Mapping of Real-time Applications on Network-on-Chip based MPSOCS

Paris Mesidis

Submitted for the degree of Master of Science
(By Research)

The University of York,
Department of Computer Science

December 2011

Abstract

Mapping of real time applications on real time NoC based MPSoCs

With the recent developments in semiconductor technology it becomes possible to integrate many different processing elements on single chip, this solution is known as a System-on-Chip (SoC). In such systems, the communication between the different components is also an important aspect. On-chip busses or point-to-point communications have been successfully used. However, as the number of elements increases, on-chip busses are not able to scale and quickly become a communication bottleneck. Packet-switched networks on chip (NoC) have been proposed as a solution for on-chip communication of SoC platforms overcoming many of the limitations of on-chip buses. An important design aspect in such a system is the relative placement of tasks on the processing elements of the platform so that some metrics of interest are optimised commonly referred to as the mapping problem. NoC platforms, because of the benefits that they introduce, will eventually be used in commercial embedded and real-time systems. Despite its significance to embedded systems industry and research communities, little research has been done on providing guarantees for hard real-time applications composed of multiple communicating components running over NoC platforms, in such systems both the computation and the communication between the components must complete within certain deadlines for the system to behave correctly. Application mapping has a direct impact on the interference patterns emerging on the platform where separate tasks and the communications between them interfere with each other when contending over shared resources. Even though a significant amount of research has been carried out on the mapping problem usually the aim is to optimise different metrics and reduce cost in terms of energy consumption. This work tries to solve the mapping problem from a real-time systems perspective, in such systems the overall correctness does not only depend on producing the correct output but also on the time required by the system to produce it, this requirement is most commonly expressed with the concept of a deadline which must always bound the response time of a computational task. Schedulability analysis refers to a set of analytical methods that are able to prove a set of tasks can meet their deadlines when sharing resources under a particular scheduling scheme. This work takes advantage of recent advancements on schedulability analysis that can guarantee the timeliness of tasks, as well as their communications, in distributed real-time systems which specifically run on network on chip platforms using wormhole routing, this analysis is used as a ranking function in a genetic algorithm that is able to evolve task mappings which allow all tasks and communication flows to meet their deadlines in all possible scenarios.

Table of Contents

List of Figures

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text.

Paris Mesidis

Chapter 1

Introduction

1.1 Introduction to NoC Systems and the Mapping Problem

Multiprocessor Systems-on-Chip (MPSoC) are SoC made of multiple processing elements (PEs). MPSoC is a relatively new technique and a new trend in VLSI design that has been made possible due to the advancements in IC technology. The primary advantages of SoC devices are lower costs, greatly decreased size, reduced power consumption and the ability to meet the increasing performance requirements of complex concurrent applications (real-time video, communications, control etc.). Such applications are inherently parallel and should take advantage of any available parallelism. As the computational demands of such applications increase they will no longer be supported as efficiently by single general purpose processors. Since MPSoC are SoC with multiple processing elements a communication infrastructure has to interconnect the PEs. Traditional on chip communication schemes such as buses could be used for MPSoC that consist of a few elements but as the number of PEs that can be integrated on such a system increases, due to advances in silicon technology, these communication schemes will not be able to scale well and issues like the power consumption of the system will impose limitations. So there is a need of a scalable communication infrastructure and a promising solution is found in the use of networks on chip (NoC). Network(s)-on-Chip (NoC) have emerged as a design that incorporates on-chip communication architectures that are scalable and provide a more efficient structure and better modularity than its predecessors [1, 2].

MPSoC may employ NoCs to integrate many different components such as multiple programmable processor cores of various types, specialized memories, and other intellectual property blocks (IPs) in a scalable way, giving rise to heterogeneous systems, alternatively the integrated components may be all similar resulting in homogeneous systems. Heterogeneous systems achieve better performance and lower power consumption because of the application specific components; this comes at the cost of reduced flexibility and reusability. Homogeneous architectures are more flexible and also more open to different techniques such as run time reconfiguration, task migration and fault tolerance because of their functional redundancy.

An important aspect of MPSoC design is the optimisation of the system so that the platform is fully utilized the advantages of MPSoCs. Given specific applications that can be subdivided into a set of tasks, that can be concurrent and communicating with each other, it is necessary to control task operation and system resources usage in order to better utilise the system. The mapping of the application tasks onto the PEs of the system consists of finding a placement for a set of tasks so that some specific requirements are met. The performance of a NoC system running such a highly parallel application is also communication-dependent and depends on task mapping in many ways such as overall execution time, energy consumption, packet latency, communication channel load, bandwidth reservation, and congestion levels; The mapping problem for NoC is to decide how to place the selected set of tasks onto the PEs of the network such that the metrics of interest are optimized.

In NoC architectures each processing element is connected to a router which is the actual interface to the rest of the network. Each router's work is to implement a routing policy by accepting and forwarding each incoming packet, the architecture of the router depends on the switching policy which determines the manner that the packets are forwarded through the network so that the traffic flows can share the communication resources.

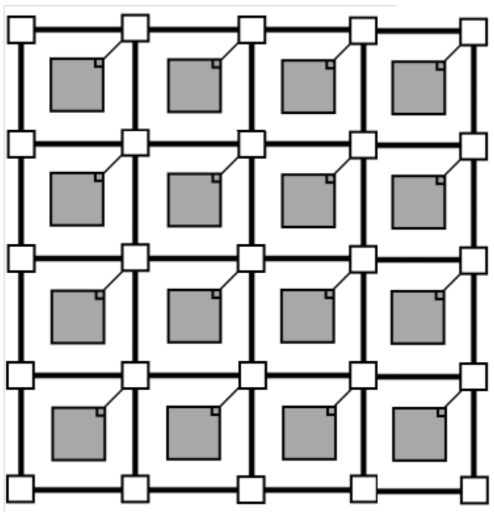


Figure 1.1 A 4x4 NoC

Applications running on MPSoC typically execute a varying number of tasks simultaneously, however at some point the number of tasks may exceed the available resources, requiring task mapping to be executed at runtime; more specifically the need to execute a new task may occur at run-time and with the current configuration the resources may be inadequate (paths connecting to master PE overloaded) so a remapping of some or all of the tasks may be necessary; in a simpler case a new task will have to be mapped somewhere on the available resources so that it's placement is optimal according to some criteria. Efficient solutions have been proposed for static mapping, static mapping defines a fixed placement and scheduling before the application is run where the application and its task set are known at design time. For this case approaches are used that can find better solutions since the time spent for finding such a solution at design time is not so important but obviously it is not appropriate for dynamic workloads.

The performance metrics mentioned above can usually be described in relation to the relative task placement so the task mapping problem, where the goal is to minimise some of these metrics, becomes similar to some well known NP-hard optimisation problems such as the quadratic assignment problem [3], [106] which is an especially hard instance of this class of problems [107] meaning that an algorithm that can solve this problem in polynomial time is unlikely to exist. The relation between different performance metrics is not always straight forward; a solution that optimises one may have adverse effects on another.

When formulating the mapping problem some properties of the system and underlying platform architecture must be known. These properties are also part of the assumptions in the problem definition, and hence they would affect the problem's complexity e.g. the ability for the PE's to execute more than one task. Additional properties such as the routing and switching policies of the network infrastructure would also play an important role in the way the problem is defined, and they could enable some analytical models that would help compute some of the metrics of interest; for example knowing the bandwidth capacity of a system's communication links and that the specific routing algorithm is deterministic could give some deterministic bounds for communication delay and throughput, however more information would be required because this model does not take into consideration the delays due to contention of the shared network resources, this in turn could be modelled only by knowing the arbitration strategy of the system. A different approach would be to use some algorithm or policy based on intuition or some partial formulation (small subset) of the problem and use simulation to verify it, the feasibility of this approach would depend on the quality of the solution required and the determinism of the output.

The mapping procedure in any such system, affects many different parameters that may be conflicting, so for a system with real-time requirements (hard, soft) a mapping that would avoid task's and communication's deadline misses would be ideal, typically however such applications are found in the embedded systems domain where power consumption is an important issue as well, so a solution that satisfies both requirements becomes harder to find. Additionally in such a system, especially if the mapping takes place dynamically, the running time of the mapping algorithm itself could have a significant impact on the overall temporal behaviour of the system.

Next another important aspect of the problem solution is the way each solution proposed by a mapping algorithm is actually evaluated, this is typically done using an analytical model and in that case an accurate model is important, for example many mapping algorithms seem to use the average packet hop count (moves to adjacent routers) as a function that determines the power dissipated for communication and also the communication time cost, however packets may usually be queued in a congested system and have to reside somewhere for some time (which with smaller transistor sizes is power consuming as well) so hop count is not necessarily always a good measure. Alternatively simulation can be used to evaluate a solutions quality at the cost of speed. The inputs to the mapping problem, which are an application and a NoC platform have to be abstracted away because they are typically very complex objects however these abstractions must be at a level that all necessary information is included while the overall input is concise. For example the representation of an application through an application task graph is a sufficient way as it conveys all the important information for an application that the mapping algorithm needs to know such as communication/control dependencies and communication volume, any further constraints that need to be considered such as task deadlines, channel bandwidth etc. can be added as additional variables to the mapping problem. Mapping as explained above is an important aspect of the design as it may drastically influence the system performance. Many approaches and algorithms have been proposed in literature for the mapping problem based on different assumptions and trying to optimise different aspects of a system (e.g. energy-aware, congestion-aware), the quality of such algorithms would be measured in relation to their execution times and overall optimisation of the system also how well they can perform in various scenarios and how well they meet certain constraints of the system (real-time, bandwidth).

However testing such algorithms is not always easy because testing in the context of real applications is not flexible or otherwise would be very time consuming. Synthetic models can be used that model traffic and computation loads according to different classes of applications with worst case bounds or alternatively simulation based approaches with some well known applications since specific benchmarks have not yet been defined for those aspects of NoC design. The aim of this work is to review the existing literature relevant to the mapping problem and then examine this problem from a real-time system perspective where the timing aspects of a system are the most important.

1.2 Problem Formulation

The mapping problem can be generally described as the problem of finding a topological placement of the application tasks onto the PEs of the system so that the metrics of interest are optimised. For an application that is partitioned into *n* tasks the problem of mapping those tasks on a network on chip with *n* processing elements would have *n*! possible solutions, the solution space factorially increases with the problem size so it is infeasible to exhaustively search it. The topological mapping of the application onto the NoC platform would obviously have an impact on the energy spent on the communication between the tasks and the communications latency as they are a function of the distance between the tasks. In some application domains, the communication volume between tasks can be very large and may be using the entire network link bandwidth; in this case a mapping that maps the tasks in a way so that the traffic on the network links does not exceed the bandwidth capacity of the network may be necessary. According to what metric the mapping problem is trying to optimise, the problem can be formulated differently, however most commonly the metrics and variables that formulate the problem are functions of distance and mapping becomes an instance of the NP-hard *quadratic assignment problem* [3]. This is the case when using the energy consumed on communication as a cost metric, as it is proportional to the distances between the communicating cores, also on such systems (NoCs) the energy spent on communication is a significant amount making it a popular metric in the relevant literature. The energy-aware formulation of the problem is described below to demonstrate a form that the problem may generally take.

The mapping problem in this case can be described as: For a set of communicating tasks where the communications and their volumes between them are known (described by a directed weighted graph *G* (T, *C*) and a set of cores with their positions on the network *N* (P, *L*) find a mapping function that maps tasks to cores *f* : T → P so that:

$$\text{Min}\,\{\sum_{\forall ci,j} (c_{\text{i}}, c_{\text{j}}) \, * \, \frac{E}{bit} \, * \, (f\ (Ti), f\ (Tj))\} \quad (1)$$

Explaining the terms in the equation above:

- $(c_{\text{i}}, c_{\text{j}})$ A pair of communicating tasks
- $\frac{E}{bit}$ The energy that the system spends to communicate a bit per unit distance.
- $(f\ (Ti), f\ (Tj))$ The distance between the communicating tasks as a result of the mapping function *f*.

It can be seen that the goal is to minimise the sum of products between a cost function and weights. In this case the cost function is the energy that is consumed for a bit to travel unit distance (between adjacent routers) and the weights are the communication volume and the distance between two communicating tasks. Additional constraints may be added to the problem because of features of the platform architecture or the application. For example the problem above could have some additional constraints as it does in [4]:

$$\forall T_i \neq T_j, f(T_i) \neq f(T_j) \quad (2)$$

$$Bw\ (link\ ) \geq \sum_{\forall Flow\ i,j} Bw\ (Flow\ i,j) * f\ (link, R\ i,j) \quad (3)$$

Where $R_{i,j}$ is the set of links that make the route between i and j.

$$f(link, R_{i,j}) = \begin{cases} 1\ if\ link \in R_{i,j} \\ 0 \quad otherwise \end{cases} \quad (4)$$

The first constraint means that each processing core can only have one task running on it, which is a platform specific feature. The second constraint demands that the aggregated communication traffic on any link does not exceed the available bandwidth so that all the communication flows can be serviced. Any additional constraints may be added to the problem. In the case described above the goal of the problem is to minimise the quantity in (1) which depends on the distance the traffic has to traverse. This distance in turn depends on the path the packet has to traverse through the network which is determined by the system's routing algorithm. Because of this close relation, many authors have treated the topological mapping and path selection/ routing as one problem [4], [5]. In many instances it is suitable for NoCs to use deterministic minimal routing algorithms, because it becomes easy to calculate the path and distance of a specific traffic flow, and it also becomes possible to calculate the quantity in (1).

Whether there exists an accurate analytical model for this metric depends on how these features are implemented in this system. Finding good analytical models for different metrics in such systems is regarded as a difficult problem as well [6].

1.3 Goals and Objectives

The main objective and scope of this work is to study the application mapping problem specifically from a distributed real-time systems perspective. In such systems the communications between real time tasks must complete within bounded time as well to ensure the correctness of the system, this is necessary as these communications contain data that the receiving tasks will need to operate on within bounded time. As a result this work will study the effects of task placement on both the timeliness of communication and computation in such systems. In order to do so it will be necessary to study solutions to this problem already proposed in a broader set of domains apart from real-time systems and also see what platforms and tools can be applied in this specific instance of the problem. Next utilising this knowledge we will be in a position to propose a solution to the problem based on evolutionary algorithms and finally evaluate it by producing experimental results and conclusions using an appropriate framework.

Chapter 2

Network on Chip Platforms

This chapter aims to provide a brief description of Network On Chip platforms by explaining some of the most important architectural characteristics and by giving an overview of the typical communication protocol stack. Next follows a review of current NoC platforms with regard to predictability, a concept central to real-time system analysis, and finally a review of methods which can be used to model such a system.

2.1 System Architectures and Modelling

The advancement in microchip technologies has enabled many components to be integrated together and form large scale systems-on-chip (SoC). Typically the on-chip communication between the components of such systems has been bus-based or a mixture of buses and point-to point links. Communication on chips with few components ought to be fast and reliable while the computation aspect is usually is the performance limiting factor. With the technology scaling down and by adding more components onto a single chip this changes; while resources for computation (PEs, memory) mainly benefit from scaling down the communication infrastructure does not. In a highly interconnected multi core system the energy required for communication does not scale down well, the power consumed to drive the wires increases (wire resistance increases) and physical parameters of the wires become unpredictable because it becomes harder to produce uniform structures, thus in future deep submicron (DSM) designs, the interconnect medium will definitely affect performance [14]. In addition noise from crosstalk, delay variations and synchronisation failures between large numbers of components may cause transmission failures. As a result estimating delays accurately will become increasingly harder, and the transmission will be power consuming and inherently unreliable. For maximum flexibility and scalability, it is generally accepted that a move towards a shared, segmented global communication structure is needed [15]. This motivation led into a new approach which is a data-routing network that consists of communication links and routing nodes built on the chip, hence the name network on chip. Similar to macro network communication, the network on chip approach tries to provide scalability, performance and reliability as well as modularity. At the same time however networks on chip must exhibit less non determinism while having tight resource constraints (e.g. energy, area).

2.2 Network on Chip Features

NoCs are packet switched multi hop networks. The processing elements access the network using point to point connections to the interface, and their packets are forwarded to their destinations by hoping through a number of routers. One of the basic characteristics of NoC is the topology which determines the layout and connectivity of the on chip components. Most NoCs implement regular network topologies that can be laid out on a 2D chip surface. These topologies are known as grid-based topologies. Torus, tree and irregular structures have also been proposed that may exploit different characteristics of the applications traffic patterns such as locality of traffic [36]; however mesh based topologies are the most compatible in terms of simple routing algorithms, re-usability and scalability hence the most commonly used.

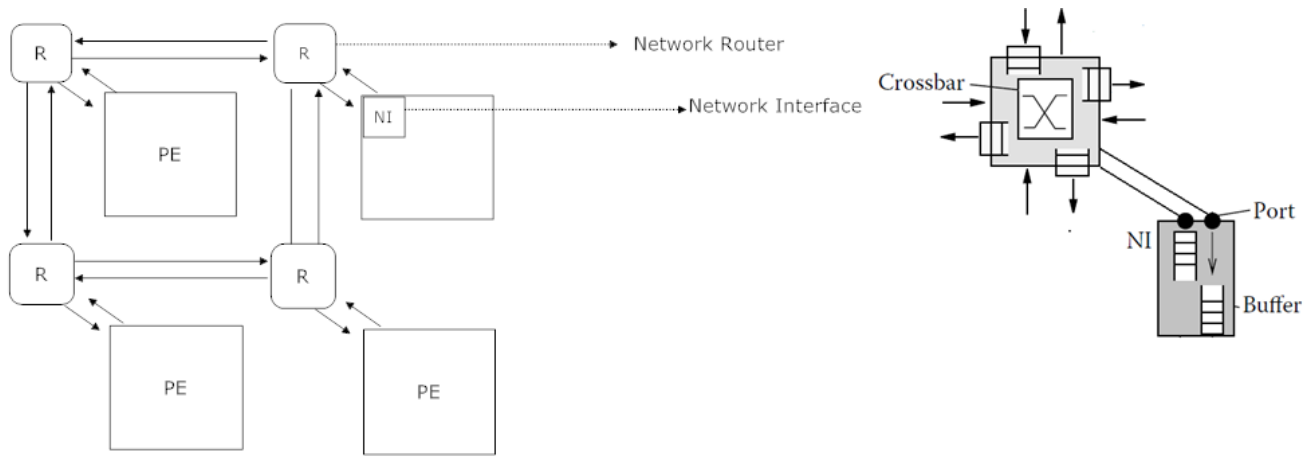


Figure 2.1 A 2D mesh based NoC.

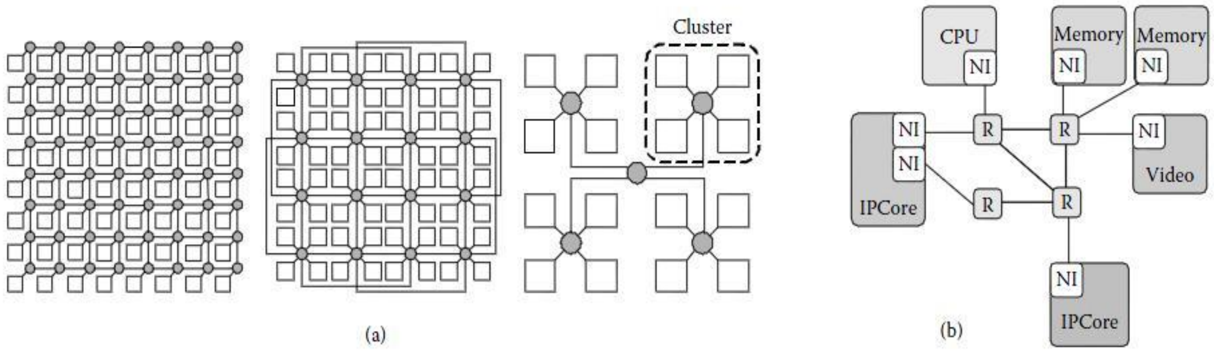


Figure 2.2 Regular (a) and irregular (b) NoC topologies.

Networks on Chip can be designed in different ways, based on the network architecture and communication protocol used. Networks on chip can also be designed and described by using a layered model as it is done in macro networks, but in a much simpler fashion. The NoC design phase would involve designing a system specific architecture and a communication protocol/infrastructure that will be compliant with this architecture; hence according to these design decisions a protocol stack can be defined as in [1]. In macro networks the abstractions offered by the layered protocol allow communication between different systems. Networks on chip have the advantage that the system is composed from static elements so the communication protocol becomes simpler based only on the attributes of the specific system. Overall the network characteristics such as the communication mechanism, switching mode, and routing algorithm depend on the network topology and together they define the services provided by the NoC.

2.3 Network Layers

The lowest level in a typical communication stack is the physical layer where specific signalling techniques are implemented to provide the most fundamental service of lossless data transmission through the links; according to the signalling technique (encoding, synchronization) and the physical properties of the communication links a trade off between energy consumption and transmission errors is achieved; based on the specific signalling technique the energy spend on transmitting data can be calculated. At this level, the basic unit of data is a *phit* (physical unit), which is the minimum datagram of the system and the building block of packets and streams.

The layer above the physical layer would be the data link layer. The main role of this layer is to improve the reliability of the physical layer so that the system can operate at the required level. This can be done at the expense of energy and system complexity using error detection and correction protocols. The data link layer would also have to implement a policy that resolves contention between different traffic flows for the network resources (arbitration policy) so that no errors occur from simultaneously accessing the network. The layer above the data link layer, the network layer is responsible for end to end delivery of messages, in such a complex network which is composed of many links this layer has to address both the way the data will be transported through the network or in other words how the connection and transmission occurs between successive links and how routes are chosen between the source and destination of the data sent. The tasks mentioned above are those of switching and routing of data. The network interface (NI) decouples the processing core's functionality from the network and provides a well defined interface to the network layer.

The two main switching techniques are circuit and packet switching. In the case of circuit switching a path from the source to the destination of data is reserved prior to the transmission for the duration of the data transmission and is accessible only by this set of data; in this case there is no need for the data to be packetized. Circuit switching, however, is very inefficient in terms of link capacity utilisation and set-up latency unless used for long infrequent messages.

Packet switching overcomes the limitations of circuit switching by using packetized traffic. The data to be transmitted is divided into packets that also contain routing information. Typically the whole packets are buffered at each intermediate node before they are forwarded to the next (*store-and forward*); packet switching offers much higher utilization of the network resources and higher flexibility (production rates, arbitration schemes, out-of-order delivery) with respect to circuit switching, at the price of an increase in non-determinism and silicon area (switches, buffers) as each packet may experience queuing delays (residing on buffers). The disadvantage in the case of packet switching is the buffer space requirements, specifically flow control techniques that govern the way the packets use the buffering resources deal with this limitation. In order to overcome this limitation packet switching is further improved by using different flow control techniques as in cut-through switching. With this technique each packet can start leaving the switch on which it resides before the entire packet is present (received by the switch) hence reducing the buffer size requirements. In this way, however, the total resources a packet will occupy increase as it may occupy many switches at once. In the case where a packet is blocked, waiting on resources, it will occupy many resources itself which may cause other packets to block so the overall blocking caused by a packet's transmission increases hence the overall variance in traffic response times increases [1].

A further improvement on packet switching is achieved by wormhole switching; in this scheme each packet is divided into flow control *units* (*flits*) which are fixed size segments. Each packet has a header flit which traverses first and reserves a channel on each switch; the remaining flits will have the same route information as the header flit. Wormhole flow control works at the flit level and because it further divides packets into smaller chunks it reduces the required buffer spaces and the store and forward delay on each switch. Unfortunately the side effect of blocking in cut-through switching becomes even greater in wormhole switching as a stalled packet may occupy all the switches it spans (same as the number of flits).

This effect can be overcome by using resource sharing techniques like virtual channels. Virtual channels is a flow control method where many logical channels may be implemented sharing a single physical channel. This is achieved by having separate buffers for each logical channel. Hence with an overhead in silicon area and power more packets may share the network resources simultaneously which allows wormhole switching to become much more efficient. The number of virtual channels used is a design option based on the amount of traffic that needs to be supported. In such systems there is also the need to have a flow control mechanism that regulates the amount of traffic sent so that a receiver is not overwhelmed; in the case of wormhole switching this can be implemented using a credit based approach which can guarantee that data is only forwarded from a router to the next when there's enough buffer space to hold it otherwise it is blocked and may reside at the current VC buffer until space at the next hop node is available.

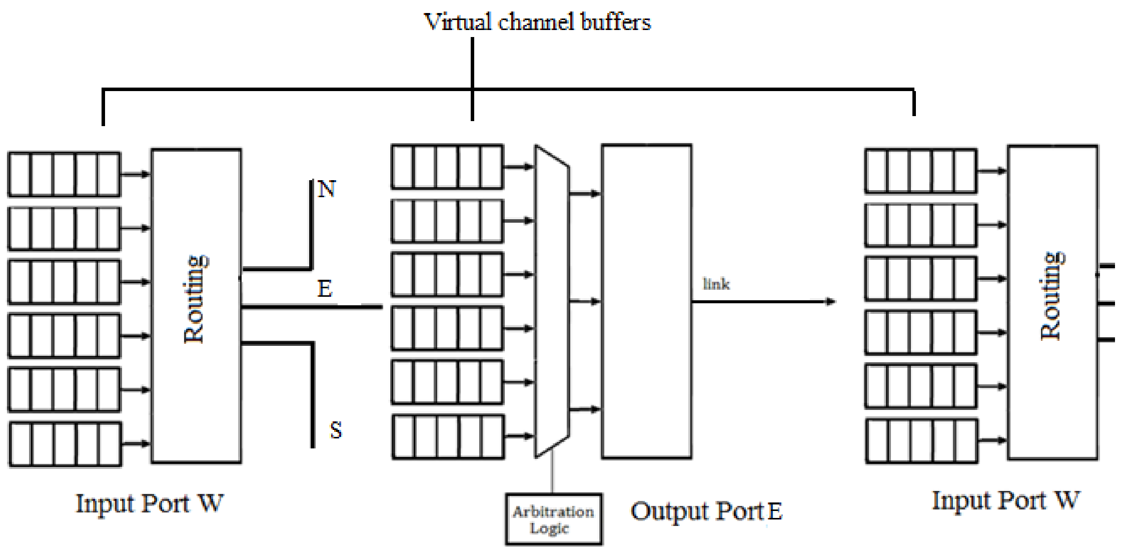


Figure 2.3 Implementation of VC using multiple buffers.

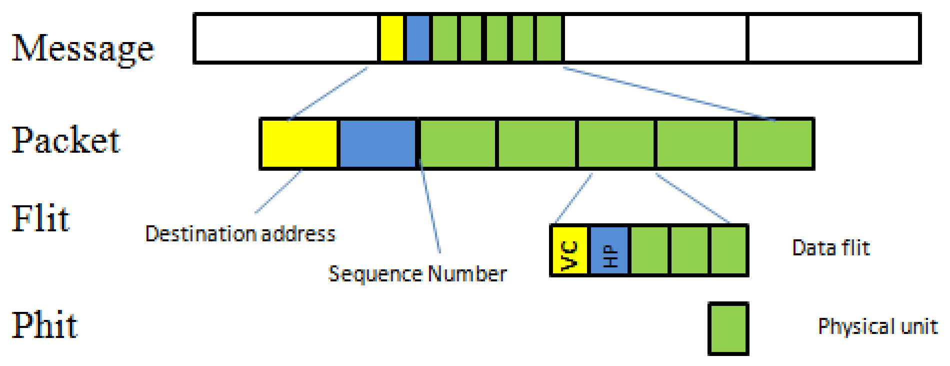


Figure 2.4 Message sub-components for transmission on a wormhole based NoC.

The task of routing in the context of NoCs is that of determining the path of a message travelling from a source PE to destination PE. For this purpose many routing algorithms have been developed. Routing algorithms can be either *deterministic* or *adaptive*. Typically a routing algorithm would have a source and a destination core as inputs and it would produce a path between them. A deterministic algorithm will always produce the same path for the same inputs while the adaptive algorithm will take into account dynamic aspects of the system like congestion and silicon faults and may produce different paths each time in order to improve performance; another class of algorithms is the *flexible* routing algorithms, these algorithms are restricted to using only a subset of the network links from which they may choose a path according to some criteria. Because of the additional resources that adaptive algorithms would require (silicon area for more complex routers and energy to drive them) and the additional non-determinism they would introduce, they are not regarded a very suitable solution for NoCs (a comparison in [30]) and especially for systems with strict timing requirements. Most systems presented in literature use forms of source routing where the path of a packet is known before it is transmitted and each packet carries this information with it allowing for each network node to send it towards the right direction, this decision at every node does not require much logic if the routing scheme is simple.

Some properties of routing algorithms that are usually desirable for NoCs are minimality and freedom from deadlock and live lock, conditions caused by circular dependencies. The prominent strategy for avoiding deadlocks caused by routing paths in NoCs is by using certain routing algorithms which have the property that they do not produce paths that can cause a deadlock; deadlock is caused by cyclic dependencies on shared resources which in a network are the physical channels, the virtual channel scheme overcomes these dependencies by providing multiple channels [83] so different packets contend only for the network links. Minimal routing algorithms always produce a path that is of minimum hop distance between source and destination, this property is desirable in NoCs as many performance and cost metrics are proportional to the communication distance. For a detailed review and comparison of different NoC designs we refer the reader to [15].

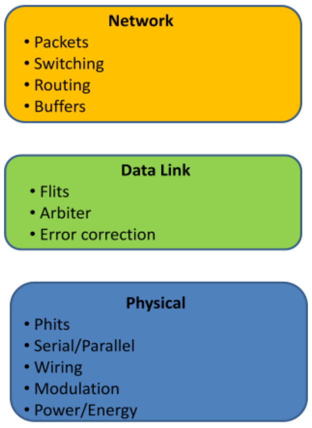


Figure 2.5 Protocol stack for a wormhole based NoC.

2.4 Predictable Communication on Network on Chip platforms

The aspect of predictability is crucial from the perspective of real-time systems. In the packet switched schemes mentioned above packets still contend for shared resources such as the physical links (router output ports) of the network; this contention is the reason for the unpredictability of the network traffic, it can be resolved in many different ways providing flexibility and differentiated services, for example it can be done in a simple FIFO manner [30], where no distinction needs to be made between packets from different traffic flows or it can be based on time division multiplexing [32] where specific traffic flows can be assigned different time slices so that they never contend for the network links. Using this method traffic flows can have a guaranteed bandwidth and latency. A case where both of these schemes exist together on the same platform is that of the *A*Ethereal network on chip [32], this system offers two classes of communication services that use the same infrastructure: guaranteed service (GS) where the communication flows have reserved time slots (TDM arbitration) and best-effort (BE) where the traffic flows use the unreserved time slots using FIFO arbitration; obviously in the later case no guarantees can be provided.

The TDM approach described earlier is characterized by resource reservation where traffic flows must set up paths on the network and reserve resources (time slots) at each node; this can be thought of as a virtual circuit implemented on top of a packet switched network. The advantage of such a scheme is that it can provide deterministic bounds for traffic parameters (latency, bandwidth) however this determinism comes at the cost of poor utilisation (especially for variable rate traffic) and runtime inflexibility furthermore TDM requires a highly synchronous system whereas NoC designs tend towards a GALS approach. A similar arbitration scheme is introduced in [31] which can overcome the limitations of the TDM approach but still providing bandwidth guarantees. This arbitration mechanism is again based on a virtual circuit scheme; each GS flow reserves a virtual channel on every node of the virtual circuit it establishes, then the access of all virtual channel buffers to the link is served at an equal rate, as a result the bandwidth guarantee for a GS flow will be the service rate of the reserved virtual channel (e.g. for 8 traffic flows it will be 1/8 of the total link bandwidth) [31]. This design together with credit based flow control is used in an asynchronous NoC design called MANGO. Other platforms use methods that do not fully utilise the packet switching capabilities of the NoC architecture but are still able to provide GS traffic and QoS guarantees, such a case is the *Nostrum* NoC [71] that sets up a form of virtual circuit for different nodes to communicate; the authors in [75] also propose a circuit switched communication network (SoCBUS) to provide hard real-time guarantees.

Another approach is priority based arbitration [34] [35] where higher priority packets may pre-empt lower priority ones accessing communication resources. Such an approach can provide guaranteed throughput to high priority traffic flows in a more efficient way as it does not reserve resources like the virtual circuit scheme does and can even allow analysis that estimates worst case end to end packet latency [33] which is suitable for hard real time systems. In the case where all traffic flows are equally important it may be hard to determine different priorities; furthermore for the relevant analysis to be applicable the system must again be static with fixed traffic and routes, a scheme similar to virtual circuit connections, even so it will be easier for system to reconfigure itself without suffering the penalty of virtual circuit set up times. In general the priority arbitration scheme seems to be much more flexible; both approaches however are able to provide the desired determinism for time-critical traffic flows. A comparison between the two schemes takes place in [67]. The priority arbitration scheme can also implement flow control techniques which improve the quality of service by using dynamic priorities as done in [74], improving the QoS cannot offer hard guarantees but can increase the performance in soft real-time systems. The QoS that a NoC platform can offer can be expressed with various parameters like availability, jitter, packet loss, and throughput.

One of the main challenges in the design of such a NoC system is the way the network resources are allocated such that the hard temporal requirements of GS traffic are satisfied while the system is not underutilised and the reserved resources are actually used. An approach trying to improve the overall utilisation of the NoC resources is used in the Aethereal NoC; this approach combines both GS and BE traffic flows where BE flows use the network resources when the GS flows are idle to avoid the low utilization, furthermore the BE traffic does not use the virtual circuit scheme so it is more flexible. The low utilization occurs because in real-time systems the resources are always reserved according to worst-case scenarios. Similarly in a priority based system packets with low priority and without hard deadlines may try and use the NoC along with higher priority packets as they will not affect their transmission.

2.5 Network on Chip Modelling

An important step in the SoC design process is the modelling of the system, it is necessary as it helps the exploration of the design space as through modelling it is possible to evaluate various design trade-offs, it is also a means of evaluating a system in terms of performance and also validating the system in terms of requirements and constraints. A good model is also essential for evaluating the performance and cost trade-offs of different mappings. Modelling of complex MPSoCs and NoCs can be either analytical or simulation based, considering that simulation can happen at various levels of abstraction an analytical model is usually needed to abstract away the details of the lower levels.

A way to define different abstraction levels is based on time granularity. Finer time granularity implies lower levels of abstraction (more detailed) as it describes events that occur at smaller time periods hence given a certain duration it describes more states of the system. Using the clock cycle of the computation and communication components as a point of reference abstraction levels can be defined starting from cycle-accurate register transfer level or instruction set simulation level.

