Jiří Jaroš

Evolutionary Design of Collective Communications on Wormhole Networks

Dissertation thesis

Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic

Cover design 2010 by Dagmar Hejduková

Published by Faculty of Information Technology,
Brno University of Technology, Brno, Czech Republic

Printed by MJ servis, spol. s r.o.

ISBN 978-80-214-4208-5

Preface

This thesis describes the technique of the evolutionary design aimed at scheduling of collective communications on interconnection networks of parallel computers. In order to avoid contention for links and associated delays, collective communications proceed in synchronized steps. A minimum number of steps is sought for the given network topology, wormhole (pipelined) switching, minimum routing and given sets of sender and/or receiver nodes. The proposed technique is not only able to re-invent optimum schedules for known symmetric topologies like hypercubes, but it can find schedules even for any asymmetric, irregular, multistage and fat 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 communication pattern; if it does not, non-minimum routing can provide further improvement. Optimal schedules may serve for writing high-performance communication routines for application-specific networks on chip or for the development of communication libraries in the case of general-purpose interconnection networks.

Acknowledgement

I would like to thank my supervisor Josef Schwarz who has been a permanent source of stimulation and encouragement through 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 Václav Dvořák for his inspiration and productive discussions. Special thanks must go to Lukáš Sekanina for a lot of helpful suggestions and comments to my research work. I am particularly grateful to Kenneth Froehling for his high-quality proof-reading. I would also like to thank my colleagues Michal Bidlo and Zbyšek Gajda for many discussions and ideas we shared. I would like to thank to all other members of the Department of Computer System, Faculty of Information Technology, Brno University of Technology with whom I have had the pleasure to work.

My heartfelt appreciation must go to my dear parents Jiřina Jarošová and Bohuslav Jaroš for their endless encouragement and steadfast support in all my endeavors and for believing in me. I would like to thank my sweetheart Alena Komárková for her love, patience and sacrificing much of our leisure time. Last but not least I would like to thank all my friends from the Marshal band for keeping me sane and providing welcome distractions.

Over the years, this work has been partially supported by:

- *Security-Oriented Research in Information Technology*, Research Plan MSM 0021630528, CEZ MŠMT, 2007-2013.
- *Natural Computing on Unconventional Platforms*, GP103/10/1517, Czech Science Foundation, 2010-2013.
- *Safety and security of networked embedded system applications*, GA102/08/1429, Czech Science Foundation, 2008-10.
- *Secured, reliable and adaptive computer systems*, FIT-S-10-1, Brno University of Tehcnology, 2010.

Contents

List of Figures

List of Tables

List of Abbreviation

AAB	All-to-All Broadcast
AAG	All-to-All Gather
AAR	All-to-All Reduce
AAS	All-to-All Scatter
ACO	Ant Colony Optimization
AMP	A Minimum Path topology
AOG	All-to-One Gather
AOR	All-to-One Reduction
BD	Bayesian-Dyrichlet metric
BFS	Breadth-First Search algorithm
BIC	Bayesian Information Criterion
BMDA	Bivariate Marginal Distribution Algorithm
BOA	Bayesian Optimization Algorithm
CC	Collective Communication
cGA	Compact Genetic Algorithm
COMMIT	Combining Optimizers with Mutual Information Trees
CPU	Central Processing Unit
CS	Circuit Switching
CX	Cycle Crossover
DMA	Direct Memory Access
DT	Double Tree algorithm
EA	Evolutionary Algorithm
ECGA	Extended Compact Genetic Algorithm
EDA	Estimation of Distribution Algorithm
EDN	Extended Dominating Node
EP	Evolutionary Programming
ES	Evolutionary Strategies
FD	Full Duplex link
FDA	Factorized Distribution Algorithm
FFT	Fast Fourier Transform
FIFO	First In First Out
GA	Genetic Algorithm
GCR	General Corodal Ring
GP	Genetic Programming

HCwL	Hill Climbing with Learning
HD	Half Duplex link
HKA	Ho-Kao Algorithm
HPC	High Performance Computing
HW	HardWare
IDEA	Iterated Density Estimation Algorithm
IUMDA	Incremental Univariate Marginal Distribution Algorithm
LC	Link Controller
MBOA	Mixed Bayesian Distribution Algorithm
MC	MultiCast
MDL	Minimum Description Length Metric
MILP	Mixed Integer Linear Programming
MIMIC	Mutual Information Maximizing Input Clustering
MIN	Multistage Interconnection Network
MMB	Many-to-Many Broadcast
MNS	Many-to-Many Scatter
MPI	Message Passing Interface
MPP	Massive Parallel Processor
NE	Number of Evaluations to reach global optima
NoC	Network on a Chip
OAB	One-to-All Broadcast
OAS	One-to-All Scatter
ODD	Optimal Diameter-Degree topology
OX	Ordered Crossover
PBIL	Population Based Incremental Learning
PE	Processing Element
PMBGA	Probabilistic Model Building Genetic Algorithm
PMX	Partially Mapped Crossover
PSO	Particle Swarm Optimization
SBT	Spatial Binomial Tree
SE	Switching Element
SF	Store-and-Forward switching
SGA	Standard Genetic Algorithm
SoC	System on a Chip
SR	Success Rate of the global optima discovery
ST	Sequential Tree algorithm
SW	SoftWare
TADT	Time-Arc-Disjoint Tree
TFLOP	Tera FLoating point OPerations per second
TSG	Transmission SubGraph
TSP	Traveling Salesman Problem
UMDA	Univariate Marginal Distribution Algorithm
VCT	Virtual Cut-Through switching
WH	WormHole switching

List of Symbols

Δ	Number of messages that cross the bisection twice
λ_{CC}	Link utilization
$\boldsymbol{B}$	Link frequency
$\boldsymbol{B_C}$	Bisection width
$\boldsymbol{c}$	Channel
$\boldsymbol{C}$	Set of channels
$\boldsymbol{C(x,y)}$	Cut of the network
$\boldsymbol{COMM}$	Set of communications
$\boldsymbol{D}$	Network diameter
$\boldsymbol{d}$	Node degree
$\boldsymbol{d_c}$	Destination node of the channel *c*
$\boldsymbol{E}$	Multiset covering all shared channels within CC
$\boldsymbol{E_{a,b}}$	Set covering all shared channels between transfers *a* and *b*
$\boldsymbol{E}\mathbf{s}$	Multiset covering all shared channels within a step *s*
$\boldsymbol{h}$	Hop count
$\boldsymbol{H(x,y)}$	Hop count of the minimal path between *x* and *y*
$\boldsymbol{k}$	Number of internal ports
$\boldsymbol{l}$	Path
$\boldsymbol{m}$	Message length
$\boldsymbol{M}$	Number of transmitters
$\boldsymbol{MSG}$	Set of messages
$\boldsymbol{N}$	Number of receivers
$\boldsymbol{P}$	Number of terminal nodes
$\boldsymbol{Q}$	Number of nodes operating simultaneously as a transmitter and a receiver
$\boldsymbol{r}$	Restricted surrounding radius
$\boldsymbol{R}$	Set of receivers
$\boldsymbol{R}'_{\mathrm{x,y}}$	Set of all paths between *x* and *y*
$\boldsymbol{R_{x,y}}$	Set of minimal paths between *x* and *y*
$\boldsymbol{s}$	Communication step
$\boldsymbol{S}$	Set of communication steps
$\boldsymbol{s_c}$	Source node of the channel *c*
$\boldsymbol{SS}$	Same step relation
$\boldsymbol{T}$	Set of transmitters

$\boldsymbol{t_r}$	Routing delay
$\boldsymbol{t_s}$	Switching delay
$\boldsymbol{t_w}$	Propagation delay
$\boldsymbol{V}$	Set of terminal nodes
$\boldsymbol{V^*}$	Set of nodes
$\boldsymbol{W}$	Channel width
$\boldsymbol{x}$	Source node
$\boldsymbol{y}$	Destination node
$\boldsymbol{\Sigma}$	Cumulative inter-nodes distance
$\boldsymbol{\tau_{CC}}$	Lower bound of communication complexity
$\boldsymbol{\tau^{CC}}$	Upper (reached) bound of communication complexity

1 Introduction

A recent trend in high performance computing (HPC) has been towards the use of parallel processing to solve computationally-intensive problems. Several parallel architectures, which offer corresponding increases in performance as the number of processors is increased, have been designed in the last few years. Nowadays, with the enormous transistor budgets of 45-nm and 32-nm technologies on a silicon die, large CPU clusters are feasible to place on a single chip (System on Chip, SoC) allowing both large local memories and the high bandwidth of on-chip interconnection. Using this chip-scale multiprocessing, the number of processors on a chip may in the near future scale to dozens or hundreds, depending on their complexity. The basic requirement for building such a SoC has turned out to be the low power consumption, in order that system parts could be close together and communication time would be thus minimized. For the same reason, the CPU cores should be simple and processing nodes should be interconnected as effectively as possible.

Buses and point-to-point connections are the main means to connect the components. Buses can efficiently connect 3-10 communication partners but they do not scale to higher numbers. Even worse, they behave very unpredictably, as seen from an individual component, because many other components also use them. A second problem comes from the physics of deep submicron technology. Long, global wires and buses become undesirable due to tight timing constraints and skew control, high power consumption and noise phenomenon [86], [96].

As a consequence, in 1999 several research groups started to investigate systematic approaches to the design of the communication part of SoCs. This research area has been called Network on Chip (NoC). NoCs [38], [81], [110] are constructed from multiple point-to-point data links interconnected by switches (routers), so that messages can be relayed from any source module to any destination module over several links by making routing decisions at the switches. Although NoCs can borrow concepts and techniques from the well-established domain of computer networking, it is impractical to blindly reuse features of "classical" computer networks and symmetric multiprocessors. In particular, NoC switches should be small, energy-efficient, and fast. The routing algorithms should be implemented by a simple logic, and the number of data buffers should be minimized. These requirements have converged on the use of pipelined, distance-insensitive wormhole (WH) [146] message switching and source-based routing algorithms [39].

There are several examples of NoCs that have found applications in the commercial sphere at the present time. One of the leaders in this area is Tilera Corporation that has introduced Tilera Tile-Gx processor family with 16 to 100 full-featured processing cores interconnected by a 2D mesh [199]. Other compact high performance systems have been produced by SiCortex, Inc. This vendor has offered highly compacted systems based on unidirectional Kautz networks and scalable up to 5832 cores with only 20kW of power consumption [184]. Also IBM has brought its solution called IBM Cell broadband engine on the market. This multimedia chip integrates a Power PC processor and eight SPE elements interconnected by a ring NoC called Element Interconnection Bus. The chip has found many commercial applications not only in the gaming industry (Sony Playstation 3) [186], but also in the second most powerful supercomputer in the world called Roadrunner [202]. Similarly, Intel is working on its own NoC solution under the Intel Tera-scale research project (also called Larabee). The goal of the project is to design a NoC with 1TFLOP performance composed of about 80 cores interconnected by either a mesh or a hierarchical ring topology [94]. The small scale systems can be also based on common AMD Opteron and Intel Nehalem architecture processors. The Network on Board is then created using HyperTransport or QuickPath links organized into bidirectional rings (Intel Nehalem Ex) [93] or hypercubes (AMD Direct Connect Interconnection [4]). And this is only the beginning of the NoC area. Many other implementations will certainly come into existence in a few years.

