BRNO UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUTIONARY DESIGN OF COLLECTIVE COMMUNICATION BASED ON PREDICTION OF CONFLICTS IN INTERCONNECTION NETWORKS

DOCTORAL THESIS

AUTHOR Ing. MILOŠ OHLÍDAL

SUPERVISOR Doc. Ing. JOSEF SCHWARZ, CSc.

BRNO 2007

Abstract

This work describes the application of a hybrid evolutionary algorithm to scheduling collective communications on the interconnection networks of parallel computers. To avoid contention for links and associated delays, collective communications proceed in synchronized steps. The minimum number of steps is sought for any given network topology, store-and-forward switching, minimum routing and given sets of sender and/or receiver nodes. Used algorithm is able not only to re-invent optimum schedules for known symmetric topologies such as hyper-cubes, but it can find schedules even for asymmetric or irregular topologies in case of general many-to-many collective communications. In most cases the number of steps reaches the theoretical lower bound for the given type of collective communication; if it does not, non-minimum routing can provide further improvement. Optimum schedules are destined for writing high-performance communication routines for application-specific networks on chip or communication libraries for general-purpose interconnection networks.

Keywords: collective communications, communication scheduling, evolutionary optimization, topology of interconnection network, multiprocessor, parallel processing, routing algorithm, store-and-forward switching technique, model of communication, prediction of conflicts

Acknowledgements

First and foremost I wish to thank to my supervisor Docent Josef Schwarz who has been a permanent source of stimulation and encouragement throughout my research. His advice and constructive criticism have helped me to progress towards the successful completion of my research work. I am also grateful to Professor Václav Dvořák for his inspiration and productive discussion. Special thanks also to my parents and the rest of my closest family for their understanding and support. Last, but not least, I would like to thank to my colleagues from the Faculty of Information Technology (FIT VUT Brno) with whom I have had the pleasure of working.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The demand for even greater computing power has never stopped. Although the performance of processors has doubled (approximately) every three years, the complexity of the software and the scale and solution quality of applications, have continuously driven the need and development for yet faster processors. However, the frequency of processors cannot be increased to the infinity. The creation of more powerful computers is through parallelization.

A parallel computer requires some kind of communications subsystem to interconnect the processors, memories and other devices. The specific requirements of these communication subsystems depend upon the architecture of the parallel computer. The simplest solution consists of connecting processors to memories and disk. Processors can be interconnected using the interface to local area networks.

1.1 Parallel Architectures

One of the fundamental taxonomies of computer architectures, proposed as early as 1966 by Flynn [1], but still useful today, is a model of categorizing all computers into four classes according to parallelism at the instruction stream and data stream levels. These categories combine single/multiple data streams and single/multiple instruction streams. From the four possible combinations, the only category, which emerged as the parallel architecture of choice for general-purpose multiprocessors, is MIMD (multiple instruction streams, multiple data streams) [2]. This is primarily due to two reasons:

- MIMDs offer flexibility. With the correct hardware and software support, MIMDs can function as single-user multiprocessors on high performance for one application, as multiprogrammed multiprocessors running many tasks simultaneously or in some combination of these functions.
- MIMDs can build on the cost-performance advantages of off-the-shelf microprocessors. In fact nearly all multiprocessors built today use the same microprocessors as those to be found in workstation and single-processor servers.

Existing MIMD multiprocessors fall into two classes, depending on the number of processors involved, which in turn dictate a memory organization and interconnection strategy. The first group called *centralized shared-memory* architectures usually does not have more than a few tens of processors. The second group, which consists of multiprocessors with physically distributed memory, scales to hundreds or thousands of processors.

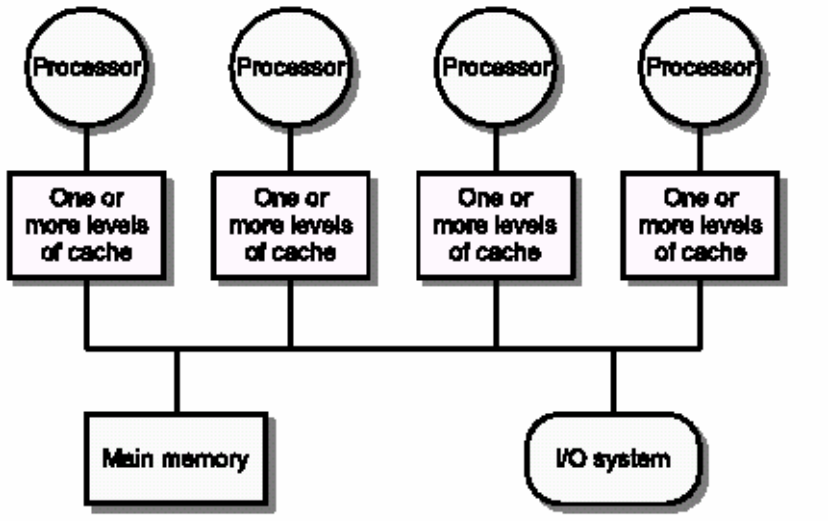


Figure 1: Basic structure of a shared-memory multiprocessor.

For multiprocessors with small processor counts, it is possible for the processors to share a single centralized memory and to interconnect the processors and memory by interconnection network. Due to a single main memory that a symmetric relationship to all processors and uniform access time from any processor these multiprocessors are often called *symmetric* (*shared-memory) multiprocessors* (SMPs). This style of architecture is sometimes called *uniform memory access* (UMA). Figure 1 shows the basic structure of these multiprocessors [2].

To support a large processor count memory in parallel architectures must be distributed among the processors rather than centralized: otherwise the memory system would not be able to support the bandwidth demands of a large number of processors without incurring excessively long access latency. The large number of processors raises the need for high bandwidth interconnections. The basic structure of these multiprocessors is illustrated in Figure 2.

There are two alternative architectural approaches that differ in the method used for communication data among processors in a distributed-memory system: single address space and multiple address spaces.

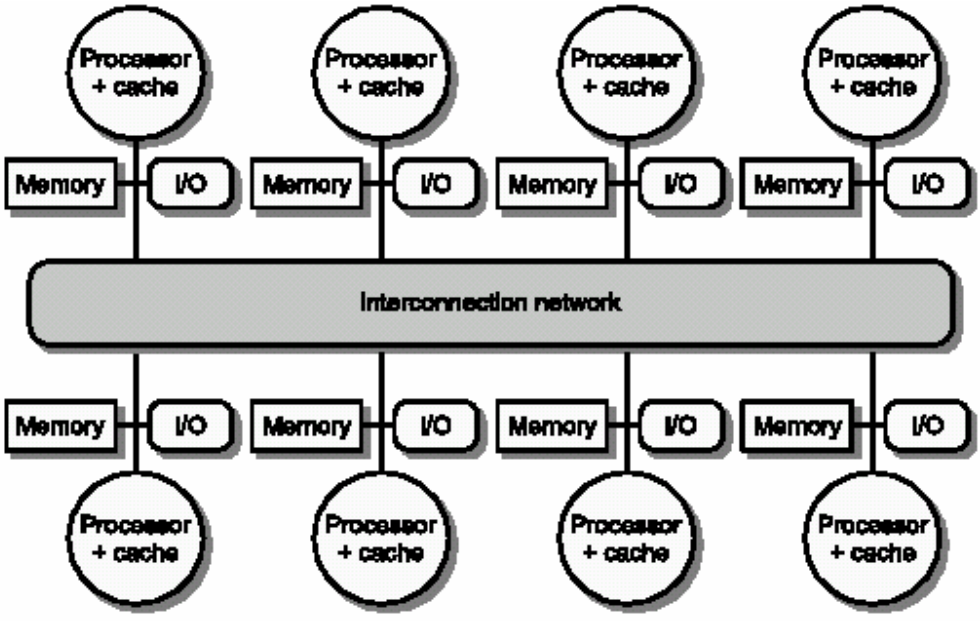


Figure 2: Basic structure of a distributed-memory multiprocessor.

Using the first method, physically separated memories can be addressed as one logically shared address space. Meaning that any processor can make a memory reference to any memory location, assuming it has correct access rights. These multiprocessors are called *distributed shared memory* (DSM) architectures. The term shared-memory refers to the fact that *address space* is shared, but it does not mean that there is a single centralized memory. In contrast to the symmetric shared-memory multiprocessors, also known as UMA (uniform memory access), the DSM multiprocessors use NUMA (nonuniform memory access), since the access time depends on the location of a data word in memory. Alternatively, the address space can consist of multiple private address spaces that are logically disjoint and cannot be accessed by the remote processor. In such multiprocessors the same physical address for two different processors refers to two different locations in two different memories. Each processor-memory module is essentially a separate computer. These parallel processors have been called multicomputers. It is now widely recognized that a *cluster of workstations* (COW) or *network of workstations* (NOW) offers a very attractive alternative to expensive supercomputers and parallel computer systems for high-performance computing. [3]

1.2 State of the Art

Processors with two cores are now here, and quad-core processors will very soon be available. In the coming years, the number of cores on a chip will continue to be increased, launching an era of vastly more powerful computers. These are the machines that will deliver teraflop performance with the efficient capabilities needed to handle tomorrow's emerging applications.
Why such a leap forward? These developments are necessary because incremental improvements in performance and capabilities would be unable to support real-time data mining across teraflops of data; artificial intelligence (AI) for smarter cars and appliances; virtual reality (VR) for modeling, visualization, physics simulation, and medical training; and other applications that are still on the edge of being science fiction. Also, data stores are becoming larger and more complex. In medical healthcare, a full-body medical scan already contains terabytes of information. In our homes, people are generating large amounts of data, including hundreds of hours of video, thousands of documents, and tens of thousands of digital photos that need to be indexed and searched. Teraflop

computing is the way to bring the massive compute capabilities of supercomputers to everyday devices, from servers, to desktops, to laptops.
With an increasing number of processor cores, memory modules and other hardware units in System on Chips (SoCs), the importance of communication within the system and with its related interconnection networks is steadily growing. These demands for increased communication within our computing systems has recently opened up research work in the Network on Chip (NoC) area, encompassing the interconnection/communication problem at all levels, from physical to the architectural to the OS and application level [4], [5].
Some embedded parallel applications, such as network or media processors, are characterized by independent data streams or by a small amount of inter-process communications [4]. However, many general-purpose parallel applications display a bulk-synchronous behavior: the processing nodes access the network according to a global, structured communication pattern. They can, for example, execute a personalized all-to-all information exchange, global synchronization, gather/scatter to/from one node, etc. The performance of these collective communications has a dramatic impact on the overall efficiency of parallel processing. Provided that computation times are known, as is usually the case in application-specific systems, the sole criteria thing for obtaining the highest performance is the duration of the various collective communications.
Bus-based synchronous communication structures in SoC, operating at several hundreds MHz, are no longer attractive, due to tight timing constraints and skew control [5]. Transition to point-to-point high speed networks, that happened on system boards (e.g. from PCI to PCI/Express), is taking place on SoCs, too. Much research and practical interest has recently focused on interconnection networks implemented on chip.
Currently, there are many different interconnection network topologies for general purpose multiprocessors, but new networks for specific parallel applications can still be created. Whereas the lower bounds on the time complexity of various group communications (in terms of required number of communication steps) can be mathematically derived for any network topology and its given communication pattern. Finding a corresponding schedule of communication is more difficult and, in some cases, not, as yet, an established matter.

The goal of this thesis is to create a general method based on evolutionary algorithm for optimal scheduling of a given collective communication and for arbitrary topologies. This optimal schedule has to be deadlock-free (some messages cannot advance toward their destination because the resources are full) and conflict-free (only one message can be sent via given channel in the same direction at the same time) and also has to be executed in the shortest possible time, i.e. in minimal number of communication steps.

1.3 Overview of the Following Chapters

The opening of this chapter presented an introduction to parallel computation, followed by a short description of widely used parallel architectures in section 1.1. An overview of a state of art in the multiprocessors area was given and, finally, the goal of this work was introduced.

All the needed terms of interconnection networks area are explained and defined in chapter 2, which is necessary for a full understanding of our proposed method. At the end of this chapter, the view of multiprocessor chips' future is presented.

The third chapter describes in detail and explains switching techniques. In the fourth chapter the most widespread routing methods are presented, which are utilized with smaller or higher variations in almost all proposed routing algorithms to this time.

The fifth chapter is dedicated to a detailed description of collective communication and models of communication. The mathematical approach is shown to calculate a lower bound of time complexity of optimal schedule, which can be utilized for some type of interconnection networks.

Chapter 6 deals with the design of a new hybrid evolutionary method, and simultaneously our new method is compared with other similar hybrid evolutionary techniques in this chapter. Firstly, quality and efficiency of the proposed new method was tested on the traveling salesman problem and after verification of its quality, this method was utilized in the solving of a real problem from practice, i.e. the scheduling of collective communications for arbitrary interconnection networks. Chapter 7 is also dedicated to this scheduling problem. The method was developed based on prediction of conflicts to design of optimal schedule. From the achieved results it can be stated that this method is very effective and it is able to schedule arbitrary topologies for arbitrary communication.

Finally, chapter 8 summarizes the main contributions of the thesis and proposes possible directions for future research.

Appendix A contains the pseudo-code of our proposed routing algorithm. All investigated topologies are presented in appendix B. The examples of OAB, AAB, OAS, AAS and MNB collective communication schedules are illustrated in the remaining appendixes.

Chapter 2

Interconnection Networks

An interconnection network is a programmable system that enables fast data communication between the components of a digital system. The network is programmable in the sense that it enables different connections at different points in time. The network is a system because it is composed of many components: buffers, channels, switches, and controls that work together to deliver data.

The functional view of an interconnection network is illustrated in Figure 3. Six terminal nodes are connected to the network with *bidirectional channels*. When a source terminal (say T3) wants to communicate with a destination terminal (say T5), it sends data in the form of a *message* into the network and the network delivers the message to T5. Using the same resources, the network can deliver the above message in one cycle (cycle expresses time), and a different message in the next cycle. [6]

Interconnection networks are used in almost all digital systems that are large enough to have two components to connect. The most common applications of interconnection networks are in computer systems and communication switches. In computer networks, they connect processors to memories and input/output (I/O) devices to I/O controllers. They connect input ports to output ports in communication switches and network routers.

Interconnection networks may also connect sensors and actuators to processors in control systems, host and disk nodes in I/O networks and on-chip cores in chip multiprocessors.

Today, all high-performance interconnections are realized by point-to-point interconnection networks rather than buses, and many systems that historically have been bus-based are being converted to the networks systems every year. The demand for interconnection performance is increasing with processor performance and network bandwidth. As a result, buses have been unable to keep up with the bandwidth demand, and point-to-point interconnection networks, which operate faster than buses while also offering concurrency, are rapidly taking over. [7]

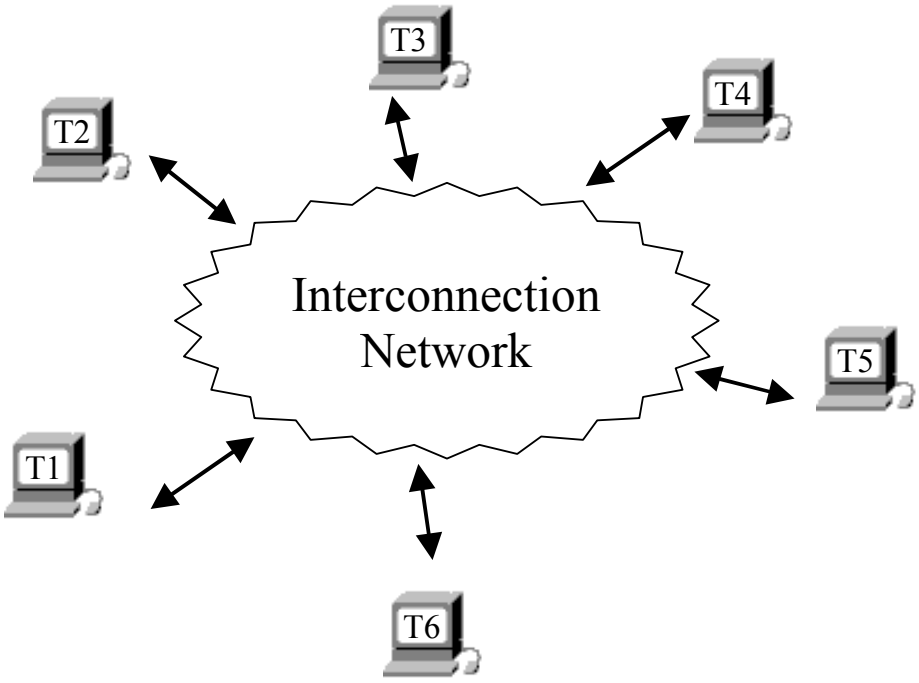


Figure 3: The functional schema of an interconnection network. Terminals T1 through T6 are connected to the network with bi-directional channels.

The performance of most digital systems today is limited by their communication or interconnection, not by their logic or memory. Hence, it is imperative that the underlying interconnection network performs efficiently in order to improve the efficacy of the entire system. For instance, in a computer system, the interconnection network between processor and memory determines key performance factors such as the memory latency and

memory bandwidth. The performance of the interconnection network in a communication switch largely determines the capacity (data rate and number of ports) of the switch.

2.1 Basics in Interconnection Networks

A key to the efficiency of interconnection networks comes from the fact that communication resources are shared. Instead of creating a dedicated channel between each terminal pairs, the interconnection network is implemented with a collection of shared router nodes connected by shared channels. The connection pattern of these nodes defines the network's *topology*. A message is then delivered between terminals by making several *hops* across the shared channels and nodes from its source terminal to its destination terminal.
Once a topology has been chosen, there can be many possible paths (sequences of nodes and channels) that a message could take through the network to reach its destination. *Routing* determines which of these possible paths a message actually takes. A good choice of paths minimizes their length, usually measured as the number of nodes or channels visited, while balancing the demand placed on the shared resources of the network. The length of a path obviously influences latency of a message through the network and the demand or *load* on resource is a measure of how often that resource is being utilized. If one resource becomes over-utilized while another sits idle the total bandwidth of messages being delivered by the network is reduced.
Flow control dictates which messages gets access to particular network resources over time. This influence of flow control becomes more critical as the utilization of the resource increases and good flow control forwards packets with minimum delay and avoids idling resources under high loads. [7]
A detailed description and concept definition will be presented in the following chapters.

2.1.1 Packet and Message

It is necessary to distinguish between *messages* and *packets* in a network. A message is the logical unit of data transfer provided by network interfaces. Its size is limited only by the user memory space. Because messages do not always have a bounded length, they are often broken into smaller packets for handling within the network. Packets are fixed-size

smallest unit of communication containing *routing information* (e.g., a destination address) and sequencing information in its header. All data contained within a packet follows the same *route* through the network and packets are reassembled into messages at the destination. Its size is of the order of hundreds or thousands of bytes or words, consisting of *header flits* and *data flits*. The structure of a message is illustrated in the Fig. 4 [8].
Packets are divided into fixed length flits (flow control digit) to simplify the management and allocation of resources. A flit is the smallest unit of resources allocation in a router. It may be divided further into phits (physical digit) for handling by the router datapath.

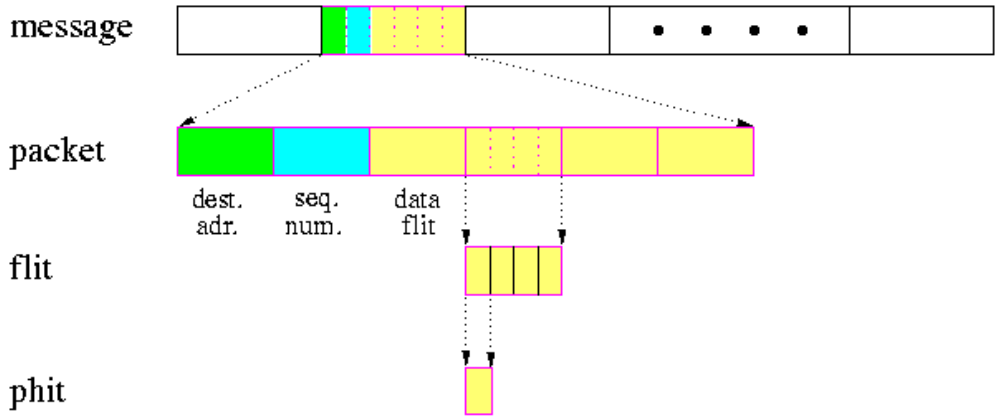


Figure 4: The structure of message.

2.1.2 Topology

Interconnection networks are composed of a set of shared router nodes and channels. The *topology* of the network refers to the arrangement of these nodes and channels. The topology of an interconnection network is analogous to a roadmap. The channels (the roads) carry packets (the cars) from one route node (the intersection) to another. For example, the network shown in Fig. 5 consists of 8 nodes, each of which is connected to 8 channels / 4 links. One link between nodes consists of 2 channels, 1 to neighbor and 1 from neighbor.

- **Channels and Nodes**

The topology of an interconnection network is specified by a set of nodes N connected by a set of channels C. Each channel $c = (x, y) \in C$ connects a source x to a destination node y, where $x, y \in N$. We denote the source node of a channel c as s_c and the destination as d_c.

A channel $c = (x, y) \in C$ is characterized by its latency t_c or t_{xy} the time required for a bit to travel from x to y. For most channel the latency is directly related to the physical length of the channel $l_c = vt_c$, by a propagation velocity v.

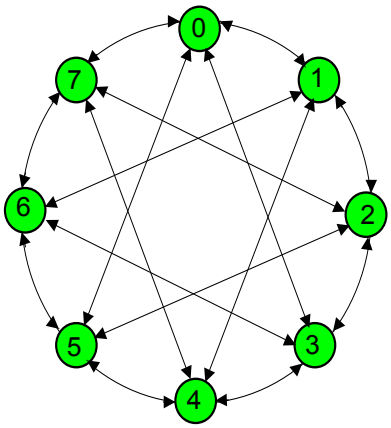


Figure 5: Example of network topology. Illustrated topology is K-ring.

Each node x has a channel set $C_x = C_{Ix} \cup C_{Ox}$, where $C_{Ix} = \{c \in C \mid d_c = x\}$ is the input channel set and $C_{Ox} = \{c \in C \mid s_c = x\}$ is the output channel set. [7]

- **Node degree**

A node degree is the number of channels entering and leaving node.

The degree of x is $d_x = |C_x|$ which is the sum of the in degree $d_{Ix} = |C_{Ix}|$ and the out degree $d_{Ox} = |C_{Ox}|$, where $x \in N$.

- **Bisections**

A bisection of a network is a *cut* that partitions the entire network nearly in half:

$$|N_2| \leq |N_1| \leq |N_2| + 1 \tag{2.1.1}$$

The channel bisection of a network (bisection width) B_C is the minimum channel count over all bisections of the network:

$$B_C = \min_{bisections} |C(N_1, N_2)| \tag{2.1.2}$$

A cut of a network $C(N_1,N_2)$ is a set of channels that partitions the set of all nodes N into two disjoint sets N_1 and N_2. Each element of $C(N_1,N_2)$ is a channel with a source N_1 and destination N_2. The total bandwidth of the cut is:

$$B(N_1, N_2) = \sum_{c \in C(N_1, N_2)} b_c \tag{2.1.3}$$

where b_c is bandwidth of channel.

The bisection bandwidth of network B_B is the minimum bandwidth over all bisections of the networks [7]:

$$B_B = \min_{bisections} B(N_1, N_2) \tag{2.1.4}$$

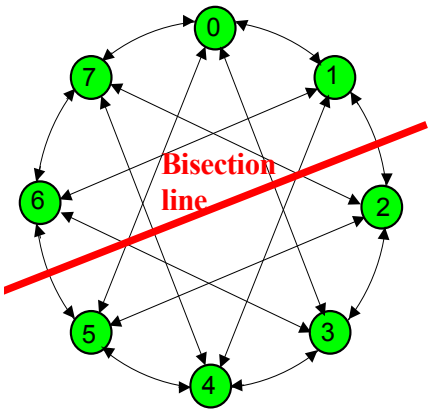


Figure 6: The bisection $Bc = 16$ and degree $d = 8$ of K-ring topology. Each edge represents two unidirectional channels going in opposite directions.

- **Paths**

A path in a network is an ordered set of channels $Pa = \{c_1, \ldots, c_n\}$. Paths are also referred to as *router*. The source of a path is s_{c1}. Similar, the destination of a path is d_{cn}. The length or *hop count* of a path is $|Pa|$. If, for a particular network and its routing function at least one path exists between all source-destination pairs, it is said to be *connected*.
A *minimal path* from node x to node y is a path with smallest length $H(x, y)$ connecting these two nodes.
The *diameter* of a network D is the largest minimal length over all pairs of terminal nodes in the network. [7]

$$D = \max_{x,y \in N} H(x, y) \tag{2.1.5}$$

- **Direct and Indirect Network**

A network node may be a *terminal node* that acts as a source/destination and a *switch node* (router) that forwards packets from input ports to output ports. In a direct network every node in the network is both a terminal and a switch, such as the K-ring topology of Fig. 6. In an indirect network a node is either a terminal or a switch. It cannot serve both functions. In a direct network packets are forwarded directly between terminal nodes while in an indirect network they are forwarded indirectly by means of dedicated switch nodes. Every direct network can be redraw as an indirect network by splitting each node into separate terminal and switch node, as illustrated Figure 7.

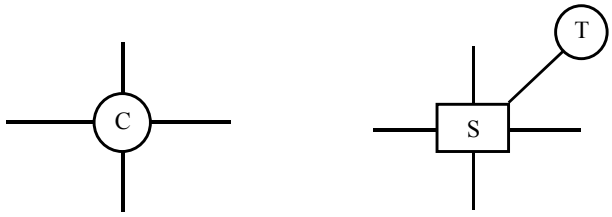


Figure 7: Node of direct network and node of indirect network – it consists of a terminal node and a switch node.

In some early networks the switching function was implemented in software running on the terminal CPU and buffering was performed using the terminal computer's memory

[9], [10]. Software switching is very slow and demanding upon the terminal's resources and is thus is rarely used today.
A potential advantage of an indirect network is that more than one terminal node can be connected to one switch node.
This manner of creating a topology with a higher number of terminal nodes. It is a simpler process to implement routing in these fat node topologies than in a topology with the same number of node with a single terminate node. The fat node topologies have far fewer channels and the interconnection is simpler than the topology with single terminal node. An example of fat node topology is given in the Fig. 8.

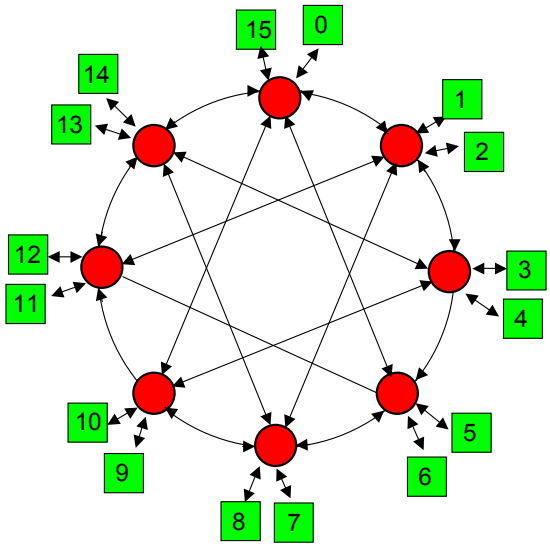


Figure 8: The fat node topology with two terminal nodes connected to one switch node.

- **Symmetry and Regularity**

The symmetry and regularity of a topology play an important role in routing, as will be discussed in a later section. A network is vertex-symmetric if there exist automorphism that maps any node *a* into another node *b*. In a vertex-symmetric network the topology looks the same from the point-of-view of all the nodes. In a regular network the nodes of the topology have the same degree. Symmetry and regularity can simplify routing because all nodes share the same point-of-view of the network and therefore can use the same directions to route to the same relative position.

- **Throughput**

The *throughput* of the network is the data rate in bits per second that are delivered to the destination node of the network. Throughput is a property of the entire network which depends on the routing (details are presented in chapter 2.1.3) and flow control (see chapter 2.1.5) as much as it depends upon the topology. *Ideal* throughput can be achieved by perfect flow control and routing. In this case the routing perfectly balanced the load over alternative paths in the network and the flow control left no idle cycles on the saturated channels. *Maximal* throughput occurs when each channels in the network becomes saturated.

- **Traffic**

Traffic is the amount of information delivered per time unit. This amount of information is often modeled by a *traffic pattern* that determines how a packet travels between a particular source-destination pair and an arrival process. Historically, several of these patterns are based on communication patterns that arise in particular applications.

- **Latency**

The *latency* of a network is the time required for a packet to traverse the network from the time the head of the packet arrives at the input port to the time the tail of the packet departs the output port. We separate latency T into two components:

$$T = T_h + T_s \tag{2.1.6}$$

The *head latency* T_h is the time required from the head of the message to traverse the network and the *serialization latency*:

$$T_s = L/b \tag{2.1.7}$$

is the time required for the tail to catch up – the time for a packet of length L to cross a channel with bandwidth b.

Latency depends not only on a topology but also on routing, flow control and the design of the router.

The head latency is the sum of two factors determined by the topology: router delay T_r and time of flight T_w. Router delay is the time spent in the routers, whereas time of flight is the time spent on the wires. Combining these components gives the following expression for latency [7]:

$$T = T_r + T_w + L/b \tag{2.1.8}$$

2.1.3 Routing

The routing method employed by a network determines the path taken by a packet from a source terminal node to a destination terminal node. A route or path is an ordered set of channels $Pa = (c_1, \ldots, c_n)$ where the output node of channel c_i equals the input node of channel c_{i+1}. The source is the input to channel c_1 and the destination is output of channel c_n. In some networks there is only one path from the source to its destination, whereas in others, which are much more common, there are many possible paths. When there are many paths, a good routing algorithm makes the decision which path should be used and in which time [7].

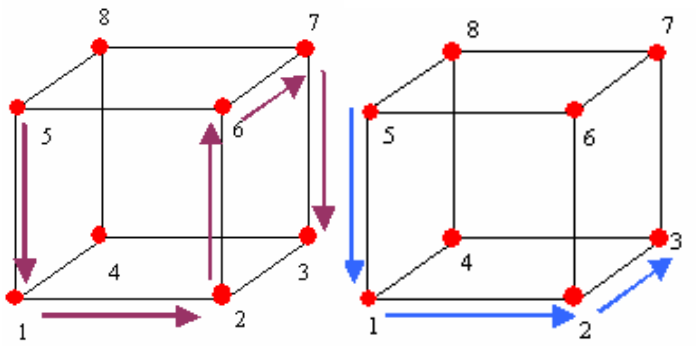


Figure 9: Two ways of routing from node 5 to node 3 in the hyper-cube. (a) A non-minimal route requires more than the minimal path length. (b) A minimal routing using minimal path length.

Figure 9 shows two different paths from node 5 to node 3 in the interconnection network. Routing (a) illustrates a non-minimal routing in the situation, taking 5 hops. In routing (b) one of the shortest paths from 5 to 3 has been chosen and this route is minimal. (There are six shortest paths with length 3.)

2.1.4 Routing Model

The architecture of a router, where the packets can be stored for a time, is shown in Figure 10 and is comprised of the following major components:

- *Buffers* – these are first-in first-out (FIFO) buffers for storing packets in transit. In the model shown in Figure 10 a buffer is associated with each input physical channel and each output physical channel. In alternative design buffers may be associated only with inputs (input buffering) or outputs (output buffering).
- *Packet memory* – for storing incoming packet from input buffers. Packets are stored here for the necessary time. The storage time for individual packets differs, to ensure the best throughput of interconnection network.
- *Switch* – this component is responsible for connecting router input buffers to router output buffers. High-speed routers will utilize crossbar networks with full connectivity. Lower-speed implementations may utilize networks that do not provide full connectivity between input buffers and output buffers.
- *Routing and scheduling logic* – this component implements the routing and scheduling of incoming packets. It decides when and which packet will be chosen from packet memory. It selects input channel to switch and output channel from router for a chosen packet and accordingly sets the switch. This scheduling avoids the situation in which multiple packets simultaneously request the same output link. It causes some packets stay in the packet memory longer than others but how long is dependent upon the routing algorithm, which has been implemented in this component. Detailed descriptions of the routing algorithms are illustrated in chapters 4 and 7.
- *Link controllers (LCs)* – the flow of packets across the physical channel between adjacent routers is implemented by the link controller. The link controllers on either side of a channel coordinate the transfer units of flow control.
- *Processor interface* – this component simply implements a physical channel interface to the processor rather than to an adjacent router. It consists of one or more injection channels from the processor and one or more ejection channels to the processor.

When a packet first arrives at a router it must be examined to determine the output channel over which the packet is to be forwarded. This is referred to as the *routing delay* and typically includes the time to set the switch. Once a path has been established through a router by the switch, of critical interest is the rate at which the packets can be forwarded through the switch. This rate is determined by the propagation delay through switch and the signaling rate for synchronizing the transfer of data between the input and output buffers. This delay has been characterized as the *internal flow control* latency [11]. The

delay across the physical links is referred to as the *external flow control* latency. The routing delay and flow control delays collectively determine the achievable packet latency through switch and determine the network throughput. [12]
The detail description of latency depending on switching technique is presented in chapter 3.

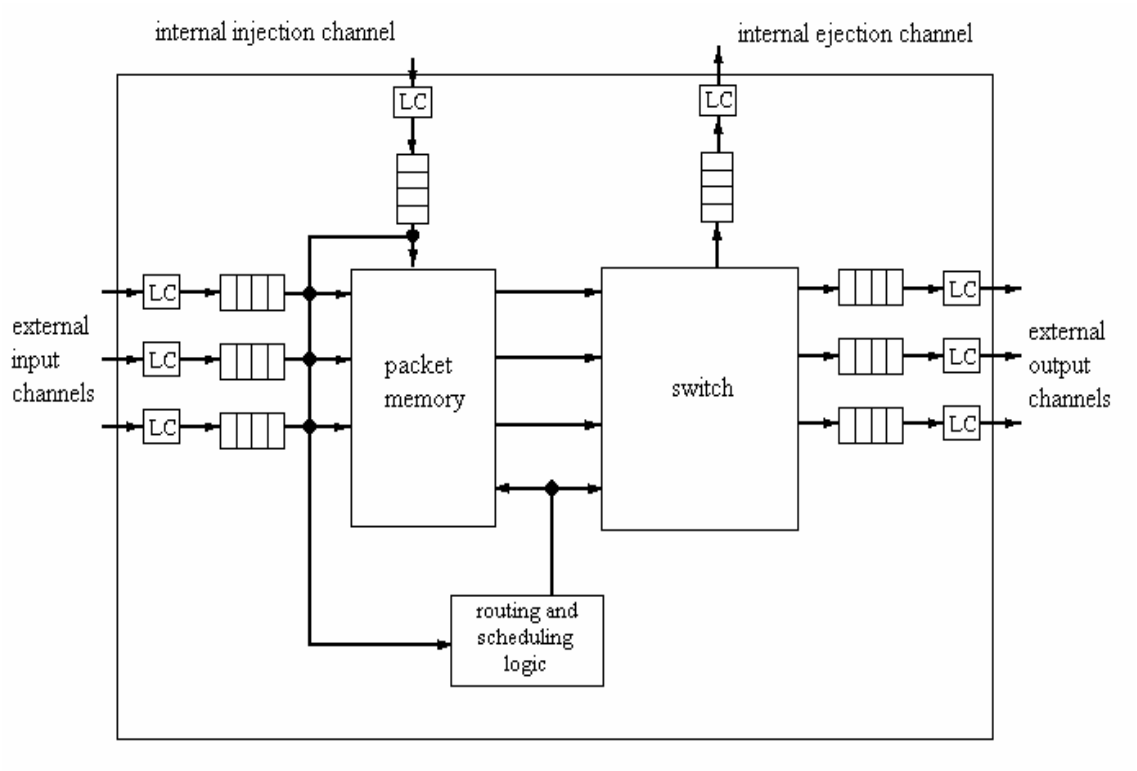


Figure 10: Router model with ability to store packets for a time.