Running simulations at this level of abstraction is very time consuming as incorporating this level of detail for a large system would be computationally intensive, but it provides good accuracy in terms of the system behaviour. An RTL cycle- accurate model is used in

[37] in order to evaluate the power and delay in a NoC. Power and delay are first evaluated for even smaller components that make up the RTL components using circuit-physical level simulations and are then incorporated into the RTL blocks making the simulation even more accurate at the expense of running time.

Simulation at this level can be achieved using hardware description tools such as VHDL. An advantage of this level of simulation/modelling is that by incorporating monitors into the system design it becomes easy to obtain accurate figures for metrics that would otherwise be hard to measure (e.g. packet latency). At the other end models that abstract all the implementation details to the system level can be used of course at the cost of accuracy, such an abstract software model is implemented in [39] which models both a NoC and an RTOS managing its resources at system level, this model [40] aids in exploring the design space through different task mappings, RTOS policies and NoC protocols, using the calculated network latency (inaccurate) as a metric for the network performance.

Because of the particular complexity of NoCs, the simulation times at this accurate level are increasingly higher, so alternative solutions have been proposed such as FPGA-based emulation [40]. In such systems where communication between components is largely present it can be abstracted separately from computation in different levels according to which aspect needs to be simulated accurately and also which of the two aspects can be accurately described at higher levels of abstraction in order to speed up simulation time. A concept that works on this idea is that of transaction level models [41]. The basic characteristic of transaction level models is that they abstract away low-level details of the communication and model it as large granularity data transfers. This is done using the concept of channels to model communication where the transactions between computation elements take place through calls to the interface functions of the channels. Transaction level models use three degrees of time accuracy for both computation and communication and different abstraction levels are generated by possible combinations, the three degrees of time accuracy used are untimed, approximate-timed, and cycle-timed.

Cycle-timed computation/communication contains implementation and architecture details at all levels down to the register transfer level, a level that allows cycle-accurate estimation.

Approximate-timed computation/communication contains system-level implementation details and is the level that corresponds to the transaction level for communication; finally the un-timed computation/communication contains only the functionality of the system hiding away all the implementation details. The different abstraction levels can aid at different stages of the design flow in various systems however the more complicated the system architecture the greater the loss of accuracy of these models hence the ability of a TLM to accurately model systems depends on the system communication architecture/ protocols and the time granularity the model used. TLM has been successfully applied to NoC modelling [42], [43] giving significant improvements in simulation time and a small loss in accuracy. In [44] an approximate-timed TLM model of a network-on-chip using wormhole switching with priority pre-emptive virtual channel arbitration is proposed in order to reduce the simulation time and to obtain accurate packet latency figures, the time granularity instead of every clock cycle is the time instants when packets enter or exit the NoC nodes, experimental results showed that this approach achieved a significant speed-up (three orders of magnitude) with a small loss of accuracy (90%) compared to a cycle accurate simulation of the same system. A potentially useful application of this approach would be in cases where only the communication aspect of a design needs to be accurately simulated and the computation can be abstracted away in order to speed up simulation time.

2.6 Summary

This chapter demonstrated some of the distinguishing characteristics of NoC platforms; also it provided a brief introduction to the different architectural variants that can be used each with different costs and benefits. Of specific interest are the different approaches used to provide predictable communications, reviewing the qualities of each aids this work with focusing on a platform type that will be more suitable to the achieve the desired result.

Chapter 3

Real – time Applications and Relevant Models

The goal of this chapter is to provide an overview of the real –time systems domain both from a soft and hard real-time perspective. It is necessary to clarify the requirements present in such systems so that they can then be related to the aspects of the mapping problem that we are trying to solve. Furthermore in this chapter models and tools that can be used to analyse the properties of interest for these systems are introduced.

3.1 Application Domains

This research is mainly interested in reviewing the application mapping problem from a real-time system point of view; in real – time systems the correctness of a system does not only depend on it producing the correct results but also on the time it takes to produce them, so we are generally concerned about applications with real time requirements. An important distinction in real time systems is that of hard and soft real time systems. Soft real time applications are present in many areas such as multimedia and distributed and/or high performance computing, such systems are subject to temporal constraints however for these applications it is acceptable for some traffic to not be serviced correctly as long as the overall throughput maintains a certain level (quality of service). No absolute guarantee for the temporal behaviour of such systems is required, yet performance must be at a certain level on average. This allows the requirements to be expressed using statistical measures. Because of this flexibility such systems are termed as soft real-time.

Other types of applications, such as those found in the hard real-time domain, are not as flexible and require all computational tasks to meet their deadlines at all times [87]. In such a system the temporal requirements are expressed explicitly with deadlines. In general hard-real time systems cannot tolerate any deadline misses the occurrence of which may be catastrophic, for this reason all the aspects of the system need to be taken into consideration at design time so that guarantees may be provided; because of this inflexibility in such systems dynamic behaviour is undesirable unless it can be modelled as a state of the system that can be analysed at design time.

For streaming applications which are very common in various consumer systems the number of different modes that the system has to support is steadily increasing. These modes of operation could be either different combinations of a number of independent applications running on the same platform or various versions of an application’s sub-systems where each may offer different services; for example in a video processing application different codec’s could be used allowing for a trade-off between the quality of the image and the stream bit rate; we could treat those different scenarios as separate use cases. Each different use case will clearly have different requirements in terms of computation, storage and communication. In such an application the temporal requirements may be expressed in various terms such as throughput and end to end latency.

3.2 Application Modelling

This section aims to list the ways that an application that consists of communicating concurrent tasks can be modelled so that it can be mapped on to a multiprocessor system. An important aspect of the mapping problem is the way the applications are modelled. Having an accurate model that conveys information about an application can either allow exploiting this information to reach a better solution quicker or can help validate solution reasoning about the application behaviour under different circumstances. By modelling an application the goal is to abstract the application's functionality regardless of implementation details by simplifying the information that is given about the task set, hence abstract it to a level so that only the information relevant to the problem at hand (mapping) is displayed. Applications that are likely to benefit from networks on chip are complex inherently parallel applications with many communicating subsystems and high computing power requirements mainly because their constraints on power consumption and size can be better met by on chip communication. A concurrent application or system in general is a set of interacting components and can be described by a concurrent model of computation, where this model would be the set of rules that determine the interaction between these components. A component of such a set could be a task, where a task could be defined as a computation that is executed by a processor in a sequential fashion. The use of a model of computation can help design an application and also describe and abstract an existing one. Many computational models have been introduced that describe computational systems e.g. Turing machines and finite state machines, however the need to efficiently exploit task level parallelism demands a concurrent model of computation that can model the dynamic aspects and the concurrent behaviour of the modelled applications. Such models of computation need to describe a concurrent system at such a level so that the non-determinism is minimized by taking into consideration all of the dynamical aspects.

3.3 Concurrent Models of Computation

A distinction between concurrent models of computation can occur by the way they represent time and by whether the system is synchronous or asynchronous. A system's state or behaviour with respect to time can be determined at different granularity (continuous time/discrete time/ untimed). In timed models time is a reference against which all computation can be defined and measured where in untimed models there is only the notion of a logical sequence, in general there is a certain ordering of actions to ensure data precedences and causality but actions can also occur in parallel. Precedences and dependencies can be defined for each system through a concurrent MoC but are also interpreted differently according to it. The notion of the synchronous/ asynchronous model has the typical meaning of a synchronous system where the concurrent tasks are synchronized centrally by a fixed rate clock while in asynchronous system the concurrent tasks synchronize through communication/events.

Concurrent models of computation give a set of rules that describe the interaction of communicating tasks. Those models can be represented in various ways, for example in the model of finite state machines, the behaviour of tasks can be described by state transition rules based on inputs for each individual task-machine (e.g. FSM composite model), multiple concurrent tasks may be executing independently or they can be interacting together asynchronously, in which case sets of state sequences (one set per task) can describe a certain system routine [7].

However such state-based models cannot explicitly express the rules of concurrency. A similar model is Petri-nets [8], in Petri-nets a computation can be modelled as a set of places/states and a transition between places, the system is also token based where tokens can move from one place to another when a state transition happens (fires). In turn a transition fires when there are enough tokens in its input place, since there can be many tokens Petri-nets can model concurrency, and on a higher level tokens can represent different inputs (amount of data, control, external stimuli) again in an asynchronous untimed manner. These models can help with analysing the reachability and determinism of a task set however they do not incorporate information about execution times.

Other asynchronous models are the communicating sequential processes (CSP) model [9] and the process network model. In CSP the interactions between concurrent processes are modelled through the notion of a rendezvous where two processes need to reach a specific state in order to communicate, hence communication is synchronised and thus a central monitor mechanism is necessary to coordinate the communicating tasks/processes. This model is formally described by a process algebra with specific operations between processes where large concurrent task sets may be described and analysed. A different concurrent MoC, that of shared variables, is based on the notion of the processes communication using shared memory locations; this model is not strictly synchronous but some level of synchronisation between the processes must be established when accessing the shared resources to ensure data consistency and freedom from deadlock (locks, semaphores).

3.3.1 Data flow Networks

Another important family of MoC is that of data flow networks. Data flow networks describe the movement of data through various processes in a system; this is achieved by describing the data routes between the processes (dependencies between concurrent tasks) that make up the system and the data storage elements; however they do not consider control flow and the system processes and routes are assumed to be static. In data flow models processes, called *actors*, communicate between them by sending streams of data called *tokens*, the communication between elements is made explicit and represented by edges in the respective graph called *channels*; channels may have some initial tokens called *delays*. The interface of an actor to a channel is a *port* (input or output port). The execution of an actor is termed as a firing, when a firing occurs the actor consumes tokens available in its input port, performs some computation on these tokens and produces an output at its output port; the amount of output tokens is always equal to the amount of input tokens, these amounts are termed *port rates*;. The duration of one complete execution of an actor is termed as its *execution time*. An actor may only fire if sufficient tokens are present in its input thus tokens may also capture dependencies between actor firings and not only data movement.

One characteristic data flow network model is the Kahn process networks (KPNs) [10]. In data flow models a task set is represented as a directed graph where the nodes are tasks and the arcs are channels, in such models the tasks communicate by exchanging tokens through FIFO-buffered *channels* instead of sharing memory. In the Kahn Process network model of computation, concurrent processes communicate through unidirectional unbounded FIFO Channels (buffers). The data transmitted through the communication channels are represented as streams (sequences of data elements or tokens), each process may block on a read when an input channel is empty but writing is non-blocking, the system is hence asynchronous and can also provide determinism which means that each different firing sequence of the actors (schedule) will provide the same output for a specific input sequence. However because of the unbounded buffers assumption the model is somewhat ideal furthermore this assumption seems to give the property of determinism without caring about the order in which the actors perform the series of transformations on the data stream however it is important to find a schedule that allows a KPN to run on bounded buffers without overflowing them.

Another data flow model is the synchronous data flow (SDF) [11]; it is an interconnection of tasks whose firings are enabled by a fixed number of input tokens. The production and consumption rates of tokens by processes are fixed and known. This property makes it possible to determine a schedule where there is no token accumulation on buffers exceeding the buffer spaces; this is equivalent to no rate inconsistency between production and consumption of data. Actors of an SDF graph may fire in various different sequences some of which may lead to deadlock or to rate inconsistencies. For a given schedule there is a property that ensures that the system will not enter such states, this property is called *consistency*; this property means that the initial state of the system (token distribution) can be restored after a finite number of firings (the schedule). So the system executes periodically in a fixed pattern called an *iteration*. The number of iterations that can be completed during a time interval is a measure of the throughput of the graph and is directly related to the execution time of the actors.

Because of imposing the specific sequence (schedule) the notion of synchrony appears. This model is highly applicable as it can accurately model actual data flow systems. Also techniques exist to determine the storage requirements i.e. buffer sizes necessary to support an application at a required throughput [76] or conversely determine the worst case throughput based on buffer constraints and a given schedule.

When multiple actors share a processor their execution must be scheduled so as to provide a bound on each actor's response time. There are several possible strategies for the scheduling of actors sharing resources [77] like static-order, Time-Division Multiple Access (TDMA), fixed-priority and dynamic priority scheduling. Based on these strategies the worst case response time of each actor can be calculated.

We can define a synchronous dataflow graph as a finite set of actors A, ports P and channels C, each actor consists of two sets of input and output ports $a = (Ip, Op)$ where $Ip \subseteq P$ and $Op \subseteq P$, and each channel consists of two ports $c = (po, pi)$; every port is connected to exactly one channel. The rates of the ports are again functions $Rate(P) \rightarrow \mathbb{N}$. The execution time of each actor can be added to this model as a function $E: A \rightarrow \mathbb{N}$ which maps every actor to its execution time. An iteration (consistent firing sequence) can be described in terms of a *repetition vector.* A repetition vector is a function $v: A \rightarrow \mathbb{N}$ such that for every channel connecting actors a and b transmitting data from a to b $Rate\ (po) \cdot v(a) = Rate(pi) \cdot v(b)$. A repetition vector is called *non-trivial* if for every actor a, $v(a) \neq 0$. For an SDF graph a schedule is consistent if it produces a repetition vector that is not trivial.

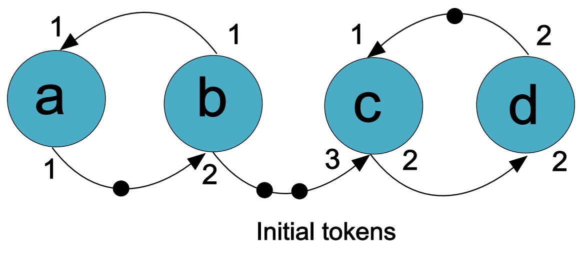


Figure 3.1 Synchronous Dataflow. Token production/consumption rates.

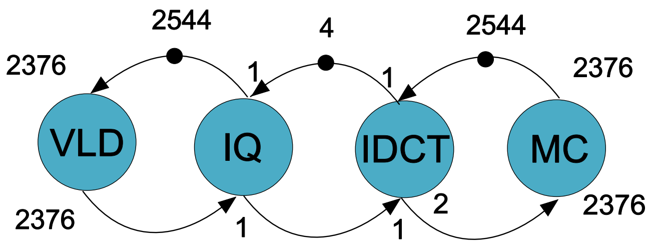


Figure 3.2 The SDF graph of an H.236 decoder. A video codec standard where each token is a block of 63 pixels.

A more general case of synchronous data flow is the cyclo-static data flow model. In this case the requirement that the each actor produces and consumes fixed numbers of token at each firing is relaxed and is replaced by the constraint that the amount of tokens produced/consumed at actor firings can be variable but still can be described by a repeating finite sequence. This model is harder to schedule and it imposes further requirements onto the communicating interfaces (buffers etc.).

The SDF and CSDF models are especially suited for modelling streaming applications such as multimedia and DSP where the system operates on streams of data in a pipelined fashion but some scheduling is required in order to guarantee the temporal requirements of the application, which in those domains is expressed as constraints on the overall throughput/ data rates; this model is both expressive and analysable so it becomes possible to determine the temporal behaviour of the system. The main difference with the traditional schedulability analysis is that the requirement of meeting deadlines is replaced by requirements on statistical metrics which should fall within specified values. The scheduling and placement problem in this case would become that of meeting these requirements by using as less resources as possible. From this description it can be seen that this model is more appropriate to a soft real-time system where the aim is to provide a required level of service.

3.4 Meta-Models

There also exist meta-models that can combine various models together [12] in a hierarchical way where different models can represent different levels, and also meta-models that can further abstract implementation details e.g. in the actors model generic concurrent objects with some attributes communicate via message passing. The model defines interfaces for these objects to their communication channels (ports, queues). Such models may also capture additional information about a system which that may not be possible to convey using the semantics of the models described earlier, for example in order to capture the dynamic behaviour of a streaming application a composite model of a state machine and a dataflow network could be used [81].

Most of the above models describe behavioural aspects of an application, mainly the interactions between communicating tasks and they also tend to have a graphical representation that is based on the rules that govern the system (interaction between components) and represent the specific application features (components and interrelations/communications). These representations can be mainly divided into graph-based models and state-based models, in graph based models the application is modelled by directed graphs where the nodes are tasks and the arcs denote communication relations between the tasks (direction-dependencies/precedences, weights-communication volumes). In state based systems again graphs are used but the nodes represent states and the edges rules that transit from state to state, used in parallel they can model concurrent tasks; also a combination of the two approaches can be used with a token based approach (Petri-nets).

Each representation can convey different information, assuming that the underlying architecture is capable of providing resources for the specific application so that this information is accurate; when the mapping problem is considered a graph based approach is more appropriate because it contains more relevant information like all communications between tasks and their volumes. Considering that the application is at a level that can be described using an untimed synchronous or asynchronous model these representations can give a lot of information about the system/application. Process networks and Data flow models rely on a graph-based representation where the nodes are actors that respond to firing rules and the arcs are channels through which the processes can communicate. Models that assume unbounded buffers when implemented may have issues such as token accumulation however provided enough information they can be scheduled in such a way that those problems may be avoided. As mentioned above these models of computation are especially suitable to DSP and multimedia applications which have a temporally and spatially predictable behaviour.

Regardless the underlying computational model an efficient way of representing an application is required that will contain the important information required for the mapping problem. The computational model of the application gives a lot of information about the behaviour of the system however a more abstract way of representing an application can be used such as communication task graphs (CTG) which characterize the application partitioning into tasks, tasks' type (preferred processor type), communication patterns/volumes, and task execution time. A task graph contains useful information that can be used for spatial mapping according to some optimisation criteria while abstracting away some of the details such as the internal operations and data structures of tasks.

A CTG may represented as a pair (*T*, *C*), where T is the set of nodes modelling generic tasks and C the set of arcs modelling communication between the tasks (control, data etc.). Graph based models such as process networks etc. are already represented in such a way. The importance of using models of computation in this context lies in the fact that they allow to interpret this abstract information and determine the temporal and functional aspects of the application behaviour; for example different communications between tasks may occur at different times according to a MoC and so they may be able to use the same hardware resources (e.g. NoC links) without any problem, information that a CTG is not be able to include. The computation and communication characteristics and requirements of processes can be obtained through various simulation and profiling techniques which are not in the scope of this work.

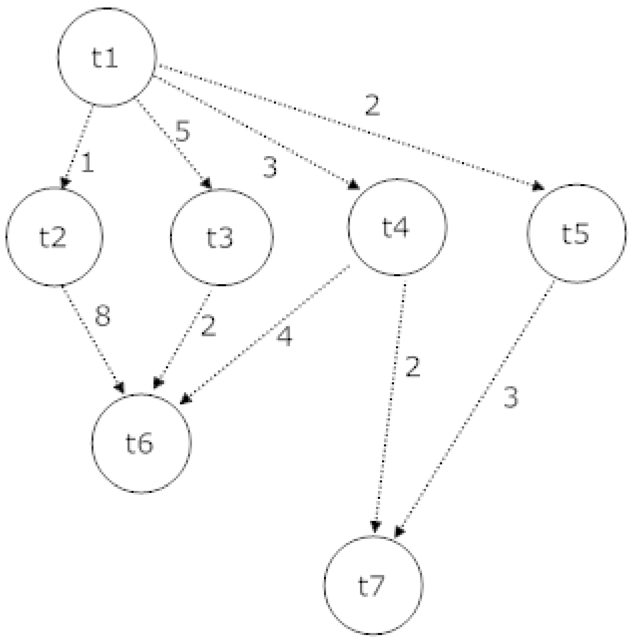


Figure 3.3 A communication task graph – a directed weighted graph.

3.5 Application Models and Analysis for Real-Time Systems

Considering hard real-time applications the model typically adopted is that of a periodic or sporadic task set with or without communications between the tasks allowing pre-emption based on the relative priorities of tasks. This model treats tasks as periodic, sporadic and aperiodic; *periodic* tasks are released within regular intervals and are hence characterised by a period *P*, their longest possible execution time is known a priori and so is their relative deadline *D*. *Sporadic* tasks differ in that they can be released at arbitrary points in the system but with defined minimum inter-arrival time between consecutive tasks instead of a period. Those models have been used extensively in the real time community and there is an impressive amount of literature covering relevant issues such as the scheduling of tasks among single or multiple resources. The main feature of this model is that it allows analysis of the worst case response time of a task based on the interference it may receive from other tasks for a shared resource (processor, memory) and the attributes of the tasks described above, this type of analysis is called schedulability analysis. As the temporal requirements are specified in terms of task deadlines the calculated response time of each task will have to be less than its deadline. Applications within the hard real time domain are typically composed of periodic or sporadic tasks that communicate their results by sending messages. For the timing requirements of all the individual tasks to be met the end-to-end latency of the communications between them has to be bounded; tasks in such systems are released in response to the arrival of a triggering event generated by the external environment, periodically by the system itself, or on the arrival of a message from another task containing the data to be processed within the receiving task deadline. In the later case the tasks period will correspond to the availability of the input data, an increase in the end to end communication delay when transmitting messages will have an effect on the release time of the task making it start executing later than its calculated period, not only making it possible for the specific task to miss its deadline but also for other tasks scheduled together on the same resource. As a consequence it is necessary for communication deadlines to be imposed on communications as well.

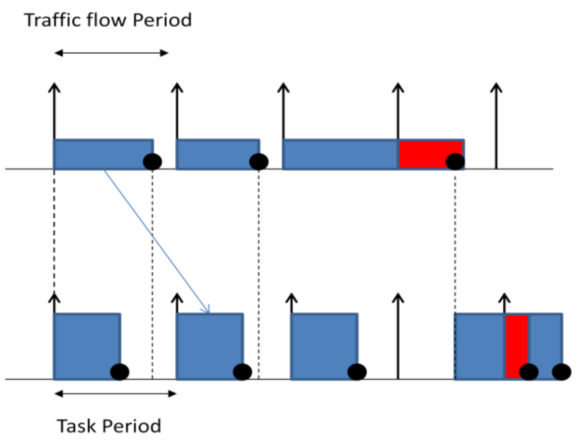


Figure 3.4 Period and release time of traffic flows and their receiving tasks.

Scheduling algorithms can be used to schedule tasks that share a single resource so that they all meet their deadlines. This is usually achieved by assigning fixed priorities to tasks in a pre-emptive system [114]. These algorithms will take as input the attributes of the tasks mentioned above and produce the relevant priorities [89]. In addition to fixed priority schemes there also exist dynamic priority algorithms which have many advantages (optimality, platform utilisation) [116], [115] at the expense of resources that the algorithm requires to run dynamically. There has also been extensive research in the field of multiprocessor scheduling algorithms [113]; the mapping of tasks in this context is termed task allocation and is performed in a way that allows the tasks to meet their deadlines, this problem has also been studied but the communication aspect is typically considered and analysed as competing for a single resource which is the message bus [117], [118], [119], [120]. The two main approaches for scheduling tasks on multiprocessor systems are *partitioned* and *global* scheduling; in a partitioned scheduling algorithm the migration of tasks to other cores is not permitted at runtime so the task allocation problem would have to be solved statically, then communications can be routed and scheduled according to the fixed allocation if possible, global algorithms do permit task migration in which case however without fixed placement deterministic communication would not be easily achieved (analysing worst case communication delays, transmitting messages). As mentioned above real time communications are not usually included in the analytical models except for when competing for a single resource like a bus, for a hard-real time system with communicating tasks a model that could provide guarantees for the traffic on a networked platform (NoC) would be essential; in the next section we will describe how such a fixed priority model can be extended to analyse real–time communications competing for NoC resources treating it as periodic traffic flows with deadlines and priorities.

High performance and low cost are desirable in the real-time embedded systems domain however predictability is the most required feature as it allows any relevant analysis to hold true so in relation to the previous chapter this property would also depend on the platform architecture.

3.6 Schedulability Analysis for NoC Traffic Flows – an Analytical Model

Motivated by the need for NoC platforms that can provide real time services a schedulability analysis approach was presented in [33] based on which the worst case network packet latency can be calculated. For such analysis to be possible the underlying network architecture would have to behave predictably, for this reason this analysis is restricted to NoCs with predictable characteristics such as deterministic routing. Regarding the switching method the case of wormhole switching is considered because of its ability to achieve high throughput with lesser buffer requirements when compared to other packet switching techniques however because of the way the packets are spread throughout the network contention between traffic flows for the network resources occurs frequently and its effects are hard to predict. While a packet is transmitted it occupies network resources such as links and buffers, in such an architecture the shared resource is the links of the network, during the transmission of the packet if a packet from another resource tries to access the network resources contention occurs and it has to be resolved according to some arbitration policy. In order to increase the predictability the network architecture examined is further constrained to NoCs that use wormhole switching with priority based pre-emptive arbitration. Based on this arbitration strategy a real-time schedulability analysis becomes possible. In fact an analytical model is derived based on the system-level behaviour of the system. The analysis proposed can predict the packet worst case transmission latency for a given traffic-flow by successfully adopting a single multitasking processor scheduling model. In order to avoid deadlock one virtual channel per priority level is assumed as proposed in [84] so that there is no dependency between traffic flows and channels.

In priority pre-emptive arbitration if any number of flows traffic flows contend for the same resources the one with the highest priority gains access to the resources blocking the other traffic flows packets and forcing them to reside at the buffers they currently occupy until the resources are free again. In this model all traffic flows τ_i are either periodic or sporadic and have a period T_i, they are assigned a priority P_i and have a deadline D_i. Another parameter that is defined for traffic flows is the release jitter J_i^R which is the maximum deviation of the release of packets from the period of their traffic flow, this attribute helps with the accuracy of the estimation.

Another important attribute of a traffic flow is the maximum network packet latency C_i which is the time a packet of maximum size that belongs to a traffic flow would take to reach its destination node when no contention occurs and is calculated by:

$$C_i = \left\lceil \frac{L}{f} \right\rceil \frac{f}{B} + \mathrm{H} \cdot \mathrm{S} \quad (5)$$

Where L is the packet size, *f* is the flit size and B is the link bandwidth. The distance the packet has to traverse is H number of hops and S is the processing time each router takes. Each traffic flow in the system will be assigned a specific route generated by a routing algorithm. The route of a traffic flow τ_i is a set of links $\beta_i = \{l_1, l_2, l_3... l_n\}$. If a traffic flow's route β_i shares at least on link with another flow's route β_j then the two traffic flows τ_i and τ_j have a direct competing relation in which case the higher priority flow will pre-empt the lower priority one. For a traffic flow the direct interference set S_i^D is defined which is the set of all the traffic flows that have a direct competing relation with τ_i and have higher priority. Two traffic flows instead of a direct competing relation can have an indirect one. In the case where there are two flows τ_i and τ_j and their routes do not share any links but there is one (or more) traffic flow τ_k that has a direct competing relationship with both of these traffic flows (interleaved) and it's priority is such that $P_j > P_k > P_i$ then τ_i will have an indirect competing relationship with τ_j, as τ_j will influence the release patterns of τ_k which will in turn influence τ_i. An indirect interference set S_i is now defined similarly to before which contains all the traffic flows that have an indirect competing relationship with τ_i.

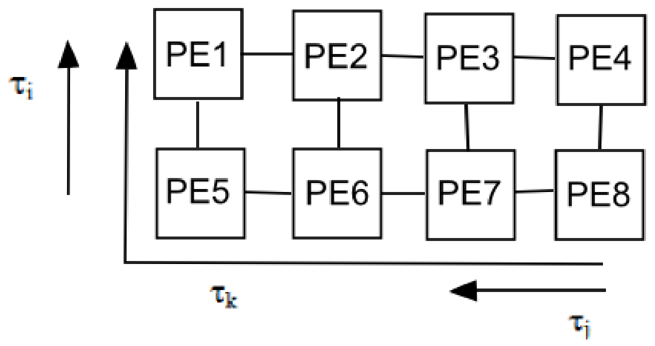


Figure 3.5 Indirect interference between τ_i and τ_j where Pj > Pk > Pi.

The analysis proposed in [33] calculates the worst case network latency R_i for each traffic-flow. The upper bound of the network latency depends both on the basic maximum packet latency and the time a flow is blocked due to interference from higher priority traffic flows.

Under the assumption that for all traffic flows $D_i < P_i$ the network latency upper bound R_i for a traffic flow τ_i is given by:

$$R_i = \sum_{\forall \tau_i \in S_i^D} \left\lceil \frac{R_i + R_j + J_j^R - C_j}{T_j} \right\rceil C_j + C_i \quad (6)$$

For further details, we refer the reader to [33], but the main intuition is that the worst case latency of a traffic flow depends on the distance between its source and destination and on the interference it may experience from higher priority traffic flows which in turn depends on the current communication pattern on the network. The correctness of this analysis is proved in [33] and is also backed up by simulation results in [62].

3.7 Summary

In this chapter the different models that can be used to express the properties of systems composed of concurrently executing tasks were reviewed. The goal of this chapter was to motivate the need to for a model which allows the analysis (available techniques) of the temporal characteristics of both computation and communication in a real time system. As this research will focus on the mapping problem from a hard real-time systems perspective it was explained why the model reviewed in 3.6 would be more appropriate as analytical tools are available that may provide worst case bounds for the network latency of the communications as well.

Chapter 4

Mapping Strategies

This chapter aims to give a comprehensive review of previous works on mapping algorithms. These works are categorized according to the different contexts the problem is solved in (dynamic, static mapping) and different types of solutions. The overall goal is to describe the "state of the art" in mapping algorithms and use this information to inspire a novel approach.

4.1 Approaches to the Mapping Problem

The mapping of computational clusters onto the physical topology of processors has been studied in the field of parallel processing [16], [17], [18] however the mapping of tasks onto NoC cores presents additional challenges because of latency, bandwidth and energy requirements on the links of a NoC and the silicon area limitations of the network components. Mapping of tasks onto the MPSoC platform as mentioned above requires finding the placement of tasks into the platform in view of some optimization criteria. In addition the problem conditions may vary in different contexts; for example mapping communicating tasks onto MPSoC platforms can be accomplished at either design-time or run-time, each case having different constraints on the algorithm running time. Next if the MPSoC platform is heterogeneous, then task binding has to be taken into consideration, where an appropriate platform resource type has to be defined for each task type. The mapping problem varies depending:

- On the metrics that have to be optimised
- Any constraints that are imposed by the platform or the application
- On the tools available to evaluate the metrics and constraints
- Any assumptions made about the system
- When it has to take place i.e. static, dynamic
- On the information available about the system

Because the mapping problem becomes a combination of all these factors there is no solution that applies to every instance of this problem and usually proposed solutions are devised for, and may only apply, to a specific pair of platform and application types. The main purpose of this work is to approach this problem in the real – time application context and find appropriate solutions.

4. 2 Static Mapping

In the case where the application to be run on the platform is known at design-time and it is not going to change dynamically it would reasonable that the mapping is performed at design-time as well. In this situation, the platform is dedicated to running the specific application and no dynamic features like task insertion, removal or migration are allowed during runtime. The algorithms that deal with mapping in this case are called static mapping algorithms. The advantage of this situation is that the algorithm has all the necessary information about the application (which will remain static throughout the lifetime of the system) and that the running time of the algorithm is not too important. Therefore static mapping algorithms can be expected to provide close to optimal solutions.

4.2.1 Evolutionary Methods

Because of the nature of the static mapping problem meta-heuristics are usually employed. This is the case in [19] where a two stage genetic algorithm is used with the objective of optimising the execution time of the application. It assumed that the network is packet switched and that the routing algorithms give the shortest path routes between source and destination (minimal). The effects of contention for the communication resources are not considered mainly because the routing algorithm used is adaptive so in case of congestion it will pick different minimal paths that avoid it, because of this fact however it is not possible to provide any latency guarantees. Similarly in [22] a genetic algorithm has been used. In this case the algorithm tries to optimise multiple objectives at once by searching a Pareto front of solutions and trying to find a solution from the optimal set. The NoC architecture in this case is a 2D-mesh with static XY routing and wormhole switching and is abstracted as a graph that contains information which specifies the functional behaviour of each element in the NoC such as timing and power consumption parameters, while the applications are either statistically synthesised or come from real traces. Solutions are evaluated using simulations instead of analytical models and the simulation framework is capable of evaluating many different metrics such as energy and processing time. The advantage of this simulation based approach is that it can take into account dynamic effects which affect the quality of the solutions. In this approach the genetic algorithm is not entirely random based, the crossover and mutation operations are applied in a way so that they alter the positions of heavy/demanding communication flows in particular as a result the produced solutions are more diverse. This may lead to a better exploration of the solution space also in the case where searching for solutions that provide latency guarantees as the heavy flows are more likely to affect the overall latency in the system.

4.2.2 Greedy Heuristic Methods

In [23] a different approach is proposed, again assuming a packet switched mesh based NoC with wormhole switching the main cost objective to minimise is the aggregated bandwidth used on each of the network links, hence the goal is to balance the bandwidth requirements of the application across the different links. In this way it is suggested that indirectly congestion and latency will also be reduced. The congestion on the network links affects the remaining available bandwidth of the links and increases latency due to contention that may lead to blocking (wormhole switching). Again the application is modelled by an application graph, in this work called a core graph, which includes information about the application behaviour. The algorithm used is greedy like and has an initial phase where first the task with the highest communication demands is mapped first and then the other tasks are selected in turn; in the selection step the task that communicates most with the already mapped ones is picked next. Then this task is placed on a core so as to minimise the communication cost of the set of flows of the application (E).

$$Comm.Cost = \sum_{i=1}^{|E|} bw\ (i) \cdot dist(i)\ (7)$$

For each traffic flow *i* in E where *dist*(i) is the minimum number of hops between the source and destination of the flow nodes. This best core in each step is obtained by examining every available core in the platform then a routine will give a minimal route. The procedure is repeated until all the cores are mapped. If the bandwidth constraints are satisfied the solution is saved otherwise discarded. This initial solution or partial solution up to the point that the constraints are satisfied is then improved upon by pair-wise swapping tasks for all possible pairs and saving the best mapping.

In addition to this algorithm an approach is proposed where a single traffic flow can be split in two minimum paths in order to better distribute the bandwidth requirements of the application across the platform and minimise congestion, latency and the jitter between subsequent packets, with the small overhead of additional routing tables at the router buffers. This approach is validated by cycle-accurate simulation of a DSP system. Again this approach could be used in order to reduce the overall congestion on the network links however on a priority based network it could increase contention as by splitting the traffic-flows would increase their total while applied on a TDM based network could lead to better utilisation of resources.

