic 'xor' gate are selected, which is described in Figure 5.3b and 5.3c. The example in Figure 5.3b inherits the sources from Figure 2.8, page 41, Figure 5.1 and Figure 5.3a. In this example, $Flow_4$'s Direct Interference Set is one of the inputs of the 'or' gate and 'xor' gate. If $Flow_3$ can directly interrupt $Flow_4$, $Flow_3$'s Direct Interference Set will always be checked, regardless of whether $Flow_3$'s Direct Interference Set is empty (cannot be preempted by other flows) or not. Therefore, the other input of the 'or' gate is $Flow_3$'s Direct Interference Set. Similar to Get Direct Interference operation, if the result is not all zeros, the relevant bit will be set to '1' to indicate that the higher priority flow ($Flow_2$) can indirectly interrupt $Flow_4$. However, if a flow can preempt both $Flow_3$ and $Flow_4$, the result will remain as 0. This

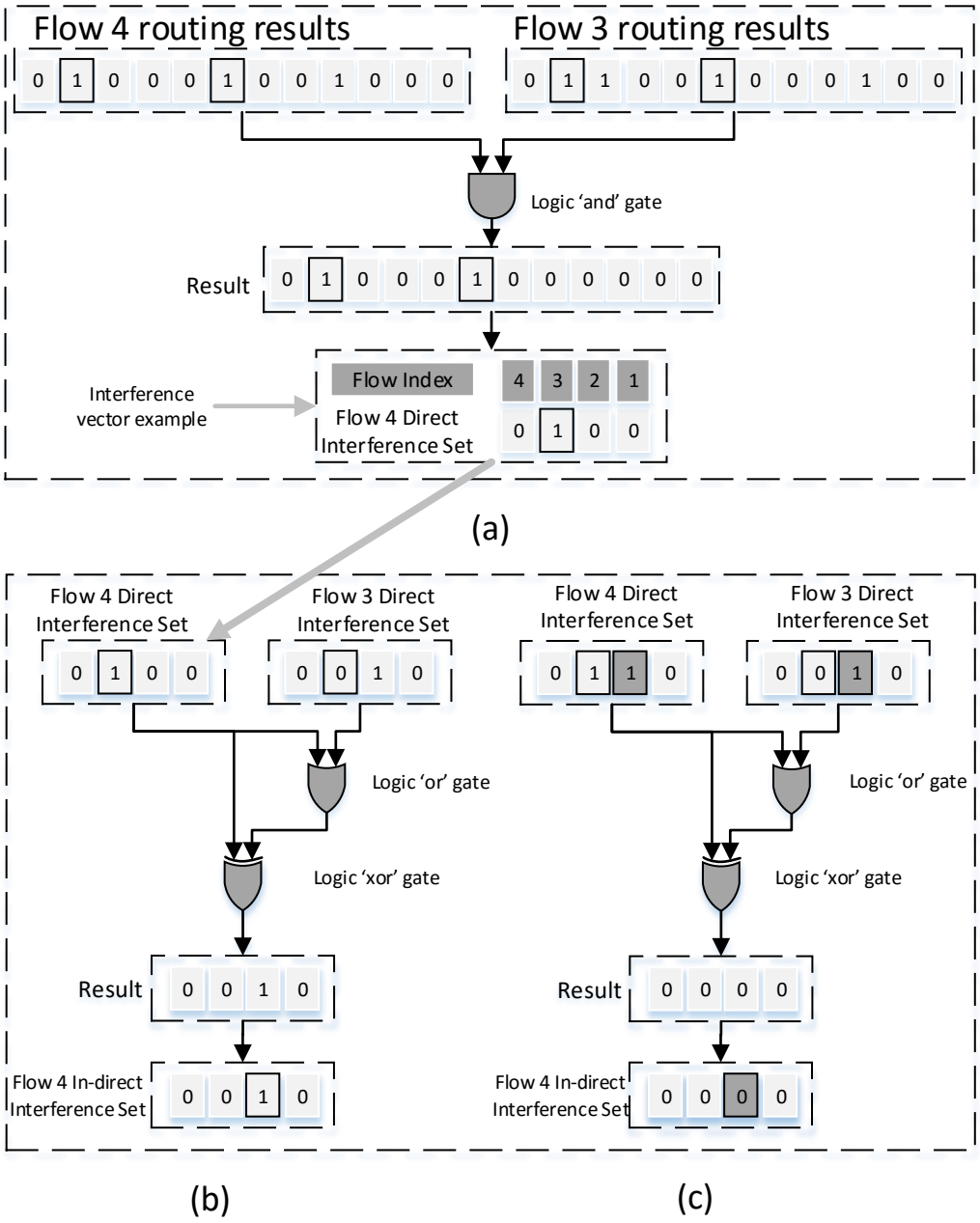

Figure 5.3: (a) Example of Get Direct Interference Hardware Implementation Operation, (b) and (c) Examples of Get Indirect Interference Hardware Implementation Operation.

phenomenon is illustrated in Figure 5.3c; $Flow_2$ labelled with dark gray is the flow that can preempt both $Flow_3$ and $Flow_4$.

5.2.3 Experiment and Results Analysis

To evaluate the performance of HW-E2ERTA (implemented on a FPGA platform), an experiment is established. In this subsection, the experiment platform will first be described, followed by the experiment configuration. The results will be discussed at the end.

Experiment Platform

To evaluate the performance of HW-E2ERTA, an experiment platform is used. This is an embedded system implemented on Xilinx VC707 development board shown in Figure 5.4a.

On this platform, first the SW-E2ERTA [58] (as a base line) is fully implemented on a MicroBlaze, exactly following the instructions of [58] in language C and compiled using the C compiler of GNU version 2.16. After that, the HW-E2ERTA is implemented with VHDL (Very-High-Speed Integrated Circuit Hardware Description Language) and compiled by Xilinx Vivado 14.3. Then, the HW-E2ERTA is mounted on an AXI bus (an on-chip interconnect link used in Xilinx system-on-chip design) through an AXI bus interference. The MicroBlaze is the main testing controller on this platform. It is in charge of benchmark generating, executing SW-E2ERTA, loading and collecting data from HW-E2ERTA and final results summarizing.

A hardware timer (36-bit) is introduced to gain an accurate execution of the computation time of SW-E2ERTA in the number of clock cycles. At the same time, we consider the possibility of an overflow. From the results in Table 4.4, page 103, we estimate the number of clock cycles used by E2ERTA on the software platform. The worst case (1261.62 seconds) happens when NoC

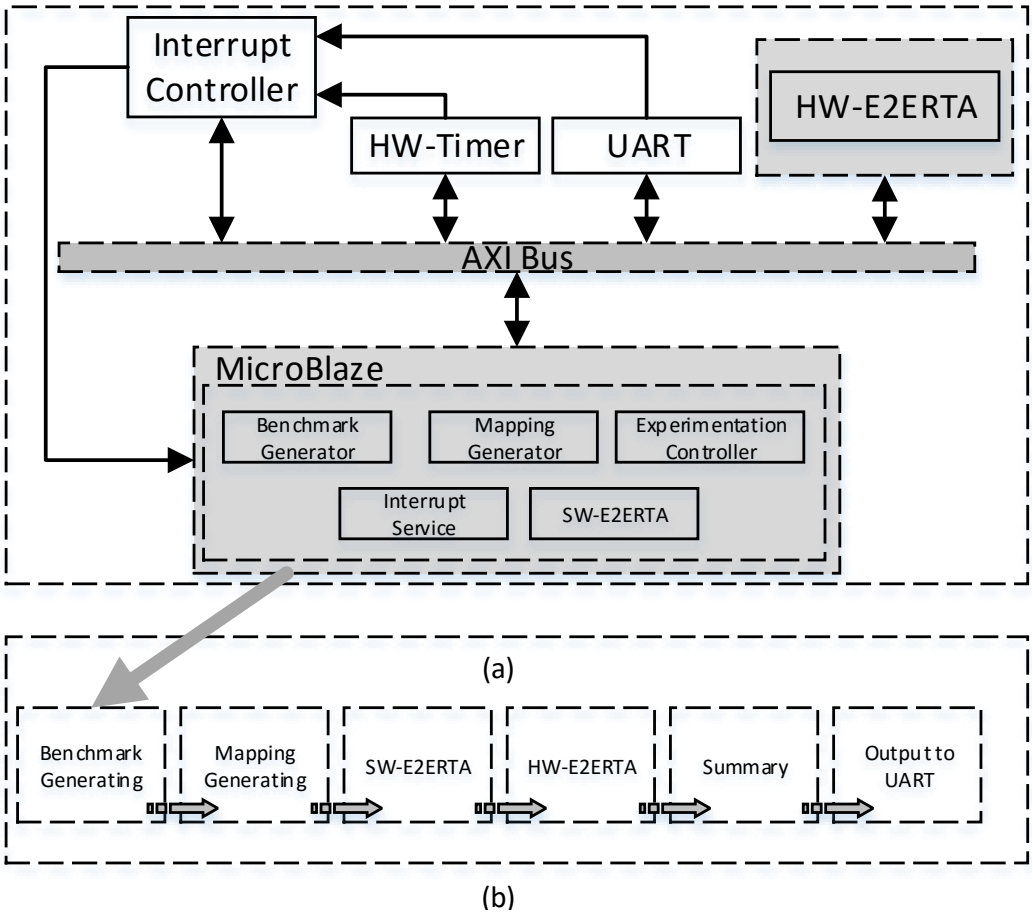

Figure 5.4: (a) Experiment Platform, (b) Testing Process.

size is 9*9 and benchmark is TB3. By applying this case through Equation 5.1, we can generally estimate the number of clock cycles as $8.754 * 10^8$. This value contains not only the partial execution time of other operations such as mutation, crossover and selection, but also the running time of operating system (Windows 7) functions. Although this value is not exact, it can still be used as a reference for us to select the range for the hardware timer. 2^{32} is larger than $8.754 * 10^8$, but not tenfold larger ($8.754 * 10^9$). So a single 32-bit counter may be not suitable in all cases. Thus, to enable the use of more complex applications, another 4-bit timer is added as a backup, thereby introducing 36-bits of fidelity.

$$
\begin{aligned}
No._{clock\ cycle} &= frequency\ system * execution\ time \\
&= frequency\ system * \frac{Acc}{Gens * Population\ size} \\
&= 3.4 * 10^9 * \frac{1261.62}{49 * 100} \\
&= 8.754 * 10^8
\end{aligned}
\tag{5.1}
$$

Each test starts with the benchmark generating process and ends when all processes or components are tested. Figure 5.4b shows the testing process. The MicroBlaze firstly generates a Synthetic benchmark which includes task parameters and flow parameters and a random task mapping. Then the MicroBlaze launches SW-E2ERTA. When SW-E2ERTA has finished, the MicroBlaze loads the testing data to HW-E2ERTA and enables it. After all tests have finished, the MicroBlaze collects data from HW-E2ERTA and organises these results. The results are output through a UART port.

Experiment Configuration

To measure the performance of HW-E2ERTA, the experiments are configured as follows:

- NoC platform configuration:
  - the system clock speed is 50Mhz,
  - four NoC size configurations, 3*3, 4*4, 5*5, 10*10, are enough to show the improvement increasing with the expansion of the NoC size;
- Benchmark configuration:
  - four task set size configurations, 16, 32, 64, 128, are enough to show the affection form the number of task to evaluation exectuion

time,

  - the priority of a task is increased with its task index, e.g. the lowest priority task is $Task_1$,
  - the utilisation of task and flow is from 10% to 90%,
  - each task is considered to generate one flow which inherits the priority of the task;

- VHDL compiler is Xilinx Vivado 14.3.

Because each experiment will generate a random mapping and synthetic benchmark, one time testing does not illustrate the difference between the HW-E2ERTA and SW-E2ERTA. Therefore, the number of testing times is increasedto 1,000,000, in order to obtain a better coverage.

Results Analysis

The results from the experiment, are organised as follows: Figure 5.5 shows comparison over various experiment configurations, while more details are shown in Table 5.1; in Figure 5.5, the top half and bottom half present the SW-E2ERTA and HW-E2ERTA respectively. The Y-axis shows the average numbers of clock cycles used to finish an E2ERTA computation. Because the numbers are large, they are arranged in log_{10} scale.

For SW-E2ERTA, it can be seen that the larger the size of NoC or the number of tasks the SW-E2ERTA has to calculate, the longer the evaluation time required. However, the influence from utilisation of task and flow is in a parabola instead of linear. This can be seen from the examples in Table 5.1, which are labelled in gray. The reason for this phenomenon is that the extremely low or high utilisation can terminate the iterative calculation of E2ERTA early. We can make an assumption as follows:

- the current observed objective (flow as an example) is i,

- the number of iterative calculations required by lower, moderate and higher utilszation are N_{lower}, $N_{moderate}$ *and* N_{higher} respectively,
- the number of clock cycles used to finish a single iterative calculation is nearly the same or equal to N_{Sic}.

When the utilisation is extremely low, the E2ERTA may be terminated by $R_i^{n+1} = R_i^n$ within a very few iterations (two or three). The calculation will become more difficult as the utilisation increases and result in more iterations being required. However, when the utilisation becomes extremely high, the calculation complexity of E2ERTA will decrease. This is because E2ERTA can easily be determined and the observed objective will miss the deadline by finding $R_i \geq D_i$ within a few iterations. Therefore, we get the inequality $N_{lower} \leq N_{moderate} \geq N_{higher}$ and further obtain the total execution time, which is $N_{lower} * N_{Sic} \leq N_{moderate} * N_{Sic} \geq N_{higher} * N_{Sic}$. Thus, the influence from utilisation follows a parabola style.

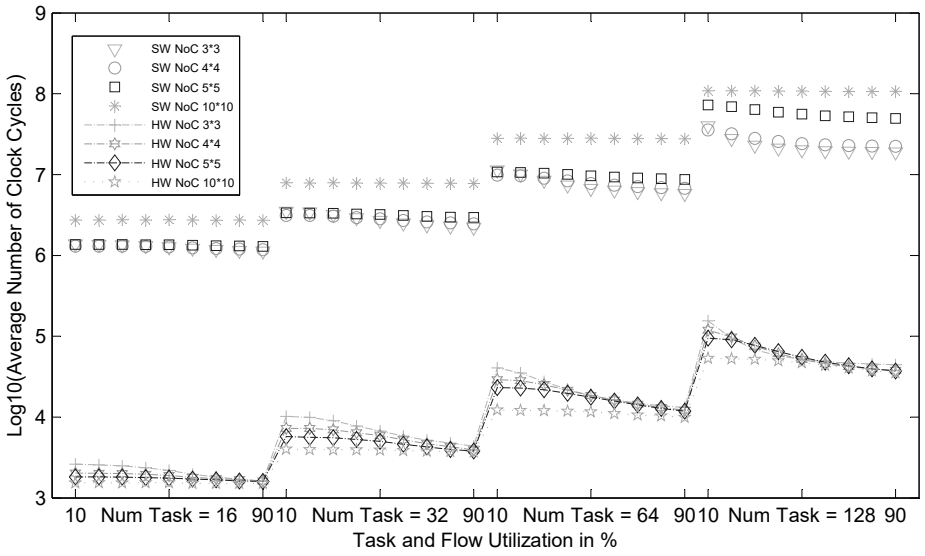

Figure 5.5: The influence from NoC size, Number of Tasks and Utilisation for E2ERTA

Note: Num, SW and HW refer to Number, on Software Platform and on Hardware Platform respectively.

Table 5.1: HW-E2ERTA vs SW-E2ERTA Results Table.

		3*3			4*4			5*5			10*10		
No. Task	U(%)	SW	HW	Improve	SW	HW	Improve	SW	HW	Improve	SW	HW	Improve
	10	6.140	3.417	2.723	6.113	3.308	2.805	6.134	3.261	2.873	6.437	3.185	3.252
	20	6.136	3.409	2.727	6.113	3.305	2.808	6.135	3.261	2.874	6.438	3.190	3.248
	30	6.130	3.398	2.732	6.113	3.301	2.812	6.135	3.256	2.879	6.440	3.186	3.254
	40	6.120	3.373	2.747	6.110	3.292	2.818	6.132	3.250	2.882	6.438	3.189	3.249
16	50	6.104	3.339	2.765	6.104	3.278	2.826	6.131	3.244	2.887	6.442	3.188	3.254
	60	6.087	3.294	2.793	6.094	3.255	2.839	6.126	3.232	2.894	6.438	3.181	3.257
	70	6.073	3.260	2.813	6.084	3.238	2.846	6.121	3.223	2.898	6.436	3.180	3.256
	80	6.060	3.230	2.830	6.079	3.227	2.852	6.117	3.209	2.908	6.438	3.181	3.257
	90	6.053	3.210	2.843	6.071	3.209	2.862	6.112	3.202	2.910	6.435	3.179	3.256
	10	6.544	4.007	2.537	6.496	3.861	2.635	6.526	3.758	2.768	6.898	3.604	3.294
	20	6.537	3.998	2.539	6.494	3.859	2.635	6.523	3.750	2.773	6.896	3.598	3.298
	30	6.506	3.955	2.551	6.487	3.842	2.645	6.520	3.743	2.777	6.898	3.596	3.302
	40	6.466	3.890	2.576	6.469	3.807	2.662	6.513	3.724	2.789	6.897	3.597	3.300
32	50	6.431	3.827	2.604	6.453	3.770	2.683	6.506	3.699	2.807	6.897	3.594	3.303
	60	6.399	3.766	2.633	6.432	3.719	2.713	6.494	3.664	2.830	6.896	3.587	3.309
	70	6.377	3.714	2.663	6.413	3.669	2.744	6.484	3.630	2.854	6.893	3.578	3.315
	80	6.362	3.678	2.684	6.403	3.633	2.770	6.473	3.598	2.875	6.894	3.575	3.319
	90	6.347	3.641	2.706	6.391	3.594	2.797	6.470	3.580	2.890	6.891	3.568	3.323
	10	7.055	4.608	2.447	6.994	4.462	2.532	7.031	4.364	2.667	7.449	4.088	3.361
	20	7.005	4.546	2.459	6.986	4.451	2.535	7.027	4.358	2.669	7.450	4.083	3.367
	30	6.929	4.438	2.491	6.955	4.400	2.555	7.016	4.337	2.679	7.448	4.080	3.368
	40	6.870	4.341	2.529	6.921	4.333	2.588	7.002	4.295	2.707	7.448	4.074	3.374
64	50	6.828	4.264	2.564	6.891	4.268	2.623	6.984	4.245	2.739	7.450	4.064	3.386
	60	6.806	4.212	2.594	6.870	4.211	2.659	6.970	4.198	2.772	7.447	4.045	3.402
	70	6.788	4.173	2.615	6.852	4.160	2.692	6.959	4.150	2.809	7.445	4.027	3.418
	80	6.774	4.140	2.634	6.839	4.117	2.772	6.948	4.105	2.843	7.444	4.013	3.431
	90	6.766	4.119	2.647	6.829	4.086	2.743	6.941	4.074	2.867	7.443	3.996	3.447
	10	7.602	5.192	2.410	7.553	5.074	2.479	7.860	4.977	2.883	8.035	4.726	3.309
	20	7.445	4.974	2.471	7.503	4.993	2.510	7.840	4.956	2.884	8.036	4.722	3.314
	30	7.363	4.834	2.529	7.447	4.879	2.568	7.805	4.887	2.918	8.035	4.715	3.320
	40	7.325	4.758	2.567	7.409	4.783	2.626	7.773	4.809	2.964	8.032	4.697	3.335
128	50	7.305	4.714	2.591	7.383	4.712	2.671	7.749	4.739	3.010	8.032	4.673	3.359
	60	7.294	4.687	2.607	7.367	4.658	2.709	7.728	4.680	3.048	8.030	4.641	3.389
	70	7.288	4.669	2.619	7.359	4.623	2.736	7.715	4.634	3.081	8.028	4.607	3.421
	80	7.284	4.657	2.627	7.353	4.598	2.755	7.704	4.597	3.107	8.027	4.574	3.453
	90	7.280	4.648	2.632	7.349	4.579	2.770	7.694	4.570	3.124	8.029	4.549	3.480
Average				2.625			2.701			2.865			3.333

Note: SW refers to SW-E2ERTA, HW refers to HW-E2ERTA and
U refers to the utilization of task or flow.

Similar to SW-E2ERTA, the HW-E2ERTA can also be influenced by NoC size, number of tasks and utilisation. There are, however, also some differences. If only the NoC size is considered and the number of tasks and utilisation are fixed, we find that the larger the size of the NoC, the less evaluation time is needed. To understand this performance, we can make an assumption as follows:

- the current observed objective (flow as an example) is i,
- the number of clock cycles used to find both direct and indirect interference set is N_{di},
- the number of clock cycles used to calculate response time is N_{crt},
- the total number of clock cycles used to finish E2ERTA is N_{E2ERTA}.

Reducing the NoC size can directly affect the N_{crt} in both SW-E2ERTA and HW-E2ERTA, when the number of tasks and utilisation are fixed. A smaller NoC means limited resources will be shared by more tasks and flows. This directly causes more iterations to be required during the calculation. In other words, the calculation will become more complex. For N_{di}, reducing the NoC size can only affect the SW-E2ERTA. A smaller NoC will have fewer links and require a smaller vector to encode the results of the Routing Algorithm. This can directly reduce the number of clock cycles used to identify the relation between flows in bit-by-bit comparison. However, changing the NoC size will not affect N_{di} in HW-E2ERTA, because the method used to identify the relationship among flows in HW-E2ERTA are logic operations. Regardless of the size of the analysed NoC, the result will always obtained within a single clock cycle.

In SW-E2ERTA, with the NoC enlarged, the contention becomes mitigatory. Therefore, calculation complexity can be alleviated and thus N_{crt} goes down; while N_{di} will increase quickly, as there are more links to be checked. The relationship between the number of links and the size of a 2D mesh based

Table 5.2: The Influence from Number of Tasks and Utilization Explanation Table

	NumTask	U	NoC size	N_{crt}	N_{di}	N_{E2ERTA}
HW	-	-	$\Uparrow$	$\Downarrow$	-	$\Downarrow$
	-	-	$\Downarrow$	$\Uparrow$	-	$\Uparrow$
SW	-	-	$\Uparrow$	$\Downarrow$	$\Uparrow$	$\Uparrow$
	-	-	$\Downarrow$	$\Uparrow$	$\Downarrow$	$\Downarrow$

Note: SW, HW, U, NumTask refers to SW-E2ERTA, HW-E2ERTA, utilization of task or flow and number of task respectively.

NoC can be calculated by Equation 5.2. As the mapping used is randomly generated, the benefit from mapping improvement is very limited. Although the N_{crt} can be reduced, its magnitude is not as significant as N_{di}. Thus, the influence in SE-E2ERTA is opposite to that of HW-E2ERTA.

$$Number\ of\ links = 4 * size^2 - 4 * size \tag{5.2}$$

Focusing on the Table 5.1, the results show that the hardware version is up to 3020 ($10^{3.480}$) times faster than the software version. If we further compared with the best software version Inexact E2ERTA, which is illustrated in Chapter 4, the HW-E2ERTA is still much faster than it. This can be seen from the comparision bellow. The best Inexact E2ERTA achieved in pervious chapter is PRE+NLB which can improve the software version E2ERTA (implemented on a PC) by 64.21%, shown in Table 4.4. It is tested on a 9*9 NoC with TB3 (100 Tasks with average utilisation 41.3%). Its average execution time of SW-E2ERTA on a PC can be generally estimated by Equation 5.3.

$$\frac{Accumulated\ time}{Number\ of\ Generations\ * Population\ Size} = \frac{405.6}{44 * 100} \approx 92.2\, milliseconds \tag{5.3}$$

An example with harder configuration can be found in Table 5.1. The HW-E2ERTA is tested on a 5*5 NoC with 128 number of tasks which average utilisation is around 50%. The execution time of HW-E2ERTA can be calculated by Equation 5.4. It shows advantage of HW-E2ERTA.

$$\frac{Number\ of\ Clock\ Cycle}{Working\ Frequency} = \frac{10^4.712}{100 * 10^6} \approx 1 * 10^{-0.288}\ milliseconds \quad (5.4)$$

5.2.4 Summary

In this section, we propose a hardware accelerated architecture and a vector processing accelerator to implement E2ERTA in hardware. We compare the HW-E2ERTA and SW-E2ERTA in terms of number of clock cycles. The results show that the hardware version is up to 3020 ($10^{3.480}$, by experiments) times faster than the software version (implemented on MicroBlaze in C compiled by the C compiler of GNU version 2.16). In addition, we generally eastimate the execution time of the best Inexact E2ERTA we proposed in previous chapter, which is implemented on a PC. The comparison between the estimation and a result of HW-E2ERTA shows the HW-E2ERTA is still advanced. Furthermore, the benefit brought by the hardware implementation will increase with the size of the NoC and the complexity of the applications.

5.3 Inexact HW-E2ERTA

As discussed in Section 4.1 and Section 5.1, the efficiency of the E2ERTA can be affected by both the characteristic of E2ERTA and the implementation method. Section 4.2 suggested an inexact E2ERTA to alleviate the limitation of the characteristic of E2ERTA on a software platform, while Section 5.2 tried to accelerate E2ERTA by hardware implementation. However, the techniques of these two sections can be combined. In this section, we explore

the possibility of combining the two sections to implement an Inexact HW-E2ERTA, in order to obtain further optimisation on the E2ERTA. This section is organised as follows: hardware accelerated inexact components; assembly schemes; experiments and results analysis.

5.3.1 Hardware Accelerated Inexact Components

The Inexact E2ERTA proposed in Section 4.2 tried to improve E2ERTA by applying either boundary to the response time or a larger initial value. However, in hardware implementation, the original equations used in Inexct E2ERTA (Equation 4.5, page 91 and 4.4, page 90 for task and flow boundaries, Equation 4.5a and 4.4a for task and flow new initial value) are not suitable for direct application, considering the hardware resource use. This is because some of these equations could be replaced by others with less complex computations, but relatively more uncertentity. Thus, in this subsection, how to select suitable equations for boundaries and initial values will first be discussed. The normalisation of the selected equations and how to implement them in hardware is then introduced.