2.1.5 Flow Control

Flow control manages the allocation of resources to packets as they progress along their route. The key resources in most interconnection networks are the channels and the buffers. We have already seen the role of channels in transporting packets between nodes.

Buffers are storage implemented within the nodes and allow packets to be held temporally at the nodes. For simplicity we can say that: the topology determines the roadmap, the routing method manages the car and the flow control controls the traffic lights, determining when a car can advance over the next stretch of road (channels) or when it must pull off into a parking lot (buffer) to allow other cars to pass/overtake.
To realize the performance potential of the topology and routing method, the flow control strategy must avoid resource conflicts that can hold a channel idle. For example, it should not block a packet that can use an idle channel because it is waiting on a buffer held by a packet that is blocked on a busy channel.
A good flow control strategy is *fair* and avoids *deadlock*. An unfair flow control strategy can cause a packet to wait indefinitely. Deadlock is a situation that occurs when a cycle of packets is waiting for one another to release resources.
We often describe a flow control method by using a time-space diagram, such as those shown in Figure 11. The figure shows a time-space diagram for (a) store-and-forward flow control and (b) wormhole flow control.

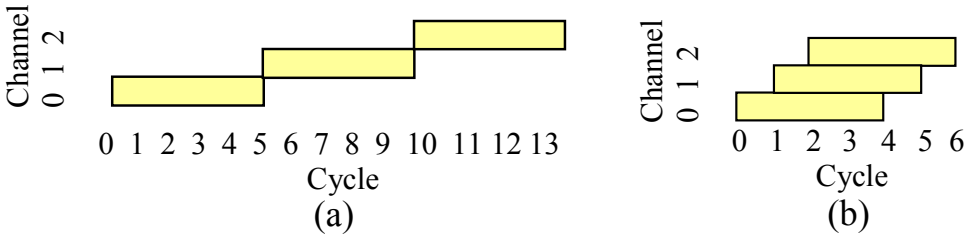


Figure 11: Time-space diagram shows two flow control methods. (a) Store-and-Forward flow control – a packet is completely transmitted across one channel before transmission across the next channel is started. (b) Wormhole flow control – a packet transmission over the channels is pipelined.

In both diagrams time is shown on the horizontal axis and space is shown on the vertical axis. Time is expressed in cycles and space is shown by listing the channels used to send the packet. As seen in Fig. 11 the choice of flow control techniques can significantly affect the latency of a packet through network.

More details of store-and-forward and wormhole flow control are described in chapter 3. The problems of deadlock and livelock are dealt with in the next section.

2.2 Deadlock, Livelock and Conflict

In interconnection networks packets usually travel across several intermediate nodes before reaching the destination. However, it may happen that some packets are not able to reach their destinations because they are waiting on one another to release resources (channels and buffers). Consider the situation shown in Figure 12.

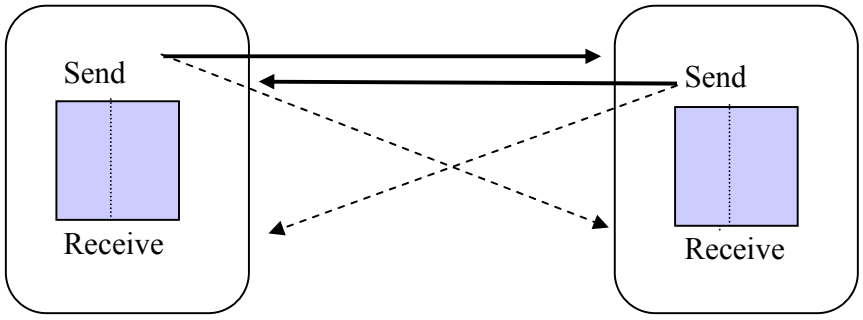


Figure 12: Deadlock in communication. Both partners start to send their packets while at the same time they are waiting for confirmation of receiving these packets. Both are sending and, therefore, they cannot receive.

Two partners need to communicate with each other. Both start to send their packets and both are waiting for confirmation of receiving these packets. However, because both are sending and neither can receive, *deadlock occurs*. A buffer deadlock occurs when some packets cannot advance toward their destination because the buffers required by them are full. Deadlock is catastrophic situation within a network. After a few resources are occupied by deadlock packets, other packets block on these resource, paralyzing the network operation. To prevent this situation, networks must either use deadlock avoidance [13] (methods that guarantee that a network cannot deadlock) or deadlock recovery [14] (in which deadlock is detected and correct, e.g. packet is killed and sends again). Almost all

modern networks use deadlock avoidance, usually by imposing an order on the resources in question and insisting that packets acquire these resources in order.
A different situation arises when some packets are not able to reach their destination, even if they never block permanently. A packet may be traveling around its destination node, never reaching it because the channels required to do so are occupied by other packets. This situation is known as *livelock*. It can only occur when packets are allowed to follow non-minimal paths.
The last situation, which necessary to mention, is referred to as *conflict*. Conflict is the situation when two source nodes want to use the same channel in the same direction and at the same time. During a conflict situation it is not known what happens to packets, which were not assigned to channel, whether they reach their destination and when, and if they continue their path or will be sent from the source node again.

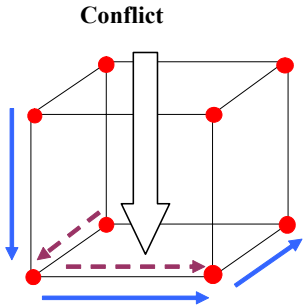


Figure 13: Two source nodes want to use the same channel in the same direction and at the same time – conflict situation has occurred.

2.3 Case Study: Intel Tera-scale

In IDF Fall 2006 Intel announced their research prototype of their possible future processor's architecture. In contrast to present architectures, this prototype is 8 x 10 of the same computing cores on one chip. The concentration of possible computing performance with eighty of these processors on wafer is hardly imaginable. The exact dimensions of this chip are 22 x 13.75 mm. The details are illustrated in Figure 14 [15]. This chip achieves

the possible computing performance of 1 TeraFLOPS (Floating Point Operations Per Second) with a frequency of 3.1 GHz. This computing performance is with inaccessible using currently available architectures and SMP (Symmetric Multiprocessing).

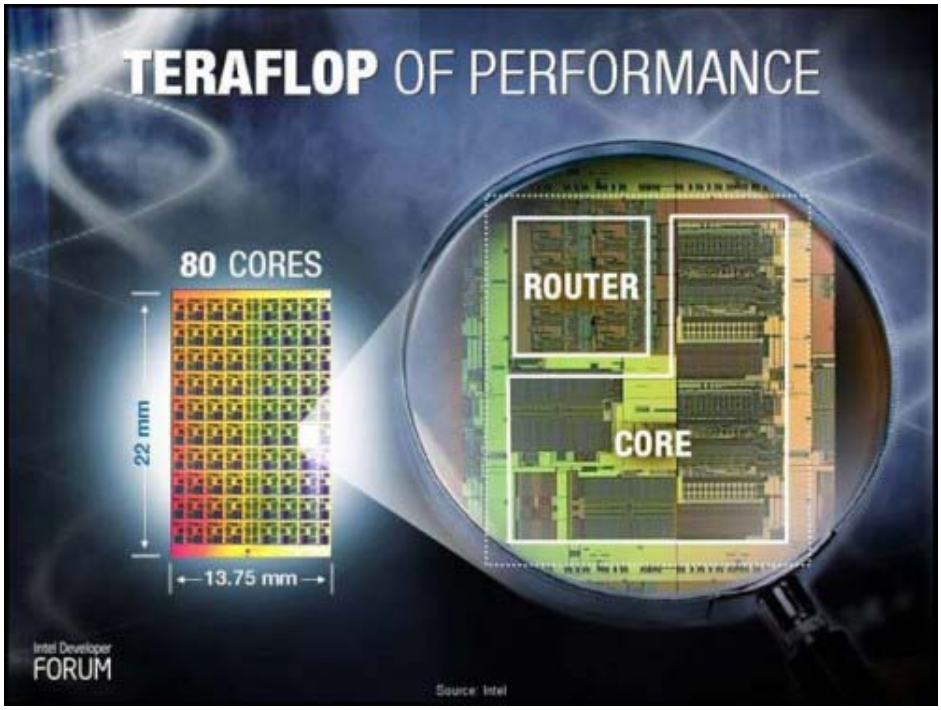


Figure 14: Prototype of multiprocessor Intel "Tera-scale" with eighty cores.

This prototype Intel has given a suitable name, "Tera-scale" and Platform 2015 at this stage in its development. The prototype has a number of developmental features, change in architecture and the system of communication in highly parallel systems, which take into consideration hundred's of cores.

The basic system of the "Tera-scale" will be a highly parallel architecture with many cores with emphasis on the support of virtualization and security. The target requirements have also insisted on lower consumption, high efficiency and the support of accelerators.

The increase in the number of cores has encoutered a number of problems (mainly in consideration of available manufacturing technology and level of integration). This architecture allows scalability not only on the multiprocessor level and the number of cores,

but also with connection of more these multiprocessors or whole systems based on "Tera-scale" principle with the help of high-speed optical interface, see Fig. 15.

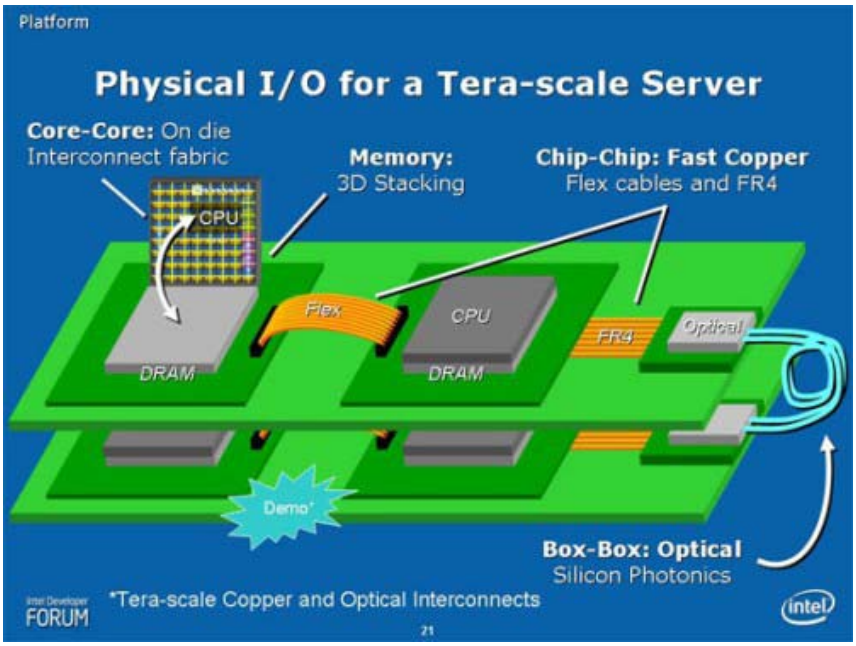
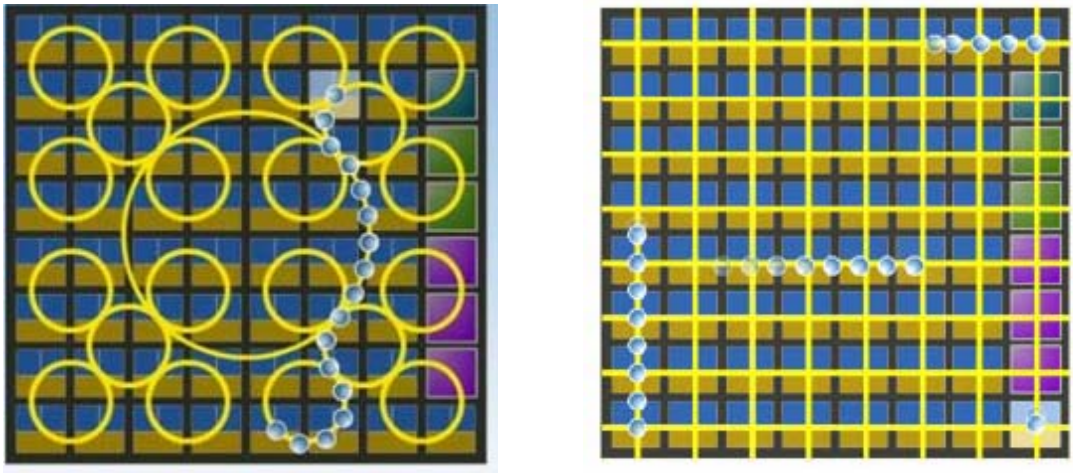


Figure 15: Scalability is ensured on all levels. In this figure, multiprocessor is denoted as CPU.

The architectures of processors in the next few years will focus mainly on efficiency and lower consumption, evidently in the form of power management of computational module. Cores, which are be not being used, will be slept. In the case overheating of the computational core, redirection to a different core will be realized.
Due to scalability and performance of highly parallel systems a vital feature of this system design is speed of communication between cores – both on the level of composite computational modules and whole chip (of course on the level of packets). The complex new architecture also increases the complexity of communication and distribution of data between the cores. Intercommunication using traditional high frequency interconnections will be replace with laser in the course of time. The new technology of hybrid laser implemented on the level of single core, will brings useful scalability and lower price. The last achievable bound is 40 Gbps (Gigabit per second) on the single hybrid laser. Twenty-

five such lasers with multiplexor give other “Terabps” [16]. Communication on the level of core, more multiprocessors on a motherboard or connection of more such systems would not create any significant problem, see Fig. 15.
The ring topology and 2D-Mesh topology are considered mainly for future platforms, see Fig. 16. Mesh topology allows good scalability with a large number of cores and ring topology offers the advantage of less skipping to the required destination [15].



(a) (b)

Figure 16: (a) Ring topology and (b) 2D-Mesh topology of Intel “Tera-scale”.

Chapter 3

Switching Techniques

This chapter focuses on the switching techniques that are implemented within the network routers. These techniques differ in several respects. The switching techniques determine when and how internal switches are set to connect router inputs to outputs and the time at which message components may be transferred along these paths. These techniques are coupled with flow control mechanisms for the synchronized transfer of units of information between routers and through routers in forwarding messages through the network. Implementation of the switching differs in their relative timing, that is, when one operation can be initiated relatively to the occurrence of the other.

For the purpose of comparison, for each switching technique we will consider the computation of the base latency of an L-bit message in the absence of any traffic. The phit size and flit size are assumed to be equivalent and equal to the physical data channel width of W bits. The routing header is assumed to be 1 flit, therefore the message size is $M = L + W$ bits. A router can make a router decision in t_r seconds. The physical channel between two routers operates at B Hz, that is, the physical channel bandwidth is BW bits per second. The propagation delay across a channel is denoted by t_w. Once a path has been set up through the router, the intrarouter delay or switching delay is denoted by t_s. The router internal data paths are assumed to be matched to the channel width of W bits. Therefore

in t_s seconds a W bit flit can be transferred from input of the router to the output. The source and destination processor are assumed to be D_d links apart [12]. The relation between these components as they are used to compute message latency is shown in Fig. 17 [8].

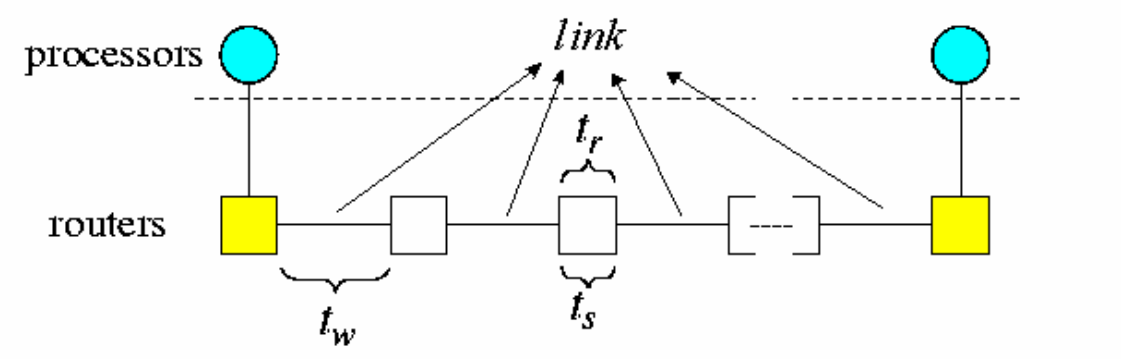


Figure 17: View of the network path for computing switching latency.

3.1 Circuit Switching

In circuit switching a physical path from source to destination is reserved prior to the transmission of the data. This is realized by injection the routing header flit into the network. This routing probe contains the destination address and some additional control information. The routing probe progresses toward the destination reserving the physical links as it is transmitted through intermediate routers. When the probe reaches the destination, a complete path has been set up and an acknowledgment is transmitted back to the source. The message contents may now be transmitted at the full bandwidth of the hardware path. A time-space diagram of the transmission of a message over three links is shown in Fig. 18.
Circuit switching is generally advantageous when messages are infrequent and long. The disadvantage is that the physical path is reserved for the duration of the message and may block other messages.

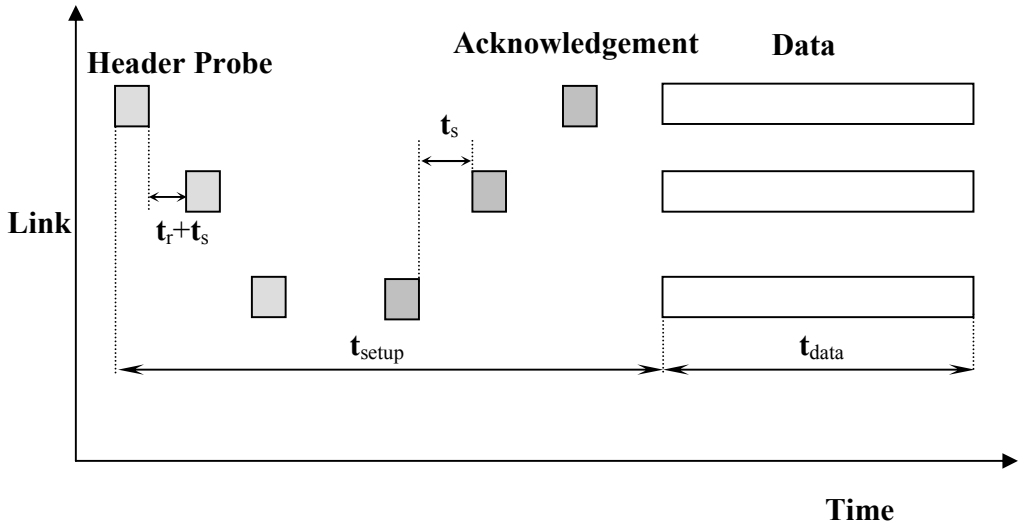


Figure 18: Time-space diagram of a circuit-switched message.

The base latency of a circuit-switched message is determined by the set up time of a path and the subsequent time the path is busy transmitting data [12]. We can express the base latency of a message of length *M* as follow [8]:

$$t_{circuit} = D_d\,(t_r + 2(t_s + t_w)) + Mt_w \tag{3.2}$$

3.2 Store and Forward Switching

In store-and-forward switching the message is partitioned and transmitted as fixed-length packets. The first few bytes of packet contain routing and control information and are referred to as the *packet header*. Each packet is individually routed from source to destination. A packet is completely buffered at each intermediate node before it is forwarded to the next node. This is the reason why this switching technique is referred to as *store-and-forward* (SF) switching. This technique is also alternatively called *packet switching*. The header information is extracted by the intermediate router and used to determine the output link over which the packet is to be forwarded. A tome-space diagram of progress of a packet across three links is shown in Fig. 19. From the figure it can be seen that the

latency of a packet is proportional to the distance between the source and destination nodes.

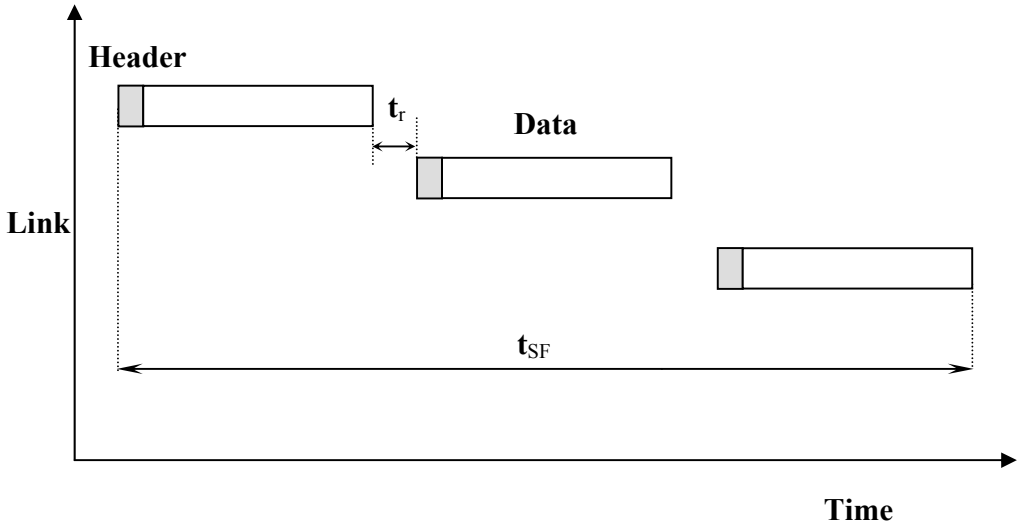


Figure 19: Time-space diagram of a store-and-forward-switched message.

Packet switching is advantageous when messages are short and frequent. Unlike circuit switching, where a segment of a reserved path may be idle for a significant period of time, a communication link is fully utilized when there are data to be transmitted. In addition, every packet must be routed at each intermediate node. It is evident that the storage requirements at the individual router nodes are extensive if packets are large and multiple packets must be buffered at a node [12].
The base latency of a store-and-switched message of length *M* can be computed as follow [8]:

$$t_{SF} = D_d\,(t_r + (t_s + t_w)M) \tag{3.3}$$

3.3 Virtual Cut-Through Switching

The *virtual cut-through switching* (VCT) technique allows packets to be forwarded although they are not completely stored in the current buffer. The router can start forwarding the header and following data bytes as soon as routing decisions have been made and the output buffer is free. In fact the message does not even have to be buffered at the output and can cut through to the input of the next router before the complete packet has been received at the current router. The message is effectively pipelined through successive switches. If the header is blocked in a busy output channel, the complete message is buffered at the node. Figure 20 illustrates a time-space diagram of a message transferred using virtual cut-through switching where the message is blocked after the first link. However, from the figure it can be seen that the message is successful in cutting through the second router and across the third link [12], [17].

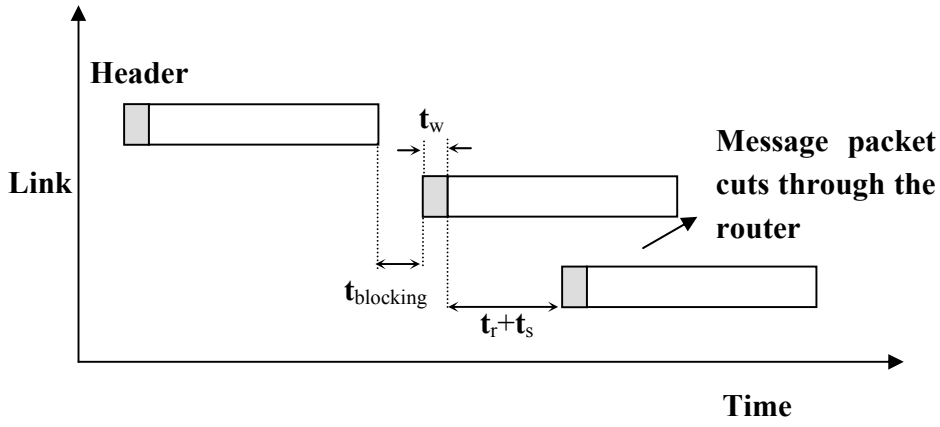


Figure 20: Time-space diagram of a virtual cut-through switched message. ($t_{blocking}$ = waiting time for a free output channel.)

The base latency of a message that successfully cuts each intermediate router can be computed as follow [8]:

$$t_{VCT} = D_d\,(t_r + t_w + t_s) + max(t_w, t_s)M \tag{3.4}$$

This model assumes that there is no time penalty for cutting through a router if the output buffer and output channel are free. Note that only the packet header routing delay as well as switching and inter-router latency. This is because the transmission is pipelined and the switch is buffered at the input and output. Once the header flit reaches the destination, the cycle time of the pipeline of packet flits is determined by the maximum of the switching time and inter-router latency. It can be assumed that channels have both input and output buffers. In the case of input buffering only, for example, we would have t_w+t_m instead of $max(t_w\ t_m)$.

3.4 Wormhole Switching

In *wormhole switching* message packets are also pipelined through the network. However, the buffer requirements within the routers are substantially reduced over the requirements for VCT switching. A message packet is broken up into flits. The message is pipelined through the network at the flit level and is typically too large to be completely buffered within a router. It means that a blocked message occupies buffers in several routers at any instant in time. The time-space diagram of a wormhole-switched message is shown in Fig. 21. Routing delays and intrarouter propagation of the header flits are also captured in this figure.

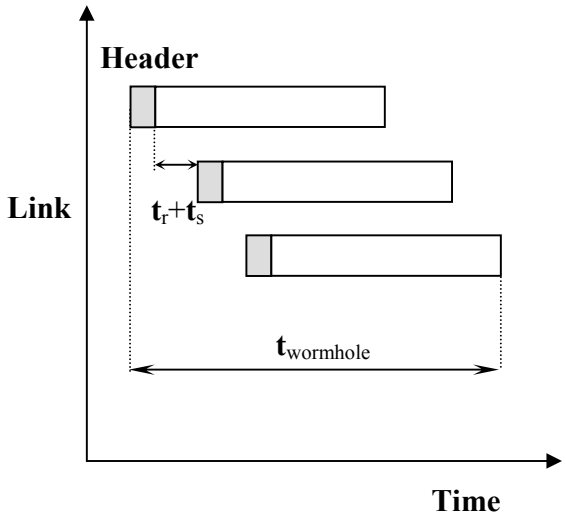


Figure 21: Time-space diagram of a wormhole-switched message.

The primary difference between wormhole switching and VCT switching is that, the unit of message flow control is a single flit and as a consequence small buffers can be used. Just a few flits need to be buffered at a router.
The blocking characteristics are very different from that of VCT. If the required output channel is busy, the message is blocked “in place”. This may lead to more deadlock situations than for other implemented strategies.
The base latency of a wormhole-switched message can be computed as follow:

$$t_{wormhole} = D_d\,(t_r + t_w + t_s) + max(t_w, t_s)M \tag{3.5}$$

This expression assumes flit buffers at the router inputs and outputs. Note that VCT and wormhole have the same latency. Once the header flit arrives at the destination the message pipeline cycle time is determined by the maximum of the switch delay and wire delay. For an input-only or output-only buffered switch, this cycle time would be given by the sum of the switch and wire delays.

Chapter 4

Routing Algorithms

Routing involves selecting a path from source node to a destination node in a particular topology. The routing algorithm used for a network is critical for several reasons. A good routing algorithm balances load across the network channels. The more balanced the channel load, the closer the throughput of the network is to the ideal. However many of the routing algorithms that have been proposed and are currently in use today fail to achieve an acceptable level of balancing load. This limitation can be partially explained because the routing has been designed to optimize a second important aspect of any routing algorithm – short path lengths [6].

A well-designed routing algorithm also keeps path lengths as short as possible, reducing the number of hops and the overall latency of a message. What may not be immediately obvious is that, routing minimally, which uses shortest paths, maximize throughput [7].

The list of routing algorithms proposed in the literature is almost endless. We will focus on a representative set of approaches, being used or proposed in modern and future multiprocessors interconnects [12]. The routing algorithms presented in this chapter are valid for all switching techniques. Special emphasis is given to design methodologies because they provide a simple and structured way to design a wide variety of routing algorithms for different topologies.

4.1 Taxonomy of Routing Algorithms

Routing algorithms can be first classified according to the number of destinations. Packets may have a single destination (*unicast routing*) or multiple destinations (*multicast routing*). Multicast routing will be studied in more details in chapter 5 and is included here for completeness [18].

Routing algorithms can also be classified according to the place where routing decisions are taken. Basically, the path can be either established at source node prior to packet injection (*source routing*) or determined in distributed manner while the packet travels across the network (*distributed routing*). Hybrid schemes are also possible. These hybrid schemes are termed, *multiphase routing* [12].

Routing algorithms can be implemented in different ways. The most interesting ways proposed up to now consist of either looking at a routing table (*table lookup*) or executing a routing algorithm in software or hardware according to a *finite-state machine*. In both cases the routing algorithm can be *deterministic, oblivious* or *adaptive*. Deterministic routing algorithms always supply the same path between a given source-destination pair. Oblivious routing algorithms send each packet first to a random node and from there directly to its destination. Adaptive routing algorithms use information about network traffic and/or channel status to avoid conflict in the network.

Another way, how to classified routing algorithm, can be according to their minimality as *profitable* or *misrouting*. Profitable routing algorithms only supply channels that bring the packet closer to its destination. They are also referred to as *minimal*. Misrouting algorithms may also supply channels that send the packet away from its destination. They are also referred to as *nonminimal*. The next classification is according to the number of alternative paths as *completely adaptive* (also know as *fully adaptive*) or *partially adaptive* [19].

In source routing, the source node specifies the routing path on the basis of a deadlock-free routing algorithm (either using lookup or not). The computed path is stored in the packet header. Source routing has been mainly used in networks with irregular topologies [20]. The first few flits of the packet header contain the address of the switch ports on intermediate switches.

For efficiency reasons most hardware routers use distributed routing. In distributed routing each intermediate node has to make a routing decision based on the local knowledge of the network. By repeating this process at each intermediate node, the packet should be able to reach its destination. This can be achieved because the designers know the topology of the whole network. Distributed routing algorithms are mainly used in regular topologies so that the same routing algorithm can be used in all the nodes. As a conse-

quence, routing decisions are much simpler than in other topologies. An alternative implementation approach consists of using table lookup [20].
An obvious implementation of table-lookup is to place a routing table at each node, with the number of entries in the table equal to the number of nodes in network. This routing can be performed either at the source node or at each intermediate node. In the first case, given a destination node address, the corresponding entry in the table indicates the whole path to reach that node. In the second case, each table entry indicates which outgoing channel should be used to forward packet toward its destination.
Misrouting algorithms are based on an optimistic view of the network: taking an unprofitable channel is likely to bring the header to another set of profitable channels that will allow further progress to the destination. Although misrouting algorithms are more flexible, they usually consume more network resources. Misrouting algorithms may also suffer from livelock [12].

4.2 Deterministic Routing

The simplest routing algorithms are deterministic (the routing is after dimensions). They establish the path as a function of the destination address, always supplying the same path between every pair of nodes. This lack of path diversity can create large load imbalances in the network. In fact, there is a traffic pattern that causes large load imbalance for every deterministic routing algorithm. So, for a designer these algorithms would not be their first choice. However deterministic algorithms still have their merits.
Many early networks adopted deterministic routing because it was so simple and inexpensive to implement. Therefore deterministic routing is permanently used in the network today. Especially, it is used in topologies, which can be decomposed into several orthogonal dimensions. This is the case of hypercube [21], mesh and tori [22]. Other types of interconnection networks, utilizing deterministic routing are irregular topologies. For these topologies it is more difficult to design good randomized or adaptive algorithms. Finally, for networks in which the ordering of messages between particular sort-destination pairs is important, deterministic routing is often a simple way to provide this ordering.

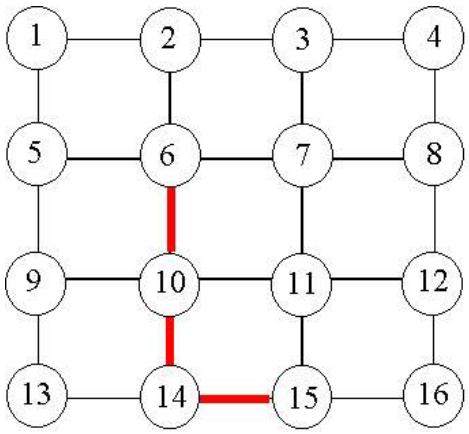


Figure 22: An example of deterministic routing. A packet is routed from node 15 to node 6 first by routing in the *x* dimension and then in the *y* dimension.

4.3 Oblivious Routing

Oblivious routing, in which we route packets without regard for the state of the network, is simple to implement. While adding information about network state can potentially improve routing performance, it also adds considerable complexity and if not done carefully can lead to performance degradation.

The main tradeoff with oblivious routing is between locality and load balance. By sending each packet first to a random node and from there directly to its destination, see Fig. 23, Valiant's randomized routing algorithm [23] exactly balances the load of any traffic pattern. However, this load balance comes at the expense of destroying any locality in the traffic pattern – even nearest neighbor traffic gives no better performance than worst-case traffic [24]. Minimal oblivious routing [25] on the other hand preserves locality and generally improves the average case throughput of a network over all traffic patterns.

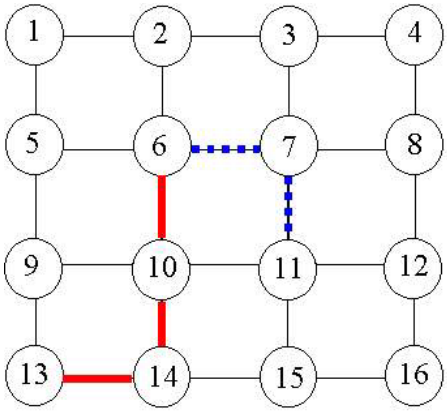


Figure 23: An example of randomized routing (Valiant's algorithm) on 4x4 mesh. A packet is routed from node 13 to node 11 in two phases. In the first phase the packet is routed to random selected intermediate node 6 as shown the bold solid lines. The second phase delivers the packet from node 6 to node 11 as shown the dotted lines.