In [25] a similar approach is used where the objective is to minimise the communication energy in a 2D mesh NoC with XY routing. Here two greedy algorithms produce solutions which are used as seeds to a simulated annealing algorithm. The two greedy algorithms are largest communication first (LCF) and greedy incremental (GI). In the LCF heuristic tasks that communicate the most are mapped first. The amount of communication for each task is the total incoming and outgoing communication with other tasks. The algorithm places highly communicating tasks from the centre of the platform outwards so that there are more route options. The greedy incremental algorithm works on a randomly generated initial mapping and is based on two nested iterations. The first loop defines a core as a pivot iterating for all the cores of the platform; the nested iteration performs swaps of tasks between all possible pairs of the platform's PEs except for the PEs already used as pivots; at the end of each nested iteration the mapping with the least energy consumption is saved and finally when the algorithm terminates the best mapping is chosen; this approach is very similar to an exhaustive search.

The solutions of these algorithms were used as seeds to the simulated annealing and tabu search heuristics and the results were compared against those of the heuristics without seed solutions. These algorithms were tested adopting the communication model of [26] for randomly generated application sets varying in degree of connectivity, communication weight, and for varying NoC sizes. The combined approaches of the greedy algorithms and the heuristics showed good improvements in terms of execution time and energy saving comparing to the simple heuristics.

This is an idea that can be easily adapted to different cases as well. In [47] the authors have developed a greedy algorithm which works according to different application templates.

The input applications are classified according to the connectivity of the vertices of their communication task graphs. The weights of the communication traces are derived similar to [27]. The algorithm classifies the input applications by keeping a sorted list of edges in decreasing order of edge weight, the applications are of two different types, the applications that fall into the first type are applications that have at least a node of degree four or higher and at least one of the edges connecting to this node is at the higher 50% of the sorted list (highly weighted).

The presence of such nodes (tasks) indicates that there are one or more hot nodes which communicate heavily with flows that either require high bandwidth or have tight latency constraints. The second types of applications are applications that do not have such nodes hence the weight of their communication is more evenly distributed across connections. In the first case the algorithm maps the hot nodes first along with their four most significant neighbours which are determined by the weights of the edges connecting them at a minimum hop distance. These nodes are mapped on tiles of the platform that have the highest amount of neighbouring tiles (central tiles with 4 neighbours). The remaining nodes are mapped in descending order based on the weights of their edges. In the case where the application falls into the second category the algorithm divides the NoC into regions and the communication graphs into blocks in a in a divide-and-conquer manner similar to [27] then it maps the tasks on IP cores inside each block again in a priority based on communication weights. XY and odd-even routing are considered as the routing algorithms used by the platform in order to generate deadlock-free and minimal routing paths. Simulation experiments using [48] on multimedia and random benchmarks show that this algorithm produces good quality results in terms of energy spent with low run times, however the algorithm is not described to take any action in cases where constraints are violated hence the relation of the algorithms performance to the size of the platform used is not apparent. Such an approach could be also applied to a system with latency requirements where the communication flows could be weighted according to their deadlines producing optimal or close to optimal solutions that could be further processed by other steps.

4.2.3 Unified Approaches

The mapping problem is tightly coupled with other design aspects of the system such as the system topology and the routing function. This fact motivated many researchers to try and solve the mapping problem as a part of a larger design space solution for NoCs allowing for solutions optimised more towards specific applications.

In [4] Hu and Marculescu propose a branch-and-bound algorithm for the mapping problem which aims to minimise the total communication energy and meet the application constraints by bandwidth reservation. The algorithm tries to find both an optimal mapping and a routing function that minimise the energy spent on communication, also having the bandwidth usage as a constraint. The concept of bandwidth reservation means that after the mapping, the communication traffic on any of the networks links will not exceed the available bandwidth, hence it will be serviced. The energy model used to evaluate the solutions calculates the energy used to send one bit from one core to another by:

$$E_{bit} = No.of\ hops \cdot E_{S\,bit} + (No.of\ hops - 1) \cdot E_{L\,bit} \quad (8)$$

Where E_S bit is the energy the switch consumes to process one bit and it is measured by running circuit simulations in SPICE and E_L bit is the energy consumed when the bit travels along a wire (one hop distance). The Branch and Bound algorithm represents the solution space as a tree and it traverses it by generating children nodes for each parent node. In the specific case each node of the tree represents a partial solution and each leaf node a complete solution, the solutions consist of both a partial mapping and the corresponding routing paths of the communication flows. In such systems as mentioned above the routing has to have some fundamental properties such as being minimal (shortest route) and deadlock free. The above approach does not use a single routing algorithm instead it incorporates the routing paths as part of the solution. During each level of the algorithm for each node the next children nodes are generated by mapping the next unoccupied task enumeratively on to the remaining unoccupied cores. Then a routing path allocation heuristic that runs in reasonable time generates routing paths which are both minimal and deadlock free. Then the bounding stage is based on evaluating the solutions generated and discarding the ones that fall above certain energy thresholds and or violate bandwidth constraints. Experimental results show substantial communication energy savings compared to other heuristics like simulated annealing while the running time of such an algorithm is relatively high as expected.

Benini et. al in [24] adopt a unified approach for designing application specific NoCs where the specified hardware cores suitable to the applications tasks are mapped onto different NoC topologies. In this approach the hardware cores on which tasks will run are known a priory and a tabu search algorithm tries to map the cores on different topologies trying to find the best combination of mapping/topology. This work also differentiates by the fact that a heterogeneous platform is assumed where predetermined specialised PEs for each task are mapped on a topology instead of tasks being mapped on various PEs. The algorithm perturbs the mappings in order to traverse the solution space by pair wise swapping mapped cores. The objective in this case is to minimise the delay of the traffic flows measured in hop counts in order to meet the delay constraints and provide QoS and real-time services. The traffic characteristics of the application like the permissible delays (deadlines) of the traffic flows are obtained by simulating the application. This process in the tabu algorithm is combined with a physical planning stage which optimises the design area, power consumption or hop delay by regulating the size and positions of network components. This approach is able to provide high quality solutions as it customises the whole system for a specific application incorporating mapping as one of the many design aspects.

Murali et. al[20] introduce a methodology aiming to find a mapping that meets the bandwidth and latency constraints of multiple use cases of an application that are known a priori and capture some dynamic characteristics of the applications communication patterns. In this approach, path selection for different traffic flows (routing) and reservation of bandwidth (TDMA slot-table entries) for the GT traffic flows are unified with the mapping process in a manner similar to [5]. The different use cases of an application all are assumed to have the same set of tasks and they only have different communication flows. The algorithm is restricted to the *A*Ethereal architecture and mainly considers GT flows and their requirements. The algorithm is a heuristic and it works by iterating through all the use cases, looking for the most bandwidth demanding traffic flows. The algorithm begins by sorting all the traffic flows from all use cases by decreasing bandwidth values in a list and then it iterates through that list visiting high bandwidth demanding flows first. When considering a traffic flow the algorithm finds a path that has the least cost in terms of latency (measured in hop counts) and bandwidth (available slot reservation) in order to satisfy the flows requirements, then it maps the traffic flows source and destination tasks on the paths edges. The mapping of the tasks is shared between all the use cases and the algorithm reserves bandwidth for the mapped traffic flow on this path. This algorithm can also iterate in an outer loop for various NoC sizes so that it can find the minimum area that can satisfy the applications communication constraints. This approach contributes to the fact that it can capture dynamic aspects of the application (different modes/use cases). Experiments in [20] compared the method described above to a previous approach where a worst-case use-case was synthesised that combined all the constraints and requirements of all the applications use cases hence over-specifying the requirements, the results showed a big improvement on the NoC area that was necessary to meet the constraints. Such solutions can also be adopted in systems with tight latency requirements and by incorporating the routing in the solution the solution space increases.

4.2.4 Generic Approaches

In [26] an approach that uses a communication dependence model (CDM) is presented. The aim of this work is to minimise energy consumption. The communication dependence model provides additional information about the application into the mapping problem, specifically the information of dependencies between messages. Again assuming a 2D mesh network with deterministic XY routing which is abstracted by a NoC topology graph the application is modelled both by a communication weighted graph and a communication dependence graph. In this work the energy consumed by the system is separated in static and dynamic, dynamic energy is consumed by the communication fabric transmitting the traffic flows and the general switching activity of gates on the network (routers, buffers) and the static energy is proportional to the total number of gates dissipating static power and to the execution time (mostly by information residing and being processed). So the total energy consumed by the NoC running an application is:

$$E_{NoC} = E_{StNoC} + E_{DyNoC} \quad (9)$$

Previous works described here have tried to reduce the energy consumed by communication. The main difference in this work is that by taking into account the message dependencies, hence the sequence that they are produced, the contention between traffic flows can be reduced, so can the static energy dissipated and overall execution time. The algorithm used is simulated annealing and it tries to minimise the quantity in (7) hence the dynamic energy as well. The main observation that makes the dependencies information useful is that messages that have precedence relations will not occur at the same time while independent messages may occur simultaneously and lead to contention over the communication resources. This information allows the algorithm to narrow down its search space by trying to avoid only contentions that will actually occur. For each mapping the algorithm will maintain for each link a list of all the independent messages that share it and next it will assume the worst case where these messages contest with each other for the link. Message contention implies a larger application execution time (based on the arbitration scheme used) and consequently more static energy dissipation. For each mapping the algorithm computes the dynamic energy due to the distance and the static energy due to each message delay which is computed by calculating the delay due to precedence (add previous messages' delays) and contention (messages that share same links). Therefore this algorithm that uses the additional information from the CDA tries to minimize the contentions that will actually occur by searching core mappings that spread the messages over parallel links. The drawback of this approach is that it can be very hard to obtain a static communication dependence graph at design time that will be constantly valid throughout the application's lifetime as it may depend on input data available only at run time [26]. Testing the algorithm against a plain simulated annealing algorithm that only considers communication energy, significant reductions are achieved in execution time and smaller reductions considering static and total energy consumption, however the authors suggest that the static energy consumption will become more important in the future as technology will scale down. This approach could be very useful for traffic with real time requirements as it allows to take into consideration only the traffic that will be present on the system at any given time hence solve the same problem for less traffic flows however the relevant information must be available and accurate.

In [27] a technique is proposed aiming to meet latency constraints of communication flows while minimising the energy spent on communication. This approach is restricted to mesh based NoCs and takes an application and a mesh topology as inputs. The application is modelled as a communication trace graph which is similar to a communicating task graph with the addition that each edge also carries the information of the latency constraint and bandwidth requirements of the flow. The weights of the application graph edges are determined by the latency and bandwidth of the flow. The arc weights are calculated so that an arc representing a traffic flow with very tight latency constraints has larger weight than an arc representing a flow with high bandwidth requirements. This work assumes that the network used does not have a specific arbitration scheme and that many traffic flows may share the same links as long as their aggregated bandwidth does not exceed the bandwidth of the link, so for any given mapping the bandwidth constraints on shared links have to be satisfied. The mapping problem in this work is solved using an algorithm that solves the graph equicut problem. The equicut problem consists of finding a partition of a directed weighted graph so that the number of nodes in both partitions is equal and the amount of flow (aggregated weight of edges) crossing the partition is minimum which is very similar to the sparsest cut problem on graphs. The algorithm recursively partitions the communication trace graph vertically and horizontally solving the equipartition problem each time until the partitions contain only one node, then the relative positions of the nodes denote their placement on the XY plane. In this method the way the communication trace graph is partitioned depends on the weights of the communication flows, in turn these weights are derived according to latency and bandwidth requirements however being more biased towards latency. This fact ensures that communication flows with high latency constraints will be placed closer to each other but also represents a trade-off between placing high bandwidth traffic flows close to each other in order to minimise energy consumption. This algorithm has a polynomial time complexity and experiments demonstrated that in some cases the algorithm was able to produce close to optimal solutions however with this approach the energy requirements conflict with the latency constraints.

As it can be seen there have been various approaches presented in literature for the problem of static application mapping onto the mesh architectures. These approaches vary in the aspects of the system they try to optimise and the methods they employ. Some of the approaches tackle the mapping problem together with other design aspects of the system such as routing and time-slot allocation and as a result they provide good quality application instance specific solutions; other approaches use additional information about the application they try to map in order to achieve better results but they become applicable only in cases where this information is available. Different algorithms are used in these approaches however optimal results are not always guaranteed in addition in some cases the algorithm complexity may have limitations on the problem size it can process e.g. [23]. Most algorithms consider specific platform and application features so they may not be generally applicable.

4. 3 Dynamic Mapping

Different to static mapping, in dynamic mapping the time taken for the mapping algorithms to run and produce a valid solution is important since it becomes an overhead to the overall application execution time. A compromise has to be made between the solution quality and the running time of the algorithms used. Complex applications running on NoCs may have dynamic workloads so the number of parallel tasks executing may be varying in time. Such applications may run on homogeneous platforms where the uniformity of the system simplifies the mapping problem and also allows for easy reconfiguration (task migration), or they may run on heterogeneous platforms in order to take advantage of specialized processing elements and increase performance. Apart from the dynamic aspect of applications it may be preferred to have tasks mapped (loaded) on the system at the specific instance they are required so that the PEs of the platform are not unnecessarily occupied; doing so allows to save power and to have more mapping options at any time being able to find optimal configurations easier; it may also be the case that the entire number of tasks that an application may require during its runtime may exceed the available resources so in any case it is necessary to add tasks into the system on a per need to basis.

Different techniques have been proposed in literature so that a system may respond to varying situations at runtime, e.g. different communication volumes, which may lead to performance degradation. Task migration [63][64] is a technique where at run-time the tasks are relocated from one processing element to another when a performance bottleneck is detected or when the workload needs to be distributed more evenly across the platform e.g. thermal balancing. Task migration on such systems however is essentially a re-mapping of some tasks and would again have to employ some mapping heuristic method, in addition it comes at a high overhead such as the cost to save a tasks context (memory issues arise that are architecture specific), transmitting all of the data to a new PE and restarting the task, an accurate estimate of this overhead is necessary so that the relevant strategy can take it into consideration.

Complex NoCs can also be used in situations where the system may run a variety of software applications instead of only one and/or the applications may have many different modes of operation and various dynamic characteristics depending on user input. Because in a dynamic system the optimal resource allocation will vary different approaches to those used in static mapping are required. A major distinction in dynamic mapping algorithms is between algorithms that map the entire application task set, even though some of the tasks do not have to be present at all times, and other algorithms that map only single tasks of the application on a need to basis; algorithms that fall in the first category are termed resource-reserving because of the need to reserve resources for the not present tasks so that they can be mapped when available.

4.3.1 Per single-task Mapping Methods

One of the first approaches for NoC based MPSoCs was proposed in [48]. The authors present three heuristics for dynamic mapping all of which work on an initial placement of some of the applications tasks. The objective of these algorithms is to reduce the congestion on the NoC links (congestion aware). This work considers heterogeneous NoCs and assumes the system may be running multiple applications that do not communicate with each other. A task in this model is mapped on the platform only when another task requests to communicate with it so the mapping problem is solved for one task at a time; each application is modelled as having some initial task that has to be running at all times. First the mapping of this initial task takes place; in the case of multiple applications a clustering strategy is employed. The NoC is partitioned in clusters and each applications initial task is placed inside a cluster, all the subsequent tasks of the application will be placed into the same cluster too; the aim of this strategy is to equally reserve resources for each application and to minimise the sharing of NoC channels by communications of different applications in order to avoid link congestion.

The first heuristic proposed is called Minimum Maximum Channel Load. When a task is required this heuristic visits all possible different mappings for it on the available elements in the application's cluster. The cost of a mapping is the maximum data rate/ Bandwidth occupied across all the NoC links which indicates the maximum congestion at any time in a cluster. The algorithm chooses the mapping with the minimum cost. Next the Minimum Average Channel Load heuristic aims at reducing the average bandwidth occupancy on the NoC links. The difference of this heuristic with the minimum maximum channel load is that instead of using the maximum occupancy as the cost metric it uses the average. As a result this heuristic tries to evenly distribute the communication load throughout the NoC links. The mapping that results in the lower average channel occupancy is chosen. Another algorithm proposed in this work is Path Load. This algorithm measures the cost of a mapping only on paths used by the communication of the task being mapped thus being faster. Again the cost measured is that of bandwidth occupancy and the mapping with the least cost is selected. Finally the Best Neighbour heuristic calculates the cost of different mappings same as in the path Load heuristic but instead of all possible mappings it restricts the search only to the neighbours of the initial task requesting a new task to be mapped. If a valid placement is not found the distance is incremented from one to two hop counts so the search is extended to the NoC limits along a spiral path. Some simple heuristics were used as a point of reference to compare the ones proposed above. This heuristic was Nearest Neighbour, in this heuristic there is no cost evaluation of solutions and it works by searching for a core able to execute the requested task starting from the neighbours of the node making the request. The search iterates through all n-hop neighbours, with n varying between 1 to the NoC limits. Experiments were carried out [49] using the Hermes NoC (8x8, 9x9) described at RTL in VHDL, the processing elements were modelled using systemC. Because in the context of dynamic mapping the system itself has to run the heuristics on a specific processor one processor is dedicated to this task called the manager processor, modelled as a systemC thread it is responsible for task mapping along with other resource management tasks. The experiments were carried out for a variety of real and synthesised applications on both homogeneous and heterogeneous NoCs. All algorithms reduced the average channel load; in addition because of the simulation based result other metrics could be evaluated such as packet latency and algorithm execution time, overall Path load seems to offer the best results. The algorithms proposed in [48], [49] and [50] seem to produce good results compared to ad – hoc approaches, however they do exhaustively search the solution space as a result they may not run sufficiently fast in large NoCs. It can be argued that the optimality of a single task mapping may not lead to the optimality of subsequent task mappings so these algorithms could produce overall suboptimal mappings as the number of tasks mapped on the system increases. Further these algorithms do not guarantee to satisfy any constraints while in many cases it may be necessary. Using additional input about the system state and the application or having some kind of reconfiguration strategy could potentially address this issue.

In [53] dynamic mapping algorithms are proposed which assume that the processing elements of the system are able to execute more than one task. The goal of the heuristics presented in this work is to optimise various aspects of the system by reducing the communications' distances. This work assumes a mesh based heterogeneous NoC which also contains reconfigurable hardware areas that may be dynamically reconfigured and run more than one hardware task in parallel (FPGAs). It is also assumed that the system runs various independent applications where each application starts by running an initial task and then the rest of the tasks are mapped on demand (based on communication requests by previously mapped tasks) similar to [50]. The algorithm does not know the application task graph and characteristics in advance and is assumed to be running on a separate processor, the manager processor. This work builds on the work in [50] by taking into consideration the fact that the PEs of a system are able to run more than one tasks. These algorithms are based on an initial packing strategy which is similar to the clustering approach in [48], after the mesh NoC is partitioned in clusters one for each application (assuming that all applications' initial tasks are available at the same time) initial tasks of the applications are mapped at the centre of clusters, subsequent tasks are mapped close to the requesting task but are not restricted to the specific cluster, clustering is only used to place initial tasks evenly across the platform. The subsequent tasks are mapped according to the packing strategy which looks for available PEs in a circular manner at increasing hop distances from the PE making the request. The packing strategy specifies an order according to which the PEs at each hop distance are traversed, this order is left, down, right, top so that each application is more compact towards a specific direction (left, bottom) in the cluster allowing free space in the cluster for other applications that will look in the same direction. Using this strategy the applications can share the NoC resources more efficiently by being more compact towards a specific direction.

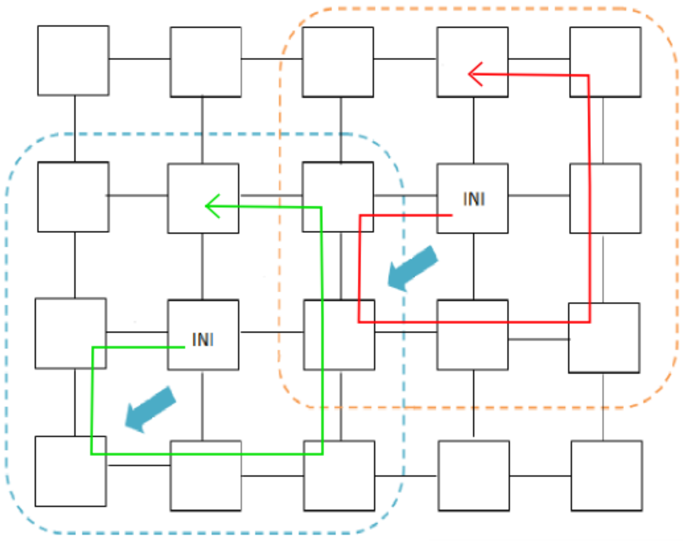


Figure 4.1 Packing strategy for different applications sharing the same NoC. Applications map their tasks towards the same direction allowing them to better share the NoC.

The algorithms proposed here are based on two algorithms proposed in [48] namely Nearest Neighbour, and Best Neighbour. The proposed approach enhances those algorithms using the packing strategy mentioned above and by introducing the notion of communication awareness resulting in communication aware packing based Nearest Neighbour and communication aware packing based Best Neighbour. In these enhanced versions of the above algorithms whenever a task is requested to be mapped fist it is attempted to map it on to the same PE in order to take advantage of this feature and minimise the communication overhead, if it is not possible the platform is scanned for an available PE of suitable type in increasing hop distances from the initial requesting task according to the packing scheme mentioned above. In the case where a suitable PE is not available then the task waits in a queue until some resource becomes free.

In communication aware packing based Nearest Neighbour a greedy search looking for an available PE takes place, if the search comes across a PE which can support the task and there are no other tasks mapped on to it then the task is mapped and the search stops. If the search comes across a suitable PE with already mapped tasks on it checks to see if those tasks communicate with the task that is to be mapped, if it is so then the task is mapped on the this PE otherwise the search continues until a suitable free PE or a PE with such tasks is found. This approach aims to map communicating tasks on the same PEs when possible in order to minimise the communication cost. The communication aware packing based Best Neighbour is based on the Path Load algorithm in [48] all the mappings of the incoming task on neighbouring PEs are evaluated according to the total bandwidth usage on the links (path load) and the one with the minimum is evaluated, because of the platforms ability to run multiple tasks on a single PE solutions that do so are favoured as they do not require any traffic on the links. In order to verify their approach the authors used a simulation framework to test the algorithms. The framework used was an 8x8 2D mesh NoC modelled in VHDL having the PEs modelled as systemC threads. One task and PE are reserved for running the above algorithms are resource management the manager processor. Using random and tree like applications as inputs with both software and hardware tasks notable reductions in the communication overhead were noted in comparison to the simpler versions of the algorithms affecting both energy and execution time.

In this work the mapping algorithm runs on a single processor (mapping control is centralized) a fact that may become a bottleneck as the system size increases. The feature of multitasking on a single PE should be supported by an operating system especially if the tasks have real time requirements where they will have to be scheduled in which case the problem complexity will increase probably posing a significant overhead to this approach. Again these algorithms are not able to provide any guarantees in terms of latency and bandwidth constraints, in cases where a mapping of a task is not possible it will have to wait on a queue thus affecting the overall performance in it's absence, a reconfiguration strategy might be able solve this bottleneck problem.

The most recent work to date targeting the dynamic mapping problem on NoCs is presented in [54]. In this work the authors target homogeneous NoCs where the PEs have the ability to multitask; the overall aim is to minimise the energy consumption in the system. The proposed algorithms are Dependencies Neighbourhood (DN), Low Energy Consumption – Dependences Neighbourhood (LEC-DN) and Premap – DN. The DN is a simple heuristic that maps incoming tasks as closely as possible to the already mapped tasks in which they communicate with; by using a proximity only cost function (no. of hops). So the task is placed in a position where the total sum of the distances to the communicating tasks is minimised. The LEC-DN uses the total communication energy of a task mapping as cost function which is a function of the distance of the communications and the communication volume between tasks. In the case were the incoming task only communicates with one task then the search for the suitable PE is done similar to NN [48] where the search starts from the requesting task's neighbours outwards in a spiral fashion, in the case where the incoming task communicates with more than one task the search is restricted in a box bounding these tasks, if a solution is not found in this box then it's size is increased by one hop.

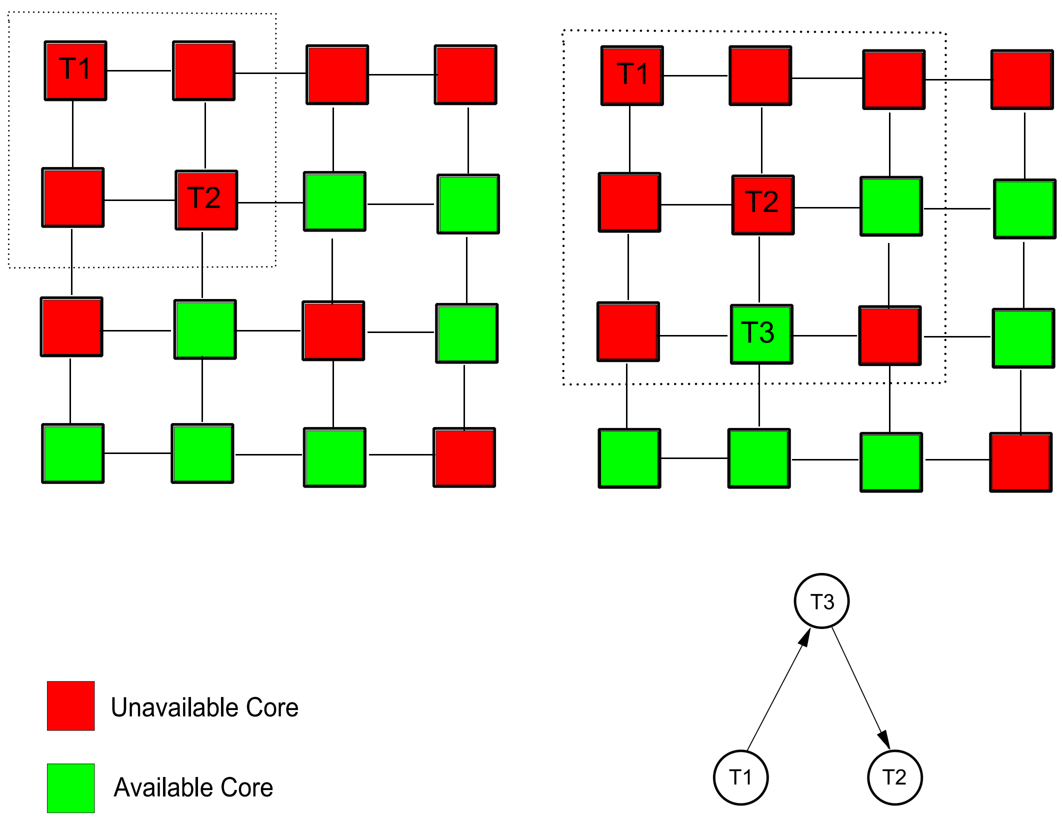


Figure 4.2 Box bounding of search region for the mapping of task 3.

Another proposed algorithm is proposed in this work is *Premap-DN*. The aim of this algorithm is to group tasks with high communication rates on the same PEs in order to utilise the multitasking capability of the system. This heuristic uses a method called pre-map which allows tasks to reserve places on the PEs of the platform so that the communication costs are minimised. This method is used after the mapping of the initial tasks of the application. For an initial task t_A after it is mapped the pre map method will reserve positions for the tasks that communicate with it on the same PE, this is done by maintaining a list with all the tasks that t_A communicates with but are not yet mapped called the communicating list. This list is sorted according to the communication volumes, next the algorithm iterates through the list and for each task t_B it visits it is pre mapped on the same PE with the source task t_A. Next for any tasks that have not been pre-mapped and are requested on the system the LEC-DN heuristic is used, in this case after the task is mapped the pre-mapping algorithm is invoked again for the communication list of this task.

Experiments using a variety of real and synthetic applications were carried out to validate the proposed approaches. The algorithm pre-map LEC-DN was tested against a multi-tasking version of NN mentioned in [48] and the mono task algorithms were tested against each other (DN, LECDN) and against the mono task version of the NN heuristic. In general experiments show that LECDN produce better results for mono task scenarios and Premap-DN heuristic effectively reduces the communication on the NoC and also produces significantly better results than the NN approach.

The works described in the section above propose simple fast solutions to the dynamic mapping problem on a per task basis, however they are not able to provide any guarantees that certain requirements will be met at all times during the lifetime of the system.

4.3.2 Resource Reserving Algorithms

Another approach for runtime spatial mapping was proposed by Hölzenspies et al. in [51]; this approach targets streaming DSP applications running on heterogeneous mesh based NoCs. The aim of the proposed method is to find a mapping at runtime that minimises the energy spent on communication and satisfies the applications QoS requirements. The algorithm is named hierarchical search with iterative refinement the main characteristic of the algorithm is that it solves the problem in different steps where each step helps to narrow down the solution space. In this approach the fact that the platform is heterogeneous is considered in order to produce an efficient mapping where each task runs on a processing element type that is most preferable e.g. an FFT kernel on a DSP. This affinity between the task and processor types somehow restricts the solution space so it is assumed that each task can have a few implementations able to run on various processor types e.g. an FFT kernel may run on both a DSP and an ARM processor but less efficiently in the later case, some tasks are less flexible as implementations for different processor types are not efficient. The fact that these various implementations are available enables multiple applications to run simultaneously on a heterogeneous system. The proposed mapping algorithm runs whenever an application needs to start executing. Considering that all tasks of such streaming applications run concurrently the algorithm maps the entire application instead of separate tasks.

The algorithm is composed of four steps; considering heterogeneous NoCs and an application that is made of various tasks the first step aims to find a processing element type (DSP, ISP etc.) that suits each task; this process is done iteratively for all tasks first choosing the tasks that are less flexible (regarding processor types). Assuming that each processing element can run multiple tasks the chosen tasks are mapped on to the first suitable processor found with sufficient resources (first fit). The second step of the algorithm tries to improve the current mapping obtained from the previous step by remapping tasks to more suitable processor types. In addition this step considers communication costs, having found a optimal processor type assignment for a task the algorithm will swap this task to another processing element of the same type and will choose the one element with the least communication cost. The total communication cost of assigning a task to a processor is the sum of the Manhattan distances between the mapped task and all of the tasks it communicates with. The iteration in step two can stop or continue based on a threshold in time or in the gain achieved by the alternative mappings. In step three the communication flows between tasks are routed on the NoC, the flows are sorted by non-increasing throughput so that the heavier flows are routed first. Next the algorithm finds a shortest path between the source and destination of the flows by only considering links that are able to service the throughput requirement of the flow as their capacity may be occupied by previously routed flows. Finally the last step checks that the current mapping satisfies the applications constraints.

The application is modelled initially as a Khan process network that describes the dependencies between tasks; additional to this information the QoS constraints are necessary for each communication flow between tasks. After a specific mapping of tasks on processors the resulting system can then be described as a cyclo – static data flow graph; according to the specific processor types chosen for the tasks additional information becomes available such as WCET and production and consumption rates. Based on the CSDF graph of the resulting system the buffer capacities required by each task mapped on a PE can be determined using an analysis presented in [58]. If after the last step of the algorithm the throughput constraints on the links cannot be satisfied or the required buffer sizes of tasks cannot be realised the algorithm re-runs previous steps so that they may lead to different solutions. Similarly if any step fails to find a result the algorithm re-runs previous steps so that it can reach different solutions. Experiments carried out in [51] tried to map a HIPERLAN/2 receiver system on a NoC platform composed of two ARM and two MONTIUM processors. The algorithm was compiled and run on an ARM processor and for this problem inputs it produced feasible solutions with a small running time; however it is not clear if the algorithm will be able to produce solutions sufficiently fast in cases where the application is very large as it involves various iterations. More experimentation with a wider variety of problem sizes would indicate whether the algorithm is able to scale well.

Chou and Marculescu in [56] suggest an approach which includes user behaviour information in the task mapping process. This behaviour information is employed to define tasks periodicity and communication rates. This work targets NoC platforms that may run multiple applications at the same time; the dynamic aspect of the system comes from the fact that it may load additional applications at run-time where different application tasks run on different resources (processors). In addition it is assumed that an application may also leave the system when no longer needed. The behaviour of a user is defined as a set of consecutive events requesting applications to enter or leave the system over a given period in the system lifetime. Two approaches are proposed with the aim of minimising contention and the communication energy cost between tasks which is defined as a function of distance. The contention between the traffic flows for the network resources in this work is separated into internal and external, internal contention occurs between tasks that belong to the same application while external contention between tasks belonging to separate applications.

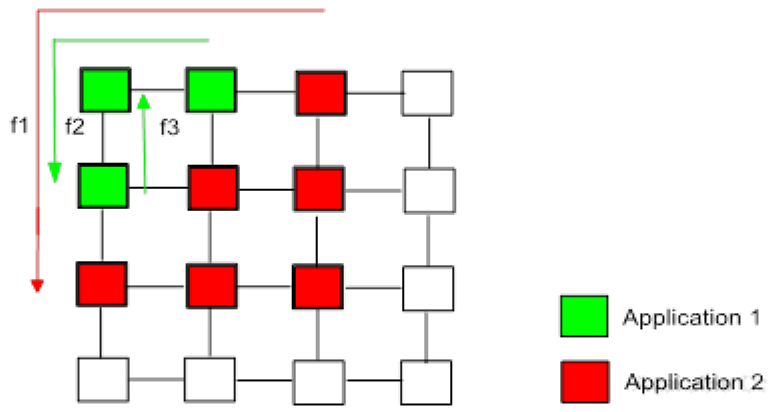


Figure 4.3 . Internal contention between traffic flows f2 and f3, external contention between flows f1and f2.

The two methods for mapping an application on to the system are the following: The first method tries to map the application in such a way so that the internal contention and communication cost (energy) is minimised without considering the external contention. The second method tries to minimise the external contention and the communication cost of the application. Each of the two approaches relates to a different problem, in the first approach an arrangement on the 2D plane containing all the application tasks is found which minimises the communication energy, based on this arrangement a region is reserved on the NoC to map the tasks; as a secondary requirement if there are more than one arrangements that minimise communication energy, the arrangement that is closer to convex is used. In the second approach a near convex region is selected on the NoC according to the free resources available at the time and the application's tasks have to be mapped on it so as to minimise the energy cost; the fact that the area chosen is convex minimises the external contention that this application may cause.