Boundary Selection

The selected boundaries for a task in the software implementation are Equations 4.5a and 4.5b. The lower bound found by Equation 4.5a is selected from a seria of lower bounds, however it increases the complexity of computation.

In contrast, the method for finding the lower bound of flows (Equation 4.4a) is much simpler. It is oritinted by replacing the celling function in E2ERTA. By applying this idea, we can find a lower bound equation Equation 5.5a which is much simllar with Equation 4.4a (r_j *and* J_j^I do not exist and can be set to zero during task lower bound calculation). Although, the lower bound found by this idea may not be the max one, the computation complexity can be reduced. Thus, we can consider using Equation 5.5a as our task lower

bound in hardware implementation.

Moving to the upper bound, the two calculation methods (Equations 4.5b and 4.4b) are similar. In addition, Equation 4.5b providers a tighter boundary and so we can keep it as the upper bound. Therefore, our new boundary for tasks are Equations 5.5a and 5.5b.

$$r_i^{lb} \geq \frac{c_i}{1 - \sum_{\forall j \in hp(j)} u_j} \tag{5.5a}$$

$$r_i^{ub} \leq \frac{c_i + \sum_{\forall j \in hp(j)} c_j(1 - u_j)}{1 - \sum_{\forall j \in hp(j)} u_j} \tag{5.5b}$$

The flow boundary can be found using Equations 4.4a and 4.4b. The input elements of Equation 4.4a are the same as the inputs of 4.4b. Therefore, combining these two equations can calculate the upper bound and lower bound simultaneously. Thus, we retain Equations 4.4a and 4.4b as our boundary of the flow response time.

Equation Normalisation

In order to minimise the cost of hardware resources, it is necessary to design a reusable calculation unit by abstracting the similarities among Equations 5.5a, 5.5b and Equations 4.4a, 4.4b. We observe that if we do not classify whether an equation is for task or flow, we can use a general equation to represent all four of these equations with the assumptions listed below:

- assumption of variables α_x and α_y:
  - $\alpha_y = \left(r_j + J_j^I\right)$,
  - for task upper bound calculation $\alpha_x = c_j(1 - u_j)$,
  - for flow upper bound calculation $\alpha_x = \alpha_y U_j + C_j$,

  - for task lower bound calculation $\alpha_x = \alpha_y U_j$,
  - for flow lower bound calculation $\alpha_x = \alpha_y U_j$;

- Equation 4.4a, 4.4b and 5.5a, 5.5b can be represented by

$$r_i = \frac{c_i + \sum_{\forall j \in hp(j)} \alpha_x}{1 - \sum_{\forall j \in hp(j)} u_j} \tag{5.6a}$$

$$R_i = \frac{C_i + \sum_{\forall j \in S_{id}} \alpha_x}{1 - \sum_{\forall j \in S_{id}} U_j} \tag{5.6b}$$

- $\sum_{\forall j \in S_{id}}$ and $\sum_{\forall j \in hp(j)}$ are similar and can be implemented by same structure,

- the calculation process of Equation 5.6 can be divided into four stages:

  stage 1 calculate $\sum_{\forall j \in S_{id}} \alpha_x$ or $\sum_{\forall j \in hp(j)} \alpha_x$,

  stage 2 $\sum_{\forall j \in S_{id}} U_j$ or $\sum_{\forall j \in hp(j)} u_j$,

  stage 3 the dividend and divisor,

  stage 4 R_i or r_i.

It can be seen that the last two stages are common calculations. Thus, we can abstract the first two stages as a black box (named α box). The data flow of Equation 5.6 a and b in hardware can be represented as Figure 5.6a, where the α box is used to deal with accumulation calculations. The α box will be repeated until all elements $j \in S_{id}$ *or* $hp(j)$ have been checked. Since the calculation for task and flow will not be executed at the same time, as seen in Figure 5.2, page 111, α box can be reused (either for tasks or flows) in HW-E2ERTA.

α Box Design in Hardware

Figure 5.6b shows the design structure of α box in hardware. The working process of α box can be divided into three steps. There are two reasons for this decomposition.

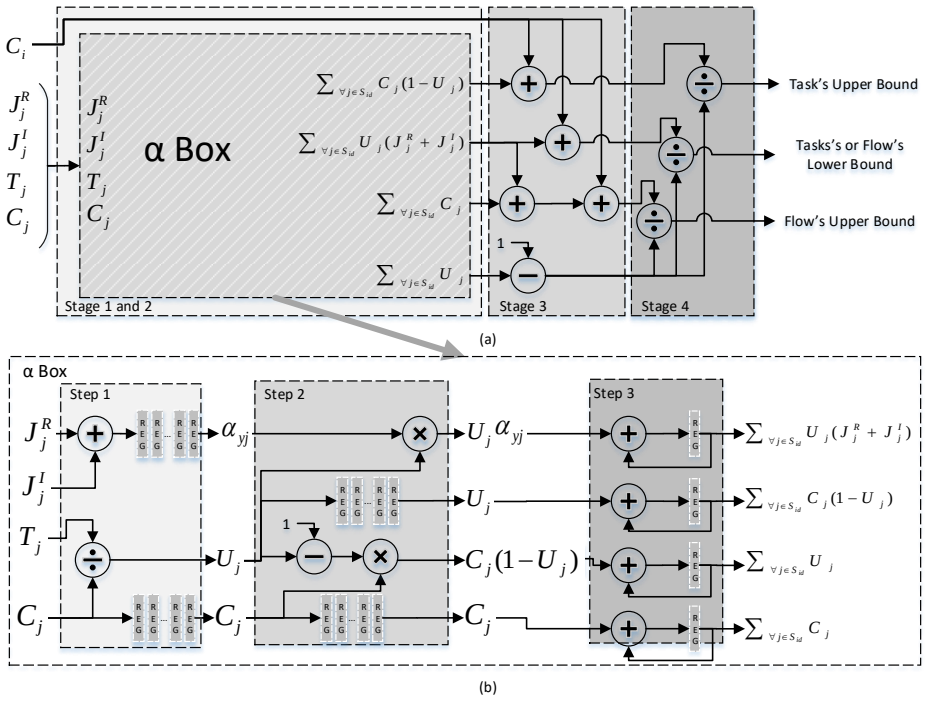

Figure 5.6: (a) Data flow of Equation. 5.6, (b) α box design structure.
Note: REG refers to register.

- From Figure 5.6b, the results of α box are all from accumulated functions. However, these accumulations involve simple addition, division, subtraction and multiplication. Therefore, it is worth dividing the calculation into several steps in order to maximise the possibility of parallel computing in hardware.

- Based on the decomposition, we can achieve a fast computation by using a pipelined structure. Normally, there are multiple elements in S_{id} *or* $hp(j)$. Therefore, the accumulation of α box is a repeated iteration. For example, assuming we have four elements in S_{id} or $hp(j)$, then the working process of α box without a pipelined acceleration can be seen from Figure 5.7a. We have to launch each iteration sequentially. However, if we follow a pipelined acceleration, we can launch one iteration at every clock cycle. In Figure 5.7b, we can see that step 1 becomes free and can accept new data when step 2 is working. Thus, a

pipelined structure can decrease the computational time. In addition, this improvement will be increased incrementally along with the number of elements in S_{id} or $hp(j)$.

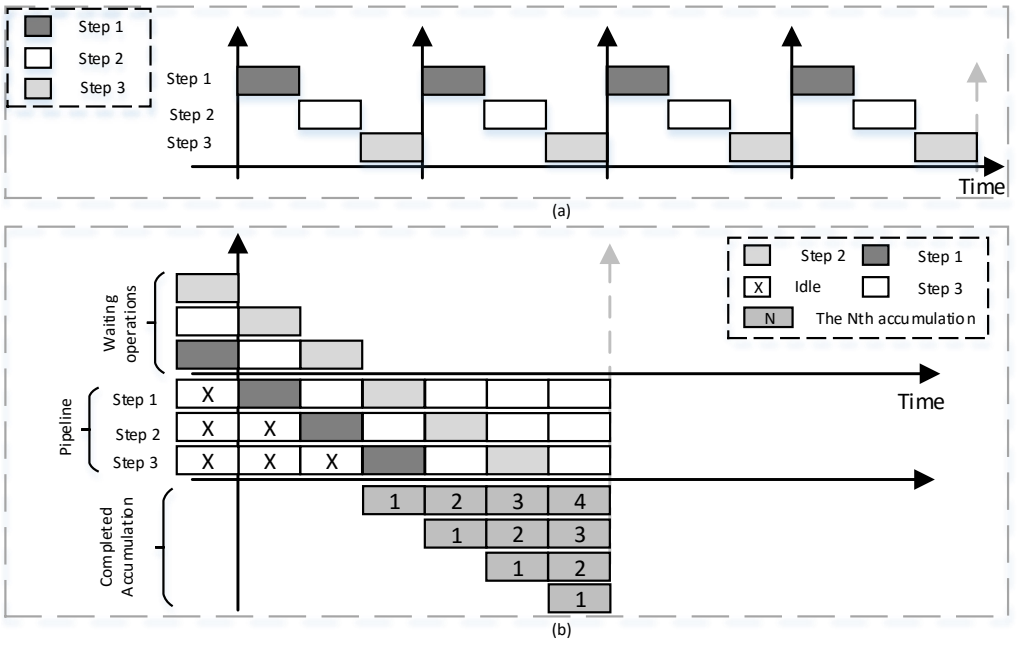

Figure 5.7: (a) Example of sequential α box, (b) Example of pipelined α box. Note: assume each unit can be finished within one clock cycle.

In addition, the division and multiplication operations are normally slower than addition and subtraction operations in a high frequency system. To alleviate this problem we pipelined the division and multiplication operations as well. At the same time, we also introduced registers on the transmission lines for some necessary signals such as α_{yj}, C_j *and* U_j.

5.3.2 Assembly Schemes

The proposed hardware accelerated inexact components can be used in both hardware implemented PRE and NLB, which can be assembled with HW-E2ERTA as an Inexact HW-E2ERTA. However, similar to the software

platform, the hardware implemented PRE is still limited as a sufficient test as it cannot guarantee the result of response time analysis. It has to co-operate with other components such as HW-E2ERTA or hardware implemented NLB. We list some possible assembly schemes in Figure 5.8, which consists of four parts (a, b, c, and d). In (a) and (c), we put PRE, HW-E2ERTA or NLB in sequential order. If PRE has indicated the final response time of a task or a flow, the following HW-E2ERTA or NLB will be skipped. Otherwise, the HW-E2ERTA or NLB will be applied.

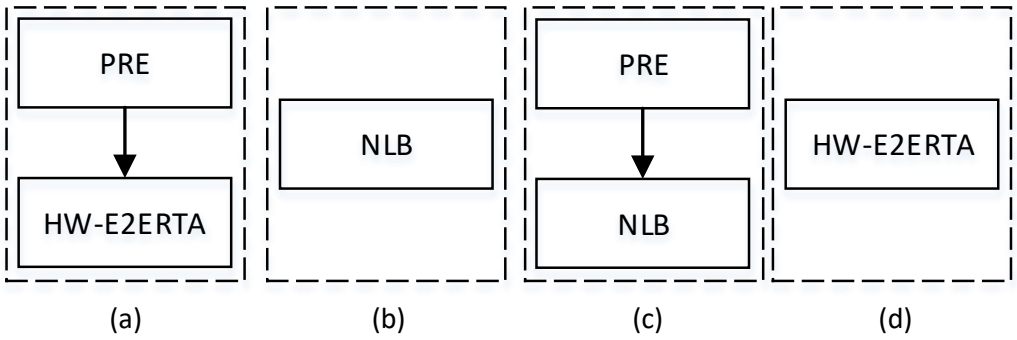

Figure 5.8: (a) PRE+HW-E2ERTA, (b) NLB, (c) PRE+NLB, (d) HW-E2ERTA

5.3.3 Experiment and Results Analysis

To evaluate how well the various assembly schemes of Inexact HW-E2ERTA improve HW-E2ERTA (designed in Section 5.2), an experiment is established. This subsection will present, in order, the experiment platform, experiment configuration, and results.

Experiment Platform

The experiment platform used here inherits the one used in Section 5.2. Similarly, the assembly schemes are mounted on an AXI bus with bus interfaces, which is shown in Figure 5.9a.

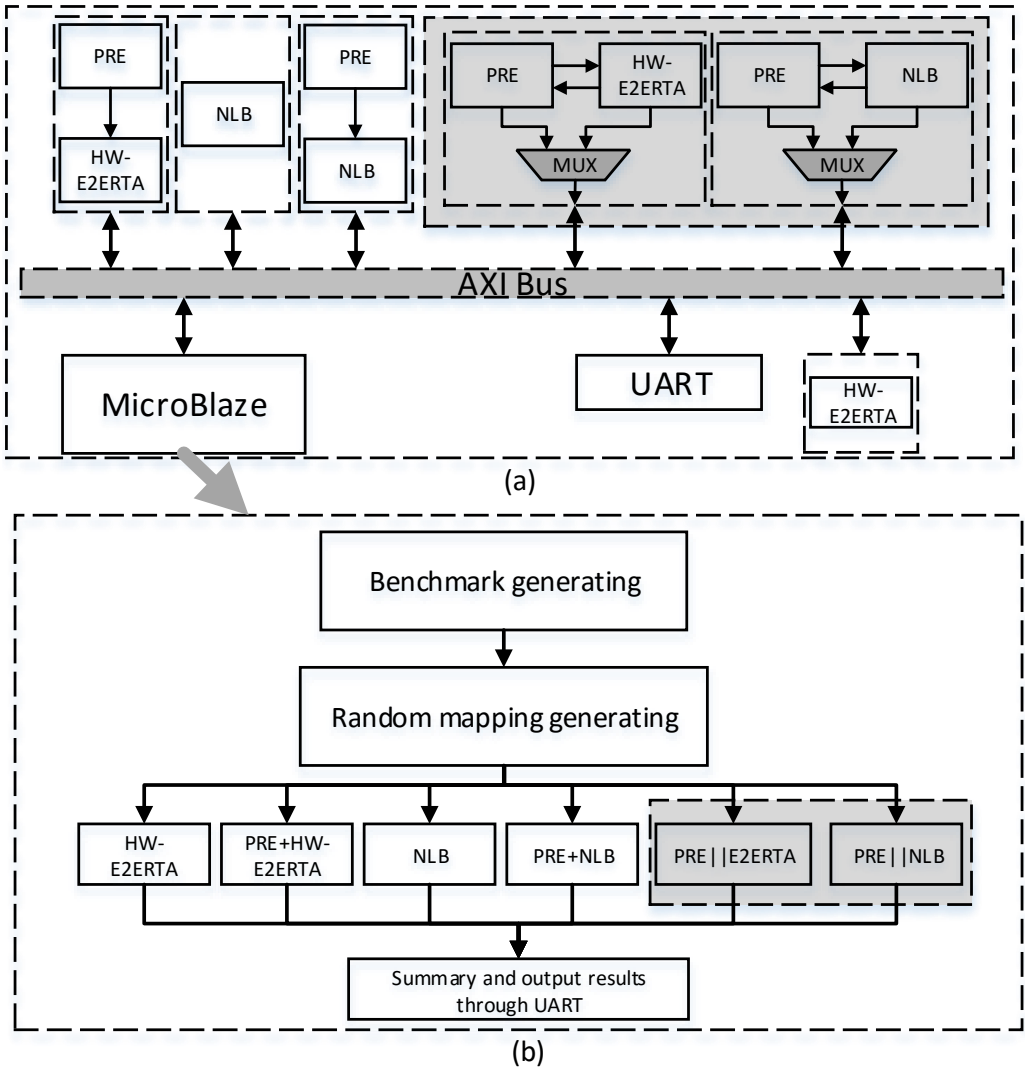

Figure 5.9: (a) Experiment Platform, (b) Testing Process.

Note: The blocks labeled in gray are parallelism implementation for future work among HW-E2ERTA and its accelerated components.

As above, each testing starts with benchmark generation until all peripherals have been evaluated. Figure 5.9b shows the testing process. The MicroBlaze first generates test data (a random task mapping and a synthetic benchmark which includes task and flow parameters). The MicroBlaze will then load the test data to each component and activate all of them simultaneously. After all peripherals have been evaluated, the MicroBlaze will collect results

from each hardware peripheral, and output them through UART port after organisation.

Experiment Configuration

We follow the experiment configuration used in Section 5.2 to test the hardware accelerated components. We also increase the number of testing times to 1,000,000, in order to obtain a better coverage.

Results Analysis

Figures 5.10, 5.11 and 5.12 show partial results of the experiment, while more details are shown in Table 5.3. All the Y-axes in these figures show the numbers of clock cycles that have been used to finish an E2ERTA computation. Because the numbers are large, they are arranged in log_{10} scale.

Figures 5.10a and 5.10b present the influence from NoC size, number of tasks and utilisation for PRE and NLB respectively. From the bar charts, we see that the tendency of PRE and NLB follows a non-linear style which is similar with the results of HW-E2ERTA shown in Figure 5.5, page 117. However, if we compare the averages of PRE, NLB and HW-E2ERTA in Table 5.3, we see that PRE cannot guarantee the improvement, and that NLB provides the least improvement.

The reason why NLB obtains the worst results is that it has to calculate the lower bound first and then start the exact calculation for each computation. We assume that:

- the number of clock cycles used to calculate the lower bound in NLB is N_{clb},
- the number of clock cycles used to compute the following exact calculation is N_{cec},

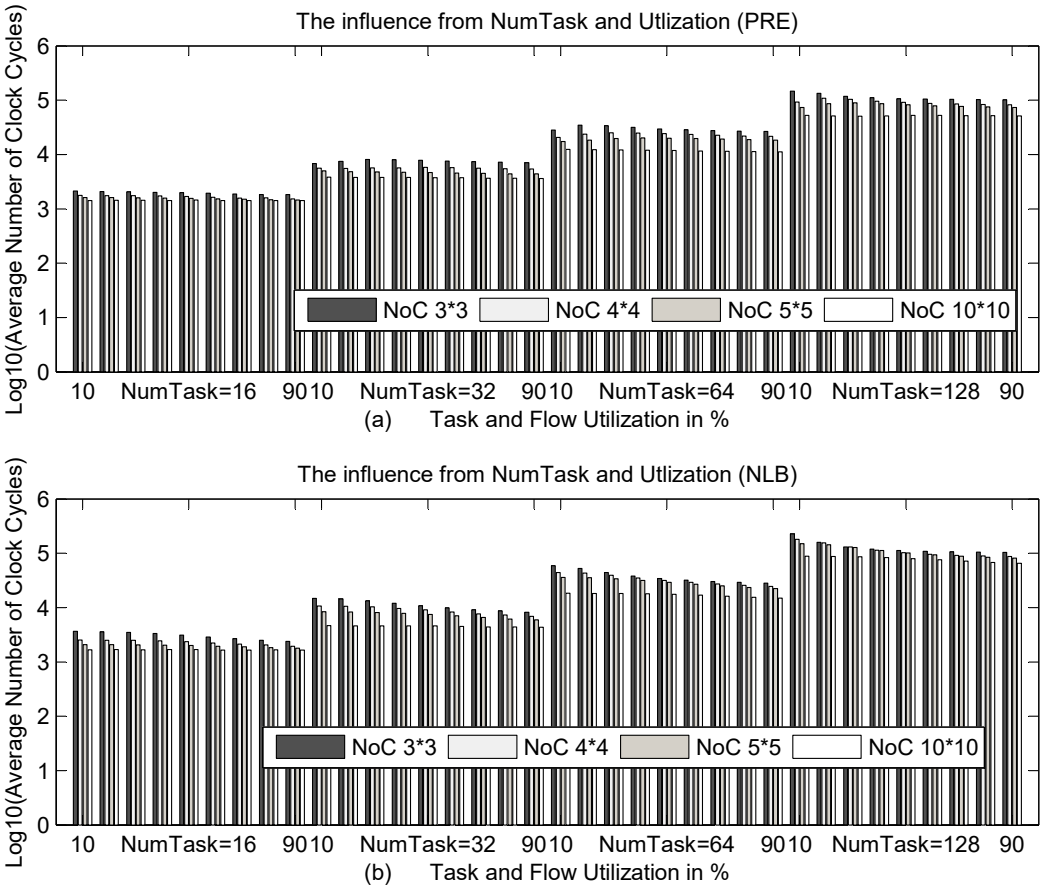

Figure 5.10: (a) The influence from NoC size, Number of Tasks and Utilisation for PRE, (b) The influence from NoC size, Number of Tasks and Utilisation for NLB.

Note: PRE refers to PRE + HW-E2ERTA.

- the number of clock cycles used by HW-E2ERTA is N_{E2ERTA}.

Here the total number of clock cycles used by NLB is $N_{clb} + N_{cec}$. We can guarantee $N_{cec} \leq N_{E2ERTA}$, but we cannot guarantee $N_{clb} + N_{cec} \leq N_{E2ERTA}$. Therefore, the use of NLB alone may be slower than HW-E2ERTA.

Since PRE is a sufficient test, only using upper bound and lower bound cannot guarantee the final results. When PRE succeeds, it can reduce the number of clock cycles by a greater degree than HW-E2ERTA by avoiding the exact test (HW-E2ERTA or NLB). However, if it fails, the number of clock cycles

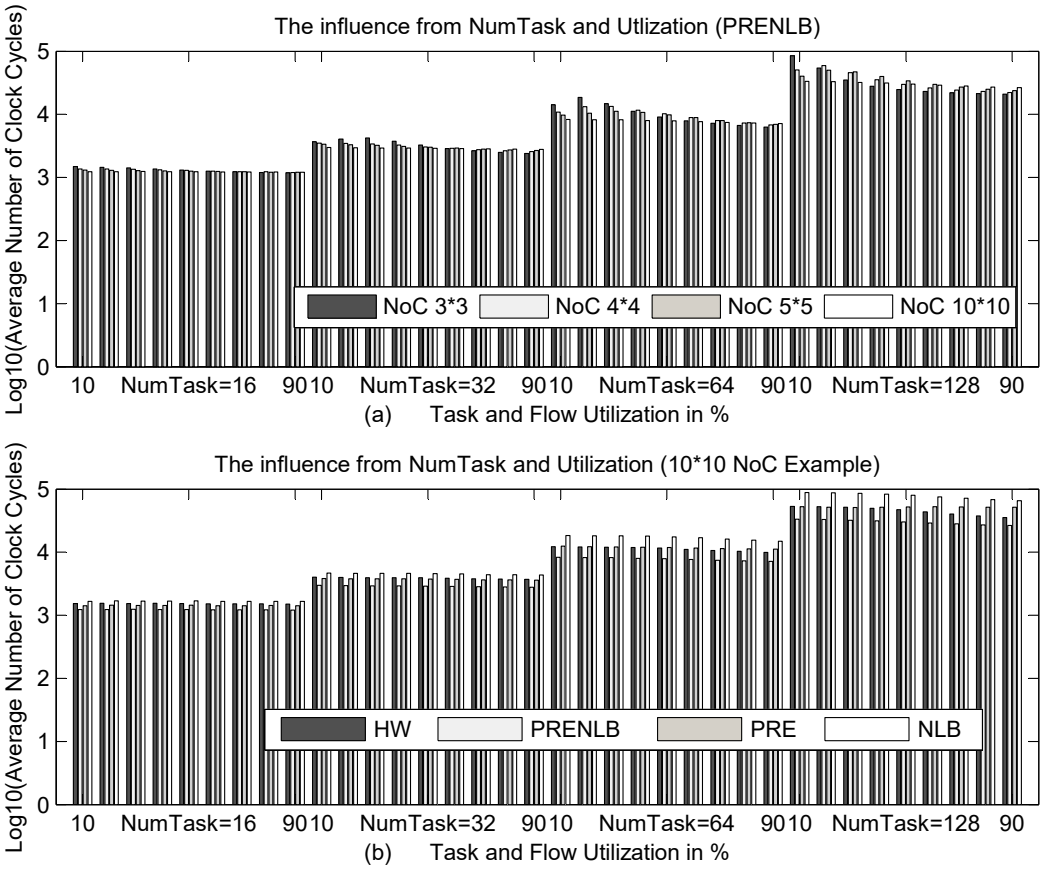

Figure 5.11: (a) The influence from NoC size, Number of Tasks and Utilisation for PRE, (b) Hardware versions on 10*10 NoC.

Note: PRE refers to PRE + HW-E2ERTA.

can increase compared to HW-E2ERTA as the subsequent exact calculation will be launched. Therefore, the performance of PRE may be worse than HW-E2ERTA.

Next is the PRE+NLB which is shown in Figure 5.11. It has the abilities inherited from both PRE and NLB. It can avoid the exact test in some situations and guarantee the final results within a shorter running time than all others when PRE is failed. We also make the following assumptions:

- the number of clock cycles used to calculate the lower bound and upper bound is N_{ulb},

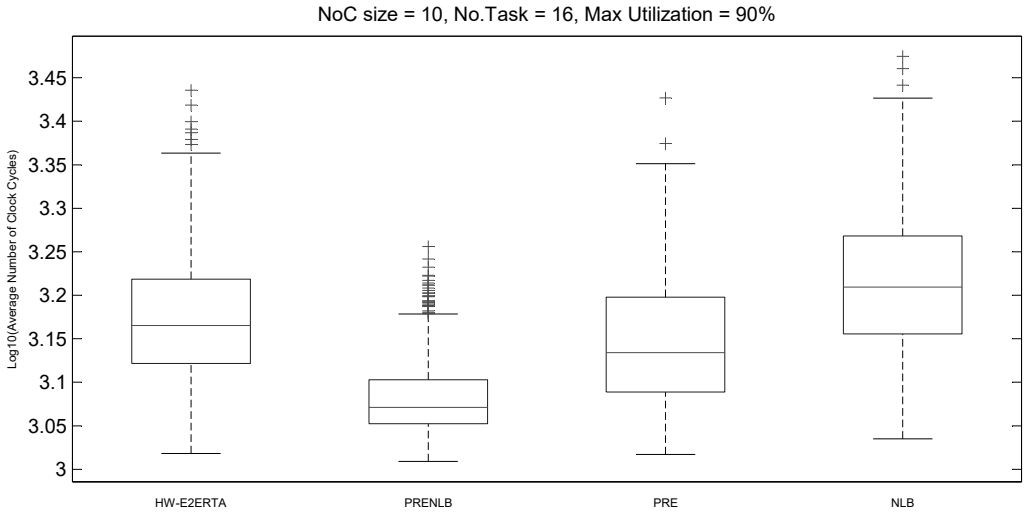

Figure 5.12: PRE+HW-E2ERTA vs NLB vs PRE+NLB vs E2ERTA.
Note: PRE refers to PRE + HW-E2ERTA.