4.4 Adaptive Routing

An adaptive routing algorithm uses information about the network state, channels occupancy, to select among alternative paths to deliver a packet [10], [26], [27], see Fig. 24. Because routing depends on network state, an adaptive routing algorithm is intimately coupled with the flow-control mechanism [28]. This is the contrast to deterministic and oblivious routing on which the routing algorithm and the flow control mechanism are largely orthogonal.

A good adaptive routing algorithm theoretically should outperform an oblivious routing algorithm, since it is using network state information not available to the oblivious routing. However many adaptive routing algorithms give poor worst-case performance. This is largely due to the local nature of most practical adaptive routing algorithms, because they use only local network state information in making routing decisions, they route in a manner that balances local load but often results in global imbalance [13].

The local nature of practical adaptive routing also leads to delay in responding to a change in traffic patterns [29]. The disadvantage of this routing method is prone to deadlocks and conflicts.

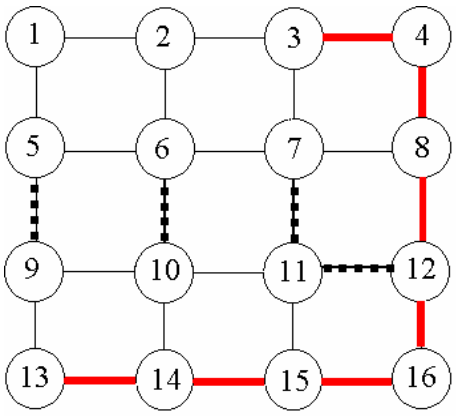


Figure 24: A packet is routed from node 13 to node 3 along the solid line. To avoid the channel occupancy, which is illustrated by dotted line, the packet is routed by the longer path, which occupies many channels of the interconnection networks.

A deadlock can occur in a channel and also at a buffer. A channel deadlock is mainly related to the situation, when a packet holds two channels and the other two channels are held by the other packet, but cannot proceed further until they acquire a third channel, currently held by the other packet. A buffer deadlock occurs when buffers are full, which are required by the arriving packet. Conflict is similar to a channel deadlock: two packets need to communicate via one channel in one direction at the same time.

A *minimal adaptive routing* algorithm chooses among the shortest path between source-destination pairs, using information about the network state in making the routing decision at each hop [30].
Partially adaptive routing algorithms represent a trade-off between flexibility and cost. They try to approach the flexibility of fully adaptive routing at the expense of a moderate increase in complexity with respect to deterministic routing. Some proposals aim at maximizing adaptivity without increasing the resources required to avoid deadlocks and conflicts. Other proposals try to minimize the resources needed to achieve a given level of adaptivity.
Fully adaptive routing algorithms are based on deadlocks and conflicts avoidance that either maximize adaptivity for a given set of channels while balancing the use of channels [31]. Some routing algorithms are proposed to deadlock recovery that accomplish both. Routing strategies based on deadlock recovery allow maximum routing adaptivity [32] as well as minimum channel requirements [14].

4.5 Routing in Irregular Topologies

Designing an efficient routing scheme for irregular networks is not trivial. While many effective routing schemes have been proposed for regular networks, there have been very few counterparts for irregular networks. A well designed routing scheme for irregular networks should be deadlock-free and provide high performance for various network topologies. Furthermore, the routing scheme should be efficiently implemented in a communication switch. Although, deadlock detection and recovery can be used to resolve the problem of routing, the complexity of the communication switch based on this is significantly increased. Several routing architectures have been proposed for irregular networks in recent years [33], [34]. These schemes are capable of routing packets in various network topologies and achieve deadlock freedom, they typically rely on routing table in the communication switch.
Most practical designs proposed recently for deadlock-free routing in irregular networks rely on deadlock avoidance.
In the next section we briefly describe the deadlock-free routing scheme used in DEC AN1 (Autonet) [34]. In addition to provide deadlock freedom, it provides adaptive communication between some nodes in an irregular network. Also we describe a fully adaptive routing algorithm for irregular topologies [35], [36] that considerably improves performance over the routing scheme proposed in [34].

4.5.1 Up*/Down* Algorithm

The up*/down* algorithm was first proposed for Autonet networks [34]. It is a distributed deadlock-free routing scheme that provides partial adaptability in irregular networks. Its general strategy is based on routing packets in a tree, where the routes go up the tree on leaving the source and then, come back down at the destination. One of the nodes is arbitrarily chosen as the root of the tree (usually, the one closest to the rest of the nodes) and all links of the topology are designated as up* or down* links with respect to this root. The up*/down* state of a link is relative to a spanning tree computed in background by a distributed algorithm. A link is up* if it points from a lower to a higher-level node in the tree (i.e. to a node closer to the root). Otherwise, it is down*. For nodes at the same level, nodes IDs break the tie. The routing from a source to a destination is established in such a fashion that zero or more up* links (towards the root) are traversed before zero or more down* links are traversed (away from the root) in order to reach the destination. The advantage of this approach is that each node's hardware and software are simple and some

adaptability is provided. The drawbacks are that the selected paths are generally not the shortest paths and that links near the root get congested (a conflict appears) and become bottlenecks leading to low throughput. Moreover, these problems become critical when the network size increases.

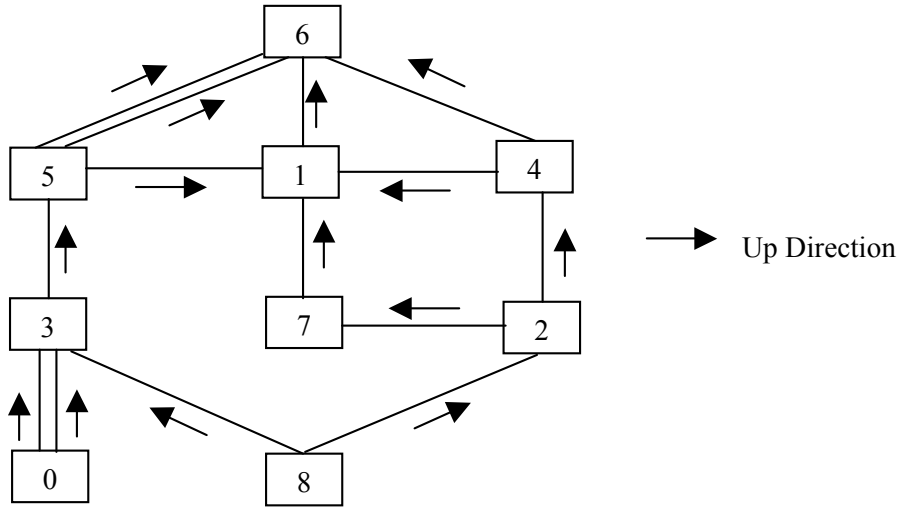


Figure 25: Link direction assignment for the irregular network [12].

4.5.2 Adaptive Routing Algorithm for Irregular Network

A general methodology for the design of adaptive routing algorithms for networks with irregular topology was proposed in [35]. That methodology can be summarized as follows. Given an interconnection network and a deadlock-free routing function definition, it is possible to duplicate all the physical channels in the network, or to split them into two virtual channels. In both cases, the graph representation of the new network contains the original and the new channels. Then the routing function is extended so that newly injected packets can use the new channels without any restriction as long as the original channels can only be used in the same way as in the original routing function. However, once a packet reserves one of the original channels, it can no longer reserve any of the new channels.

According to the extended routing function defined above, new channels provide more routing flexibility than original channels. They can be used to route packets through

minimal paths. However, once a packet reserves an original channel, it is routed through the original paths, which are nonminimal in most cases. Also routing through original paths produces a loss of adaptivity.

Following this reasoning, the general methodology, proposed in [35], can be refined by restricting the transition from new channels to original channels. Newly injected packets can only leave the source switch using new channels belonging to minimal paths and never using original channels. When the packet arrives at a switch, the routing function gives a higher priority to the new channels belonging to minimal paths. To ensure that the new routing function is deadlock-free, if none of the original channels provides minimal routing, then the original channel that provides the shortest path will be used. This enhanced design methodology was proposed in [36]. Finally, latency decreases significantly and the network is able to deliver a throughput several times higher than the one achieved by the up*/down* [35] routing algorithm [36].

Chapter 5

Collective Communication

Parallel processors conception can divide large and complex tasks into short tasks that are distributed and executed at the same time by many processors in order to achieve improved performance and to minimize the execution time. These processors need to communicate in order to exchange data and the results of their execution.

Communication operations can be divided in two types depending on how many processors are participating. If the communication involves a single source and a single destination, this type is called *point-to-point*; on the other hand, if communication involves more than one source and/or destination, this type is called *collective communication*. Provided that there is 1:1 mapping between processors and processes, we can equivalently talk about communicating process groups.

The importance of collective communications is derived from the fact that many frequently used parallel algorithms such as sorting, searching and matrix manipulation share data among groups of processes. Transmission of data to multiple destinations can be implemented with multiple calls for point-to-point transmission. However these patterns of sharing data are very regular and sufficiently important to merit special procedures.

In general, collective communication involves one or multiple transmitters and receivers, i.e. we have two sets of nodes: T – the set of transmitting nodes and R – the set of receiving nodes. The subsets T and R can be overlapping and can be as large as the full set of P

processes. Collective communication may be categorized as one to one, one to many, many to one or many to many, with many being also all.
There are also other operations that are collective in nature although no data are communicated, i.e. barrier synchronization.

5.1 Multiple One-to-One Communication

In this category each process can send at most one message and receive at most one message, see Fig. 26. If each process has to send exactly one message and receive exactly one message, there are *n!* different permutations or communication patterns. Figure 26 shows circuit shift permutation in which P_i sends a message to P_{i+1} for $1 \leq i \leq n-1$ and P_n delivers its message to P_1.
In the case that multiple one-to-one communication is the permutation, the set T is the same as the set R, then $|T \cap R| = P$. Generally $|T \cap R| \geq 1$, when some node doesn't send a message.

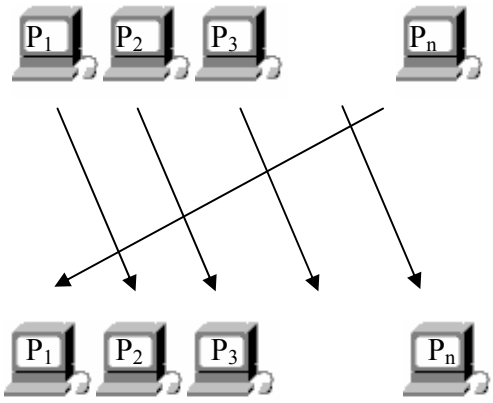


Figure 26: Multiple one-to-one communication pattern: circuit shift permutation.

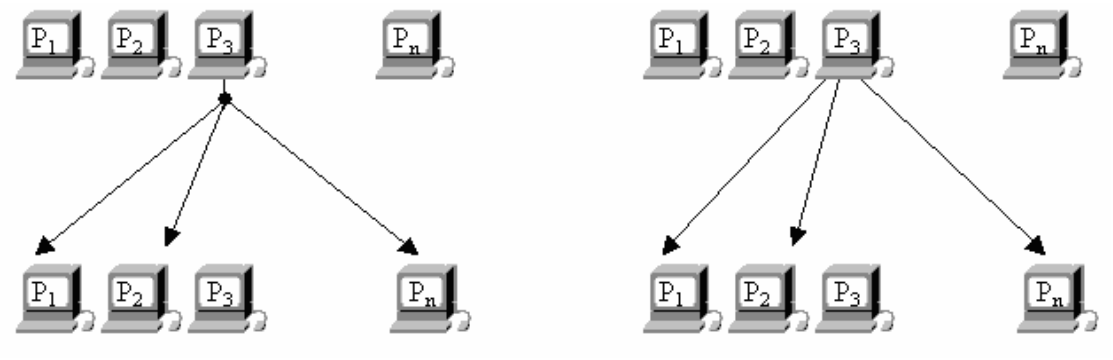
5.2 One-to-All Communication

In one-to-all communication one process is identified as the sender (called root) $|T| = 1$ and all processes are receivers $|R| = P\text{-}1$. In this communication, the sets of nodes are non-overlapping and apply $T \cap R = \varnothing$.

There are two distinct services in this category:

- *Broadcast* – the same message is delivered from the sender to all receivers.
- *Scatter* – the sender delivers different messages to the different receivers. This also referred as *personalized broadcast*.

Figure 27 shows the communication patterns of these two services.



(a) (b)

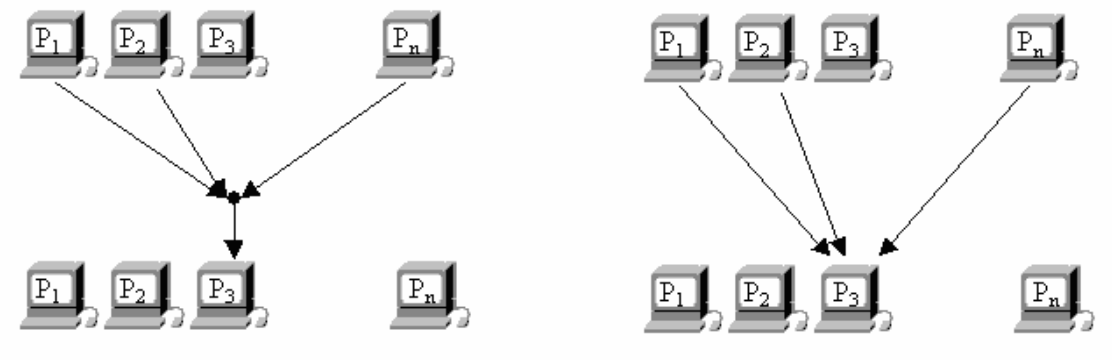
Figure 27: Two one-to-all communication patterns: (a) broadcast communication and (b) scatter communication.

5.3 All-to-One Communication

In all-to-one communication, all processes are senders $|T| = P\text{-}1$ and one process called the *root* is identified as the sole receiver $|R| = 1$. Again as previous case $T \cap R = \varnothing$, non-overlapping sets of nodes, and again, there are two distinct services:

- *Reduce* – different messages from different senders are combined together to form a single message for the receiver. The combining operator is usually communicative and associative, such as addition, multiplication, maximum, minimum, and logical OR, AND, and exclusive OR operators. This service is also referred to as *personalized combining* or *global combining*.
- *Gather* – different message from different senders are concatenated together from the receiver. The order of concatenation is usually dependent on the ID of the senders.

Figure 28 shows the communication patterns of these two services.



(a) (b)

Figure 28: Two all-to-one communication patterns: (a) reduce communication and (b) gather communication.

5.4 All-to-All Communication

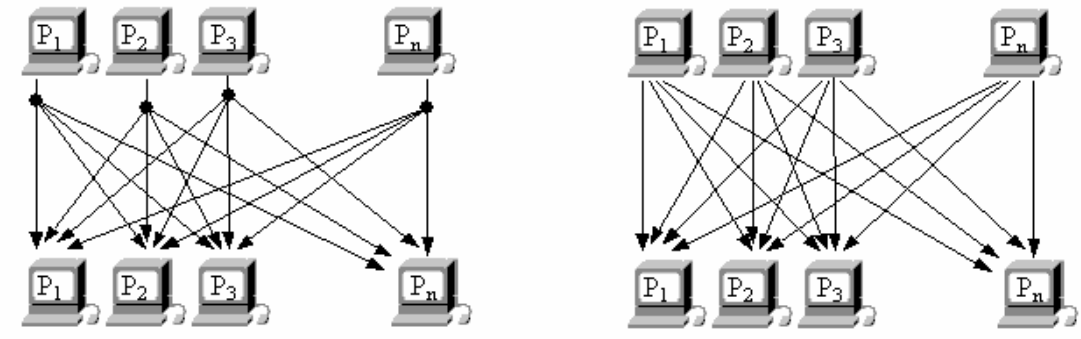
In all-to-all communication, all processes perform their own one-to-all communication. Thus, each process will receive n messages from n different senders. The sets of senders and receivers are identical $|T| = |R| = P$ and therefore $|T \cap R| = P$. Again, there are two distinct services:

- *All-broadcast* – all processes perform their own broadcast. Usually, the received n messages are concatenated together based on the ID of the senders. Thus, proc-

esses have the same set of received messages. This service is also referred to as *gossiping* or *total exchange*.

- *All-scatter* – all processes perform their own scatter. The *n* concatenated messages are different for different processes. This service is also referred to as *personalized all-to-all broadcast, index,* or *complete exchange*.

Figure 29 shows the communication patterns of these two services.



(a) (b)

Figure 29: Two all-to-all communication patterns: (a) all-broadcast communication and (b) all-scatter communication.

5.5 Many-to-Many Communication

Many-to-many collective communication is the generalization of the all communication patterns - all the preceding types of collective communication are special forms of this communication. In many-to-many communication, a certain number of processes are transmitters (senders) and simultaneously some group of processes receives messages. The groups of transmitters $|T| = M$ and receivers $|R| = N$ processes may:

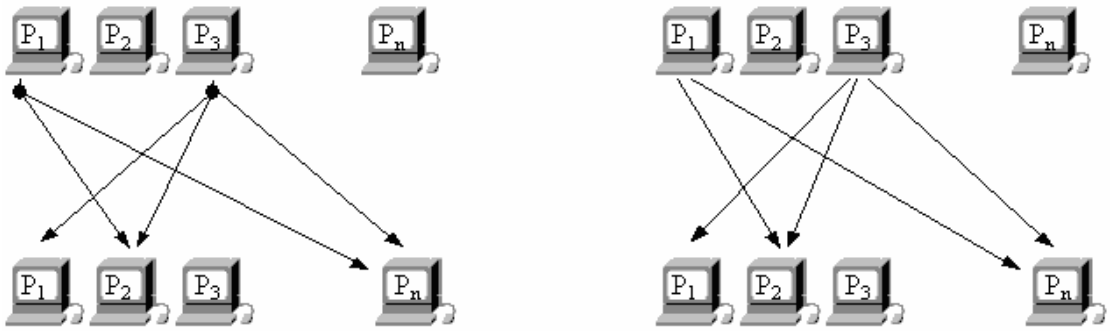
- *correspond* - thus, receivers and transmitters are the same processes.
- *separate* - transmitters and receivers are disjoint group of processes, thus, transmitters only send messages and receivers only receive messages.

- *overlap* – some processes only send messages, some processes only receive messages, and some processes send and receive simultaneously.

If groups of nodes are overlapping then $|T \cap R| \geq 1$ and $T \cap R = \varnothing$ if non-overlapping.
Many-to-many communication may be categorized into:

- *one-to-many* - one process is identified as the transmitter (called *root*) and subset of processes are receivers. Transmitter can send the same message (*broadcast*) or different messages (*scatter*) to receivers.
- *many-to-one* – a subset of processes are transmitters and one process, called the *root,* is identified as the sole receiver. Again, there are two distinct services as in chapter 5.3.
- *many-to-many* – a subset of transmitters sends the same messages (broadcast) or different messages (scatter) to a subset of receivers.

Figure 30 shows the communication patterns of many-to-many communication, where senders and receivers create overlap groups.



(a) (b)

Figure 30: Two many-to-many communication patterns, where transmitters and receivers are overlapped: (a) many-broadcast communication and (b) many-scatter communication.

5.6 Convenient Collective Communication Services

In addition to the basic types of collective communication services, some collective communication services require the combination of these basic services. Some of these frequently used collective communication services, referred to as *convenient* or *composite* collective communication services, as listed bellow [12]:

- *all combining* – the result of a *reduce* operation is available to all processes. This is also referred to as a *reduce* and *spread* operation. The result may be broadcast to all processes after the reduce operation or multiple reduce operations are performed with each process as a root.
- *barrier synchronization* – a synchronization barrier is a logical point in the control flow of an algorithm at which all processes must arrive before any of the processes are allowed to proceed further.

Collective communication services are demanded in many applications. Such services have been supported by several communication packages for multicomputers. However, efficient implementation of various collective communication services is topology dependent.

5.7 Models of Communication

The simplest time model of communication in distributed memory systems uses a number of communication steps (rounds): point-to-point communication takes one step between adjacent nodes and a number of steps if the nodes are not directly connected. In the more detailed view, the communication time is composed of a fixed start-up time t_s at the beginning and of a component that is a function of distance D_d (the number of channels on the route or hops a message has to make), and message length m in certain units (words or bytes). More details are described in chapter 3.
Further, we have to distinguish between unidirectional (simplex) channels and bi-directional (half-duplex, full-duplex) channels. The number of bi-directional channels between the CPU and a router (ports) that can be engaged in communication simultaneously (1-port or all-port models will be considered, as they are most common) has also an impact on number of communication steps and communication times, as well as if nodes can combine/extract partial messages with negligible overhead (combining nodes) or can

only re-transmit/consume original messages (non-combining nodes). Finally we have to take into account a slim/fat node (one processor/more processors at a node), a switching technique and a network topology.

A few comments are appropriate on communication among various nodes from the simplest to the most complex type. A single neighbor-to-neighbor communication, multiple neighbor-to-neighbor communications, and a single point-to-point communication are always deadlock free. But collective communications are inherently prone to a deadlock. If each node sends or receives messages to or from more than one partner in a loop asynchronously, it still faces the danger of a so called fetch deadlock (at least two nodes execute pending send operation and cannot receive). This is why a synchronized communication model, where the communication proceeds in synchronized rounds (steps), is much more popular and frequently used.

The topology is one of the key design factors of a collective communication. There is a large body of theoretical research on optimal topologies, based on graph theory metrics such as average distance, network diameter, and bisection width, among others. These parameters have a direct impact on network performance, Tab.1. As far as the broadcast communication (OAB) in SF network is concerned, the number of steps cannot be less than network diameter D, because this is the worst case even for point-to-point communication. For WH switching the distance between nodes is not that important and the lower bound $\lceil \log_{d+1} P \rceil$ is given by the number of nodes informed in each step, that is initially 1, $1+1\times d$ after the first step, $(d+1)+(d+1)\times d = (d+1)^2$ after the second step, etc.,…, and $(d+1)^k$ nodes after step k.

In case of (SF or WH) AAB communication, since each node has to accept $P-1$ distinct messages, the lower bound is $\lceil (P-1)/d \rceil$ steps. A similar bound applies to OAS communication, because each node can inject into the network not more than d messages in one step; for irregular networks with non-constant node degree d we should use the lowest value of the node degree for AAB and the value of the source node degree for OAS. The common strategy with SF OAS is to send messages to the farthest nodes first and then pipeline them with messages to the nodes less and less remote. The optimum broadcast tree is therefore different from that for OAB. In WH OAS we use different strategy: $P-1$ pair-wise communications must be packed into the lowest number of steps in such a way that there are only edge-disjoint paths in a single step.

For AAS communication pattern each of P processor sends an individual message to each of P-1 partners. If S_{sd} is the sum of the shortest distances of all node pairs, then the average distance of nodes $d_a = S_{sd}/P^2$, [37]. With concurrent communication on $2e$ ports ($2e = Pd$ in regular networks), the number of communication steps for SF switching cannot be less than $S_{sd}/(2e)$. Another lower bound for AAS can be obtained considering that one half of messages from each processor cross the bisection and the other half do not. There

will be altogether $2\lceil P/2 \rceil \lfloor P/2 \rfloor$ of such messages in both ways and up to B_C messages in one step, where B_C is the network bisection width [37]. This gives $x = 2\lceil P/2 \rceil \lfloor P/2 \rfloor / B_C$ steps. This second lower bound applies to WH as well as SF switching, but is more appropriate to WH switching, since point-to-point messages (and not neighbor-to-neighbor messages as in SF switching) are considered.

CC	SF switching		WH switching	
	1-port	all-port	1-port	all-port
OAB	max ($\lceil log_2 P \rceil$, D)	D	max ($\lceil log_2 P \rceil$, D)	$\lceil log_{d+1} P \rceil$
AAB	$P - 1$	$\lceil (P - 1)/d \rceil$	$P - 1$	$\lceil (P - 1)/d \rceil$
OAS	$P - 1$	$\lceil (P - 1)/d \rceil$	$P - 1$	$\lceil (P - 1)/d \rceil$
AAS	max (S/P, $2xe/P$)	max [$S/(2e)$, x]	$2xe/P$	$x = 2\lceil P/2 \rceil \lfloor P/2 \rfloor / B_C$

Table 1: Lower bounds on complexity of collective communications at slim node topology.

Chapter 6

Design of New Evolutionary Optimization Techniques

Evolution is the adaptation of a population to its environment. This adaptation causes the creation of individuals of increasingly higher/greater "fitness"; in environments where the definition of fitness remains static, evolution drives the population towards better and better individuals. This process is similar to approximation - the search for good solutions to a particular problem. The parallels between the concepts of evolution and approximation have lead to the creation of evolutionary approximation.

Many problems in real application have a search space that is exponentially proportional to the problem dimensions and, but for the simplest of cases, these problems cannot be solved using exhaustive search methods. Consequently, there is considerable interest in heuristic techniques that attempt to discover near-optimal solutions within an acceptable time. Evolutionary techniques provide a framework for effectively sampling large search spaces, and the basic technique is both broadly applicable and easily tailored to specific problems. All that is required to apply an evolutionary technique to any particular problem is an appropriate encoding scheme and a target function.

Chapter 6 Design of New Evolutionary Optimization Techniques

During the last decades, wide applicability has been demonstrated by successfully applying evolutionary computation techniques to various optimization problems in the fields of engineering, management science, biology, chemistry, physics and computer science.

6.1 Basics of Classical Genetic Algorithm

Genetic algorithm (GA) is a powerful, domain-independent search technique that was inspired by Darwinian theory. It emulates the natural process of evolution to perform an efficient and systematic search of the solution space to progress towards the optimum. It is based on the theory of *natural selection* that assumes that individuals with certain characteristics are more able to survive and hence pass their characteristics to their offsprings. It is an adaptive learning heuristic belonging to a class of general *nondeterministic* algorithms.

GA is any population-based computational model that uses selection and recombination operators to generate a new sample in a search space. A chromosome (individual), consisting of genes, represents one encoded solution of the search space. The values of genes are referred to as alleles. The chromosomes form a population, which changes through the process of evolution. The reproduction process is performed in such a way that chromosomes, which represent a better solution, are given more chance to reproduce than those chromosomes, which represent poorer solutions. The fitness function (a measure of quality) of the chromosomes is defined in the frame of the population. The fitness function is applied to genotype (chromosomes) for evaluating phenotype (decoded form of the individual/chromosome). While the fitness function operates with phenotype, genetic operators are defined on the genotype. Convergence of genetic algorithms has been proved by use of Markov chains and a fundamental Schema Theorem [38], [39].

- **Selection**

The operator of selection determines, which individuals will produce offsprings. A fitness function serves as a criterion (numeric evaluation of solution which represents an individual), but worse individual can participate in the creation of new population with defined probabilistic. Many selective strategies exist, which differ by in their accuracy and deterministic degree of choice.

- **Crossover**

This is the most important genetic operation. A created chromosome inherits part of the genes from one parent's chromosome and the rest from the second parent. In the case that the created chromosome is evaluated by higher price than its parents, this means, that the advantages are a connected profitable property in the chromosome and its recessive characters are inhibited.

- **Mutation**

Mutation causes a random modification of chromosomes. In the case, that the population is saturated and converges to local extreme, mutation ensures the input of new genetic information.

6.2 Simulated Annealing

The origin of simulated annealing (SA) lies in the analogy of optimization and a physical annealing process [40], [41]. In condensed matter physics, *annealing* is a thermal process for obtaining low-energy states of a solid in a heat bath. Roughly, the process can be described as follows. First, the temperature of the heat bath is increased to a maximum value at which the solid melts. Thus, all particles of the solid arrange themselves randomly. Afterwards, the temperature is carefully decreased until the particles of the melted solid reach in the ground state of the solid in which the particles are arranged in a highly structured lattice with minimum energy.
The physical annealing process can be simulated by computer programs using *Monte Carlo techniques* proposed by Metropolis et al. [42]. Given a current state i of the solid with energy E_i, a subsequent state j is generated by applying a perturbation mechanism, which transforms the current state into the next state by a small distortion, for instance by displacement of a single particle. If the energy difference $\Delta E = E_j - E_i$ is less or equal to zero, the state j is accepted as the current state. If the energy difference is greater than zero, the state j is accepted with probability $\exp(-\Delta E / (k\ T))$, where T denotes the temperature of the heat bath and k the *Boltzmann constant* [43], [44], [45]. The acceptance rule described above is known as the *Metropolis criterion*. In simulated annealing, the Metropolis criterion is used to generate sequences of solutions of combinatorial optimization problems.

The key to SA's proof of convergence is that a stationary distribution must be reached at each temperature, followed by sufficiently slow cooling. A side-effect of this proof is that other algorithms which achieve a stationary Boltzmann distribution, and which perform sufficiently slow cooling, will inherit the same convergence guarantees [44].

In the original version of simulated annealing, a final state serves from Metropolis algorithm as a started state the following temperature phase ($T= \alpha*T$). This basic premise can be modified in that way, Metropolis algorithm is initialised by the best solution, which was obtained in the previous temperature phases. That modified method; is called *simulated annealing with elitism*.

6.2.1 Control Parameters of Simulated Annealing

- Initial temperature T_0: This must be chosen in order that almost all perturbations are accepted.

x=(number of perturbations accepted) / (total number of perturbations attempted)

$$T = \frac{(\Delta\overline{Cost^+})}{\ln(\frac{m^+}{x(m^- + m^+) - m^-})}, \tag{6.1}$$

where x is the acceptance probability, $\Delta\overline{Cost^+}$ is the average change in cost over all perturbations, which lessen cost function, m^- is the number of perturbations with the cost function decrease and m^+ is the number of perturbations with the cost function increase [46].

- k_{max}: number of iterations of Metropolis algorithm in one temperature phase. The number k_{max} is based on the requirement that at each value of T quasi-equilibrium is succeeded.

$$k_{\max} = \max_{S_i \in \Omega} |N(S_i)|, \tag{6.2}$$

where $N(S_i)$ is the maximum size of the configuration subspace, Ω is the search space and S_i is current state.

- Decrement coefficient α: The coefficient α (the term in brackets) is proposed to reduce the temperature.

$$T_{k+1} = T_k \left\{ 1 + \frac{T_k \ln(1+\delta)}{3\sigma_{T_k}} \right\}^{-1}, \tag{6.3}$$

where δ is a measure of how close the equilibrium vectors of two successive iterations are to each other, σ_{T_k} is the standard deviation of the cost function up to the temperature T_k [46].

- The stopping criterion is based on the monitoring of the relevant reduction of the cost function during the optimisation process

$$\frac{d\overline{Cost_s}(T)}{dT} \frac{T}{\overline{C}(T_o)} < \varepsilon_s, \tag{6.4}$$

where ε_s is a small positive number called the stopping parameter, $\overline{C}(T_o)$ is the average value of the cost function at T_0. This condition is based on extrapolation of the smoothed average cost $\overline{Cost_s}(T)$ obtained during the optimisation process [46].

6.2.2 Parallelization of SA

It is possible to use two different techniques from the point of communication:

- In the first technique each process performs its complete SA algorithm, but each one works with a different generator of the random number but they don't communicate. The final solution is chosen at the end of optimization process.
- In the second approach all processes communicate with the master, which returns to all of them the current best solution, or with its neighbours. But in the case of asynchronous communication, the communication can be too frequent at higher temperature and the time of communication can be much greater than the time of optimisation.

We proposed the technique, which is a combination of both of the above, i.e. in the higher temperature the processes are independent and the communication is activated only for the lower temperature phase.

6.2.3 Design of a New Parallel SA

➢ **The basic terms and definitions**

- Accepted solution
  - New generated solution have smaller predefined cost function or worse but it satisfies the condition: `random()< min[1,` $e^{-\frac{f(x')-f(x)}{T}}$], where f(x´) is new cost, f(x) is old cost, T is temperature and random is generated randomly with uniform distribution.
- Structure of parallel processes, see Fig. 31.
  - One control process (master)
  - n – slave processes (slaves)
- Synchronization: the way of communication between master and slaves
  - Asynchronous mode – a process communicates with the master independently of the other processes
  - Synchronous mode
    - All slaves communicate with master in one predefined time period
    - All slaves wait for the message from master at the end of the temperature phase to continue in computation, i.e. in one temperature period the communication slave – master proceeds asynchronous but the whole execution appears as synchronous. The processes are namely synchronized at the end of temperature phase.

In the lower temperatures the processes cooperate by using the architecture master – slave and all slaves (and also master) work on its sequence of solutions. If some slave process finds an acceptable a solution, it sends it to master, which determines its acceptance according to its own rule. If accepted this solution or a new solution found by the master, it is sent to all slave processes.

Each communication slave/master runs asynchronously in one temperature phase. A problem appears with termination and with delay of processes (e.g. it is caused by different frequency of communication of each process with master). The principle of how to solve this problem is based on the usage of synchronisation at the end of temperature phase, which is controlled by the master. This approach allows that all processes work at the same temperature and also finish at the same time.

The scheduling of the messages is shown in the Fig. 31.

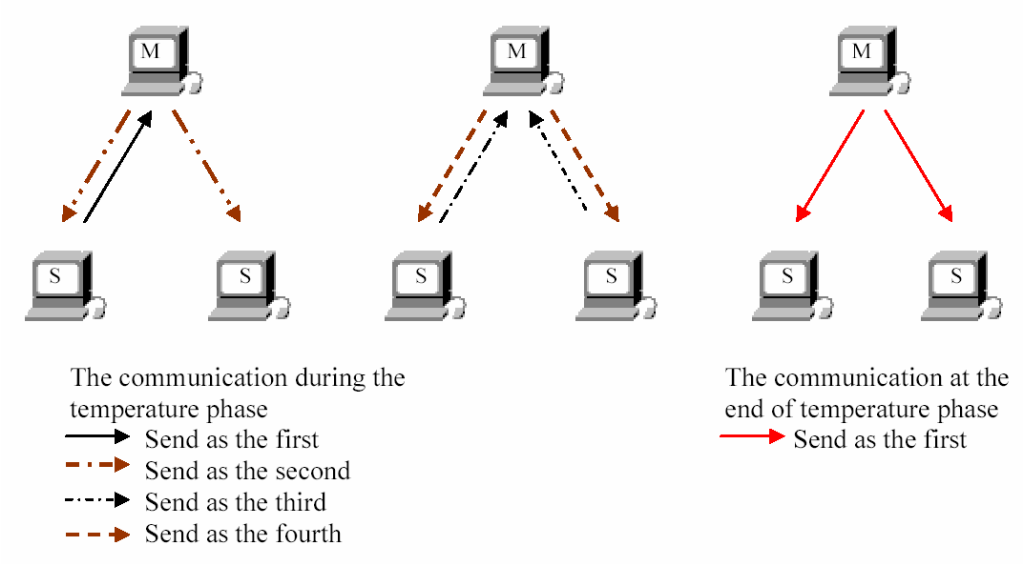


Figure 31: Illustration of the communication during the temperature phase and at the end of the temperature phase.