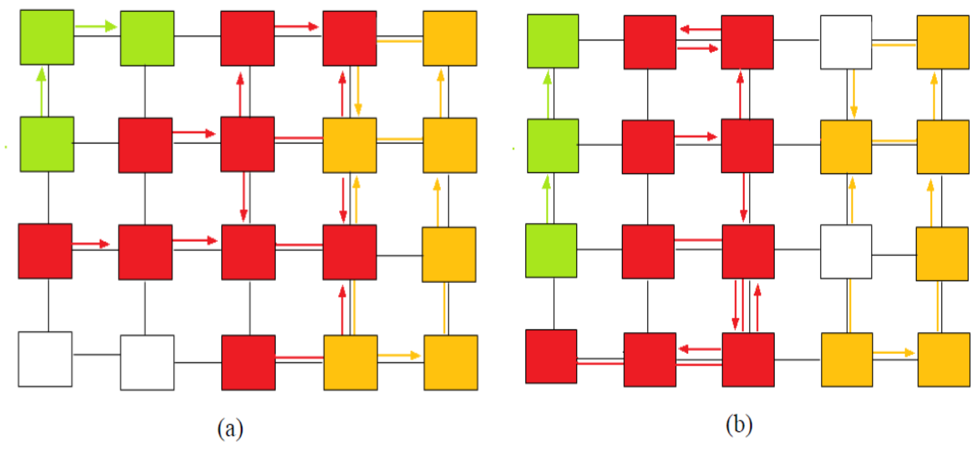


Figure 4.4 mapping solutions for three different applications, in (a) the internal contention is minimised while in (b) the external contention is minimised.

The mapping algorithm used in the second approach is a greedy algorithm and works by mapping heavily communicating tasks first. In the beginning the most communicating task is mapped at the centre of the region then at each subsequent step the task with the largest total communication to already mapped tasks is chosen and mapped at a processor so that the total communication cost is minimised. The complexity of this algorithm is given as $O(VlogV+AlogA)$ Where V is the number of vertices and A the number of arcs in the application graph.

The system chooses one of the two approaches based on the application's characteristics and the previous user behaviour (previously mapped applications). Applications that have a total communication rate higher than other applications or applications that have been responsible for most of the communication energy spent by the system in the past (lifetime of the system) are mapped using method one which minimises only the communication cost of the application, such applications are termed critical. The reasoning behind this choice is that using the second approach for critical applications which is friendlier to other applications a suboptimal placement in terms of energy will be found while for critical applications it would be more suitable to find an optimal solution regarding the overall system performance.

Experiments were carried out using applications from a benchmark suite [60]. The NoC platform is a 5x5 NoC that uses wormhole switching and minimal path routing. A custom made simulator was used to measure the communication energy consumption.

This approach targets a specific case where entire applications may enter and leave the system; it exhibits power consumption improvements when compared to a random approach that maps each application randomly yet in a contagious fashion. The proposed method does not take into account any constraints that the system may have such as bandwidth usage on the links or packet latency, also the user behaviour model is not necessarily applicable in systems with arbitrary behaviour however this method shows a way of relating the geometrical features of a mapping and the contention that occurs (assuming the model of multiple applications onto a single NoC).

Similarly Mehran et. al. In [55] propose an algorithm that tries to minimise the cumulative energy consumption across the NoC links and consequently the time spent on communications by placing highly communicating tasks close to each other. This work targets the mapping of a single application on a NoC similar to the work done in static mapping without taking into consideration any dynamic aspects of the system however due to its low running time the authors suggest that it may be used in a dynamic mapping scenario. Assuming a homogeneous platform the algorithm works by sorting the platform's nodes and the application tasks in lists. The nodes are sorted according to their connectivity in a non-decreasing order this results in the platform priority list (PPL) which list all the network switches starting from central switches ending to boundary switches in a spiral fashion. Next the tasks are sorted on a list according to a priority (tasks first on the list will have higher priority and should be mapped first) forming the task priority list (TPL). Each task is assigned a priority based on the number of tasks it communicates with. High priority tasks are placed on PEs next to highly connected nodes so tasks first in the TPL are placed on PEs connected to nodes first in the PPL, when a task from the TPL list all of the tasks they communicate with are placed at a minimum hop distance and the processes iterates until all tasks are placed. The main idea of the algorithm is that highly communicating tasks are placed from the centre of the mesh outwards together with the tasks they communicate at minimum hop distances. The framework used for experiments was the simulation environment presented in [61] assuming a homogeneous NoC architecture with XY routing, the algorithm was tested against a genetic algorithm and a random approach, for synthesised application graphs with high communication characteristics and on various NoC sizes ranging from 3x3 to 6x6, the number of tasks in the applications was assumed to be the same with the number of PEs on the platform. The algorithm would run considerably faster than the genetic algorithm especially as the size of the platform would increase and in some cases find more optimal solutions. The algorithms complexity is not explicitly calculated as it is essentially a greedy heuristic and simulations would help evaluate whether it would be applicable in runtime scenarios; also the algorithm does not take into consideration system constraints and does not have a strategy in case a suggested mapping violates any of them.

As shown above various heuristics have been proposed as solutions to dynamic mapping. In the context of hard real-time systems such approaches would be unsuitable as it would be impossible to analyse all the possible states of the system at design time and provide guarantees for deadlines. In a case common in dynamic systems the potential set of tasks that may run on the system is known a priory but it is not necessary that these tasks will be running simultaneously at all times; if this set at its maximum demand is feasible under some multiprocessor scheduling algorithm, as mentioned in the previous chapter, a partitioned algorithm would have to schedule and allocate the tasks at design time (static) or alternatively using a global scheme there would be no tools available to analyse communication.

In addition the execution time of a dynamic mapping algorithm that would (re) allocate a set of tasks so that their deadlines are met together with those of the relevant real-time traffic could not complete within a bounded reasonable time interval as the problem would become especially hard to solve; on the other hand using some kind of heuristic at runtime that would run up to a pre-specified interval could not guarantee a solution to this problem. On the contrary soft real-time systems are more flexible in the fact that they may tolerate deadline misses which may appear as a degradation of the provided services allowing these algorithms time to search for solutions. In many cases in a soft real-time application like streaming may be running on a fairly generic platform sharing it with other applications and it may also change according to dynamic aspects of the system so in this context dynamic mapping algorithms are more relevant even though the real-time requirements are not taken into consideration in many of the works mentioned above.

4. 4 Summary

In this chapter we tried to provide a comprehensive review of approaches to the static and dynamic mapping problem. Because of the hardness of the problem heuristic algorithms are proposed which try to optimise different metrics; these metrics vary according to the systems that each author is considering. Regarding static mapping a general observation is that typically most of the work tries to minimise communication energy consumption having the bandwidth usage on the network links as a constraint. The algorithms reviewed try to optimise different aspects making certain assumptions and using specific models hence comparing different approaches is not necessarily useful since most of these solutions target specific systems; even so this review remains useful as many of these approaches could potentially be adopted for the case of mapping real- time applications. In the case of dynamic mapping the mapping algorithms are required to provide an output quickly so greedy heuristics are used that search the solution space locally following some logic that should lead to continuously better solutions, obviously there is no guarantee on the ability of this type of heuristics to find globally optimal solutions or solutions that meet a specific quality or requirement within a specific time interval. This is a downside for their application in hard real-time applications which could not depend on dynamic mapping heuristics because of the inherent non-determinism. The review of this class of algorithms still remains useful however both for soft and hard real time systems; in the first case they could be directly applied and in the later case greedy/ quick heuristics may serve as a pre-processing step to a static mapping heuristic algorithm increasing its probability to find a high quality solution and/or reducing the time it takes to execute, also the algorithm intuition could be used to guide the search of the static heuristic instead of having an entirely random approach.

Chapter 5

Mapping Algorithms for Hard Real-Time Applications

This chapter aims to introduce a novel approach for the mapping of hard real-time applications on network on chip platforms which is based on evolutionary algorithms. The suggested approach and its motivation will be described in detail as well as further variations/additions. A brief description of the development and experimentation framework will follow and finally experimental results will be demonstrated together with the relevant conclusions.

5.1 Proposed Approach

As it is described in the chapters above ever since the introduction of NoCs there has been significant research in the field of task mapping, usually with the goal of optimizing performance and cost. Many works have identified the relationship between packet latency and the task mapping on NoC platforms and have also have used it as a metric to optimize through various mapping methods. However a significant portion of embedded system applications have real-time requirements and little research has been done on providing guarantees for hard real-time applications running on such systems.

Applications that fall into the hard real time domain are typically composed of periodic or sporadic tasks that communicate their results by sending messages. For the timing requirements of all the individual tasks to be met the end-to-end latency of the communications between them has to be bounded, so communication deadlines are imposed on traffic flows. In a packet based network the packet latency is directly dependent on the distance between communicating nodes and the contention from other traffic flows hence it directly depends on the current topological mapping of the tasks onto the processing elements and the resulting communication patterns. The aim in this case is to find a mapping where this latency for every traffic flow does not exceed its deadline.

Several NoC architectures can provide guaranteed throughput to some of the transmitted traffic flows, while providing a best effort service to the rest, as described in [31]. Most NoCs that provide such services use variants of time division multiplexing (TDM). In order to guarantee the required bandwidth for each traffic flow a pre-assigned time-slot is allocated to use resources. Alternatively NoCs with priority pre-emptive virtual channels can be used. In such systems by allowing high priority packets to pre-empt the transmission of low priority ones, network contention becomes predictable; in addition the ability of traffic flows to meet their deadlines (schedulability) can be analyzed using the approach presented in [33]. In priority pre-emptive virtual channels if any number of traffic flows compete for the same link the one with the highest priority gains access to the resources blocking the other traffic flows. Because of this fact, it is possible to calculate the maximum amount of time a particular flow will have to wait before it can transmit its payload completely. The main advantage in using priority pre-emptive arbitration over the TDM approach is that it does not unnecessarily reserve resources, so low priority traffic can always use the NoC resources if there are no requests from high priority traffic, additionally by using simpler arbiters a good trade-off between latency guarantees and hardware overhead is achieved.

In this chapter we demonstrate an attempt to solve the mapping problem in a real-time system context. Using the schedulability analysis for the type of system described above presented in [33] and the platform type described above we try to find a mapping solution that allows all traffic flows of an application to meet their deadlines.

5.2 Development Framework and Methodology

In order to develop and validate any mapping algorithms a development framework was necessary. Typically as mentioned in the review section researchers validate their mapping algorithms using a simulation framework for a given NoC architecture; the communication and computation are usually both modelled at a cycle accurate level especially in the case where the energy consumption of the system has to be calculated and when the mapping is performed at runtime and the system has many states. In this case we consider a closed system where the all the application characteristics are known at design time and are to remain the same during runtime. As we consider real-time applications using the model described above and we only consider the temporal requirements we can validate any mapping solution in terms of computation and communication by using the analysis presented in [33]. Because we can rely on a purely analytical model a simulation framework in this case is not necessary to test a proposed algorithm; instead a system model can be used that implements the relevant analysis.

A framework based on the system model approach was developed at the University of York by various contributors and was used for the purposes of testing mapping algorithms; the algorithms were developed as part of the framework which was itself modified to implement additional functionality. The framework was developed in Java which being object orientated and high level would easily implement the system level and analysis model while abstracting away low level details saving development time. The model of the NoC platform describes the major components as objects: processing elements, routers, buffers, and links and from the object relations other properties emerge e.g. topology; also the application is modelled as task and traffic flow objects with various attributes; with this model it is also possible to evaluate quickly every metric that can be accurately described analytically by implementing the relevant analysis as a function e.g. utilisation. In this work, we consider only 2D mesh topologies with homogeneous processing, but nothing prevents the application of the proposed approach to other architectures; for example it could be applied to a heterogeneous processing platform in which case a different WCET for each task to processor binding would have to be provided and the task set and communications would again have to be schedulable under the new execution rates resulting from tasks being placed on new processing elements, analysis has been extended to heterogeneous systems previously [126].

This framework was sufficient for a proof of concept approach for testing proposed algorithms but a simulation framework would be necessary in the case where more elaborate cases would have to be tested although simulation alone cannot provide validation for real time systems. Below we list various cases where a simulation framework would be beneficial. In addition to finding mappings with schedulable communication and computation additional objectives may have to be optimised; an important metric in embedded systems is the avg. power consumption which can be only accurately measured through simulation. Next mappings where the actual WCET analysis is tight (values recorded during simulation runs close to values calculated from analysis) may be desirable. Many of the design aspects for NoC and embedded systems in general trade off between silicon area/ power consumption and predictability, in order to further evaluate any technique that may offer more predictable communication it is necessary to determine the cost in relation to the relevant improvement. In the case of dynamic systems and mapping algorithms simulation is necessary to evaluate their performance and impact because such algorithms will have to execute at the runtime of the system. In the domain of soft real time systems the temporal requirements may be expressed as throughput constraints or different measures such as average packet loss or average jitter in which case simulation is needed to evaluate these metrics.

5.3 Genetic Mapping of Hard Real-Time Applications

5.3.1 Genetic Algorithm

Motivated by the fact that the network latency and the interference between traffic flows depends on the relevant traffic patterns which in turn depend on the mapping of tasks we employ a genetic algorithm with the objective of finding a mapping where all the traffic flows in the application are able to finish transmission before their deadlines even in the worst case scenario. Genetic algorithms were selected as the method to employ over a variety of heuristics used for combinatorial optimisation problems e.g. simulated annealing. The main reason that genetic algorithms were used was that the different mappings can be directly applied on by the genetic operators i.e. be directly treated as chromosomes in a genetic algorithm, allowing the heuristic to efficiently operate and the implementation to be straightforward; furthermore genetic algorithms consider many candidate solutions together (generations, population) a property which should allow the search of the solution space to be more thorough at the expense of completion time, this trade off in comparison to simpler heuristics and their execution time would depend on the problem hardness and size [121], [125] . An additional advantage would be that the operators in a genetic algorithm (crossover and mutation) can have their relative weights altered, for example a genetic algorithm relying only on mutation could resemble a local search algorithm operating on many solutions [124]; furthermore in addition to a good cost function targeting the problem at hand the operators could be guided, either in the way they mutate solutions or in the way they favour solutions with specific characteristics [122], this could allow the algorithm to find better solutions and/or to try and optimise other metrics as well [22]. These characteristics of genetic algorithms suggest that they are highly flexible and can be tuned to better suit a specific problem.

The inputs to the proposed algorithm are the models of the application and of the NoC-based platform. The application model used is that of periodic and sporadic tasks; the tasks are described with attributes such as priority, period, computation time, deadline and release jitter. Tasks communicate with each other through traffic flows, which inherit some attributes of the tasks that generate them (period, priority, deadline) and also have their own attributes, such as the amount of data they transmit and the release jitter that they suffer, which is derived from the worst case execution time of the respective task, as a result if a task misses it’s deadline the traffic flow it generates will miss its respective deadline as well; so every schedulable traffic flow implies that its generating task is schedulable as well unless there are tasks that do not generate or just receive traffic flows in which case the schedulability of traffic flows does not imply the schedulability of tasks. The attributes of the traffic flows are then used to carry out the worst case latency analysis for each different flow for a given mapping.

The following assumptions are made about the system: it is a homogeneous platform where each tile has a processing element with local memory, each processing element may accommodate many tasks and if tasks on the same processor are to communicate with each other they do so by using the local memory instead of the network links. In order to make the last assumption more realistic a constraint could be introduced on the total memory size on each tile; each communication on a tile would continuously occupy its data size in memory and the total memory requirements should not exceed the available amount.

Each solution in the algorithm can be denoted by the pairing of tasks and processing elements in a many-to-one cardinality (a task can be assigned to only one processing element, a processing element can have many tasks mapped). This form of solution is naturally suited to a genetic algorithm approach. Each solution can be ranked by using the number of traffic flows that cannot meet their respective deadlines under this mapping as the solutions score. We use the analysis presented in [33] as our ranking function: the score of each solution/mapping is calculated using equation (8) for every traffic flow.

An optimal solution should have a score of zero. In case a solution with score zero is found the algorithm halts. The genetic operations are purely random: the crossover operation combines two mappings to produce a new one and the mutation operation just perturbs an existing solution/mapping. The algorithm begins by generating a population of random solutions/mappings. The selection step of the algorithm calculates the score of each solution and the average score of all the solutions in the current generation, it then discards all the solutions with score higher than average. Next the current generation is repopulated using crossover at one random point between the accepted solutions.

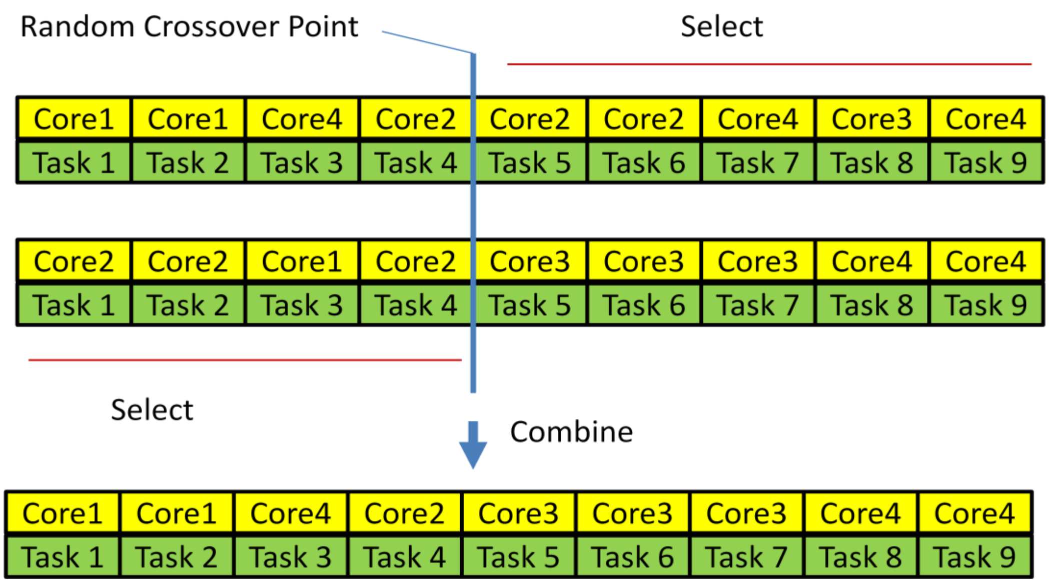

Random Crossover Point
Select
Core1 Core1 Core4 Core2 Core2 Core2 Core4 Core3 Core4
Task 1 Task 2 Task 3 Task 4 Task 5 Task 6 Task 7 Task 8 Task 9
Core2 Core2 Core1 Core2 Core3 Core3 Core3 Core4 Core4
Task 1 Task 2 Task 3 Task 4 Task 5 Task 6 Task 7 Task 8 Task 9
Select
Combine
Core1 Core1 Core4 Core2 Core3 Core3 Core3 Core4 Core4
Task 1 Task 2 Task 3 Task 4 Task 5 Task 6 Task 7 Task 8 Task 9

Figure 5.1 Crossover operator

Finally each member of the population is mutated with a probability of 50% (similar to mutating a fixed percentage of the population but more random). The mutation operation in this instance is just a pair wise mapping swap of two tasks picked at random. A more guided mutation operation picks an unschedulable task at random and remaps it to the least utilised core which is easy to find by maintaining the list of cores in a data structure sorted by utilisation. The mutation operation in a genetic algorithm plays an important role as a factor that introduces randomization and allows the search to escape local minima for this reason we adopt a strategy where both the random and guided mutation operations are applied on a population to get the benefits of both approaches.

The first mutation operation mentioned above tries to make the computation more evenly distributed across the platform however numerous approaches exist for the task allocation problem in real-time systems that could potentially produce better results while our main concern is to achieve schedulable communications. Intuitively this would require that the relevant contention between traffic flows is minimised so the next mutation operator suggested above aims to put communicating tasks together on the same core so that less traffic has to traverse the network thereby reducing interference. Minimising the contention cannot only be achieved through the mapping of the application but also through the routing of the traffic flows. As explained in previous chapters the mapping has an obvious impact on the communication distance and contention so the mapping could be guided towards a solution which optimises these metrics; a more obvious way to optimise communications on a network would be through the routing function that produces paths for the system traffic, the benefits of this approach are that it would in a way decouple the mapping of the tasks from optimising communication, so for a given mapping done based on a placement/scheduling heuristic a routing function would try and find the best possible paths for the communications; next incorporating routing into the search space would allow to better utilise the network interconnect capabilities in comparison to a simpler algorithm such as XY routing however the produced routes should have some certain characteristics such as being minimal.

```
Generate an initial population of random mappings of size n
while (!solutionFound){
  evalutate(population)      //Calculate the score of the solutions
        if(!solutionFound){  //Look for solution with no deadline misses
             for all solutions in population
                    if (solution.DeadlineMisses >Avg.DeadlineMisses)
                       {Remove solution}
             while (populationSize!= initialSize) {
              Solution s1 = random Solution from population
              Solution s2 = random Solution from population
              Solution s3 = randomCrossover(s1, s2)
               Population.add(s3) }
           for all solutions in population
           mutate with probability 50% }   }
```

Listing 1. Pseudo-code of the genetic algorithm

The steps described above are iterated until a suitable solution is found. The current approach was chosen because it was relatively straightforward to implement in addition to the fact that the computation time is not a limitation in static mapping. As a first approach, the proposed algorithm applied genetic operators randomly as this will help with comprehending the problem complexity and the size of the solution space and will give us a point of reference to compare with more elaborate algorithms.

5.3.2 Route Selection

The problem of finding a selection of paths for the traffic flows of a network so that the contention is minimum has been studied before and was shown to be NP-hard in [85], [86]; in [85] the authors propose an interesting heuristic that attempts to find a solution to this problem by trying to generate paths with low contention values. The amount of routes that intersect a given route is termed the *overlap,*

For a set of routes P and a route $\beta_i \in P$,

$$overlap\ (\beta_i) = |\beta_i : \beta_k,\ \beta_k \in P \text{ and } \beta_k \text{ intersects with } \beta_i\,|. \quad (10)$$

In the context we are interested in we could translate the overlap of routes to direct interference between traffic flows [33]. A metric that measures the overall overlap for a set of routes is *contention* and is defined as: given a set of *m* routes $P = \{\beta_1, \beta_2 ... \beta_m\}$,

$$Contention\ (P) = \sum_{i=1}^{m} overlap(\beta i)\ \ (11)$$

So contention is defined as the total number of overlap between routes which is equivalent to direct interferences between traffic flows. More formally the minimum contention search problem is defined as follows: given a graph $G = (V, M)$, where the vertices V represent network nodes with tasks mapped onto them and the edges M represent communications between these nodes, find a set of $|M|$ routes $P_{all} = \{\beta_1, \beta_{2,...}\ \beta_m\}$ so that the contention between all routes (P_{all}) is minimised.

The algorithm described in [85] is based on the idea of generating a set of alternative routes for each traffic flow P_f and then combining those sets together in a set P_{all} which represents the solution space for this problem. The algorithm is based on iterating on the set P_{all}. At each step of the iteration the route in P_{all} with the minimum overlap $\beta_{\min}$ is found next all other routes of the same traffic flow $P_f\,(\tau\,(\beta_{\min}))$ plus all other routes that intersect with $\beta_{\min}$ are deleted and placed in set P_{del}. Next the overlap values for the routes in P_{all} are recalculated and the iteration continues. At the deletion step if the overlapping route to be deleted is the last remaining route of the respective traffic flow the initial choice for $\beta_{\min}$ is cancelled and the route with the next highest overlap is chosen; this happens because the last remaining route of a traffic flow cannot be removed. If after a full iteration all traffic flows have one route and no route deletions have been cancelled then the solution is fully contention free, if traffic flows with multiple routes remain the route with the algorithm reiterates (minimum overlap is chosen, the rest are deleted and the overlap is recalculated).

The intuition behind the algorithm is that by finding the route with the minimum number of intersecting routes (min. overlap) in each iteration and by deleting these intersecting routes a route with no contention is generated, since the amount of the intersecting routes is minimum over an iteration so is the impact of removing them on the solution space.

An adopted version of the algorithm in [85] is described below.

```
1  For each traffic flow $\tau_i$ generate $P_f$, and $P_{all} \leftarrow P_{all} \cup P_f$
2  Initialise $P_{del} \leftarrow \emptyset$, $P_{mc} \leftarrow \emptyset$
3  Until $P_{all} \leftarrow \emptyset$
4          For each $\beta_i \in P_{all}$ calculate overlap ($\beta_i$), find $\beta_{min}$
5              delete all routes in $P_f(\tau(\beta_{min}))$, $P_{del} \leftarrow P_{del} \cup P_f(\tau(\beta_{min}))$
6              for all routes $\beta_k$ that intersect with $\beta_{min}$
                   if ($P_f(\tau(\beta_{min})) - \beta_k \neq \emptyset$ )
                       delete $\beta_k$, $P_{del} \leftarrow P_{del} \cup \beta_k$
7                  else   $P_{all} \leftarrow P_{all} - \beta_{min}$, $P_{mc} \leftarrow P_{mc} \cup \beta_{min}$, continue step 3
8                  if all intersected routes deleted successfully
                           $P_{mc} \leftarrow P_{mc} - P_{del}$,      $P_{all} \leftarrow P_{all} - P_{del}$
9  Until $P_{mc} \leftarrow \emptyset$
10   For each $\beta_i \in P_{mc}$ calculate overlap ($\beta_i$), find $\beta_{min}$
11     if ($P_f(\tau(\beta_{min})) - \beta_{min} \neq \emptyset$ )
12       delete all routes in $P_f(\tau(\beta_{min}))$, $P_{del} \leftarrow P_{del} \cup P_f(\tau(\beta_{min}))$, $P_{mc} \leftarrow P_{mc} - P_{del}$
```

Listing 2. Pseudo-code of the minimum contention route generation algorithm

A main difference in the way we apply the algorithm in is the set of available paths for each traffic flow, a characteristic that is necessary in this context is minimality which can guarantee freedom from live lock and minimal distances. The set of permissible routes P_p for all traffic flows τ_i then becomes the set of all minimal paths from *source*$_i$ to *dest*$_i$. We can generate these paths by only allowing a number of hops in each direction (*S*, *E*, *W*, *N*) and generating all possible combinations of these hops. The Manhattan distance between two node coordinates (x_1, y_1) and (x_2, y_2) can be calculated as $d_x = (x_1 - x_2)$, $d_y = (y_1 - y_2)$, $d_M = | d_x + d_y |$ (12), then if the number hops of a generated route is equal to the Manhattan distance for a 2D mesh the route is minimal. The sign of the terms d_x and d_y in the above equation determines the orientation of the hops of the route according to the coordinate system adopted; the route generation algorithm used here is based on the observation that d_x hops on the *x* axis and d_y hops on the *y* axis with the correct orientation would always produce minimal routes reaching the correct destination no matter the order of the hops.

The authors in [85] prove that this algorithm has the following complexity $O(m^2r^2d)$, where *m* is the number of communications/ traffic flows, *r* is the number of alternative routes considered per flow and *d* is the distance (no. of hops in our case) of the longest route. For traffic flows that need to traverse a long distance (minimal distance of *n* hops) many different permutations are possible (!*n*), even though many of the hops in a 2D system will be towards the same direction so their relative ordering will not count as a permutation, this amount corresponds to the *r* factor whose square bounds the execution time of the algorithm so it is obvious that this factor significantly affects the performance; by restricting the number of possible paths considered for any traffic flow to a certain constant or to a proportion of all possible paths (e.g. 1/3) the algorithm should speed up at the expense of solution quality.

This algorithm can also be integrated into the genetic mapping algorithm in which case whenever a mapping is evaluated instead of using a simple XY routing algorithm to generate the traffic flow routes the minimum contention route generation algorithm is used.

5.3.3 Priority Assignment

The communication model we adopt in this work is based on a fixed priority scheme; the schedulability of such a system apart from interference is also based on the relevant priorities. Priority assignment policies have been well studied especially in the case of fixed priority single processor based systems; various priority assignment policies have been proposed beginning from the seminal work of Liu and Layland [87] to more elaborate approaches such as in [89], [90] where optimal algorithms for this problem are proposed. Considering real-time communications the authors of [93] among various other approaches proposed the *Least Laxity First* assignment policy, the laxity of a message *i* is defined as:

$$L_x i = deadline_{\mathrm{i}} - network\ latency_{\mathrm{i}} \quad (13)$$

which intuitively is proportional to the amount of interference that a message may tolerate as the latency is calculated at the relative packet position; based on this scheme the message with the least laxity gets assigned the highest priority. In [94] an improvement to this scheme was proposed were the laxity of a message is again considered as defined in [93] only now the metric becomes more elaborate as apart from the absolute network latency, calculated from the number of remaining hops that a message has to traverse, the number of hops is taken into account as well; this is based on the fact that a message travelling a large number of hops may experience more interference.

An important difference however is that in these works the priority assignment policies are used for runtime readjustment (dynamic priority system) of each separate message priority according to its laxity at any given instant while the analysis we are considering can only be applied in a fixed priority system. In [95] Shi and Burns give a branch and bound algorithm that solves the priority assignment problem of traffic flows in a fixed priority context and is shown to be optimal, that is according to the analysis presented in [33] if a set of interfering communication flows is feasible the algorithm will assign the relevant priorities so that all the traffic flows in the set will be schedulable. Due to the longer running time of a dedicated heuristic which in combination with the genetic algorithm and the route selection algorithm would be too slow we chose to implement a simpler approach similar to the ones proposed in [94], [95]. Since we are interested in a fixed priority scheme and considering the static mapping approach in which the communication flows get assigned a fixed path we could adopt any of the above schemes to a become fixed a priority assignment policy. For our static system the relative laxity will be the same for all instances of a traffic flow so will be the network latency and the number of hops it will have to traverse, so we can use the static laxity of a flow to assign a fixed priority similar to the *Rate Monotonic* assignment policy. This assignment will have to take place after the traffic flows are mapped and the network latency for any given flow can be calculated based on the distance it will have to traverse. We can also take into consideration more information that is available after the tasks are mapped such as the release jitter of the flow and formulate a metric as:

$$L_x i = (deadline_i - network\ latency_i - release\ Jitter) / No.\ of\ Hops \quad (14)$$

We could refer to this metric as *weighted laxity* for the purposes of this work. The traffic flow with the least weighted laxity would get assigned the highest priority.

5.4 A Heuristic Mapping Algorithm

In addition to the genetic algorithm proposed earlier we propose a constructive heuristic that tries to take advantage of some characteristics of a 2D mesh based NoC and produce sufficiently good solutions to the mapping problem. An important characteristic of static mapping algorithms is their ability to evaluate a multitude of different solutions and hence explore a large solution space; further more in such a context it is possible to decouple communication and computation and efficiently solve the problems of task partitioning, mapping and communication routing separately. Dynamic mapping algorithms on the other hand usually need to produce a solution to a smaller instance of the problem i.e. for a partial mapping and an incoming task find an optimal placement for the incoming task, but the aspects of the problem such as computation and communication have to be considered together.

The motivation to develop a constructive heuristic algorithm is that it can be versatile enough to be used in different contexts. The logic that the algorithm uses to make the placement at each step can be adopted to suit both static and dynamic versions of the algorithm. In the case of static mapping adding a backtracking step can make the algorithm similar to an exhaustive search of the solution space

Here we propose a constructive heuristic that works by iteratively mapping communicating tasks on the NoC platform, in each iteration the algorithm has to make a decision about where to place a single task and the way this decision is made forms the core of this algorithm. The algorithm is inspired by various approaches proposed for dynamic mapping; the basis is similar to the Dependencies Neighbour [54] and Nearest Neighbour [48] heuristics where the search starts from the requesting task's node neighbours outwards in a spiral fashion. Allowing the spiral search to always have the same orientation we seek to take advantage of a. The bi-directionality of the NoC links and b. Achieve a spatial arrangement that “packs” the resource usage close together so that the resource fragmentation is minimised (leading to less interference) for any incoming requests similar to the packing strategy used in [53].

Here we will only describe the algorithm version for static mapping as it was the context in which it was developed and tested. The algorithm works by iterating over the task set of the application, at each iteration it finds the task with the highest connectivity (in terms of outgoing communications) that is not yet mapped; for this task the algorithm selects a *pivot core* which is the initial core around which the search will evolve, next the algorithm tries to map the task on the pivot core if it is not possible the search for another core starts spirally outwards the pivot core in a specific orientation up to a certain number of hops in distance.

If a highly communicating task is mapped next all the tasks it communicates with are put in a list and are mapped next. From this communicating list again the task with the highest connectivity is selected and the algorithm tries to map it as closely as possible to the parent task following again the specific search pattern.

The algorithm is described by three main steps:

- The selection of tasks based on their connectivity.
- The selection of tasks that communicate with previously placed tasks
- The search for a placement of the tasks in a spiral fashion

The algorithm performs the search spirally with the following orientation East, North, West, and South.

```
1  For an application tasks graph G = ($T$, $C$) Sort the tasks of the application by their
   connectivity (High to low) .
2  For all tasks $t_i$ in the sorted list
3          if $t_i$ is not already mapped
4             Place task on pivot core
5          else continue to next task
6               for all tasks $t_j : \exists\ \tau \in C \mid source(\tau) = t_i \wedge dest(\tau) = t_j$
7                    Look for a placement for the task spirally outwards from $core(t_i)$.
```

Listing 3. Pseudo-code for the constructive heuristic mapping algorithm

Two important characteristics of the algorithm are a. the criteria based on which a task “fits” a core and b. the selection of a pivot core at each step. This search technique offers two advantages, first the search tries to place communicating tasks as close to each other as possible minimising the communication distance and the therefore the possible interference on the network links, next because the probability of a selection for a placement becomes higher for a specific direction this heuristic imitates the effect of the packing strategy in figure 10 as it packs tasks and traffic towards a specific direction.

We used the schedulability analysis of a task set on a single core as the criteria for task fitness on a placement; at each such step the priorities of a task set on a core plus the incoming task where re-assigned according to RM and a schedulability test was then carried out (response time analysis) if the set was schedulable the placement was complete otherwise the priorities of the tasks where restored as previously and the search would continue. The selection of a pivot core at each step could be either random or guided, for the guided case the same pivot core could be used for all steps so that the placement would be more packed towards some point on the platform or the least utilized core on the platform could used so that the distribution on the platform was more even, a combination of the two schemes could use the least utilized core closest to the initial starting core. A shortcoming of the existing implementation is that it does not consider the case where the pivot cores reside at the edge of the NoC platform separately; in this case the search for the placement of the subsequent tasks would be possible only towards a specific direction potentially disrupting the algorithms ability to achieve its goals. In this situation the bi-directionality of the NoC links could be taken advantage of as the search could switch to the opposite orientation so that the communication between subsequently mapped tasks would use less utilized links.

The algorithm when tested did not have some coherent behavior pattern so no results are displayed, further improvements and additions to this algorithm could be part of future research on mapping heuristics. Below we list some observations that were made during the testing of this algorithm.

The algorithm was rarely able to provide a solution with all tasks and traffic flows schedulable but the solutions produced had fewer unschedulable flows compared to a random mapping of the application; a backtracking step was next added in the algorithm that would make it behave more like an exhaustive search of the solution space, in this case in some instances it was able to find a solution much faster than the genetic algorithm and in other cases the genetic algorithm would find a solution and the heuristic would not do so in reasonable time, a final observation is that the ability of the algorithm to find a solution fast would depend on the initial selection of the pivot core.