- the number of clock cycles used to compute the following exact calculation is N_{cec}.

For a single test, the total number of clock cycles used by PRE+NLB is either N_{ulb} or $N_{ulb} + N_{cec}$. Theoretically, the PRE+NLB cannot guarantee that its performance is better than HW-E2ERTA in a single test run. However, after testing 1,000,000 times, the average number of clock cycles required by PRE+NLB is around $1 * 10^{3.084}$, while that for HW-E2ERTA is about $1 * 10^{3.179}$. We can generally summarise that PRE+NLB is approximately 1.25 times faster than HW-E2ERTA. This can be seen from Figure 5.11b and Figure 5.12, where the PRE+NLB is always the best.

5.4 Summary

In this chapter, we explore how the efficiency of E2ERTA can be affected by the existing implementation method. A parallel computation has been proposed and implemented in hardware, in order to enhance the performance of E2ERTA. In addition, two hardware accelerated inexact E2ERTA components

which inherit the technique from Chapter 4 are also introduced. Various assembly schemes which can explore the trade-off between the performance and coverage of an Inexact HW-E2ERTA are applied. Similar to the findings of Chapter 4, applying single hardware implemented inexact E2ERTA component cannot guarantee a better performance, but combining these two components can do so.

The improvement achieved in Chapters 4 and 5 are all focused on the efficiency of E2ERTA, which is one barrier to applying search-based algorithms and E2ERTA for dynamic task allocation in a hard real-time system. Improvements due to other factors remain to be explored. These will be discussed in the following chapter.

Table 5.3: Results Table

		3*3					4*4					5*5					10*10				
No. Task	U(%)	SW-E2ERTA	HW-E2ERTA	PRENLB	PRE	NLB	SW-E2ERTA	HW-E2ERTA	PRENLB	PRE	NLB	SW-E2ERTA	HW-E2ERTA	PRENLB	PRE	NLB	SW-E2ERTA	HW-E2ERTA	PRENLB	PRE	NLB
	10	6.140	3.417	3.171	3.326	3.561	6.113	3.308	3.136	3.249	3.401	6.134	3.261	3.116	3.209	3.316	6.437	3.185	3.093	3.151	3.222
	20	6.136	3.409	3.160	3.319	3.554	6.113	3.305	3.133	3.245	3.399	6.135	3.261	3.114	3.208	3.319	6.438	3.190	3.093	3.158	3.228
	30	6.130	3.398	3.153	3.313	3.544	6.113	3.301	3.128	3.241	3.396	6.135	3.256	3.108	3.205	3.315	6.440	3.186	3.094	3.156	3.225
	40	6.120	3.373	3.136	3.301	3.524	6.110	3.292	3.121	3.237	3.390	6.132	3.250	3.104	3.197	3.308	6.438	3.189	3.092	3.154	3.226
16	50	6.104	3.339	3.117	3.296	3.495	6.104	3.278	3.113	3.230	3.373	6.131	3.244	3.099	3.192	3.302	6.442	3.188	3.091	3.161	3.227
	60	6.087	3.294	3.101	3.287	3.458	6.094	3.255	3.099	3.215	3.350	6.126	3.232	3.094	3.185	3.288	6.438	3.181	3.088	3.153	3.220
	70	6.073	3.260	3.092	3.275	3.427	6.084	3.238	3.091	3.200	3.327	6.121	3.223	3.092	3.178	3.277	6.436	3.180	3.085	3.153	3.219
	80	6.060	3.230	3.079	3.263	3.398	6.079	3.227	3.089	3.201	3.311	6.117	3.209	3.083	3.166	3.261	6.438	3.181	3.086	3.155	3.222
	90	6.053	3.210	3.072	3.261	3.380	6.071	3.209	3.080	3.182	3.288	6.112	3.202	3.082	3.165	3.254	6.435	3.179	3.084	3.152	3.220
	10	6.544	4.007	3.566	3.836	4.168	6.496	3.861	3.543	3.748	4.029	6.526	3.758	3.526	3.699	3.924	6.898	3.604	3.474	3.582	3.671
	20	6.537	3.998	3.607	3.875	4.160	6.494	3.859	3.541	3.746	4.026	6.523	3.750	3.516	3.683	3.917	6.896	3.598	3.470	3.579	3.665
	30	6.506	3.955	3.624	3.908	4.126	6.487	3.842	3.531	3.752	4.012	6.520	3.743	3.509	3.681	3.911	6.898	3.596	3.468	3.578	3.664
	40	6.466	3.890	3.576	3.906	4.078	6.469	3.807	3.513	3.754	3.986	6.513	3.724	3.494	3.675	3.896	6.897	3.597	3.466	3.579	3.666
32	50	6.431	3.827	3.515	3.896	4.034	6.453	3.770	3.485	3.764	3.957	6.506	3.699	3.481	3.670	3.875	6.897	3.594	3.461	3.574	3.662
	60	6.399	3.766	3.458	3.879	3.993	6.432	3.719	3.463	3.758	3.921	6.494	3.664	3.466	3.661	3.848	6.896	3.587	3.458	3.571	3.655
	70	6.377	3.714	3.422	3.868	3.959	6.413	3.669	3.441	3.747	3.886	6.484	3.630	3.449	3.655	3.819	6.893	3.578	3.452	3.561	3.645
	80	6.362	3.678	3.398	3.857	3.937	6.403	3.633	3.425	3.740	3.862	6.473	3.598	3.438	3.645	3.790	6.894	3.575	3.449	3.561	3.642
	90	6.347	3.641	3.381	3.851	3.915	6.391	3.594	3.412	3.733	3.837	6.470	3.580	3.429	3.646	3.776	6.891	3.568	3.445	3.557	3.637
	10	7.055	4.608	4.153	4.452	4.771	6.994	4.462	4.036	4.315	4.644	7.031	4.364	3.989	4.240	4.555	7.449	4.088	3.921	4.095	4.265
	20	7.005	4.546	4.268	4.541	4.722	6.986	4.451	4.120	4.373	4.634	7.027	4.358	4.019	4.263	4.549	7.450	4.083	3.916	4.087	4.262
	30	6.929	4.438	4.169	4.528	4.645	6.955	4.400	4.127	4.402	4.594	7.016	4.337	4.048	4.295	4.532	7.448	4.080	3.914	4.085	4.260
	40	6.870	4.341	4.049	4.498	4.581	6.921	4.333	4.068	4.396	4.547	7.002	4.295	4.032	4.306	4.500	7.448	4.074	3.903	4.077	4.255
64	50	6.828	4.264	3.959	4.471	4.533	6.891	4.268	4.008	4.383	4.502	6.984	4.245	3.992	4.302	4.464	7.450	4.064	3.896	4.074	4.245
	60	6.806	4.212	3.898	4.455	4.504	6.870	4.211	3.949	4.368	4.465	6.970	4.198	3.951	4.296	4.432	7.447	4.045	3.885	4.065	4.228
	70	6.788	4.173	3.857	4.442	4.481	6.852	4.160	3.900	4.354	4.433	6.959	4.150	3.904	4.285	4.401	7.445	4.027	3.874	4.059	4.209
	80	6.774	4.140	3.823	4.431	4.463	6.839	4.117	3.863	4.341	4.408	6.948	4.105	3.867	4.274	4.371	7.444	4.013	3.864	4.055	4.191
	90	6.766	4.119	3.800	4.426	4.452	6.829	4.086	3.833	4.334	4.390	6.941	4.074	3.841	4.267	4.352	7.443	3.996	3.855	4.049	4.175
	10	7.602	5.192	4.927	5.167	5.355	7.553	5.074	4.703	4.967	5.255	7.860	4.977	4.607	4.866	5.177	8.035	4.726	4.523	4.721	4.945
	20	7.445	4.974	4.733	5.126	5.201	7.503	4.993	4.775	5.038	5.193	7.840	4.956	4.701	4.936	5.157	8.036	4.722	4.517	4.712	4.941
	30	7.363	4.834	4.545	5.072	5.116	7.447	4.879	4.661	5.015	5.114	7.805	4.887	4.674	4.951	5.107	8.035	4.715	4.507	4.707	4.934
	40	7.325	4.758	4.447	5.044	5.074	7.409	4.783	4.549	4.983	5.054	7.773	4.809	4.600	4.936	5.053	8.032	4.697	4.496	4.712	4.920
128	50	7.305	4.714	4.394	5.028	5.050	7.383	4.712	4.474	4.959	5.012	7.749	4.739	4.531	4.916	5.008	8.032	4.673	4.482	4.719	4.902
	60	7.294	4.687	4.363	5.020	5.036	7.367	4.658	4.419	4.941	4.981	7.728	4.680	4.474	4.898	4.972	8.030	4.641	4.465	4.721	4.879
	70	7.288	4.669	4.341	5.014	5.026	7.359	4.623	4.386	4.930	4.963	7.715	4.634	4.432	4.886	4.945	8.028	4.607	4.450	4.717	4.855
	80	7.284	4.657	4.329	5.010	5.020	7.353	4.598	4.362	4.922	4.949	7.704	4.597	4.398	4.875	4.924	8.027	4.574	4.434	4.714	4.832
	90	7.280	4.648	4.320	5.007	5.016	7.349	4.579	4.345	4.917	4.939	7.694	4.570	4.376	4.868	4.909	8.029	4.549	4.424	4.711	4.816

Note: SW refers to SW-E2ERTA, HW refers to HW-E2ERTA and U refers to the utilization of task or flow.

Chapter 6

Performance Optimisation of Genetic Algorithm

In consideration of the research problem described in Section 1.3, apart from the guarantee of timing performance after remapping, another factor which can cause a low success rate of dynamic remapping for a hard real-time NoC is the existing dynamic mappers being limited by their mapping search or generating ability. They either do not consider running task reallocation, or are limited by the predefined construction criteria. This can be understood as follows. First, intense resource competition among tasks or flows will result in available computation or communication resources on each IP or link of an NoC being limited. A new added task would be unable to obtain enough resources to execute or communicate with other tasks and would easily be rejected if there is no optimisation of the current resource allocation (e.g. moving some tasks to other IPs and reserving enough resources for the new task). An example is shown in Figure 1.4, page 24. Second, predefined construction criteria (e.g. mapping tasks to IPs in a spiral style from centre to boundaries) can provide a clear guidance for fast mapping candidate construction. However, this will mean that only a small design space can be explored and directly results in a low success rate for dynamic remapping. In

addition, both of these two factors will become worse when the complexity of application is increased. Exhaustively checking all simulation becomes impractical with increases in the design space, especially for a dynamic mapping problem whose working time slot and resources are limited. In this situation, search-based algorithms could be considered as an alternative solution, since they can provide a global optimisation ability and a trade-off between timing performance and remapping cost (number of tasks in migration) with finite resources. Genetic Algorithm (GA), as a representative of search-based algorithms, will be considered the new dynamic mapper in this research to achieve an efficient mapping search.

As discussed in the problem breakdown in Section 3.3, the efficiency of GA search can be affected by both its fitness function and its search operations (such as fitness loading, crossover and mutation). Although, in Chapters 4 and 5, we tried to reduce the execution time of E2ERTA which could be used as the fitness function in GA to evaluate mappings' hard real-time performance, the search efficiency will remain low without any optimisation of GA itself. Therefore, in this chapter, we discuss the possibility of optimising GA in respect of platform selection, model selection with optimisation, and accelerator design, in order to propose a parallel GA architecture along with two accelerated GA operators to enhance the performance of GA search.

6.1 Platform Comparison

As discussed in the review in Section 2.4, a GA can be implemented as a search tool, in either software or hardware, to optimise a complex problem such as task allocation in NoCs. Compared with software GA (SWGA), the hardware version is a dedicated component designed for a specific optimisation problem. Its architecture is customised. The data transfer and processing will not be affected by the computation core and bus width, and the efficiency is therefore much higher than SWGA's. However, questions to be considered include how fast a hardware GA (HWGA) can run, and whether an HWGA

can compensate the disadvantage of applying GA as a dynamic mapper in the dynamic mapping problem of a hard real-time NoC. These can be answered by the following experiment and comparison through fitness function selection, experiment platform, experiment configuration and results analysis.

6.1.1 Fitness Function Selection

The search time of GA can be affected by both the search operation and its fitness function. Thus, to evaluate accurately how much faster than the software version an HWGA can run, the influence from fitness function should be minimised. In other words, the execution time of fitness function should be fixed or nearly fixed. Therefore, from this point of view, applying E2ERTA as a fitness function is not suitable in this kind of experiment, because the execution time of E2ERTA is highly dependent on the task mapping. Therefore, we propose a simple fitness function (Max One which counts the number of logic zero in a bit vector) with fixed execution time.

6.1.2 Experiment Platform

This experiment continues to use the platform used in Section 5.2.3, but with a larger hardware timer (64-bit). On this platform, a sequential GA was fully implemented on MicroBlaze in language C with the C compiler of GNU version 2.16. Then, a hardware implemented sequential GA (in VHDL) was mounted on the AXI bus with an AXI bus interface. Similar to the experiment process in Section 5.2.3, after initialisation, the MicroBlaze will first activate the SWGA and then the hardware one. The data is collected and organised at the end.

6.1.3 Experiment Configuration

The experiment follows the following configurations:

- the system clock is 50Mhz;
- the frequency of AXI bus is 100Mhz which also drive the HWGA;
- the same GA configuration could have the same affact to both software and hardware version of GA,
- we could following the GA configuration used in Chapter 4:
  - crossover rate is 0.5%,
  - mutation rate is 0.01%,
  - max number of generation is 50;
- for fast testing, population size is 6;
- fitness function is the Max One which counts the number of logic zero in a 256-bit vector;
- number of repeated tests is 1,000,000.

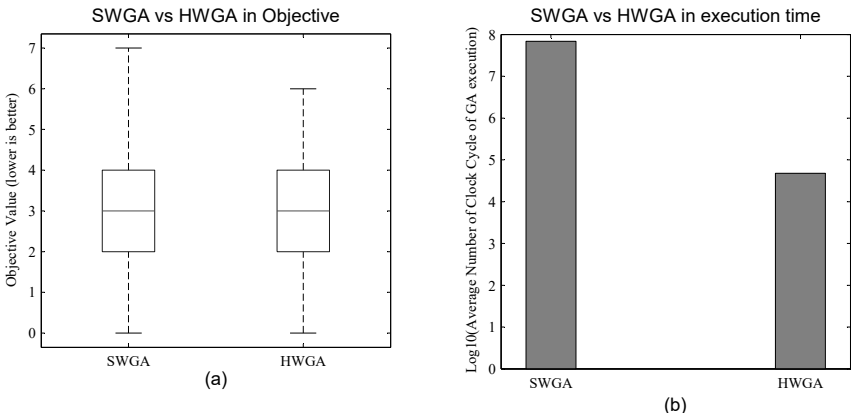

Figure 6.1: Software VS Hardware GA.

6.1.4 Results Analysis

The results are shown in Figure 6.1. From Figure 6.1a, it can be seen that these two implementations achieve the same optimisation of the average final objective. Their distributions are similar, which means the HWGA has the same optimisation ability as SWGA. However, the required search time in HWGA is much less than that in SWGA, if Figure 6.1b is considered. The values in Figure 6.1b indicate that SWGA is around 1450 times ($10^{7.84-4.68}$) slower than HWGA. However, whether this improvement can support a dynamical task allocation search for hard real-time NoCs can be analysed as follows. As discussed in Section 3.3, the search time of GA could be described by Equation 3.3b, page 84. Hardware implementation will be beneficial to not only the search operation but also the fitness. Although we apply Max One to minimise the affection form fitness function, the improvement form hardware version Max One cannot be fully avoided, such as data reading and writing. In other words, in reality this 1450 times advantage is partially contributed to by the optimisation of the Max One. If we assume this 1450 times advantage is contributed by search operation alone and use it to estimate the speed we could achieve in a mapping search with HW-E2ERTA as fitness function with the same clock used in Chapter 5, the execution time of HWGA and SWGA could be written as follows:

$$
\begin{aligned}
ET_{SWGA} &= ET_{SWGA\ search\ operation} + ET_{SW-E2ERTA} \qquad (6.1a)\\
&= 1450 * ET_{HWGA\ search\ operation}\\
&\quad +10^{best\ average\ improvement\ (from\ Table\ 5.1)} * ET_{HW-E2ERTA}\\
&= 1450 * ET_{HWGA\ search\ operation}\\
&\quad +10^{3.33} * ET_{HW-E2ERTA}\\
&\approx 1450 * (ET_{HWGA\ search\ operation}\\
&\qquad +ET_{HW-E2ERTA})
\end{aligned}
$$

$$ET_{HWGA} = ET_{HWGA\ search\ operation} + ET_{HW-E2ERTA} \tag{6.1b}$$

In addition, normally the fitness function will be much more complex than the GA search operation, especially for E2ERTA which is based on a series of iterative calculations. Thus, the difference in the value between $ET_{HWGA\ search\ operation}$ and $ET_{HW-E2ERTA}$ will be very large. If we assume the $ET_{HWGA\ search\ operation}$ can be ignored, then we could generally estimate that the HWGA+HW-E2ERTA can be 1450 times faster than SWGA+SW-E2ERTA. Moreover, by considering the worst case (1261.65 seconds, SWGA+SW-E2ERTA implemented on PC), which is illustrated in Table 4.4, page 103, the execution time for this case can be estimated as 0.87 second, if it is searched by HWGA+HW-E2ERTA. This result can be accepted by some application with loose timing requirements, but not for those ones whose timing requirements are harsh. Therefore, applying HWGA alone cannot adequately support the problem in this research. It is necessary to find another method to further improve the search efficiency of applying GA for dynamically optimising task allocation in NoCs.

6.1.5 Summary

In this section, an experiment is applied to explore whether using a hardware GA alone can adequately support the dynamic task allocation search in NoCs by using GA as the dynamic mapper. From the results, we can generally conclude that, while HWGA is much faster than the software one, it still cannot adequately fit the requirements of the problem in this research. It is necessary to consider the improvement methods from other directions, such as search structure, which will be done in the following section.

6.2 Model Selection and Optimisation of HWGA

In line with the literature review (Section 2.4), apart from hardware implementation, parallelisation is another method to enhance GA search. There are three models that can be selected: Master-Slave model, Island model and hybrid model. They can be used to further improve the search efficiency of HWGA. Among them, the Master-Slave model is most similar to sequential GA. At the same time, it is relatively easy to implement. As a single population evolution, the resource requirements of the Master-Slave model are low and do not require other communication infrastructure, whereas other models do. However, how well it can support a HWGA and whether there is a possibility to further optimise the search efficiency are the questions that should be considered. These will be discussed in this section, in the following order: performance verification of Master-Slave GA (MS GA); then a Free-Step Master-Slave GA (optimised version).

6.2.1 Performance Verification of Master-Slave GA

The question of the extent to which a Master-Slave model can improve the search efficiency of HWGA is answered using an experiment in this subsection. For the Master-Slave model under consideration, its working process is shown in Figure 2.18, page 70 and described in Section 2.4. The following sections detail, in order, experiment platform, experiment configuration and results analysis.

Experiment Platform

The Master-Slave model used here is modified based on the HWGA implemened in Section 6.1.1. The fitness function, GA operators (crossover and mutation component) and experiment platform are maintained.

Experiment Configuration

This experiment is configured as follows:

- all GA components are the same as the pervious, only the data loading method has been changed;
- we can continue using the GA confiuration used in pervious section;
- number of fitness is 2, 3, 4, 5, 6.

Results Analysis

The results are shown in Figure 6.2. From this figure we note that increasing the number of fitness functions in MS GA can reduce the search time. This is because the Master-Slave model only changes the methods of how to distribute candidate solutions to fitness functions, rather than optimising the fitness function itself. In other words, it can divide a job which is originally undertaken by one fitness function to several parts and distribute them to several fitness functions to work simultaneously. Although the working efficiency of each fitness function is not increased, the overall working time can be reduced.

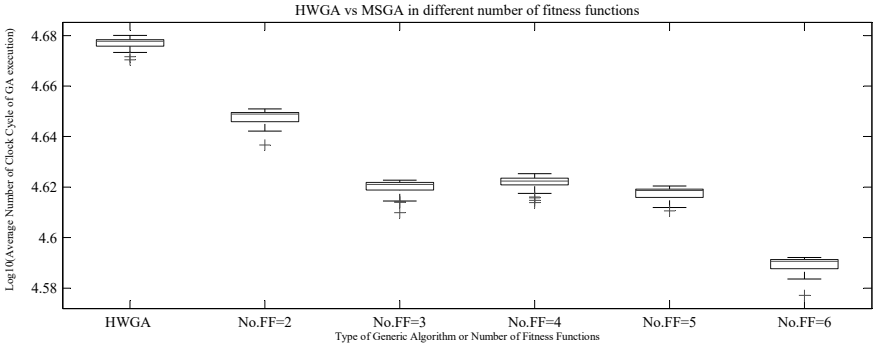

Figure 6.2: HWGA VS MS GA.

However, by considering the results, the execution time of MS GA is not less than that of HWGA by any significant factor. For example, the average

execution time of HWGA and MS GA with two fitness functions are around $10^{4.68}$ and $10^{4.65}$ respectively, which means the real improvement is around $10^{(4.68-4.65)}$, instead of 2 as expected. This is because the fitness function selected for evaluation is Max One, whose execution time is fixed and not significantly different to the running time used to finish GA search operations such as mutation, initialisation and replacement. If we use an equation to estimate the execution time used during the GA search, it can be represented as Equation 6.2a.

$$\begin{aligned} ET_{HWGA} &= ET_{HWGA\ search\ operation} + ET_{Max\ One} \\ &= ET_{Initialisation} + ET_{Crossover} + ET_{Mutation} \\ &\quad + ET_{Replacement} + Population\ Size * ET_{Single\ Max\ One} \end{aligned} \tag{6.2a}$$

$$\begin{aligned} ET_{Initialisation} &= Population\ Size * ET_{Single\ Max\ One} \\ ET_{Crossover} &= Population\ Size \\ ET_{Mutation} &= Population\ Size * ET_{Single\ Max\ One} \\ ET_{Replacement} &= (2 * Population\ Size)^2 \end{aligned} \tag{6.2b}$$

If we assume other operations (such as system preparation, signal hand shake for data loading and collecting) can be ignored and consider using ranking for replacement and single crossover, $ET_{Initialisation}$, $ET_{Crossover}$, $ET_{Mutation}$ and $ET_{Replacement}$ can be assigned values as in Equation 6.2b.

Then, the execution time of HWGA and MS GA can be described as Equation 6.3a and 6.3b. In addition, the $ET_{Single\ Max\ One}$ is nearly equal to 256, since it is a 256-bit Max One. Then the $\frac{ET_{HWGA}}{ET_{MSGA}} \approx 1.1925 \approx 10^{0.0765}$. The theoretical result seems better than our experiment reuslts. This is because we assume the time used by some operations can be ignored. In fact, when the time used by these operations is long (compared with Max One), they will affect the experiment results. In addtion, we can note that the difference

between Equation 6.3a and 6.3b. is the evaluation part. If we can apply a fitness function (such as Inexact HW-E2ERTA) which is more complex than Max One and has much longer execution time than GA search operations, the execution time of HWGA and MS GA will be mainly determined by evaluation part.

$$\begin{aligned} ET_{HWGA} &= 12 * ET_{Single\ Max\ One} + 12^2 + 6 \\ &\quad +6 * ET_{Single\ Max\ One} \end{aligned} \tag{6.3a}$$

$$\begin{aligned} ET_{MSGA} &= 12 * ET_{Single\ Max\ One} + 12^2 + 6 \\ &\quad +6 \div 2 * ET_{Single\ Max\ One} \end{aligned} \tag{6.3b}$$

Moverover, we notice that the improvement curve is non-linear. This is caused by the candidate distribution strategy of MS GA. From Figure 2.18, page 70, we can seen that the new round of candidate loading can only occur when the evaluation of previously loaded candidates has been finished. This will cause a phenomenon illustrated in Figure 6.3. From Figure 6.3a, we can see that three rounds of release are needed when the number of fitness functions is 2. Although we add two more fitness functions in Figure 6.3b, the MS GA still requires two rounds of release, even there are two fitness functions are idle in the second round. This phenomenon can happen when the number of fitness functions is 5 as well. It will directly affect the evaluation time and further result in improvement curve is non-linear. This phenomenon will not be a problem when the execution time of fitness is fixed. But, its drawback will emerge and become worse when the fitness function has variable execution time. This will be disscussed in the following subsection.