- **Advanced modification of parallel simulated annealing (PSA)**

A. Communication only at the end of temperature phase:
In comparison to the described version in the previous point (the basic terms and definitions) slaves communicate with master exclusively at the end of temperature. In this case all processes communicate at the same time and therefore there is no need of synchronisation. This way of communication is already synchronous, i.e. processes work at the same temperature phase.

B. Communication after defined number of iterations:
The same idea as for case A, but in this version the communication is performed after defined number of iterations at each temperature phase of Metropolis algorithm (e.g. after each 10^{th}, 100^{th} or 1000^{th} iteration). This way of communication is implicitly synchronous again.

C. Usage of elitism:
In this case, the Metropolis algorithm is initialised by the best solution, which was obtained during previous temperature phases. Otherwise the output from Metropolis algorithm is taken as starting state of the next temperature phase ($T= \alpha*T$). Communication is proceeded asynchronous after each iteration, but it is synchronised at

the end of temperature phase. The principle of synchronisation was described above.

D. Sequential version of SA. It was described in the chapter 6.2. The sequential SA algorithm performs as an interesting comparison to PSA algorithms.

- **Experimental results**

All parallel versions of SA were implemented in C using MPI [47] routines for message passing and it can therefore be compiled and run on any architecture (clusters of workstations, MPPs, SMPs, etc.) for which an implementation of MPI standard is available.
Parallel variants of SA were not considered a possible test for the problem of collective communication scheduling, because this problem is too complex. Therefore it was decided that PSA is tested using a simpler problem. We chose the well known problem of TSP (traveling salesman problem), as optimal results of this problem as well as optimal setting SA parameters are known. TSP benchmarks were published on the web site [48]. The principle of this problem is to search as much as possible the shortest path between all cities. Using this problem, it was possible to detect the differences and abilities of individual PSA algorithms.
Most of experimental work was performed on the benchmark of 52 cities see Fig. 32 to 34. It was performed 15 runs for tested versions of PSA. The efficiency of PSA versions were also proved by benchmark of 79 cities, see Fig.35.

Optimal solution of TSP problems:

- berlin52 - TSP52 (52 cities) - tour length equals to 7542
- eil79 - TSP79 (79 cities) - tour length equals to 538.

In all experiments the following control parameters were used:

K_{max}	10000
T_{max}	100
T_{min}	1
T_{change}	20
Alpha	0,9
Count of processors	8

Table 2: The setting of SA control parameters.

Notice: The value T_{change} was changed to 30 to achieve better lucidity of optimization curves of tour length in Fig. 34 and 35. We didn't use mutual communication between processes in all temperatures phases, because the asynchronous variant was tested. In this asynchronous version, the time of optimisation can be rapidly degraded by frequent time of communication at higher temperature. Because of the comparison of execution time, all versions of PSA mutual cooperated in architecture master-slave only in interval T_{change} - T_{min}.

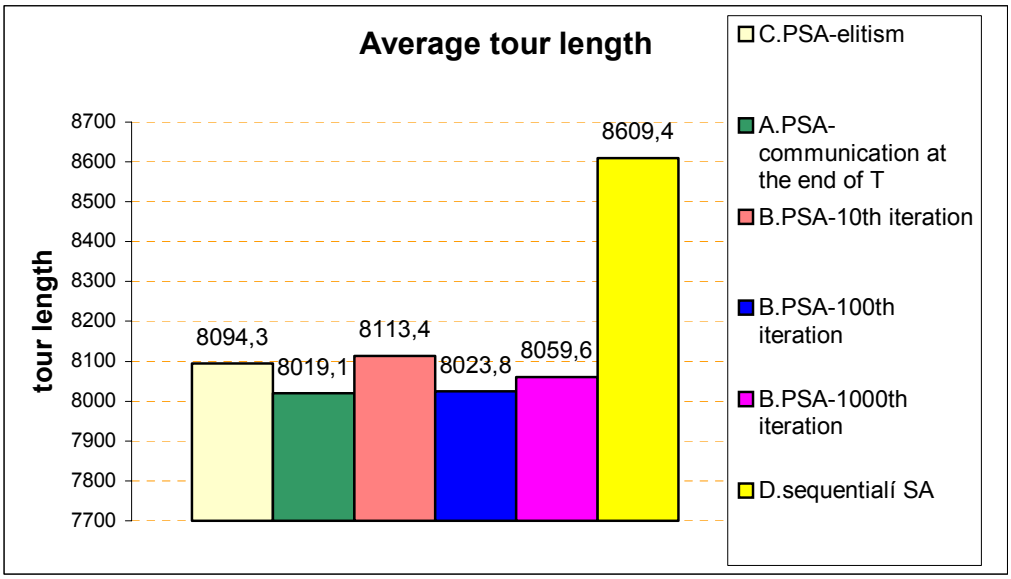


Figure 32: Average tour length of TSP 52 for several versions of PSA.

In Fig. 32 the performance of the sequential SA and five PSA algorithms are illustrated. Sequential SA, which is shown in yellow (the first from right side), uses the same parameters as PSA versions. It is evident that the best versions of PSA are those, in which relatively small intensity of communication is used. The variant B provides the best results with communication after each 100^{th} iteration of Metropolis algorithm.

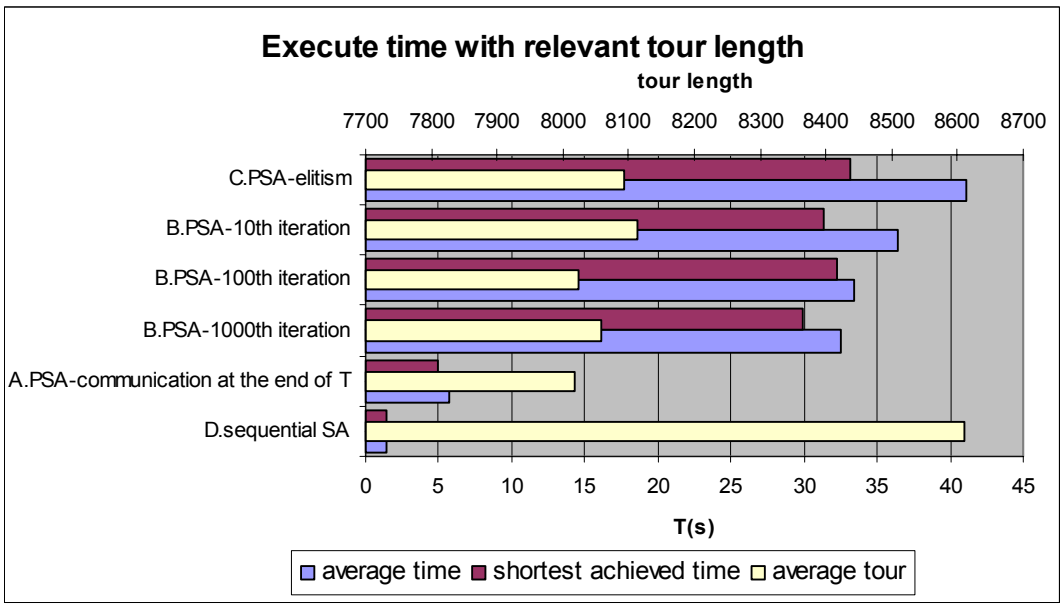


Figure 33: Computational time with relevant average tour length at each PSA versions and sequential SA versions.

In Fig. 33 computational time and average tour length for each version is shown. The sequential version doesn't achieve such quality of results as the parallel versions indeed it runs the same time as the best parallel versions. From the figure it is evident that too frequent communication increases computational time and simultaneously it cannot improve result quality. The best version of PSA is variant A according to computational time and average tour length, which communicates at the end of temperature phase and also variant B, which communicates after each 100^{th} iteration.
It is evident that the higher is the intensity of communication, the longer execution time of PSA versions - therefore trade-off must be found.

In Fig. 34 and Fig. 35 the optimization curves of length tour are presented for several PSA versions. From B version of PSA the variant was chosen with communication after each 100^{th} iteration.

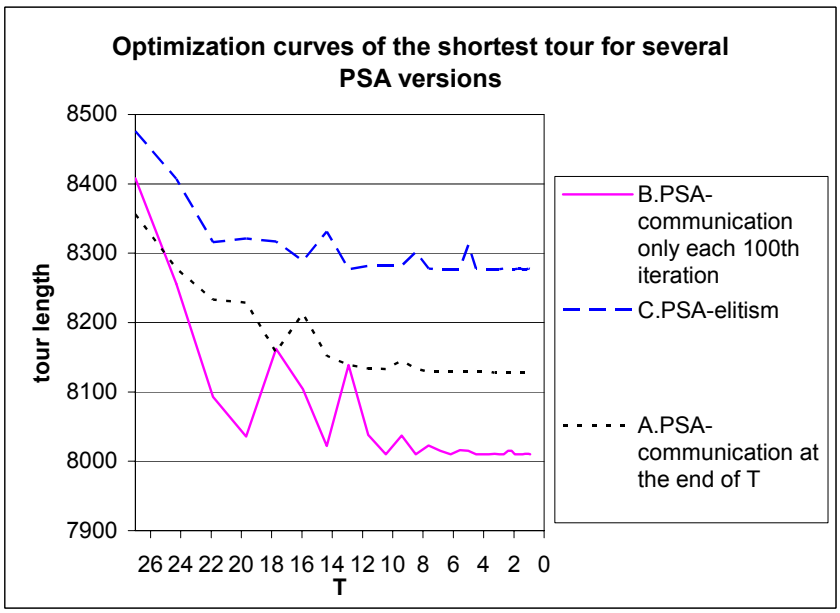


Figure 34: Optimization curves for TSP 52 (52 cities).

In Fig. 35 optimization curves of several PSA versions and sequential SA are presented for TSP 79 problem.

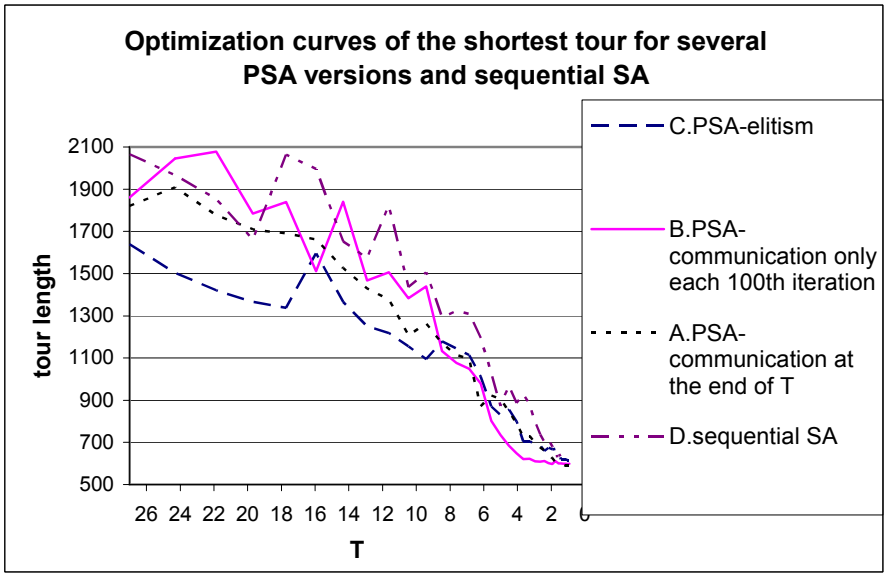


Figure 35: Optimization process of tour length for TSP 79 (79 cities).

Chapter 6 Design of New Evolutionary Optimization Techniques

➢ **Summary**

All versions of PSA are based on mutual cooperation of the master-slave processes. In the first phase all processes are independent at temperature intervals T_{max}-T_{change} and each of them produces its optimised solution. In the second phase slave processes cooperate via control master process.

Individual versions differ only by form and by count of communication. The new conception of parallel algorithm of simulated annealing is based on two basic modifications, which was applied in the designed PSA versions.

- Communication at a given iteration epoch or at the end of temperature phase
  - All processes communicate at defined time interval or after given iteration number, which provides synchronization
    - Advantage - excessive communication is reduced and problem with acceptance of worse solution at low temperature is solved
    - Disadvantage – at very low temperature the processes produce the similar solutions. The profit of parallelization is decreased
- Usage of elitism
  - It uses synchronization at the end of temperature phase, otherwise the communication proceeds asynchronous after each iteration.
    - The disadvantage of this approach lies in excessive communication, which results in computation time increase.
    - Advantage – elitism removes the possible problem with the acceptance of worse solutions at low temperature phase

All variants were tested on two problems of travelling salesman problems TSP52 and TSP79. In all versions of PSA and SA, no heuristics were used, because we investigated influence of cooperation processes on quality of achieved results and convergence speed to global solution. The sequential SA algorithm performs as an interesting comparison to PSA algorithms. From received results, it follows the necessity of trade-off between intensity of communication and computational time. The best results produce one of PSA - B versions, which uses communication after each 100^{th} iteration of Metropolis algorithm, see Fig. 32. In the case of communication after each 1000^{th} iteration or at the end of temperature phase it didn't achieved such superior results. In a such small communication intensity the processes already generate similar results. In the case of the intensive communication (almost at each iteration) the execution time is very high and the results are much worse than for the minimal communication. This fact was illustrated in Fig. 33.

6.3 Hybridization of Evolutionary Algorithms

Hybrid evolutionary algorithms are built by combination of stochastic evolutionary algorithm and problem specific algorithm or by aggregation of mostly two evolutionary algorithms. In our case, we target the aggregation of simulated annealing (SA) and genetic algorithm (GA). In implementing a SA using GA principles, we seek to incorporate strengths and eliminate weaknesses of both methods. In practice both of them may converge prematurely to suboptimal solutions but only SA currently possesses a formal proof of convergence to global optimum. This proof depends on SA's cooling schedule [49].
Next contrasts between SA and GA have to do with loss of solutions, redundancy and deceptive problems. Since SA maintains only one structure (solution) at a time, whenever it accepts a new structure, it must discard the old one; there is no redundancy and no history of past structures. The end result is that good structures and substructures (in the extreme case the global optimum) can be discarded, and if cooling proceeds too quickly, may never be regained. SA compensates by increasingly sampling good solutions as temperature decreases. GAs are also prone to the loss of solutions and their substructures or bits [55] due to the disruptive effects of genetic operators. Upon disruption, the simple GA will not maintain an old, but better solution; it must accept any newly generated solution, regardless of fitness. However, the GA partially overcomes this, especially at larger population sizes, by exponentially increasing the sampling of above-average regions of the search space, known as *schemata* [39], [58]. A schema is a subset of the solution space, whose elements are identical in certain fixed bit-positions. Schemata act as a partial history of beneficial components of past solutions. However, for each above-average schema duplicated, a competing, below-average schema must be discarded. This can lead to trouble on difficult or deceptive problems, where low-order, low-fitness schemata are needed for the construction of an optimal higher-order schema [41]. SA is similarly subject to deception, as the algorithm will have a difficult time paying extended visits to the high-cost neighbors of a deceptive problem's global optimum.
A final but important difference between GA and SA is the ease with which each algorithm can be made to run in parallel. GA is naturally parallel - they iterate an entire population using a binary recombination operator (crossover) as well as a unary operator (mutation). SA, on the other hand, is naturally sequential and therefore it is not easily run on parallel processors; and works only with one structure (solution). While attempts have been made to parallelize SA, a general-purpose method has no yet been demonstrated.

6.3.1 A Short Survey of Hybrid Parallel Simulated Annealing Using Genetic Operators

There are many hybrid parallel genetic simulated annealing algorithms, but they use one of two possible concepts. The first one is based on the algorithm SA, which is enhanced with particular genetic operations. The second is based on the concept of GA, which uses Metropolis algorithm at the selection process. In this paper we analyzed three variants of aggregation SA and GA:

- S. W. Mahfoud and D. E. Goldberg proposed algorithm based on the concept of GA, which uses the Metropolis algorithm in the selection process [50].
- M. Krajíc described parallel hybrid genetic simulated annealing, which is based on the concept of SA and it uses genetic operations (mutation and crossover) [52].
- N. Mori, J. Yoshida and H. Kita suggested the thermodynamical selection rule in genetic algorithm [51].

We proposed the hybrid parallel genetic simulated annealing (HGSA) using architecture master-slave. HGSA is based on the best parallel version of SA, which is discussed in section 6.2.3. Each process includes the master process execute simple SA algorithm. The crossover and mutation GA operations are used just after the communications master-slave. A detail description is presented in the next chapters.

6.3.2 Design a New Hybrid Parallel Genetic Simulated Annealing (HGSA)

HGSA is a hybrid method utilizing parallel SA with genetic operators. The flow of the algorithm is shown in Fig. 36. The parallel SA is built on the base of a master-slave configuration of processes. In each process the sequential SA is running (point 1 in Fig. 36). During the communication (Fig. 36 - point 2) which is activated each 100^{th} iteration of Metropolis algorithm, each process sends their solution to a master. The master keeps one solution for himself and sends one randomly chosen solution to each slave. The selection is based on the roulette wheel, where the biggest probability of selection has the individual with the best fitness function. After the communication phase all processes have two individuals. Now, starts the phase of a genetic crossover (Fig. 36 - point 3). Two additional children solutions are generated from two parent solutions using double-point crossover.

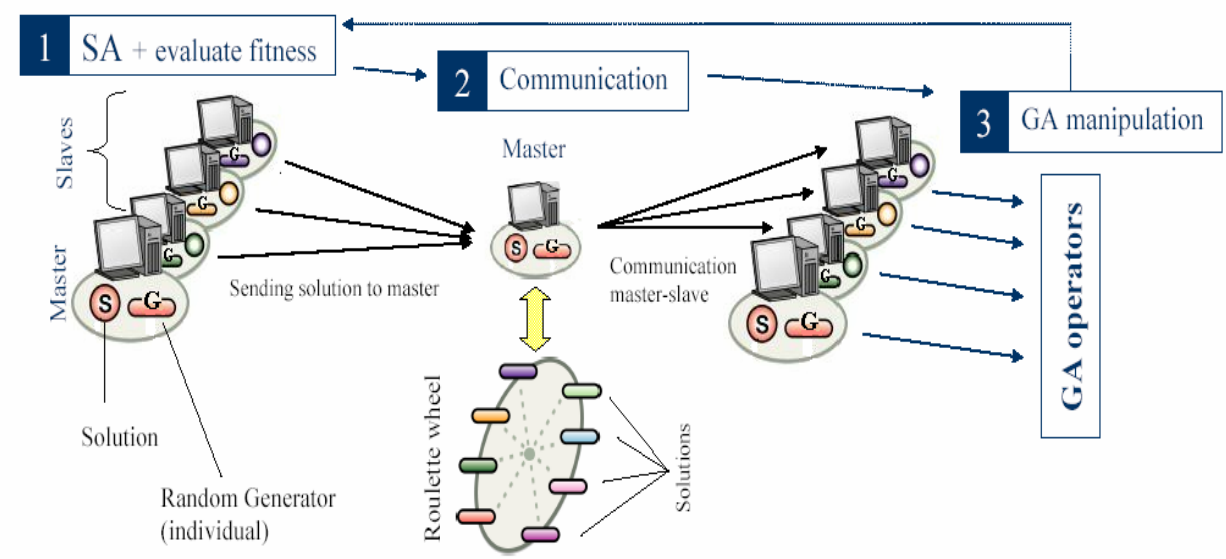


Figure 36: Structure of Hybrid parallel genetic simulated annealing.

The best solution from two parents and two children is selected and mutation is performed always (in the parent solution) or with a predefined probability (in the children solution). Mutation is performed by randomly selecting genes and by randomly changing their values. A new solution for each process is selected from the actual solution provided by SA and from the solution obtained by genetic manipulation. The selection is controlled by well-known Metropolis criterion.

6.3.3 General Differences between New HGSA and Other Approach of SA and GA Aggregation

We will take interest only in SA and GA aggregation, which are based on parallel SA using GA operators.

Our algorithm HGSA differed from other hybrid algorithms mainly in the way of communication between processes. Our approach is based on master-slave architecture, when master sends individuals to slave by *roulette wheel,* such that the individual with better evaluation of fitness function is sent more often than others. Other published algorithms randomly send *all* solution between processes or only neighbors.

In HGSA communication is activated each n^{th} iteration of Metropolis algorithm. The other authors of hybrid algorithms activate a communication *at the end* of temperature phase. After the receiving of each individual the phase of genetic operations starts. Two new solutions are generated, using crossover from parent solutions. In our approach, the best individual can be one from parent or offspring solutions and it is generated by *two-point* crossover. Selection and crossover differ also from other hybrid algorithms. In the case that the winner is parent solution the mutation is always performed, otherwise the mutation is performed by predefined probability. In other hybrid algorithms, only the offspring solution is chosen for mutation, which is always performed.
These are the main differences and some further differences we can search by detailed comparing of HGSA and other hybrid algorithms.

6.4 Experimental Results

Hybrid parallel genetic simulated annealing and selected variants of PSA algorithm were tested on three TSP problems, which were published on the website [48]. We compared our HGSA with PSA versions described in section 6.2.3, to determine which of them achieve the best results according to the solution quality and the computing time. In all tested versions of PSA and HGSA, no heuristics was used, because we compared the abilities of proposed algorithms without utilization of accelerating methods for local searching.

In all experiments the following control parameters were used:

K_{max}	10000
T_{max}	100
T_{min}	1
Alpha	0,9
$Pr_{mutation}$	0,1
Iteration of communication	100
Number of processors	8

Table 3: The value of HGSA parameters.

Most of the tests were performed on the benchmark of 52 cities see Fig. 37 and 38. In total 15 runs for 52 cities were performed, in each versions of PSA. The efficiency of PSA versions was also proved by benchmark of 79 cities and by benchmark of 100 cities.

In Fig. 37 are shown the experimental results of three selected PSA versions and HGSA algorithm. The PSA versions differ only by the used time interval between master-slave communication. HGSA algorithm has used fixed communication period - each 100th iteration. All PSA versions found similar average tour length. The optimal tour was not found in any of the 15 runs. But in case of HGSA the optimal solution was achieved in each of 15 runs.

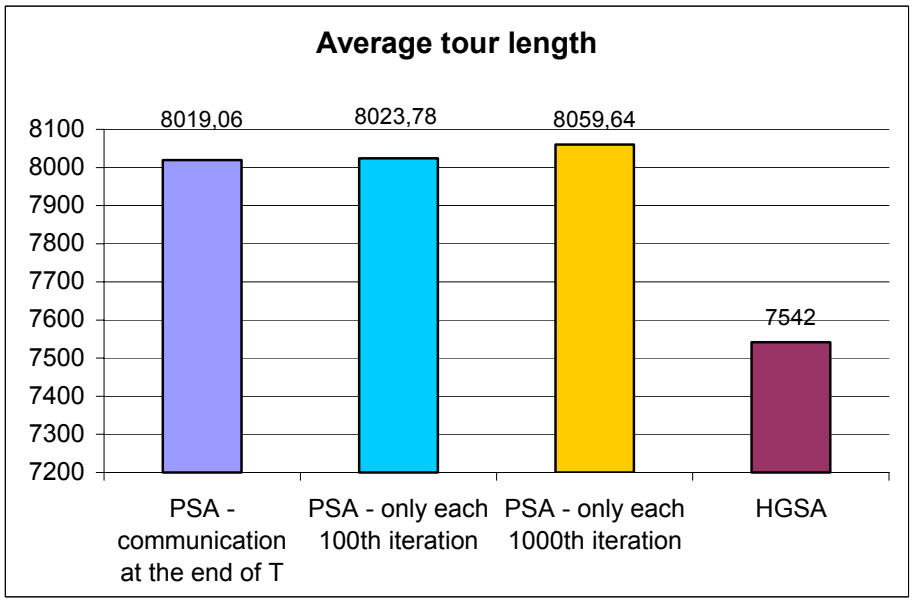


Figure 37: Average tour length of TSP 52 for HGSA and three versions of PSA.

In Fig. 38 the computational time and average tour length is shown. It is evident that the best solution provides HGSA and its computation time is equal to the fastest PSA version.

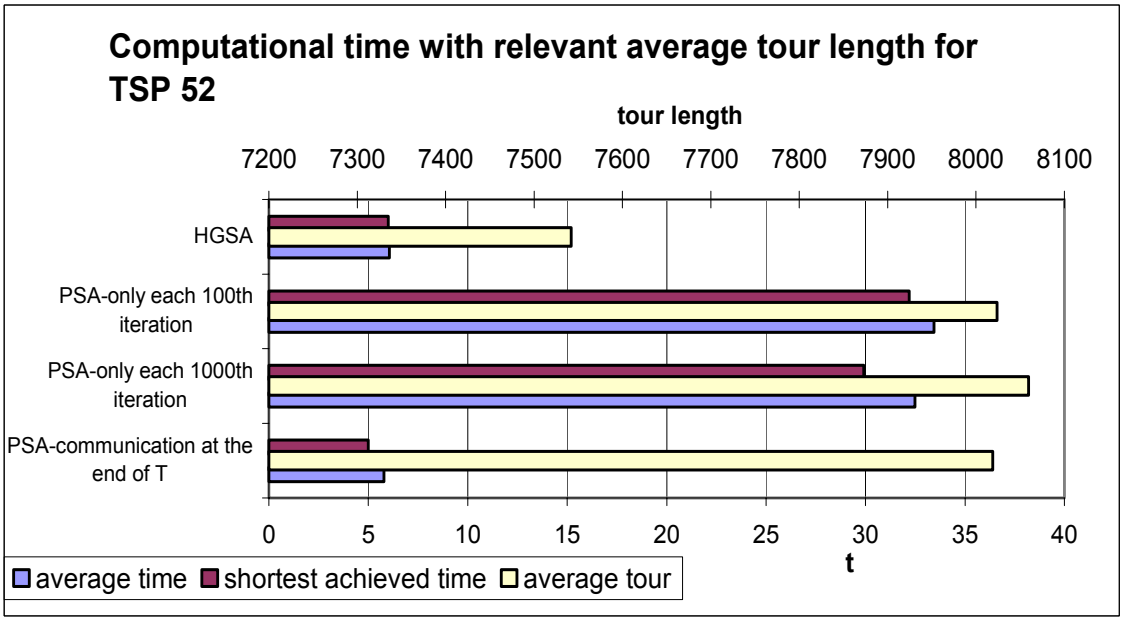


Figure 38: Computational time with relevant average tour length for HGSA and PSA versions.

An extra experiment was applied for comparison the performance of HGSA with PAGASA [52] on the TSP benchmark of 100 cities. HGSA achieves the tour length equals to 21295, which is better than the tour length 21443 achieved by PAGASA. Using the benchmark of 79 cities, PAGASA algorithm achieved the best tour length of 540 units and HGSA achieved 538 unit tour length, which is the global optimum.

6.5 Summary

We have developed a new hybrid optimization algorithm HGSA as an aggregation of parallel simulated annealing PSA and genetic algorithm. HGSA was built on the parallel SA version, which achieved a good trade-off between achieving the quality (sub)optimal solutions and computing time. In all tested versions of PSA and HGSA, heuristic was not used, because we compared the abilities of proposed algorithms between each other with-

out utilization of accelerating methods to local searching. We have tested HGSA on three benchmarks of the travelling salesman problems: TSP52, TSP79 and TSP100. The comparison of the performance of HGSA and PSA was realized. HGSA algorithm achieved the global optimum in each of 15 runs for TSP52. The PSA versions received only a local solution. The execution time was the same as the fastest PSA version. Comparison of the speed of convergence and achieved solution, proposed HGSA appears to be a very good compromise. Another experiment was arranged as a comparison of HGSA and a version of hybrid PSA called PAGASA published in [52]. HGSA outperforms PAGASA in all tested benchmark.

HGSA achieved very good results in all tested benchmark and it appears as feasible algorithm for solving some complex problem in practise. This hypothesis will be verified in the next chapter on the problem of collective communication scheduling for arbitrary interconnection networks.

Chapter 7

Evolutionary Design of Collective Communication

As chip multiprocessors are quickly penetrating new application areas in network and media processing, their interconnection architectures become an object for optimization. Collective communications involving all processors are frequently used in the solution of demanding problems in parallel and their timing complexity has a dramatic impact on performance.

In this chapter, we describe the application of hybrid evolutionary algorithm HGSA to scheduling collective communications on interconnection networks of parallel computers. The goal of the proposed algorithms is to find a deadlock-free and conflict-free schedule of a collective communication with the number of steps as close as possible to the lower bounds derived analytically, see Table 1.

The optimization problem using HGSA evolutionary algorithms may be decomposed into several phases. In the first phase, it is necessary to choose a suitable encoding of the problem into a chromosome. The second phase is a proper definition of the fitness function, which determines quality of the chromosome. The next phase is the design of the input data structure for the evolutionary algorithm. The last phase is experimental runs of the evolutionary algorithm and search for the best set of its parameters. The proper choice

of parameters can speed-up the convergence of the algorithm and simultaneously minimizes the probability of getting stuck in local minima.
The proposed algorithm is able not only to re-invent optimum schedules for known symmetric topologies like hyper-cubes, but it can find schedules even for any asymmetric or irregular topologies (Fig. 39) in case of general collective communications. Optimum schedules are destined for writing high-performance communication routines for application-specific networks on chip or communication libraries for general-purpose interconnection networks.

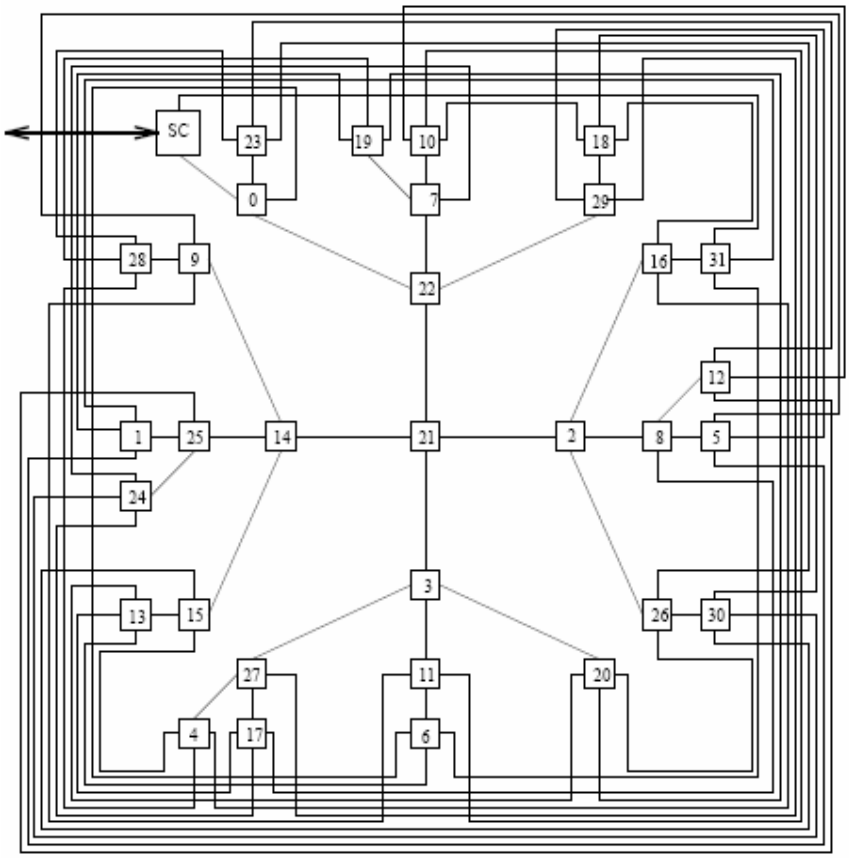


Figure 39: 32 processors in AMP topology. The SC node denotes a system controller (host computer) that sends input data to processing nodes and collects results.

7.1 Model of Communication

Implementation of a collective communications is inherently prone to a deadlock. If we let each node send or receive messages to or from more than one partner in a loop asynchronously, we still face the danger of a so called *fetch deadlock* (at least two nodes execute pending send operation and cannot receive). This is why a synchronized communication model is used and assume that the communication proceeds in synchronized rounds (steps). However, a deadlock doesn't occur only on channels but also on a router's buffer. Buffer deadlock is situation, where there is not enough space to store a message in the router's buffer. (However this paper has not dealt with buffer deadlock because it is assumed that routers' buffers are of sufficient capacity to store all incoming messages.)
This paper will focus especially on OAB, AAB, OAS, AAS and MNB communication patterns on interconnection networks with the following parameters:

- full-duplex links — messages can be transferred in both directions at the same time
- store-and-forward switching — whole packets are buffered in routers
- non-combining nodes — every received packet is sent on separately
- one-port/all-port communication facility — one/all ports of a single node can be used for communication simultaneously
- slim/fat node – one/more then one processor is connected with a router

These communication tasks [OA(B/S), AA(B/S) and MNB] cause the highest communication traffic and their timing overhead greatly depends on capabilities of particular communication hardware. OAB communication (as well as OAS) can be performed by sequentially sending the message to particular nodes. This system is very inefficient because only one node sends the message in each communication step. However, we can use a better technique using a broadcast tree when every node that received the message in previous communication step becomes an initiator of new multicast communication. If only node subset takes part in communication, we talk about multicast communication pattern. Consequently, the number of informed nodes increases by d^k instead by d, where d is the node degree and k is the number of communication step. One well-known method of how to solve OAS communication pattern, is the design *spanning tree*. However, on the other hand, construction of the spanning tree is difficult, namely in asymmetrical topologies. AAB communication is solved in a similar way as OAS, but the spanning tree is simultaneously generated by every node. This method termed *time-arc-disjoint trees* (TADT). However, this method is based on a similar principle, this method has the simi-

lar disadvantage. One possibility implementing AAS is to use *permutations separated* by barriers. The number of steps is then $P - 1$, where P is processor count. However, this number of steps is too large and can be reduced significantly.
The communication pattern MNB can be solved by MILP method (*Mixed Integer Linear Programming*). This method can achieve an exact solution, but, very long solutions are required for network sizes of any practical interest. The principle of the MILP can be formulated as a graph coloring.
Since we deal with regular and also irregular topologies, the only way to find optimal or sub-optimal schedules of communication steps is by utilizing combinatorial optimization. The following describes our method of achieving optimal or sub-optimal schedules of communication using combinatorial optimization.