In order to be able to utilize the performance of such a SoC, the parallel programming paradigms have to be taken into account. Provided that computation times of executed tasks are known, as is usually true in case of application-specific systems, the only thing that matters in obtaining the highest performance are durations of various collective communications. 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 [98]. However, many general-purpose parallel applications display a bulk-synchronous behavior: the processing nodes access the network according to a global, structured communication pattern. Examples of collective communication (CC) patterns include broadcast, in which a message is sent from one process to all the other processes in a group; global combine, in which a global operation, such as maximum or sum, is performed on a distributed set of data items; and barrier synchronization, in which every member of a set of processes must reach a given point in its execution before any member can proceed [111]. The growing interest in the use of collective routines is evidenced by their inclusion in the Message Passing Interface (MPI) standard [131] and by their increasing role in supporting data-parallel languages [83]. Many existing SoCs do not support collective operations in hardware. In these environments, collective operations must be supported in software by sending multiple point-to-point messages. Such implementations are termed unicast-based [130] and typically are implemented as a sequence of synchronized steps, each of which involves the sending of one or more messages among processes. However, in situations where many messages exist in the network concurrently, a large internode distance can lead to contention among messages. Therefore, unicast-based collective operations, which typically involve many messages, should be designed so that they not only minimize the number of message-passing steps, but also minimize or eliminate contention among the constituent messages [129]. The achievement of this goal depends on additional architectural characteristics which is the main goal of this thesis.

At present, many different universal topologies of interconnection networks are in use and other application-specific ones can be created on demand. While the time complexity of a certain communication pattern has a lower bound given by a particular interconnection, finding a sequence of communication steps (a schedule) approaching this limit is more difficult, and in some cases, such schedules are not known as of yet.

Naturally, many projects have addressed the design of fast collective communication algorithms for wormhole-switched systems in recent years. Since any data loss is not acceptable in NoC, the deadlocks, livelocks and starvations, even links/node overloads, have to be prevented in such schedules. Hence, many approaches have analyzed the structure and properties of the underlying NoC topology and communication pattern with the aim of designing contention-free communication schedules that attain the lower bound of time complexity of given CC patterns [1], [120], [174]. Unfortunately, these schedules are not general at all, and only work for a few regular topologies such as hypercube or square mesh/tours even then in only a couple of possible instances. Another idea is to design some families of parameterized algorithms that can be tuned to perform well on different architectures under various system conditions [15]. Unfortunately, this kind of CC schedules is not optimal in most cases, and moreover they are restricted by other parameters of the NoC such as port model, minimal routing strategy, symmetry of the network and so on.

With an increasing number of novel NoC topologies (e.g. spidergon, Kautz, fat topologies) desire for a general technique capable to produce optimal or near optimal schedules for an arbitrary network topology and a given CC pattern steadily grows. The designed schedules could serve for writing high-performance communication functions for a concrete topology. Consequently, these functions could be included into, for example, the well-known OpenMPI library [92] to accelerate given CCs and prevent data losses.

Due to the complexity of this task and lack of suitable conventional techniques, evolutionary algorithms (EA) are going to be employed in this work. Since EAs were introduced in 1960s [87], several researchers have demonstrated that EAs are effective and robust in handling a wide range of difficult real-world problems such as optimization, decomposition, scheduling and design [25], [33], [65], [122].

This thesis deals with the experimental confirmation of this hypothesis: *Evolutionary design is able to produce optimal or almost optimal communica-*

tion schedules comparable or even better than which have been obtained by a conventional design for the networks sizes of interest. Moreover, evolutionary design will reduce many drawbacks of present techniques and invent still unknown schedules for an arbitrary topology and scatter/broadcast communication patterns.

1.1 Thesis Structure

This thesis describes the universal technique that is able to design optimal or near optimal communication schedules for an arbitrary network topology and an arbitrary distribution of communicating nodes. Moreover, this technique will overcome many bottlenecks of traditional techniques like port-model restriction, minimal routing restriction, regularity and symmetry of the network. Thereupon, it will be able to design schedules for general many-to-many communication patterns, fat and multistage networks and even deal with the problem of the faulty topologies.

The thesis is structured into nine chapters describing the theoretical background, the analysis of the state of the art, the description of the proposed method, the experimental validation of the method and the contributions of the thesis.

Chapter 2 describes the interconnection networks basis. The chapter starts with the description of the graph structure and the parameters of NoC topologies. Then, the switching techniques and their influences on the network performance are discussed. Next, the chapter summarizes the principles of routing mechanisms used in NoCs. Finally, the problems of deadlock and livelock are introduced.

Chapter 3 deals with the definition of collective communications in wormhole networks. First, the model of communication is introduced, and then, the CCs are classified into several classes according to the distribution of transmitters and receivers. The estimation of the lower bounds on time complexity and examples of optimal schedules are given for investigated CC patterns. Finally, the general many-to-many CC are examined.

Chapter 4 explores in depth the state of the art in the area of NoC and collective communication scheduling. There are several promising NoC architectures described in the first subsection. The rest of the chapter addresses the issue of optimal CC scheduling. The best known approaches are introduced for all communication patterns, and their pros and cons are then discussed. The knowledge in the area is concluded and the outstanding problems are identified at the end of the chapter.

Chapter 5 provides the detailed description of principles of evolutionary algorithm, their classification, capabilities and drawbacks. In addition, all the parts of evolutionary algorithms are examined in depth and designers' hints are provided.

The technique of evolutionary design of CCs on wormhole networks is introduced in chapter 6. First, the input data structure and their preprocessing are described. After that, the structure of schedules encodings into chromosomes is introduced. The mathematical definitions are completed by simple graphic visualizations for both scatter and broadcast based CCs. The definition of the fitness function follows in succession. Next, the acceleration and restoration heuristics are investigated. These heuristics exploit the search space restriction and take the evolution into more promising areas. On the other hand, the restorations heuristic repairs the illegal solutions and allows evolution to operate in the strongly restricted search space of broadcast based CCs.

The goal of chapter 7 is the selection of the most suitable evolutionary optimization tool. Three potential candidates are chosen and their optimal parameters are searched for in the first part of the chapter. Next, the quality of obtained solutions is investigated and compared with each other. The comparisons of optimization speed and optimization scalability for given tools follow. Finally, the most suitable optimization tool is elected and used for other work.

Chapter 8 provides a comprehensive study of the capabilities of the proposed technique. The technique is applied to a lot of kinds of interconnection networks and the time complexities of the evolved schedules are compared with the mathematically derived lower bounds. Moreover, the most interesting schedules are provided in the tabular form. The networks of interest in this chapter cover common topologies such as hypercube, ring, mesh and torus, special diameter degree networks and novel application-specific networks. Another effort is devoted to the multistage, fat and faulty topologies that have not been studied before. Finally, the basic results for general many-to-many communications are given.

Chapter 9 concludes the thesis, identifies the main contributions of the proposed technique and outlines the most promising direction for future work.

2 Interconnection Networks

Digital electronic systems of all types are rapidly becoming communication limited. Movement of data, not arithmetic or control logic is the factor limiting cost, performance, size and power in these systems. At the same time, buses, long the mainstay of system interconnect, are unable to keep up with increasing performance requirements.

Interconnection networks offer an attractive solution to this communication crisis and are becoming pervasive in digital systems. A well-designed interconnection network makes efficient use of scarce communication resources - providing high-bandwidth, low-latency communication between clients with a minimum of cost and energy [98].

Historically used only in high-end supercomputers and telecom switches, interconnection networks are now found in digital systems of all sizes and types. They are used in systems ranging from large supercomputers to small embedded systems-on-a-chip (SoC) [3] and in applications including inter-processor communications, processor-memory interconnect, input/output and storage switches, router fabrics, and to replace dedicated wiring.

Indeed, as system complexity and integration continues to increase, many designers are finding it more efficient to route packets, not wires [38]. Using an interconnection network rather than dedicated wiring allows scarce bandwidth to be shared so it can be used efficiently with a high duty factor. In contrast, dedicated wiring is idle much of the time. Using a network also enforces a regular, structured use of communication resources, making the system easier to design, debug, and optimize.

2.1 Network Basics

In order to meet the performance specifications of a particular application, the network designer must work within topology constraints to implement the topology, routing and flow control of the network. 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 pair, 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 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 (hops), while balancing the demand placed on the shared resources of the network. The length of a path obviously influences the latency of a message through the network, and the demand or load on a resource is a measure of how often that resource is being utilized. If one resource becomes over-utilized while another sits idle, known as a load imbalance, the total bandwidth of messages being delivered by the network is reduced.

Flow control dictates which messages get access to particular network resources over time. This influence of flow control becomes more critical as the utilization of resource increases and good flow control forwards packets with minimum delay and avoids idling resources under high loads.

2.2 Topology

Network topology refers to the static arrangement of channels and nodes in an interconnection network - the roads over which packets travel. Selecting the network topology is the first step on designing a network because the routing strategy and flow-control method depend heavily on the topology.

Selecting a good topology is largely a job of fitting the requirements of the network into the available packing technology. We can choose a topology based on its cost and performance. The cost is determined by the number and complexity of the chips required to realize the network, and the density and length of interconnections between these chips on boards or over cables. Performance has two components: *bandwidth* and *latency*. Both of these measures are determined by factors other than topology (flow control, routing strategy and traffic pattern). In order to evaluate just the topology we develop measures, such as bisection bandwidth, channel load, and path delay. They reflect the impact of the topology on performance.

Figure 2.1 shows the basic interconnection network topologies: (a) and (b) show 16 nodes orthogonal topologies, 2D torus [223] and 4D hypercube [222] while (c) and (d) show another commonly used network topologies based on ring: corodal ring known also as K-Ring [112], and hierarchical rings [94].

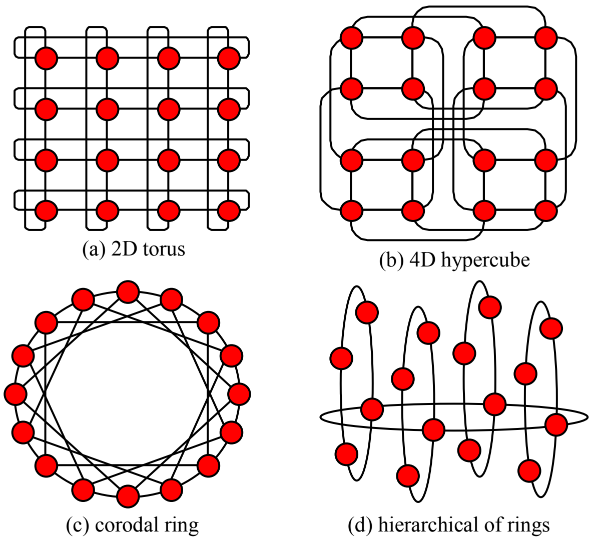


Figure 2.1. Four examples of interconnection network topologies.