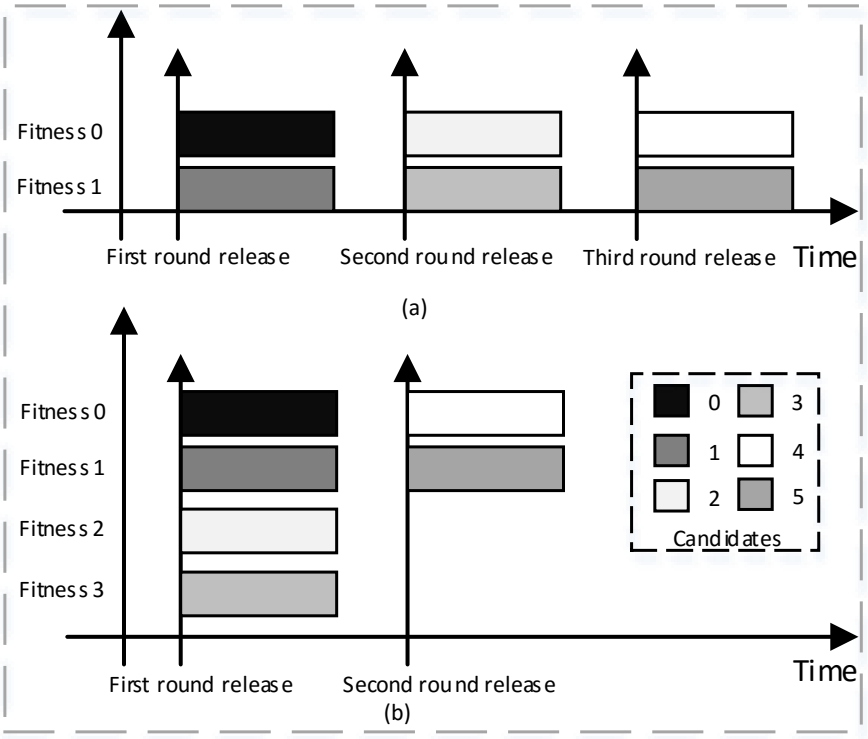

Figure 6.3: MS GA Low fitness Utilization Phenomenon.

Summary

In this subsection, we implement a sequential HWGA and show its improvement compared with sequential SWGA. By combining the Master-Slaver model with HWGA, the search efficiency can be further improved. In addition, we also find a disadvantage of the existing Master-Slaver model: the distribution strategy can affect the evaluation efficiency, if the fitness function execution time is variable. Because, Inexact HW-E2ERTA (our expected fitness function used for mapping evaluation) is a kind of fitness function with variable execution time, it is necessary to find a method to alleviate this disadvantage. This will be discussed in the following subsection.

6.2.2 Free-Step Master-Slave GA

The Master-Slave model can be used to achieve a fast evaluation in GA search. However, it will suffer a lock-step problem if its fitness function has variable execution time. This subsection will try to explore which reasons cause this shortcoming and whether there are some potential solutions that can be applied to compensate or alleviate it. In order to distinguish between the state-of-the-art and the proposed architecture, the existing MS GA is named LS-MS GA. The following parts discuss, in order: problem definition and analysis, possible architecture description, and performance evaluation.

Lock-Step Problem

The lock-step problem is a shortcoming caused by the distribution strategy of LS-MS GA. It only has a significant adverse impact on the evaluation time if the fitness functions' computation times are variable and depend on different candidate solutions. Following the example in Figure 2.18, page 70, Figure 6.4 shows this phenomenon. It can be seen that fitness 1 can only store its results after fitness 0 has been completed and the results have been recorded, regardless how quickly fitness 1 can be executed. This influence will become more severe when the size of GA population, the number of fitness functions and the variability of fitness function execution time are increased.

Architecture Description

Based on the phenomenon of the lock-step problem, we can find that this problem is caused by the system synchronisation. In each release round, the master has to synchronise the data for both fitness function execution and results collection. This blocks the further step of idle fitness functions getting unevaluated candidate solutions when other fitness functions are still executing. To alleviate it, the solution is to replace the existing system

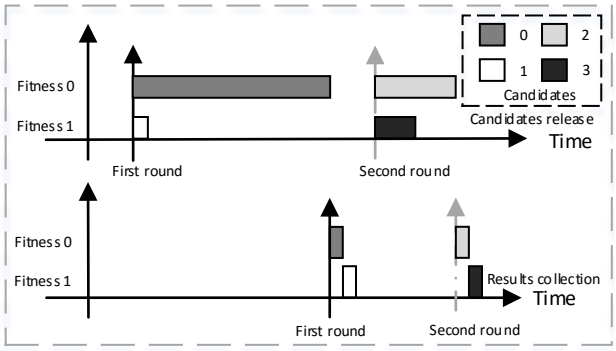

Figure 6.4: LockStepProblem.

Note: Following the example in Figure. 2.18.

synchronisation by introducing an asynchronous model. One possible example that follows the example in Figure 6.4 is shown in Figure 6.5.

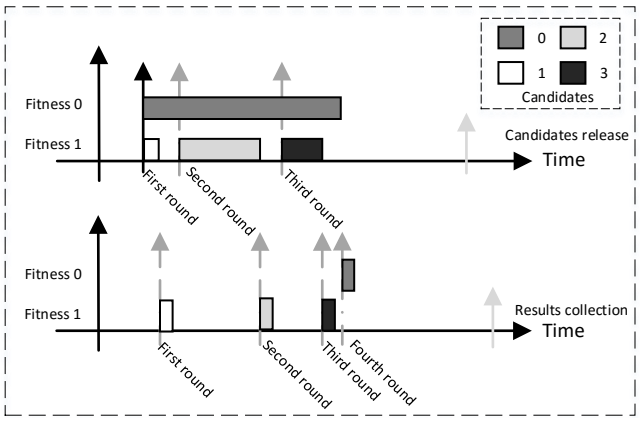

Figure 6.5: Example of an asynchronous model.

Note: Following the example in Figure 6.4

In this example, the two light grey solid arrows indicate the timing points of the completion of executing all fitness functions and collecting all results of Figure 6.4 respectively. From Figure 6.5, we can see that if we can load, release fitness and collect result individually, the overall execution time of candidate evaluation can be reduced significantly.

Architecture Implementation

A possible implementation of this asynchronous model can be described in Figure 6.6 and named as Free-Step Master-Slave GA (FS-MS GA).

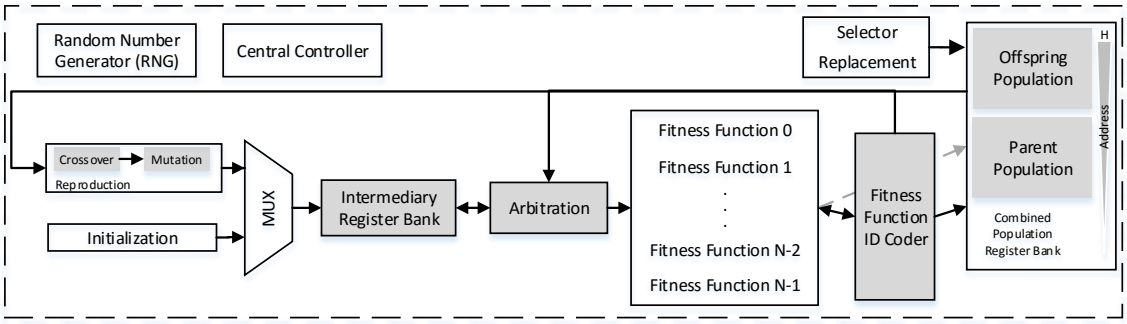

Figure 6.6: FS-MS GA Architecture.

The working process of the FS-MS GA is similar to the one of LS-MS GA. In order to launch fitness functions and collect feedback asynchronously, several components such as Intermediary Register Bank (IRB), Arbitration, Fitness Function ID Coder (FFID-Coder) and Combined Population Register Bank (CPRB) are also introduced. The IRB is used to temporarily store the new candidate solutions, which can be generated randomly through initialisation or bred by crossover and mutation in reproduction component, if these solutions cannot be evaluated immediately. Whenever new chromosomes arrive in IRB, the Arbitration will try to distribute them to fitness functions according to the indication from FFID-Coder. The FFID-Coder collects the busy and done signals from each fitness function. It generates two address signals for both Arbitration and CPRB to support candidate distributing and results storing respectively. Several written feedback signals will also be generated by the FFID-Coder to fitness functions. The CPRB will store both parent and offspring populations. Its size is twice the parent population's size. The replacement will sort the CPRB according to a given strategy, such as ranking (the better a solution is, the lower address it will be given). The selector will generate two addresses to select two parent chromosomes from the parent population for reproduction according to the selection strategy.

Arbitration and Fitness Function ID Coder

Distributing candidate chromosomes to each fitness function asynchronously is achieved by the Arbitration and FFID-Coder. The Arbitration is triggered by the coded fitness ready address and IRB ready signals. Its architecture is shown in Figure 6.7a. In its working process, the 'coded fitness ready address' can indicate whether there are fitness functions ready to receive new candidate chromosomes. If there are and the IRB ready signal is valid (there is at least one candidate chromosome in IRB that has not been evaluated), the Arbitration will enable the 'read enable' signal to read one chromosome from IRB and distribute it to the right slot of chromosome vector according to 'coded fitness ready address'. Otherwise, both the 'read enable' signal and chromosome will be disabled by logic '0' and 'Zero Vector' respectively.

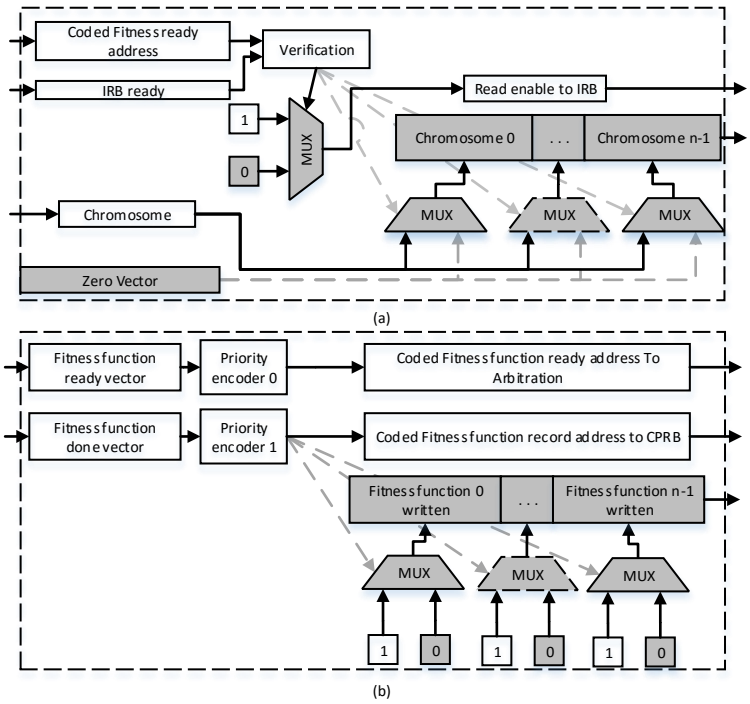

Figure 6.7: (a) Arbitration Architecture, (b) Fitness Function ID Coder Architecture.

Note: 'Zero Vector' consists of logic '0';
IRB refers to Intermediary Register Bank;
CPRB refers to Combined Population Register Bank.

The FFID-Coder shown in Figure 6.7b collects the ready and done signals from each fitness function. The ready signals are used to generate a fitness function ready address which can guide the chromosome distribution in Arbitration. The done signals will be encoded to indicate to the CPRB to read the result from which fitness function. Whenever the result has been recorded, the related bit in the written vector will be set as the acknowledgement back to the fitness function. If there are more than one fitness functions idle or finished, the FFID-Coder will code based on priority of fitness functions. The priority is assigned according to fintness function index, zero is highest. If there is no fitness functions idle or finished, the FFID-Coder will set all output signals as invalid.

Experiment and Results Analysis

The evaluation of how well FS-MS GA can improve performance over LS-MS GA can be analysed on the basis of an experiment, which described in this part. In order, we deal with fitness function description, experiment platform, experiment configuration and analysis of results.

Fitness Function

A fitness function with variable execution time is the trigger that causes the lock-step problem in LS-MS GA. Thus, we proposed a fitness function, which is Slice Logic One Counter (S-LOC) to imitate this situation. The input of S-LOC consists of a slice range and a test vector. It can return the number of Logic '1's in a slice of a test vector. A slice can cover from 1-bit to the whole test vector. Its range is represented by the exponent of a given base. Therefore, the width of a slice can be represented by Equation 6.4. Since the finish condition of S-LOC is when all bits in the slice have been checked, the variation of its execution time can be significant. In addition, we also continue to use Max One, which is a fitness function with fixed execution time, to imitate the performance of FS-MS GA with this kind of function.

$$Slice = Test\ Vector\ (Base^{SliceRange+1} - 1\ down\ to\ 0) \tag{6.4}$$

Experiment Platform

To evaluate the performance of FS-MS GA, we propose an experiment platform which is an embedded system based on Xilinx VC709 . On this platform, we implement the FS-MS GA in VHDL and continue to use the LS-MS GA evaluated in Section 6.2.1. We mount these two implementations on the AXI bus with interfaces. The FS-MS GA and LS-MS GA operate simultaneously with either Max One or S-LOC as their fitness function, since the resources cost of Max One and S-LOC are low.

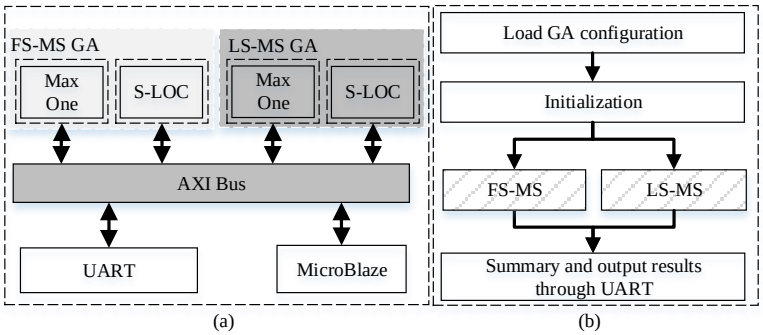

Figure 6.8: (a) Experiment Platform, (b) Testing Process.

The testing process has been shown in Figure 6.8b. It starts from loading GA configuration (number of fitness, crossover rate, mutation rate, size of population and so on) to testing components by a MicroBlaze. Thereafter, the FS-MS GA and LS-MS GA will be enabled for search or evaluation, until the finish condition (either a specific number of generations has been analysed or at least one suitable candidate has been found) has been achieved. Then, the MicroBlaze collects data from each component and organises these results. The results are output through a UART port.

Experiment Configuration

The experiment follows the configuration as follows:

- the same GA configuration could have same affaction on both FS-MS GA and LS-MS GA since the GA operations components (crossover and mutation) are maintained,
- we use the GA configuration in pervious section
  - crossover rate is 0.5%,
  - mutation rate is 0.01%,
  - max number of generation is 50,
  - number of repeated tests is 1,000,000;
- various population and number of fitness function will affect the performance:
  - population size is 6, 8, 16,
  - number of fitness functions is 2, 3, 4, 5 when Population is 6,
  - number of fitness functions is 2, 4 when population is 8 and 16,
  - fitness functions are Max One (256 bits) and S-LOC (259 bits).

Results Analysis

The detailed results are shown in Table 6.1 (No.FF refers to Number of Fitness Functions; the results in this table are the average number of clock cycles used by each generation). The percentage improvement of FS-MS GA over LS-MS GA with both Max One and S-LOC are shown in Figure 6.9 and 6.10 respectively.

From Figure. 6.9, it can be seen that although the FS-MS GA can reduce the number of clock cycles used for fitness functions, the improvement is slight, when the fitness function is Max One. In addition, this improvement has an upper bound which arises when the number of fitness functions is half the population size. The reason is the same as the phenomenon that is described in Section 6.2.1 and explained by Figure 6.3.

Table 6.1: FS-MS GA vs LS-MS GA Results Table.

		Max One			S-LOC		
PopSize	No.FF	LS-MS	FS-MS	Improvement(%)	LS-MS	FS-MS	Improvement(%)
6	2	786	777	1.15	107	77	27.65
	3	530	520	1.89	97	61	37.1
	4	530	520	1.89	101	61	39.82
	5	530	520	1.89	102	60	40.82
8	2	1048	1036	1.15	173	112	34.88
	4	536	522	2.61	142	81	43.05
16	2	2096	2072	1.15	487	296	39.09
	4	1072	1056	1.49	381	118	69.00
				1.65			41.43

Note: No.FF refers to number of fitness functions.
It indicates how many fitness functions (same fitness function) will be used.

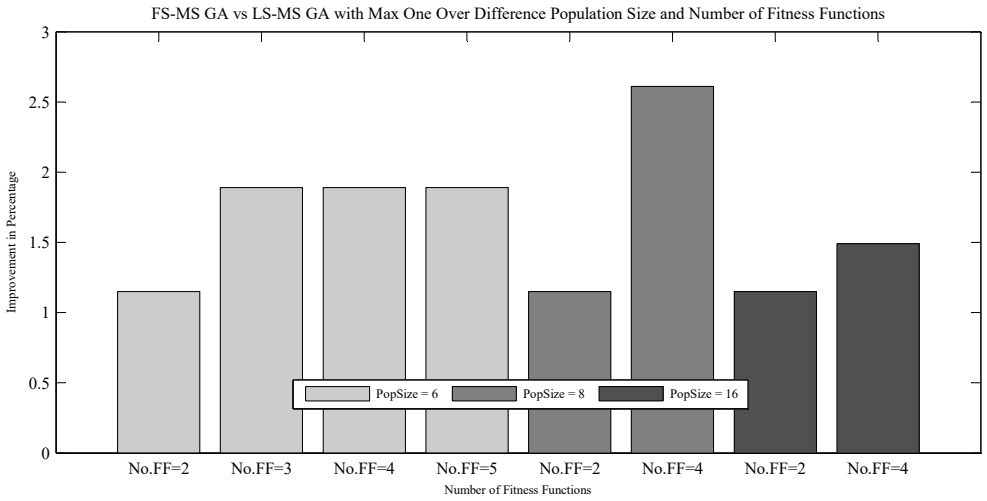

Figure 6.9: FS-MS GA vs LS-MS GA with Max One.

Note: No.FF refers to number of fitness functions.

However, for S-LOC the performance can be significantly enhanced, as shown in Figure 6.10. It can be seen that the improvement of S-LOC is increased with the number of fitness raised. Taking a population size of 6 as an example, the improvement is rapid at the beginning and converges afterwards. Therefore, this improvement has an upper bound, which is determined by the variability and number of fitness functions against population size.

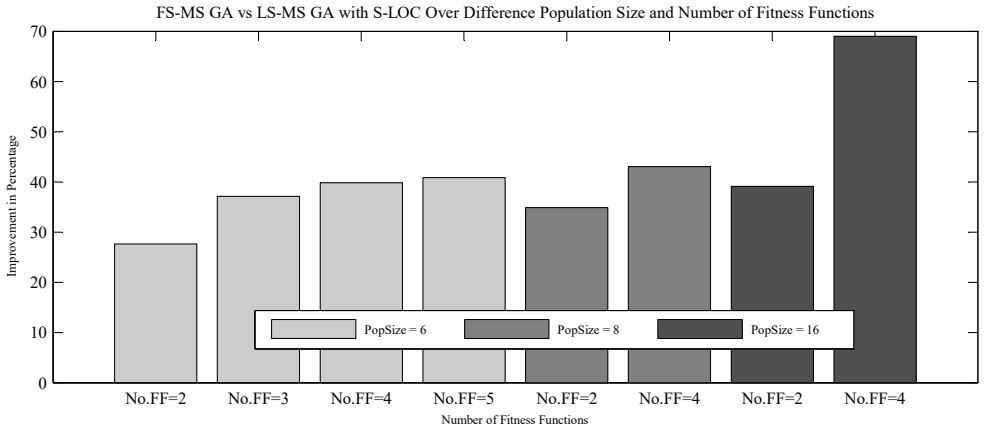

Figure 6.10: FS-MS GA vs LS-MS GA with S-LOC.

Note: No.FF refers to number of fitness functions.

6.2.3 Summary

In this section, we estimate the performance of using a Master-Slave GA to enhance the search efficiency, in order to apply it as a dynamic mapper for task mapping problem in hard real-time NoCs. From the results we can generally expect a linear improvement with the number of fitness, when this fitness is much complex than GA search. At the same time, we also find a shortcoming of the existing MS GA affected by its fitness function loading strategy, when the fitness funtion has variable execution time. Therefore, to alleviate this disadvantage, we proposed a possible architecture and evaluated it by experiments. The results show that the overall evaluation time can be improved in each evolved generation but with an upper bound which is determined by the variability and number of fitness functions against population size. In addition, considering the implementation platform, there are other benefit we can obtain by using hardware charateristics. It will be discussed in the following section.

6.3 Accelerated GA Operators

As discussed before, the GA search efficiency can benefit from both HWGA and the Master-Slave model. The GA operators (crossover and mutation) in both of them, however, still inherit the processing strategy used in sequential SWGA. The question focused on in this section is whether there are any changes that could be made to enhance the efficiency of these two operators, in order to further accelerate the HWGA or FS-MS GA. This is approached by considering, in order, strategy limitation analysis, accelerated operators implementation, and an experiment with results analysis.

6.3.1 Strategy Limitation

The processing strategy of GA operator adapted by HWGA and FS-MS GA is inherited from sequential SWGA, whose target is to execute it in software on regular CPUs which are limited by operating bit-vectors (similar to the example shown in Figure 5.1, page 108) and pipelined architecture. This subsection will concentrate on attempting to improve these two aspects.

Low Efficiency of Bit-Vector Operation

The low efficiency of bit-vector operation in the existing processing strategy can be seen in terms of both crossover and mutation. The crossover operator in GA requires the swap of selected parts of two parent chromosomes (if the crossover condition has been satisfied), as shown in Figure 6.11a. Since chromosomes are normally stored as arrays, software will inevitably swap these two arrays element by element, as presented by the black dash arrows. Such operations will invariably take several clock cycles of a typical CPU, even in the case of partial swaps. In addition, the number of clock cycles required will be further increased along with extensions in the size of chromosomes. The mutation operator can suffer similar limitations in the existing processing

strategy. Its working process requires checking each gene to determine whether the observed gene should be mutated or not and generating a mutated value if needed. An example is shown in Figure 6.11b. This process also requires multiple clock cycles of a typical CPU, as well as a scaling up of that time with the increase of the number of genes.

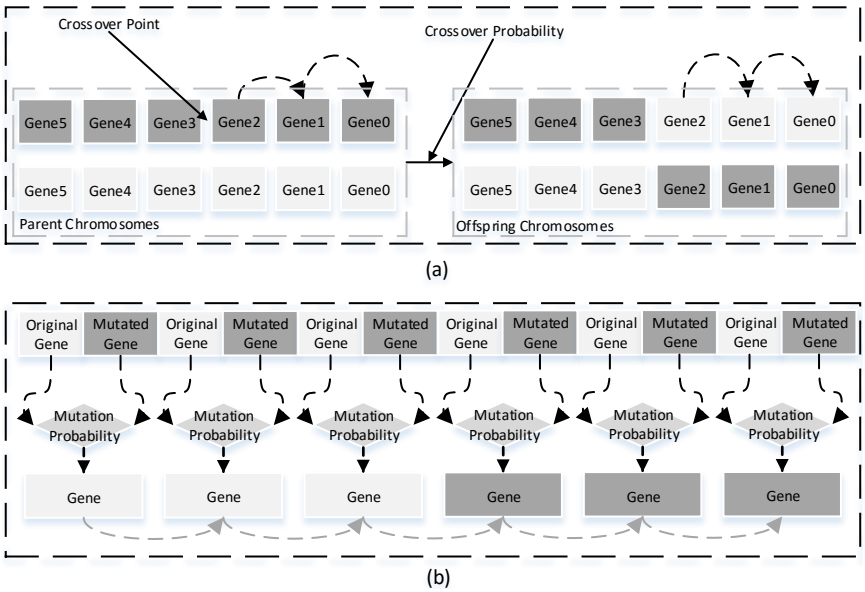

Figure 6.11: (a) Vector Operation of Crossover Operation, (b) Vector Operation of Mutation Operation

Lack of Pipelined Architecture

In addition, the existing processing strategy also lacks provision of an effective pipelined architecture. This can be seen from the working procedure of reproduction (producing offspring chromosomes by using crossover and mutation). Its working procedure can be either applying crossover over the whole parent population first and then using mutation to generate the final offspring population, or each time applying crossover and mutation sequentially only over two selected parent chromosomes and repeating this process until the whole offspring population has been generated. These two kinds of procedures are illustrated in Algorithm 2.

Algorithm 2 Reproduction Working Process

```
1: procedure TYPE 1
2:     for Number of offsprings < Population Size do
3:         Select parent chromosomes
4:         Crossover
5:     for Number of offsprings < Population Size do
6:         Mutation
7: End Procedure
```

```
1: procedure TYPE 2
2:     for Number of offsprings < Population Size do
3:         Select parent chromosomes
4:         Crossover
5:         Mutation
6: End Procedure
```

Regardless which type is selected for use, when one operator is executing, the other has to be paused. This phenomenon is shown in Figure 6.12a and 6.12b. This will increase the computation time compared with a pipelined architecture, with the timing difference shown in Figure 6.12c.

It can be seen that reproducing the first offspring chromosome uses the same amount of time as the two procedures in Algorithm 2. However, after the first offspring chromosome, in each stage there will be one new candidate chromosome generated. Assuming the number of clock cycles used by selection, crossover and mutation are the same and equal to N, and the population size follows the example in Figure 2.18, page 70, the total numbers of clock cycles to finish reproduction can be represented by $6 * 2 * N$ for both Type 1 and 2 Algorithm 2. The total number used by pipelined architecture should be $(3 + 3) * N$. Thus, pipelined architecture can be used to improve the timing performance of reproduction and this improvement will be significant when the population size increases.