7.2 Methodology of Design of Optimal Communication Schedules

The optimal communication schedule must be conflict-free and deadlock-free. Therefore we use a synchronized communication model and we assume that router's buffers' are of a sufficient capacity to avoid buffer's deadlocks, we don't deal with deadlocks, while conflicts may occur. If the conflict occurs, the schedule cannot be used in a real application. A conflict appears on a channel if two different messages want to utilize one communication channel in one direction in the same communication step, see Fig. 40. The main idea of both proposed methods is based on testing conflict-free condition.
For the scheduling of optimal collective communication for arbitrary topologies and communication patterns two, differing, methods can be considered.

7.2.1 Searching of Conflicts

This first method is based on the fact, that an optimal communication is one complex problem. This method of solution of investigated problem is possible, but it is very time-consuming and also the probability of achieving an optimal schedule rapidly degrades for topologies with a higher number of nodes. Really, it works for topologies up to 23 nodes

by using AA (All-to-All) and 64 nodes by using OA (One-to-All) communication pattern. Mainly for AA, it is relatively small topologies. The algorithm searches for an appropriate path (it is able to create an optimal schedule) between sending and receiving node and simultaneously it must correctly set a communication step to individual channels on the investigated path. This technique has a serious problem. It is not able to recognize whether it is possible to create optimal schedule by the selected path. Therefore it can happen, mainly for complex topologies, that a correctly selected path with the wrong setting of a communication step of some channel is replaced by an unsuitable path (it is not able to create a conflict-free schedule). In other words, the algorithm does not know, whether the whole path must be replaced or only to modify the communication step to channel or channels. Selecting an incorrect channel or set of incorrect channels is also a very complex problem for reconfiguration of communication step.
The next demand, which increases the execution time, is detection number of conflicts. The demand for conflict detection is performed in every modification of schedule. The conflicts are detected in the proposed schedule by the mutual checking of paths as to whether they use the same channel for communication. This is repeated for every communication channel.

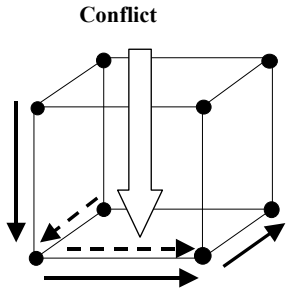


Figure 40: Conflict on a communication channel.

7.2.2 Prediction of Conflicts

From above mentioned difficulties, the new method was developed for the design of optimal schedule, which partitions collective communication scheduling into two complex sub-problems. The first sub-problem is defined as a search for appropriate paths (it is able to create an optimal schedule) between source and destination node. The second sub-

problem is stated as that of finding a conflict-free communication step to each channel for each found path. Without this partitioning the basic complex problem the method would need to be solved simultaneously the two sub-problems, as described in previous section.
The time of finding the optimal schedule can be reduced by the use of conflict prediction. It is possible to discover during the solution of the first sub-problem, whether the communication schedule will be conflict-free or not (detailed description is bellow). Using this technique of prediction, it is not necessary to assign the communication step to individual channel and detect, if the conflict occurs.
Necessary (but not sufficient) conditions for a schedule to be conflict-free are:

1. Utilization of investigated channel in one step in one direction equals at most or just one.
2. Utilization of each channel in one direction equals at most the number of communication steps S of the whole schedule. It can be described by equation (7.1), where the number of communication steps of the whole schedule is equal or greater than the number of the utilization of investigated channel in the whole schedule.

$$S >= Rc \tag{7.1}$$

where S is the desired number of communication steps of whole schedule, which is set by user at the beginning of the program as the input parameter and R_C is the number of all paths, which utilize the investigated channel.
So the conflict appears in case of inequality:

$$S < Rc \tag{7.2}$$

The next case of the conflict detection in the designed schedule expresses the situation that it is not possible to assign a communication step to the channel to be conflict-free although the equation (7.1) is true. This case can be described by the equation (7.3) with the interval (7.4), which includes all possible communication steps for the investigated channel.

$$bound = (S - (L\text{-}O)) \tag{7.3}$$

where *bound* is the higher border of communication steps of the investigated channel, L is the length of the investigated path and O is the channel position on the path. Finally, the communication step of an investigated channel on the path is chosen from the interval, where lower bound is the channel position O and the higher bound is value calculated from the equation (7.3):

$$step \in \langle O, bound \rangle \tag{7.4}$$

In the case, that it is impossible to choose two different values from interval (7.4) for the investigated channel into two different paths, the conflict appears in the generated communication schedule. This situation is illustrated by the example given in Table 4. This table explains the idea of conflicts prediction. In the first column is a set of investigated paths (from the generated schedule), which contains the investigated channel *8_ 9*. In the second column there are intervals of possible communication steps to investigated channel on the investigated path.

path	**interval** of all possible communication steps
2 0 *8 9* 25	3
8 9	1
14 12 *8 9*	3, 4
16 12 *8 9*	3, 4

Table 4: Assignment of communication steps to channel from the interval (7.4).

To illustrate the above results of Table 4, we have path "14_12_8_9" and investigated channel is 8_9. The number of communication steps of whole schedule $S = 4$, then position of the investigated channel 8_9 on the path is 3, i.e. $O = 3$ and the length of the path is 3, i.e. $L = 3$. According to equation (7.3),

$$bound = (S - (L - O)) = (4 - (3 - 3)) = 4.$$

Substituting the values into interval (7.4) we get

$$step \in \langle O, bound \rangle = \langle 3, 4 \rangle.$$

These inequation (7.2), equation (7.3) and interval (7.4) perform only the detection of the conflict and also the evaluation of the fitness function.

7.3 Input Data

At the beginning, the program reads its input data from a specified configuration file. It contains the values of the following parameters:

- name of the file with a description of the network topology (its graph)
- the target number of communication steps for communication task
- source node (used only for OAB and OAS)
- starting temperature
- cooling gradient
- the number of iterations of Metropolis algorithm
- frequency of mutual communication master-slave
- mutation probability
- iteration, at which the parameters of function of the communication step setting are changed according to predefined values

node	neighbor 1	neighbor 2	neighbor 3	neighbor 4
0	1	3		
1	0	4	2	
2	1	5		
3	0	4	6	
4	1	3	5	7
5	2	4	8	
6	3	7		
7	4	5	8	
8	5	7		

Table 5: 9-processor Mesh routing table.

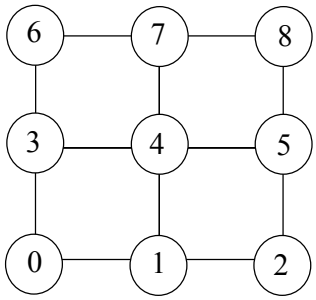


Figure 41: 9-processor Mesh configuration.

.

Description of the network topology for which a particular collective communication is being optimized is specified in a separate *topology file*. This file contains a list of each node's neighbors, where two nodes are considered to be neighbors only if they are connected by a single direct link. Table 5 shows sample data which describe eight-processor Mesh topology shown in Fig. 41. Further description of the other parameters (e.g. set of senders/receivers, the number of processors at a node) is given in the following sections.

7.4 Search of The Shortest Paths

This algorithm generates all the shortest paths and saves them in the operating memory into a specific data structure. The algorithm [2] is inspired by the breadth-first search algorithms (BFS). BFS is based on the searching of a graph, where the source processor is chosen as a root. The edges create a tree used in the searching process. A tree is gradually constructed, one level at a time, from the root that is assigned an index of a source node. When a new level of the tree is generated, every node at the lowest level (leaf) is expanded. When a node is expanded, its successors are determined as all its direct neighbors except those, which are already located at higher levels of the tree (it is necessary to avoid cycles). The construction of the tree is finished when a value of at least one leaf is equal to the index of a destination node. Destination leaves' indices confirm the existence of searched paths, which are then stored as sequences of incident node indices.
A sample tree constructed while searching for shortest paths from node 0 to node 5 in the 9-processor Mesh topology is shown in Fig. 42. Three paths were actually found in the tree: 0-1-2-5, 0-1-4-5 and 0-3-4-5. If one-to-all communication is being scheduled, only paths from a single source node to all other nodes are searched for. On the other hand, for optimization of all-to-all communication AA, all paths between every pair of source-destination nodes are considered.
In certain cases when the target topology has nonuniform numbers of links per node, it may happen that an optimal routing schedule cannot be constructed from a set of the only shortest paths. Use of only the shortest paths may cause heavy utilizing of some links but the rare utilization of others, which can prevent the finding of an optimal solution. To avoid this problem, the algorithm must consider not only the shortest paths but also paths whose length may be longer. This approach can be used only for small topologies, because in case of the large topologies the searching space of possible paths increases dramatically. For example, consider with 9-processor Mesh, AA communication pattern and

the paths that are elongated by 1 according to shortest paths; whole number of paths is increased from 112 to 202, for more complex topologies with 32 processors and more, rising to thousands of paths. Consequently, optimization algorithms are not able to search an optimal solution for more complex topologies by non-minimal routing.

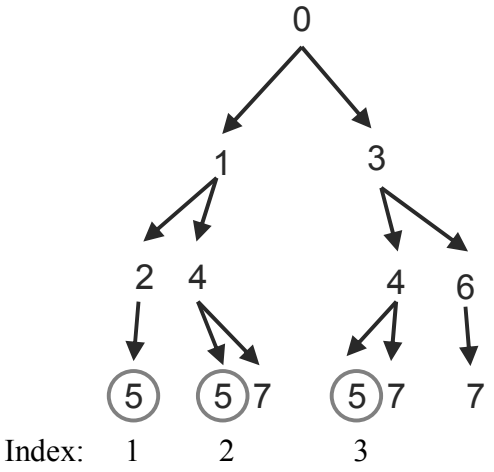


Figure 42: Construction of the shortest paths list from node 0 to node 5 in the 9-processor Mesh topology.

7.5 Solution Encoding

Very simple encoding has been chosen for HGSA. The solution is represented by the chromosome as an array of genes, see Fig. 43. Every gene encodes schedule of a single message transmission from a given source node to a destination node. The gene's position (locus) represents the source and the destination node for a message. The source node is calculated from the gene's position by the equation (7.5) and destination node according to the equation (7.6).

$$source\_node = gene's\_position \ \ div \ \ P \tag{7.5}$$

$$destination\_node = gene's\_ position \ \ mod \ \ P \tag{7.6}$$

where P is the number of nodes in target architecture.
Every gene consists of an identity of a path which is used for routing the message and a sequence of time steps at which every node on the path (except the destination node) sends the message to the next node on the path. Assignment of the time steps is constrained by a rule that transmission of any message must not take more steps than a maximum value, which is set as an input parameter when the program is run. The number of genes G in every chromosome is determined by the type of communication to be scheduled and by the number of nodes P as:

$$G = P - 1 \tag{7.7}$$

for one-to-all communication pattern.

$$G = P*(P - 1) \tag{7.8}$$

for all-to-all communication pattern.

$$G = M*N - Q \tag{7.9}$$

for many-to-many communication pattern, where M – set of senders, N - set of receivers and Q – set of simultaneous senders and receivers.

The main advantage of this encoding is a relatively short chromosome and the absence of inadmissible solutions (every message is transmitted from a source to a destination). The main disadvantage is a large number of possible values/alleles of the first gene component, which represents index of shortest path - the number of the paths rapidly increases with the distance from source *src* to destination *dst*.

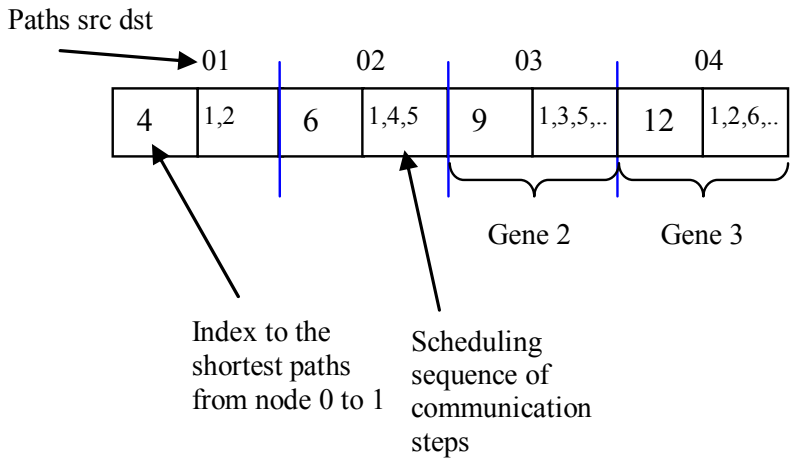


Figure 43: The structure of chromosome.

7.6 Definition of Fitness Function

Having defined solution encoding, we can now describe fitness function. For our representation of chromosomes, fitness of an individual is directly derived from its cost, which is determined by a number of conflicts among genes of its chromosome. It is obvious that we are concerned only with such final solutions which have no conflicts, i.e. whose cost is equal to zero. Only these conflict-free schedules are applicable in real applications.

7.6.1 The Fitness Function Based on Searching of Conflicts

Fitness function based on searching of conflicts can be described as follow:
Firstly, we create the set M, which has elements as much as value S of the required number of steps.

$$M = \{1,\ldots, S\} \subseteq N \tag{7.10}$$

From the set M we create the set Q, which has elements as much as paths using the investigated channel.

$$Q = \{M_1,\ldots., M_{Rc}\} \tag{7.11}$$

Next, we create the set T, for which holds:

$$T = \{y|\ y \text{ is just one randomly selected element from every } M \in Q\} \tag{7.12}$$

Now, we define the set R, whose cardinality is equal the number of paths using the investigated channel:

$$R = \{1,\ldots, Rc\} \subseteq N \tag{7.13}$$

Further, we define function Ω: $T \rightarrow R$, which assign to every $t \in T$ element $r \in R$, so $t = r$. In the case, that Ω is only surjection a conflict appears, if it is bijection then a conflict does not occur.
Then, we create the set K, containing elements from T mapping on the same element from the set R:

$$K = \{x \mid \exists y \in T,\ y \neq x:\ \Omega(x) = \Omega(y)\} \tag{7.14}$$

Finally, the number of conflicts in investigated channel $conflict_{new}$ can be described as:

$$conflict_{new} = |K| \tag{7.15}$$

Now, we can evolve the pattern of total number of conflicts:

$$\text{conflict} = \sum \text{conflict}_{new} \tag{7.16}$$

7.6.2 The Fitness Function Based on Prediction of Conflicts

In the first phase the prediction is used to find out how many conflicts appear in the whole schedule. This is based on the inequation (7.2), on the equation (7.3) and on the interval (7.4). The counting of conflicts can be formally described as follows:
On interval (7.4) it can be viewed as the set, because it is subset of nonnegative numbers:

$$\text{Step} \subseteq \text{N} \tag{7.17}$$

From the set *Step* we create the set *W*, which has elements as much as paths using the investigated channel:

$$\text{W} = \{\text{Step}_1, \ldots, \text{Step}_{Rc}\} \tag{7.18}$$

Now, we create the set *U*, for which holds:

$$\text{U} = \{\text{y} |\ \text{y is just one randomly selected element from every Step} \in \text{W}\} \tag{7.19}$$

Further, we define the set *R*, whose cardinality is equal the number of paths using the investigated channel:

$$\text{R} = \{1, \ldots, Rc\} \subseteq \text{N} \tag{7.20}$$

Now, we define function Ω: $U \rightarrow R$, which assign to every $u \in U$ element $r \in R$, so $u = r$. In case, that Ω is only surjection a conflict appears, if it is bijection then a conflict does not occur.
Then, we create the set *G*, containing elements from *U* mapping on the same element from the set *R*:

$$\text{G} = \{\text{x} \mid \exists \text{y} \in \text{U},\ \text{y} \neq \text{x}:\ \Omega(\text{x}) = \Omega(\text{y})\} \tag{7.21}$$

Finally, the number of conflicts in the investigated channel $conflict_{new}$ can be described as:

$$\begin{aligned} &1.\ \ \text{conflict}_{new} = (Rc - \text{S}) + |\text{G}|, \ \text{ if } Rc > \text{S} \\ &2.\ \ \text{conflict}_{new} = |\text{G}|\ , \ \text{otherwise} \end{aligned} \tag{7.22}$$

Now, we can evolve the pattern of total number of conflicts:

$$\text{conflict} = \Sigma\, \text{conflict}_{\text{new}} \tag{7.23}$$

The parameters *S* and *Rc* are defined in section 7.2.2, $conflict_{new}$ is the number of conflicts detected for the investigated channel and *conflict* is the number of conflicts of the whole schedule.

In case, that prediction is conflict-free (the number of conflicts is equal to 0), the investigated paths create admissible communication schedule. Now, after the prediction phase the step assignment must be found for to each channel for all paths. Step assignment is based on equation (7.24) and interval (7.25).

$$difference = (S - CS_B) - (L - O) \tag{7.24}$$

$$comm\_step = \text{rand} \langle O,\ O+difference \rangle \tag{7.25}$$

where *difference* is the number of possible communication steps of the investigated channel, CS_B is the communication step of the previous channel, *L* is the length (number of channels) of the investigated path. The equation (7.24) with the interval (7.25) performs the counting of conflicts and also the communication step assignment to the channel on the investigated paths.
In the case, that a communication step is randomly chosen to one channel in two different paths, the conflict appears. In this case another randomly generated communication step is applied to that channel for the conflict path.
However, the situation can appear, when the equation (7.23) is equal 0, but the conflict-free schedule cannot be constructed, see Tables 4 and 6. This situation is based on the prediction conditions, which are not sufficient, but only necessary. In this case, the computation must be returned to the first phase, i.e. searching for appropriate paths. The prediction only indicates that it is impossible to create conflict-free schedule or it might be created some conflict-free schedule, but it is not sure.

Path	**Interval** (7.25)
2 0 *8 9* 25	3
8 9	1
14 12 *8 9*	4
16 12 *8 9*	3 4

Table 6: An assignment of communication steps to channel from the interval (7.25).

In Table 6 the possibility of the communication step assignment to investigated channel on the investigated path from the interval (7.25) is illustrated. Tables 4 and 6 are very similar but the difference is in the third row. The possible step interval in Table 6 has only one value instead two, as in Table 4. The step interval in the channel *8_ 9* cannot include the 3rd step because this communication step is assigned to channel *12_ 8,* see (7.24).

7.7 Heuristic

In our proposed algorithm one heuristic is used to speed up the convergence to a sub-optimal solution. It decreases the probability of being trapped in local optima during the execution. This heuristic is only used for the broadcast communication for the maximal utilization of channels.

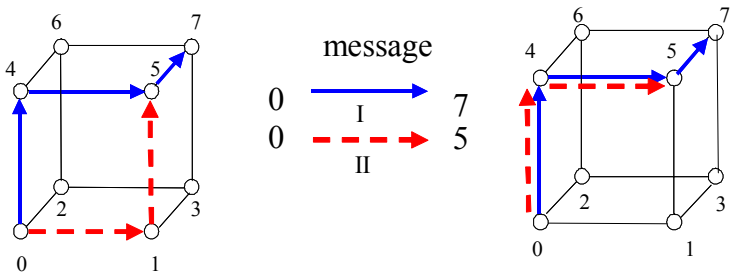


Figure 44: Modification of shorter path according to longer path.

The principle is simple. Path *I* from source node to destination node leads through a sequence of intermediate nodes. If other path *II*, which has shorter length and her destination node is occurred on the path *I* and has the same source node as path *I*, thus this path *II* changes according to path *I* (path *II* will go through the same nodes as path *I*), see Fig. 44.

7.8 Generalization of New Proposed Algorithm

So far, it has been unnecessary to deal with existence of channel between router and processor, since we supposed an all-port node model. In all-port model, there are channels as much as node degree between router and processor and every message, which is delivered to a router, is immediately sent from this router to a processor at the same time. There was simplification to a large extent for implementation and simultaneously for our algorithm, because the state space was reduced for searching. Furthermore, we didn't need to deal with the number of processor in one node and also whether processors transmit and/or receive. We supposed one processor to a router. For solving these problems, our algorithm must have been extended and generalized on new communication pattern and topologies with fat nodes.

7.8.1 Fat Topologies

The term *fat topologies*, describes topologies with fat nodes, where nodes have more than one processor. Generally, each node can have a different number of processors and therefore the information concerning how many processors each node contains must be saved in a topology file. In this file, a value is assigned to every node, which determines the number of processors at each node and then a description of network topology to be followed.

The next change is one-port node model in contrast to the all-port model that was used in direct networks. Hence, we must take into consideration the channels between router and processor. In this node model, there aren't enough channels to deliver every message from the router to a processor at the same time. This change becomes evident partly by increasing the number of new paths lead through the router to all of its processors and also by extension of individual paths about two channels, at the beginning and at the end of the path. Dependent on increasing the number of the shortest paths, the search space is expanded and therefore the whole problem must be encoded by a higher number of genes in chromosome.

7.8.2 Many-to-Many Broadcast Communication

We will write many-to-many broadcast as MNB, because the letters M and N represent different sets of senders and receivers. According to notation order, the first letter *M* represents senders and the second letter *N* represents receivers.
In the literature, we can meet with theoretical lower bounds for one-to-all and all-to-all communication patterns but not for the many-to-many communication pattern. Since we dealt with MNB, we tried to derive the lower bounds as follow:
MNB is limited by OAB or AAB bound from Table 1, whichever is greater, because some nodes may absorb *M* messages, only when $N \subseteq M$ then nodes absorb M–1 messages.

- one-port model

$$T_{MNB} = \max (\lceil \log_2 N \rceil, D, M - 1), \text{ where } N \subseteq M \quad (7.26)$$

$$T_{MNB} = \max (\lceil \log_2 N \rceil, D, M), \text{ otherwise} \quad (7.27)$$

- all-port model

$$T_{MNB} = \max (D, \lceil (M - 1)/d \rceil), \text{ where } N \subseteq M \quad (7.28)$$

$$T_{MNB} = \max (D, \lceil M/d \rceil), \text{ otherwise} \quad (7.29)$$

These theoretical lower bounds were compared by computed bounds, which were achieved by our proposed algorithm and simultaneously we could have proved accuracy of these analytically derived and computed bounds.
To illustrate the above results, MNB on the Fat Octagon in Fig. 45 all tree possibilities of sending/receiving processors will be analyzed:

1. We have $M = 8$ sending nodes, $N = 16$ receiving nodes, and $Q = 8$ nodes in intersection $M \cap N$. According to equation (7.27),
   $T_{MNB} = \max (\lceil \log_2 16 \rceil, 8) = \max (4, 8) = 8$ steps.

2. We have $M = 8$ sending nodes, $N = 8$ receiving nodes, and $Q = 8$ nodes. According to equation (7.26),
   $T_{MNB} = \max (\lceil \log_2 8 \rceil, 8 - 1) = \max (3, 7) = 7$ steps

3. We have $M = 8$ sending nodes, $N = 8$ receiving nodes, and $Q = 0$ nodes. According to equation (7.27),
   $T_{MNB} = \max (\lceil \log_2 8 \rceil, 8) = \max (3, 8) = 8$ steps.

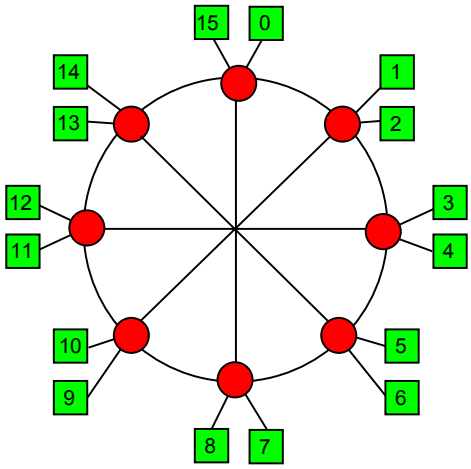


Figure 45: Fat Octagon topology with full duplex links and one-port model.

Only one change needed to be made to our algorithm to implement the MNB communication pattern. This one change related to the recognition of senders and receivers by the proposed algorithm. This information is written to the topology file as two lists, the list of senders and receivers. Each element of the list consists of two values, which are separate by "/". The first value of element is an index of node and the second value is an index of processor at this node. Every processor of one node is evaluated by index 0 to *n*. It may appear simple, each processor calls by a unique index. However the ability to design of MNB was added after implementation of OA and AA communication patterns and therefore we chose this approach, which minimized the number of implemented changes in the proposed algorithm. Consequently, the topology file appears as in Fig. 46:

1. sum of processors and maximal number of links per processor
2. number of processor at each node
3. source processors (first number is node, second number is processor)
4. destination processors
5. matrix of direct neighbors

```
# sum of processors and maximal number of links per processor
8 3

# number of processor at the node
0 2
1 2
2 2
…

# source processors (first position is node, second position is processor)
0/0
0/1
1/0
1/1
…

# destination processors (first position is node, second position is processor)
4/0
4/1
5/0
5/1
…

# node_num and its direct neighbors
0 1 4 7
1 0 2 5
2 1 3 6
…
```

Figure 46: Illustration of a file, in which a network topology is description for MNB.

7.9 Analyze of Proposed Algorithm

A primary architecture intended for testing the proposed program was a cluster of workstations because it was readily available. Since the message passing was programmed using MPI library [47] routines, the program can easily be compiled and run on any architecture (clusters of workstations, MPPs, SMPs, etc.) for which an implementation of MPI standard is available.

To achieve the best parallel performance of the proposed program, analysis was used to identify bottlenecks. It also appeared that analysis was necessary because test executions of the initial implementation showed that its performance dramatically reduces with the increasing number of processors P of the optimized topology. The problem was severe especially for all-to-all communication, which requires very long chromosomes.

Analysis of the proposed algorithm showed that a majority of processing time was spent in evaluation of a fitness function, which determines cost of individual in every iteration. On the contrary, the time taken by genetic operators and communication master-slave is negligible. Even though the search for shortest paths requires also significant time, it is performed only once at the beginning of the computation and does not affect the overall run-time as much as the evolutionary algorithm.

The reason for this bad performance was that the initial implementation of the fitness function used a relatively trivial approach to determining a cost of a chromosome. The cost is defined as a number of conflicts (the same channel used by different paths in the same step) by checking all possible pairs of genes whether they use the same channel for communication. This is repeated for every channel. So the main part of the calculation uses two nested loops both iterating through all genes of a chromosome yielding an asymptotic complexity $O(G^2)$, where G is a number of genes determined by the pattern of communication to be scheduled. The next very complex reason for increasing computation time was, that the algorithm wasn't able to recognize, if the whole selected path must be replaced or only modified communication step to channel or channels on this selected path. Therefore this happened mainly for the more complex topologies very often that a correctly selected path (it is able to create conflict-free create schedule) was replaced by an unsuitable path (it is not able to create conflict-free schedule), due only to the reason that it was improperly set the communication step to some channel on correct path.

Although some optimizations were already used to simplify a body of each loop, e.g. a set of all genes which already caused a conflict was constructed on the fly so that a single check of the set can avoid expensive path reconstructions, the complexity was still unacceptable. Especially if we consider the AA communication, which requires long chromosomes (number of genes proportional to P^2), we get a complexity $O(P^4)$ which is unfeasible for optimization of architectures with more than a few processors, see Table 12 and 14.

To decrease the high complexity, methodology of prediction was developed. The principle of prediction is based on a partition scheduling problem of a collective communication into two sub-problems. The first sub-problem is defined as a search for appropriate paths between source and destination node. The second sub-problem is stated as a finding of conflict-free communication step to each channel for each founded path. It is possible to discover during the solution of the first sub-problem, whether the communication

schedule will be conflicting. If the prediction doesn't find a conflict, the second part of problem can start and try to set up a communication step to each channel. Utilization of prediction we needn't deal with investigation, if a conflict is caused by unsuitable selection of path or unsuitable setting of steps to individual channel on the path. In every iteration we know what to change (path or step) to achieve the searched schedule.
Performance of the new approach is improved since it reduces an asymptotic complexity to $O(G)$ from the original $O(G^2)$. The complexity of the loop's body is similar or even smaller than in the original implementation so that a multiplicative constant hidden in the asymptotic complexity formula is decreased.
The time spent by individual parts of algorithm is showed in Table 7. Analyze was performed on 32-slim node bidirectional all-port hyper-cube. We focus on implementation based on prediction, because such implementation outperforms the original implementation in time of execution, probability of achieving a global optimum, speed of convergence and mainly in the greater number of nodes in the investigated topologies as can be see in chapter 7.10.

Parts of algorithm	Time [%]	
	Broadcast	Scatter
Search of the shortest paths	0.771	0.0263
Prediction	76.256	75.282
Setting of com. steps	8.416	0.0378
Genetic operators	0.003	0.004
Communication M-S *)	0.135	1.505
Synchronization	11.194	18.231
Others	3.229	4.9177

Table 7: Time complexity of individual parts of algorithm based on conflict prediction in percent on the 32-slim node bidirectional all-port hyper-cube; *) this value corresponds with frequency of communication n*100 iteration ($n > 1$) from 300 iterations of Metropolis algorithm.

From previous chapters it follow that the fitness function is composed from two phases:

1. the prediction phase (search for appropriate paths) and after finishing the prediction
2. the phase of communication steps setting.

In the Table 7, are presented average values from 10 runs of each communication patterns. These values of variables are only orientation and illustrate properties of proposed algorithm. The time strongly depends on properties of interconnection networks and also on the setting of algorithm's input parameters. For example, the time of *synchronization* is for large topologies relatively small, but it doesn't remain true for small topologies. It can be as high as ten percent. Since, we chose the architecture master-slave, when all participants communicate simultaneously, synchronization is necessary. A similar case is the time of *communication* master-slave. The time of communications is based on frequency, which is defined as the input parameter by user.
From the Table 7 is evident that we can recognize relationships between communication patterns of broadcast and scatter. Although the time of *prediction* is nearly the same, the time of the *setting of communication steps* is very different. It is based on definition of both communication patterns. In scatter, there is a problem to search suitable paths to create an optima schedule in comparison with broadcast, where the problem is in correctly setting of the communication steps to suitable paths. It is clear from the fourth row of the Table 7 (Setting of the communication steps). There are significantly stronger constraint rules in broadcast than in the scatter for implementation. In broadcast, the messages, which travel from the same sending node, pass through the same intermediate nodes in the same communication steps. In scatter, the rule is opposite. No message is allowed to be sent through the same channel in the same step. Therefore, there is greater problem to search for suitable paths, over which these two messages should travel, than with correctly set communication steps.
We analyzed our proposed algorithm on using the *same topology* (32-node hyper-cube), but the real time was very different; for AAB it was 37.5 minutes and for AAS it was 3.15 hours. From this difference of real times, it can appear that individual times of prediction are similar at both AAB and AAS. However this is not a valid conclusion. The comparison of topologies, which has the same executing time, would have better predicative ability. For example: for AAB 64-node hyper-cube, where the time of prediction is 53.2% and the time of setting the communication steps is 36.5%, and for AAS 32-node hyper-cube, where the time of prediction is 75.4% and 0.1% is the time of setting the communication steps. We assume that the model of topologies is the same in both cases.
The item *Others* of Table 7 involves only helping operations such as printing of schedule, creating, computing and deleting helping variables, data conditioning etc.

7.10 Experimental Results

In the first experiment the proposed algorithm, using HGSA, was verified using a variety of multiprocessor topologies (e.g. Midimew, K-Ring, Octagon...). We examined the algorithm from the aspect of the ability to design optimal schedule in any arbitrary topology. All topologies had 8 nodes (except 10-node Moore topology) and the regular and the irregular (AMP with SC [53], Ladder and Twisted Ladder) topologies were examined. The results achieved of the optimal/sub-optimal communication schedule and the theoretical lower bound (see Table 1) on the number of the communication steps of the tested topologies are presented in the Table 9 and 10. An optimal communication schedule was achieved in all tested topologies, which is illustrated in the third and fifth column in the Table 9 and 10.

The control parameters of proposed algorithm were set to the same values for all runs, i.e. P_c computers in the master-slave architecture, each computer work on one individual, the length of communication interval between master and slave was each I_c's iterations of Metropolis algorithm, the start temperature equal to T_{max}, K_{max} iterations in each temperature phases, gradient of cooling equals to *Alpha*. 30 runs of HGSA were performed for each topology. In Table 8 are shown the values of parameters, which were reached experimentally on hyper-cube. The target was to reach optimal schedule in the shortest possible time.

K_{max}	200
T_{max}	100
T_{min}	0,1
Alpha	0,99
I_c	50
P_c	10
Number of individuals	10

Table 8: The value of control parameters of proposed algorithm.

Topology	OAB Lower bounds	OAB HGSA	AAB Lower bounds	AAB HGSA
Hyper-cube	2	2	3	3
Hyper-cube with body diagonal	2	2	2	2
AMP with SC	-	2	-	3
AMP without SC	2	2	2	2
K-ring	2	2	2	2
Midimew	2	2	2	2
Moore	2	2	3	3
Octagon	2	2	3	3
Ladder	-	3, 4	-	4
Twisted ladder	-	2, 3	-	5 *)

Table 9: Experimental results and the theoretical lower bound of the broadcast collective communication for the all-port topologies with 8-nodes.

The asterisk "*)" indicates a non-optimal schedule. During the search for the optimum schedule, it may be necessary to include not only multiple minimum paths, but, at times, non-minimum paths! This is the case of OAS and AAB for Twisted ladder topology. This problem doesn't occur in AAS communication pattern, because this communication is more complex and the total time of communication compensates a delay on a channel. It is possible to consider non-minimal paths for such small topologies (8-node) but if we consider more complex topologies, this would lead to an enormous increase of possible paths from source to destinations and, thus, to prohibitive computer memory and time requirements. Therefore we didn't include this ability in our proposed algorithm.
Notation "–" in tables means that theoretical lower bound cannot be derived analytically as in the Table 1, because these topologies are irregular and mathematical methods cannot be applied.
For some topologies two values were achieved, see Ladders and Mesh topologies in Table 9 and 10. It is from the reason that one-to-all communications depend on the node degree of the source node, e.g. mesh topology has three values of d = 2 (corner node), 3 (edge node) and 4 (central node), see Table 11.

Topology	OAS Lower bounds	OAS HGSA	AAS Lower bounds	AAS HGSA
Hyper-cube	3	3	4	4
Hyper-cube with body diagonal	2	2	3	3
AMP with SC	-	3	-	5
AMP without SC	2	2	3	3
K-ring	2	2	3	3
Midimew	2	2	3	3
Moore	3	3	5	5
Octagon	3	3	4	4
Ladder	-	3, 4	-	8
Twisted ladder	-	4, 5 *)	-	6

Table 10: Experimental result and the theoretical lower bound of the scatter collective communication for the all-port topologies with 8-nodes.

In the second experiment the proposed algorithm was tested with broadcast communication pattern and with a higher number of nodes in three different architectures - hyper-cube, mesh and AMP. The number of nodes varied from 8 to 128. A hyper-cube has been chosen because of its regular topology with, known optimal scheduling, as it can serve as a convenient benchmark.