2.2.1 Channels and Nodes

The topology of an interconnection network is specified by a set of *nodes* V^* connected by a set of *channels* C. Messages originate and terminate in a set of *terminal nodes* V where $V \subseteq V^*$. In networks where all nodes are terminals, we simply refer to the set of nodes as V. Each channel $c = (x, y) \in C$ connects a source node x to a destination node y, where $x, y \in V^*$. We denote the source node of a channel c as s_c and the destination as d_c. This definition of a topology is equivalent to a directed graph [211]. Not surprisingly, much of the terminology used to describe a topology borrows heavily from graph theory. Note that each link (edge) can consist of two unidirectional channels considering full-duplex links or only one channel in the case of half-duplex or simplex link.

A channel $c = (x, y)$ is characterized by its width w_c or w_{xy}, the number of parallel signals it contains; its f_c or f_{xy}, the rate at which bits are transported on each signal; and its latency t_c or t_{xy}, the time required for a bit to travel from x to y.

In *direct networks*, besides external channels (simply called channels) connecting particular switches/routers, we also introduce internal channels (simply called *ports*) to connect local processor/memory to the switch. Although it is common to provide only one port (one pair of internal channel, $k = 1$), some systems use more ports ($k \leq d$) in order to avoid a communication bottleneck between the local processor/memory and the switch [22].

The architecture of a generic one-port router with four input/output links is shown in Figure 2.2. The packets arrive at input link controllers and are stored in buffers. Routing and arbitration are responsible for the next cannel selection and the switching fabric allocation. The packets can be either forwarded to one of the output buffers or consumed by a local ejection channel. The output link controllers dispatch the packets from the output buffers as soon as their output links are free. New messages are injected into the network using a local injection channel.

2.2.2 Direct and Indirect Networks

A network node can be a terminal node that acts as a source and destination for packets, a switch node that forwards packets from input ports to output ports, or both. In a direct network [146], such as the 2D mesh in Figure 2.3a, every node in the network includes both a terminal and a switch/router. On the other hand, in an *indirect network*, such as the 2D coated mesh in Figure 2.3b, a node can be either a terminal (rectangular nodes) or a switch (round nodes). It cannot serve both functions. In direct networks, packets are forwarded directly between terminal nodes, while in an indirect network they are forwarded indirectly by means of dedicated switch nodes. Some networks, like the random network in Figure 2.3c, are neither direct nor indirect. Every direct network can be redrawn as an indirect network by splitting each node into separate terminal and switch nodes. The distinction between direct and indirect networks is largely academic with such networks.

One potential advantage of a direct network is that resources of a terminal (which usually include a computer) are available to each switch. 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 [181]. Software switching is, however, both very slow and demanding of the terminal's resources. Thus, it is rarely used today.

2.2.3 Cuts and Bisection

A *cut of network* $C(V_1, V_2)$ is a set of channels that partitions the set of all nodes V^* into two disjoint sets, V_1 and V_2. Each element of $C(V_1, V_2)$ is a channel with a source in V_1 and destination in V_2, or vice versa. The number of channels in the cut is $|C(V_1, V_2)|$.

A *bisection* of a network is a cut that partitions the entire network nearly in half, so that $|V_2| \leq |V_1| \leq |V_2| + 1$, and also partitions the terminal nodes nearly in half, so that $|V_2 \cap V| \leq |V_1 \cap V| \leq |V_2 \cap V| + 1$. The channel bisection of a network B_C is the minimum channel count over all bisections of the network

$$B_C = \min_{bisection} |C(V_1, V_2)|. \tag{2.1}$$

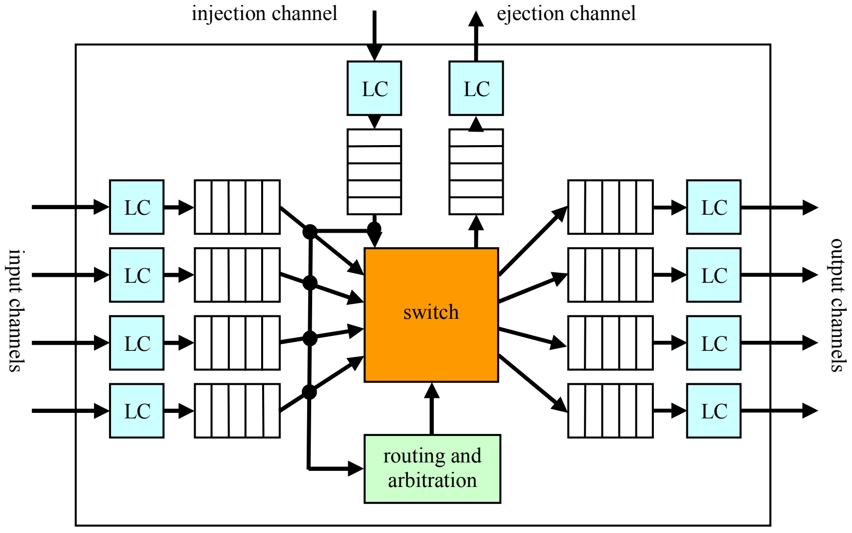


Figure 2.2. Generic one-port router model (LC = link controller).

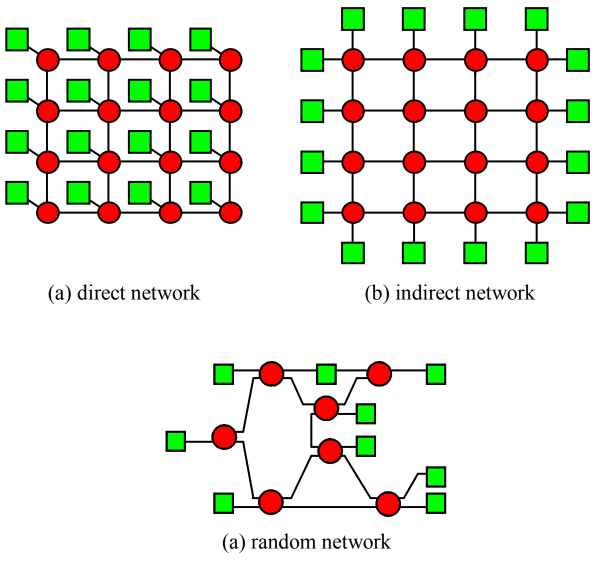


Figure 2.3. Direct, indirect and random network topologies.

2.2.4 Paths

A *path* in a network is an ordered set of channels $l = \{c_1, c_2,..., c_L\}$, where for $i = 1...L-1$. Paths are also referred to as routes. The source of a path is $s_l = s_{c1}$. Similarly, the destination of a path is $d_l = d_{cL}$. The length or hop count of a path is $h = |l|$. 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.

The *minimal (shortest) path* from node x to node y is a path with the smallest hop count connecting these two nodes. The set of all minimal paths from node x to node y is denoted $R_{x,y}$. $H(x, y)$ is the hop count of minimal path between x and y. The *diameter* of a network D is the largest, the minimal hop count over all pairs of terminal nodes in the network being

$$D = \max_{x,y \in V} H(x, y) \cdot \tag{2.2}$$

A network with multiple shortest paths between most pairs of nodes, $|R_{x,y}| > 1$ for most $x, y \in V$ is more robust than a network with only a single route from node to node $|R_{x,y}| = 1$. This property, which is called path diversity, adds to the robustness of our network by balancing across channels and allowing the network to tolerate faulty channels and nodes.

2.2.5 Symmetry

The symmetry of a topology plays an important role in load-balance and routing. A network is *vertex-symmetric*, as in Figure 2.4a, if there exists an automorphism that maps any node x into another node y. Informally, in a vertex-symmetric network, the topology looks the same from the point-of-view of all the nodes. This can simplify routing because all nodes share the same roadmap of the network and, therefore, can use the same directions to route to the same relative position. In an *edge-symmetric network*, as in Figure 2.4b, there exists an automorphism that maps any channel a into another channel b. Edge symmetry can improve load balance across the channels of the network since there is no reason to favor one channel over another.

2.3 Switching

Inter-processor communication can be viewed as a hierarchy of services starting from the physical layer that synchronizes the transfer of bit streams to higher-level protocol layers that perform functions such as packetization, data encryption, data compression, etc. We find it useful to distinguish between three layers in the operation of the interconnection networks: the routing layer, the switching layer and the physical layer [95]. The *physical layer* refers to link-level protocols for transferring messages and otherwise managing the physical channels

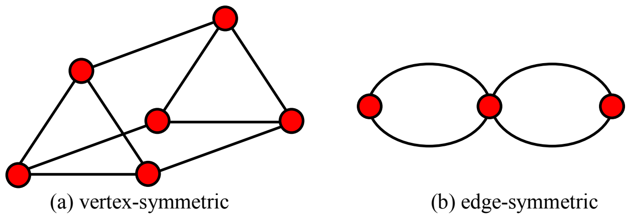


Figure 2.4. Vertex and edge symmetric networks.

between adjacent switches/routers. The *switching layer* utilizes these physical layer protocols to implement mechanisms for forwarding messages through the network. Finally, the *routing layer* makes decisions to determine candidate output channels at the intermediate router node and, thereby, establish the path through the network. The design of routing protocols and their properties (e.g. deadlock and livelock freedom) are largely determined by the services provided by the routing layer.

Switching techniques employed in multiprocessor networks initially followed those techniques employed in local and wide-area networks. However, as the application of multiprocessor systems spread into increasingly compute-intensive domains, the traditional layered communication designs borrowed from LANs became a limiting performance bottleneck. New switching techniques and implementations evolved that were better suited to the low latency demands of parallel programs.

In order to traverse an interconnection network, a message must be allocated resources: channel bandwidth, buffer capacity, and control state. In order to improve efficiency of this resource allocation, we divide a message into packets for the allocation of control state and into flow control digits (flits) for the allocation of channel bandwidth and buffer capacity.

2.3.1 Messages, Packets, Flits and Piths

Figure 2.5 shows the units in which network resources are allocated [113]. At the top level, a *message* is a logically contiguous group of bits that are delivered from a source terminal to a destination terminal. Because messages can be arbitrarily long, it is necessary to break them into one or more *packets* that have a restricted maximum length.

A packet is a basic unit of routing and sequencing. A packet consists of a segment of a message to which a packet header is pretended to exist. The packet header includes routing information and, if needed, a sequence number. The routing information is used to determine the route taken by the packet from source to destination. The sequence number is needed to reorder the packets of a message if they can get out of order in transit (transmitting via different paths).