Both bit-Vector operation and pipelined architecture are shortcomings present in the current processing strategy GA operators adopted by HWGA and FS-MS GA. They can affect the execution time of operators themselves and

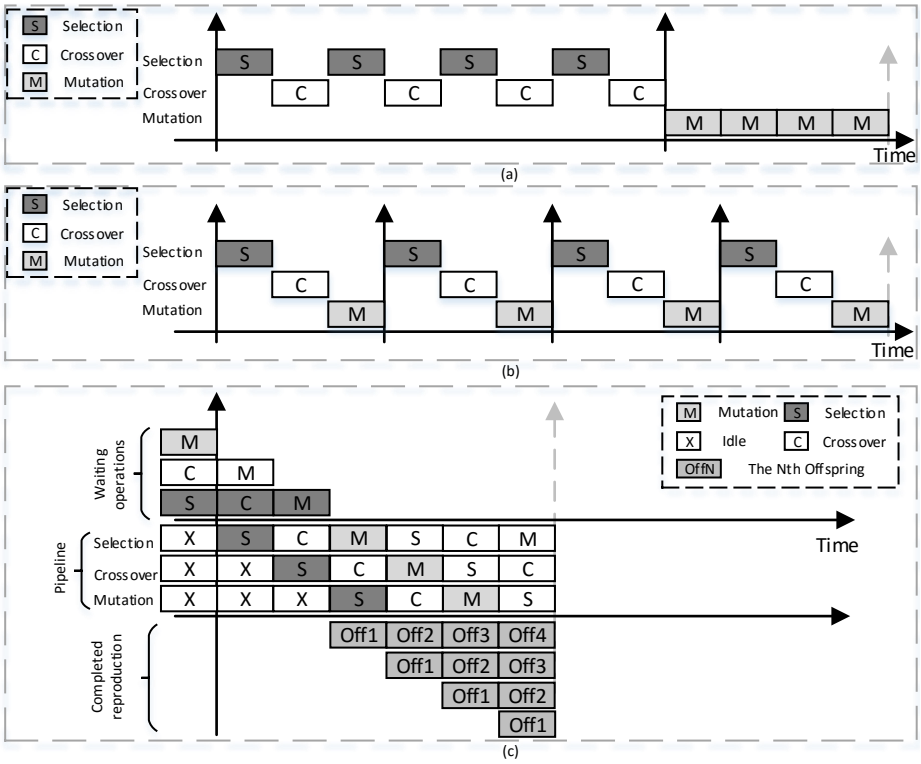

Figure 6.12: (a) Type 1 running time, (b) Type 2 running time, (c) Pipelined structure running time.

Note: Following example of Figure. 2.18.

further influence the overall processing speed. Therefore, modifying the current processing strategy could be a way to enhance the performance of GA operators and thus improve the search efficiency of the GA.

6.3.2 Accelerated GA Operators

The bottle-neck of GA operators which is caused by the existing processing strategy is considered in this subsection, which proposes two hardware accelerators to improve the crossover and mutation operators respectively. In addition, a pipelined architecture for reproduction is also introduced and assembled with these two accelerators for an optimisation of the overall execution time of reproduction.

Crossover

For crossover operator, we try to swap two parent chromosomes by using a Crossover Mask and several logic gates (such as 'NOT', 'AND' and 'OR') according to crossover point and possibility. The Crossover Masks are a series of pre-designed vectors determined by both the number of genes and the width of gene (the number of bits to represent one gene). An example of how the proposed crossover works is shown in Figure 6.13 where each gene is represented by 1-bit and the total number of genes is 10.

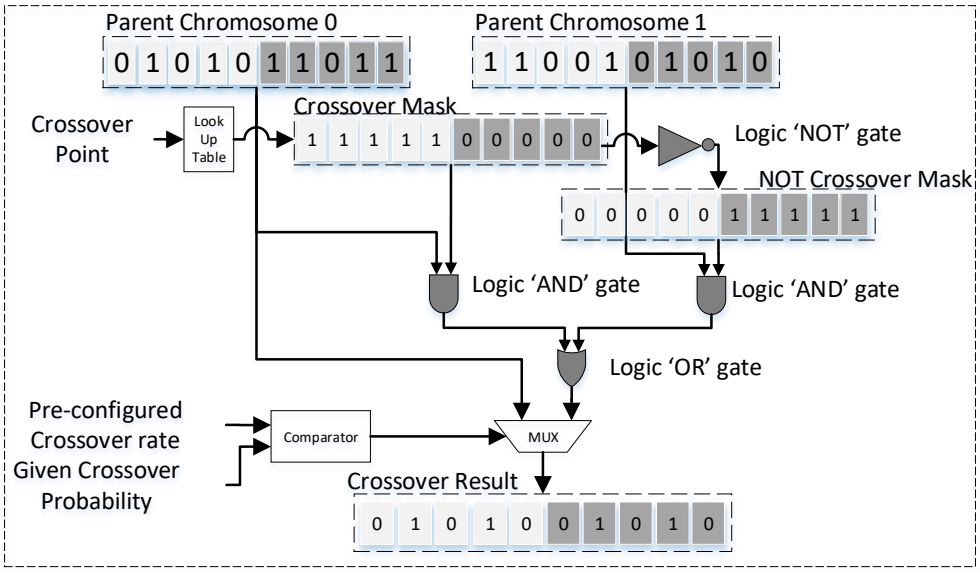

Figure 6.13: Crossover Component

The crossover point (randomly generated by a Random Number Generator) will be used as the index of a look-up table which stores the Crossover Masks. The two parent chromosomes will be transferred through two logic 'AND' gates with Crossover Mask and NOT Crossover Mask respectively. The results of logic 'AND' gates will be applied as the inputs of a logic 'OR' gate to generate the potential crossover result. Whether the final crossover result should be the swapped result (potential crossover result) or the original parent chromosome is determined by a comparison between the pre-configured crossover rate and a given crossover probability, which is also randomly generated by a Random

Number Generator. If the crossover condition has been satisfied, the final crossover result will be the swapped result, otherwise the parent chromosome will be selected.

Since the propagation delay among ROM and combinational logic is slight, when the frequency of the whole system is not extremely high, the proposed crossover operation can be finished within one clock cycle no matter where the crossover point is or how many genes there are. For an extremely high frequency system, we can introduce a pipelined structure to break down this working process, in order to allow this operation to be finished within one clock cycle. Thus, by using this accelerated crossover operator the processing speed can be improvement significantly when compared with the existing process strategy which is shown in Figure. 6.11a.

Mutation

Similar to crossover, the idea of accelerating the mutation operator is also based on mask vectors. The working process of the proposed accelerated mutation operator, which can be divided into Mutation Template Generator and Mutating component, is illustrated in Figure. 6.14b and 6.14c.

Mutation Template Generator

In Figure 6.14b, the Mutation Template Generator consists of a Mutation Possibility Mask and a Mutated Value Mask. These are used to determine whether each gene of a chromosome should be mutated and provide the mutated values when needed. Their generating procedure can be described as follows. In each clock cycle, a given mutation probability (randomly generated) will be compared with the pre-configured mutation rate (similar with crossover rate). The result of this comparison will be used to indicate whether the current gene should be mutated. If the current gene needs to be changed, one bit logic '0' will be shifted into the Mutation Possibility Mask and a random generated mutated value will be shifted into the Mutated Value

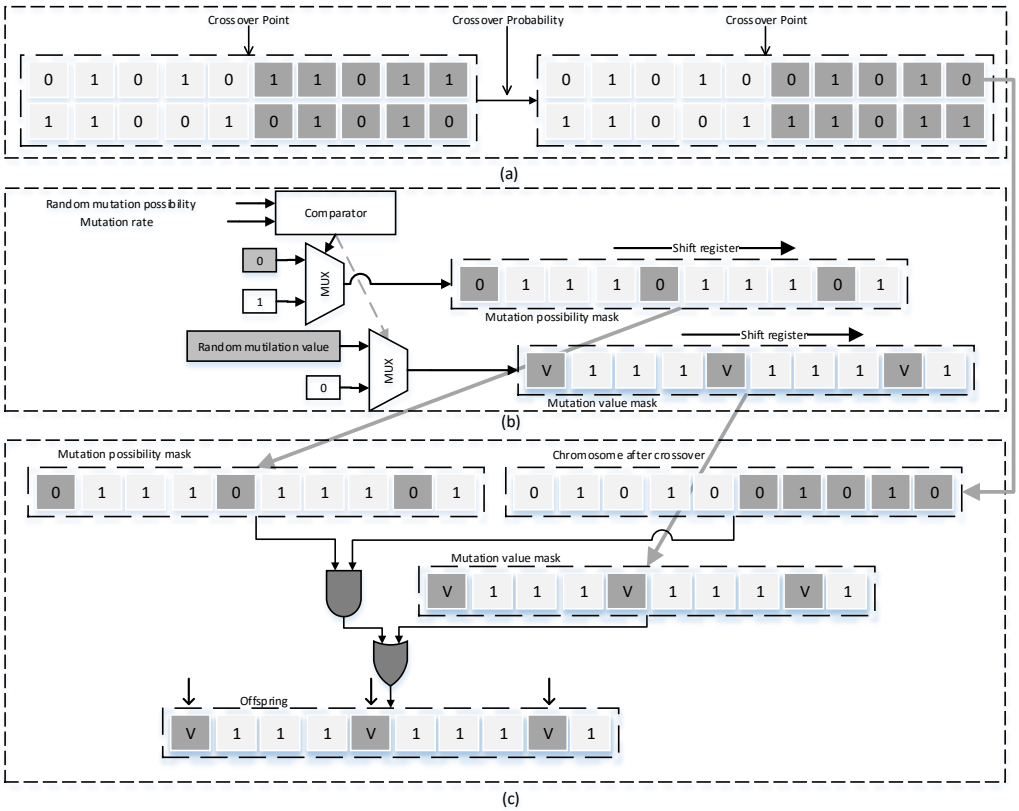

Figure 6.14: (a) Crossover Strategy Example, (b) Mutation Template Generator, (c) Mutating component Example.

Note: Number of gene is 10; width of gene is 1;
the input vectors of Figure 6.14c are from both Figure 6.14a and 6.14b;
the "V" represents the mutated value.

Mask. Otherwise, if the current gene should be maintained, one bit logic '1' and a logic '0' vector (all bits are logic '0' if a gene is represented by multiple bits) will be shifted into the Mutation Possibility Mask and the Mutated Value Mask respectively. This process will be repeated until all genes of a chromosome have been checked.

Mutating component

The mutating component will read the mutation template. In its working process, shown in Figure 6.14c, the result of the accelerated crossover operator

will be assigned to a logic 'AND' gate with the Mutation Possibility Mask. Their result will be applied as one input of a logic 'OR' gate to calculate the final offspring with the Mutated Value Mask. Similar to the accelerated crossover operator, all the operations in this mutating component follow combinational logic, and the delay among them is only a propagation delay. Therefore, the mutation operator can be finished within one clock cycle regardless of how many genes a chromosome has. The accelerated mutation operator can be much faster than the existing processing strategy , especially when the number of genes is large.

Further Optimisation

From the Mutation Template Generator (Figure 6.14b), it can be noticed that only one gene's template can be generated within one clock cycle. However, this does not mean that this idea cannot be used to accelerate the mutation operator. In order to solve this problem, a template FIFO (First-In-First-Out) is introduced to store mutation templates before the mutation operator is executed. Since one of the natural characteristics of hardware is parallel computing, we can easily launch the Mutation Template Generator to produce and store templates when GA is in another stage, such as candidate evaluation. In addition, some chromosomes' templates can be generated when the mutation operator is executing as well. We can use the following variables to find the minimum depth of FIFO:

- the population size is m;
- the number of gene is n;
- the minimum depth of FIFO is x.

In this accelerated mutation operator m clock cycles are required to finish the mutation operation over the whole population, n clock cycles are needed to generate one template. Currently only x chromosomes' templates are ready in FIFO. The worst-case situation can be that the accelerated mutation operation

and the Mutation Template Generator (to generate the rest templates) are released at the same point. In other words, the remaining templates have to be prepared within m clock cycles. Thus, we can get Equation 6.6. Therefore, the depth of FIFO can be minimised to $\frac{(n-1)m}{n}$, in order to reduce the resource cost.

$$m \geq (m - x) * n \tag{6.5}$$
$$x \geq \frac{(n-1)m}{n}$$

Reproduction Pipeline

As mentioned above, the existing processing strategy also lacks a pipelined architecture. To solve this limitation, we assembled the proposed crossover and mutation operators with a possible pipelined architecture with additional registers, as presented in Figure 6.15. By applying this architecture, the first two clock cycles will generate two invalid offspring. However, after that there will be two valid offsprings reproduced every clock cycle. Therefore, the execution time of reproduction can be reduced.

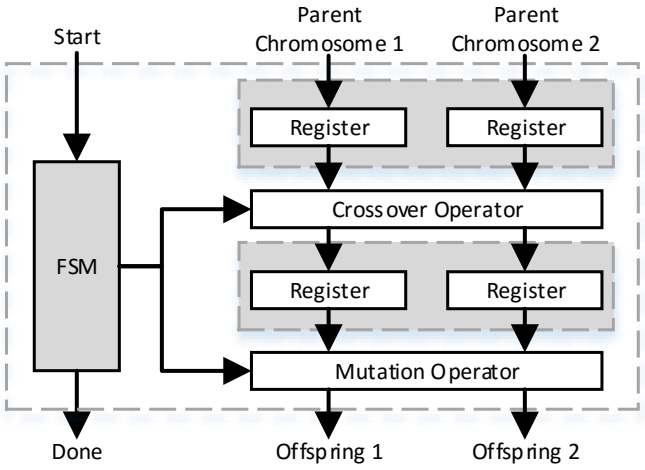

Figure 6.15: Reproduction Pipeline Architecture.

6.3.3 Experiment and Results Analysis

The extent to which the proposed accelerated GA operators can alleviate the shortcoming caused by the existing processing strategy of GA operators can be assessed by experiments. In this experiment, these two accelerated GA operators will be evaluated within FS-MS GA on the platform used in Section 6.2.2 and with the same GA configuration, since the GA configuration will have no affection on accelerated GA operators efficiency.

Results Analysis

Detailed results are shown in Table 6.2 (the results in this table are average number of clock cycles used by each generation). It can be seen that in all situations the proposed accelerated reproduction will use a lower number of clock cycles than existing GA operators.

Table 6.2: Existing GA Operators vs HW Accelerated GA Operators Table.

		Max One			S-LOC		
PopSize	No.FF	E-GA-O	A-GA-O	Improvement(%)	E-GA-O	A-GA-O	Improvement(%)
6	2	1539	693	54.97	1576	1394	11.55
	3	1543	937	39.27	1564	1358	13.17
	4	1540	941	38.90	1561	1359	12.94
	5	1537	934	39.23	1558	1361	12.64
8	2	2179	1104	49.33	2097	1853	11.64
	4	2196	1561	28.92	2194	1924	12.31
16	2	4138	1824	55.92	4160	3653	12.19
	4	4178	2816	32.60	4335	3779	12.83

Note: No.FF refers as number of fitness functions.

E-GA-O and A-GA-O refer as existing GA operations and accelerated GA operations.

The improvement over Max One is shown in Figure 6.16. It can be seen that the performance will suffer a decrease by changing the number of fitness function from 2 to 3, but it remains afterwards (does not decline significantly further after this). This is because Max One is a fixed and relatively slow

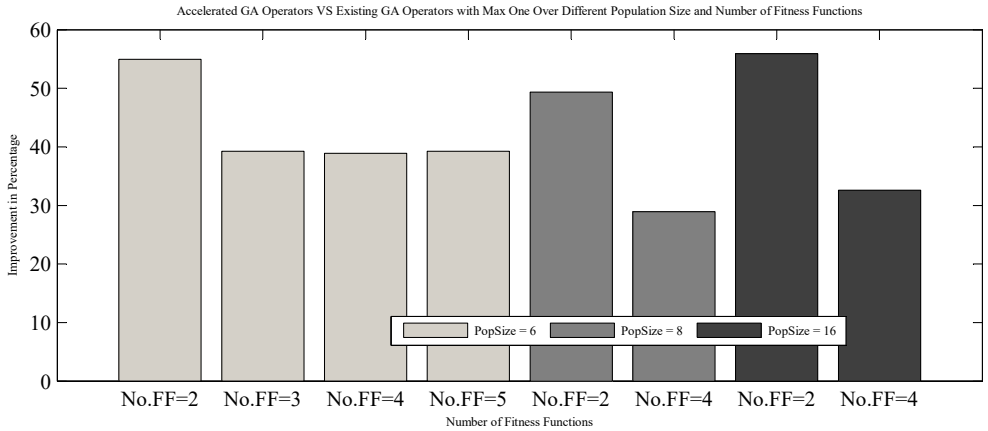

Figure 6.16: Accelerated GA Operators vs Existing GA Operators with Max One as Fitness Function.

Note: No.FF refers as number of fitness functions.

evaluation method compared with S-LOC. When there are two fitness slaves, mutation templates are not consumed very fast. The pre-stored templates are enough to support Max One. However, when the number of fitness functions increases to 3, the speed at which templates are consumed increases. The mutation template FIFO will soon be out of stock and make the following operations pause while waiting for new templates. Fortunately, there is an upper bound for this pause, because the template consumption speed has an upper bound. This non-linear phenomenon can be seen from Figures 6.2, 6.9 and can be explained by Figure 6.3.

For S-LOC, the improvement is not significant, and difficult to change along with the variation of number of fitness functions. This is because it can be very fast in most situations, and even two fitness functions can already push it to its upper bound pause.

Therefore, these two accelerated GA operators can make a significant improvement to fitness functions that have a long execution time such as Inexact HW-E2ERTA. These two improvement methods proposed here can be assembled together in a single architecture, as presented here, or applied individually in the case of platforms with limited hardware resources, since they are not mutually dependent.

6.4 Summary

In this chapter, in line with the problem breakdown discussed in Section 3.3, we focused on the search algorithm itself to explore the factors that can affect its search efficiency. To indicate which type of GA can be used in this research, we evaluated various GAs step-by-step by using experiments, from both implementation and model aspects. From the experiments, we found two limitations in the current Master-Slave GA: lock-step problem and implementation limitation in GA operators (crossover and mutation). We then proposed an asynchronous architecture and accelerators to alleviate these two shortcomings. The results show that by applying these modifications and accelerators, the search efficiency can be improved. This improvement will be applied in the next chapter to enhance the dynamic mapping search for hard real-time NoCs in order to contribute to the resolution of the research problem in this thesis.

Chapter 7

Dynamic Mapping in Hard Real-Time NoCs

As reviewed in Chapter 2, the timing performance of a NoC is improved by carefully planned task allocation. However, the state-of-the-art responses cannot adequately address dynamic mapping of task distribution, since they are either affected by low rates of remapping success or lack a guarantee for systems timing performance after remapping, especially in hard real-time systems. Thus, these two reasons motivate this research to focus on the mapping optimisation of real-time NoCs, in particular, dynamic task allocation in hard real-time systems.

In order to understand how to find/construct a suitable mapping for a hard real-time NoC to satisfy the timing requirements efficiently and dynamically, Section 3.2 divides this problem into two steps, depending to whether task migration cost is considered. In addition, using a system model, search and evaluation methods to be used are also suggested. However, these methods are not problem free due to their complex search and calculation processes. Therefore, Chapters 4, 5 and 6, propose techniques and implementation methods to accelerate their execution speed.

In this chapter, we gather together the improved methods (Inexact HW-E2ERTA and HW FS-MS GA) to verify their performance through experiments and discussions in applying them to address the hypothesis proposed in Section 3.2.

7.1 Hypothesis Step One - Fast Static Mapping

Following the analysis in Section 3.2, without considering task migration time after remapping, the remapping overhead can only be affected by the execution of the dynamic mapping algorithm, which consists of mapping search or construction and mapping evaluation. In addition, before a new event (e.g. the change of system working environment) which triggers a system model change and task remapping occurs, a dynamic task allocation process is treated as a fast static mapping process. This is because the activated tasks have already been determined by the last event and the system status is maintained until a new event happens. Therefore, the first step needed in verificayion is how quickly the proposed dynamic mapper makes a mapping decision. The verification can be obtained via the experiment detailed in this section. This is discussed in the experiment platform, experiment configuration, and results.

7.1.1 Experiment Platform

To evaluate the performance of the proposed dynamic mapper, an experiment platform was established which inherits from the one used in Section 5.2.3, shown in Figure 7.1a. On this platform, we mount the proposed potential mappers (listed in Table 7.1) on an AXI bus with bus interfaces.

Since the FPGA resources are limited, only one mapper can be executed at any

Table 7.1: Proposed Potential Dynamic Mappers.

Index	Potential Mapper
1	HW LS-MS GA + Inexact HW-E2ERTA
2	HW FS-MS GA + Inexact HW-E2ERTA

Note: HW refer to hardware implementation,
LS-MS and FS-MS refer to Lock-Step Mast-Slave and Free-Step Master-Slave.

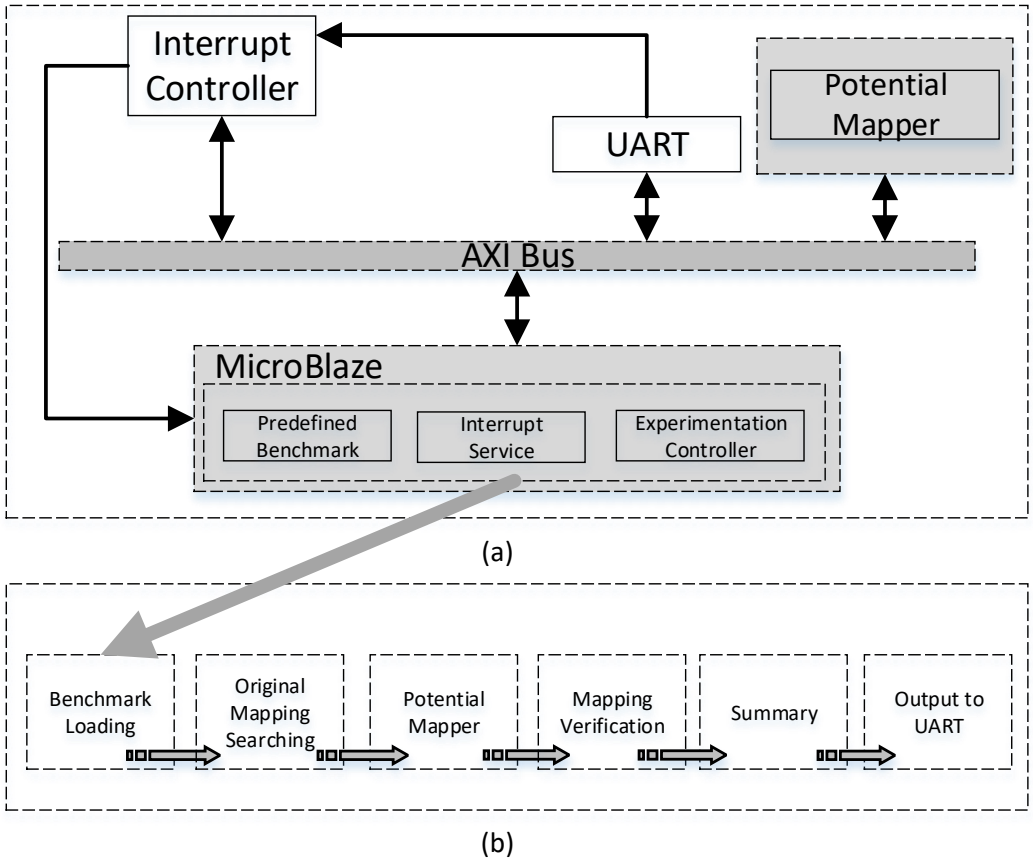

Figure 7.1: (a) Experiment Platform, (b) Testing Process.

one time. This means that a test set consists of a series of sub-tests. To secure a fair experiment environment, the synthetic benchmarks which are generated at run time cannot be used directly, as they may change in different executions. Therefore, a predefined benchmark (a synthetic benchmark generated offline) is stored on the platform. Each sub-test begins with the benchmark loading

process through MicroBlaze to mappers. Thereafter, MicroBlaze will enable the evaluation and collection of data from mappers. The organised results are output through a UART port.

7.1.2 Experiment Configuration

To measure the performance of the potential dynamic mappers, the experiment configuration is divided into four parts, as follows:

- FPGA platform:
  - system input clock frequency is 50 Mhz,
  - AXI bus operation frequency is 100 Mhz, it is also used by dynamic mappers,
  - the system is compiled by Xilinx Vivado 14.3;
- NoC platform configuration:
  - the size of a target NoC is 10*10, since this the largest size we can test on VC707 platform, with current implementation;
- Benchmark configuration:
  - the size of task sets is 128 (the largest task set can be supported by VC707),
  - maximum period is $2^{16}-1$ clock cycles (much more than the period used by realy bench mark Autonomous Vehicle, and easy to match the bit width of Xilinx IP cores such as RAM, divider, multiplier and so on,
  - the utilization of task and flow is from 10% to 90%, average around 40%,
  - the number of flows is considered as the size of the task set,
  - each task generate one flow, so there is no shared priority,

  - the destination of a flow is selected randomly,
  - the priority and deadline of a flow inherit from its originated task;

- GA configuration:
  - following the suggested GA configuration in [101],
    - crossover rate is 0.5%,
    - mutation rate is 0.01%,
  - variours between population size and number of fitness functions will affact the performance of dynamic mapper,
  - following the configuration used in Chapter 6 to cover small, medium and larger (largest can be supported by VC707) population and number of fitness functions,
    - population size is 6, 8, 16,
    - number of fitness functions is 2, 3, 4, 5 when population is 6,
    - number of fitness functions is 2, 4 when population is 8 and 16, because of the hardware resources limitation,
  - dynamic mapping
  - max number of generation is 500,
  - number of repeated tests is 100,
  - selection strategy is modified ranking (keeps species diversity).

The reason for setting the size of NoC to 10*10 and number of tasks to 128 can be explained as follows. First 10*10 and 128 are the largest configurations can be support by VC707 with current implementation.