OAB and AAB communication complexities, measured by the number of communication steps in schedules found by proposed algorithm so far, are shown in Table 11 and 12 (columns three, four and five). The first column includes the node count in the target architectures and the second column presents the number of steps in the optimal schedule achievable for a hyper-cube (the reachable lower bound).

Finding an optimal schedule for the AAB communication pattern is a significant problem. The proposed algorithm is able to find an optimal schedule for AMP topology in all tested cases but for a greater number of nodes its success rate is significantly reduced. Our algorithm achieves global optimum in almost all tested topology. In only two cases the optimal solution failed to be achieved, at 64-node mesh and 128-node hyper-cube. However the achieved sub-optimal solutions are close to the optimum, see Table 12.

Number of nodes	Hyper-cube minimal	Hyper-cube HGSA	Mesh HGSA	AMP without SC HGSA
8	3	3	4, 3	2
16	4	4	6, 5, 4	-
23	-	-	-	3
32	5	5	-	3
36	-	-	10, 9, 8	-
42	-	-	-	3, 4
53	-	-	-	3
64	6	6	14, 13, 12	-
128	7	7	-	-

Table 11: Number of steps for OAB optimization.

Number of nodes	Hyper-cube minimal	Hyper-cube HGSA	Mesh HGSA	AMP without SC HGSA
8	3	3	-	3
16	4	4	8	-
23	-	-	-	8
32	7	7	-	11
36	-	-	18	
42	-	-	-	14
53	-	-	-	18
64	11	11	**33**	-
128	19	**20**	-	-

Table 12: Number of steps for AAB optimization (bold digits represent cases when lower bounds were not reached).

If we compare the solution gained with the help of the prediction and without prediction in scheduling of collective communication, it is evident, that the prediction provides rapid improvement, see Table 13. Finally, using prediction, final solution is superior and simultaneously the probability of achieving such a solution is higher.

Number of nodes	Hyper-cube		AMP	
	With Prediction	Without Prediction	With Prediction	Without Prediction
8	100%	100%	100%	100%
16	100%	60%	-	-
23	-	-	100%	70%
32	100%	0%	100%	10%
42	-	-	95%	0%
53	-	-	100%	0%
64	65%	0%	-	-

Table 13: Success rate of 15 runs in achieving the optimal communication scheduling for AAB.

The presented data of proposed algorithm deserves some comments. Firstly, OAB is quite a simple operation and therefore the algorithm is likely to find an optimal solution even for larger architectures up to 128 nodes. Optimal solutions have already been found for topologies with up to 64 nodes and quality sub-optimal solution for topology with up to 128 nodes for AAB. To achieve an optimal solution for the AAB communication pattern, many hours are required for more complex topologies. On the other hand, if we need an sub-optimal solution quickly, the proposed algorithms is allowed to accept a larger number of communication steps and the solution is found in much shorter time, see Fig. 47.

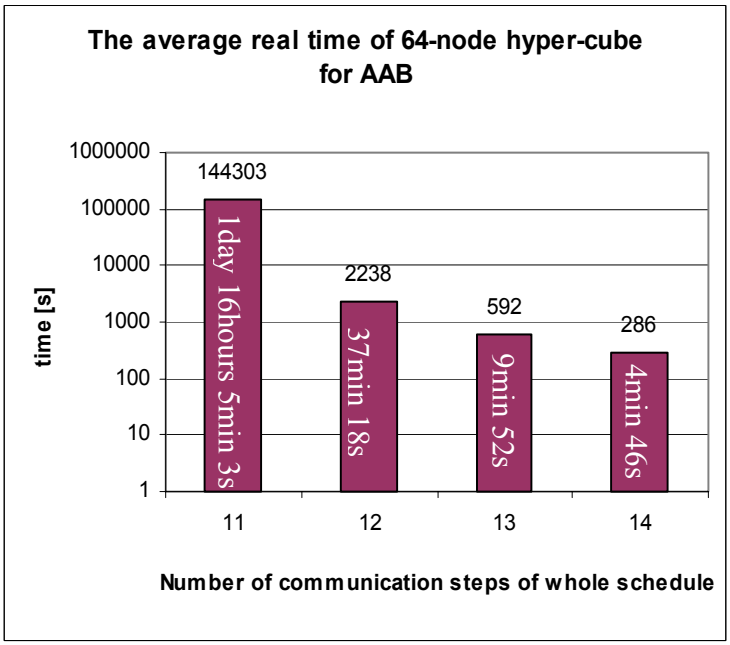


Figure 47: The real time complexity of AAB on 64-node hyper-cube with different number of communication steps.

The computational platform used was IBM BladeCenter® [54] with 12 HS20 blades, each fitted with 2 CPU Xeon 2,8GHz/533MHz, 1GB RAM, 40GB HD and with 14 HS20 blades each fitted with 2 CPU Xeon 3,2GHz/800MHz, 2GB RAM, 36GB interconnected by gigabit router-switch. Algorithm ran under Unix OS.
In Fig. 48, is shown the average time complexity of reaching global optima for four instances of 64-node hyper-cube in the term of the number of fitness function evaluations.

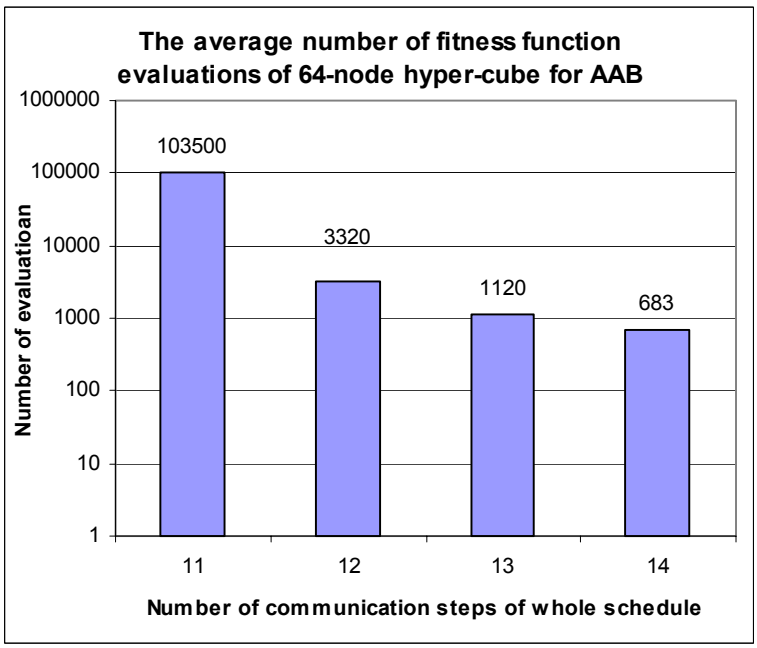


Figure 48: Time complexity of AAB on 64-node hyper-cube with different number of communication steps.

After implementation and testing of broadcast communication pattern, we modified our algorithm for further type of communication pattern – namely scatter. The proposed modification consists of the elimination of some modules from the code, since broadcast is much more complex than scatter in the sense of implementation, because broadcast has more constrain rules, which flow from definition of broadcast.

The achieving results for OAS and AAS are presented in the Table 14. Again as in previous case, the hyper-cube served as a convenient benchmark and simultaneously we tested this proposed algorithm on the irregular topology AMP. We achieved optimal solutions for OAS up to 128-node and for AAS we obtained optimal solution up to 32-node and quality sub-optimal solutions up to 128-node.

Finding of the optimal communication schedule for AAS presents the most complex problem of all four investigated communication patterns, because searching space of AAS is the greatest, see comparative graphs of the number of evaluations in Fig. 49, 50 and 51.

Number of nodes	OAS			AAS		
	Hyper-cube minimal	Hyper-cube HGSA	AMP without SC HGSA	Hyper-cube minimal	Hyper-cube HGSA	AMP without SC HGSA
8	3	3	2	4	4	4
16	4	4	-	8	8	-
23	-	-	6	-	-	13
32	7	7	9	16	16	20
42	-	-	11	-	-	29
53	-	-	13	-	-	40
64	11	11	-	32	**33**	-
128	19	19	-	64	**66**	-

Table 14: Number of steps for OAS and AAS optimization (bold digits represent cases when lower bounds were not reached).

The Table 15 illustrates the efficiency of the algorithm with the help of the prediction and without prediction. By utilization of the prediction, our algorithm is able to solve more complex topologies with higher probability of achieving the optimal schedule than without prediction. Algorithm using prediction provides significantly better solutions

Number of nodes	**Hyper-cube**		**AMP**	
	With Prediction	Without Prediction	With Prediction	Without Prediction
8	100%	100%	100%	100%
16	90%	40%	-	-
23	-	-	100%	40%
32	80%	0%	80%	0%
42	-	-	75%	0%
53	-	-	60%	0%
64	60%	0%	-	-

Table 15: Success rate of 15 runs for AAS communication scheduling (the communication complexity is illustrated in Table 14).

The graph of fitness function evaluations for OAB is not presented here, because conflicts do not occur at this communication. OAB is based on a broadcast tree. We can create this broadcast tree simply from the set of the shortest paths between any pair sender - receiver. Since, this set of minimal paths is created in the first phase of our algorithm, thus the whole algorithm is reduced when creating this set. Consequently, the total execution time is equal to creating time of the set of the shortest paths and therefore the number of fitness function evaluations is equal to *one* for arbitrary topology.

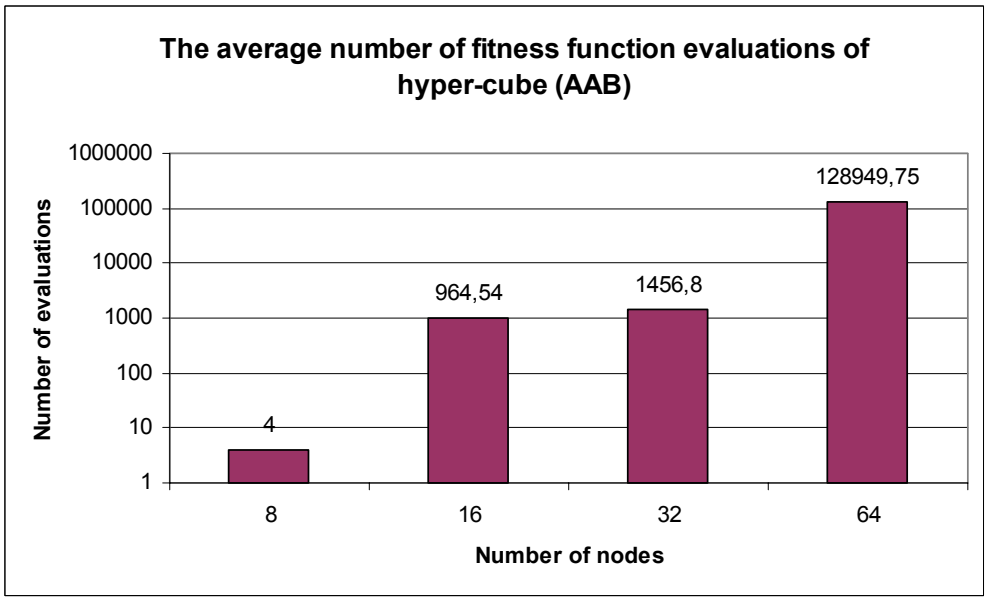


Figure 49: Time complexity of AAB.

Fig. 49 shows the average time complexity of reach optimal schedule for four instances of hyper-cubes in the term of the number of fitness function evaluations. In the case of 16-node and 32-node topology, the number of evaluations is almost the same. The time of AAB execution depends much more on properties of the interconnection networks than on the number of nodes.

Our proposed algorithm is very effective for OAS collective communication as the Fig. 50 presents. We are able to find the optimal solution up to 128-node topologies in a short time. It is for this reason that a very effective method was proposed to solving collective

communications and also the searching space is only n and not n^2 as in at AA communications.

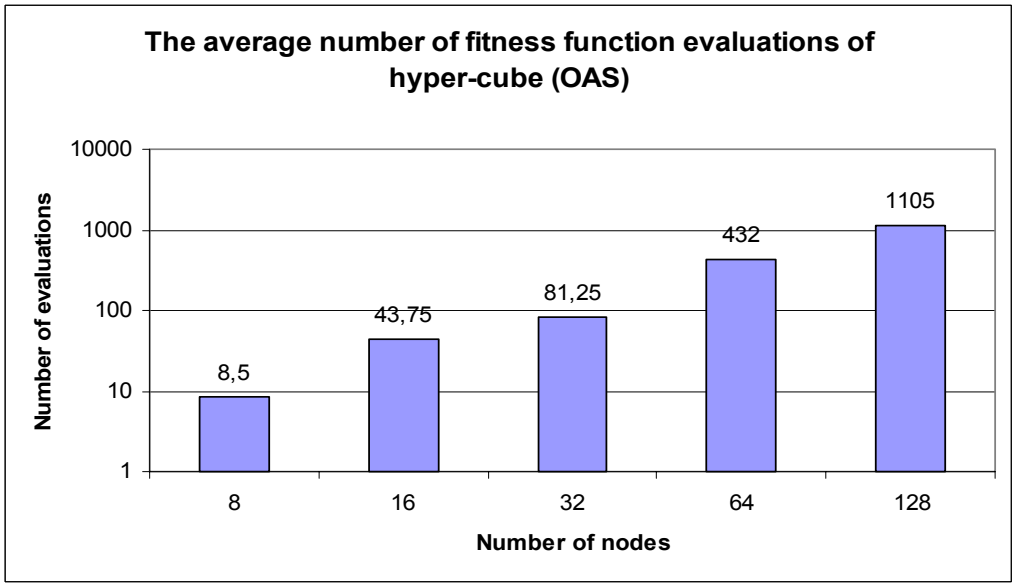


Figure 50: Time complexity of OAS.

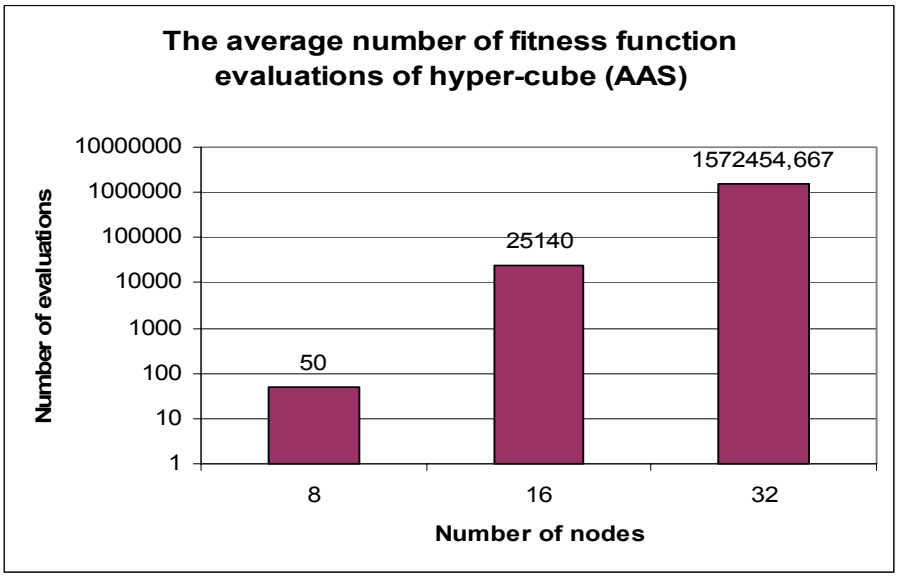


Figure 51: Time complexity of AAS.

As it is evident from Fig. 51, the most complex collective communication for scheduling is AAS. We obtained the optimal solution for three instances of hyper-cube, but it doesn't mean that our algorithm isn't able to solve more complex topologies. For these more complex collective communications, we are able to obtain a superior sub-optimal solution. However, we interested only in optimal schedules.

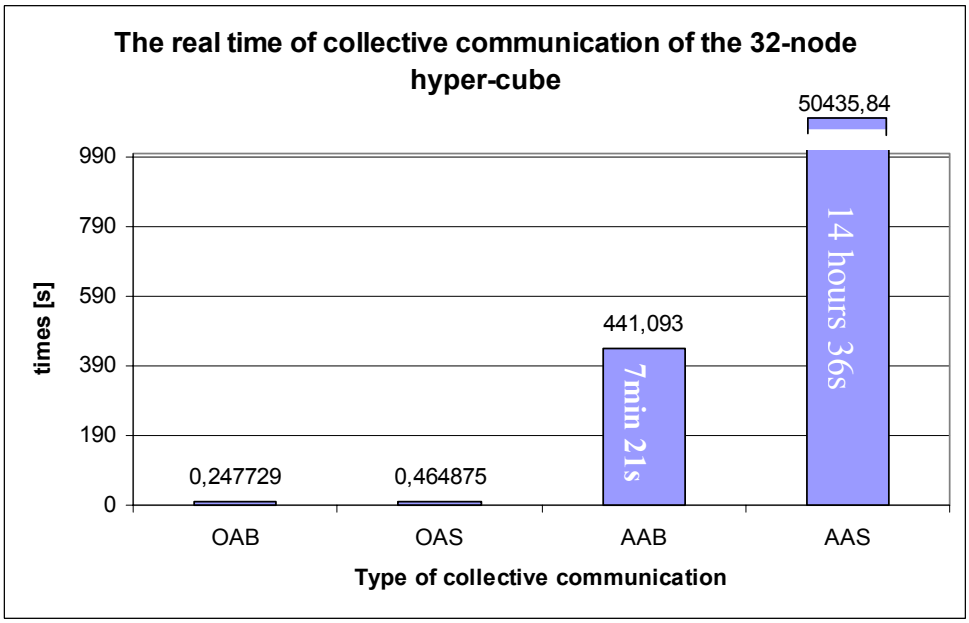


Figure 52: The real time complexity of four communication patterns.

All experiments were conducted using a supercomputer IBM BLADE system [54]. To illustrate how the execution times of individual communications differ, measurements were taken of the number of fitness function evaluations and also in seconds elapsed, see Fig. 52.

All experiments were made in all-port topologies with slim nodes, but for generalization of our algorithm, we must consider also 1-port topologies with fat nodes. Since to change a slim node to a fat node is very simply way how to increase the number of processors in an interconnection network, it is, therefore, utilized often. Experiments were performed on AAB communication pattern, because this pattern is the most complex for implementation, as described earlier in chapter 7.9. We performed these tests on the direct and the

indirect networks. The illustrative examples of one indirect and one direct network are in Fig. 53 - coated Mesh (CM) [4] and 2D-Mesh (M). Only 4 x 4 meshes are presented for simplicity

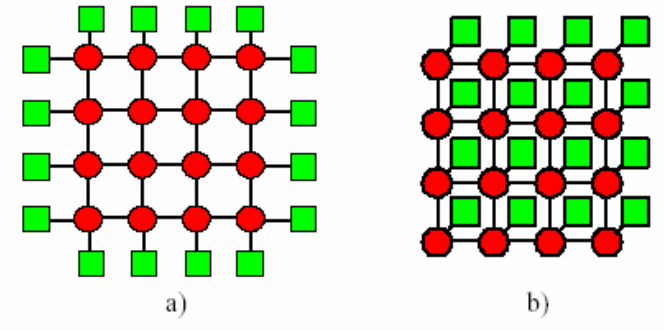


Figure 53: a) 4 x 4 CM, b) 4 x 4 2D-M.

The results of scheduling all-to-all broadcast communications are shown in Table 16. Optimum algorithms (lower bounds) have been obtained for all 20 runs. We considered that the fat 3D-Hyper-cube and the Fat Octagon (Fig. 45) had two processors at each node in our experiments.

Network topology	Lower bounds	AAB
M 4x4, 1-port	15	15 (100%)
M 4x4, all-port	8	8 (100%)
CM 4x4, 1-port	15	15 (100%)
Hyper-cube, 1-port	15	15 (100%)
Hyper-cube, all-port	5	5 (100%)
Octagon, 1-port	15	15 (100%)
Octagon, all-port	5	5 (100%)

Table 16: Results of AAB optimization.

The last experiments were devoted to the collective communication many-to-many broadcast. We solved all three possible situations, where the sets of senders and receivers are disjoint, overlapping and corresponding. As Table 17 illustrates, we again achieved optimal schedules. The previously unsolved problem of, “the overlapping sets of senders and receivers” was resolved during our investigation.

M-to-N	Lower bounds	MNB
8 to the same 8	7	7
8 to other 8	8	8
8 to all 16	8	8
all 16 to all 16	15	15

Table 17: M-to-N communication on the Fat Octagon topology (P = 16; 2 processors at a node).

Finally, we would claim that our algorithm is generally usable and solves arbitrary collective communication, including one-port and all-port topologies. Only many-to-many scatter communication have yet to be investigated, however all principles and techniques were developed and it only remains to modify the algorithm’s code to include this communication pattern.

Chapter 8

Conclusion and Future Research Directions

In this work we have focused on the design of optimal collective communication schedules for arbitrary topologies with SF switching technique. Optimal schedule achieves the minimal number of communication steps, which can, in an ideal case, be equal to the lower bound for any given topology. Using this schedule we can speed-up many parallel algorithms that use collective communication as a part of their algorithm. The finding of the optimum schedule is a very difficult combinatorial problem, therefore, we decided to use an evolutionary algorithm. We created the hybrid evolutionary algorithm HGSA, which is based on aggregation of parallel simulated annealing and standard genetic operations. The presented algorithm is able to find a schedule of a collective communication pattern for arbitrary network topologies and a related number of communication steps. The ability of our proposed algorithm to find good or even optimal solutions was proven by the use a hyper-cube benchmark. It can schedule collective communications for various networks with unknown optimal (minimal) number of steps and is useful especially for irregular topologies, where the analytical approach cannot be applied.

Of course, the fact that the lower bound may not always be reached by presented algorithms is to be expected because this may not be attainable in principle by any algorithm.

Sometimes lower bounds can be obtained in schedules with non-minimum routing. However, only minimum routing has been considered in this work because inclusion of the non-minimum routing would lead to an enormous increase of possible paths from sources to destinations and to prohibitive computer memory capacity and time requirements.
The probability rate of achieving an optimal solution is increased, if a prediction utility is used.
The proposed algorithm was tested successfully using networks with both slim node and fat node topologies.
The technology of fat interconnection networks has several advantages over traditional networks:
- makes some small networks more scalable, even though the interconnection graph of a network is not scalable at all (Moore, Twisted ladder) or only partially scalable (Octagon, AMP);

- in many cases provides cheaper network implementation in terms of hardware cost and is often more suitable for networking systems on chip;

- performance in one-to-all collective communications OAB and OAS is comparable to generic base networks, at times, even better;

- the performance in all-to-all collective communications AAB and AAS is inferior to that of base networks, but it can be controlled by multiplicity of links and by overlapping local and global communications.

We tested our algorithm by MNB collective communication with identical, overlapping and disjoint sets of senders and receivers. We derived the theoretical lower bound for this type of communication and our algorithm was able to design an optimal schedule of MNB with overlapping sets of senders and receivers, the first in history. Application-oriented many-to-many collective communications are of increasing importance on multiprocessor SoCs. One example is when one group of processors finishes a task and a different size group continues and needs the intermediate results from the first group.

Importance and novelty of above goals should be emphasized. Algorithms, which would be able to find all types of collective communication on any regular or irregular topology, were not published so far in spite of a growing importance especially for multiprocessors on chips.
Finally, we are able to state that our algorithm is the first, which is able to solve arbitrary type of collective communication for variety networks topologies.

8.1 Future Research Directions

The algorithm, described in this research paper, is the most generally useful of all proposed algorithms to date. However, it is necessary to state that something is missing before for total generalization can be claimed. Our algorithm we tested with one-port and all-port model, but except these models there can exist also *k*-port model, when the processor can communicate with the router via *k* internal channels simultaneously. In fact, both of the models, one-port and all-port, are special cases of the "*k*-port" model. Although our algorithm was not tested using this model, the source code is written in such a manner that we can achieve the required ability by means of a simple modification of the algorithm's code.

Another factor, not included in this research, was the communication pattern many-to-many scatter. Modification of our algorithm isn't complex for this communication pattern, because all methodologies and principles were developed and examined. To achieve this ability, we remove some constraint rules from code of communication pattern MNB (e.g. creation of broadcast tree).

In our work we assume only the same capacity of channels in whole interconnection network. However, in practise, we can meet with cases, when some channels have different capacities. This extension will slightly more complicated integrate into our algorithm's code.

The last improvement is related to length of messages. In our conception of problems we assumed that all messages have the same length for simplicity. However, real applications can work with different length of individual messages. The solving of this problem requires a deeper intervention into the proposed methodology.

Bibliography

[1] Flynn, M. J.: *Very high-speed computing systems*. Proc. IEEE, Vol. 12, pp. 1901–1909.

[2] Staroba J.: *Parallel Performance Modeling, Prediction and Tuning*, PhD. Thesis, Faculty of Information Technology, Brno University of Technology, Brno, Czech Rep., 2004.

[3] Anderson, T. E., Culler, D. E., Patterson, D.: *A Case for NOW (Networks of Workstations)*. IEEE Micro, Vol. 15, No. 1 (Feb.), pp. 54–64.

[4] Jantsch, A., Tenhunen, H., *Networks on Chip*, Kluwer Academic Publ., Boston, 2003.

[5] A. Ivanov, G. De Micheli, "Guest Editors' Introduction: *The Network-on-Chip Paradigm in Practice and Research*", IEEE Design & Test of Computers, IEEE Los Alamitos CA, Sept.-Oct. 2005, pp. 399-403.

[6] Singh A.: *Load-Balanced Routing in Interconnection Networks*, PhD. Thesis, The Department of Electrical Engineering, Stanford University, USA, 2005.

[7] Dally W.J., Towles B.: *Principles and Practices of Interconnection Networks*, Morgan Kaumann publishers, SF, USA, ISBN 0-23-200751-4, 2003.

[8] Tvrdík P.: *Parallel algorithms and computing*, CVUT, skripta (FEL), 2003.

[9] Charles L.: *The Cosmic Cube*, Communication of the ACM, 1985.

[10] Whitby-Srevebs C.: *The transputer*, In Proc. of the international Symposium on Computer Architecture (ISCA), 1985.

Bibliography

[11] Chien A. A.: *A cost and speed model for k-ary n-cube wormhole routers*, Proceedings of Hot Interconnects'93, 1993.

[12] Duato, J., Yalamanchili, S., Ni, L.: *Interconnection Networks*, Morgan Kaufmann publishers, SF, USA, ISBN 1-55860-852-4, 2003.

[13] Ni L. M., McKinley P. K.: *A survey of wormhole routing techniques in direct networks*, IEEE Computer, 1993.

[14] Khonsari A., Sarbazi-Azad H., Ould-Khaoua M.: *Analysis of true fully adaptive routing with software-based deadlock recovery*, Journal of Systems and Software, Computer Systems, vol. 71, 2004.

[15] Svethardware, Intel Tera-scale, URL: *http://www.svethardware.cz/art_doc216 DF91988A99F97C1257215003E2DDD.html*, February 2007.

[16] Intel, Intel Tere-scale, URL: *http://www.intel.com/technology/techresearch/teracale/index.htm*, February 2007.

[17] Walter A., Kuhm M, URL: *http://dontcry.cs.tu-berlin.de/cinsim/docbook/html/handbook.html*, Berlin, 2005.

[18] Wikipedia, The Free Encyclopedia, URL: http://en.wikipedia.org/wiki/Anycast, February 2007.

[19] Graveno L. et al.: *Adaptive deadlock- and livelock-free routing with all minimal paths in torus network*, IEEE Transaction on Parallel and Distributed Systems, vol. 5,no.12, 1233-1251, 1994.

[20] Tanenbaum A. S.: *Computer Networks*, 2nd ed., Prentice Hall, Englewood Cliffs, NJ, 1988.

[21] Hypercube topology, URL: *http://www.sgi.com/products/remarketed/origin/pdf/hypercube.pdf*, January 2006.

[22] Pinkston T. M., Duato J.: *Interconnection Networks*, Computer Architecture, vol. 4, University of Southern California, 2006.

[23] Valiant L. G., Brebner G. J.: *Universal schemes for parallel communication*, In Proc. of ACM Symposium of the Theory of Computing, Milwaukee, Minn., 1981.

Bibliography

[24] Towles B, Dally W. J.: *Worst-case traffic for oblivious routing functions*, In Proc. of the Symposium on Parallel Algorithms and Architectures (SPAA), Winnipeg, Manitoba, Canada, 2002.

[25] Nesson T., Johnson L. S., *ROMM routing on mesh and torus networks*, In Proc. of the Symposium on Parallel Algorithms and Architectures (SPAA), Santa Barbara, CA, 1995.

[26] Chien A.A., Kim J. H.: *Planar-adaptive routing: low-cost adaptive networks for multiprocessors*, In Proc. of the International Symposium on Computer Architecture (ISCA), 1992.

[27] Chiu G.: *The odd-even turn model for adaptive routing*, IEEE Tran. on Parallel and Distributed Systems, 2000.

[28] Ye T. T., Benini L., De Micheli G.: *Packetization and routing analysis of on-chip multiprocessor networks*, Journal of Systems Architecture, 2004.

[29] Glass C. J., Ni M. L.: T*he turn model for adaptive routing*, In 25 years ISCA: Retrospective and Reprint, 1998.

[30] Leiserson Ch. E., Abuhamdeh Z. S., Douglas D. C., Feynman C. R., Ganmukhi M. N., Hill J. V., Hillis W. D., Kuszmaul B. C., St Pierre M. A., Wells D. S., Wong-Chan M. C., Yang S., Zak R.: *The network architecture of the Connection Machine CM-5*, Journal of Parallel and Distributed Computing, 1996.

[31] Sarbazi-Azad H., Ould-Khaoua M.: *Modelling and evaluation of adaptive routing in high-performance n-D tori networks*, Simulation Modeling Practice and Theory, Distributed Systems Simulation, vol. 14, 2006.

[32] Schwiebert L., Jayasimha D. N.: *Optimally fully adaptive minimal wormhole routing for meshes*, Journal of Parallel and Distributed Computing, 1995.

[33] Qiao W., Ni L. M.: *Adaptive Routing in Irregular Networks Using Cut-Through Switches*, Proc. Int'l Conf. on Parallel Processing, 1996.

[34] Schroeder M. D.: *Autonet: a High-Speed, Self-Configuring Local Area network Using Point-to-Point Links*, SRC Research Report 59, DEC, 1990.

[35] Silla F.: *Efficient adaptive routing in networks of workstation with irregular topology*, Proc. of the Workshop on Communications and Architectural Support for Network-Based Parallel Computing, 1997.

[36] Silla F., Duato J.: *Improving the efficiency of adaptive routing in networks with irregular topology*, Proc. of the 1997 Conference on High Performance Computing, 1997.

Bibliography

[37] Keltcher C.N. et al.: *The AMD Opteron Processor for Multiprocessor Servers*. IEEE Micro, March/April 2003.

[38] Fogel D. B.: *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, 1995.

[39] Goldberg D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Complex Systems, 1989.

[40] E. H. L. Aarts and J. H. M. Korst: *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*, John Wiley & Sons, Chichester, 1989.

[41] D. E. Goldberg: *Simple genetic algorithms and the minimal deceptive problem*, in: L. Davis, ed., Genetic Algorithms and Simulated Annealing (Pitman, London, 1987).

[42] Metropolis N., Rosenbluth A. W., Rosenbluth M. N., Teller A. H., Teller E.: *Equation of state calculations by fast computing machines*, Journal of Chemical Physics 21(6), 1953.

[43] Ackley D. H., Hinton G. E., Sejnowski T. J.: *A learning algorithm for Boltzmann machines*, Cognitive Science 9, 1985.

[44] Goldberg D. E.: *A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing*, Complex Systems 4, 1990.

[45] Romeo F., Sangiovanni-Vincentelli A.: *A theoretical framework for simulated annealing*, Algorithmica, 1991.

[46] Sait M. S., Yoissef H.: *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*, Wiley-IEEE Computer Society Press, ISBN: 978-0-7695-0100-0, 2000.

[47] Manual MPI. Document reasonable on URL: *http://www-unix.mcs.anl.gov/mpi*, April 2007.

[48] MP– TESTDATA, URL:
http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html, April 2007.

[49] Romeo.F., Sangiovanni-Vincentelli A.: *A theoretical framework for simulated annealing*, Algorithmica, 1991.

[50] Mahfoud, S. W., Goldberg, D. E.: *A genetic algorithm for parallel simulated annealing*, Parallel Problem Solving from Nature 2, pp. 301–310, 1992.

Bibliography

[51] Mori, N., Yoshida, J. and Kita, H.: *Suggestion of thermodynamical selection rule in genetic algorithm*, Transaction of Institute of Systems, Control and Information Engineers, Vol. 9, No. 2, 1996.

[52] Krajíc, M.: *Algorithm of parallel hybrid genetic simulated annealing to solve of traveling salesman problem,* ČVUT FEL, Prague, 2002.

[53] Chalmers, A.-Tidmus, J.: *Practical Parallel Processing,* International Thomson Computer Press, 1996.

[54] IBM BLADE system, URL: http*://www-03.ibm.com/systems/bladecenter/*, April 2007.

[55] K. A. De Jong: *An analysis of the behavior of a class of genetic adaptive systems*, Dissertation Abstracts International, Ph.D. Thesis, University of Michigan, 1975

[56] J. H. Holland: *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, 1992

[57] Wiley J. & Sons: The Handbook of Information Security, URL: *http://www.cse.scu.edu/~hpdommel/publications/hpd.wiley06.pdf*, March 2005.

[58] Marsan Ajmone M., Bianco A., Giacoone P., Leonardi E., Neri F.: *Router Architecture Exploiting Input-Queued*, Cell-Based Switching Fabrics, Politecnico di Torino, Italy, August 2000.

[59] Rexford J, Hall J, Shin K. G.: *A Router Architecture for Real-Time Communication in Multicomputer Networks*, IEEE Transactions on Computers, vol 47, no. 10, October 1998.

[60] Schmid S.: *A Component-based Active Router Architecture*, PhD. Thesis, Computing Department, Lancaster University, United Kingdom, November 2002.

[61] Rexford J., Hall J., Shin K. G.: *A Router Architecture for Real-Time Point-to-Point Networks*, Department of Electrical Engineering and Computing Science, University of Michigan, USA, 1996.

[62] Bang Y., Choo H.: *On Bandwidth Adjusted Multicast Communications in Pipeline Router Architecture*, Springer, 2005.

[63] Duato J.: *A Necessary and Sufficient Condition for Deadlock-Free Routing in Cit-Through and Store-and-Forward Networks*, IEEE Translations on parallel and distributed systems, vol. 7, no. 8, August 1996.

[64] V. Puente V., Gregorio J.A., Beivide R., Vallejo F., Ibańez A.: *A New Routing Mechanism for Networks with Irregular Topology*, Proc. of the ACM/IEE SC2001 Conference, 2001.