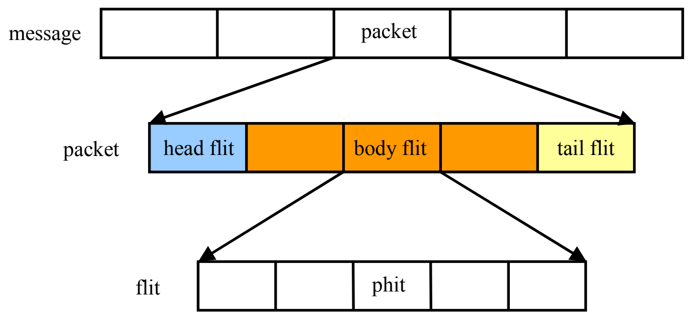


Figure 2.5. Units of resource allocation (message, packet, flit and phit).

A packet can be further divided into flow control digits or *flits*. A flit is the basic unit of bandwidth and storage allocation used by most flow control (switching) techniques. Flits carry no routing and sequencing information and thus must follow the same path and remain in order. The position of a flit in packet determines whether it is a head flit, body flit, tail flit, or a combination of these. A head flit is the first flit of a packet and carries the packet's routing information. A head flit is followed by zero or more body flits and a tail flit. In a very short packet, there can be no body flits, and in the extreme case where a packet is a single flit, the head can also be a tail flit. As the packet traverses the network, the head flit allocates a channel state for the packet and the tail flit deallocates it.

A flit is itself subdivides into one or more physical transfer digits or *phits*. A phit is the unit of information that is transferred across a channel in a single clock cycle. Although no resources are allocated in units of phits, a link level protocol must interpret the phits on the channel to find the boundaries between flits.

For purposes of comparison, for each switching technique we will consider the computation of the *base latency* of an *m*-bit message in the absence of any traffic. The pith size and the 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; thus the message size is $m + W$ bits. A router can make a routing decision in t_r seconds. The physical channel between two routers operates at B Hz: that is, the physical channel bandwidth is $B \cdot W$ bits per second. In this work, we assume that channel wires are short enough to complete a transmission in one clock cycle. Therefore, the *propagation delay* across this channel is denoted by $t_w = B^{-1}$. Once a path has been set up through the router, the intra-router delay or *switching delay* is denoted by t_s. The source and destination terminal nodes are assumed to be h link apart.

2.3.2 Circuit Switching

In *circuit switching* (CS), a physical path is reserved from source to destination before the transmission. A header of the message is injected into the network, and when it reaches the destination, the complete path has been set up and an acknowledgment is sent back to source so that message contents may be transmitted at the full bandwidth of the hardware path (telephone system). The circuit may be released by the destination or by the last few bits of the message [187].

Circuit switching is advantageous when messages are infrequent and long (i.e. when the transmission time is long compared to the path setup time). On the other hand, the physical path is reserved for the duration of the message and can block other messages.

The base latency of a circuit-switched message is determined by the time to set up a path and the subsequent time the path is busy transmitting data. For circuit-switching we can write an expression for the base latency of a message as follows:

$$\begin{aligned} t_{circuit} &= t_{setup} + t_{data} \\ t_{setup} &= h\,[t_r + 2(t_s + t_w)] \\ t_{data} &= \frac{1}{B}\left\lceil \frac{m}{W} \right\rceil . \end{aligned} \tag{2.3}$$

Actual latencies clearly depend on a myriad of implementation details. In particular, it is assumed that once the circuit has been established, propagation delay through the entire circuit is negligible compared to clock cycle. Hence, t_{data} does not depend on that delay. The factor of 2 in the setup cost represents the time for the forward progress of the header and the return of the acknowledgement. The use of *B* Hz as the channel speed represents the transition across a hardwired path from source to destination.

Figure 2.6 presents the time-space diagram showing transmission of one 4-data flit packet over 5 hops using circuit switching under congestion (blocked channel 3). The request flit allocates the path and the tail flit deallocates it.

2.3.3 Store-and-Forward Switching

If the size of the message is not much greater than the size of the routing header, it becomes advantageous to partition and transmit the message as fixed-length packets. The first few bytes of a packet contain routing and control information stored in the packet header. Every packet is individually routed from source to destination. This is the reason why this switching technique is called as *store-and-forward* switching (SF) [180]. 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 time-space diagram is shown in Figure 2.7. From this

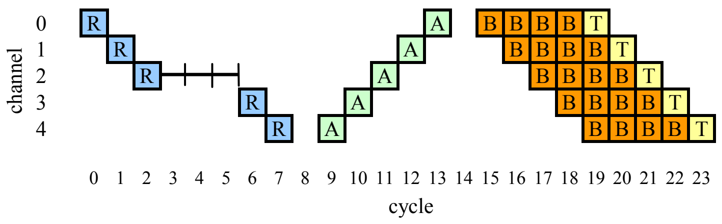


Figure 2.6. Time-space diagram showing circuit switching under congestion(R - request, A - acknowledgement, B - body flit, T - tail flit).

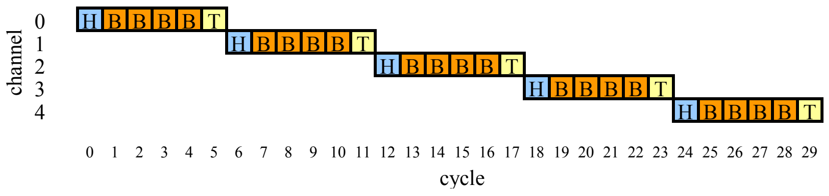


Figure 2.7. Time space diagram showing store-and-forward switching without congestion (H - head flit, B - body flit, T - tail flit).

figure we can see that the latency experienced by the packet is proportional to the distance between the source and destination nodes.

Store-and-forward switching is advantageous when messages are short and more frequent, so that full utilization of the communication link can be realized. Many packets belonging to a message can be in the network simultaneously, even if the first packet has not arrived to the destination yet. In this switching mechanism every node must buffer every incoming packet, consuming memory space. In multidimensional, point-to-point networks it is evident that the storage requirements at the individual router nodes can become extensive if packets can become large and multiple packets must be buffered at a node.

The base latency of the store-and-forward switching and other packet switched messages can be computed as follows:

$$t_{packet} = h\left\{ t_r + (t_s + t_w)\left\lceil \frac{m+W}{W} \right\rceil \right\}. \tag{2.4}$$

This expression follows the router model in Figure 2.2 and, as a result, includes a factor to represent the time for the transfer of a packet of length $m + W$ bits across the channel (t_w) as well as from the input buffer of the router to the output buffer (t_s). The important point to note is that the latency is directly proportional to the distance (hop count) h between the source and destination node.

2.3.4 Virtual Cut-Through Switching

In order to decrease the amount of time spent transmitting data and full buffer requirement, a *virtual cut-through* method [45] is introduced in which a packet is stored at an intermediate node, only if the next required channel is busy. As soon as enough information is available in the intermediate nodes, forwarding begins even before the entire message has been received. At high network loads virtual cut-through (VCT) behaves like packet switching.

Figure 2.8 illustrates the time-space diagram of a message transferred using VCT switching where the message is blocked after the first link waiting for an output channel to become free. In this case we can see that the complete packet has to be transferred to the first router where it remains blocked waiting for the free output port. However, from the figure we can see that the message is successful in cutting through the second router and across the third link.

The based latency of a message that successfully cuts-through each intermediate router can be computed as follows:

$$t_{vct} = h(t_r + t_s + t_w) + \max(t_s, t_w)\left\lceil \frac{m}{W} \right\rceil. \tag{2.5}$$

As we can see from this formula, the first addend is not a function of the message size and the transmition delay is not a function of the hop count. Considering t_r, t_s and t_w insignificant and no congestion, the virtual cut-through is often distinguished as distance insensitive.

2.3.5 Wormhole Switching

Wormhole switching (WH) [146] is a special case of the cut-through switching. Instead of storing a packet completely in a node and then transmitting it to the next node, wormhole switching operates by advancing the head of a packet directly from incoming to outgoing channels of the routing chip. A packet is divided into a number of flits (flow control digits) for transmission. The size of a flit depends on system parameters, in particular, the channel width *W*. The header flit (or flits) governs the route. As soon as a node examines the header flit(s) of a message, it selects the next channel on the route and begins forwarding flits down that channel. As the header advances along the specified route, the remaining flits follow in a pipeline fashion. Because most flits contain no routing information, the flits in a message must remain in contiguous channels of the network and cannot be interleaved with the flits of other messages. The time-space diagram of wormhole-switched message is shown in Figure 2.9.

Wormhole switching avoids memory bandwidth in the nodes through which messages are routed. Only a small FIFO (First-In-First-Out) flit buffer can be used. It also makes the network latency largely insensitive to path length. The blocking characteristics are very different from VCT. If the required output

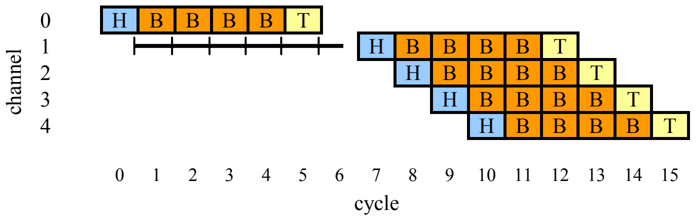


Figure 2.8. Time-space diagram of virtual cut-through switching under congestion (H - head flit, B - body flit, T - tail flit).

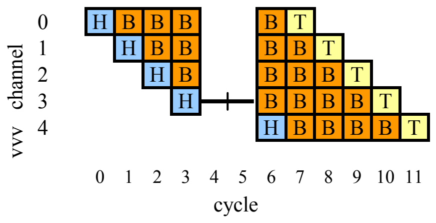


Figure 2.9. Time-space diagram of wormhole switching under congestion (H - head flit, B - body flit, T - tail flit).

channel is busy, the message is blocked "in place". The particular flits of a blocked message are stored along a part of a path from the source to the blocking node. In order to reduce the effect of message blocking, physical channels can be split into virtual channels and these will be used to increase the total throughput of the physical channel [37]. Virtual channels are logical entities associated with a physical link used to distinguish multiple data streams traversing the same physical channel. They are multiplexed over a physical channel in a demand-driven manner, with bandwidth allocated to each virtual channel as needed.

The base latency of a wormhole-switched message can be computed as follows:

$$t_{wormhole} = h(t_r + t_s + t_w) + \max(t_s, t_w)\left\lceil \frac{m}{W} \right\rceil. \tag{2.6}$$

This expression assumes flit buffers at the router inputs and outputs. Note that in the absence of contention, VCT and wormhole switching show the same packet latency.

2.4 Routing

Routing involves selecting a path from the source node to the destination node in a particular topology. The routing algorithm used for a network is critical for several reasons. A good routing algorithm balances a load across the network channels, even in the presence of a non-uniform traffic pattern such as permutation traffic [205]. The more balanced the channel load, the closer to ideal is the throughput of the network.

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. Algorithms exploiting only the minimal, or shortest paths, from the set of $R_{x,y}$ are very often referred to as *minimal*. What might not be immediately obvious is that routing minimally (always choosing the shortest path) is often at odds with a balancing load and maximizing throughput. Therefore, routing functions are often allowed to choose paths from the set of all minimal and non-minimal routes R'_{xy}. Such algorithms are referred to as *non-minimal* [127], [143].