Second, from the mapping difficult point of view, the applied bench mark is not a easy configuration to be mapped. An easy configuration can be a small NoC with simple application such as 4*4 NoC with Autonomous Vehicle Application (TB1, 33 tasks and 38 flows, average utilization around 19.15%). In 4*4 NoC with TB1, the average utilisation on each IP is around 39.50%

($\frac{33}{16} * 19.15\%$) which is a low competition on each IP. In addition, the lower number flows can be organised well with low communication congestion on the NoC. Therefore, it is relatively easy to find a suitable mapping solution. This can be seen from Figure 4.4. The search process could be finished within 28 generations, even with PRE alone, which cannot guarantee the final evaluation results for an end-to-end response time analysis. Moreover, the exapmles of moderate configuration can also be seen on large NoCs, such as Figures 4.7 and 4.8, with the Synthetic Application TB3 (100 tasks and 100 flows, average utilization around 41.30%). Although the average use of tasks has increased to 41.30%, these examples benefits from the extended NoC platform (from 4*4 to 9*9 and 10*10). The larger NoC results in the average use on each IP remaining moderate (50.99% and 41.30%).

Third, compared with hard configurations, the proposed benchmark will be easier to explore. A hard configuration can be asmall NoC with complex application such as 4*4 NoC with Synthetic Application TB2 (50 tasks, 50 flows, average utilisation around 29.73%). In a 4*4 NoC with TB2, the average number of tasks on each IP is around 3.125 ($\frac{50}{16}$). This results in the average usage on each IP being around 92.91% ($3.125 * 29.73\%$). Therefore, a suitable mapping solution is almost impossible to find. This phenomenon has been shown in Figure 4.5. No mapping solution was found within 50 generations, and the best result it can achieve is 13 (number of unschedulable tasks and flows).

Table 7.2: Difficulty of Benchmark Configuration.

Difficulity	Average Utilization on Each IP
Hardest	92.91%
Second Hardest	51.20%
Moderate	50.99% and 41.30%
Easy	39.50%

By contrast, the benchmark applied in this experiment (average utilization on each IP in this benchmark would be 51.20% ($\frac{128}{100} * 40\%$)) is the second hardest configuration, shown in Talbe 7.2. It keeps the possibility of find a

suitable mapping and is also a problem that not easy to be solved. Therefore, we choice this benchmark in our experiment.

Furthermore, the diversity of species was also considered. This is because directly using ranking to imitate natural selection would result in the system making the selection division using only the fitness value of candidate solutions. The fittest candidates will survive even if they are duplicated: this happens when two candidates have been selected to produce offsprings, but the crossover and mutation conditions are not all satisfied, in which case offsprings will be identical with their parents. This will soon make the population lose its diversity, especially in a small population like the one used in this experiment. This means the GA search will fall into a local optimal. Therefore, instead of only using ranking to imitate natural selection, we rank all the choromsomes first and then select the unduplicated from them in order to maintain the diversity of species. This benefits not only the search results but also search speed, because different candidate solutions may maximise their evaluation execution variability and maximise the improvement by using FS-MS GA.

7.1.3 Results

The results of the experiment are shown in Table 7.3. The data not shaded with grey shows the average number of clock cycles in each generation. The data shaded with gray shows the number of generations and time taken by FS-MS GA + Inexact HW-E2ERTA to find a schedulable task allocation.

Focusing on the number of clock cycles of each generation, it can be seen that the FS-MS GA improves the evaluation efficiency. However, the improvement is not significant compared with S-LOC, which improves it by around 41.43% on average, as shown in Table 6.1, page 153. This is because the execution variability of Inexact HW-E2ERTA is not as great as that of S-LOC, especially when all candidates tend to be optimal, which is hard to evaluate and extends the E2ERTA execution. For GA operators, they can be accelerated remarkably,

Table 7.3: Fast Static Mapping Evaluation

		LS-MS			FS-MS			Improvement (%)			FS-MS	
PopSize	No.FF	RD	FF	Overall	RD	FF	Overall	RD	FF	Overall	No.Generation	Time (s)
	2	793	243840	245639	4	228307	229332	99.49%	6.37%	6.64%	355	0.81
	3	792	229760	231572	4	214849	215859	99.49%	6.49%	6.79%	339	0.73
6	4	792	229540	231339	4	214735	215758	99.49%	6.45%	6.74%	358	0.77
	5	791	229180	230984	4	213985	214995	99.49%	6.63%	6.92%	334	0.72
8	2	1060	500711	503156	5	471670	473078	99.53%	5.80%	5.98%	310	1.46
	4	1059	251672	254131	5	228971	230361	99.53%	9.02%	9.35%	262	0.60
16	2	2115	896560	901937	9	870918	874214	99.57%	2.86%	3.07%	140	1.23
	4	2112	812150	817468	9	760172	763397	99.57%	6.40%	6.61%	138	1.06

Note: No.FF refers as number of fitness functions.
The fitness function is Inexact HW-E2ERTA.

since the execution of Inexact HW-E2ERTA is long enough for mutation accelerator to refill the mutation template FIFO, as explained in Section 6.3.2. However, the improvement from GA operators is not dominant in the overall search time, which is determined by the fitness function.

Focusing on search performance, all tests find a suitable mapping solution before the maximum number of generations (500). Although a greater number of generations are required than in the similar experiments in Figures 4.7 and 4.8, the search time used is much lower. The experiments in Figures 4.7 and 4.8 are supported by powerful PC (Intel Core i7-3770 CPU @ 3.4GHZ) and larger population (size=50). Their search time is around 1,200 and 700 seconds. Moreover, their benchmark (average utilization around 50.99% and 41.30%) is also easier than the one used here (51.20%). Although the maximum working frequency for FS-MS GA + Inexact E2ERTA is only 100Mhz, the search speed is accelerated significantly. In addition, if the population size is extended and more fitness functions are introduced, this improvement could be further increased. For the value of the search time, the value obtained is around millisecond level, which is also the level used to define the task deadline in real benchmarks (such as Autonomous Vehicle shown in Appendix.A). Therefore, the FS-MS GA + Inexact E2ERTA search can be treated as a real-time task used in real-time NoCs to search for mapping solutions dynamically.

7.1.4 Summary

In this subsection, following the first step in the hypothesis, the dynamic mapping search between the change of two system modes was trated as a fast static mapping process and combined the improvements achieved in Chapters 4, 5 and 6 to enhance the mapping search. From the results we can see that the FS-MS GA + Inexact HW-E2ERTA can significantly accelerate the mapping search compared with sequential GA and original E2ERTA in software version. The required search time can be reduced to millisecond level, which is similar to the deadline in real-time tasks in the real world. Therefore, it can be used as a dynamic mapper in hard real-time NoC dynamic mapping problems. In addition, dynamic mapping is a kind of fast static mapping. Therefore, FS-MS GA + Inexact HW-E2ERTA can also be used as an accelerated mapping search tool in the static mapping area to explore large design space (with a larger NoC or more complex applications).

7.2 Hypothesis Step Two - Minimizing Task Migration Time

Following the analysis in Section 3.2, if task migration time is taken into account, the remapping overhead will include not only the time used to make the mapping decision, but also the task migration time. In the previous section, the mapping algorithm execution time was reduced by applying HW FS-MS GA + Inexact HW-E2ERTA; however, the task migration cost was not considered. In this section, we try to reduce the task migration cost after a mapping decision has been made. This involves some modifications of the fitness function used in the previous section for task allocation evaluation (Inexact HW-E2ERTA) by introducing consideration of the number of tasks which need to be moved based on the new task allocation. This section discusses fitness function modification, experiment platform, configuration and result.

7.2.1 Multiple Objectives Fitness Function

One of the advantages of applying GA as a dynamic mapper is that it can easily be extended to consider multiple objectives simultaneously. Many examples of this have been shown in the static mapping literature, such as [61], [9] and [100]. Therefore, by following this idea, the task migration cost could be reduced by reducing the number of tasks that are required to be migrated from the original IPs to new IPs. In other words, it is the attemption to minimise the mapping differences between the original task allocation and the new one.

Fitness Function Modification

The methods to introduce multiple objectives in GA optimisation can be generally divided into NSGA and mathematical formula.

NSGA

In NSGA [32] and [126], the performance of a candidate is represented by a matrix, an example has been shown in Talbe 7.4. The matrix consits of various objectives of a candidate. Ranking among candidates is determined by matrix comparison and pareto classification, which provide comprehensive evaluation among candidate solutions, ensuring the diversity of species at the same time. An example can be seen from Table 7.4. The candidate A is the best, since it has no object which is worse than B and C. A is on the first pareto front, as shown in Figure 7.2. B and C are equally good candidates. Although, B is better than C in Physics, it is not as good as C in Math. They are allocated on the second pareto front. The NSGA selection starts from the first pareto front, due to the solutions on the first pareto front can dominate the one on other fronts, and move to the next front if necessory. At the same time, NSGA keeps the diversity of species by using the distance among the solutions on the same pareto front. The detials can be seen in [32].

Table 7.4: Example for NSGA Selection.

Candidate	Physics	Math
A	80	90
B	80	80
C	60	90

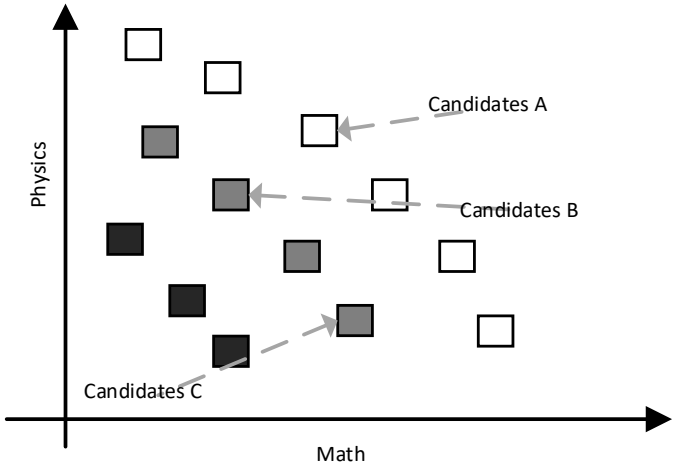

Figure 7.2: Example for Pareto Front.

However, the computation cost is very high. This means that the NSGA is not suitable in dynamic NoC problems, since these kinds of problems happen at runtime and are not supported by powerful computational resources, unless the NSGA can be optimised.

Mathematical Formula

Combining the evaluation of multiple objectives results in a mathematical formula can be an alternative method. The result of the mathematical formula represent the fitness of a candidate solution to the problem it targets. Although this combination may require extra knowledge about the relationship among the objectives, it requires far fewer resources than NSGA, which makes

it suitable for use in online optimisations. Therefore, it is selected in this research.

Implementation

Two potential formulas can be used in this research. One is $A + B$ style, which is suitable for multi-objective optimisation with equally important objectives. The other is $(A * K) + B$ style, which indicates the importance among objectives. It can be implemented by $(A * K) + B$ or $A + (B \div K)$. In this research the $A + B$ and $A + (B \div K)$ are selected and implemented. By considering the implementation platform, the division is implemented using the shift function. In other words, the divisor can only be 2^k. Since the timing performance after remapping is considered more important than task migration cost in this research, the new fitness functions are modified as follows:

- Inexact HW-E2ERTA + $Number_{mapping\ differences}$;
- Inexact HW-E2ERTA + $\frac{Number_{mapping\ differences}}{2^k}$.

Potential Limitations

The limitations of the mathematical formula method can be understood as the effects of the fitness landscape that can indicate the guiding ability of a fitness function. The reason for this can be explained in the following example. $m_{candidate}$ and $n_{candidate}$ are two candidate solutions for optimising objectives $A_{objective}$ and $B_{objective}$ simultaneously. By simply adding the value of objectives together, we can use variable Z to represent the final fitness value of $m_{candidate}$ and $n_{candidate}$ shown in Equation 7.1 with assumed values. It can be seen that although $m_{candidate}$ and $n_{candidate}$ perform differently in the view of both $A_{objective}$ and $B_{objective}$, their final fitness values are the same. In this case, this equation cannot identify which solution is better. This

phenomenon exists in E2ERTA, since the results of E2ERTA are the sum of number of unschedulable tasks and number of unschedulable flows. It cannot be distinguished which part makes greater contribution, only considering the results of E2ERTA. Nevertheless, from the previous experiments, E2ERTA and its improved versions can still guide the mapping search. However, whether this influence will become significant through being combined with number of mapping difference can only be verified with the following experiments. In addition, this effect could also influence the searching efficiency and extend the optimisation time.

$$\begin{aligned} Z(m_{candidate}) &= A_{objective}(m_{candidate}) + B_{objective}(m_{candidate}) &= 1 + 2 = 3 \quad (7.1) \\ Z(n_{candidate}) &= Aobjective(n_{candidate}) + B_{objective}(n_{candidate}) &= 2 + 1 = 3 \end{aligned}$$

7.2.2 Experiment Platform

To verify how well the proposed methods can reduce the task migration cost, an experiment platform (shown in Figure 7.3) is applied. It inherits the one used in Section 7.1, but only using the HW FS-MS GA + modified fitness functions. A predefined benchmark is stored on the platform. It is a synthetic benchmark, which is generated off-line with the assumption that it consists of pairs of sender and receiver. The pair of tasks are not related. An event (system model change) will cause the addition or removal of a pair/s of tasks. Since we consider the mapping difference, we search a schedulable mapping as the original task allocation before remapping on this platform (adding or removing tasks from it). As shown in Figure 7.3, each testing starts from the benchmark loading and mapping search process. Thereafter, the MicroBlaze will imitate the system change to enable the mapping search and collect data after it is finished. Another mapping evaluation process will be processed to verify the success of remapping before the organised data export.

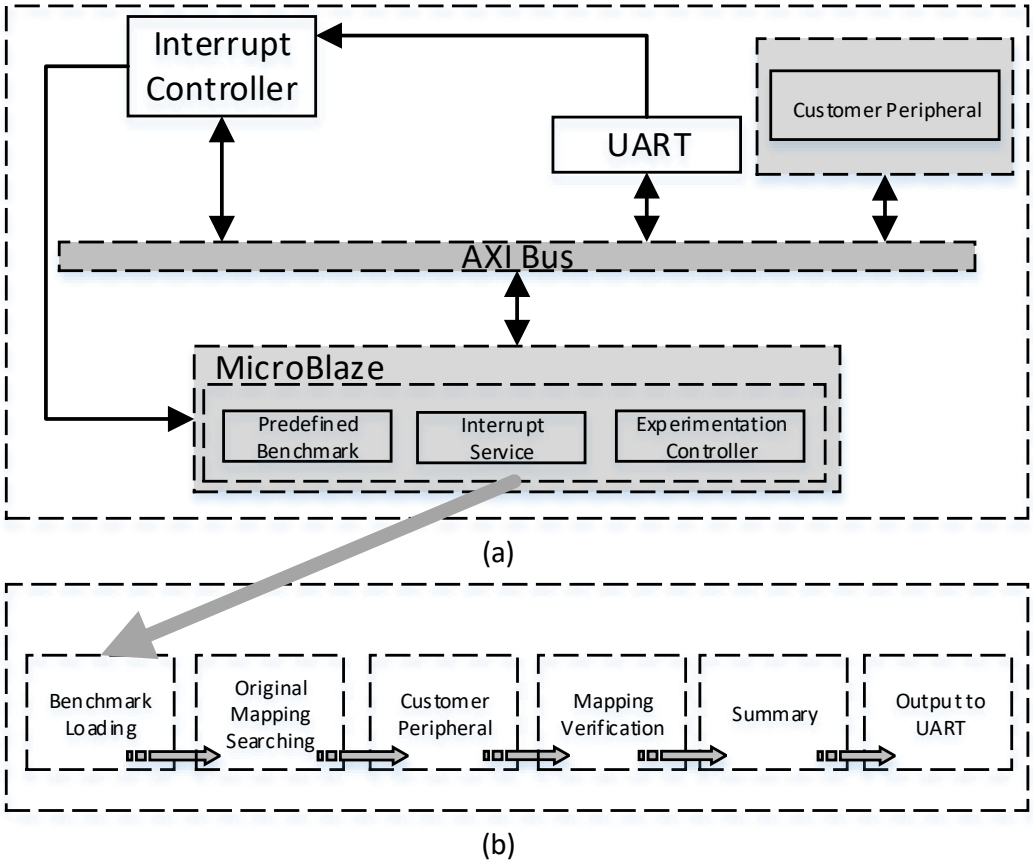

Figure 7.3: (a) Experiment Platform, (b) Testing Process.

7.2.3 Experiment Configuration

The configuration used in this experiment is listed as follows:

- FPGA Platform:
  - System input clock frequency is 50 Mhz,
  - AXI bus operation frequency is 100 Mhz, it is also used by the proposed mappers;
- NoC platform configuration:
  - the size of NoC tested are 6*6,

- Benchmark configuration:
  - the size of task sets is 50,
  - other parameters follows the configuration used in previous section;
- GA configuration:
  - following the GA configuration used in previous section,
  - population size is 6,
  - number of fitness is 2, 3, 4, 5,
  - number of repeated tests is 100,
  - terminate condition can be:
    - maximum generations having been evolved, or
    - fitness value of best solution is equal to number of newly added tasks for $A + B$ style, or
    - fitness value of best solution is smaller than one for $A + B \div K$ style;
- Fitness Function configuration:
  - K is set as 8 as an example;
- Task movement:
  - number of original running tasks on NoC is 30,
  - the number of adding tasks is 2, 4, 10, 20.

From previous experiment, we see a suitable mapping solution found in a 10*10 NoC with 128 tasks and flows. However, the fitness function it used is only Inexact HW-E2ERTA, since the task migration time after remapping is not considered. In this section, the fitness function we used is more complex than Inexact HW-E2ERTA and makes the dynamic mapping search become harder. Therefore, an appropiate configuration should make the maximum utilization on each IP arround the level used in previous experiment (51.20%).

In addition, the dynamic mapping experiment needs a base task allocation to imitate the system old state. Thus, we first randomly select 15 pairs of tasks and allocate them on the NoC and with respect to systems timing constraints. This will make the congestion level around 33.33% ($\frac{30}{36} * 40\%$) which is smaller than the easy level in Table 7.2 and gurantee the base task allocation can be found. Then we add pair/s of tasks (randomly selected) to the base task allocation. This increases the congestion level on the 6*6 NoC to 35.56% ($\frac{32}{36} * 40\%$), 34.78% ($\frac{34}{36} * 40\%$), 44.44% ($\frac{40}{36} * 40\%$) and 55.56% ($\frac{50}{36} * 40\%$), which covers from the under easy level to over the second hardest level of Table 7.2.

The reason we do not select other size of NoCs such as 4*4, 5*5 and 7*7 is because the maximum utilization on each IP with them is either too high or too low. This makes the dynamic mapping search too hard or too easy. For example, the maximum utilization on each IP with 4*4 NoC is 125% ($\frac{50}{16} * 40\%$), with 5*5 is 80% ($\frac{50}{25} * 40\%$) and with 7*7 is 40.82% ($\frac{50}{49} * 40\%$). Thus, we select 6*6 as the size to configure a NoC platform.

7.2.4 Results Analysis

The experiment results shown in Figures 7.4 and 7.4. From Figure 7.4 it can be seen that both $A + B$ and $A + (B \div K)$ styles can find a schedulable mapping solution at the same time minimising the task migration cost. The overall remapping time can be reduced to millisecond level.

$A + (B \div K)$ can achieve better results than $A + B$ style. This is because although $A + (B \div K)$ cannot provide linear guiding ability and is relatively easier to fall into local optimial, it can distinguish the fitness of candidates better than $A + B$. In this example, it always perform better over 100 times repeated evaluations.

Focusing on the change in the number of fitness functions, it can be noted that even if more fitness functions are introduced to evaluate candidates in parallel, the improvement is not significant when there are more than two

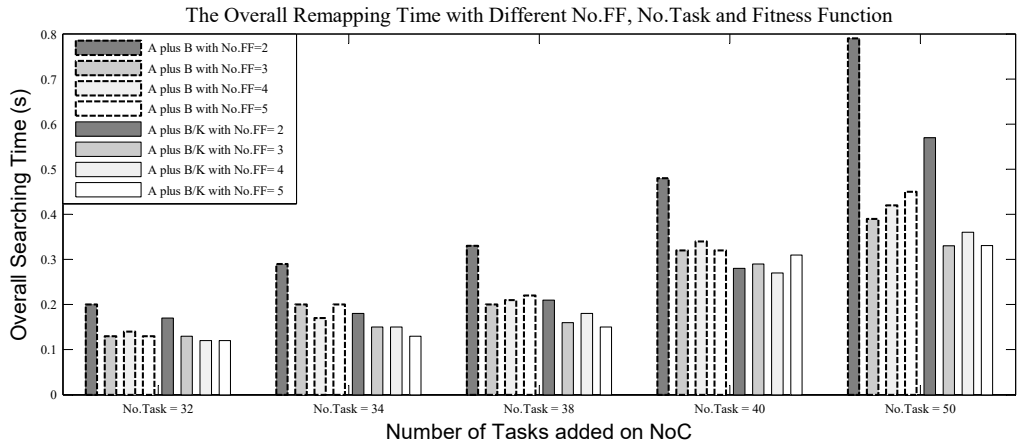

Figure 7.4: The Overall Remapping Time with Different No.FF, No.Task and Fitness Function Types.

Note: No.FF refers to number of fitness functions (same type, either $A+B$ or $A+(B \div K)$.)

fitness functions, which also happens in Section 6.2.2. This can be understood as follows. Firstly, although the execution time of Inexact HW-E2ERTA is variable, its variability is not as significant as the S-LOC used in Section 6.2.2, especially when the mapping search is approaching the optimal or best enough solution. This is because, after evolution, even if candidates are neither optimal nor good enough, they have already been improved and hardly to be evaluated. More execution time has to be spent in order to process fitness functions. The evaluation becomes slow and the variability of execution time fades away. Therefore, it will suffer the phenomenon we have found in Section 6.2.2 with Max One. The higher convergence rate the evolution can achieve, the quicker it will encounter the bottleneck of Master-Slave GA. Secondly, the execution time of Inexact HW-E2ERTA is much longer than the GA operations. The time saved by GA architecture is limited, especially with a small population. Thus, increasing the number of fitness functions more than two cannot make a significant improvement.

Another perspective which is worth mentioning is the remapping success rate. As discussed in literature review, a GA cannot guarantee to find optimal solutions, it can only provide good enough solutions. This means that the remapping may fail in some cases. Therefore, the remapping success rate was

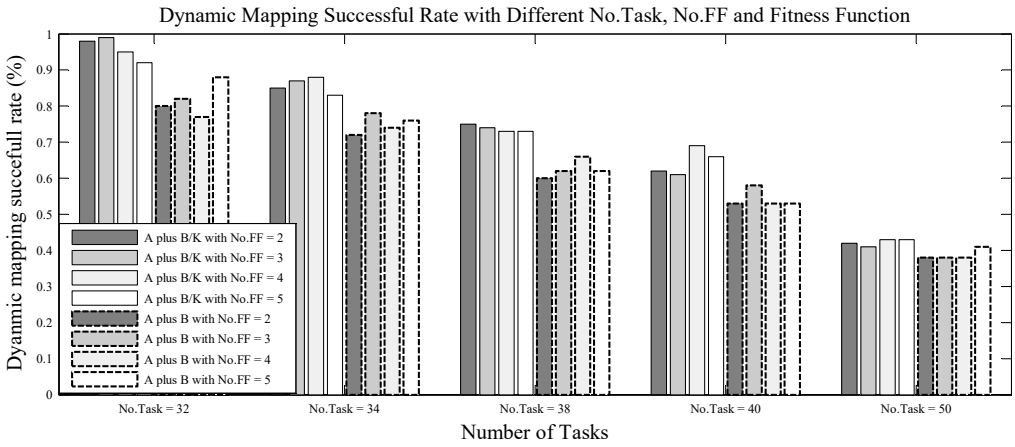

Figure 7.5: Dynamic Mapping Succes Rate with Different No.Task, No.FF and Fitness Function Types.

calculated over various configurations, which are shown in Figure 7.5. It can be seen that with increasing numbers of tasks, the remapping success rate reduces. This is caused by both the enlarged search area (more tasks evoloved) and the higher fitness requirement (minimising the mapping difference). Regardless of the experiment in previous section, attention must be paid here to the mapping difference, which increases the level of difficulty of the evolution. Therefore, in some cases the remapping may fail. Moreover, the increase in the number of fitness functions cannot change the remapping success rate remarkably. This is because the termination of evolution is after a fixed number of generations, if no good enough solution is found. Increasing the number of fitness functions can only accelerate the search speed. The search area is not extended. If there is no good enough solution in the search area, the remapping success rate cannot be improved. However, if the remapping time is fixed, the faster the dynmaic mapper can search, the higher success rate it will achieve. Therefore, in those condition, more fitness functions will make significant improvement.

This dynamic mapper is not limited to dynamic mapping problem solving. It can also be used in the static field as an accelerated optimisation tool in the NoC study.

7.3 Summary

Dynamic mapping in hard real-time NoCs cannot be adequately supported by existing mapping algorithms. They are affected by either low remapping success rates or lack of timing guarantee after remapping. In this chapter, following the problem analysis carried out in Section 3.2, a global remapping algorithm to alleviate this problem was proposed. This combined the improvements achieved in Chapters 4, 5 and 6 to enhance the mapping search and evaluation efficiency. The performance of the hypothesis was verified step-by-step.

In the first step it was shown that the FS-MS GA + Inexact HW-E2ERTA can be used as a fast mapping optimisation algorithm. This enables not only the possibility of remapping hard real-time NoCs dynamically, but also a fast optimisation for large design space which can be caused by extended NoC size or increased the complexity of application/s. Thereafter, the task migration cost was taken into account. By reducing the number of tasks needing to be migrated, the task migration cost can be reduced. Therefore, the research problem – dynamically finding a schedulable task allocation with respect to system hard real-time timing requirements in NoC – can be alleviated by applying the techniques and implementation presented in this chapter.