[65] Chi H.C., Tang C. T.: *A Deadlock-Free Routing Scheme for Interconnection Networks with Irregular Topologies*, Proc. of the 1997 International Conference on Parallel and Distributed Systems (ICPADS '97), 1997.

[66] Sait, S., M. Youssef H.: *Iterative Computer Algorithms with Applications in Engineering*, IEEE Computer Society, Los Alamos, California, 1999.

[67] Pao, D. C. W., Lam, S. P., Fong A. S.: *Parallel simulated annealing using transaction processing*, IEEE Proceedings, Hong Kong, 1999.

[68] Miky, M., Hiroyasu, T., Wako, J., Yoshida, T.*: Adaptive Temperature Schedule Determined by Genetic Algorithm for Parallel Simulated Annealing*, Doshisha University, Kyoto, 2003.

Author publications

[I] Dvořák Václav, Jaroš Jiří, Ohlídal Miloš: *Optimum Topology-Aware Scheduling of Many-to-Many Collective Communications*, In: Proceedings of The Sixth International Conference on Networking, New York, US, IEEE CS, 2007, s. 6, ISBN 0-7695-2805-8

[II] Jaroš Jiří, Ohlídal Miloš, Dvořák Václav: *An Evolutionary Approach to Collective Communication Scheduling*, In: Genetic and Evolutionary Computation Conference GECCO 2007, London, 2007

[III] Jaroš Jiří, Ohlídal Miloš, Dvořák Václav: *Complexity of Collective Communications on NoCs*, In: Proc. of 5th International Symposium on Parallel Computing in Electrical Engineering, Los Alamitos, CA 90720-1314, US, IEEE CS, 2006, s. 127-132, ISBN 0-7695-2554-7

Bibliography

[IV] Ohlídal Miloš, Jaroš Jiří, Dvořák Václav, Schwarz Josef: *Evolutionary Design of OAB and AAB Communication Schedules for Interconnection Networks*, In: Lecture Notes in Computer Science, roč. 2006, č. 3907, DE, s. 267-278, ISSN 0302-9743

[V] Ohlídal Miloš, Jaroš Jiří, Dvořák Václav*: Performance of Collective Communications on Interconnection Networks with Fat nodes and Edges*, In: Proceedings of the Fifth International Conference on Networking ICN 2006, Los Alamitos, US, IEEE CS, 2006, s. 619-624, ISBN 0-7695-2570-9

[VI] Ohlídal Miloš, Schwarz Josef: *Collective Communication AAB for Regular and Irregular Topology Based on Prediction of Conflicts*, In: Proc. of 2006 IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop, Praha, CZ, IEEE CS, 2006, s. 224-225, ISBN 1-4244-0184-4

[VII] Ohlídal Miloš: *Plánování skupinové komunikace All-to-All Broadcast pomocí predikce konfliktů v propojovacích sítích*, In: Zborník príspevkouv pracovného semináre Počítačové architektúry a diagnostika pre studenty doktorského štúdia, Bratislava, SK, UI SAV, 2006, s. 25-30, ISBN 80-969202-2-7

[VIII] Jaroš Jiří, Ohlídal Miloš, Dvořák Václav: *Evolutionary Design of Group Communication Schedules for Interconnection Networks*, In: Lecture Notes in Computer Science, 2005, č. 3733, DE, s. 472-481, ISSN 0302-9743

[IX] Ohlídal Miloš, Schwarz Josef: *Design of Group Communication for Regular and Irregular Networks*, In: Mendel 2005 11th Internacional Conference on Soft Computing, Brno, CZ, FSI VUT, 2005, s. 45-50, ISBN 80-214-2961-5

[X] Ohlídal Miloš: *Plánování skupinových komunikací v propojovacích sítích*, In: Sborník příspěvků ze semináře Počítačové Architektury a Diagnostika, Praha, CZ, FEL ČVUT, 2005, s. 129-134, ISBN 80-01-03298-1

[XI] Ohlídal Miloš, Schwarz Josef: *HYBRID PARALLEL SIMULATED ANNEALING USING GENETIC OPERATIONS*, In: Mendel 2004 10th Internacional Conference on Soft Computing, Brno, CZ, FSI VUT, 2004, s. 89-94, ISBN 80-214-2676-4

[XII] Ohlídal Miloš, Schwarz Josef: *Parallel Simulated Annealing Applied to the Traveling Salesman Problem*, In: Proceedings of 38th International conference MOSIS'04, Rožnov pod Radhoštěm, CZ, MARQ, 2004, s. 155-162, ISBN 80-85988-98-4

[XIII] Ohlídal Miloš: *Hybrid parallel simulated annealing using genetic operations*, In: Zborník príspevkov ze seminara Počítačové Architektury a Diagnostika, Bratislava, SK, SAV, 2004, s. 48-53, ISBN 80-969202-0-0

Appendix A

Pseudo-code of Conflicts Prediction

```
Procedure Simulated_Annealing(T_min,T_max,k_max,α:input;
                              chromozome_opt:output);
  begin
    chromozome_ini:=randomly_generated_chromosome;
    T:=T_max;
    phase := 1;
    //detection of conflict channels – prediction
    Prediction(chromozome_ini, links);
    //all channels are searched in whole schedule
    SearchChannel(chromosome_ini, channels);
    for i:=1 to i = channels.size() do
    //search every paths with investigated channel and set com.
    //step to all channels in chosen paths
    //and simultaneously conflicting channels are searched
    OffsetSetting(chromosome_ini, i, conflict_channel);
    //evalution of individual
    price_st := links.size() + conflict_channel.size();
    While(T > T_min and price_st <> 0)do
      begin
        Metropolis_alg(k_max,T, chromozome_ini, phase, links,
                       conflict_channel, chromozome_out, phase,
                       price_st, links, conflict_channel);
        chromozome_ini:= chromozome_out;
        T:= α*T;
      end;
    chromozome_opt:= chromozome_out;
end;
```

```
Procedure Metropolis_alg(k_max,T, chromozome_ini, phase, links,
                         conflict_channel:input;
                         chromozome_out, phase, price_st, links,
                         conflict_channel:output);
  begin
    k:=0;
    chromozome:= chromozome_ini;
    while (k< k_max and price_st <> 0) do
      begin
//---------------------Prediction phase-----------------------
        if(phase = 1) then
          begin
            if(randnum(100) < pr_mut)
               //randomly selected path is changed
               Mutation(chromozome);
            else
              begin
                 //conflicting channel is randomly selected
                 inv_link := random(links.size());
                 //detection of paths, on which the
                 //investigation channel is occurred
                 Detection(inv_link, chromosome, index_paths);
                 //one path is selected
                 path := randomly(index_paths);
                 //selected path is changed
                 Mutation(chromosome, path);
              end;

            //substitution of the shorter path consists in
            //copying subpath of the longer path whose
            //intermediate node is simultaneously terminated
            //node of the shorter path and source node is the
            //same in the both paths
            Heuristic(chromosome);
            Prediction(chromosome, links);
            if(links.empty()) then
               begin
                 phase := 2;
                 SearchChannel(chromosome, channels);
                 for i:=1 to i = channels.size() do
                   OffsetSetting(chromosome, i,
                                 conflict_channel);
               end;
          end;
//------------------------------------------------------------

//----------- Phase of communication step setting -------------
        else
          begin
             //a conflicting channel is selected
             channel := random(conflict_channel.size());
```

```
            OffsetSetting(chromosome, channel,
                          conflict_channel);
        end;
//-------------------------------------------------------------
        //evalution of new individual
        price_nov := links.size() + conflict_channel.size();

//---------------- Communication master slave -----------------
        if(k mod comm = 0) then
          begin
            if(numID = SLAVE) then
              begin
                //its solution is sent to master
                Send(chromosome, master);
                //the solution is obtained from master
                Recv(parent, master);
              end;
            if(numID = MASTER) then
              begin
                //solutions are obtained from all slaves
                for i:=1 to i=coutProc do
                  Recv(chromosome_array, i);
                //roulette is performed
                Roulette(chromosome_array);
                //distribution of solutions according of
                //roulette selection
                for i:=1 to i=coutProc do
                  Send(chromosome_array, i);
              end;
            //it performs crossover, mutation and selection
            //of individual to next execution
            GAOperations(chromosome,parent);
          end;
//-------------------------------------------------------------
        if price_nov - price_st < 0 then
          begin
            chromozome:= chromozome´
            price_st = price_nov;
          end;
         else
                                  -(price_nov - price_str)/T
           if random()< e                                  then
            begin
              chromozome:= chromozome´
              price_st = price_nov;
            end;
      end;
end;
```

```
Procedure Prediction(chromosome:input; links:output);
  begin
    //all channels are searched in whole schedule
    SearchChannel(chromosome, channels);
    //search every paths with investigated channel
    for channel:=1 to channel=channels.size() do
      begin
        SearchPath(chromosome, channel, paths);
        //the number of paths using the channels is greater
        //than the value of comm. step of whole schedule
        if(paths.size() > comm_step)
          //conflict is detectable
          links.push_back(channel);
        else
          begin
            //detection of possible steps for channels on
            //investigate paths
            ok := StepDetection(chromosome, paths);
            //it is not possible to set two different value of
            //step to channel on two different paths
            if(ok = FALSE) then
              begin
                links.push_back(channel);
              end;
          end;
      end;
  end;
```

```
Procedure OffsetSetting(chromosome, channel:input;
                        conflict_channel:output);
  begin
    //search every paths with conflicting channel
    SearchPath(chromosome, channel, paths);
    //search remaining channels occur on paths with conflicting
    //channel
    SearchChannel(chromosome, channels, paths);
    for ch:=1 to ch=channels.size() do
      begin
        //search every paths with investigated channel
        SearchPath(chromosome, ch, ch_paths);
        //calculation of interval of possible communication
        //steps to channel on selected paths
        BoundCompute(ch, ch_paths);
        // an assignment of communication steps to channel on
        //selected paths
        ok := SetStep(chromosome, ch, ch_paths);
```

```
        //it is not possible to set two different value of
        //step to channel on two different paths
        if(ok = FALSE) then
          begin
            conflict_channel.push_back(ch);
          end;
      end;
   end;
```

```
Begin
  input, output : file;

  //read parameters from input_file
  ReadInput(input);
  //create set of all shortest paths between every pair of
  //source-destination nodes
  CreateSetOfShortestPaths(paths);
  //initialization of the chromosome
  Inicialization(chromozome_ini);
  if(numID = MASTER) then
    begin
      //initialization of output file to writing
      output := open(write);
    end;
  //evolutionary optimization technique - searching of conflict-
  //free schedule
  Simulated_Annealing(chromosome_ini,T_min,T_max,k_max,α,chromozome_opt);
  if(numID = MASTER) then
    begin
      //printing of achieved schedule
      PrintChromosome(chromozome_opt, output);
    end;
End.
```

Appendix B

Investigated Topologies

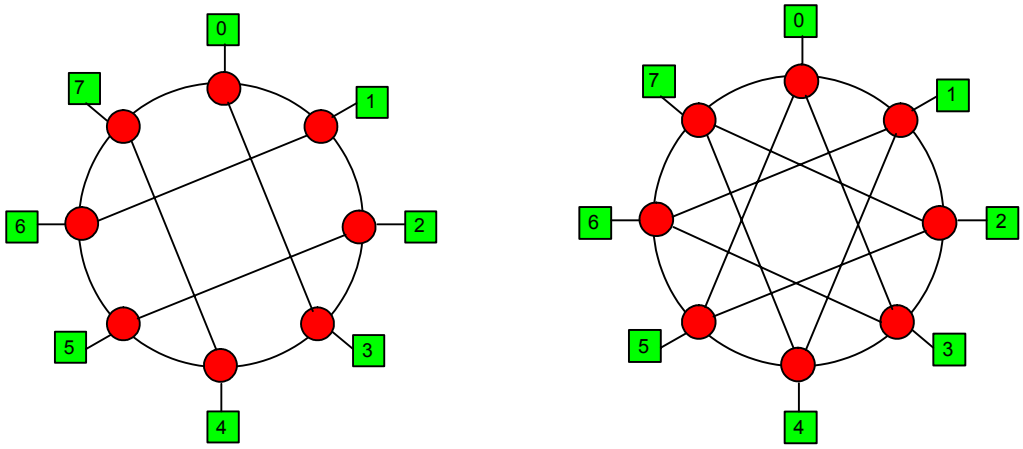
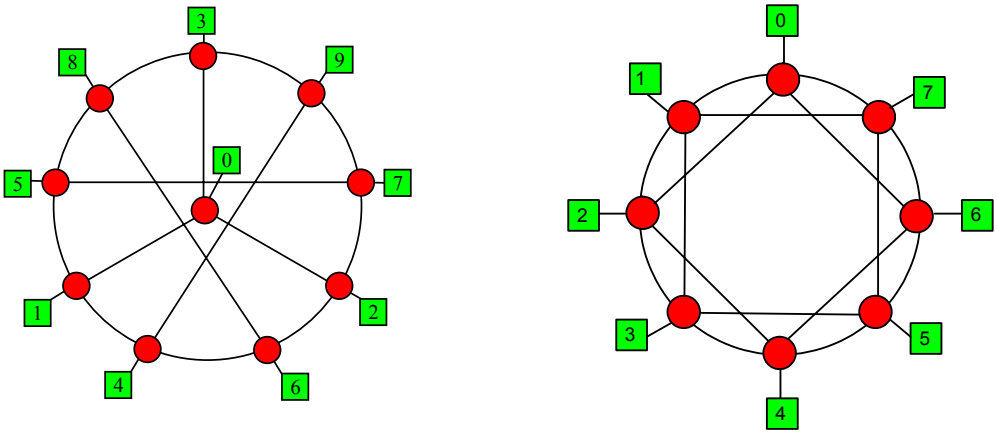
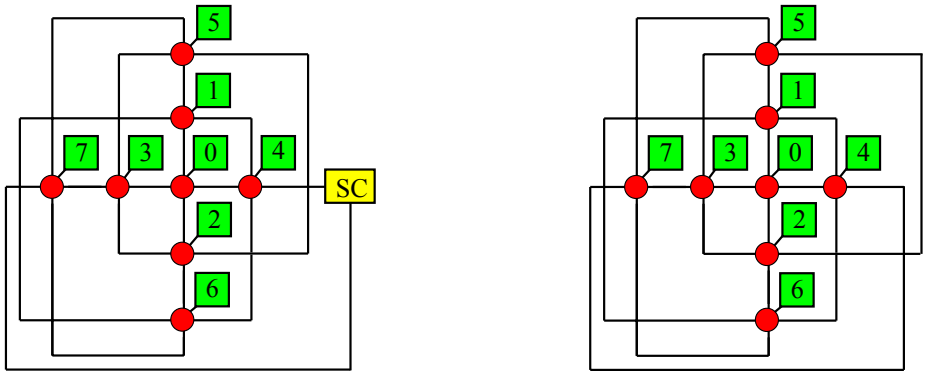


Figure 54: Interconnection networks: a) Hyper-cube and b) K-ring



c) d)

Figure 55: c) Moore graph and d) Midimew

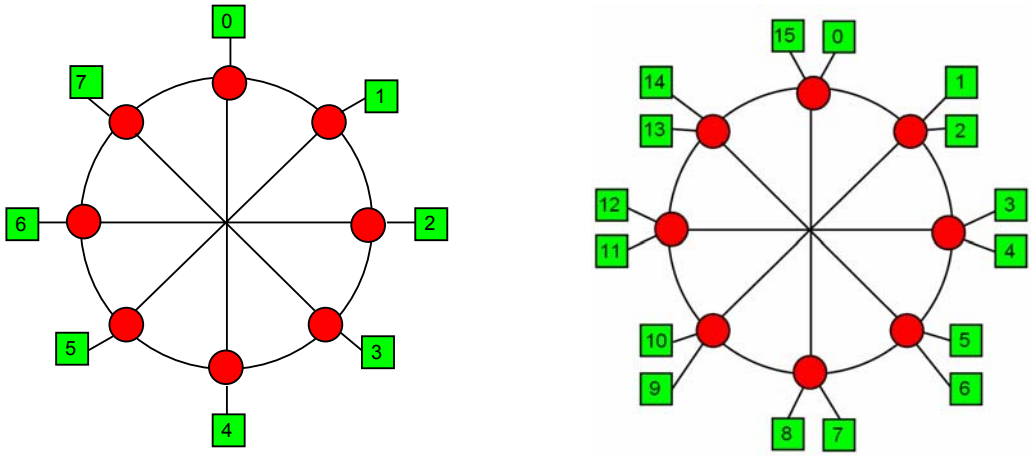


e) f)

Figure 56: e) AMP with SC and f) AMP without SC

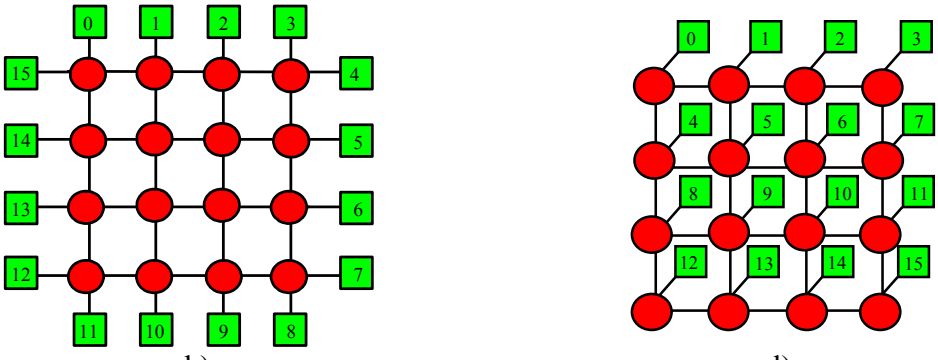
g) h)

Figure 57: g) Ladder and h) Twisted ladder



i) j)

Figure 58: i) Slim Octagon and j) Fat Octagon



k) l)

Figure 59: k) Coated Mesh and l) 2D-Mesh

Appendix C

Example of Optimal OAB Communication Schedule on Moore Graph Topology

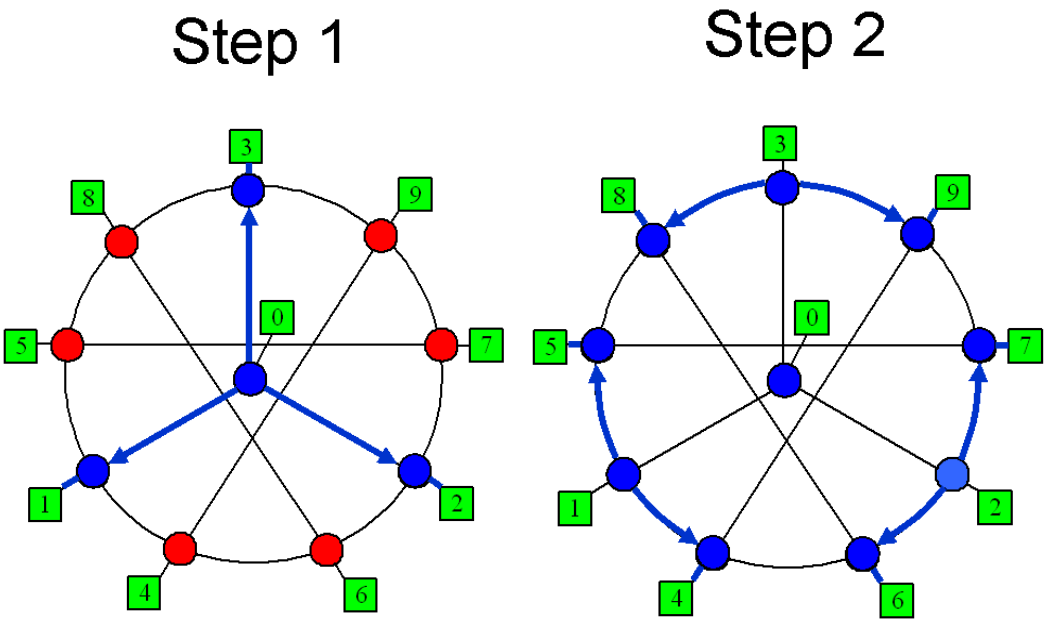
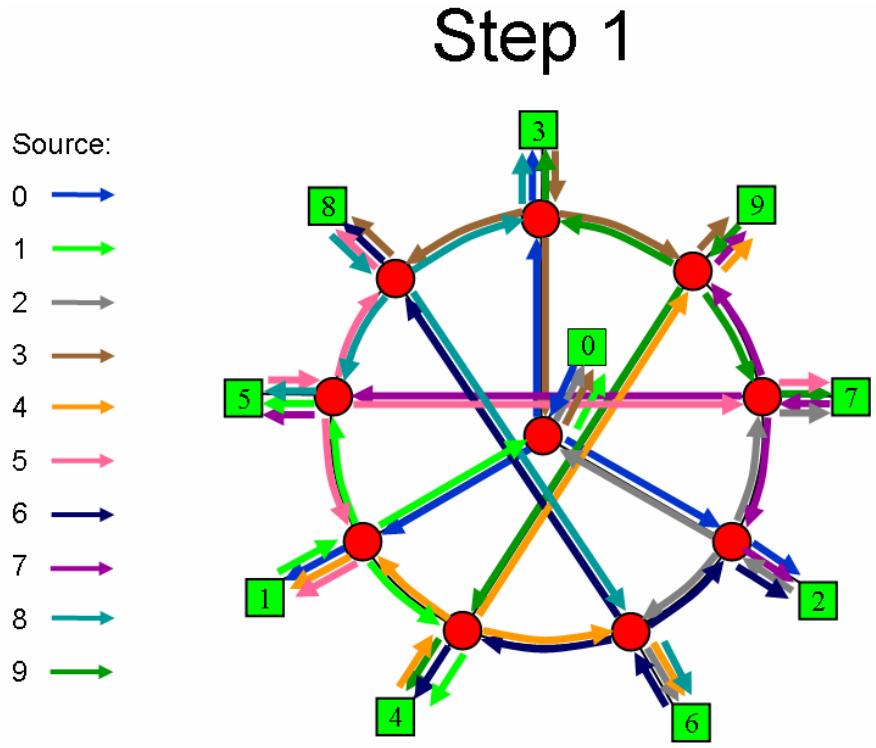


Figure 60: Model of OAB communication: store-and-forward switching, full duplex links, all-port non-combining model.

Appendix D

Example of Optimal AAB Communication Schedule on Moore Graph Topology



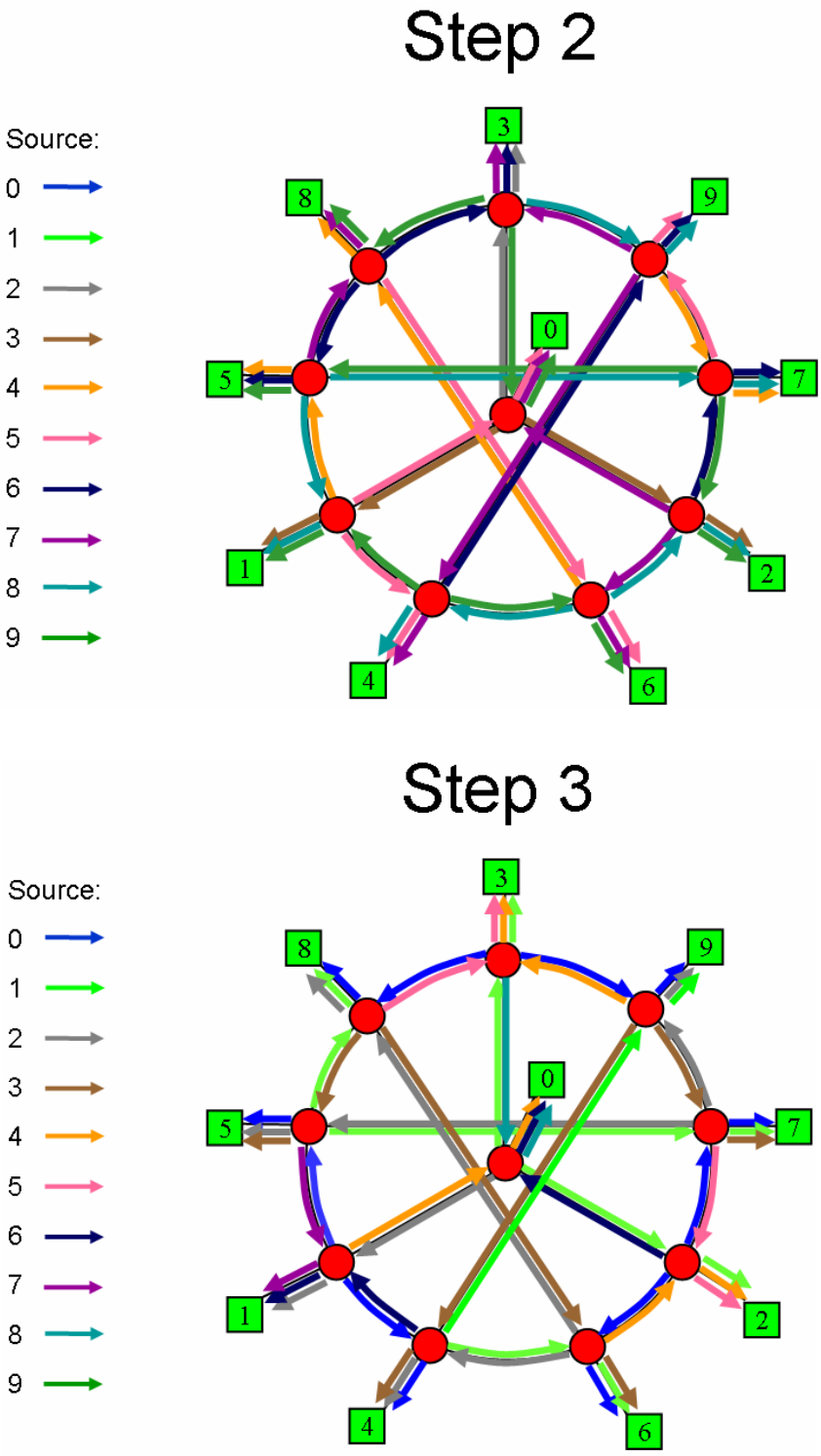
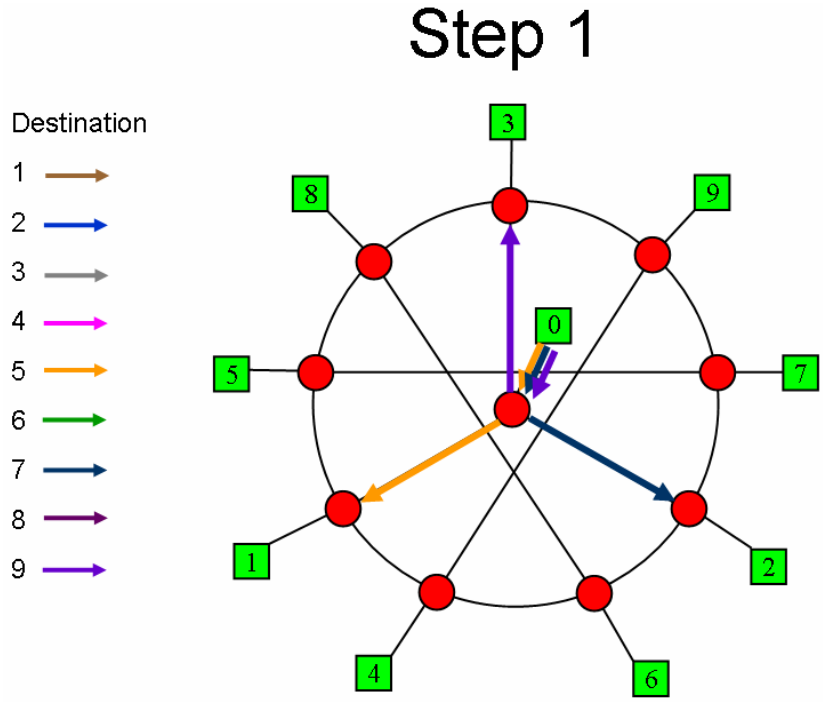
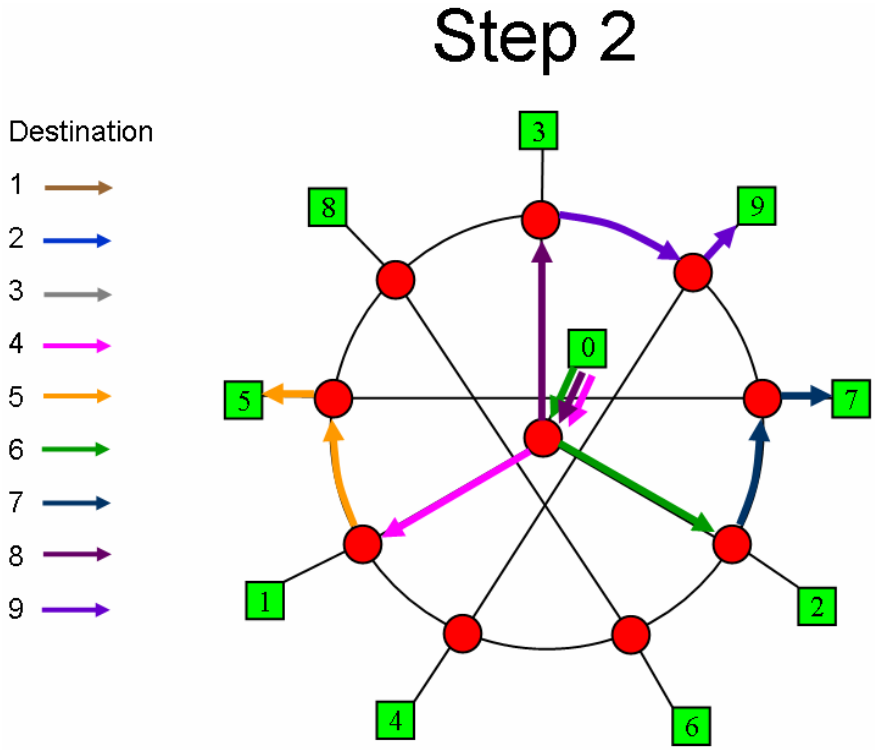


Figure 61: Model of AAB communication: store-and-forward switching, full duplex links, all-port non-combining model.