5.5 Experimental Results

In order to validate the proposed approach we developed a framework that implements both application and platform models described earlier, as well as the schedulability analysis from [33]. Unlike most of the related work on task mapping our approach did not require cycle accurate simulation as it is based on an analytical model, simulation nonetheless could provide further insight to the quality of the results produced. The software framework provided us with a NoC system model where an application can be mapped onto. Such model describes the topology of the NoC using collections of PEs and links on a 2D coordinate system interconnected to each other. It also implements a routing function. This system model provides all the information needed to carry out the analysis described in the previous sections.

5.5.1 Input Application Models

Two main application models were used for this case study. The first one is the controller of an autonomous vehicle, which is a good example of a highly parallel application running on an embedded system. This application was introduced in [62] and it is very communication intensive, because of the successive processing of different video streams for stereo photogrammetry and visual odometry. The subsystems of the application are those of video processing, navigation and stability control. In total it comprises of 33 tasks and 38 inter-task fixed-priority communication flows. Each traffic flow has all the attributes mentioned in the previous sections such as period and deadlines. In addition all the traffic flows have fixed-size payloads, ranging from 7kbits to 525kbits. A second application was used which was completely synthesized. It was composed of more tasks and traffic flows hence it would be harder to find a solution for a same size NoC. The synthetic application has 50 tasks and 40 inter-task communication flows again with all traffic flows having fixed-size payloads, ranging from 42kbits to 2400kbits, and shorter periods so that this application is much more communication intense than the previous one.

Next in order to test the algorithms using a wider variety of application instances various random applications were generated. For the generation of random applications a method was used which would bound some of the application's variables and components; these applications were generated according to the model described.

The type of generated applications most suitable for testing would have an overall computation utilisation less than that which the platform may offer; even though this does not guarantee that a schedulable placement of tasks exists it is a coarse measure indicating that the application may 'fit' the platform and also an attempt to make the effect of unschedulable computation on the communication as less frequent as possible as we are mostly interested in the communication aspect. Next following the same reasoning no tasks were generated that did not communicate with ingoing or outgoing traffic flows.

The period and WCET of each task where generated according to an overall task set utilization for a fixed number of tasks. For a number of tasks *n* and an overall utilization U we generated the task set with random task utilization values u_i so that $\sum_{i=1}^{n} u_i \leq U$ (15). The period of each task in each application instance is generated randomly within an interval. Next each task deadline is taken as equal to its period and finally the execution time of the task instead of being generated randomly in the open interval (0, *period*) is defined by the task utilization value and the period as $C_i = u_i P_i$.

Some important properties that a task set generation algorithm must possess have been identified in various works [96], [97], [98], these properties are those of being *independent* and *unbiased*. Independence means that different parameters of the task set can be varied independently while others remain constant e.g. utilization and number of tasks; next being unbiased means that the task sets generated have a uniform distribution within the set of all possible suitable task sets. In addition to those properties the algorithm should generate such task sets reasonably fast. For a given number of tasks *n* and an overall utilization various algorithms have been examined that have the properties mentioned above however the problem for U > 1 becomes more complicated because of the added restriction that each task utilization has to be valid $u_i \leq 1$. In [97] along with a comprehensive survey of different approaches an algorithm is proposed that can generate such task sets efficiently for any utilization value, for detailed explanation of this algorithm we refer the reader to [99].

Having generated such a task set a set of traffic flows should be generated accordingly, the difference with generating traffic flows is that the traffic flows will inherit almost every attribute for the tasks that generate them except for their payload. The inherited attributes will be generated uniformly at random for tasks so we could assume that this property is maintained for the traffic flows as well, the remaining attribute of payload can be generated randomly within an interval for all flows but with the restriction that for each flow the relevant utilization ratio *C/P* with network latency *C* measured over one link does not exceed 1. Regarding the amount of traffic that will be generated the number of traffic flows in any case should be lower bound by the amount necessary for every task to be communicating i.e. be the source of a traffic flow which means that for *n* tasks and *m* traffic flows $m \geq n$. Finally the destination task of each traffic flow will be randomly chosen.

There is no apparent relation between the schedulability of traffic flows and their relevant utilization ratio of *C/P* over one or more links such as a feasibility test because the distribution of the traffic on the mesh is an equally important factor and the effect of indirect interference makes this test on a link impossible since schedulability does not only depend on traffic flows that share a single resource; the utilization of a link *i* which is used by *n* traffic flows can be defined as:

$$U_{link} = \sum_{j=1}^{n} C_j / T_j \quad (16)$$

The average (mean) and maximum link utilization can also be defined using (16) over all the *m* links of the network.

$$U_{link\ \text{Avg.}} = \sum_{i=1}^{m} U_i / m \quad (17)$$

Using those metrics we would like to measure the spatial distribution of the overall traffic over the network links and relate it to schedulability of traffic. In order to measure the spatial distribution of traffic the utilisation of the network links will be used, then calculating the mean absolute deviation of the measured values will provide a good measure of dispersion. The mean absolute deviation (MAD) is measured as:

$$MAD = \frac{1}{m} \sum_{i=m}^{m} \left| U_i - U_{link\ \text{Avg.}} \right| \quad (18)$$

Small values of MAD mean that all the network links have similar utilisation values close to the mean hence we can assume that the traffic is evenly distributed on the network.

5.5.2 Experimental Results using Custom Benchmarks

The figures below show the results obtained from running the baseline version of the proposed genetic algorithm. The results show the minimum (best) score achieved in every iteration of the algorithm (generation). The solutions tend to get closer to optimal as the algorithm runs. We tested the two applications mentioned above for a population size of 100 solutions and various NoC sizes. The algorithm tried to map the autonomous vehicle and the synthetic applications on a 5x5, 4x4, 4x3 and a 3x3 NoCs.

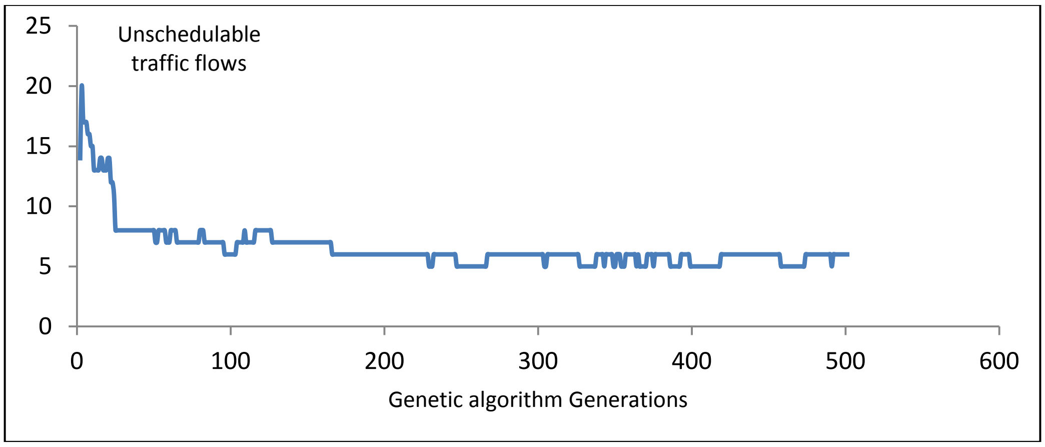


Figure 5.2 (i). Mapping of Autonomous Vehicle application onto a NoC platform with 3x3 topology. Vertical axis shows the number of unschedulable traffic flows of the best mapping of each generation of the genetic algorithm execution (only the best of 15 executions is shown).

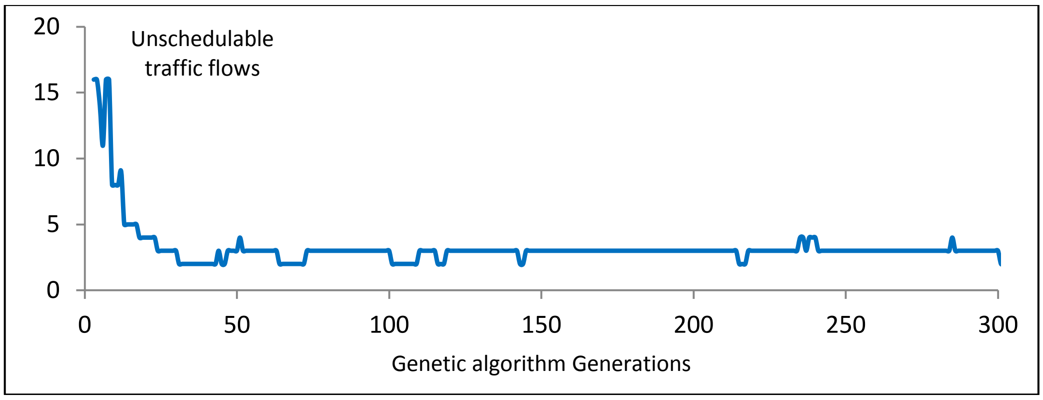


Figure 5.2 (ii). Mapping of Autonomous Vehicle application onto a NoC platform with 3x4 topology.

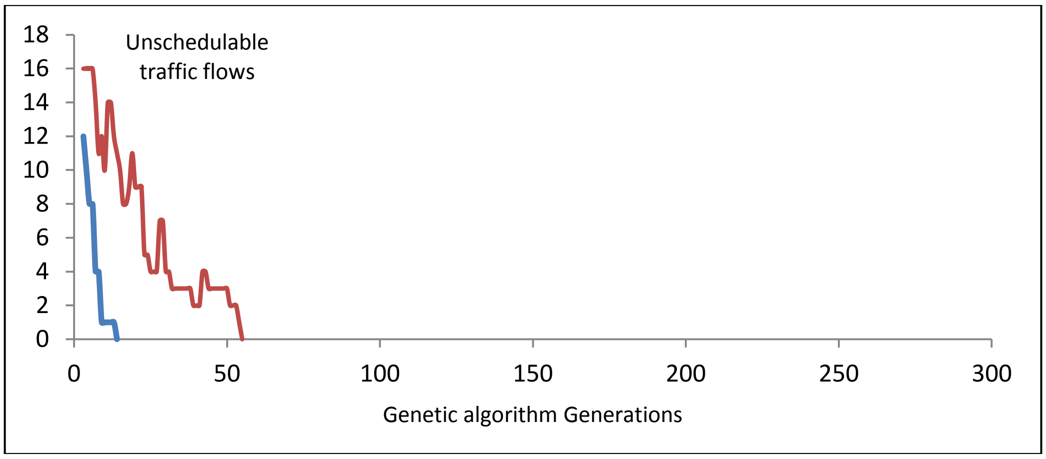


Figure 5.3 (i). Mapping of Autonomous Vehicle application onto a NoC platform with 4x4 topology. (blue line shows the fastest of 15 executions that found a solution, red line shows the slowest).

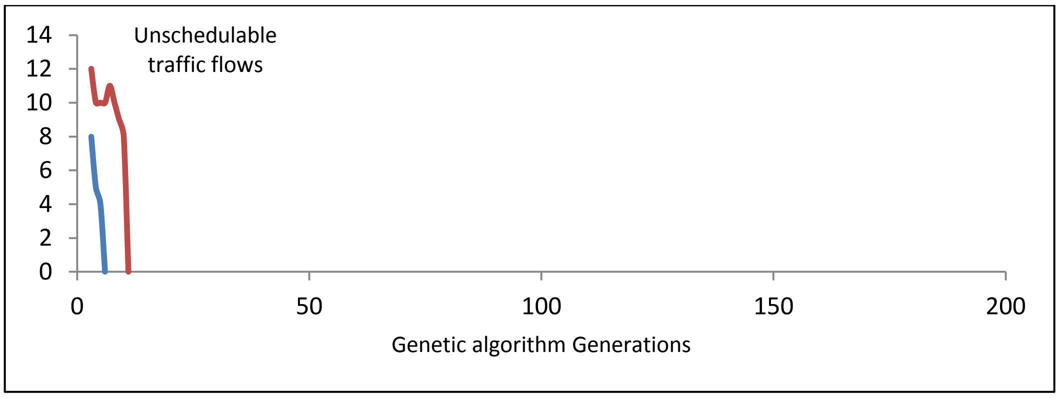


Figure 5.3 (ii). Mapping of Autonomous Vehicle application onto a NoC platform with 4x5 topology.

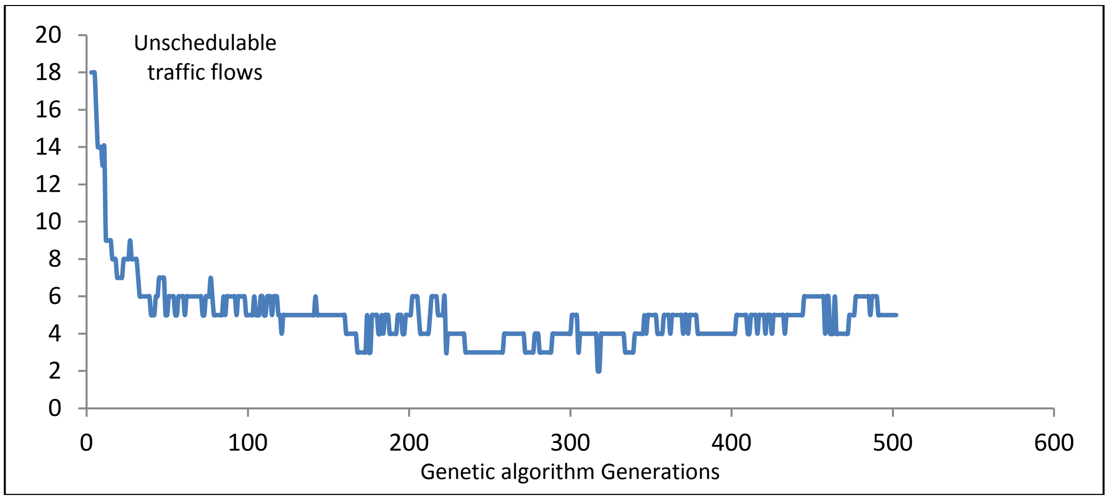


Figure 5.4 (i). Mapping of Synthetic application onto a NoC platform with a 4x4 topology.

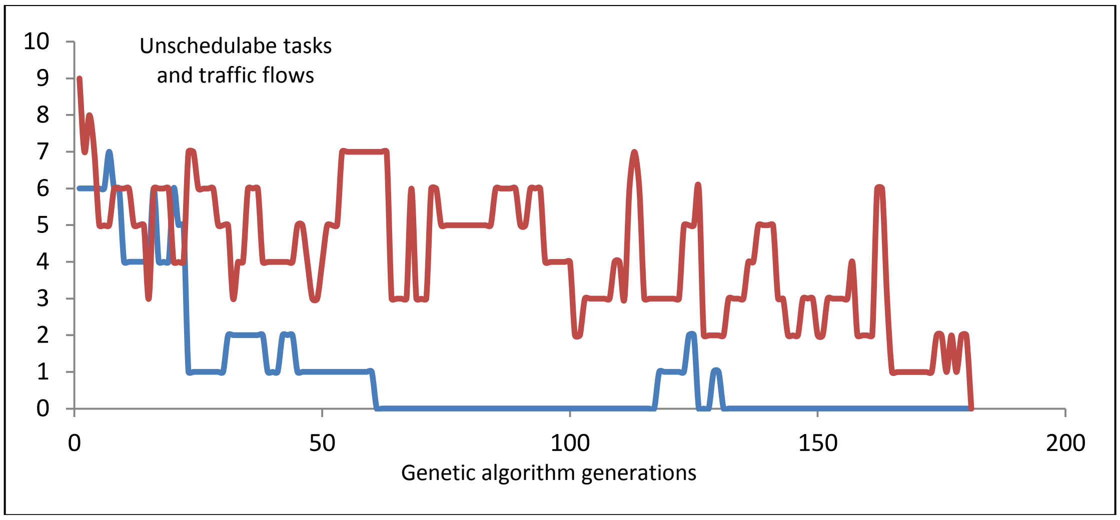


Figure 5.4 (ii). Mapping of Synthetic application onto a NoC platform with a 5x4 topology.
The blue line shows the number of unschedulable flows, the red line shows the number of unschedulable tasks. The results displayed come from the fastest run from 15 consecutive runs.

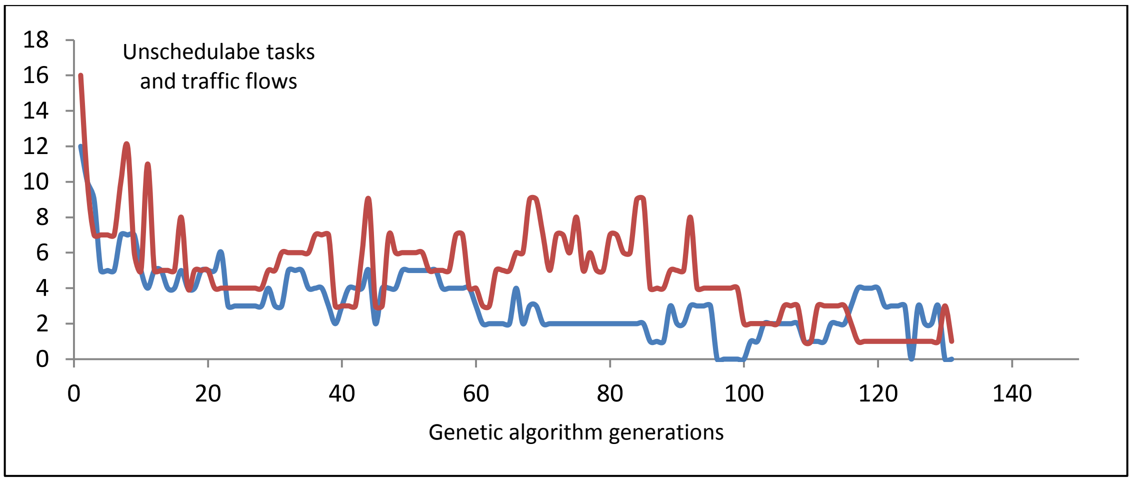


Figure 5.4 (iii). Mapping of Synthetic application onto a NoC platform with a 5x5 topology.
The blue line shows the number of unschedulable flows, the red line shows the number of unschedulable tasks. The results displayed come from the fastest run from 15 consecutive runs.

The algorithm was run 15 times for each scenario and to a maximum of 500 generations with 100 solutions per generation. This amount of iterations would allow the algorithm to visit enough solutions. The algorithm would also run for 15 times for each scenario so as to make the overall solution pool is independent of the initial random population's quality.

In figure 13 the algorithm attempts to map the autonomous vehicle application on a 3x3 and 4x3 NoC, It can be seen that in the case of the 3x3 NoC the algorithm cannot find a suitable mapping for the application (a solution with 0 deadline misses) a fact that is expected as the platform gets smaller because the interference between traffic flows increases and the task set does not have enough resource to execute. Next for the 4x3 platform the algorithm was able to find a solution only twice during 15 runs.

Next in figure 14 as the platform size increases to a 4x4 and 5x5 NoC and more resources become available the algorithm is able to converge to a solution much quicker; each figure shows both the quickest and the slowest run of the algorithm for each case.

In figure 15 we can see that the synthesized application which was more communication-intensive could only be mapped on 4x5 and a 5x5 platforms. This synthesized application is different to the previous one in the fact that it also has tasks that only receive communication flows a fact which leads to a situation where there may no unschedulable communication but the computation can be unschedulable. This is shown in figure 15 (ii) where the blue line shows the number of unschedulable flows and the red line shows the number of unschedulable tasks.

Motivated by this fact we introduced a more guided mutation operator in our genetic algorithm which finds the least and most utilised cores and transfers a task at random from the most to the least utilised core.

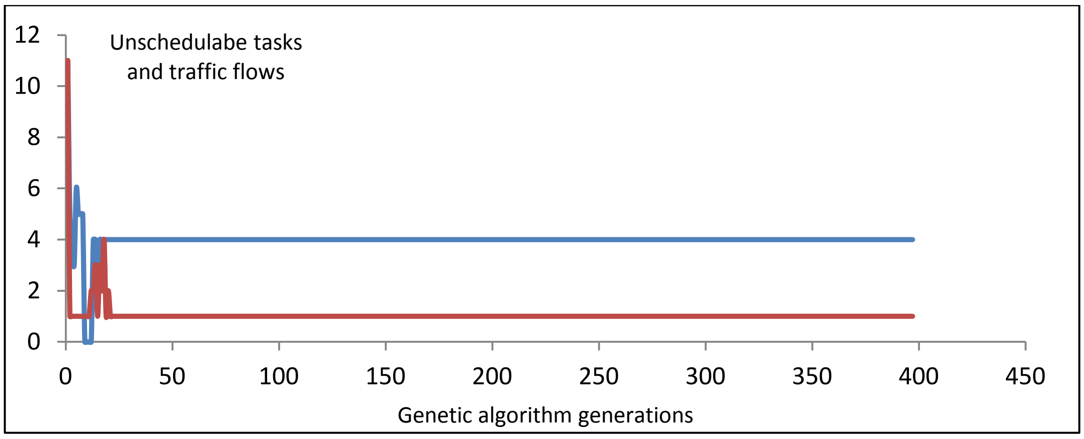


Figure 5.5. Genetic mapping of Synthetic application onto a NoC platform of size 5x5 using the guided mutation operator. The blue line shows the number of unschedulable flows, the red line shows the number of unschedulable tasks. This is a run of the algorithm chosen at random from 5 consecutive runs.

In figure 16 the problem with the use of the specific mutation operator is evident, although the algorithm converges faster to solutions with few task deadline misses the algorithm gets trapped in a local minima; this happens because the mutation operator is no longer random. In order to mitigate this effect we can combine the two different mutation operators in the same algorithm.

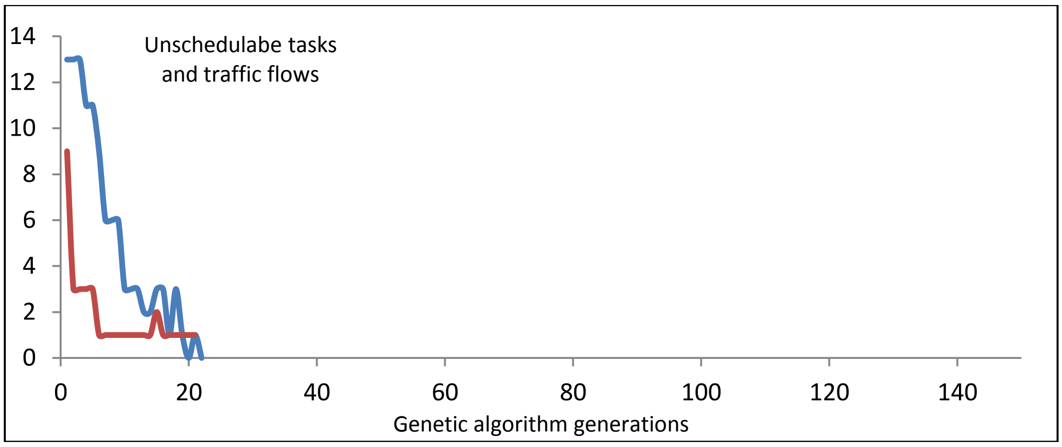
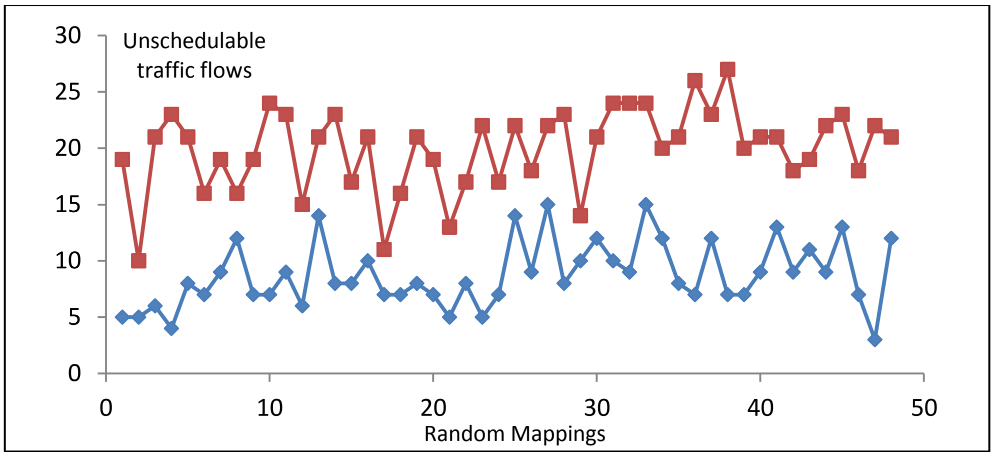


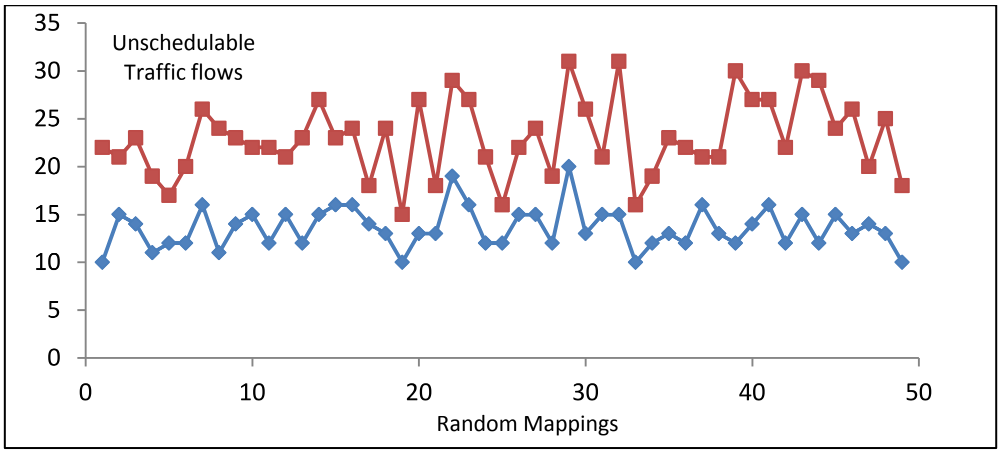
Figure 5.6. Genetic mapping of Synthetic application onto a NoC platform of size 5x5 using a combination of both guided and random mutation operators. The blue line shows the number of unschedulable flows, the red line shows the number of unschedulable tasks. This is the fastest run of the algorithm chosen from 15 consecutive runs.

Figure 17 shows the combination of the two mutation operators which quickly converges to a schedulable mapping mainly with respect to computation in comparison to figure 15 (ii) and 16.

Next the minimum contention route generation algorithm was tested separately in order to see its effects on the schedulability of the traffic flows. For the two applications described above a number of randomly generated mappings were generated and evaluated regarding only the number of unschedulable traffic flows; the traffic flows of the applications in each instance were routed by both the XY routing algorithm and next by the minimum contention algorithm. As figure 18 below shows over 50 random mappings for each application the minimum contention algorithm when compared to XY routing would always produce routes that lead to less or the same amount of unschedulable traffic flows. The idea behind using the synthesized application on a 4x4 NoC to test this algorithm is that since the genetic algorithm was not able to find a solution with schedulable flows for this combination a lot of contention must exist between traffic flows in this scenario.



(i)

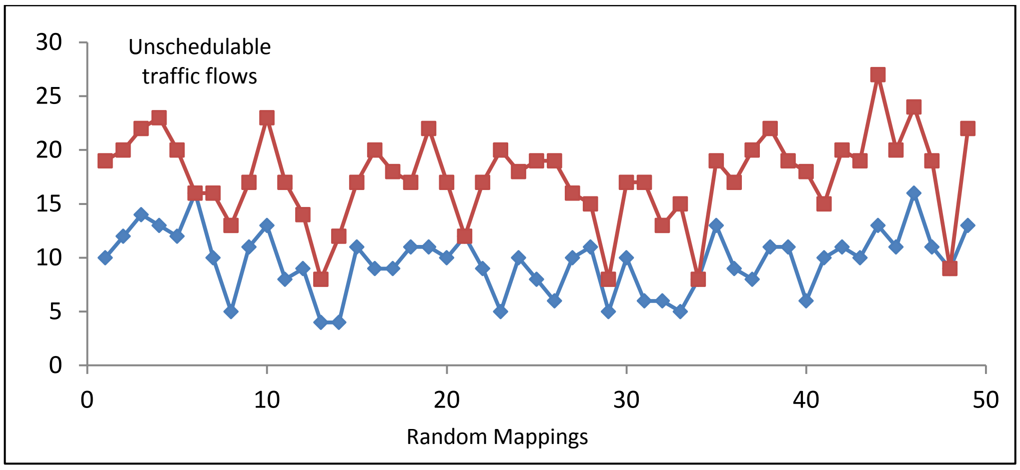


(ii)

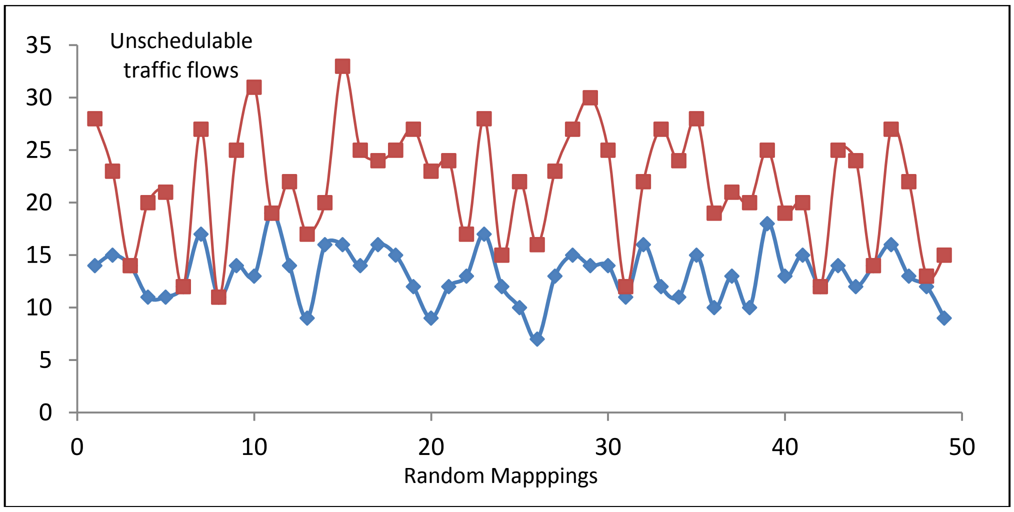
Figure 5.7. Random mapping of the (i) Autonomous Vehicle and (ii) the Synthetic applications onto a NoC platform of size 4x4, the results display the number of unschedulable traffic flows - Y axis for each random mapping - X axis. The blue markers show the number of unschedulable flows when the XY routing algorithm is used, the red markers show the number of unschedulable flows when the minimum contention routing algorithm is used.

Another important aspect of this algorithm is its running time which depends on the number of alternative routes considered for all traffic flows (R_{all}). Empirically through experimentation it was noticed that the algorithm would run acceptably fast on our Java framework as long as the size of R_{all} would be less than 1000 routes so the algorithm was sped up by imposing this additional constraint and possibly sacrificing the produced solutions quality.

Next the proposed priority assignment policy was tested in the same manner in order to see its improvement on traffic flow schedulability compared to the default priority assignment policy based on the flows rate which is the default policy used; the use of the default policy stems from the fact that the traffic flows inherit their attributes (period and priority) from the tasks that generate them, the task set of the application is assigned priorities according to a rate monotonic policy and since the traffic flows have the same periods and inherit the same priority indirectly the rate monotonic scheme is applied on them as well. The proposed approach calculates the metric in equation (14) and assigns a priority accordingly.



(i)



(ii)

Figure 5.8. Random mapping of the (i) Autonomous Vehicle and (ii) the Synthetic applications onto a NoC platform of size 4x4, the results display the number of unschedulable traffic flows - Y axis for each random mapping - X axis. The blue markers show the number of unschedulable flows when the proposed priority assignment algorithm is used, the red markers show the number of unschedulable traffic flows when the default priority assignment policy is used.

Figure 19 above shows again over 50 random mappings for each application the routing algorithm used was XY routing, when the routes of traffic flows were generated the different priority assignment algorithms were used for same mapping and routing; the proposed priority assignment algorithm for those applications on a 4x4 NoC would always produce routes that lead to less or the same amount of unschedulable traffic flows.

Next the genetic mapping algorithm was tested together with using the minimum contention route generation and the priority assignment algorithms in order to see its overall improvement. As mentioned earlier this route generation algorithm has a long running time so in order to apply it to each solution of a generation we had to cap the number of all possible routes that route generation the algorithm would consider per mapping solution.

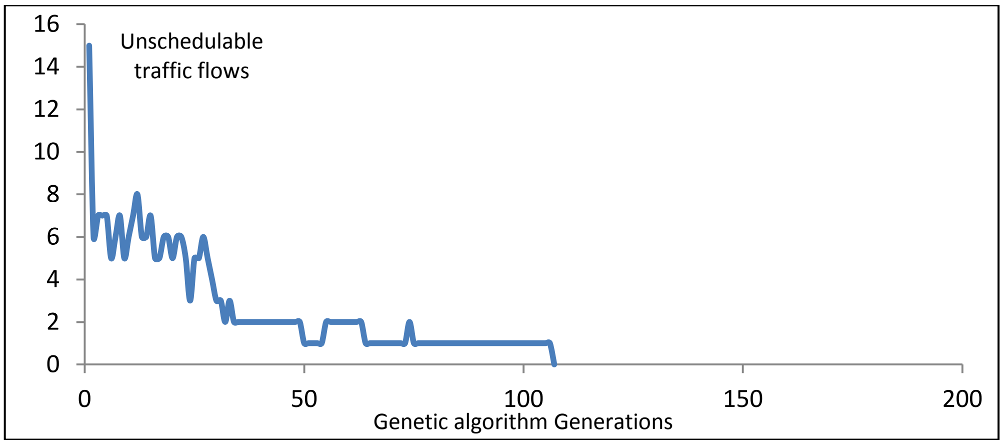


Figure 5.9. Mapping of Synthetic application onto a NoC platform of size 4x4 using the genetic algorithm in combination with the minimum contention route generator. Results display the fastest run over 15 consecutive runs. The blue line shows the number of unschedulable flows.

In comparison to figure 15 (i) we can see in figure 20 (i) that the genetic algorithm combined with the minimum contention route generator was able to find a schedulable (with regards to communication) solution in a few generations while the basic version of the genetic algorithm was unable to do so over 400 generations, however the running time of the combined algorithm was considerably higher per generation so it is hard to say at this stage if there was some absolute performance increase.