Another important aspect of routing algorithms is their ability to work in the presence of faults in the network. If a particular algorithm is hardwired into the routers and a link or node fail, the entire systems fails. However, if an algorithm can be reprogrammed or adapt to the failure, the system can continue to operate with only a slight loss in performance [144].

We classify routing algorithms in terms of how they choose between possible paths of $R_{x,y}$ from source node x to destination node y. *Deterministic routing* [114] algorithms always choose the same path between x and y, even if there are multiple possible paths ($|R_{x,y}| > 1$). These algorithms ignore path diversity of the underlying topology and hence do a very poor job of balancing load. Despite this, they are quite common in practice because they are easy to implement and easy to make deadlock-free. *Oblivious routing* algorithms [205] choose a route without considering any information about the network's present state. For example, random algorithm that uniformly distributes traffic across all of the paths in $R_{x,y}$ is an oblivious algorithm. *Adaptive routing* algorithms [51] adapt to the state of the network using this state information in making routing decisions. This information can include the status of a node or channel, length of queries for network resources, and historical channel load information. Adaptive routing algorithms can be classified as *progressive* or *backtracking*. Progressive routing algorithms move the header forward, reserving a new channel at each routing operation. Backtracking algorithms allow to the header to backtrack, releasing previously reserved channels.

Routing algorithms can also be classified according to the place where routing decisions are taken. Basically, the path can be either established by a centralized controller (*centralized routing*) [161] at the source node prior to packet injection (*source routing*) [39] or determined in a distributed manner while the packet travels across the network (*distributed routing*) [97].

Routing algorithms can be implemented in different ways. Many routers use *routing tables* either at the source or at each hop along the route to implement the routing algorithm [26]. With a single entry per destination, a table is restricted to deterministic routing, but oblivious and adaptive routing can be implemented by providing multiple table entries for each destination. An alternative to tables is *algorithmic routing* [2], in which specialized hardware computes the route or next hop of a packet at runtime. However, algorithmic routing is usually restricted to simple routing algorithms and regular topologies.

2.4.1 Deterministic Routing

Deterministic routing algorithms establish the path as a function of destination address, always supplying the same path between every pair of nodes. A lot of earlier networks adopted deterministic routing because of being simply inexpensive to implement. Simple deterministic approaches actually load balance ones for some topologies as well as other minimal routing algorithms, including adaptive.

Determinist routing is especially used with topologies which can be decomposed into several orthogonal dimensions (e.g. hypercubes and meshes). Deterministic algorithms are easy to compute the distance between current and destination nodes as the sum of the offsets in all of the dimensions. These routing algorithm route packets by crossing dimensions in strictly increasing (or decreasing) order, reducing to zero the offset in one dimension before routing in the next one. This type of routing is known as *dimension-ordered routing* [172].

For *n*-dimensional meshes and hypercubes, dimension-ordered routing produces deadlock-free routing algorithms. These algorithms are very popular and are given several names, like *XY* routing (for 2D meshes) [190] or e-cube (for hypercubes) [44]. Figure 2.10 shows an example of dimension-order e-cube routing. The digits of the destination address, interpreted as a radix-*k* number, are used one at a time to direct the routing. Each digit is used to select a node in a given dimension.

2.4.2 Oblivious Routing

Oblivious routing, where packets are routed without regard for the state of the network, is simple to implement and analyze. 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 trade-off with oblivious routing is between locality and load balance. Load can be balanced for any traffic pattern on almost any topology using *Valiant's algorithm* [216], in which a packet sent from *x* to *y* is first sent from *x* to a randomly chosen intermediate terminal node *z* and then from *z* to *y*. An arbitrary routing algorithm can be used for each of two phases, but in general a routing algorithm that balances load under uniform traffic works best. So, for orthogonal networks, dimension-order routing is an appropriate choice.

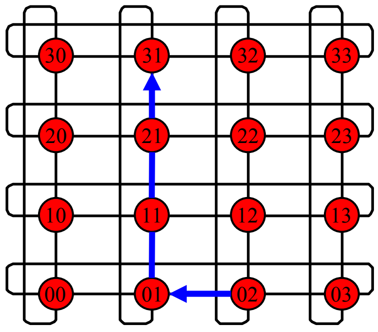


Figure 2.10. An example of dimension-ordered routing in 4-ary 2-cube (4×4 torus). A packet is routed from node 02 to node 31 by first routing in the *x* dimension and then in *y* dimension.

However, this load balance comes at the expense of destroying any traffic pattern [205].

Figure 2.11 shows an example of using Valiant's algorithm to deliver a packet from node *x* = 00 to node *y* = 12. The packet is routed via randomly-selected intermediate node *z* = 21. In the first phase, the packet employs dimension-order routing to travel from *x* to *z*, taking 3 hops as shown in the blue bold arrow. In the second phase, the packet routes from *z* to *y* taking an additional 2 hops. Randomized routing takes 5 hops to reach the destination that could have been realized in 3 hops by a minimal routing algorithm.

Minimal oblivious routing attempts to achieve the load balance of randomized routing without giving up the locality by restricting routes to be minimal (shortest). While a non-minimal oblivious routing algorithm can exploit any path in R'_{xy} to route a packet form *x* to *y*, minimal oblivious routing restrict its choice to paths in $R_{x,y}$.

A minimal version of Valiant's algorithm [169] can be implemented on *k*-ary *n*-cube topologies by restricting the intermediate node *z* to lie in the minimal *quadrant* between *x* and *y*. The minimal quadrant is the smallest *n*-dimensional subnetwork that contains *x* and *y* as corner nodes. Once the minimal quadrant has been identified, an intermediate node *z* is selected from within the quadrant. The packet is then routed from *x* to *z* and then from *z* to *y* using e-cube routing.

Figure 2.12 shows minimal oblivious routing from *x* = 00 to *y* = 12. The route is restricted to remain within the minimum quadrant (shown shaded). There are ten possible routes, corresponding to the six possible intermediate nodes (yellow nodes), with *x*-first routing or y-first routing.

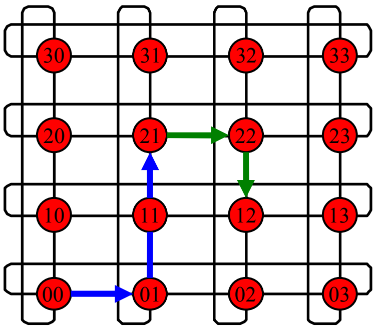


Figure 2.11. An example of randomized routing (Valiant's algorithm) on 4-ary 2-cube (4×4 torus). The message is routed to a randomly chosen node before it is transmitted to the destination.

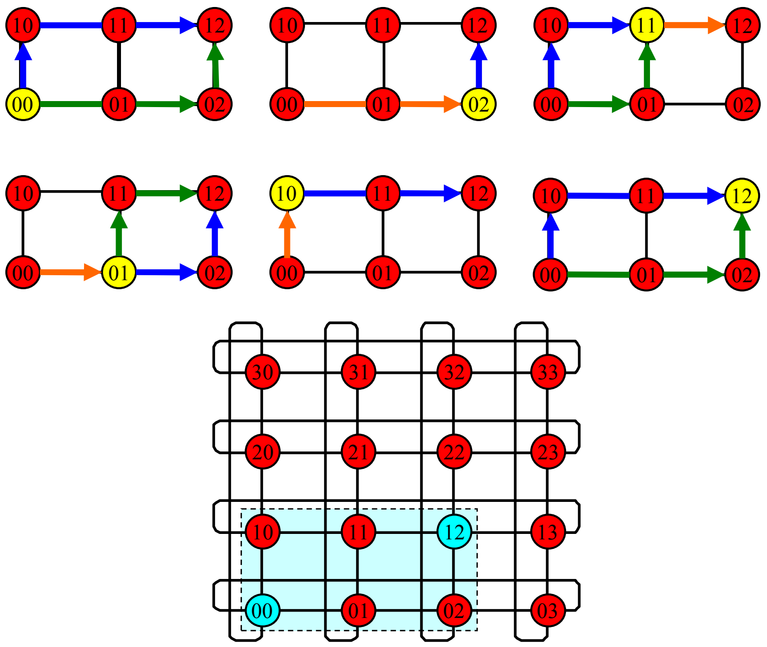


Figure 2.12. An example of minimal oblivious routing on 4-ary 2-cube (4×4 torus) showing minimal quadrant and six possible intermediate nodes.

2.4.3 Adaptive Routing

An *adaptive routing* algorithm uses information about the network state, typically queue occupancies, to select among alternative paths to deliver a packet and therefore should theoretically outperform an oblivious routing algorithm with no available information about network state. In practice, however, many adaptive routing algorithms give poor worst-case performance. This is largely due to the local nature of most practical adaptive routing algorithms [61].

Almost all adaptive routers use flit-based or packet-based flow control and use the state of the flit or packet queues at the present node to estimate congestion on local links. They have no direct information on the state of links elsewhere in the network. On the other hand, routers are able to indirectly sense congestion elsewhere in the network through *backpressure* [51]. When the queues on one node fill up, a backpressure signal stops transmission from the preceding node and hence cause the queues on that node to fill as well. Backpressure propagates backward through the network in the direction opposite traffic flow. However, backpressure propagates only in the presence of traffic routing into the congestion. In the absence of traffic, there is no propagation of backpressure and hence no information on remote congestion.

A minimal adaptive routing algorithm chooses among the minimum (shortest) routes from source x to destination y, using information about the network state in making the routing decision at each hop. At each hop, a routing function identifies which output channels of the current node will move the current packet to its destination. Network state, typically queue length, is then used to select one of the output channels to the next hop.

Minimal adaptive routing is good at locally balancing channel load, but poor at a global load balance. The route from 00 to 21 in Figure 2.13 illustrates how local congestion, channel (10, 20) is avoided by adaptive routing. As with any minimal routing algorithm, minimal adaptive routing algorithms are unable to avoid congestion for a source-destination pair with no minimal path diversity ($|R_{x,y}| = 1$). This situation is illustrated in the route from 02 to 32 in Figure 2.13. We can see below how non-minimal adaptive routing avoids such bottlenecks.

With non-minimal, or fully adaptive, routing, we no longer restrict packets to travel along the shortest path to the destination. Packet can be directed over channels that increase the distance from the destination to avoid a congested or failed channel. For example, Figure 2.14 shows how adaptive routing can avoid congestion on the route from 02 to 32 from Figure 2.13. At node 12, the packet is directed to node 13, increasing the distance to the destination from 2 to 3 hops, to avoid a congested channel (12, 22). Directing a packet along the channel which increases distance is often called *misrouting* [123].

A typical fully adaptive routing algorithm gives priority to an output channel which does not increase the path, so packet are routed toward the destination in the absence of congestion, but allows misrouting to increase path diversity.

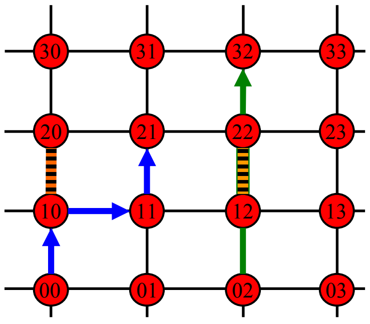


Figure 2.13. Two examples of minimal adaptive routing behavior under congestion.