Chapter 8

Conclusion and Future Work

The Network-on-Chip (NoC) is an on-chip interconnection architecture to replace the currently established ones, such as Point-to-Point or shared bus, in both academic and industry. A hard real-time NoC needs to guarantee the system timing performance even under the worst-case scenarios. A task mapping indicates how tasks are distributed on a given NoC platform. A suitable task mapping can improve or guarantee the system timing performance by affecting task execution on IP (Intellectual Property) cores and message propagations on a NoC. It can be searched or constructed at design or run-time.

A mapping can be well optimised at design time with static mapping algorithms. It provides guaranteed system performance with foreseeable and constant information about the application and working environment. However, the system flexibility and adaptivity will be limited and the design exploration process will be prolonged if the design space is enlarged by a larger NoC, more complex application, or both.

A run-time mapping can manage task allocation dynamically. It can enhance the system flexibility, adaptivity and fault-tolerance. However, the current dynamic mapping algorithms are affected by low remapping success rates or

lack of a guarantee for the system timing performance after remapping. This becomes worse in hard real-time systems.

The research presented in this thesis was based on the hypothesis:

"A schedulable task allocation can be found dynamically and efficiently to meet the application's hard real-time timing requirements and reduce task migration cost in an NoC based Multi-Processor System-on-Chip".

This research investigated techniques for enhancing both static and dynamic mapping scenarios.

Chapter 4 focuses on End-to-End Response Time Analysis (E2ERTA), which is an exact evaluation method for indicating whether a given task allocation can satisfy system hard real-time timing constraints on a specified priority pre-emptive arbitration NoC. An inexact analysis method was adapted to alleviate the calculation complexity, which precludes its application in large design space exploration in both static and dynamic mapping problems.

Chapter 5 investigates the efficiency of E2ERTA from a practical point of view and introduces a parallel computation architecture and accelerated components for the inexact analysis method proposed in Chapter 4 to enable on-line hard real-time timing analysis. The implementation is undertaken in hardware describe language (VHDL) and evaluated on an FPGA based experiment platform.

Chapter 6 concentrates on dynamic mappers. A search-based algorithm (GA) was proposed to enhance the successful remapping rate. A parallel model with modification and extra-accelerated GA operators were also suggested to accelerate the search speed to enable consideration of global task reallocation at run-time on an NoC platform.

Chapter 7 combines the improvements from both evaluation method and dynamic mapper to verify the mapping search and evaluation performance in hard real-time NoC dynamic mapping problems in two steps. The first step

is to assess search speed and ability without considering task migration cost. The performance assessment is supported by an FPGA based experiment platform. The results indicate that a schedulable mapping solution can be found quickly (millisecond level). This makes feasible the optimising of task allocation at run-time for hard real-time NoCs. The second step extends the requirements from only considering successful remapping to take task migration costs into account. The results show that the timing performance and the migration costs can be optimised simultaneously.

8.1 Future Work

The hard real-time NoC dynamic mapping problem can be addressed by applying the techniques and implementation proposed in this thesis. However, there are still some issues relating to this area that can be investigated in a future study.

- Acceleration:
  - From the perspective of evaluation method, the proposed HW-E2ERTA, PRE and NLB can be placed in parallel, as shown in Figure 8.1. Two kinds of evaluation can be enabled concurrently. The one finishing first can interrupt the later one.

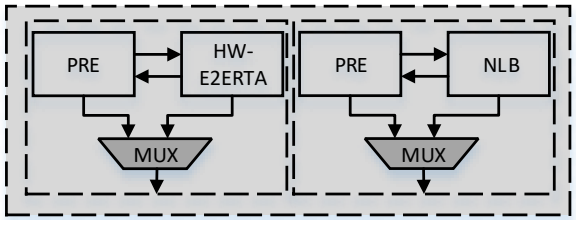

Figure 8.1: Evaluation Method Parallelism.

  - An island model parallel GA can be adapted to enable multi-population searches in parallel, in order to explore larger design

space within a fixed search time. In addition, a hierarchy or hybrid GA model is also a possible direction.

- The priority of tasks and flows in this research is fixed and known in advance. This will limit the pre-emptive among tasks of flows in fixed order. Run-time priority assignment could adjust the system performance for a specific event or system mode and may result in an easier mapping search. However, it will enlarge the search space. Therefore, a trade-off analysis may be required.

- Centralised dynamic mapping searches provide better search performance by gathering the system information. However, it may increase the local power consumption and lead to a hot core, which is not easy to resolve in chip design. In addition, if it is broken or has any faults, dynamic mapping cannot be achieved. Therefore, distributed methods could be an alternative direction.

- Timing performance is only one of evaluation criterias for NoCs. There are other aspects such as power consumption. In some situations, power is as important as timing, for example, when a hard real-time system changes to a low battery mode. In this case multi-objective search could be considered.

- The method used to reduce task migration cost in this research is minimising the mapping differences between original task allocation and the newly generated one. This will increase the search time. However, whether the time used to reduce the mapping differences is less than the time applied to moving tasks without considering mapping difference in reality may need to be considered. In addition, the migration methods or migration routing could also be taken into account.

Appendix.A

Appendix A illustrates the details of the benchmarks mainly used in Chapter 4 and summarizes their comparisons. Table A.1 and Table A.2 show how TB3 is generated by extending TB2 which can be found in [101]. Table A.3 and Table A.4 illustrate the Autonomous Vehicle benchmark with details of its tasks and traffic flows. Table A.5 summarizes the comparisons among these three benchmarks.

Table A.1: Extended Synthetic applications (TB3)

Source	Destination	Computation Time	Packet size (in bytes)	Period	Priority
p50	p28	0.611	1805	0.959	50
p51	p19	0.092	315	0.476	51
p52	p16	0.628	1747	0.761	52
p53	p46	0.210	1232	0.810	53
p54	p56	0.369	942	0.556	54
p55	p37	0.450	1587	0.629	55
p56	p58	0.445	2208	0.882	56
p57	p79	0.482	1243	0.817	57
p58	p35	0.192	1624	0.564	58
p59	p49	0.407	819	0.586	59
p60	p89	0.919	549	0.997	60
p61	p4	0.016	2231	0.273	61
p62	p39	0.571	2397	0.879	62
p63	p93	0.336	904	0.671	63
p64	p54	0.313	870	0.324	64
p65	p63	0.519	325	0.559	65
p66	p21	0.144	1918	0.668	66
p67	p65	0.240	1363	0.797	67
p68	p95	0.521	1898	0.715	68
p69	p51	0.272	1263	0.512	69
p70	p33	0.323	461	0.506	70
p71	p61	0.081	747	0.145	71
p72	p99	0.208	878	0.233	72
p73	p13	0.066	732	0.848	73
p74	p74	0.094	1665	0.456	74
p75	p18	0.094	126	0.142	75
p76	p44	0.218	103	0.582	76
p77	p24	0.077	1134	0.322	77
p78	p24	0.275	2084	0.315	78
p79	p46	0.351	2291	0.404	79

*TB3 follows TB2's structure but extends TB2's number of tasks from 50 to 100. The period is generated randomly. $computationtime = period * percentage$ ($randomly\ generated$). The size of each packet is a randomly selected number between the minimum and maximum number of packet size in TB2.

Table A.2: Extended Synthetic applications (TB3)

Source	Destination	Computation Time	Packet size (in bytes)	Period	Priority
p80	p32	0.279	1816	0.304	80
p81	p60	0.252	1333	0.372	81
p82	p83	0.037	797	0.280	82
p83	p9	0.227	1707	0.351	83
p84	p54	0.263	2046	0.730	84
p85	p71	0.487	277	0.924	85
p86	p16	0.301	68	0.646	86
p87	p24	0.532	1425	0.606	87
p88	p2	0.427	694	0.618	88
p89	p79	0.210	1189	0.220	89
p90	p73	0.255	180	0.328	90
p91	p61	0.072	2090	0.103	91
p92	p25	0.209	2275	0.440	92
p93	p69	0.016	1412	0.158	93
p94	p28	0.060	428	0.267	94
p95	p11	0.013	1859	0.036	95
p96	p66	0.009	1945	0.136	96
p97	p60	0.050	1718	0.268	97
p98	p8	0.119	1696	0.254	98
p99	p18	0.199	99	0.986	99

*TB3 follows TB2's structure but extends TB2's number of tasks from 50 to 100. The period is generated randomly. $computationtime = period * percentage$ ($randomly\ generated$). The size of each packet is a randomly selected number between the minimum and maximum number of packet size in TB2.

Table A.3: Autonomous Vehicle Application Tasks

Task	Task description	Computation time (s)	Period (s)	Utilization
TPMS	Tyre pressure monitoring system	0.005	0.5	1.00%
VIBS	Vibration sensor	0.005	0.1	5.00%
SPES	Speed sensor	0.005	0.1	5.00%
POSI	Position sensor interface	0.005	0.5	1.00%
USOS	Ultrasonic sensor	0.005	0.1	5.00%
FBU1	Frame buffer - Left camera, upper-left quadrant	0.01	0.4	2.50%
FBU2	Frame buffer - Left camera, upper-right quadrant	0.01	0.4	2.50%
FBU3	Frame buffer - Left camera, lower-left quadrant	0.01	0.4	2.50%
FBU4	Frame buffer - Left camera, lower-right quadrant	0.01	0.4	2.50%
FBU5	Frame buffer - Right camera, upper-left quadrant	0.01	0.4	2.50%
FBU6	Frame buffer - Right camera, upper-right quadrant	0.01	0.4	2.50%
FBU7	Frame buffer - Right camera, lower-left quadrant	0.01	0.4	2.50%
FBU8	Frame buffer - Right camera, lower-right quadrant	0.01	0.4	2.50%
STAC	Stability control	0.01	1	1.00%
TPRC	Tyre pressure control	0.001	0.01	10.00%
DIRC	Direction control	0.001	0.01	10.00%
OBDB	Obstacle database	0.15	0.5	30.00%
BFE1	Background estimation and feature extraction 1	0.02	0.04	50.00%
BFE2	Background estimation and feature extraction 2	0.02	0.04	50.00%
BFE3	Background estimation and feature extraction 3	0.02	0.04	50.00%
BFE4	Background estimation and feature extraction 4	0.02	0.04	50.00%
BFE5	Background estimation and feature extraction 5	0.02	0.04	50.00%
BFE6	Background estimation and feature extraction 6	0.02	0.04	50.00%
BFE7	Background estimation and feature extraction 7	0.01	0.04	25.00%
BFE8	Background estimation and feature extraction 8	0.01	0.04	25.00%
FDF1	Feature data fusion 1	0.01	0.4	2.50%
FDF2	Feature data fusion 2	0.01	0.4	2.50%
STPH	Stereo photogrammetry	0.03	0.04	75.00%
THRC	Throttle control	0.001	0.01	10.00%
VOD1	Visual odometry 1	0.02	0.04	50.00%
VOD2	Visual odometry 2	0.02	0.04	50.00%
OBMG	Obstacle database manager	0.02	1	2.00%
NAVC	Navigation control	0.01	0.5	2.00%
Average				19.15%

Table A.4: Autonomous Vehicle Application Traffic Flows between Tasks

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

Table A.5: Benchmark Summary

Benchmark	Task Period(s)	Utilisation
1	0.01∼1	19.15%
2	0.01∼1	39.2%
3	0.01∼1	41.3

Appendix.B

Appendix B provides the data distribution, with boxplots, to show that the average values used in Figure 4.4 to 4.8 are meaningful.

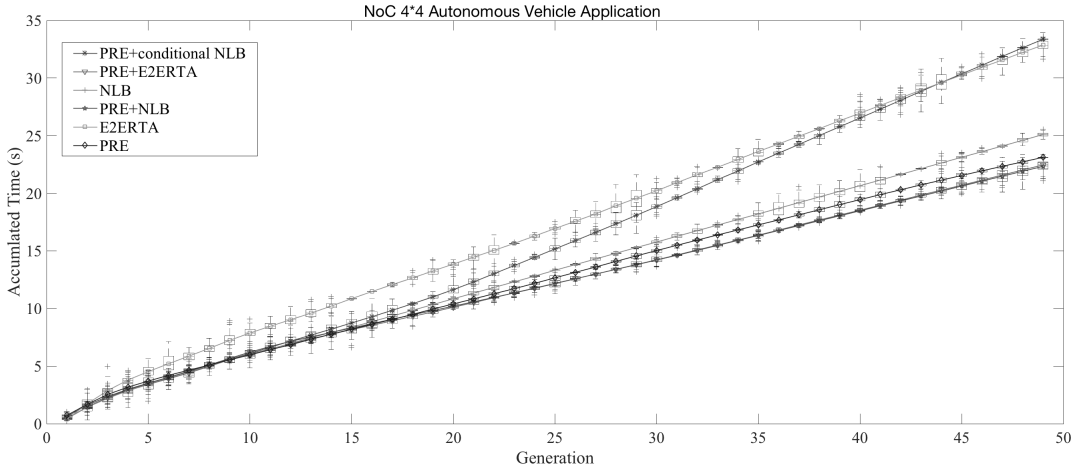

Figure A.1: Autonomous Vehicle application on 4*4 NoC Boxplot for Accumulated Time.

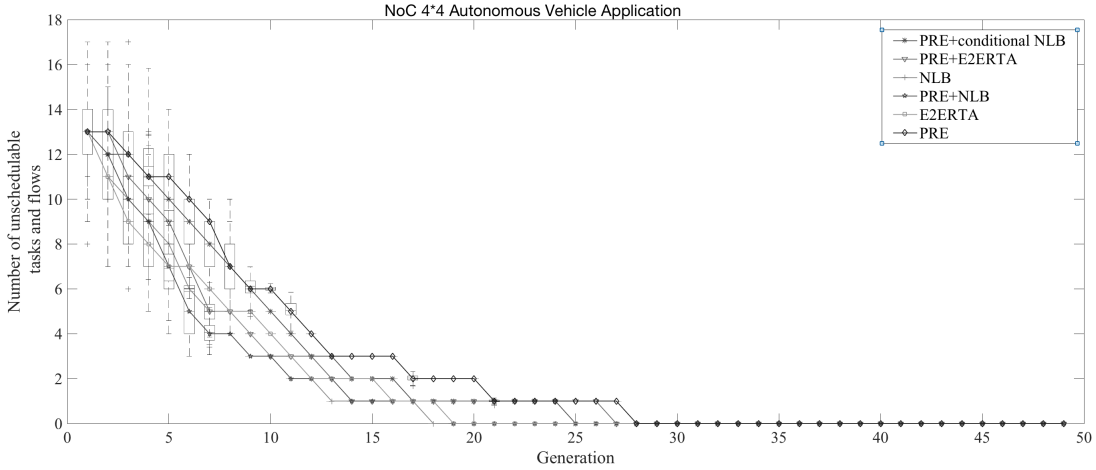

Figure A.2: Autonomous Vehicle application on 4*4 NoC Boxplot for Number of Unschedulable Tasks and Flows.

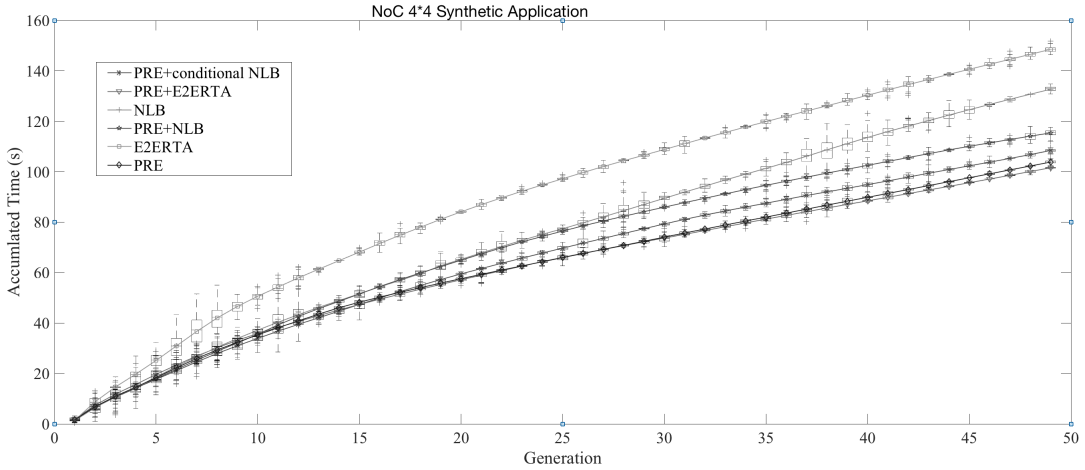

Figure A.3: Synthetic application on 4*4 NoC Boxplot for Accumulated Time.

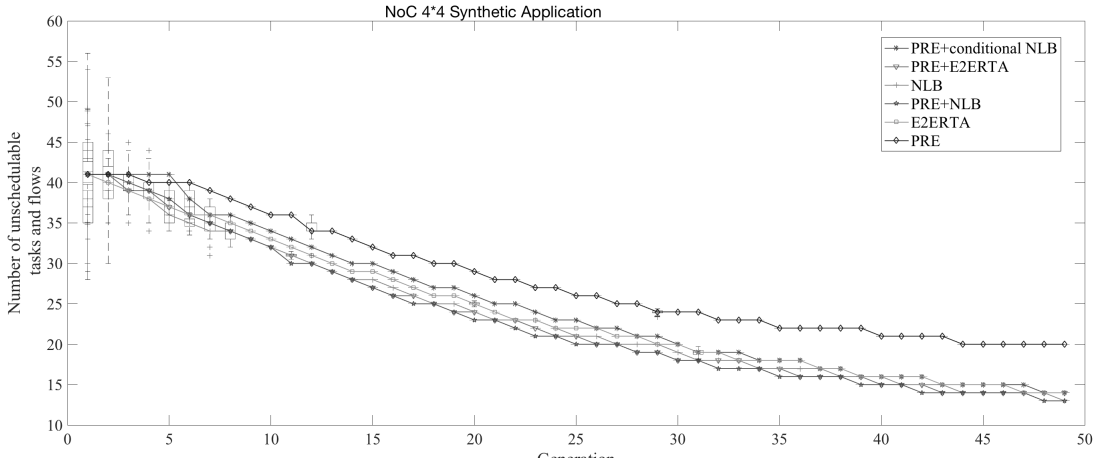

Figure A.4: Synthetic application on 4*4 NoC Boxplot for Number of Unschedulable Tasks and Flows.

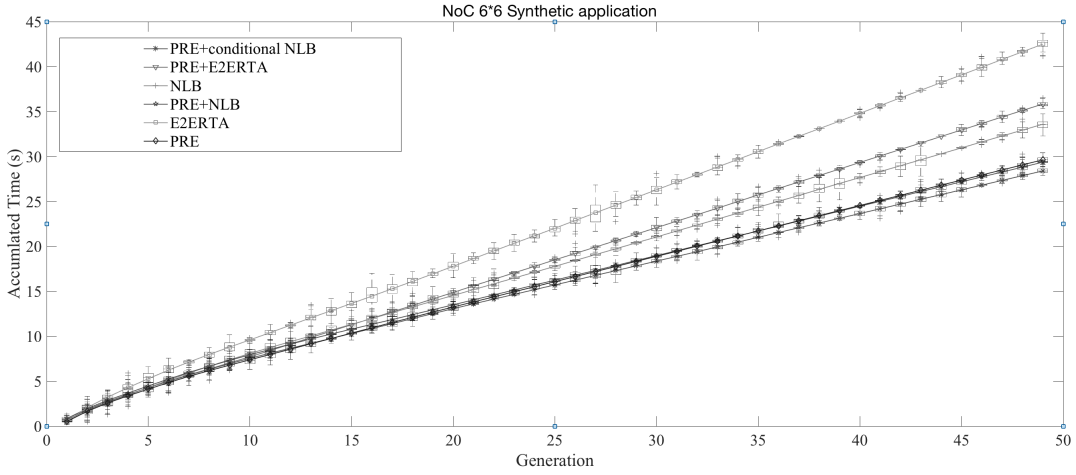

Figure A.5: Synthetic application on 6*6 NoC Boxplot for Accumulated Time.

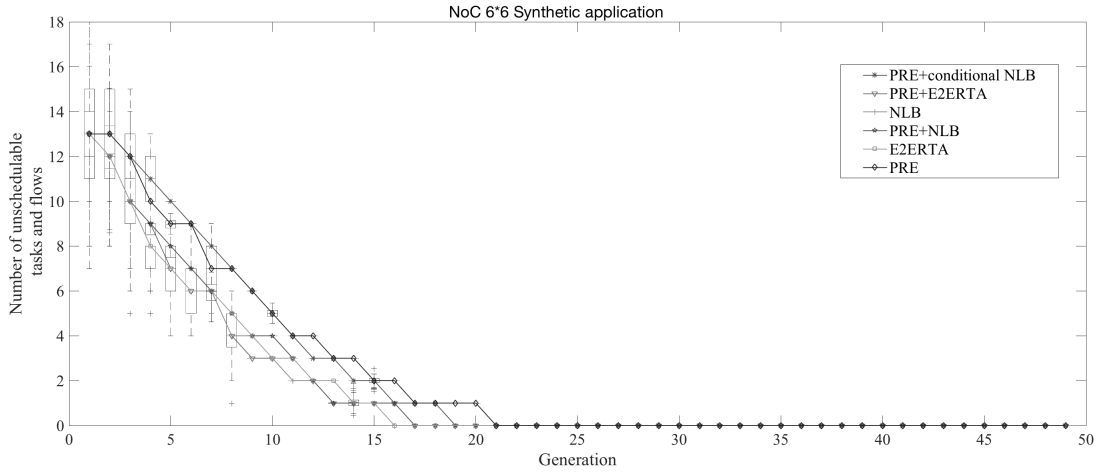

Figure A.6: Synthetic application on 6*6 NoC Boxplot for Number of Unschedulable Tasks and Flows.

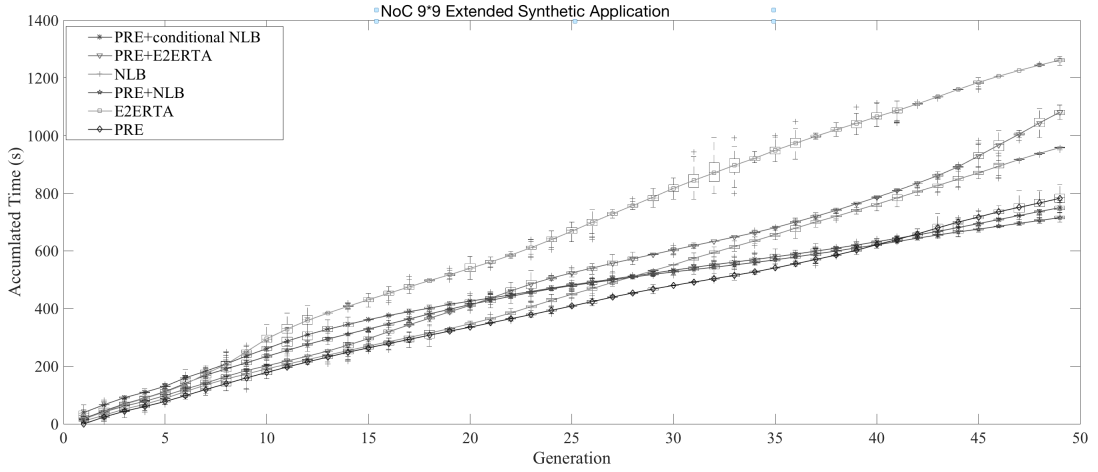

Figure A.7: Extended Synthetic application on 9*9 NoC Boxplot for Accumulated Time.

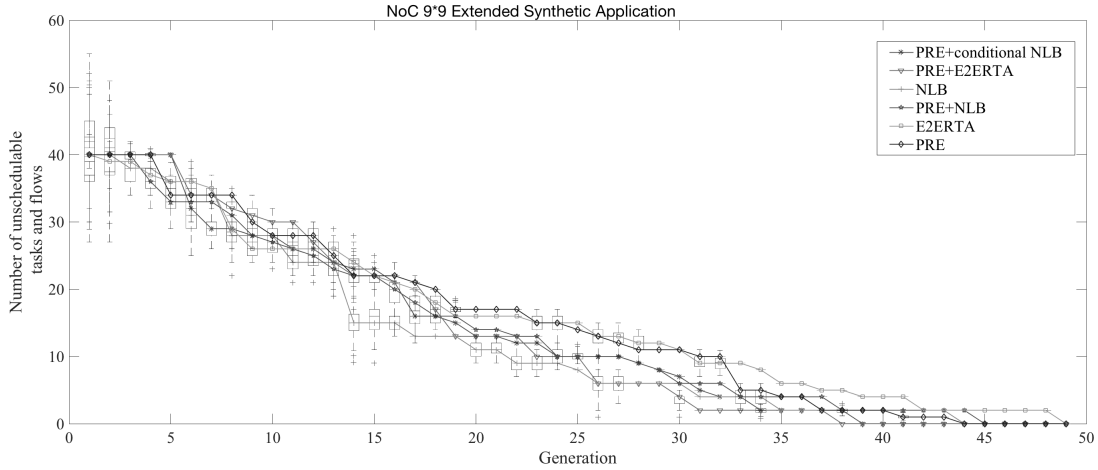

Figure A.8: Extended Synthetic application on 9*9 NoC Boxplot for Number of Unschedulable Tasks and Flows.