5.5.3 Experimental Results using Randomly Generated Applications

In order to test our algorithm over a wider variety of applications we generated random applications according to the method described in the previous section. The main motivation here is to test the different versions of the genetic algorithm and ensure that the additional features suggested do improve the algorithm's performance and ability to find solutions. Due to the fact that it is possible to generate applications with specified characteristics we chose the application to be the main variable instead of the available resources (platform size) as done in the previous experiments, the testing is thus carried out only for 4x4 and a 5x5 NoC platforms.

In this set of experiments the following variations of the genetic algorithm in combination with the different proposed improvements where evaluated:

- **GA**: baseline genetic algorithm.
- **GM**: genetic algorithm with guided mutation.
- **GR**: genetic algorithm with minimum contention route generation per solution.
- **GP**: genetic algorithm with the proposed priority assignment policy per solution.
- **GRP**: genetic algorithm with both the above additions.
- **GPRM**: genetic algorithm with all the above additions combined together.

The following twelve task sets were generated for each platform size separately:

Task sets for 4x4 and 5x5 platforms				
No. of tasks	Task set utilization			
30	U = m/3	U = m/2	U = 2/3 m	U = 3/4 m
40	U = m/3	U = m/2	U = 2/3 m	U = 3/4 m
50	U = m/3	U = m/2	U = 2/3 m	U = 3/4 m

The task set utilization U would increase in relation to the overall resources available to a platform of *m* processors (actual values used were rounded to the closest larger integer).

For each of the randomly generated applications above four more variations were generated with the following amount of traffic flows: *n*, *n* + 5, *n* + 10, *n* + 20 where *n* is the number of tasks.

The period of each task in seconds was generated randomly within the interval (0, 1); the payload of each traffic flow would have to be within some meaningful value, in the same way that a task's utilisation has to be bounded by one the relevant utilisation ratio for a traffic flow has to be less than one measured over the basic network resource that it may require which is that of one link; for every traffic flow $C/P \leq 1$, with C measured over one network link, in which case C can be calculated by equation (5) substituting 1 for H, so for all traffic flows random payload values would be generated within the interval that satisfies this inequality. Using this constraint for the payload of traffic flows is still prohibitive as it means that a traffic flow with a payload value close to this bound would be schedulable only on routes with length of at most one hop, for any longer routes the ratio C/P would increase above one as the basic latency would be greater than the period making the traffic flow unschedulable even without the presence of interference from other traffic flows; because of this fact the upper bound on the payload is further reduced so that the ratio C/P is less or equal to one but instead measured over the largest possible minimal route on the network (e.g. 8 hops for a 4x4 mesh).

In the results below we demonstrate how the various versions of the algorithm performed for all the different input cases. The quantity measured is the number of generations that each version of the algorithm had to run in order to find a solution over one random application instance.

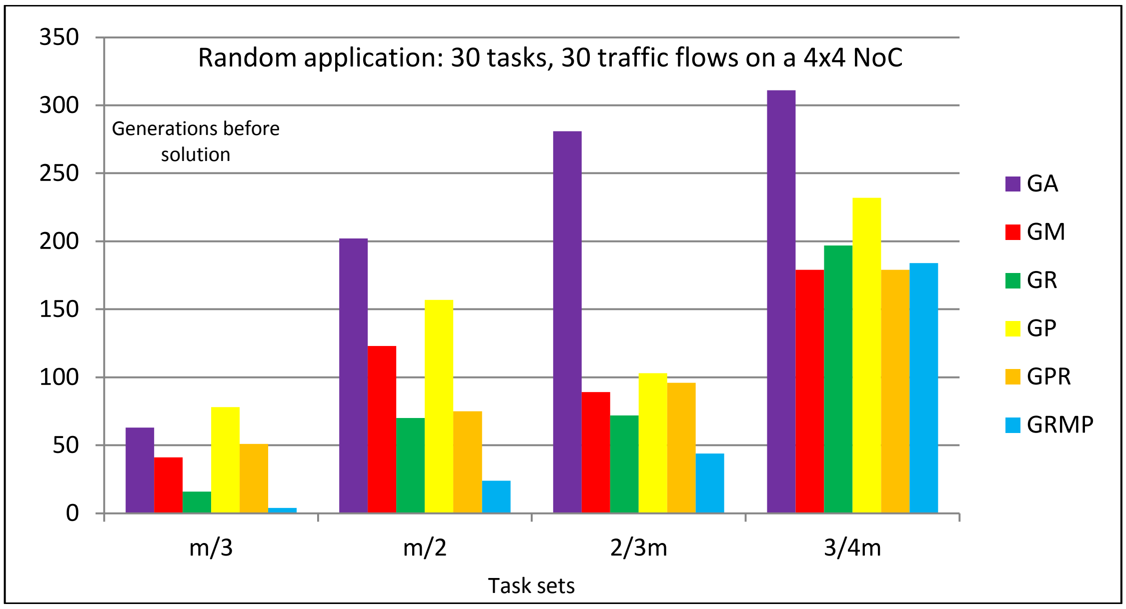


Figure 5.10 (i). Mapping of random applications onto a NoC platform of size 4x4 using the different versions of the genetic algorithm. The application is composed of 30 tasks and 30 traffic flows.

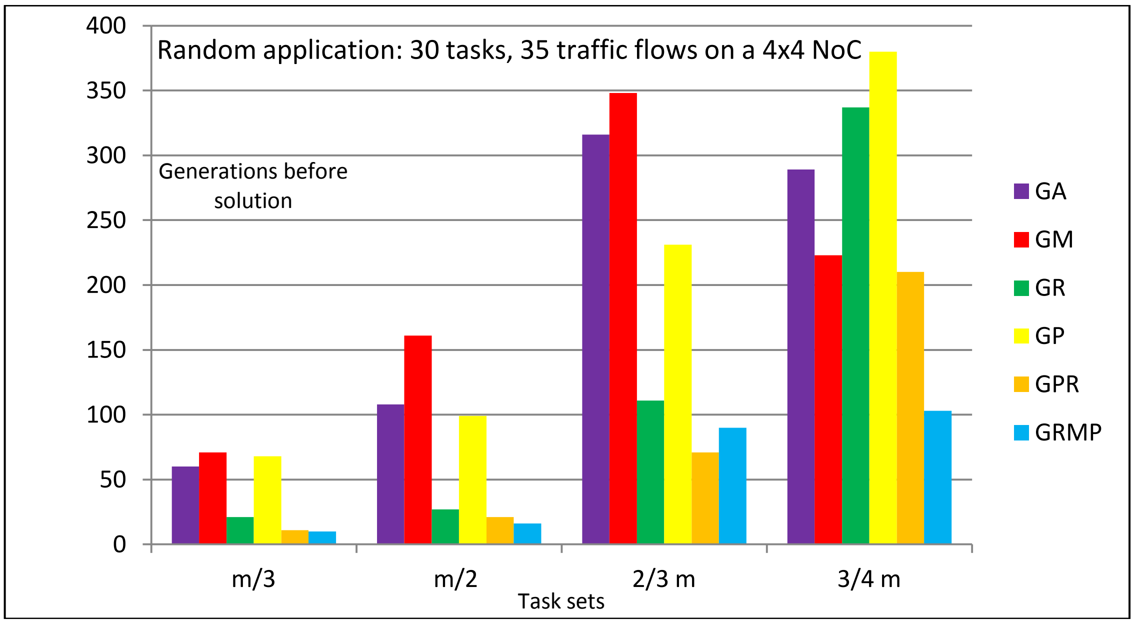


Figure 5.10 (ii). Mapping of random applications onto a NoC platform of size 4x4 using the different versions of the genetic algorithm. The application is composed 30 tasks and 35 traffic flows.

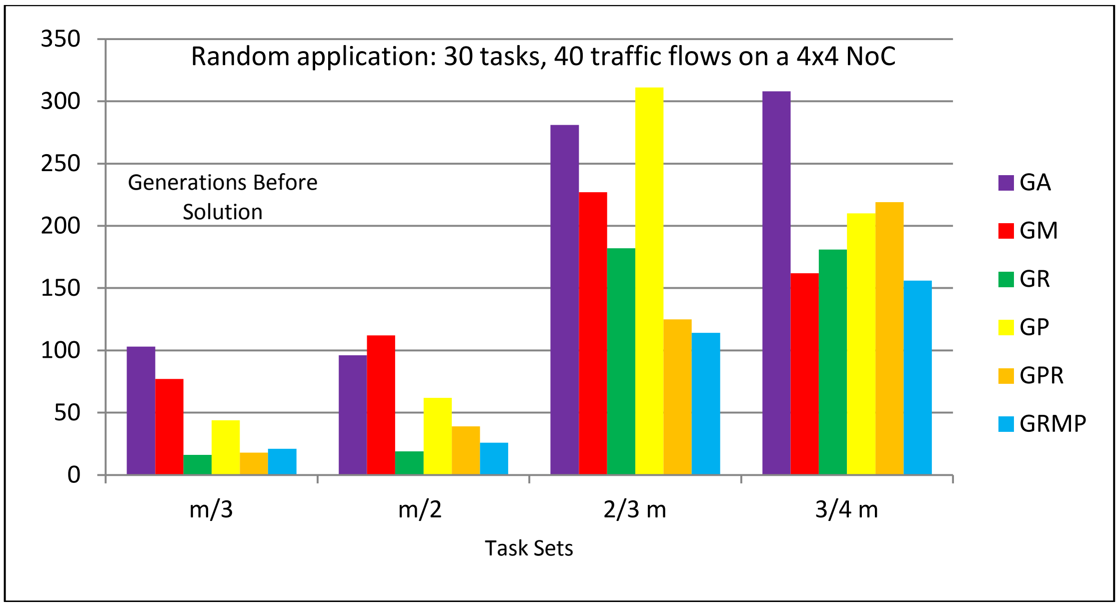


Figure 5.10 (iii). Mapping of random applications onto a NoC platform of size 4x4 using the different versions of the genetic algorithm. The application is composed 30 tasks and 40 traffic flows.

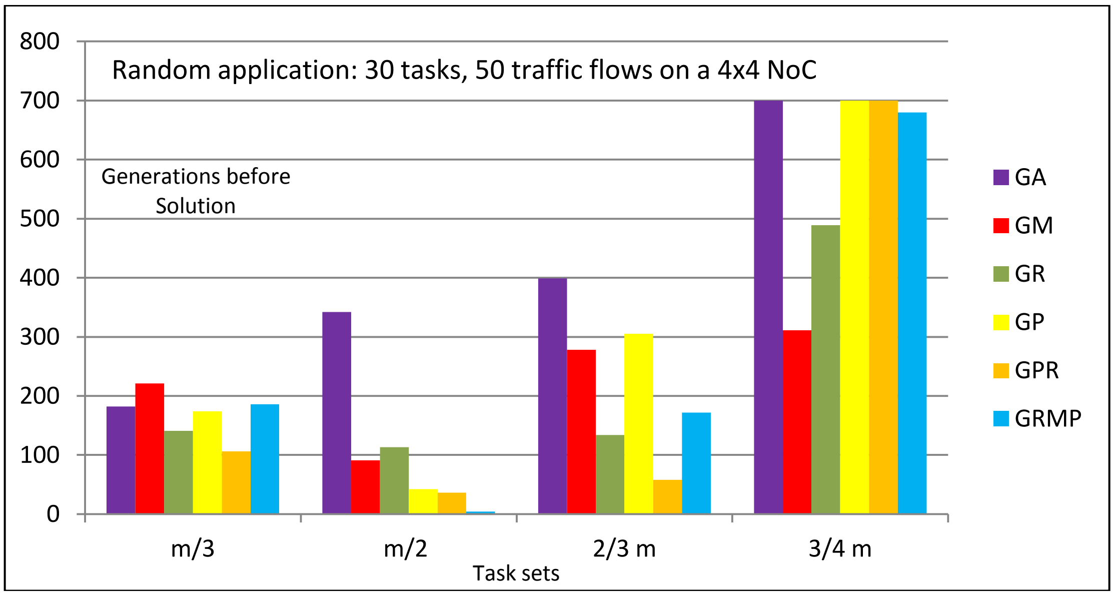


Figure 5.10 (iv). Mapping of random applications onto a NoC platform of size 4x4 using the different versions of the genetic algorithm. The application is composed 30 tasks and 45 traffic flows.

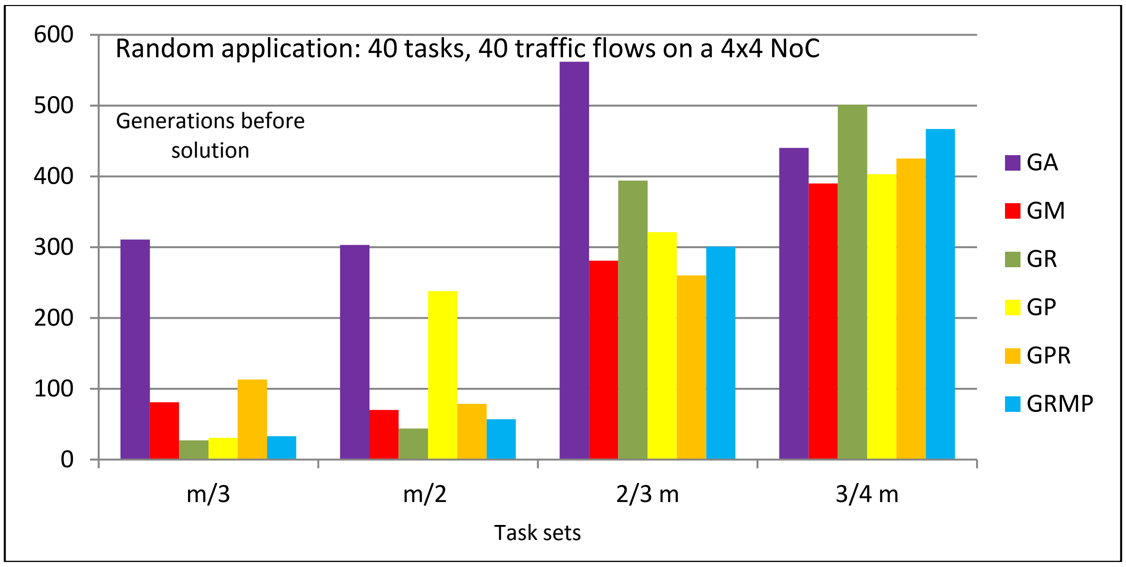


Figure 5.11 (i). Mapping of random applications onto a NoC platform of size 4x4 using the different versions of the genetic algorithm. The application is composed of 40 tasks and 40 traffic flows.

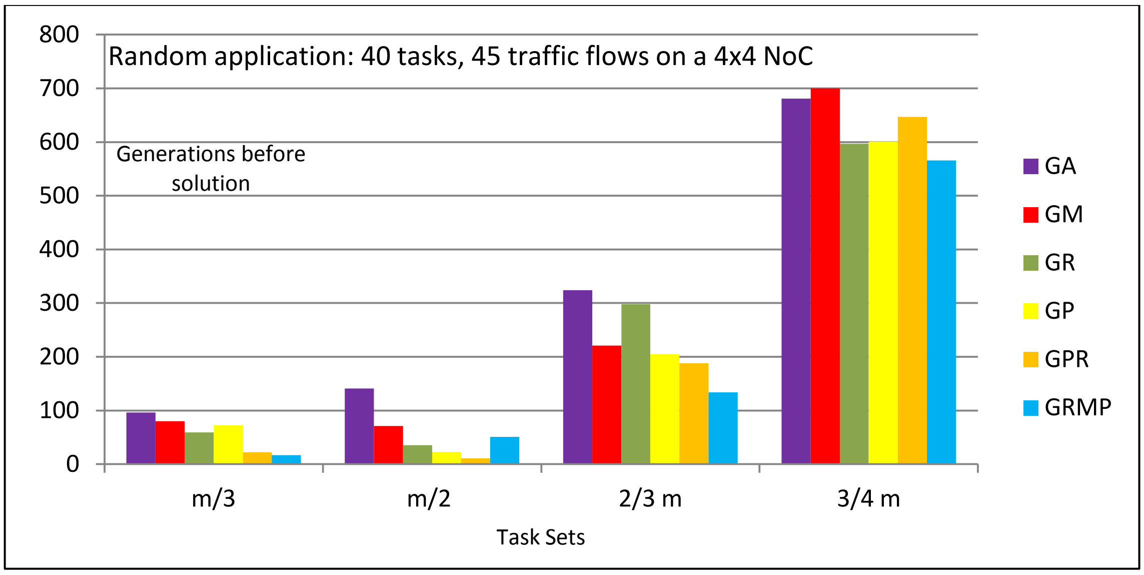


Figure 5.11 (ii). Mapping of random applications onto a NoC platform of size 4x4 using the different versions of the genetic algorithm. The application is composed of 40 tasks and 45 traffic flows.

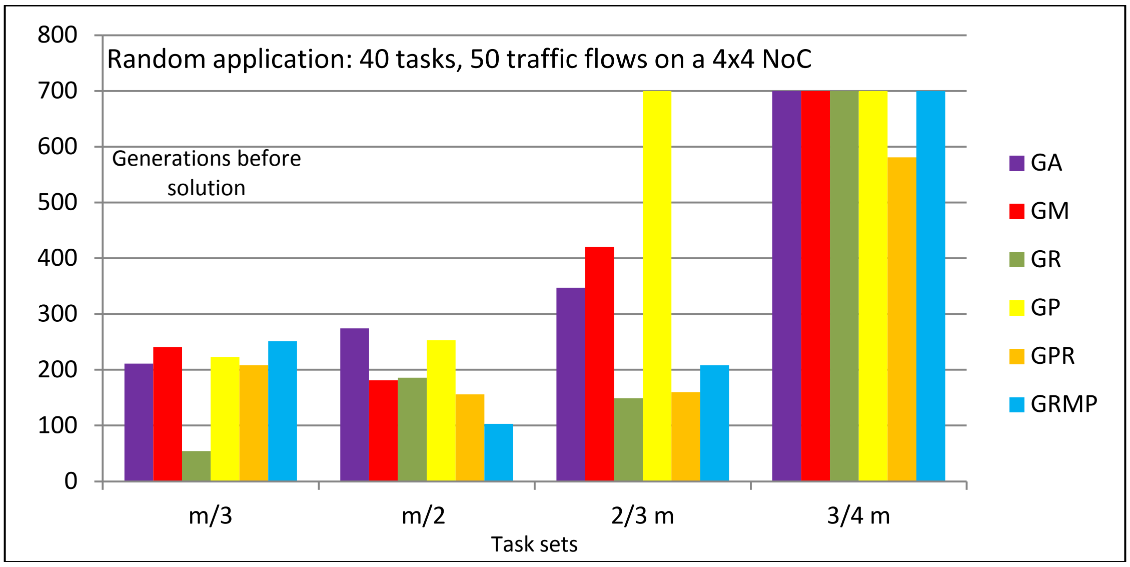


Figure 5.11 (iii). Mapping of random applications onto a NoC platform of size 4x4 using the different versions of the genetic algorithm. The application is composed of 40 tasks and 50 traffic flows.

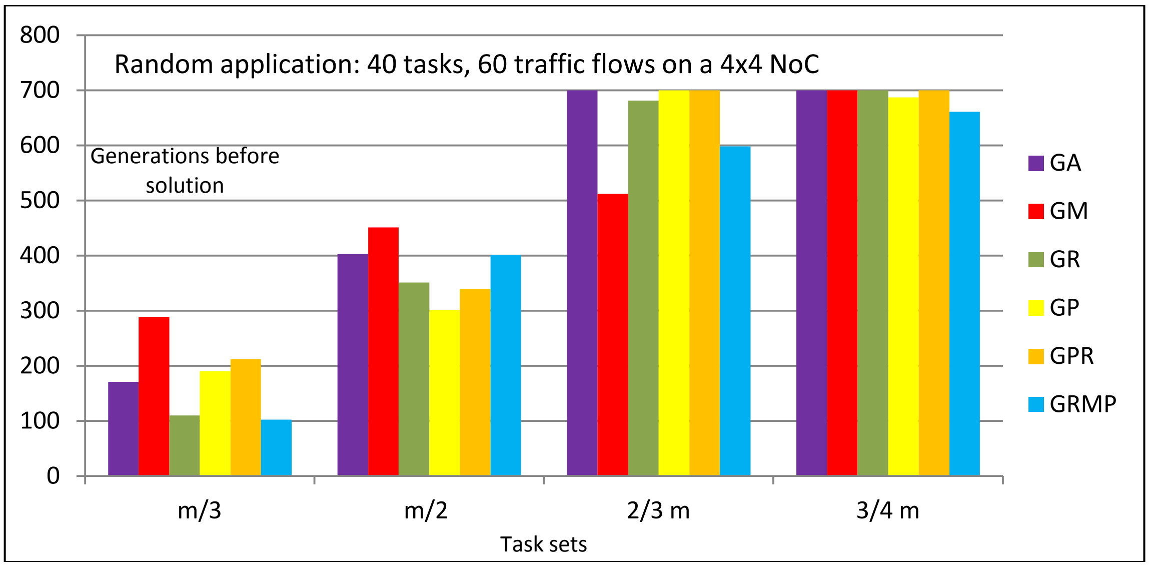


Figure 5.11 (iv). Mapping of random applications onto a NoC platform of size 4x4 using the different versions of the genetic algorithm. The application is composed of 40 tasks and 60 traffic flows.

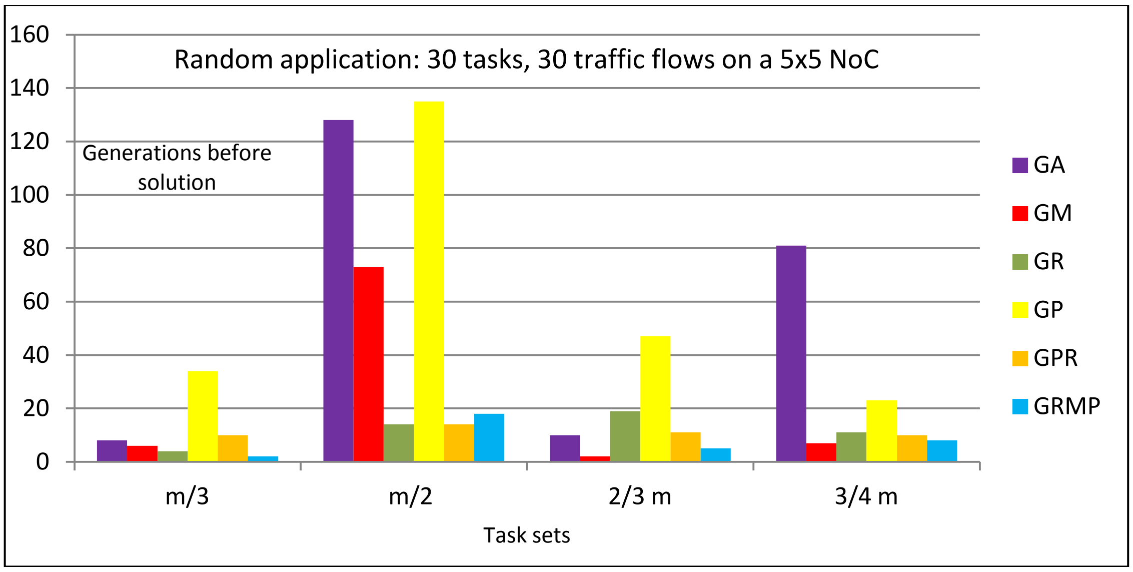


Figure 5.12 (i). Mapping of random applications onto a NoC platform of size 5x5 using the different versions of the genetic algorithm. The application is composed of 30 tasks and 30 traffic flows.

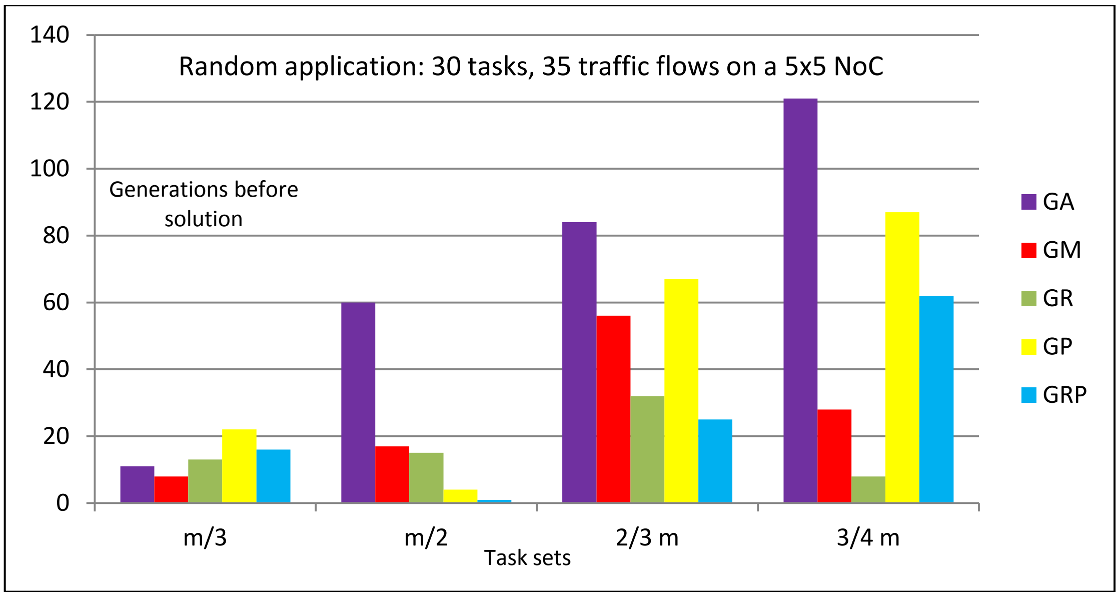


Figure 5.12 (ii). Mapping of random applications onto a NoC platform of size 5x5 using the different versions of the genetic algorithm. The application is composed of 30 tasks and 35 traffic flows.

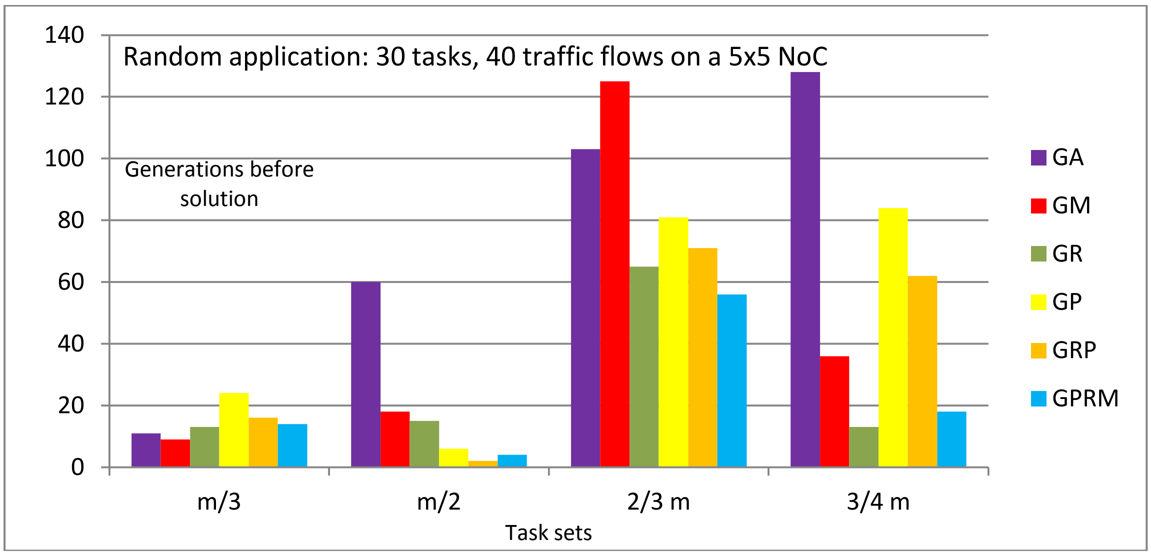


Figure 5.12 (iii). Mapping of random applications onto a NoC platform of size 5x5 using the different versions of the genetic algorithm. The application is composed of 30 tasks and 40 traffic flows.

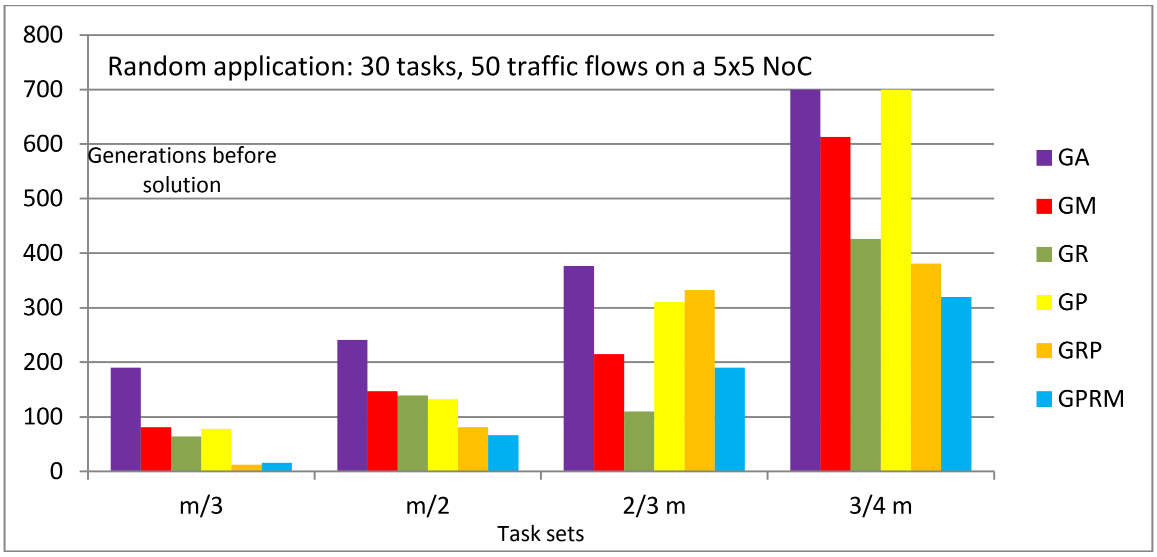


Figure 5.12 (iv). Mapping of random applications onto a NoC platform of size 5x5 using the different versions of the genetic algorithm. The application is composed of 30 tasks and 50 traffic flows.

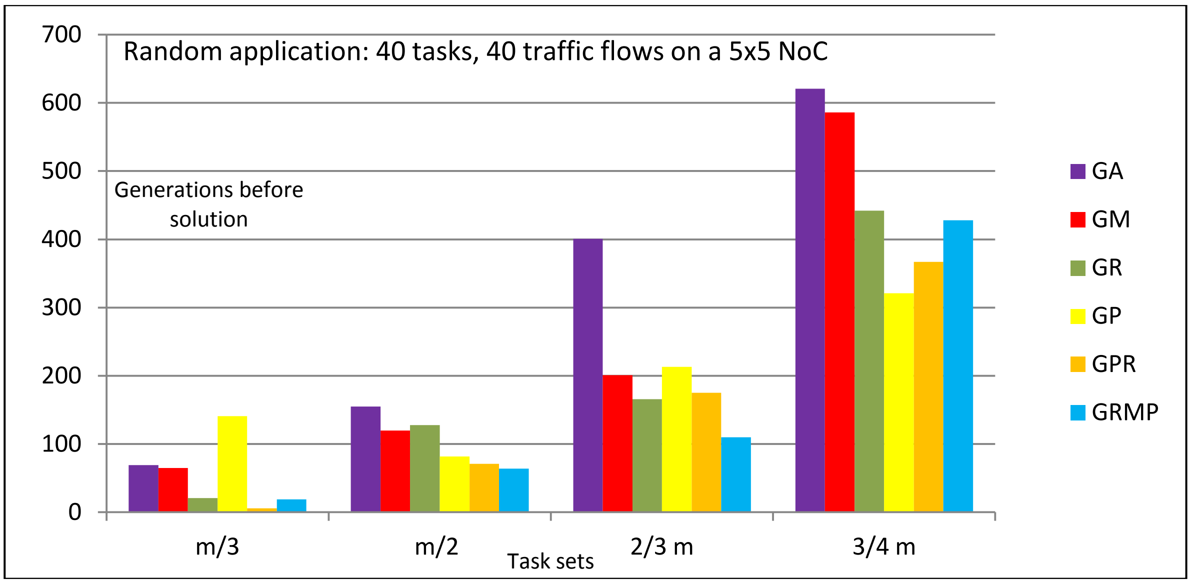


Figure 5.13 (i). Mapping of random applications onto a NoC platform of size 5x5 using the different versions of the genetic algorithm. The application is composed of 40 tasks and 40 traffic flows.

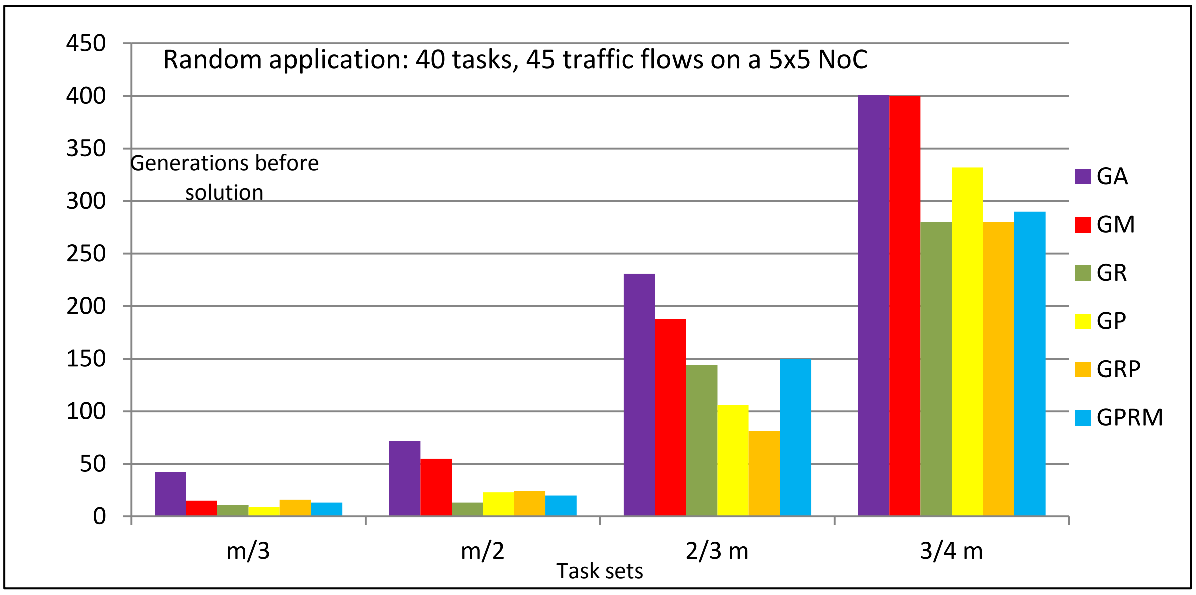


Figure 5.13 (ii). Mapping of random applications onto a NoC platform of size 5x5 using the different versions of the genetic algorithm. The application is composed of 40 tasks and 45 traffic flows.