Appendix E

Example of Optimal OAS Communication Schedule on Moore Graph Topology





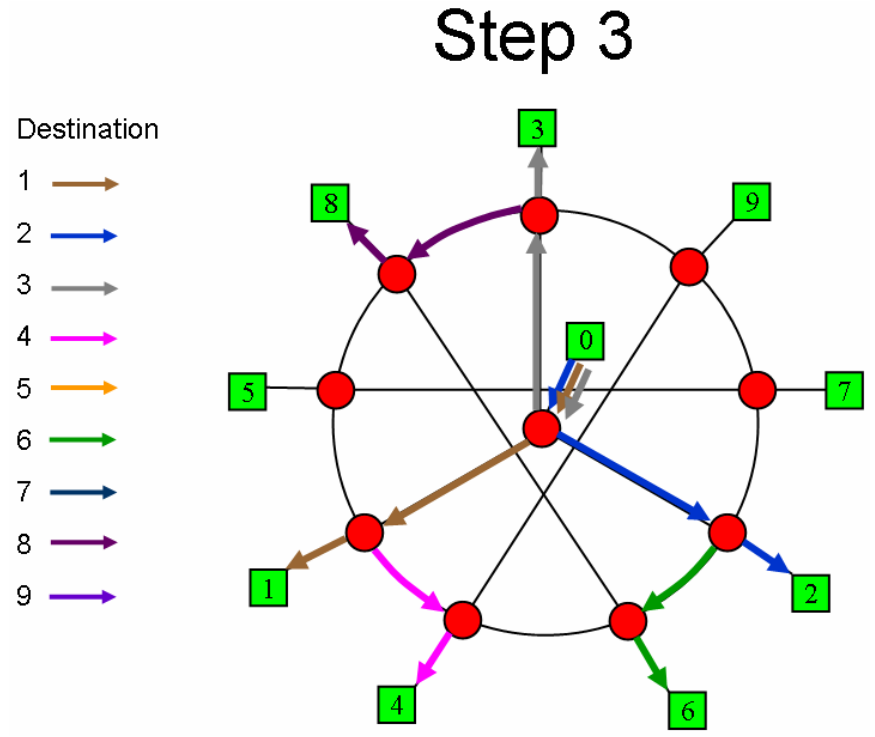
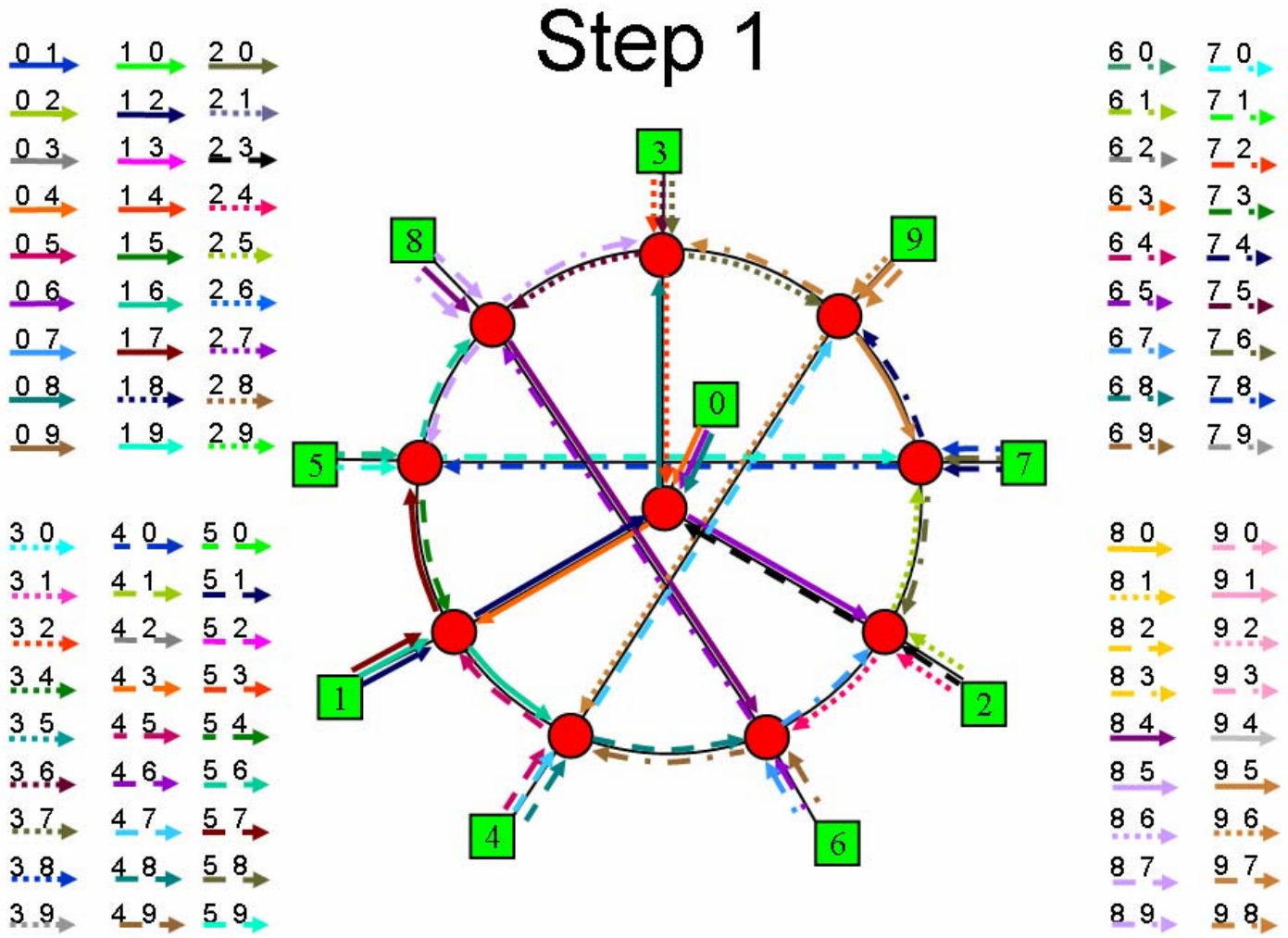
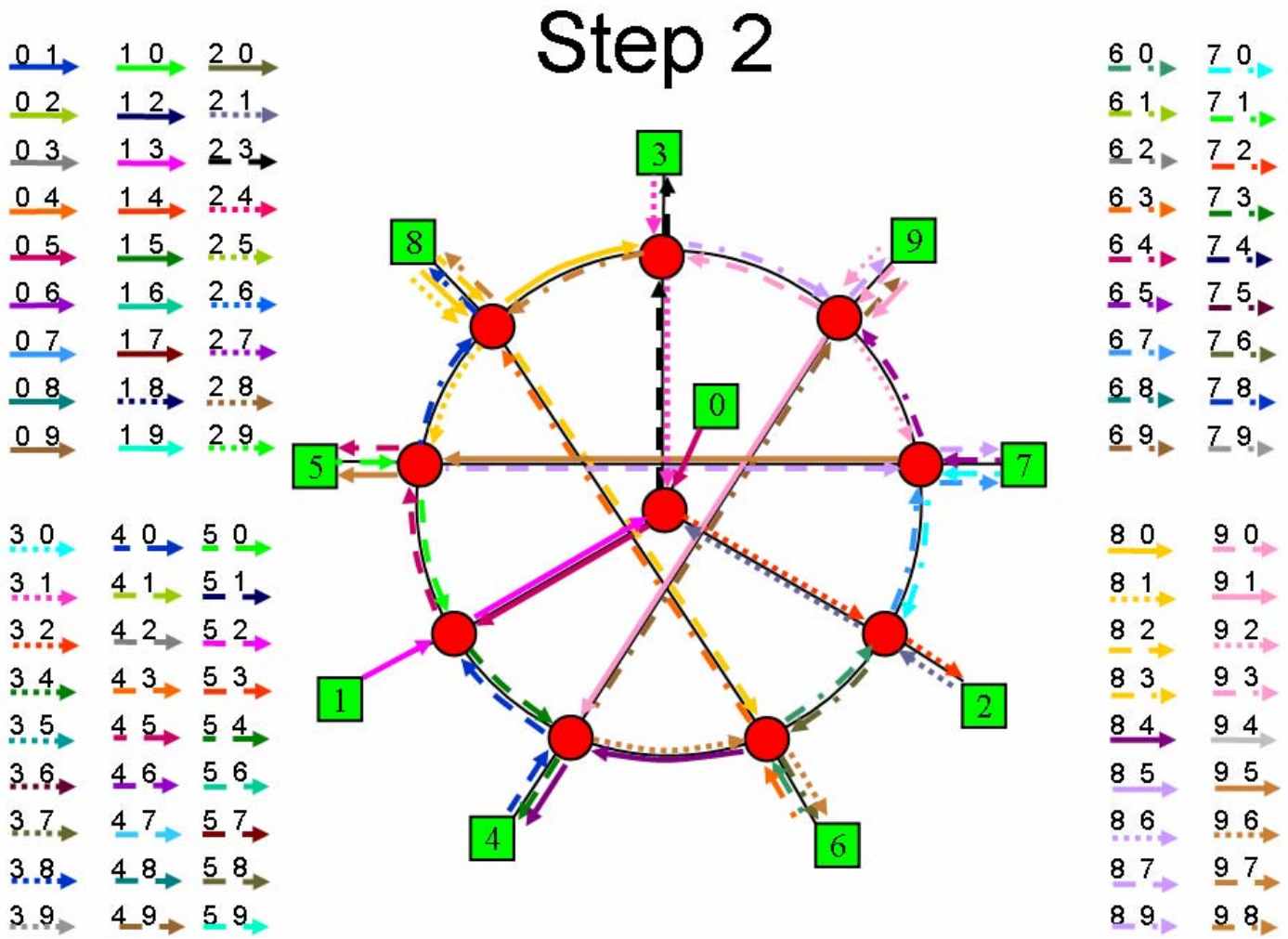


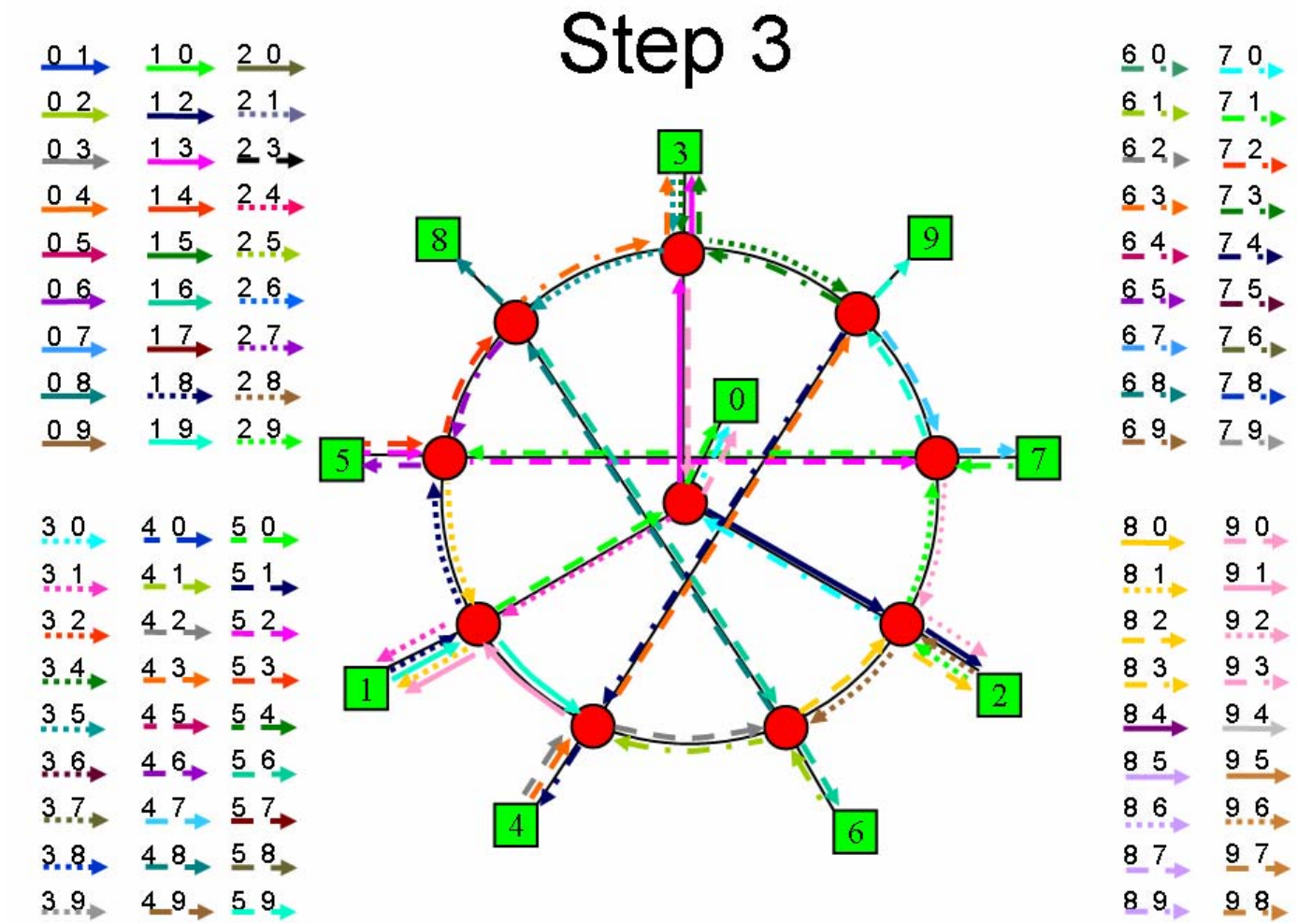
Figure 62: Model of OAS communication: store-and-forward switching, full duplex links, all-port non-combining model.

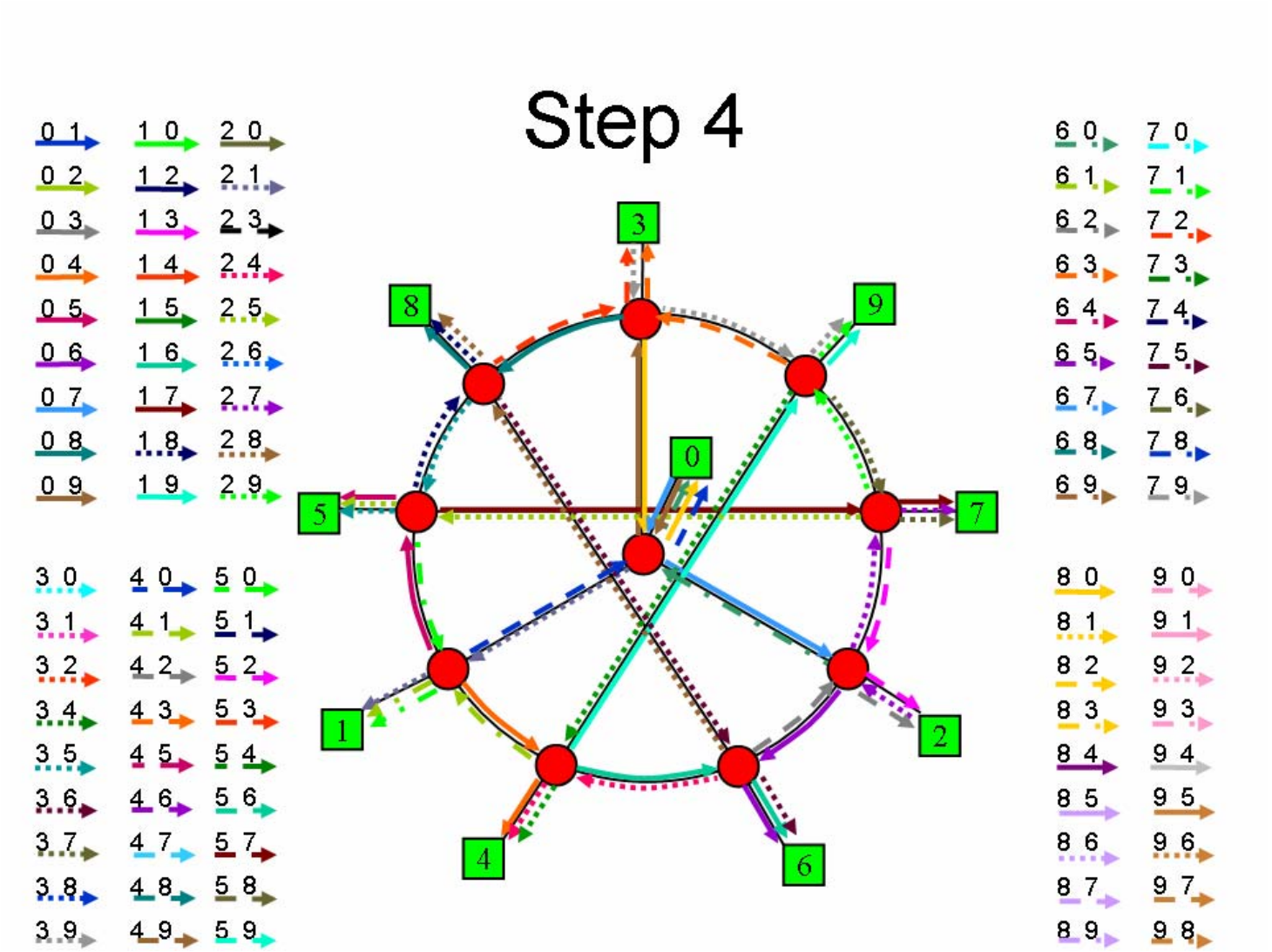
Appendix F

Example of Optimal AAS Communication Schedule on Moore Graph Topology


Step 1
0 1
1 0
2 0
0 2
1 2
2 1
0 3
1 3
2 3
0 4
1 4
2 4
0 5
1 5
2 5
0 6
1 6
2 6
0 7
1 7
2 7
0 8
1 8
2 8
0 9
1 9
2 9
3 0
4 0
5 0
3 1
4 1
5 1
3 2
4 2
5 2
3 4
4 3
5 3
3 5
4 5
5 4
3 6
4 6
5 6
3 7
4 7
5 7
3 8
4 8
5 8
3 9
4 9
5 9
6 0
7 0
6 1
7 1
6 2
7 2
6 3
7 3
6 4
7 4
6 5
7 5
6 7
7 6
6 8
7 8
6 9
7 9
8 0
9 0
8 1
9 1
8 2
9 2
8 3
9 3
8 4
9 4
8 5
9 5
8 6
9 6
8 7
9 7
8 9
9 8
0
1
2
3
4
5
6
7
8
9


Step 2
0 1
1 0
2 0
0 2
1 2
2 1
0 3
1 3
2 3
0 4
1 4
2 4
0 5
1 5
2 5
0 6
1 6
2 6
0 7
1 7
2 7
0 8
1 8
2 8
0 9
1 9
2 9
3 0
4 0
5 0
3 1
4 1
5 1
3 2
4 2
5 2
3 4
4 3
5 3
3 5
4 5
5 4
3 6
4 6
5 6
3 7
4 7
5 7
3 8
4 8
5 8
3 9
4 9
5 9
6 0
7 0
6 1
7 1
6 2
7 2
6 3
7 3
6 4
7 4
6 5
7 5
6 7
7 6
6 8
7 8
6 9
7 9
8 0
9 0
8 1
9 1
8 2
9 2
8 3
9 3
8 4
9 4
8 5
9 5
8 6
9 6
8 7
9 7
8 9
9 8
0
1
2
3
4
5
6
7
8
9


Step 3
0 1
1 0
2 0
0 2
1 2
2 1
0 3
1 3
2 3
0 4
1 4
2 4
0 5
1 5
2 5
0 6
1 6
2 6
0 7
1 7
2 7
0 8
1 8
2 8
0 9
1 9
2 9
3 0
4 0
5 0
3 1
4 1
5 1
3 2
4 2
5 2
3 4
4 3
5 3
3 5
4 5
5 4
3 6
4 6
5 6
3 7
4 7
5 7
3 8
4 8
5 8
3 9
4 9
5 9
6 0
7 0
6 1
7 1
6 2
7 2
6 3
7 3
6 4
7 4
6 5
7 5
6 7
7 6
6 8
7 8
6 9
7 9
8 0
9 0
8 1
9 1
8 2
9 2
8 3
9 3
8 4
9 4
8 5
9 5
8 6
9 6
8 7
9 7
8 9
9 8
0
1
2
3
4
5
6
7
8
9


Step 4
0 1
1 0
2 0
0 2
1 2
2 1
0 3
1 3
2 3
0 4
1 4
2 4
0 5
1 5
2 5
0 6
1 6
2 6
0 7
1 7
2 7
0 8
1 8
2 8
0 9
1 9
2 9
3 0
4 0
5 0
3 1
4 1
5 1
3 2
4 2
5 2
3 4
4 3
5 3
3 5
4 5
5 4
3 6
4 6
5 6
3 7
4 7
5 7
3 8
4 8
5 8
3 9
4 9
5 9
6 0
7 0
6 1
7 1
6 2
7 2
6 3
7 3
6 4
7 4
6 5
7 5
6 7
7 6
6 8
7 8
6 9
7 9
8 0
9 0
8 1
9 1
8 2
9 2
8 3
9 3
8 4
9 4
8 5
9 5
8 6
9 6
8 7
9 7
8 9
9 8
0
1
2
3
4
5
6
7
8
9

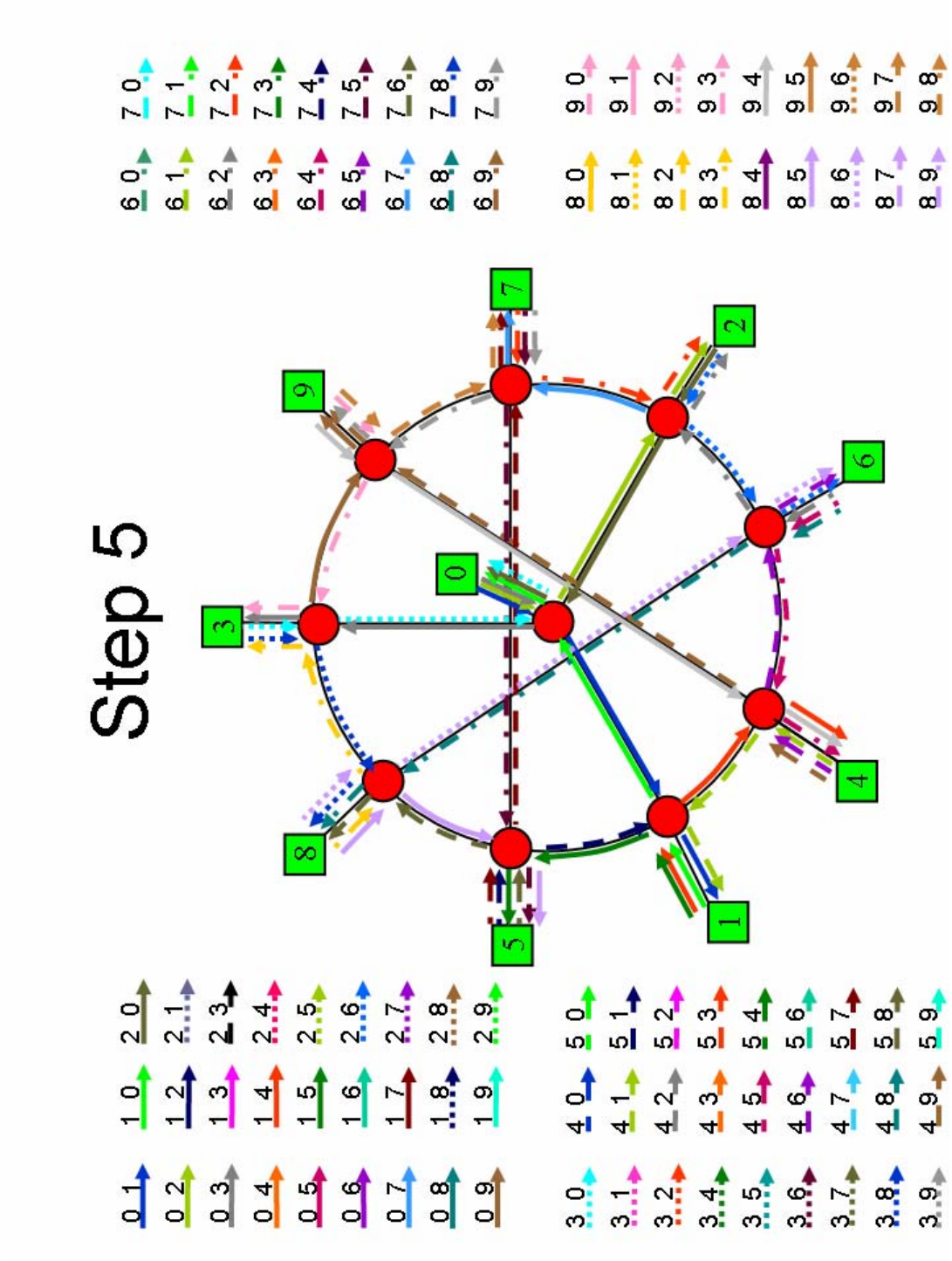
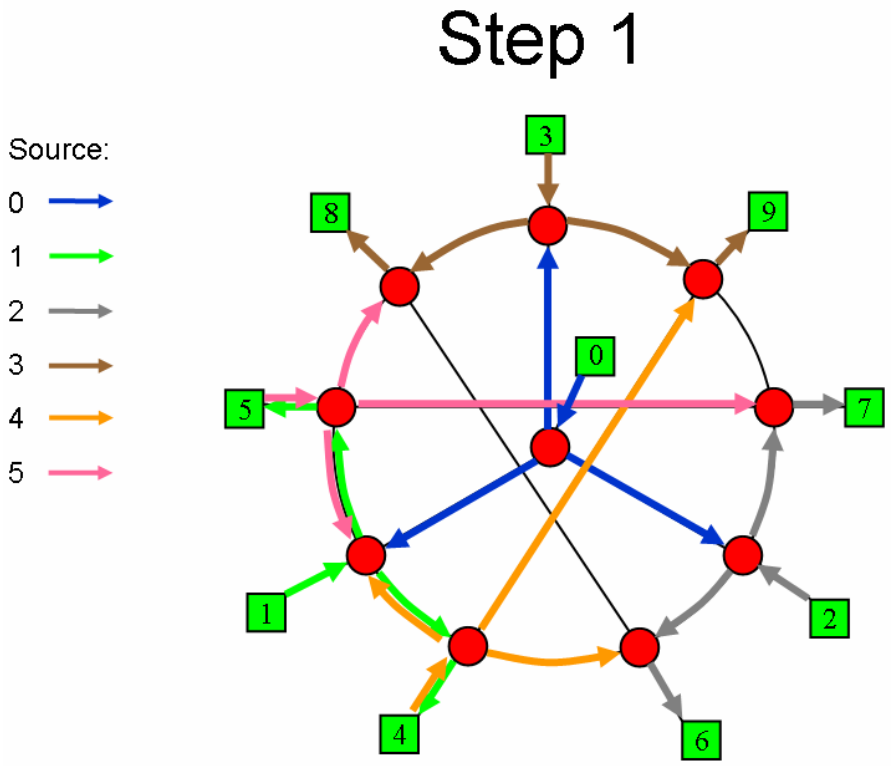
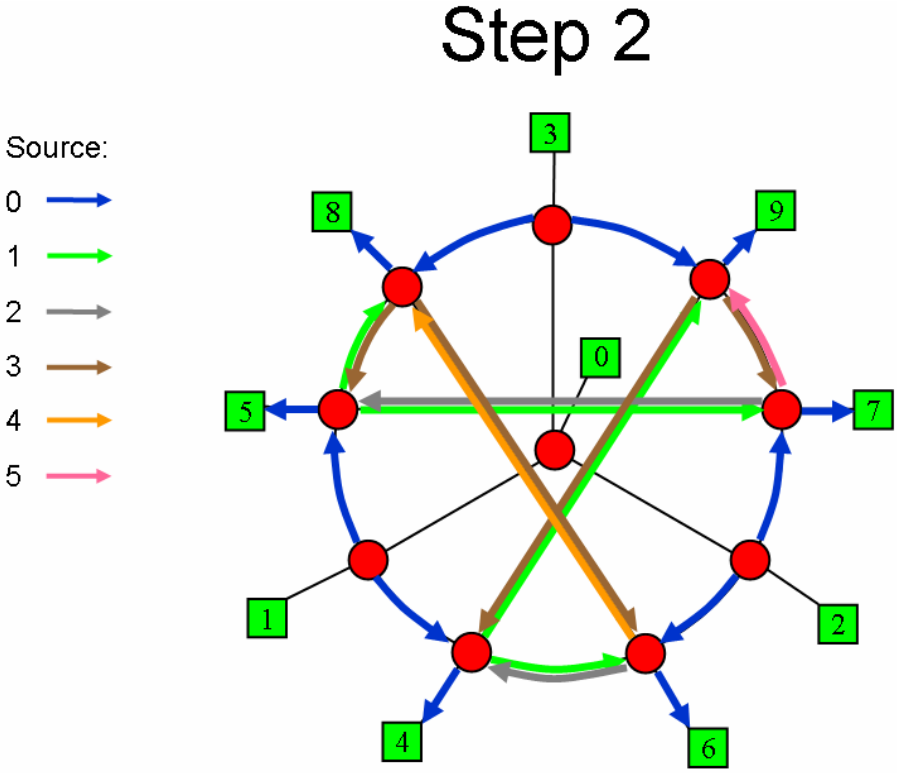


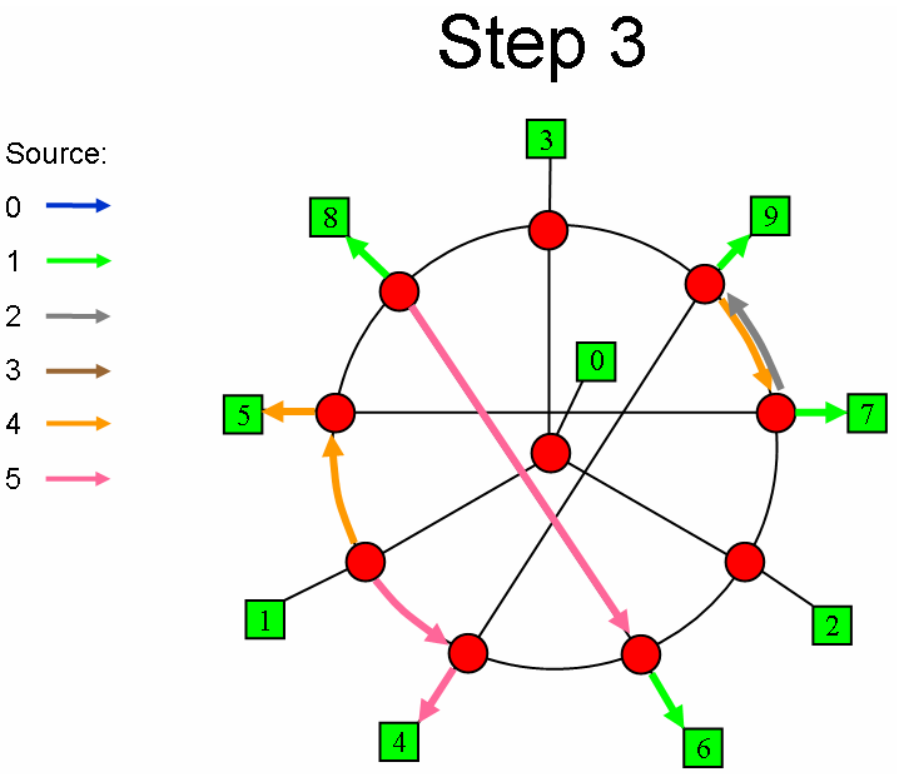
Figure 63: Model of AAS communication: store-and-forward switching, full duplex links, all-port non-combining model.

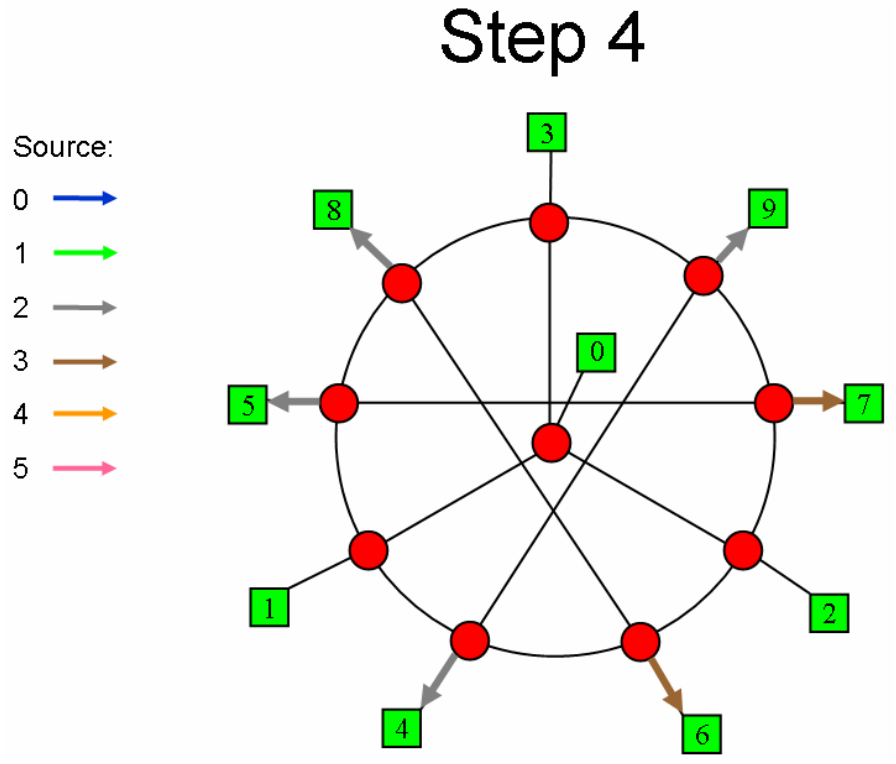
Appendix G

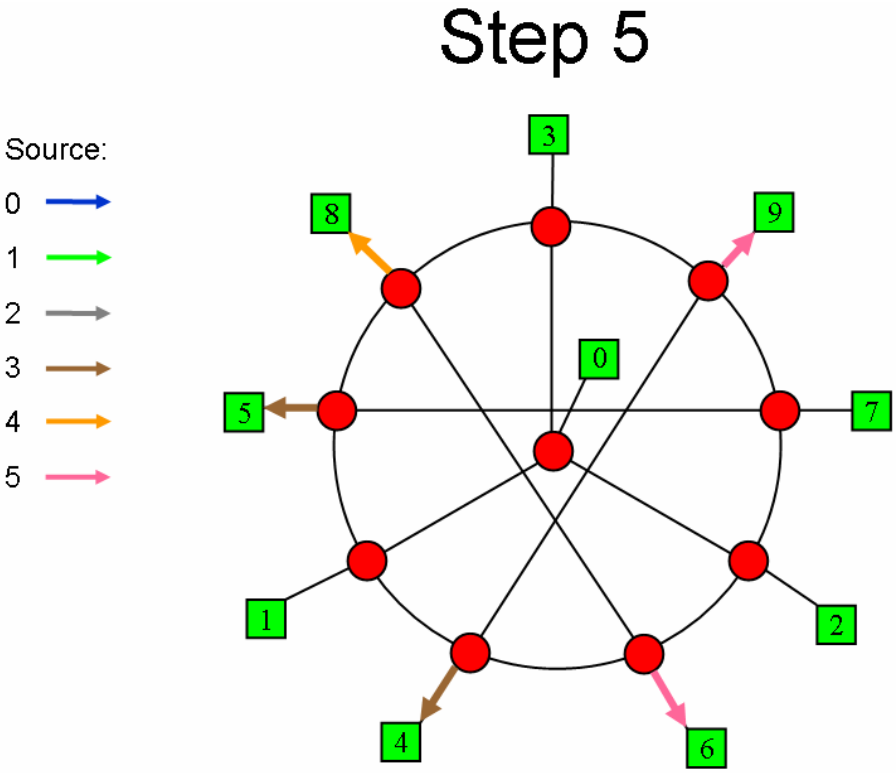
Example of Optimal MNB Communication Schedule on Moore Graph Topology




Step 2
Source:
0
1
2
3
4
5
3
8
9
0
5
7
1
2
4
6


Step 3
Source:
0
1
2
3
4
5
3
8
9
0
5
7
1
2
4
6


Step 4
Source:
0
1
2
3
4
5
3
8
9
0
5
7
1
2
4
6


Step 5
Source:
0
1
2
3
4
5
3
8
9
0
5
7
1
2
4
6

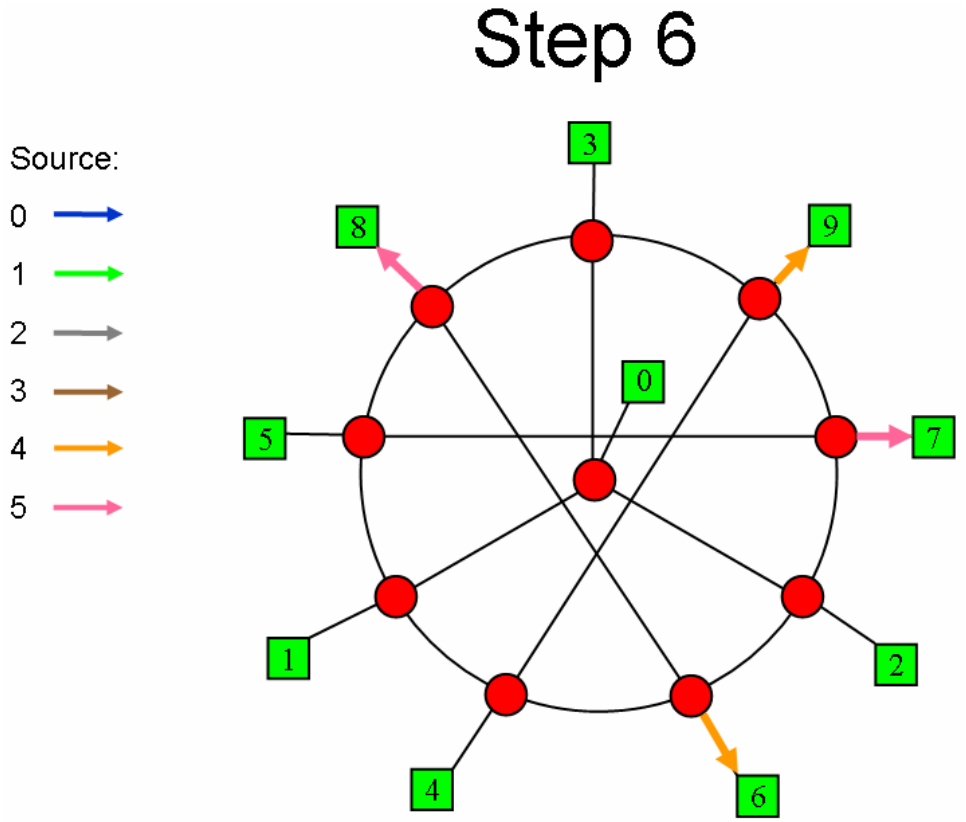


Figure 64: Model of MNB communication: store-and-forward switching, full duplex links, one-port non-combining model; the set of senders *M*: 0, 1, 2, 3, 4, 5 and the set of receivers *N*: 4, 5, 6, 7, 8, 9.