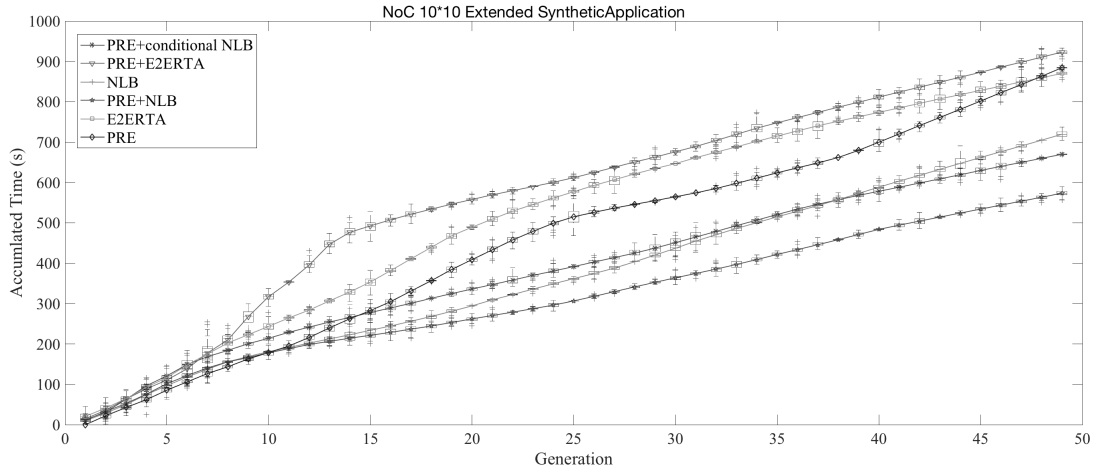

Figure A.9: Extended Synthetic application on 10*10 NoC Boxplot for Accumulated Time.

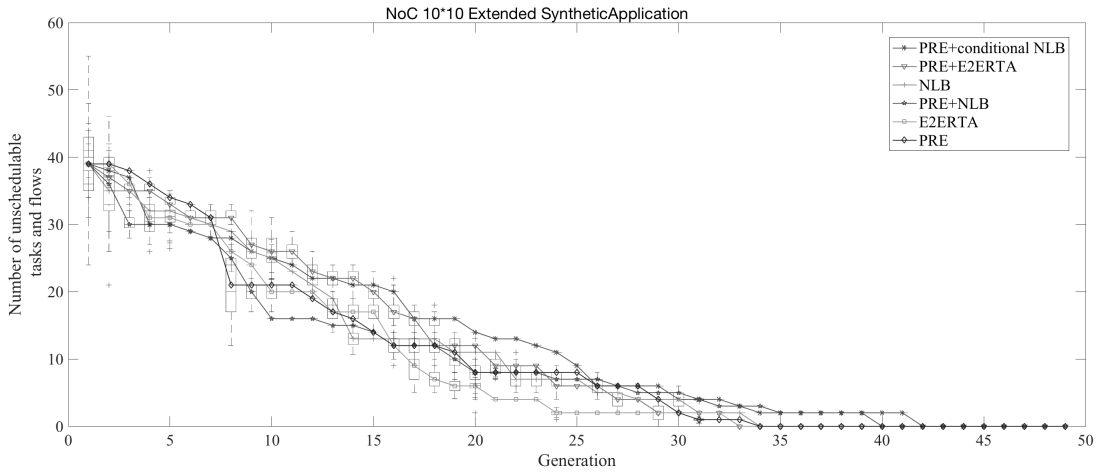

Figure A.10: Extended Synthetic application on 10*10 NoC Boxplot for Number of Unschedulable Tasks and Flows.

Bibliography

[1] Agarwal, A. and Shankar, R. (2009). Survey of Network on Chip (NoC) Architectures & Contributions. *Journal of Engineering, Computing and Architecture*, 3(1).

[2] Alba, E. and Troya, J. M. (1999). A survey of parallel distributed genetic algorithms. *Journal Complexity*, 4(4):31–52.

[3] Aporntewan, C. and Chongstitvatana, P. (2001). A hardware implementation of the compact genetic algorithm. *Proceedings of the Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*.

[4] Audsley, N., Burns, A., Richardson, M., Tindell, K., and Wellings, A. (1993). Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5).

[5] Bender, A. (1996). Milp based task mapping for heterogeneous multiprocessor systems. *Proceedings EURO-DAC European Design Automation Conference with EURO-VHDL and Exhibition*.

[6] Benyamina, A. E. H., Boulet, P., Aroui, A., Eltar, S., and Dellal, K. (2010). Mapping Real Time Applications on NoC Architecture with Hybrid Multi-objective Algorithm. *Intenational Conference on Metaheuristics and Nature Inspired Computing*.

[7] Bertozzi, D. and Benini, L. (2004). Feature - xpipes : a network-on-chip architecture for gigascale systems-on-chip. *IEEE Circuits and Systems Magazine*, 4(2):18–31.

[8] Bertozzi, D., Jalabert, A., Murali, S., Tamhankar, R., Stergiou, S., Benini, L., and Micheli, G. D. (2005). Noc synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 16(2):113–129.

[9] Bhardwaj, K. and Jena, R. K. (2009). Energy and bandwidth aware mapping of ips onto regular noc architectures using multi-objective genetic algorithms. *International Symposium on System-on-Chip*.

[10] Bini, E. and Baruah, S. (2007). Efficient computation of response time bounds under fixed-priority scheduling.

[11] Bjerregaard, T. and Mahadevan, S. (2006). A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38(1).

[12] Bjerregaard, T. and Sparso, J. (2005). A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip. *Design, Automation and Test in Europe*.

[13] Bolotin, E., Cidon, I., Ginosar, R., and Kolodny, A. (2004). Qnoc: Qos architecture and design process for network on chip. *Journal of Systems Architecture*, 50(2-3):105–128.

[14] Carvalho, E., Calazans, N., and Moraes, F. (2007). Heuristics for dynamic task mapping in noc-based heterogeneous mpsocs. *18th IEEE/IFIP International Workshop on Rapid System Prototyping (RSP)*.

[15] Carvalho, E. and Moraes, F. (2008). Congestion-aware task mapping in heterogeneous mpsocs. *International Symposium on System-on-Chip*.

[16] Carvalho, E. L. D. S., Calazans, N. L., and Moraes, F. G. (2010). Dynamic task mapping for mpsocs. *IEEE Design & Test of Computers*, 27(5):26–35.

[17] Chelcea, T. and Nowick, S. (2001). Robust interfaces for mixed-timing systems with application to latency-insensitive protocols. *Design Automation Conference*.

[18] Chen, T., Fu, W., Xie, B., and Wang, C. (2014). Packet triggered prediction based task migration for network-on-chip. *Microprocessors and Microsystems*, 38(4):316–324.

[19] Chou, C.-L. and Marculescu, R. (2007). Incremental run-time application mapping for homogeneous nocs with multiple voltage levels. *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis (CODES+ISSS)*.

[20] Chou, C.-L. and Marculescu, R. (2008a). Contention-aware application mapping for network-on-chip communication architectures. *IEEE International Conference on Computer Design*.

[21] Chou, C.-L. and Marculescu, R. (2008b). User-aware dynamic task allocation in networks-on-chip. *Design, Automation and Test in Europe*.

[22] Chou, C.-L. and Marculescu, R. (2010). Run-time task allocation considering user behavior in embedded multiprocessor networks-on-chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(1):78–91.

[23] Chou, C.-L., Ogras, U., and Marculescu, R. (2008). Energy-and performance-aware incremental mapping for networks on chip with multiple voltage levels. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1866–1879.

[24] Choudhary, N., Gaur, M. S., Laxmi, V., and Singh, V. (2010). Energy aware design methodologies for application specific noc. *NORCHIP*.

[25] Choudhary, N., Gaur, M. S., Laxmi, V., and Singh, V. (2011). Ga based congestion aware topology generation for application specific noc. *IEEE International Symposium on Electronic Design, Test and Application*.

[26] Colorni, A., Dorigo, M., and Maniezzo, V. (1991). Distributed optimization by ant colonies. *European Conference on Artificial Life*, pages 134–142.

[27] Commitee (2015). International technology roadmap for semiconductors. Online; accessed 27-09-2017 `https://www.dropbox.com/sh/8jaob1jtmkugw7d/AABnYL-nltLopTv6_nQ9xjeGa/2014%20ITRS%202.0%20FT%20White%20Papers?dl=0&preview=20150220_MoreMooreWP.pdf`.

[28] Dally, W. (1990). Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6):775–785.

[29] Dally, W. (1992). Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205.

[30] Darwin, C. R. (1859). On the origin of species. *John Murray*.

[31] Davis, R. I., Zabos, A., and Burns, A. (2008). Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Transactions on Computers*, 57(9):1261–1276.

[32] Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. *Parallel Problem Solving from Nature PPSN VI Lecture Notes in Computer Science*, pages 849–858.

[33] Derin, O., Kabakci, D., and Fiorin, L. (2011). Online task remapping strategies for fault-tolerant network-on-chip multiprocessors. *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*.

[34] Dielissen, J., Radulescu, A., Goossens, K., and Rijpkema, E. (2003). Concepts and implementation of the philips network-on-chip. *IP based SOC (IPSOC)*.

[35] Duato, J., Yalamanchili, S., and Ni, L. M. (1997). *Interconnection networks: an engineering approach*. IEEE computer Co.

[36] Dziurzanski, P., Singh, A. K., and Indrusiak, L. S. (2015). Hard real-time guarantee of automotive applications during mode changes. *In Proceedings of the International Conference on Real Time and Networks Systems (RTNS)*.

[37] Dziurzanski, P., Singh, A. K., and Indrusiak, L. S. (2016). Feedback-based admission control for hard real-time task allocation under dynamic workload on many-core systems. *In Proceedings of the International Conference on Architecture of Computing Systems (ARCS)*.

[38] Erika, C., Amory, A. d. M., and Lubaszewski, M. S. (2012). *Reliability, Availability and Serviceability of Networks-on-Chip*. Springer.

[39] Evangelia Kasapaki, Martin Schoeberl, R. B. S. C. M. K. G. and Sparso, J. (2016). Argo: A real-time network-on-chip architecture with an efficient gals implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

[40] Faraji, R. and Naji, H. R. (2014). An efficient crossover architecture for hardware parallel implementation of genetic algorithm. *Neurocomputing*, 128:316–327.

[41] Faruque, M. A. A., Krist, R., and Henkel, J. (2008). Adam: Run-time agent-based distributed application mapping for on-chip communication. *Proceedings of the 45th annual conference on Design automation (DAC)*.

[42] Felicijan, T. and Furber, S. (2004). An asynchronous on-chip network router with quality-of-service (qos) support. *IEEE International SOC Conference*.

[43] Fogel, L. J. (1966). *Artificial intelligence through simulated evolution Lawrence J. Fogel*. Wiley.

[44] Gaur, M. S., Laxmi, V., Zwolinski, M., Kumar, M., Gupta, N., and Ashish (2015). Network-on-chip: Current issues and challenges. *VLSI Design and Test (VDAT)*.

[45] Girao, G., Santini, T., and Wagner, F. R. (2013). Exploring resource mapping policies for dynamic clustering on noc-based mpsocs. *Design, Automation and Test in Europe Conference and Exhibition (DATE)*.

[46] Goossens, K., Dielissen, J., and Radulescu, A. (2005). æthereal network on chip:concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):414–421.

[47] Grefenstette, J. J. (1981). *Parallel adaptive algorithms for function optimization (preliminary report)*. Computer Science Dep., Vanderbilt Univ.

[48] Greiner, D. (2015). *Advances in evolutionary and deterministic methods for design, optimization and control in engineering and sciences*. Springer.

[49] Guerrier, P. and Greiner, A. (2008). A generic architecture for on-chip packet-switched interconnections. *Design, Automation, and Test in Europe*, pages 111–123.

[50] Heisswolf, J., Konig, R., Kupper, M., and Becker, J. (2013). Providing multiple hard latency and throughput guarantees for packet switching networks on chip. *Computers and Electrical Engineering*, 39.

[51] Hesham, S., Rettkowski, J., Goehringer, D., and Ghany, M. A. A. E. (2017). Survey on real-time networks-on-chip. *IEEE Transactions on Parallel and Distributed Systems*.

[52] Holland, J. H. (1973). Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88–105.

[53] Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press.

[54] Hong, S., Chantem, T., and Hu, X. S. (2015). Local-deadline assignment for distributed real-time systems. *IEEE Transactions on Computers IEEE Trans. Comput.*, 64(7):1983–1997.

[55] Hsiue, K. and Teague, B. (2013). Hardware implementation of genetic algorithms. *Complex Digital Systems*.

[56] Hu, J. and Marculescu, R. (2003). Exploiting the routing flexibility for energy/performance-aware mapping of regular noc architectures. *Design, Automation, and Test in Europe*, pages 141–155.

[57] Hu, J. and Marculescu, R. (2005). Energy-and performance-aware mapping for regular noc architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(4):551–562.

[58] Indrusiak, L. S. (2014). End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration. *Journal of Systems Architecture*, 60(7):553–561.

[59] Jakobovic, D. and Marko cupic, M. G. (2014). Asynchronous and implicitly parallel evolutionary computation models. *Soft Computing*.

[60] Jang, W. and Pan, D. Z. (2010). A3map: Architecture-aware analytic mapping for networks-on-chip. *15th Asia and South Pacific Design Automation Conference (ASP-DAC)*.

[61] Jena, R. and Gopal Ku, S. (2007). A multi-objective evolutionary algorithm based optimization model for network-on-chip synthesis. *Fourth International Conference on Information Technology (ITNG)*.

[62] Jerger, N. D. E., Peh, L.-S., and Lipasti, M. H. (2008). Circuit-switched coherence. *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*.

[63] Kaushik, S., Singh, A. K., Jigang, W., and Srikanthan, T. (2011). Runtime computation and communication aware mapping heuristic for noc-based heterogeneous mpsoc platforms. *Fourth International Symposium on Parallel Architectures, Algorithms and Programming*.

[64] Kavaldjiev, N. and Smit, G. (2003). A survey of efficient on-chip communications for soc. In *4th PROGRESS Symposium on Embedded Systems*, pages 129–140. STW Technology Foundation.

[65] Kiasari, A. E., Jantsch, A., Bekooij, M., Burns, A., and Lu, Z. (2012). Analytical approaches for performance evaluation of networks-on-chip. *Proceedings of the international conference on Compilers, architectures and synthesis for embedded systems (CASES)*.

[66] Kiasari, A. E., Jantsch, A., and Lu, Z. (2013a). Mathematical formalisms for performance evaluation of networks-on-chip. *ACM Computing Surveys*, 45(3):1–41.

[67] Kiasari, A. E., Lu, Z., and Jantsch, A. (2013b). An analytical latency model for networks-on-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(1):113–123.

[68] Koza, J. (1991). Genetically breeding populations of computer programs to solve problems in artificial intelligence. *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*.

[69] Krstic, M., Grass, E., Gurkaynak, F. K., and Vivet, P. (2007). Globally asynchronous, locally synchronous circuits: Overview and outlook. *IEEE Design & Test of Computers*, 24(5):430–441.

[70] Kurd, N., Chowdhury, M., Burton, E., Thomas, T. P., Mozak, C., Boswell, B., Mosalikanti, P., Neidengard, M., Deval, A., Khanna, A., and et al. (2015). Haswell: A family of ia 22 nm processors. *IEEE J. Solid-State Circuits IEEE Journal of Solid-State Circuits*, 50(1):49–58.

[71] Lee, E. and Messerschmitt, D. (1987). Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245.

[72] Lei, T. and Kumar, S. (2003). A two-step genetic algorithm for mapping task graphs to a network on chip architecture. *Digital System Design*.

[73] Lin, S.-Y., Huang, C.-H., Chao, C.-H., Huang, K.-H., and Wu, A.-Y. (2008). Traffic-balanced routing algorithm for irregular mesh-based on-chip networks. *IEEE Transactions on Computers*, 57(9):1156–1168.

[74] Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61.

[75] Liu, S., Jantsch, A., and Lu, Z. (2012). Parallel probing: Dynamic and constant time setup procedure in circuit-switching noc. *Design, Automation and Test in Europe Conference and Exhibition (DATE)*.

[76] Mandelli, M., Amory, A., Ost, L., and Moraes, F. G. (2011a). Multitask dynamic mapping onto noc-based mpsocs. *Proceedings of the 24th symposium on Integrated circuits and systems design (SBCCI)*.

[77] Mandelli, M., Ost, L., Carara, E., Guindani, G., Gouvea, T., Medeiros, G., and Moraes, F. G. (2011b). Energy-aware dynamic task mapping for noc-based mpsocs. *IEEE International Symposium of Circuits and Systems (ISCAS)*.

[78] Mehran, A., Khademzadeh, A., and Saeidi, S. (2008). Dsm: A heuristic dynamic spiral mapping algorithm for network on chip. *IEICE Electronics Express*, 5(13):464–471.

[79] Mello, A., Calazans, N., and Moraes, F. G. (2009). Qos in networks-on-chip beyond priority and circuit switching techniques. *VLSI-SoC: Advanced Topics on Systems on a Chip*.

[80] Mendis, H. R., Indrusiak, L. S., and Audsley, N. C. (2015). Bio-inspired distributed task remapping for multiple video stream decoding on homogeneous nocs. *13th IEEE Symposium on Embedded Systems For Real-time Multimedia (ESTIMedia)*.

[81] Meunier, Q., Peetrot, F., and Roch, J.-L. (2010). Hardware/software support for adaptive work-stealing in on-chip multiprocessor. *Journal of System Architecture*.

[82] Millberg, M., Nilsson, E., and Thid, R. (2004). Guaranteed bandwidth using looped containers in temporally disjoint net- works within the nostrum network on chip. *Design, Automation and Test in Europe Conference and Exhibition*.

[83] Murali, S., Benini, L., and Micheli, G. D. (2005). Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees. *Proceedings of the ASP-DAC Asia and South Pacific Design Automation Conference*.

[84] Murali, S. and Micheli, G. D. (2004). Bandwidth-constrained mapping of cores onto noc architectures. *Proceedings Design, Automation and Test in Europe Conference and Exhibition*.

[85] Ostler, C. and Chatha, K. S. (2007). An ilp formulation for system-level application mapping on network processor architectures. *Design, Automation & Test in Europe Conference & Exhibition*.

[86] Ozturk, O., Kandemir, M., and Son, S. W. (2007). An ilp based approach to reducing energy consumption in nocbased cmps. *Proceedings of the international symposium on Low power electronics and design (ISLPED)*.

[87] Parasuraman, D. (2013). *Handbook of particle swarm optimization: concepts, principles and applications*. Anmol Publications.

[88] Radu Andrei Stefan, A. M. and Goossens, K. (2014). daelite: A tdm noc supporting qos, multicast, and fast connection set-up. *IEEE Transactions on Computers*.

[89] Radulescu, A., Dielissen, J., Goossens, K., Rijpkema, E., and Wielage, P. (2005a). An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network configuration. *Proceedings Design, Automation and Test in Europe Conference and Exhibition*.

[90] Radulescu, A., Dielissen, J., Pestana, S., Gangwal, O., Rijpkema, E., Wielage, P., and Goossens, K. (2005b). An efficient on-chip ni offering guaranteed services, shared-memory abstraction, and flexible network configuration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(1):4–17.

[91] Saeidi, S., Khademzadeh, A., and Mehran, A. (2007). Smap: An intelligent mapping tool for network on chip. *International Symposium on Signals, Circuits and Systems*.

[92] Sahu, P. K. and Chattopadhyay, S. (2013). A survey on application mapping strategies for network-on-chip design. *Journal of Systems Architecture*, 59(1):60–76.

[93] Said, S. M. and Nakamura, M. (2014). Master-slave asynchronous evolutionary hybrid algorithm and its application in vanets routing optimization. *IIAI 3rd International Conference on Advanced Applied Informatics*.

[94] Salminen, E., Kulmala, A., and Hamalainen, T. D. (2009). Survey of Network-on-chip Proposals. In *OCP International Partnership*, number March, pages 1–13.

[95] Scott, S. D., Samal, A., and Seth, S. (1995). Hga: A hardware based genetic algorithm. *ACM/SIGDA 3rd Int. Symp. FPGA's*.

[96] Scott, S. D., Seth, S., and Samal, A. (1997). A hardware engine for genetic algorithms. *Technical Report UNL-CSE-97-001*.

[97] Selvi, V. and Umarani, R. (2010). Comparative analysis of ant colony and particle swarm optimization techniques. *International Journal of Computer Applications*, 5(4):1–6.

[98] Sen, P. and Pateriya, P. (2011). Implementation of generic algorithm using vhdl on fpga. *International Journal of Scientific & Engineering Research*.

[99] Sepulveda, M. J., Strum, M., and Chau, W. J. (2009). A multi-objective adaptive immune algorithm for noc mapping. *17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC)*.

[100] Sham, M. N., Sayuti, M., Indrusiak, L. S., and Garcia-Ortiz, A. (2013a). An optimisation algorithm for minimising energy dissipation in noc-based hard real-time embedded systems. *Proceedings of the 21st International conference on Real-Time Networks and Systems (RTNS)*.

[101] Sham, M. N., Sayuti, M., Indrusiak, L. S., and Garcia-Ortiz, A. (2013b). An optimisation algorithm for minimising energy dissipation in noc-based hard real-time embedded systems. *Proceedings of the 21st International conference on Real-Time Networks and Systems (RTNS)*.

[102] Shen, W.-T., Chao, C.-H., Lien, Y.-K., and Wu, A.-Y. (2007). A new binomial mapping and optimization algorithm for reduced-complexity mesh-based on-chip network. *First International Symposium on Networks-on-Chip (NOCS)*.

[103] Shi, Z. (2009). *Real-Time Communication Services for Network on Chip*. University of York.

[104] Shi, Z., Burns, A., and Indrusiak, L. S. (2010). Schedulability analysis for real time on-chip communication with wormhole switching. *Innovations in Embedded and Real-Time Systems Engineering for Communication*, pages 198–218.

[105] Siguenza Tortosa, D. A. (2005). *Proteo: the development of a practical network-on-chip*. Tampereen Teknillinen Yliopisto.

[106] Singh, A. K., Dziurzanski, P., and Indrusiak, L. S. (2015). Value and energy optimizing dynamic resource allocation in many-core hpc systems. *IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*.

[107] Singh, A. K., Dziurzanski, P., and Indrusiak, L. S. (2017). A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for many-core systems. *ACM Computing Surveys (CSUR)*.

[108] Singh, A. K., Jigang, W., Prakash, A., and Srikanthan, T. (2009). Mapping algorithms for noc-based heterogeneous mpsoc platforms. *12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*.

[109] Singh, A. K., Srikanthan, T., Kumar, A., and Jigang, W. (2010). Communication-aware heuristics for run-time task mapping on noc-based mpsoc platforms. *Journal of Systems Architecture*, 56(7):242–255.

[110] Song, H., Kwon, B., and Yoon, H. (1999). Throttle and preempt: A flow control policy for real-time traffic in wormhole networks. *Proceedings of the International Conference on Parallel Processing*, 45(8):633–649.

[111] Srinivasan, K., Chatha, K., and Konjevod, G. (2004). Linear programming based techniques for synthesis of network-on-chip architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

[112] Sudev, B., Indrusiak, L. S., and Harbin, J. (2015). Network-on-chip packet prioritisation based on instantaneous slack awareness. *Industrial Informatics (INDIN), IEEE 13th International Conference*.

[113] Tang, W. K., Man, K., Kwong, S., and He, Q. (1996). Genetic algorithms and their applications. In *Signal Processing Magazine, IEEE*, volume 13, pages 22 – 37.

[114] Vanneschi, L. and Verel, S. (2007). Fitness landscapes and problem hardness in evolutionary computation. *In Genetic And Evolutionary Computation Conference*.

[115] Vent, W. (1975). Rechenberg, ingo, evolutions strategies - optimierung technischer systeme nach prinzipien der biologischen evolution. 170 s. mit 36 abb. frommann-holzboog-verlag. stuttgart 1973. broschiert. *Feddes Repertorium*, 86(5):337–337.

[116] Wang, C., Yu, L., Liu, L., and Chen, T. (2012). Packet triggered prediction based task migration for network-on-chip. *20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*.

[117] Wang, J., Li, Y., Chai, S., and Peng, Q. (2011). Bandwidth-aware application mapping for noc-based mpsocs. *Journal of Computational Information Systems*.

[118] Wang, L. and Ling, X. (2010). Energy- and latency-aware noc mapping based on chaos discrete particle swarm optimization. *International Conference on Communications and Mobile Computing*.

[119] Wang, Y. (2015). *Circuit Clustering for Cluster-based FPGAs Using Novel Multiobjective Genetic Algorithms*. University of York.

[120] Wenbiao, Z., Yan, Z., and Zhigang, M. (2007). A link-load balanced low energy mapping and routing for noc. *International Conference on Embedded Software and Systems*, pages 59–66.

[121] Werner, S., Navaridas, J., and LujÃ¡n, M. (2016). A survey on design approaches to circumvent permanent faults in networks-on-chip. *CSUR ACM Comput. Surv. ACM Computing Surveys*, 48(4):1–36.

[122] Wiklund, D. and Liu, D. (2003). Socbus: Switched network on chip for hard real time embedded systems. *Parallel and Distributed Processing Symposium, Proceedings. International*.

[123] Xie, B., Chen, T., Hu, W., Tang, X., and Wang, D. (2013). An energy-aware online task mapping algorithm in NoC-based system. *The Journal of Supercomputing*.

[124] Xue, B., Zhang, M., Browne, W. N., and Yao, X. (2016). A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation*.

[125] Yang, B., Xu, T. C., Santti, T., and Plosila, J. (2010). Tree-model based mapping for energy-efficient and low-latency network-on-chip. *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*.

[126] Yuan, Y., Xu, H., and Wang, B. (2014). An improved nsga-iii procedure for evolutionary many-objective optimization. *Proceedings of the conference on Genetic and evolutionary computation (GECCO)*.

[127] Zeferino, C. and Susin, A. (2003). Socin: a parametric and scalable network-on-chip. *16th Symposium on Integrated Circuits and Systems Design*.

[128] Zhou, W., Zhang, Y., and Mao, Z. (2006). An application specific noc mapping for optimized delay. *International Conference on Design and Test of Integrated Systems in Nanoscale Technology*.