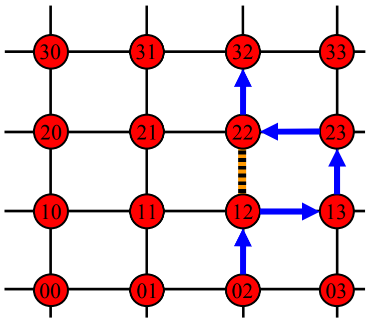


Figure 2.14. An example of fully adaptive routing under congestion.

While fully adaptive routing provides more additional path diversity it can lead to *livelock* unless measures are taken to guarantee progress. Livelock occurs when a packet travels indefinitely in the network without ever reaching its destination (travels along a closed loop) [204].

2.4.4 Deadlock and Livelock

Deadlock occurs in interconnection networks when a group of *agents*, usually packets, are unable to make progress because they are waiting for one another to release resources, usually buffers or channels [37]. If a sequence of waiting agents forms a cycle, the network is deadlocked. As a simple example, consider the situation shown in Figure 2.15. There are four diagonal communications (cyclic shift over one neighbor) in the figure. The packets cannot advance;

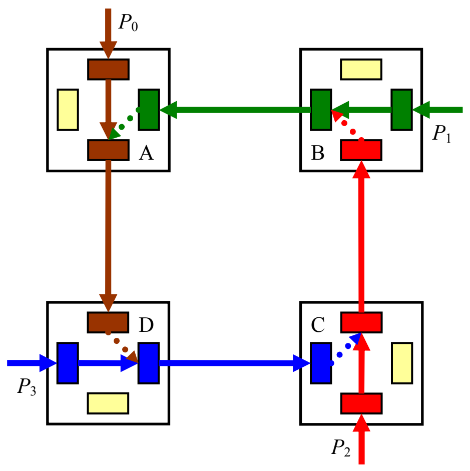


Figure 2.15. Deadlock in communication with four transfers waiting for each other.

queues at all outputs channels determined by a routing function are full. However, neither connection can release the channel needed by the other until it completes its transmission. The connections are deadlocked and will remain in this state until some intervention. Deadlock can occur over various resources. In this example, the resource is a physical channel. It can also be a virtual channel or a shared packet buffer.

Deadlock is catastrophic to a network. After a few resources have been occupied by the deadlocked packet, other packets block these resources, paralyzing network operation. In order to prevent this situation, a network must use either deadlock avoidance (methods that guarantee that a network cannot deadlock) or deadlock recovery (in which deadlock is detected and corrected) [105]. 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.

Unlike deadlock, *livelocked* packets continue to move through the network, but never reach their destination [204]. This is primarily a concern for non-minimal routing algorithms that can misroute packets. If there is no guarantee on the maximum number of times a packet can be misrouted, the packet can remain in the network indefinitely. Dropping flow control techniques can also cause livelock. If a packet is dropped every time it re-enters the network, it can never reach its destination. Figure 2.16 illustrates an example of livelock. A packet from 00 to 30 encounters congestion at 20 and is misrouted to 21, where it encounters more congestion and is misrouted again to 11. This starts a cycle where the packet takes two steps forward from 11 to 20, followed by two steps back, from 20 to 11.

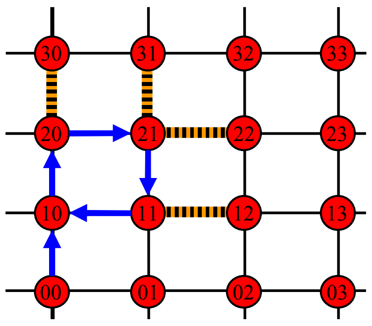


Figure 2.16. An example of livelock caused by misrouting and congested links.

3

Collective Communications in Wormhole Networks

Besides pair-wise communications (unicasts), we often find certain communication patterns in many parallel algorithms which are regular in time, space, or in both time and space; by space we understand spatial distribution of processes on processors. These communication patterns that involve global data movement and global control are known as *collective communications* (CCs) as many processes are collectively involved in performing such operations. As indicated in [54], many scientific applications exhibit the need for such communication patterns, and providing collective communication services can simplify the programming of multicomputers. Such operations include, for example, replication, reduction, segment scan and permutation, etc. Data movement operations are often applied to different dimensions of data arrays.

Collective communication [129] is often also the most efficient way to carry out a communication operation on a parallel computer. The reason is that if many nodes need to communicate with other nodes at the same point in the algorithm, a specialized communication operation in which the nodes cooperatively participate in the communication operation clearly has the opportunity for using the communication network more effectively, thus giving higher performance. Such specialized collective communication operations can and should thus be written so that they cause as little overhead as possible.

In this work, we have focused only on wormhole (WH) interconnection networks for their advantages like distance insensibility, small latency and buffer requirements which predestined them to high-speed system on chip (SoC) and high-performance multicomputers.

3.1 Model of Communication

Performance of CCs is closely related to their time complexity. The simplest time model of point-to-point communication in direct WH networks takes the communication time composed of a fixed start-up time $s_{tart\text{-}up}$ at the beginning (SW and HW overhead), a serialization delay – transfer time of m message units (words or bytes), and of a component that is a function of distance h (the number of channels on the route or hops a message has to do), see section 2.3.5:

$$t_{wormhole} = h(t_r + t_s + t_w) + \max(t_s, t_w)\left\lceil \frac{m}{W} \right\rceil. \tag{3.1}$$

The dependence on h is rather small, so that WH switching is considered distance-insensitive, $h(t_r + t_s + t_w) \approx 0$. Possible synchronization overhead involved in this communication step, be it hardware or software-based, should be included in the start-up time $s_{tart\text{-}up}$. For simplicity, we have assumed no contention (and therefore congestion, too) for channels and no associated delays (t_w).

We will also assume that the CC in WH networks proceeds in synchronized steps. In one step of CC, a set of simultaneous packet transfers takes place along complete disjointed paths between source-destination node pairs. If the source and destination nodes are not adjacent, the messages go via some intermediate nodes, but processors in these nodes are not aware of it; the messages are routed automatically by the routers attached to the processors.

Complexity of collective communication will be determined in terms of the *number of communication steps* (time slots) or equivalently by the number of "start-ups". There are two figures - theoretical lower bound $\tau_{CC}(G)$ and upper bound $\tau^{CC}(G)$ that can be achieved in a real application. These figures of merit do not take into account the message length or its variations from one step to another. For example, let a simple CC on the 8-node bidirectional ring requires $s = 8$ communication steps. Provided that a message has $m = 1024$ byte, time per byte $t_1 = 0.5$ns and start-up time $s_{tart\text{-}up} = 1\mu$s (typical NoC parameters from [82]) and by neglecting insignificant parameters, we get the total time:

$$\tau_{AAS}(ring) = s(t_{startup} + mt_1) = 8 \cdot (1000 + 1024 \cdot 0.5) = 12.1\mu s\,. \tag{3.2}$$

The *port model* of the SoC/NoC defines the number k of CPU ports that can be engaged in communication simultaneously. This means that there are $2k$ internal unidirectional (DMA) channels, k input and k output channels, connecting each local processor to its router that can transfer data simultaneously. Always $k \leq d$, where d is a node degree; a *one-port* model ($k = 1$), and an *all-port* router model (k=d) are most frequently used [215].

Figure 3.1 shows a one-port and an all-port router in a 3-regular network (e.g. 3D hypercube). In a one-port system, a node has to transmit (and/or receive) messages sequentially. Architectures with multiple ports alleviate this bottleneck. In an all-port router, every external channel has a corresponding port. The port model is important in designing collective operations as it determines the number of required start-ups and thus the CC performance.

Moreover, the CC performance is influenced by the fact on whether or not the nodes can combine/extract partial messages with negligible overhead (*combining model*) or can only re-transmit/consume original messages (*non-*

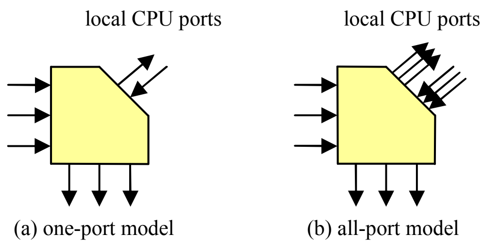


Figure 3.1. Port models for 3-regular networks.

combining model) [152]. Finally, the lower bound on number of steps $\tau_{CC}(G)$ depends on a link type and we have to distinguish between unidirectional (*simplex*) links and bi-directional (*half-duplex* HD, *full-duplex* FD) links. Typically $\tau_{CC}(G)$ will be twice as large for HD links as for the FD ones. Later on we will consider only FD channels.

Summarized, lower bounds $\tau_{CC}(G)$ for the network graph G depend on node degree d, port model k, the number of terminal nodes P, bisection width B_C, cumulative inter-nodes distance Σ, and communication patterns (see following subsections), as shown in Table 3.1. Let's note, the lower bound cannot be exactly derived for irregular topologies. It follows from inconstant node degree d. The lower bound can only be estimated taking into account the lowest and highest node degree in such a topology [111], [212] .

3.2 Classification of Collective Communications

Collective communications involve communication among subsets or among all processors. Provided that there is 1:1 mapping between processors (terminal nodes) and processes, we can equivalently talk about communicating process groups. Generally we have two process groups: T - the subset of transmitters (senders) and R - the subset of receivers. The subsets T and R can be overlapping and can be as large as the full set of P processes ($P = |V|$). We can distinguish three classes of CCs:

(1) $T \cap R = \emptyset$, non-overlapping sets of processes

(a) One-to-All, $|T| = 1$, $|R| = P - 1$, e.g. One-to-All Broadcast (OAB) or One-to-All Scatter (OAS).

(b) One-to-Many, $|T| = 1$, $|R| < P - 1$, e.g. Multicast (MC).

(c) All-to-One, $|T| = P - 1$, $|R| = 1$, e.g. All-to-One Gather (AOG) or All-to-One Reduce (AOR).

(d) Many-to-Many, $|T| = M$, $|R| = N$; $M, N < P$, e.g. non-overlapping sets of processes such as Many-to-Many Broadcast (MNB) or Many-to-Many scatter (MNS).

Table 3.1. Lower complexity bounds of selected CCs in terms of communication steps for any regular topology.

CC	WH, *k*-port, FD links, non-combining model
OAB/AOR	$\lceil \log_{k+1} P \rceil = \lceil (\log P)/\log(k+1) \rceil$
AAB/AAR	$\max(\lceil \log_{k+1} P \rceil, \lceil (P-1)/k \rceil)$
OAS/AOG	$\lceil (P-1)/k \rceil$
AAS/AAG	$\max[\lceil (P^2/2+2\Delta)/B_C) \rceil, \lceil \Sigma/(Pd) \rceil]$

(2) $| T \cap R | \geq 1$, Many-to-Many communication with overlapping sets of processes.