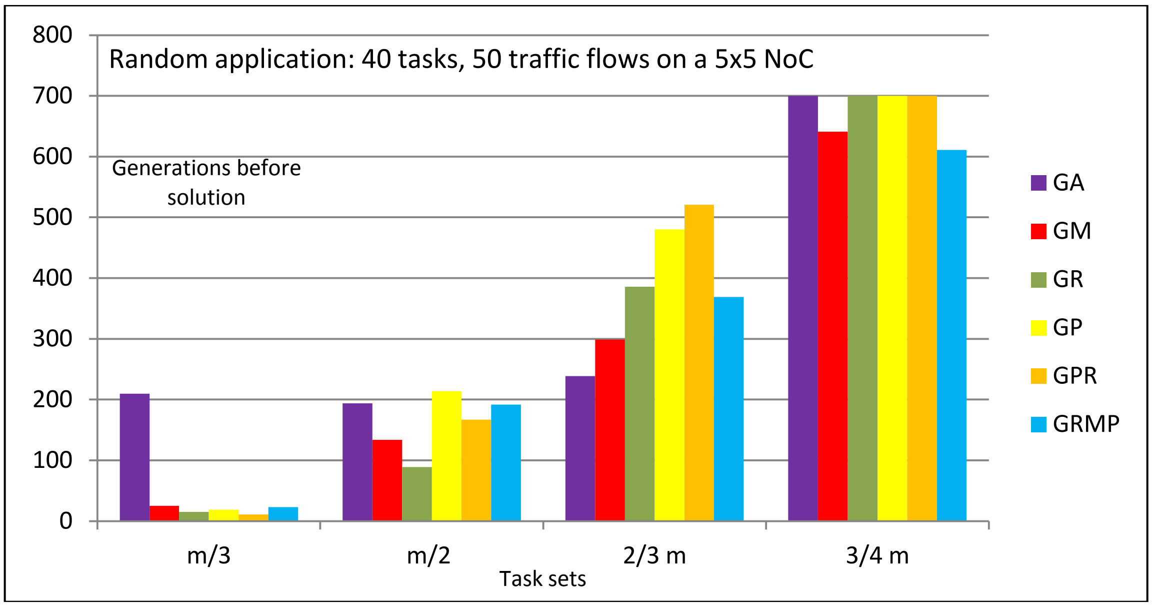


Figure 5.13 (iii). Mapping of random applications onto a NoC platform of size 5x5 using the different versions of the genetic algorithm. The application is composed of 40 tasks and 50 traffic flows.

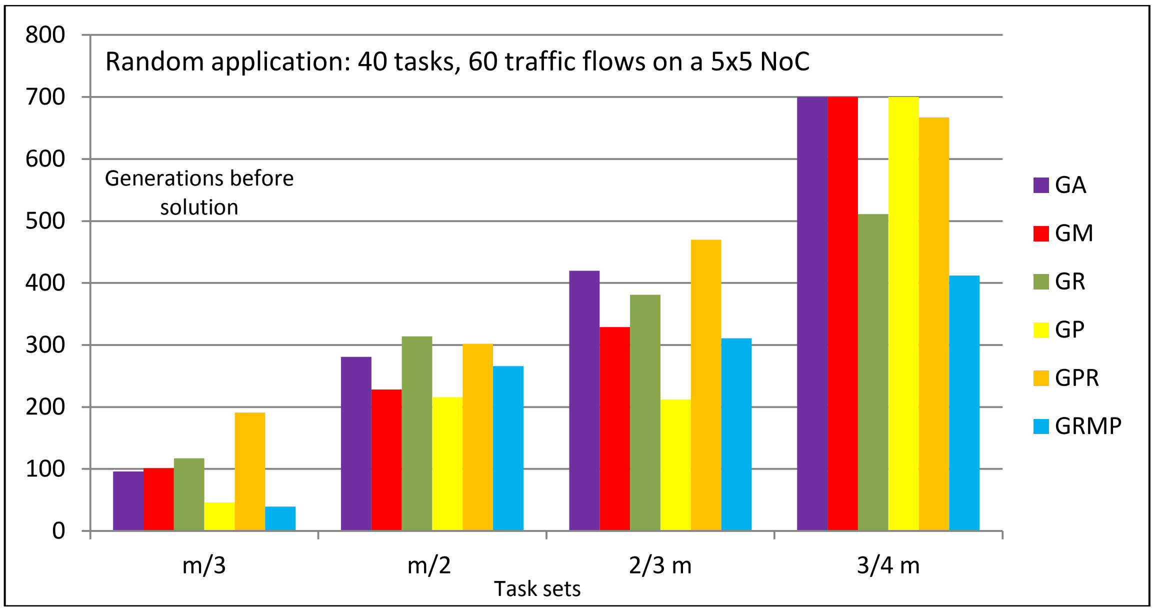


Figure 5.13 (iv). Mapping of random applications onto a NoC platform of size 5x5 using the different versions of the genetic algorithm. The application is composed of 40 tasks and 60 traffic flows.

In the results above we see how the different algorithms perform for various problem instances. In each case the algorithms share the exact application instance, the metric observed in the Y axis is the number of generations required by a version of the algorithm to generate a solution. The comparison of the different algorithms may require an even larger test set because a. different versions based on the genetic algorithm are tested which by default involves a lot of randomness, b. each run of a genetic algorithm is using a number of random mapping solutions as the initial input so the overall running time can be significantly affected by the quality of the initial solutions. The aim here was to observe any existing trends in the results of the experiments carried out. In the above experiments the number of seven hundred generations was used as a cap where if a run of the algorithm would not find a solution within this number it was considered as unable to find a solution; this interval was chosen empirically as being the reasonable time frame within which the algorithm should complete.

The basic genetic algorithm in the majority of cases needs the most generations to find a solution however this does not mean that it is slower than the other versions; as the problem instances get harder the basic version becomes to unable to find solutions (within the given timeframe and beyond it) and the different versions do so both in less generations and faster.

From the guided mutations proposed in chapter four we used only the mutation operator that would perturb a solution by moving a task from the most to the least utilised core, the guided mutation that would directly affect traffic (trying to map tasks of demanding flows together) was avoided so that the performance of the additional algorithms that operate on traffic flows (minimum contention routing, priority assignment) would not be affected when tested. This mutation operator was applied together with random mutation, at the same step of the algorithm, so that the necessary randomness would still be present. This addition to the algorithm compared to the basic version is shown to have some positive effect as the task set utilisation increases in relation to the platform resources.

The version of the algorithm using the minimum contention route generation algorithm is shown to always require less iterations than the basic version although it takes significantly longer per iteration. There is no clear trend in the above results relating the performance of this algorithm to the increasing computation or communication; intuitively it should have a great impact in the presence of many traffic flows. This algorithm also depends on the size limit set on its search space (number of routes to consider) so in this case there is a trade-off between running time and the solutions provided. In a hard problem instance the high running time may be insignificant as other algorithms may not be able to find solutions within the same amount of time.

The priority assignment policy in combination with the genetic algorithm is shown to have adverse effects on the ability of the algorithm. The reason could be the fact that the priority assignment misguides the selection stage of the genetic algorithm, in this version of the algorithm (GP) before a mapping solution is evaluated the traffic flows have their priorities re-assigned according to the specific policy then the number of unschedulable traffic flows is calculated and is assigned as the solution’s score. There is no indication showing that a candidate solution with a low score due to priority assignment may evolve to a better solution as the priority assignment itself does not affect the interference patterns.

In other versions of the algorithm where the priority assignment is used (GPR, GPRM), mainly in combination with the route generation algorithm, it does not seem to have a negative effect on the performance but it seems to further improve its ability in comparison to the GR version;

Finally the version of the algorithm that combines all of the approaches together seems to outperform all the other versions in this set of experiments, a fact that means that the different "optimisations" proposed on the original algorithm do not have adverse effects on each other when used together. The GPRM version is slower per iteration than the basic approach (mainly because of the route generation algorithm) but is able to provide solutions in almost every case. These results also indicate that a mapping algorithm can be a good "vessel" for combining various algorithms and strategies for the mapping problem together.

The results validate the intuition behind the different approaches however more extensive testing is required in order to stress the algorithms' ability to find solutions in hard instances and to provide insight to the interrelations between the different methods.

Next the basic version of the algorithm was run and for the best mapping solution of every generation additional parameters of the mapping solution were recorded. These parameters were those of the mean absolute deviation of the link utilisation and the sum total of the number of hops for every route. We would like to see if there is any clear trend between the schedulability of each solution and these parameters.

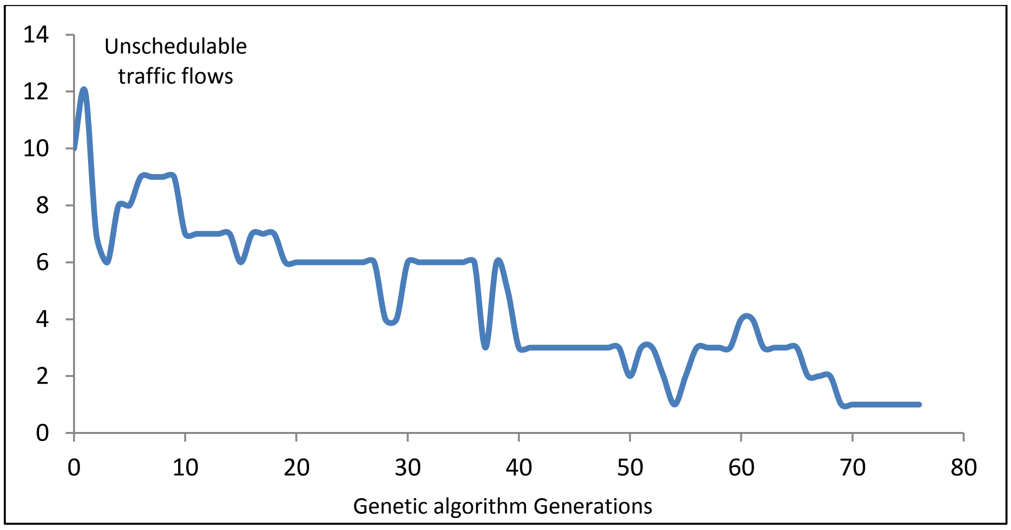


Figure 5.14 (i). The evolution of the values of unschedulable traffic flows for the Autonomous Vehicle application mapping on a 4x4 NoC

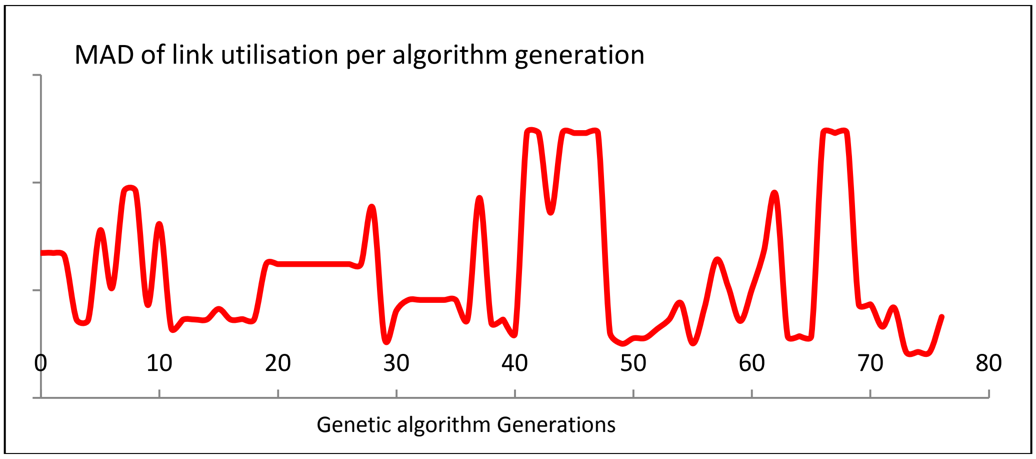


Figure 5.14 (ii). The evolution of the values of the mean absolute deviation of the utilisation of the network links versus the genetic algorithm generations for the Autonomous Vehicle application mapping on a 4x4 NoC.

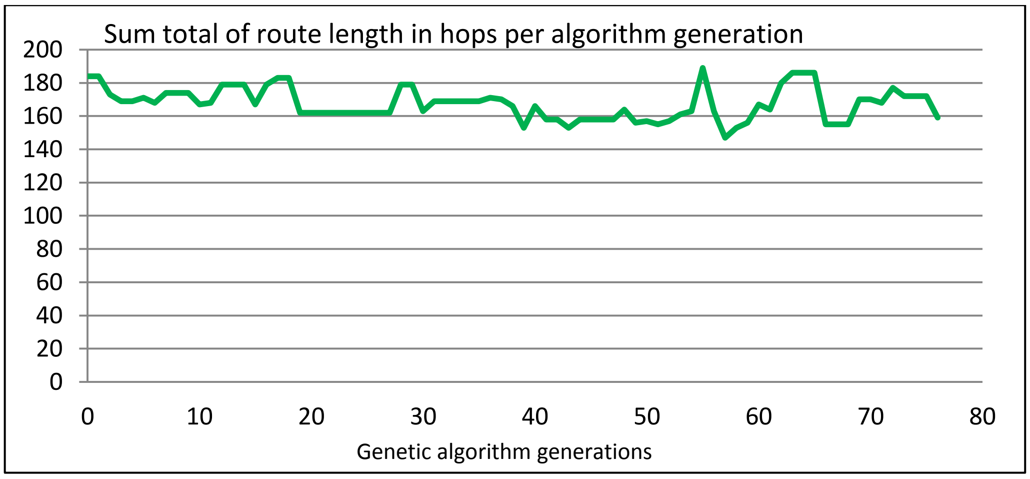


Figure 5.14 (iii). The total sum of the route length over all traffic flows per generation of the genetic algorithm, values recorded for the mapping of the autonomous vehicle application on a NoC platform of size 4x4.

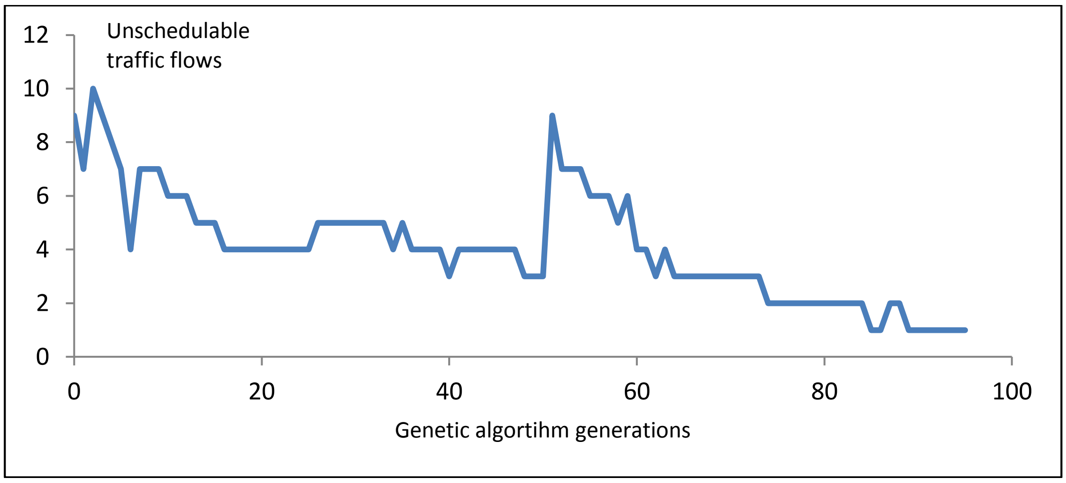


Figure 5.15 (i). The evolution of the values of unschedulable traffic flows for the synthetic application mapping on a 4x5 NoC.

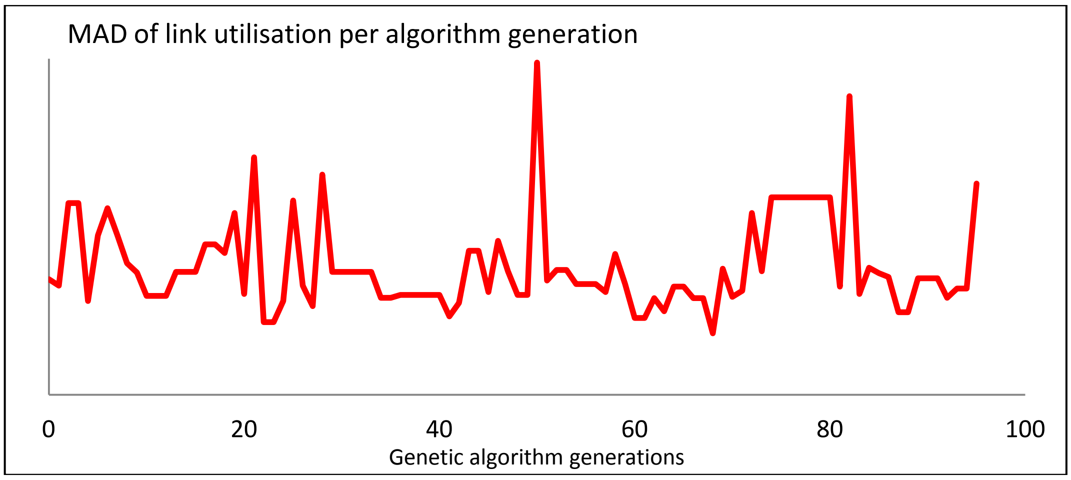


Figure 5.15 (ii). The evolution of the values of the mean absolute deviation of the utilisation of the network links versus the genetic algorithm generations for the synthetic application mapping on a 4x5 NoC.

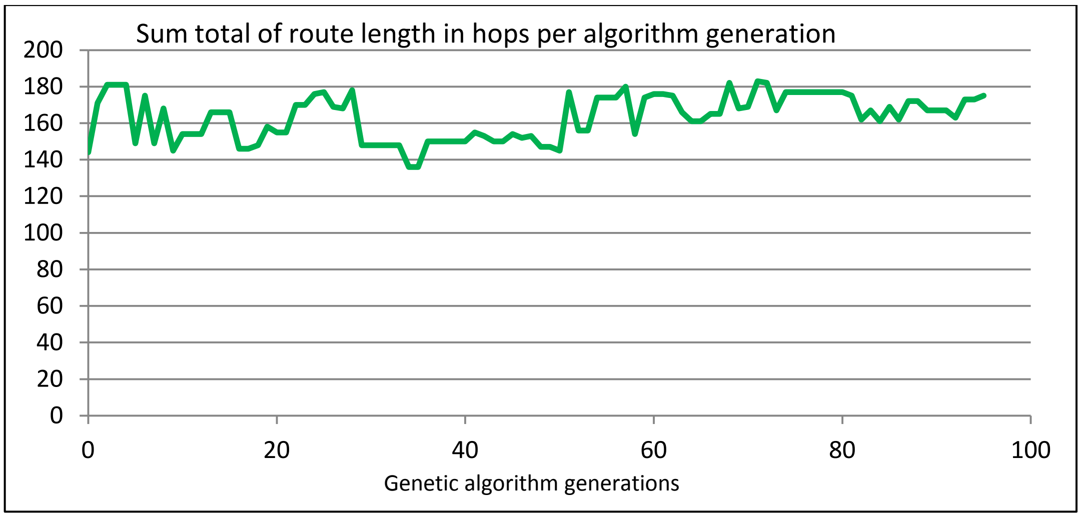


Figure 5.15 (iii). The total sum of the route length over all traffic flows per generation of the genetic algorithm, values recorded for the mapping of the synthetic application on a NoC platform of size 4x5.

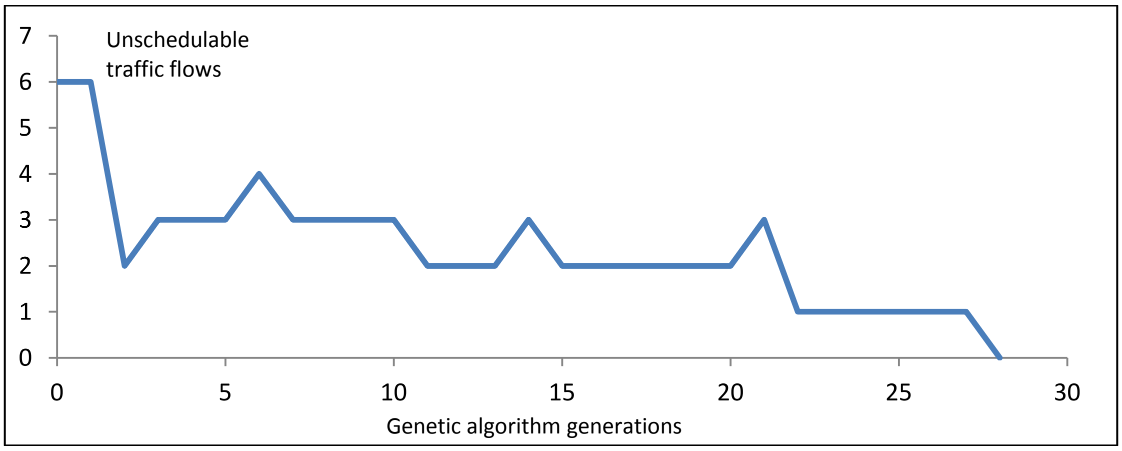


Figure 5.16 (i). The evolution of the values of unschedulable traffic flows for the mapping of a random application comprising of 30 tasks and 35 traffic flows with an overall task utilization of 8 on a NoC platform of size 4x4.

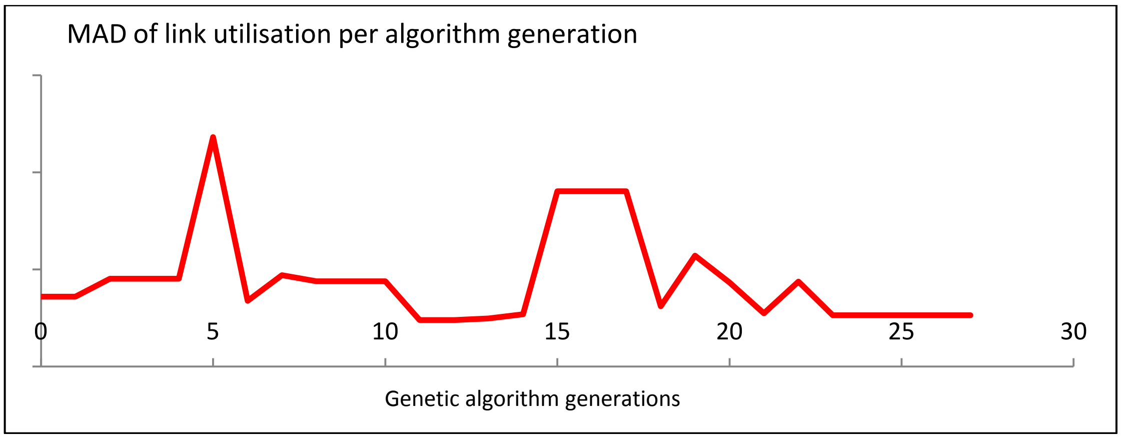


Figure 5.16 (ii). The evolution of the values of the mean absolute deviation of the utilisation of the network links versus the genetic algorithm generations for the mapping of a random application comprising of 30 tasks and 35 traffic flows with an overall task utilization of 8 on a NoC platform of size 4x4.

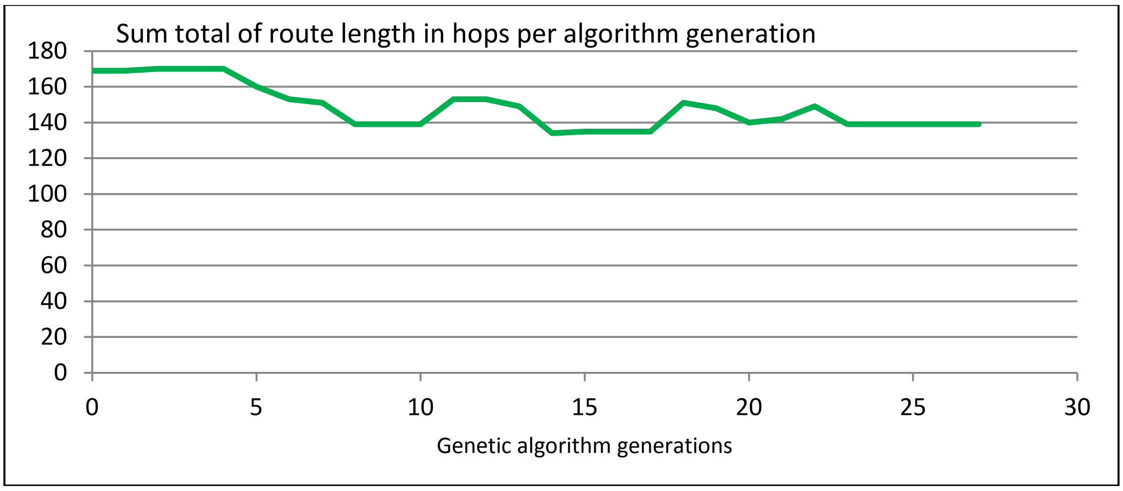


Figure 5.16 (iii). The total sum of the route length over all traffic flows per generation of the genetic algorithm, values recorded for the mapping of a random application comprising of 30 tasks and 35 traffic flows with an overall task utilization of 8 on a NoC platform of size 4x4.

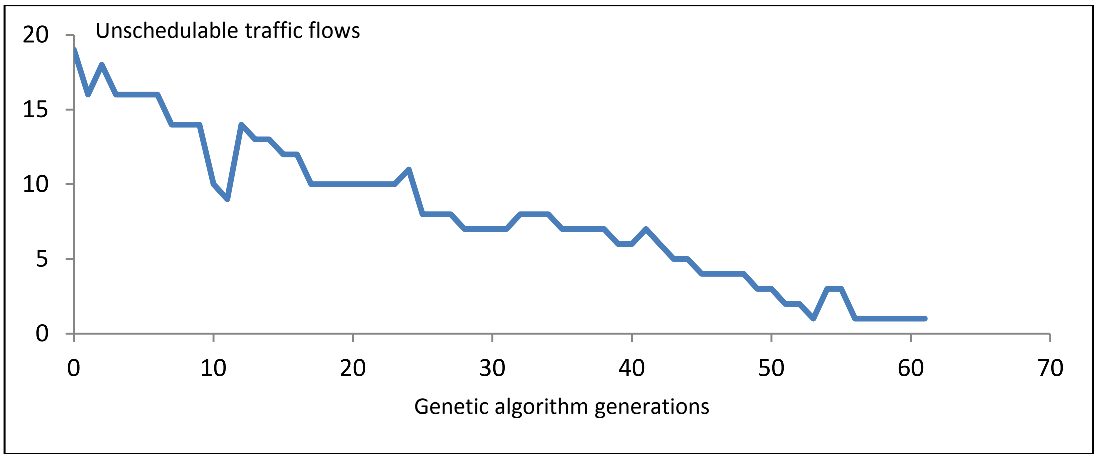


Figure 5.17 (i). The evolution of the values of unschedulable traffic flows for the mapping of a random application comprising of 30 tasks and 40 traffic flows with an overall task utilization of 8 on a NoC platform of size 4x4.

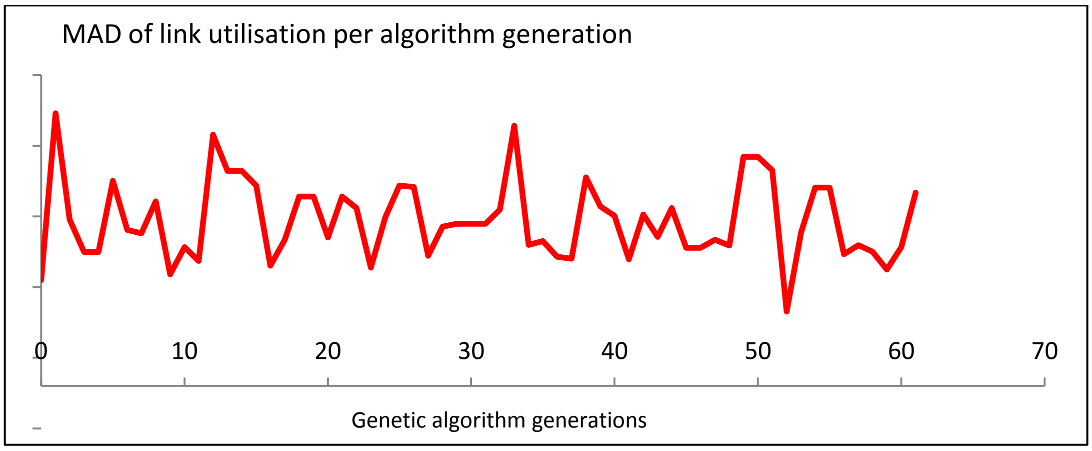


Figure 5.17 (ii). The evolution of the values of the mean absolute deviation of the network link utilisation versus the genetic algorithm generations for the mapping of a random application comprising of 30 tasks and 40 traffic flows with an overall task utilization of 8 on a NoC platform of size 4x4.

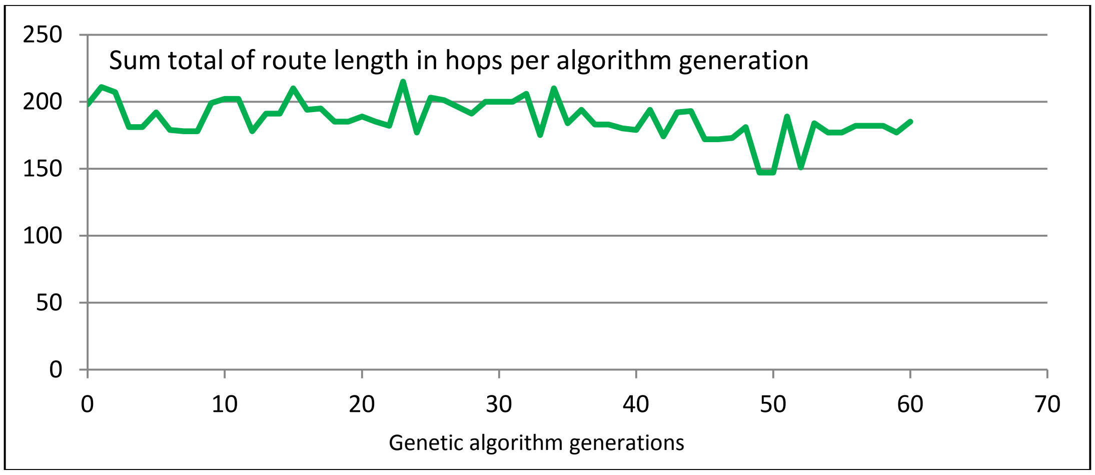


Figure 5.17 (iii). The total sum of the route length over all traffic flows per generation of the genetic algorithm, values recorded for the mapping of a random application comprising of 30 tasks and 50 traffic flows with an overall task utilization of 8 on a NoC platform of size 4x4.

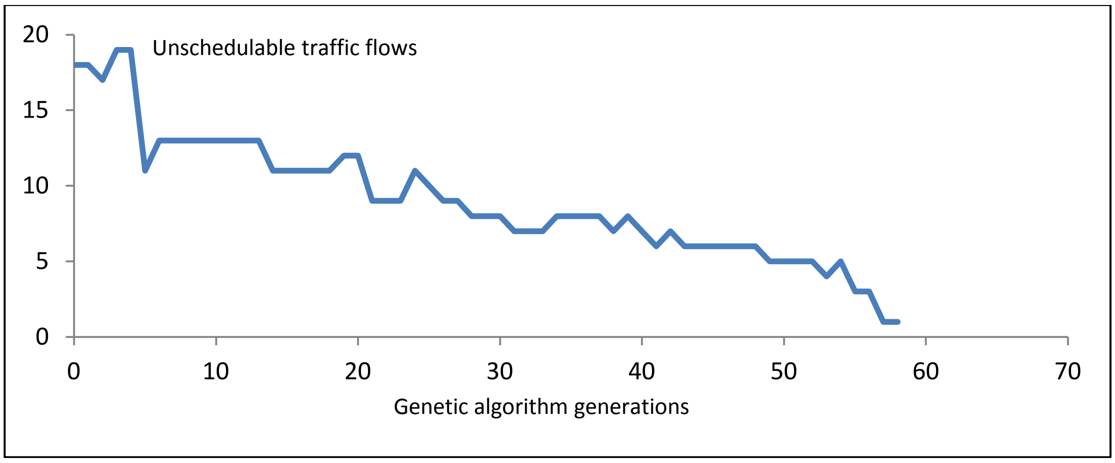


Figure 5.18 (i). The evolution of the values of unschedulable traffic flows for the mapping of a random application comprising of 30 tasks and 50 traffic flows with an overall task utilization of 8 on a NoC platform of size 4x4.

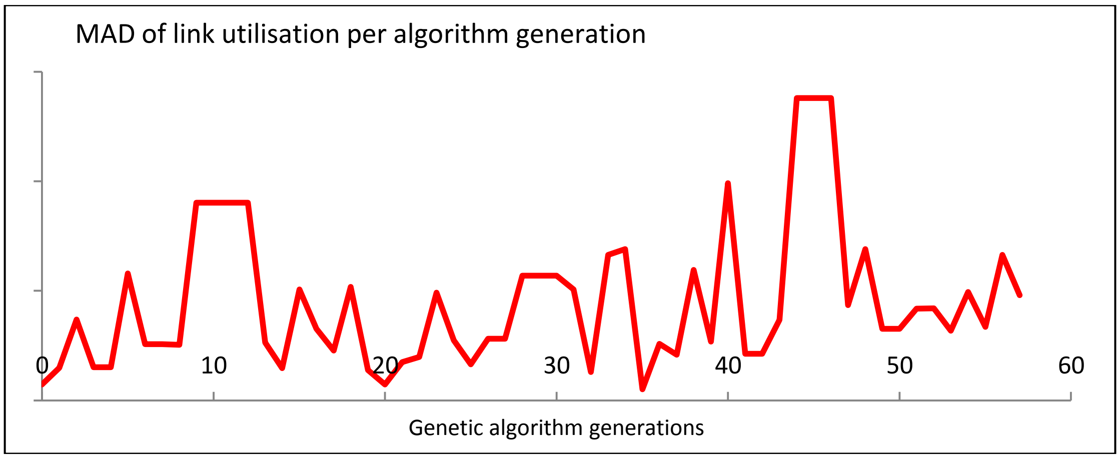


Figure 5.18 (ii). The evolution of the values of the mean absolute deviation of the network link utilisation versus the genetic algorithm generations for the mapping of a random application comprising of 30 tasks and 50 traffic flows with an overall task utilization of 8 on a NoC platform of size 4x4.

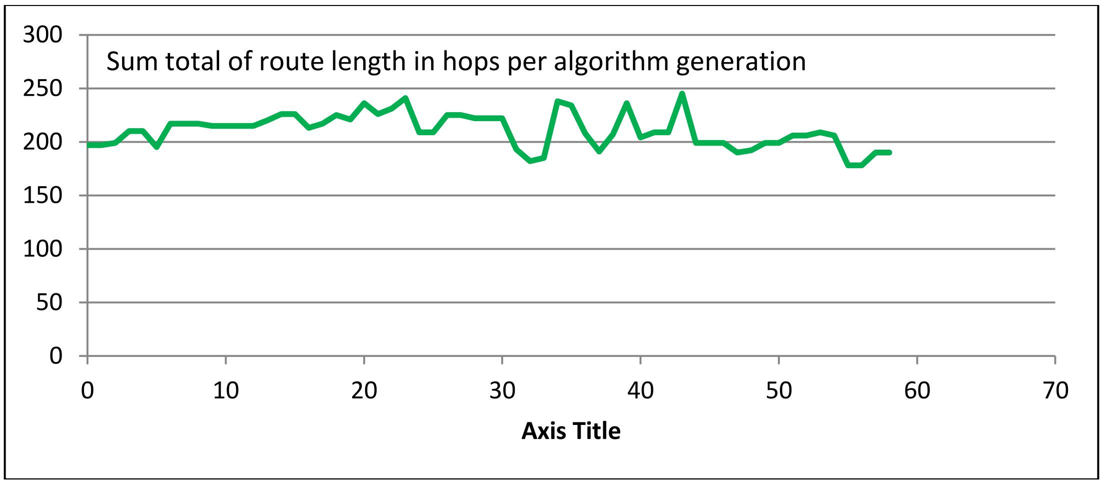


Figure 5.18 (iii). The total sum of the route length over all traffic flows per generation of the genetic algorithm, values recorded for the mapping of a random application comprising of 30 tasks and 50 traffic flows with an overall task utilization of 8 on a NoC platform of size 4x4.

In the results above there does not appear to be a clear relationship between the schedulability of traffic and the mean absolute average of the link utilisation because its value are highly variable typically ranging from $2.1678 \cdot 10^{-8}$ to $4.56734 \cdot 10^{-3}$ so no conclusion can be directly be drawn, additional experimentation and different statistical measures for variance may provide a better insight. The total sum of route lengths tends to decrease as the solutions become schedulable, an expected fact as shorter routes should interfere less, but it may still be possible for a different communication pattern involving longer routes to produce less interference a fact reflected in the results in figure 26 (iii) and 27 (iii). The relation between route length and interference may also be an attribute of the routing algorithm; an example algorithm may choose a longer route for a traffic flow instead of one of minimal length in order to avoid interference.

Chapter 6

Future Research

6.1 Suggestions for Future Research

As mentioned in the work above an important property of a mapping heuristic algorithm is the ability to find solution in instances that it is hard to do so. In order to test this ability it becomes necessary to generate applications that are hard to map on a given platform. Given such a case can help to determine the benefit of this algorithm over an exhaustive search of the solution space; given the size of the solution space which even for small instances of the mapping problem is big, the results produced clearly demonstrate that the algorithm can produce a solution much faster than such a brute force approach. Having a hard problem instance available the quality of this approach would be more apparent and it could also allow a comparison of different heuristics e.g. branch and bound. There is no proposed methodology in literature used to generate such a hard application instance with only a few unique mapping solutions for a given platform size; a possible approach would be to generate the application constructively in a way that computation and communication on a given platform are hardly schedulable.

In addition to the above the distribution and variance of traffic flow parameters such as period and payload could be related to the overall feasibility of traffic and to the hardness of a mapping problem instance. This kind of evolutionary algorithm would best suit large problem instances with many cores (dozens) and large applications in which case it should outperform simpler greedy heuristics (first fit, nearest fit etc.).

The additional algorithms proposed when combined with the genetic algorithm seem to improve its overall ability to produce solutions however the route generation and priority assignment can become part of the solution space that the genetic algorithm searches; for each given solution the algorithm could perturb the routes and priorities of traffic in a guided or random fashion and also recombine the routes and priorities of different solutions (crossover), this approach could then be compared to the one proposed here. With regards to communication an additional metric that can easily be incorporated in the cost function of the algorithm is the distance between communicating tasks, in this case the algorithm would favour solutions where this distance is minimised. Conversely any additional variable which can be expressed analytically can be integrated within this algorithm’s cost function making it a multi-objective algorithm.

Furthermore the set of possible routes considered by the minimum contention algorithm could be extended from the set of all minimal routes to a larger set which may satisfy some other properties giving the algorithm a larger search space hence more candidate solutions.

It would also be interesting to see the relation of the traffic distribution on the network and the schedulability of traffic. Intuitively such a relation must exist however it is not clear to what extent, in case there is a clear trend different routing approaches could be applied.

The above approaches can be further validated by using simulation. Through simulation it is possible to observe for what mapping solutions the analysis is actually tight (worst case response times are actually close to observed values). In addition if the algorithm would try and produce solutions that would try and optimise different metrics that cannot be expressed exactly by analytical methods but instead the algorithm tries to achieve such optimisations indirectly based on some intuitive method again simulation would be necessary. The authors in [105] have developed a simulation framework that specifically aims at real - time systems executing on platforms that implement priority based wormhole switching and directly supports the application model used in this work. In order to achieve fast and accurate simulation transaction level modelling [41] has also been successfully used for simulating such systems [44].

The mapping problem could be suitable to a divide and conquer approach, assuming that the application can be partitioned according to some meaningful criteria the different partitions of the application could then be placed on different areas of the platform, in which case the mapping problem would be solved for many small instances instead. We would like to partition the application graph in such a way that the communication between the different partitions is minimized as the tasks of a partition will be placed close together on the platform, tasks that communicate a lot between them should be placed in the same partition to ensure that communication becomes more "localized" so that the communication interference between different partitions is minimized. This problem resembles that of the minimum cut problem which is the problem of finding a cut of a graph so that the sum of the weights of edges crossing the cut is minimised. A solution to the min-cut problem will provide only two partitions that are probably highly unbalanced so it would have to be repeatedly solved on a large graph to produce balanced partitions suitable to the mapping problem. An existing algorithm by Karger et al. [100] is proposed, in this algorithm weighted edges are contracted merging their vertices in a partition, this contraction happens at random with a probability for each edge proportional to its weight. It would be interesting to see if such an approach can be used to solve the mapping problem or even as a pre-processing step that would produce a partial solution to be used as input to a heuristic algorithm.

In addition to the above the constructive heuristic algorithm proposed in this work can be further developed and enhanced into a heuristic that can tackle the mapping problem more efficiently in the context of static mapping, also the algorithm logic can be tested to see if it is able to provide good solutions in the case of dynamic mapping for soft real time applications where the approach proposed in [51] could be combined with this specific search pattern. For a real time system it would also be necessary to examine the impact of any runtime mechanisms such as task migration and re-mapping would have on the timeliness of the system traffic.

The system described in this work assumes one virtual channel per priority level which for a large number of traffic flows could be expensive in hardware and since the routes are deterministic and the application static this approach resembles a virtual circuit scheme; in such a case it could be more suitable to provide a custom topology and architecture or alternatively take advantage of the flexibility provided by a NoC (with regards to communication) to somehow decouple the computation placement and communication schedulability so that more efficient approaches can be used (global, dynamic algorithms). In addition the flexibility provided by NoC platforms may also be used to provide fault tolerant communications for hard-real time systems.

To overcome the cost incurred by using one virtual channel per priority level Shi and Burns [103] propose a priority share policy where different traffic flows may share a priority level in order to reduce the resource overhead and at the same time the proposed analysis model is extended to suit this scheme still achieving hard real-time communication guarantees, so the approach proposed here could be further enhanced by using this model.

6.2 Conclusions

In this work we tried to approach the application mapping problem in the context of real-time systems. The mapping problem is not only intractable for a certain goal but also has an impact on many different parameters of the system. The problem formulation varies for different systems based on the system model assumptions, constraints and goals, so a universal solution is not possible. We restricted our research to a NoC platform type that is suitable for real-time communications (priority based wormhole routing) and used response time analysis methods recently proposed for such systems in order to guarantee the timeliness of real-time communications. In order to solve the mapping problem for hard real-time applications on NoC we chose to use a straightforward evolutionary algorithm that uses this analytical method as its ranking function. This method is able to provide schedulability analysis for both computation and communication on a NoC platform that uses wormhole routing and pre-emptive arbitration. This algorithm, as the results show, was able to find solutions in some cases, however there is no guarantee that there are no existing solutions for those cases where this genetic algorithm cannot find one over a specific number of generations (such as mapping the autonomous vehicle application onto a 3x3 NoC or the synthetic application onto a 4x4 NoC) i.e. the algorithm is not optimal.

Various additions to the algorithm were proposed that would improve the schedulability of traffic and hence improve the algorithm's ability to find solutions in comparison to the basic version. The results for the set of experiments carried out validate the basic intuition behind the algorithms but are not able to clearly demonstrate any complex interrelationships between the different algorithms combined together and between the traffic patterns on the platform and the schedulability of traffic flows.

A constructive heuristic method was developed and tested alongside the genetic algorithm but was not able to provide any consistent results however the development of such an algorithm could be an avenue of research for the mapping problem.

The combination of a genetic algorithm with response time analysis as the ranking function may also serve as a starting/reference point for the development of other algorithms for the same problem, in which case any new approaches could be compared against it.

Many solutions have been suggested to the mapping problem however the problem has not yet been studied in depth from a real-time systems perspective in which case many problems are still open and many possibilities exist for both hard and soft real-time systems.

BIBLIOGRAPHY

[1] L. Benini and G. de Micheli, “Networks on chips: a new SoC paradigm,” *Computer*, vol. 35, no.1, pp. 70–78, 2002.

[2] W. J. Dally and B. Towles, “Route packets, not wires: on-chip interconnection networks,” in *Proceedings of the 38th Design Automation Conference (DAC ’01)*, pp. 684–689, Las Vegas, Nev, USA, June 2001.

[3] P. M. Pardalos, F. Rendl, and H. Wolkowicz, “The quadratic assignment problem: a survey and recent developments,” in *Quadratic Assignment and Related Problems*, P. M. Pardalos and H. Wolkowicz, Eds., vol. 16 of *DIMACS Series in Discrete Mathematics and Theoretical ComputerScience*, pp. 1–42, American Mathematical Society, Providence, RI, USA, 1994.

[4] J. Hu and R. Marculescu, “Energy- and performance-aware mapping for regular NoC architectures,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 4, pp. 551–562, Apr. 2005.

[5] A. Hansson, K. Goossens, and A. Radulescu, “A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic,” *Hindawi VLSI Design*, May 2007.

[6] Radu Marculescu, Umit Y. Ogras, Li-Shiuan Peh, Natalie Enright Jerger, and Yatin Hoskote, *“Keynote Paper:* Outstanding Research Problems in NoC Design: System, Micro architecture, and Circuit Perspectives” IEEE transactions on computer-aided design of integrated circuits and systems, vol. 28, no. 1, January 2009.

[7] Abhik Roychoudhury. Embedded Systems and Software Validation. The Morgan Kaufmann Series in Systems on Silicon 2009. ISBN 13: 978-0-12-374230-8.

[8] Petri, C. Fundamentals of a theory of asynchronous information flow, Proc. IFIP Congress 62, 1962. pp. 386–390.

[9] Hoare, C. A. R. Communicating Sequential Processes, Communications of the ACM, Vol. 21, No. 8 Aug. 1978.

[10] Kahn, G. The semantics of a simple language for parallel programming, in Proceedings of the IFIP Congress 74. Amsterdam, The Netherlands. North-Holland, 1974.

[11] E. A. Lee. And D. G. Messerschmitt. Synchronous Data Flow. In proceeding of the IEEE volume 75, pages 1235-1245, September 1986.

[12] E.A. Lee. Overview of the Ptolemey Project. Technical Report. University of California Berkeley. March 2001. Technical Memorandum UCB/ERL M01/11.

[13] Filip Thoen, Jan Van Der Steen, Gjalt de Jong, Gert Goossens and Hugo De Manz. Multi- Thread Graph : A System Model for Real-Time Embedded Software Synthesis. IMEC, B-300140 Katholieke Universiteit Leuven, Belgium.

[14] Sylvester, d. and Keutzer, k. 2000. A global wiring paradigm for deep submicron design. *IEEE Transactions Computer Aided Design Integrated Circuits Syst. 19,* 242–252.

[15] Tobias Bjerregaard and Shankar Mahadevan. A Survey of Research and Practices of Networkon-Chip. ACM Computing Surveys, Vol. 38, March 2006, Article 1.

[16] V. Lo et al.,"OREGAMI: Tools for Mapping Parallel Computations to Parallel Architectures", Intl Journal of Parallel Programming, vol. 20, no. 3, 1991, pp. 237-270.

[17] N. Koziris et al.,"An Efficient Algorithm for the Physical Mapping of Clustered Task Graphs onto Multiprocessor Architectures", Proc. Of 8th EuroPDP, pp. 406-413, Jan, 2000.

[18] S. H. Bokhari, "On the Mapping Problem," IEEE Trans. Comput., vol. 30, no. 3, pp. 207-214, 1981.

[19] Tang *Lei*, Shashi *Kumar*, "Algorithms and Tools for Network on Chip Based System Design,"in SBCCI, pp.163, 2003.

[20] Murali, S.; et al. A methodology for mapping multiple use-cases onto networks on chips. In: DATE, 2006. pp. 118-123.

[21] Manolache, S.; et al. Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC. DAC, 2005. pp. 266-269.

[22] Giuseppe Ascia, Vincenzo Catania, and Maurizio Palesi. A Multi-objective Genetic Approach to the Mapping Problem on Network-on-Chip. Journal of Universal Computer Science, vol. 12, no. 4 (2006), pp. 370-394.

[23] S. Murali and G. de Micheli, "Bandwidth-constrained mapping of cores onto NoC architectures," in *Proceedings of Design, Automation and Test in Europe Conference and Exposition (DATE '04)*, vol. 2, pp. 896–901, Paris, France, February 2004.

[24] S.Murali, L. Benini, and G. de Micheli, "Mapping and physical planning of networks-on-chip architectures with quality-of service guarantees," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '05),* vol. 1, pp. 27–32, Shanghai, China, January 2005.

[25] César A. M. Marcon, Edson I. Moreno, Ney L. V. Calazans and Fernando G. Moraes. Evaluation of Algorithms for Low Energy Mapping onto NoCs IEEE International Symposium on Circuits and Systems 2007.

[26] César Marcon, André Borin, Altamiro Susin, Luigi Carro, Flávio Wagner. Time and Energy Efficient Mapping of Embedded Applications onto NoCs. Proceedings of the 2005 Asia and South Pacific Design Automation Conference.

[27] K. Srinivasan and K. S. Chatha, "A technique for low energy mapping and routing in network- on-chip architectures," in *Proc. 2005 Int'l Symp. Low Power Electronics and Design*, 2005, pp. 387- 392.

[28] Souleyman Tosun. New heuristic algorithms for energy aware application mapping and routing on mesh-based NoCs. Journal of Systems Architecture: the EUROMICRO Journal archive Volume 57 Issue 1, January, 2011 Elsevier North-Holland, Inc. New York, NY, USA.

[29] J. Duato, S. Yalamanchili, L. Ni, *Interconnection Networks: an Engineering Approach*. Morgan Kaufmann, 2003.

[30] Moraes, F., Calazans, N.,Mello, A.,Moller, L., and Ost, l. 2004. HERMES: An infrastructure for low area overhead packet-switching networks on chip. *The VLSI Integration 38*, 69–93.

[31]T. Bjerregaard and J. Sparso, “Virtual channel designs for guaranteeing bandwidth in asynchronous network-on-chip,” in *Norchip Conference, 2004. Proceedings*, pp. 269-272, 2004.

[32] K. Goossens, J. Dielissen, and A. Radulescu, “AEthereal network on chip: concepts, architectures, and implementations,” *Design & Test of Computers, IEEE*, vol. 22, no. 5, pp. 414- 421, 2005.

[33] Z. Shi and A. Burns, “Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching,” in *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, 2008, pp. 161-170.

[34] Moraes, F., Calazans, N., Mello, A.,Moller, L., and Ost, L. 2004. HERMES: An infrastructure for low area overhead packet-switching networks on chip. *The VLSI Integration 38*, 69–93.

[35] Bolotin E., Cidon I., Ginosaur R., and Kolodny A. 2004. QNoC: QoS architecture and design process for network-on-chip. *J. Syst. Archit. 50*, 2-3, 105–128.

[36] Ahonen, T., Sigenza-Tortosa, D. A., Bin, H., and Nurmi, J. 2004. Topology optimization for application specific networks-on-chip. In *International Workshop on System Level Interconnect Prediction (SLIP)*. ACM, 53–60.

[37] N. Banerjee, P. Vellank, and K. S. Chatha, “A power and performance model for network-onchip architectures,” in *Proc. Des., Autom. Test Eur. Conf.*, Feb. 2004, pp. 1250–1255.

[38] N. Eisley and L. Peh, “High-level power analysis for on-chip networks,” in *Proc. Int. Conf. Compilers, Architectures Synthesis Embedded Syst.*, Sep. 2004, pp. 104–115.

[39] Adsen, J., Mahadevan, S., Virk, K., and Gonzalez, M. 2003. Network-on-chip modeling for system-level multiprocessor simulation. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS)*. IEEE, 82–92.

[40] N. Genko et al., “A novel approach for network on chip emulation,” in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pp. 2365-2368 Vol. 3, 2005.

[41] Lukai Cai and Daniel Gajski “ Transaction Level Modeling: An Overview “ in proceedings of the Int. Conference on HW/SW Codesign and System Synthesis (CODES-ISSS), Oct. 2003, pp. 19– 24.

[42] M. Hosseinabady and J. L. Nunez-Yanez, “SystemC Architectural Transaction Level Modelling for Large NoCs,” in *ECSI Forum on Specification and Design Languages (FDL),* 2010.

[43] E. Viaud, F. Pêcheux, and A. Greiner, “An efficient TLM/T modeling and simulation environment based on conservative parallel discrete event principles,” in *Proceedings of the conference on Design, automation and test in Europe: Proceedings,* pp. 94-99, 2006.

[44] L. S. Indrusiak, O. M. Santos Fast and Accurate Transaction-Level Model of a Wormhole Network-on-Chip with Priority Pre-emptive Virtual Channel Arbitration 2011. Design Automation and Test in Europe (DATE).

[45] J. Hu and R.Marculescu, “Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures,” in *Proceedings of Design, Automation and Test in Europe Conference and Exposition (DATE ’03)*, pp. 688–693, Munich, Germany, March 2003.

[46] E. Fleury and P. Fraigniaud, “A general theory for deadlock avoidance in wormhole-routed networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 7, pp. 626–638, 1998.

[47] Xiaohang Wang, Mei Yang, Yingtao Jiang, and Peng Liu, “Power-Aware Mapping for Network-on-Chip Architectures under Bandwidth and Latency Constraints” 4th International conference in Embedded and Multimedia Computing 2009 EM – Com.

[48] Ewerson Carvalho, Ney Calazans, Fernando Moraes. Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs. 18th IEEE/IFIP Rapid System Prototyping (RSP 2007).

[49] Ewerson Carvalho, Ney Calazans, Fernando Moraes. Investigating Runtime Task Mapping for NoC-based Multiprocessor SoCs. In IFIP VLSI SoC, 2009.

[50] Carvalho, E.; Moraes, F. Congestion-aware Task Mapping in Heterogeneous MPSoCs. In: SoC, 2008. pp. 1-4.

[51], Philip K.F. Hölzenspies, Johann L. Hurink, Jan Kuper, Gerard J.M. Smit. Run-time Spatial Mapping of Streaming Applications to a Heterogeneous Multi-Processor System-on-Chip (MPSoC). In DATE Proceedings of the conference on Design, automation and test in Europe 2008.

[52] Amit Kumar Singh, Thambipillai Srikanthan, Akash Kumar, Wu Jigang. Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms. Journal of Systems Architecture 56 (2010). pp. 242–255.

[53] Amit Kumar Singh, Wu Jigang, Alok Prakash, Thambipillai Srikanthan. Mapping Algorithms for NoC-based Heterogeneous MPSoC Platforms. 2009 12th Euromicro Conference on Digital System Design / Architectures, Methods and Tools.

[54] Luciano Ost, Marcelo Mandelli, Gabriel Marchesan Almeida, Leandro Moller, Leandro Soares Indrusiak, Gilles Sassatelli, Pascal Benoit, Manfred Glesner, Michel Robert, Fernando Moraes. Power-aware dynamic mapping heuristics for NoC-based MPSocs using a unified model-based approach. Submitted to ACM Tecs. special issue on-chip and off-chip network architectures 2011.

[55] Mehran, A. et al. DSM: A Heuristic Dynamic Spiral Mapping algorithm for network on chip. IEICE Electronics Express, v. 5-13, p.464-471, 2008. pp. 464-471.

[56] Chou, C-L.; Marculescu, R. User-Aware Dynamic Task Allocation in Networks-on-Chip. In: DATE, 2008. pp.1232-1237.

[57] Wronski, F.; et al. Evaluating Energy-aware Task Allocation Strategies for MPSoCS. In: DIPES. 2006.

[58] M.Wiggers,M. Bekooij, and G. Smit. Efficient computation of buffer capacities for cyclo-static dataflow graphs. In *Proc. of DAC '07,* pages 658–663, 2007.

[59] V. Nollet, T. Marescaux, D. Verkest, "Operating-system controlled network on chip," Proc. DAC, San Diego, CA, June 2004.

[60] R.Dick, Embedded system synthesis benchmarks suites (E3S), http://www.ece.northwestern.edu/~dickrp/e3s.

[61] S. Saeidi, A. Khademzadeh and A. Mehran, "SMAP: An Intelligent Mapping Tool for Network on Chip," IEEE explore 2007.

[62] Z. Shi, A. Burns, and L. S. Indrusiak, "Schedulability Analysis for Real Time On-Chip Communication with Wormhole Switching," *International Journal of Embedded and Real-Time Communication Systems,* vol. 1, no. 2, pp. 1 - 22, Jun. 2010.

[63] Stefano Bertozzi, Andrea Acquaviva, Davide Bertozzi, and Antonio Poggiali. Supporting task migration in multi-processor systems-on-chip: a feasibility study. Proceedings of the conference on Design, automation and test in Europe. . DATE 2006 Proceedings.

[64] V. Nollet et al., Centralized run-time resource management in a network on chip containing reconfigurable hardware tiles, in: Proceedings of DATE, 2005, pp. 234–239.

[65] Xin Wang, Tapani Ahonen, and Jari Nurmi. Applying CDMA Technique to Network-on-Chip. IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 15, NO. 10, OCTOBER 2007.

[66] J. W. van den Brand, C. Ciordas, K. Goossens, and T. Basten, "Congestion-controlled best effort communication for networks-onchip," in *Proc. Des., Autom. Test Eur. Conf.*, Apr. 2007, pp. 948–953.

[67] M. Harmanci, N. Escudero, Y. Leblebici, and P. Ienne, "Quantitative modeling and comparison of communication schemes to guarantee quality-of-service in networks-on-chip," in *Proc. Int. Symp. CircuitsSyst.*, May 2005, pp. 1782–1785.

[68] U. Y. Ogras and R. Marculescu, "Analysis and optimization of prediction-based flow control in networks-on-chip," *ACM Trans. Des.Autom. Electron. Syst.*, vol. 13, no. 1, pp. 1–28, Jan. 2008.

[69] Tindell, K., Burns, A., & Wellings, A. (1994). An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems Journal*, *6*(2), 133–151. doi: 10.1007/BF01088593.

[70] L. F. Leung and C. Y. Tsui, "Optimal link scheduling on improving best-effort and guaranteed services performance in network-on-chip systems," in *Proc. Des. Autom. Conf.*, Jul. 2006, pp. 833–838.

[71] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," in *Proc. Des., Autom. Test Eur. Conf.*, Feb. 2004, pp. 890–895.

[72] M. Coenen, S. Murali, A. Radulescu, K. Goossens, and G. De Micheli. A buffer-sizing algorithm for networks on chip using tdma and credit-based end-to-end flow control. In 4th International Conference on Hardware- Software Codesign and System Synthesis, CODES+ISSS 06, Proceedings, pages 130–135. ACM, 2006.

[73] R. Govindarajan, G.R. Gao, and P. Desai. Minimizing buffer requirements under rate-optimal schedule in regular dataflow networks. Journal of VLSI Signal Processing, 31(3):207–229, July 2002.

[74] Edgard de Faria Corrêa, Leonardo A.de P. e Silva, Flávio Rech, Wagner Luigi Carro, Fitting the Router Characteristics in NoCs to Meet QoS Requirements. Proceedings of the 20th annual conference on Integrated circuits and systems design 2007.

[75] Wiklund, D., Dake Liu, Dept. of Electr. Eng., Linkoping Univ., Sweden. SoCBUS: Switched Network on Chip for Hard Real Time Embedded Systems. Parallel and Distributed Processing Symposium, 2003. Proceedings. International, April 2003.

[76] S. Stuijk et al., Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs, in: DAC'06, Proc., ACM, 2006, pp. 899–904.

[77] S. Stuijk, T. Basten, M. Geilen, and H. Corporaal, "Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs," in DAC, 2007, pp. 777–782.

[78] E. Lee and D. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. IEEE Trans. on Comp. 36, 1, p. 24–35.

[79] Amir Hossein Ghamarian Timing Analysis of Synchronous Data Flow Graphs Eindhoven: Technische Universiteit Eindhoven, 2008. ISBN 978-90-386-1315-4.

[80] A.H. Ghamarian, M.C.W. Geilen, T. Basten, and S. Stuijk. Parametric throughput analysis of synchronous data flow graphs. In Conference on De-sign Automation and Test in Europe, DATE 08, Proceedings, pages 116–121. IEEE, 2008.

[81] M. Geilen, "Synchronous dataflow scenarios," *ACM Transactions on Embedded Computing Systems*, 2011.

[82] Julia Chuzhoy, Sanjeev Khanna, Polynomial Flow-Cut Gaps and Hardness of Directed Cut Problems in the Proceedings of the 39th Annual ACM Symposium on Theory of Computing, 2007.

[83] W. J. Dally and C. L. Seitz. 1987. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Trans. Comput.* 36, 5 (May 1987), 547-553.

[84] J. P. Li and M. W. Mutka, Priority based real-time communication for large scale wormhole networks", Int. Parallel Processing Symposium, Apr. 1994, pp. 433-438.

[85] Sunggu Lee and Jong Kim. 1996. Path selection for message passing in a circuit-switched multicomputer. *J. Parallel Distrib. Comput.* 35, 2 (June 1996), 211-218.

[86] Kandlur, D. D., and Shin, K. G. Traffic routing for multicomputer networks with virtual cut-through capability. *IEEE Trans. Comput.* C-41**,** 10 (Oct. 1992), 1257–1270.

[87] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," J. ACM, vol. 20, no. 1, pp. 46–61, 1973.

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

[89] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," in YCS164, Dept. Computer Science, University of York, 1991.

[90] N. C. Audsley, "On priority assignment in fixed priority scheduling," Inf. Process. Lett., vol. 79, no. 1, pp. 39–44, 2001.

[91] A. Zuhily and A. Burns, "Optimality of (dj)-monotonic priority assignment," Information Processing Letters, vol. 103, no. 1, pp. 247–250, 2007.

[92] M. Mutka, "Using rate monotonic scheduling technology for real time communications in a wormhole network," in Proceeding of the Second Workshop on Parallel and Distributed Real-Time Systems, 1994, pp. 194–199.

[93] J. Li and M. Mutka, "Real-time virtual channel flow control," J. Parallel and Distributed Computing, vol. 32, no. 1, pp. 49–65, 1996.

[94] S. L. Hary and F. Ozguner, "Feasibility test for real-time communication using wormhole routing," IEE Proceedings - Computers and Digital Techniques, vol. 144, no. 5, pp. 273–278, 1997.

[95] Zheng Shi and Alan Burns. 2008. Priority Assignment for Real-Time Wormhole Communication in On-Chip Networks. In *Proceedings of the 2008 Real-Time Systems Symposium* (RTSS '08).

[96] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," Real-Time Systems, vol. 30, no. 1-2, pp. 129–154, May 2005.

[97] Paul Emberson, Roger Stafford and Robert Davis, "Techniques For The Synthesis Of Multiprocessor Tasksets" In 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010), pp. 6-11, 2010.

[98] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS 2009), December 2009, pp. 398–409.

[99] R. Stafford. (2006) Random vectors with fixed sum. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/9700

[100] D. R. Karger and C. Stein. A new approach to the minimum cut problem. J. ACM, 43(4):601–640, 1996.

[101] C. Steiger, H. Walder and M. Platzner, "Heuristics for Online Scheduling Real-Time Tasks to Partially Reconfigurable Devices," in *FPL*, 2003, pp.575-584.

[102] D. B. Stewart and P. K. Khosla, "Real-Time Scheduling of Dynamically Reconfigurable Systems," in *IEEE International Conference on Systems Engineering*, 1991, pp. 139-142.

[103] Zheng Shi; Burns, A.; , "Real-Time Communication Analysis with a Priority Share Policy in On-Chip Networks," *Real-Time Systems, 2009. ECRTS '09. 21st Euromicro Conference on* , vol., no., pp.3-12, 1-3 July 2009 doi: 10.1109/ECRTS.2009.17

[104] http://noxim.sourceforge.net/

[105] Mingsong Lv; Ying Guo; Nan Guan; Qingxu Deng; , "RTNoC: A Simulation Tool for Real-Time Communication Scheduling on Networks-on-Chips," *Computer Science and Software Engineering, 2008 International Conference on* , vol.4, no., pp.102-105, 12-14 Dec. 2008. doi: 10.1109/CSSE.2008.275

[106] Finke, G., R.E. Burkard, and F. Rendl Quadratic Assignment Problems. Annals of Discrete Mathematics 31, 61-82, 1987.

[107] Sahni, S. and T. Gonzalez P-Complete Aproximation Problems. Journal of the ACM 23, 555-565, 1976.

[108] K. Tindell, A. Burns and A. J. Wellings, Analysis of hard real-time communications (1995) REAL-TIME SYSTEMS Volume 9, Number 2 147-171, 199.5

[109] B. Sprunt, L. Sha and J. Lehoczky, Scheduling sporadic and aperiodic events in a hard real-time system, Technical report no. CMU/SEI-89-TR-11, Carnegie-Mellon Software Engineering Institute, April 1989.

[110] Palencia Gutierrez, Gutierrez Garcia, Gonzalez Harbour, On the schedulability analysis for distributed hard real-time systems, Ninth Euromicro Workshop on Real-Time Systems, 1997. Proceedings.

[111] H. Hoang and M. Jonsson, "Switched real-time Ethernet in industrial applications - deadline partitioning," in Proc. Communications, pp. 76-81 Vol.1. Sept. 2003.

[112] Chenggui Zhao, Wenjun Xiao, Yong Qin, An EDF-Based, Hard-Real-Time Communication Engine for Distributed Embedded Systems. IFIP International Conference on Network and Parallel Computing , 2006.

[113] Robert I. Davis, Alan Burns, A survey of hard real-time scheduling for multiprocessor systems, ACM computing surveys Volume 43 Issue 4, October 2011 Article No. 35.

[114] N.C. Audsley, A. Burns, R.I. Davis, K. W. Tindell, A. J.Wellings, “Fixed Priority scheduling an Historical perspective”, Real-Time Systems 8(3). pp. 173-198. 1995.

[115] Kargahi M., Movaghar A. (2006), A method for performance analysis of earliest-deadline-first scheduling policy; Journal of Supercomputing 37(2); 197-222.

[116] Hong J., Tan X., Towsley D. (1989), A performance analysis of minimum laxity and earliest deadline scheduling in a real-time system; IEEE Transactions on Computers 38, 1736-1744.

[117] Benini, L. Bertozzi, D. Guerri, A. Milano, M. van Beek, P. Allocation and Scheduling for MPSoCs via Decomposition and No-Good Generation, Principles and Practice of Constraint Programming - CP 2005, Lecture Notes in Computer Science (107 - 121), 2005.

[118] Burns, A. Nicholson M. Tindell K. Zhang N. Allocating and Scheduling Hard Real-Time Tasks on a Point-to-Point Distributed System. Proceedings of the Workshop on Parallel and Distributed Real-Time Systems, pg. 11-12, 1993.

[119] Tindell, K. W. Burns, A. Wellings, A. J. Allocating hard real-time tasks: An NP-Hard problem made easy, Real-Time Systems Springer Netherlands Vol. 4 Issn: 0922-6443, 145-165, 1992. Doi: 10.1007/BF00365407

[120] Hladik P. Cambazard H. D´eplanche A. Jussien N. Solving a Real-Time Allocation Problem with Constraint Programming. Journal of Systems and Software 81, vol 1, 132-149, 2007, DOI : 10.1016/j.jss.2007.02.032

[121] Theodore W. Manikas and James T. Cain. Genetic Algorithms vs. Simulated Annealing: A Comparison of Approaches for Solving the Circuit Partitioning Problem. University of Pittsburgh, Dept. of Electrical Engineering, May 1996.

[122] ZHANG J, Chung H. and Lo W. L, “Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms”, IEEE Transactions on Evolutionary Computation vol.11, no.3, pp. 326–335, 2007.

[123] Agnieszka D, Zbigniew J, Zoltán J, Péter K, Dieter K, A Concurrent Implementation of Simulated Annealing and Its Application to the VRPTW Optimization Problem, Distributed and Parallel Systems, pp. 201 -209, The Kluwer International Series in Engineering and Computer Science, 2005

[124] Mahfoud S. Goldberg D. E. Parallel recombinative simulated annealing: a genetic algorithm, Parallel Computing Vol 21 pp. 1- 28, Jan 1995

[125] T.R.G. Nair and K. Sooda, Comparison of Genetic Algorithm and Simulated Annealing Technique for Optimal Path Selection In Network Routing. In Proceedings of CoRR. 2010.

[126] BARUAH, S. Task partitioning upon heterogeneous multiprocessor platforms. In Proceedings of the IEEE Real-Time Systems and Embedded Technology and Applications Symposium (2004).