(3) $| T \cap R | = P$, All-to-All communications such as permutation, All-to-All Scatter (AAS), Broadcast (AAB), Reduce (AAR), etc.

3.3 One-to-All and All-to-One Communications

In one-to-all communications, one process is identified as a sender (called also the *root* or *initiator*), and all other processes in the group are receivers. There are some variations. In some designs, the root can or cannot be a member of the process group. Also, if the root is a member of the process group, it can or cannot be a receiver. Here we assume that the root is a member of the process group and itself it is also a receiver, however, the transfer is not done by means of message sending, but only by local memory-to-memory copy. There are two distinct services in this category:

- *One-to-All Broadcast*. The same message is delivered from the root to all receivers.
- *One-to-All Scatter*. The root delivers different messages to different receivers. This is also referred to as *personalized broadcast*.

In All-to-One communications, all processes in a process group act as senders and only one process (called the *root*) is identified as the sole receiver. There are again two distinct services:

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

3.3.1 Broadcast and Reduce Communications

OAB (One-to-All Broadcast) [10] is a collective communication pattern where the root (initiator) distributes the same message to all other processes (nodes) in the interconnection network. If only a subset of nodes takes part in communication, we talk about a multicast communication pattern (MC). The basic idea of the OAB communication pattern is shown in Figure 3.2.

A naive way how to perform broadcast is to sequentially send P–1 messages from the root to the other P–1 processes. However, this is inefficient because the root process becomes a bottleneck. Moreover, the communication network is underutilized because only a small number of channels along the paths between k root-destination pairs is used at anyone time. A better broadcast algorithm can be designed based on a technique commonly known as *recursive doubling* [128]. The root process sends the message to other k processes at first. Now, each of these processes can simultaneously distribute the message to other k processes that are still waiting for the message. By continuing this procedure until all the processes have received the data, the message can be broadcast in

$$\tau_{OAB}(G) = \lceil \log_{k+1} P \rceil \tag{3.3}$$

communication steps.

An optimal OAB communication schedule on the bidirectional 8-node ring topology is shown on the left side of the Figure 3.3. There is a corresponding broadcast tree shown on the right side of the figure. The root (node no. 0) is distributing its message to nodes no. 3 and 6 via completely edge (channel) disjointed paths during the first communication step (black solid arrows). Since the broadcast message is the same for all the processes, these three nodes can become subroots for the second step. Consequently, nodes no. 7 and 5 are receiving the message from node no. 6, nodes no. 2 and 4 form node no. 3, and finally node no. 1 from node no. 0 (red dotted arrows).

The AOR (All-to-One Reduce) [72] communication pattern is a complementary operation to OAB. In AOR, each participating process starts with its own message. The messages of the size of m from all processes are combined through an associative operator and accumulated at a single destination process (root) into one message with the same size m. AOR can be performed in the same way as OAB, but in the opposite path direction, and the reversed sequence of communication steps.

These two CC patterns are used in several important parallel algorithms including matrix-vector multiplication, vector inner product, Gaussian elimination, and etc. [214]. Moreover, these patterns have a hardware support in certain networks on chip [120].

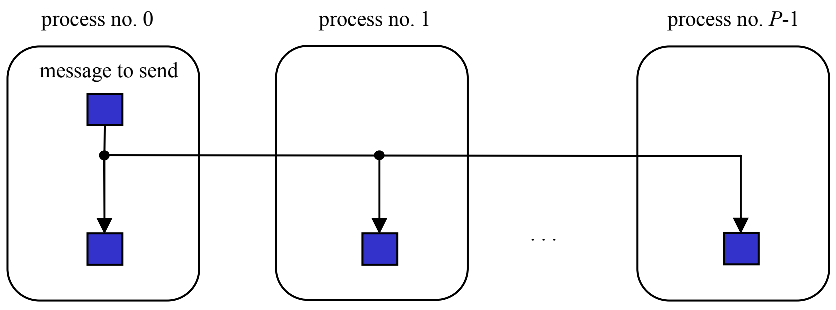


Figure 3.2. The basic idea of one-to-all broadcast communication pattern.

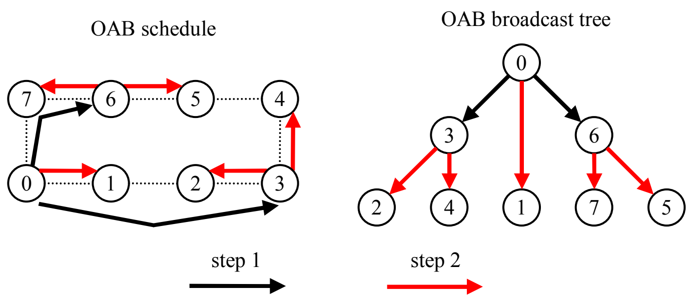


Figure 3.3. An example of optimal OAB schedule on the bidirectional 8-node ring topology and the corresponding broadcast tree.

3.3.2 Scatter and Gather Communications

In the OAS (One-to-All Scatter) [12] collective communication pattern, a single node (root) distributes a unique message to each other node. This operation is also known as one-to-all *personalized communication* [125]. OAS differs from OAB in that the root node starts with $P–1$ unique messages, one destined for each node. Unlike OAB, OAS does not involve any duplication of data (see Figure 3.4).

The lower bound of this CC can be formulated as

$$\tau_{OAS}(G) = \lceil (P-1)/k \rceil \tag{3.4}$$

steps, hence the root process can inject into the network no more than k messages in one step. The optimal schedule of communication consists of $P–1$

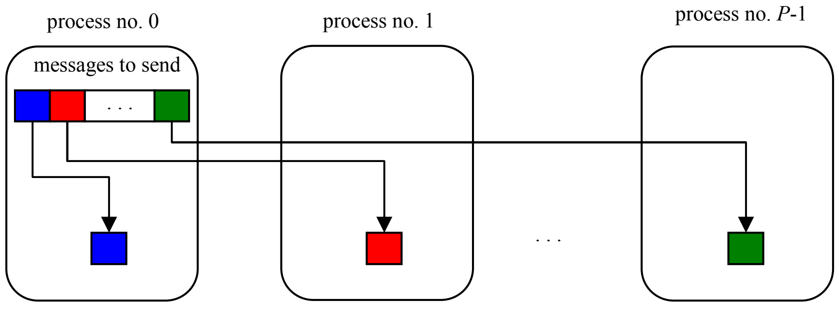


Figure 3.4. The basic idea of one-to-all scatter communication pattern.

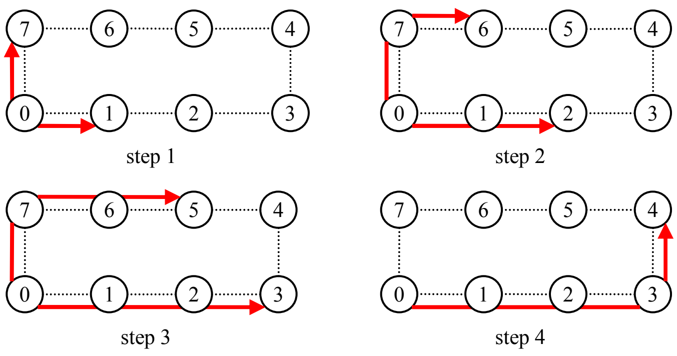


Figure 3.5. An example of optimal OAS schedule on the bidirectional 8-node ring.

point-to-point communications, all originating in the root process, packed into the lowest number of steps in such a way that there are only edge-disjoint paths in a single step.

An optimal OAS communication schedule is displayed in Figure 3.5. In each step, two scattered messages are delivered to two different destination nodes. This OAS communication finishes in four communication steps.

The dual role of OAS is the AOG (All-to-One Gather) communication, in which a single node (root) collects unique messages from all nodes. A gather operation is different from an AOR operation in that it does not involve any combination or reduction of data, so the same schedule as for OAS can be employed [111].

OAS and AOG communication is often used in Divide-and-Conquer techniques to distribute particular tasks and collect their results [198].

3.4 All-to-All Communications

In all-to-all communication, all processes in a process group perform their own one-to-all or all-to-one communication. Thus, each process will receive P messages from P different senders in the process group. There are again two distinct services:

- *All-to-All Broadcast*. All processes perform their own broadcast. This service is also referred to as *gossiping* or *total exchange*.
- *All-to-All Scatter*. All processes perform their own scatter. This service is also referred to as *personalized All-to-All broadcast*, *index*, or *compete exchange*.

3.4.1 All-to-all Broadcast Communication

In AAB (All-to-All Broadcast) [179], all processes have a unique message to share with everyone. Usually, P received messages are concatenated together based on the ID of the senders in receiving nodes. Thus, all processes have the same set of received messages. This communication is illustrated in Figure 3.6.

In the case of AAB communication, since each node has to accept $P-1$ distinct messages, the lower bound can be formulated as

$$\tau_{AAB}(G) = \lceil (P-1)/k \rceil \text{ or } \tau_{AAB}(G) = \lceil \log_{k+1} P \rceil \tag{3.5}$$

communication steps, whichever is greater. The first bound has also been applied to OAS communication and will be greater in the event that an interconnection network has a low node degree, or if only a small number of inner channels k can be employed. On the other hand, the second bound, derived from OAB communication, will outbalance it if a network is highly connected.

An example of optimal AAB communication schedule is shown in Figure 3.7. In the first communication step, all nodes inform some of their neighbors. In the second step, the messages are propagated further through the network to other nodes, in which the messages have not been delivered to yet.

All-to-all broadcast is used, for example, in matrix operations including matrix multiplication and matrix-vector multiplications or barrier synchronizations [214], [72].

3.4.2 All-to-all Scatter Communications

In the AAS (All-to-All Scatter) [171] communication pattern, each P process sends an individual message to each of the $P–1$ partners. This CC is also known as *all-to-all personalized broadcast* or *complete exchange* [57]. Typically, this operation is used for parallel matrix transposition operation. Before AAS, each process keeps one row of the original matrix. During AAS communication all the processes exchange their rows elements with each other. Each process only

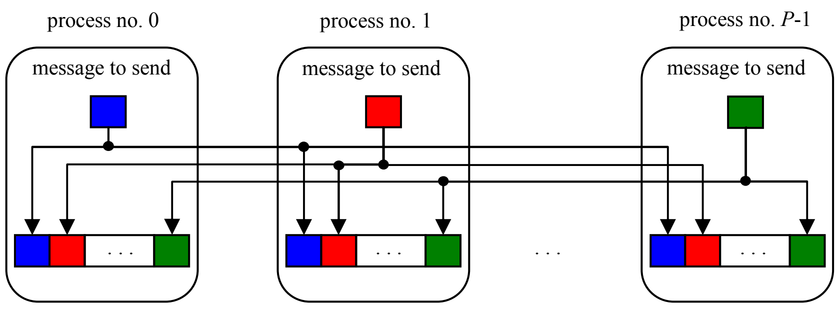


Figure 3.6. The basic idea of all-to-all broadcast communication pattern.

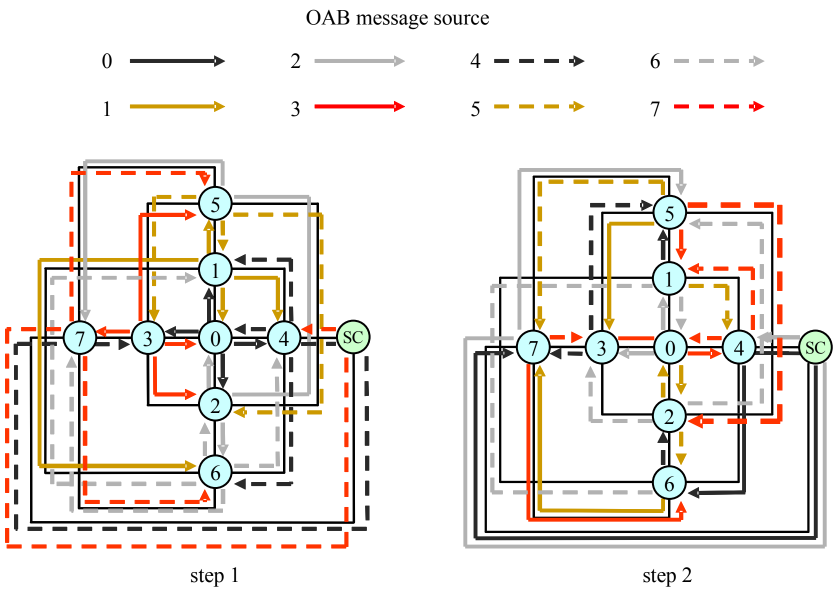


Figure 3.7. An example of optimal AAB schedule on the 8-nodes AMP topology. The AAB communication takes two steps.

obtains one element from each partner. Thus, after the completion of the communication, all processes have a column of the transposed matrix. The basic principle of this communication is shown in Figure 3.8.

A lower bound for AAS can be obtained considering that one half of the messages from each process crosses the bisection and the other half does not. There will be altogether $2(P/2)(P/2)$ such messages in both ways and up to B_C messages in one step, where B_C is the network bisection width. In cases of full

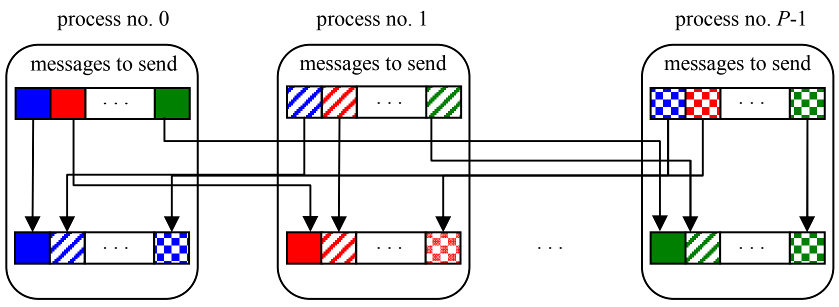


Figure 3.8. The basic idea of all-to-all scatter communication pattern.

duplex links, B_C is taken as (double) the number of (un)directed edges crossed by the bisection. However, some Δ messages originating and terminating in either half of a network cross the bisection as well. This gives the lower bound $(P^2/2 + 2\Delta)/B_C$ communication steps, since Δ messages cross the bisection twice. Another bound that concerns AAS is used to be applied to SF routing only. If Σ denotes the sum of all shortest paths in a graph (from any source to any destination node) and if we can utilize only Pd channels in one step to avoid conflicts, then we cannot schedule AAS in less than $\lceil \Sigma/Pd \rceil$ steps. We have found that for many interconnection networks of interest this latter bound is sharper, even for WH routing.

Finally, we can write the formula for the lower bound of AAS as follows:

$$\tau_{AAS}(G) = \max\left(\frac{\left\lceil \frac{P^2}{2} \right\rceil + 2\Delta}{B_C}, \left\lceil \frac{\sum}{Pd} \right\rceil \right) \tag{3.6}$$

Let us note this form of the lower bound has not been known up to now. It is one of the contributions of the thesis, summarized in the conclusions.

An example of optimal AAS communication schedule is shown in Figure 3.9. The optimal schedule can be executed in three communication steps. Let us note that there is no regularity or symmetry in the schedule. Simply said, it is a composition of point-to-point transfers that satisfy the condition of conflict freedom.

The AAS communication is used in a variety of parallel algorithms such as fast Fourier transform (FFT), matrix transpose, sample sort, and some parallel database join operations [224]

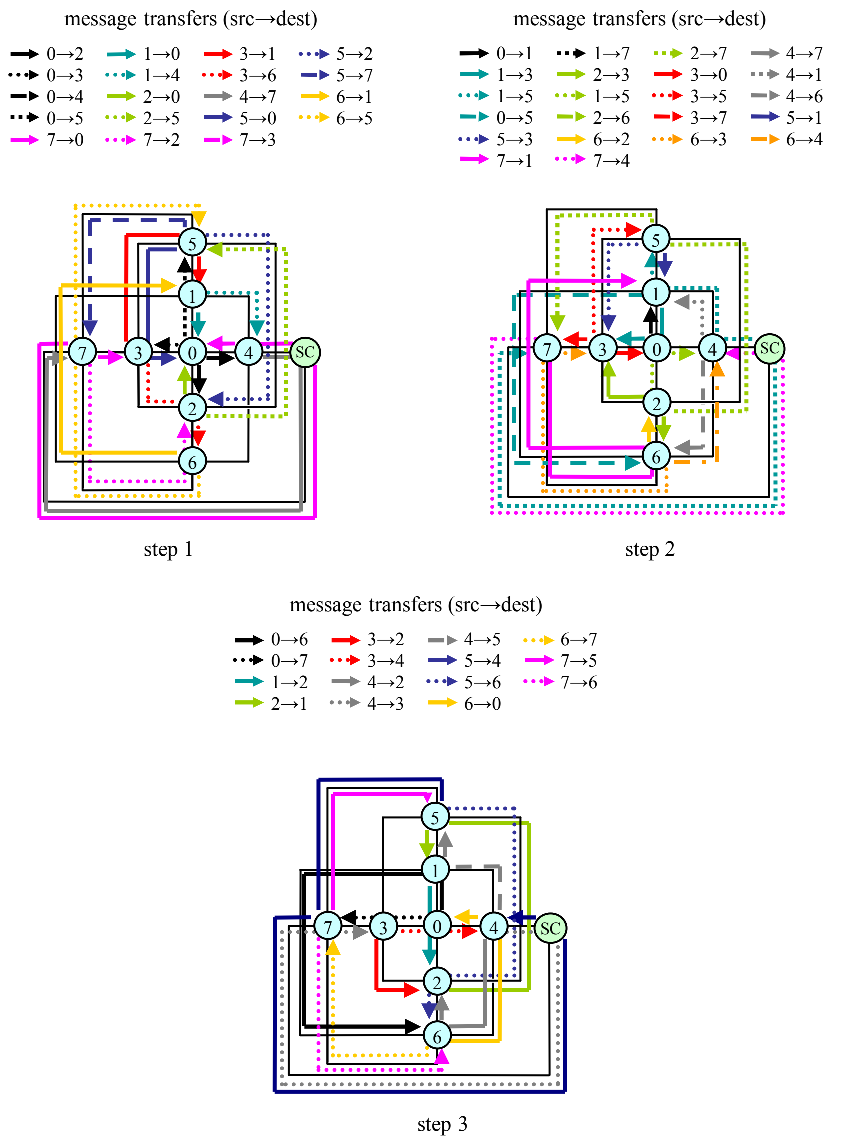


Figure 3.9. An example of optimal AAS schedule on the 8-nodes AMP topology. The AAS communications takes 3 steps

3.5 General Many-to-Many Communications

Many-to-many collective communications are the generalizations of all previously presented CCs. The classes of *M*-to-*N* broadcast and scatter communication that we are going to analyze are represented in Figure 3.10. The sets of transmitting nodes *T* and receiving nodes *R* can be:

(1) overlapping, $| T \cap R | = Q \geq 1$, $| T \cup R | = P$.

(2) some nodes can be only transmitters, $| T \setminus T \cap R | = M - Q$, $| T | = P$.

(3) some nodes can be only receivers $| R \setminus (T \cap R) | = N - Q$, $| T | = P$.

(4) some node can be only receivers, some nodes can be only transmitters, some nodes can act in both function and some nodes can be only switches.

The lower bounds $\tau_{CC}(G)$ for *M*-to-*N* communications are not known, but can be estimated as follows in Figure 3.10.

3.5.1 Overlapping Many-to-Many Communications

First, *M*-to-*N* broadcast is limited by OAB or AAB bound from Table 3.1, whichever is greater:

$$\tau_{MNB} = \max (\lceil \log_{k+1} N \rceil, M/k) \tag{3.7}$$

because some nodes have to absorb *M* (a node is not a transmitter), and not only *M*–1 (a node is also a transmitter) messages as in AAB and some nodes have to distribute *N* messages using *k* ports as in OAB.

Second, *M*-to-*N* scatter communication can be divided into four groups of communication that have related bisection widths available, see Figure 3.10a:

$$\begin{array}{ll} T \setminus (T \cap R) \to (T \cap R) & b_{1,} \\ (T \cap R) \to R \setminus (T \cap R) & b_2, \\ T \setminus (T \cap R) \to R \setminus (T \cap R) & \min (b_1, b_2), \\ (T \cap R) \to (T \cap R) & b_0. \end{array} \tag{3.8}$$

Now the first two groups can proceed simultaneously (overlapped),

$$T \setminus (T \cap R) \to (T \cap R) \,||\, (T \cap R) \to R \setminus (T \cap R) \tag{3.9}$$

and so could the other two:

$$T \setminus (T \cap R) \to R \setminus (T \cap R) \,||\, (T \cap R) \to (T \cap R). \tag{3.10}$$

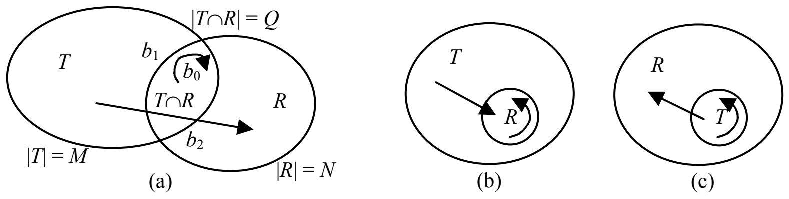


Figure 3.10. General M-to-N communication (a) $T \not\subset R$ and $R \not\subset T$, (b) $R \subset T$, (c) $T \subset R$.

Time for communication specified in (3.9) is thus

$$\tau_1 = \max\left(\left\lceil \frac{(M-Q)Q}{b_1} \right\rceil, \left\lceil \frac{Q(N-Q)}{b_2} \right\rceil\right) \tag{3.11}$$

and for communication described in (3.10) is

$$\tau_2 = \max\left(\left\lceil \frac{(M-Q)(N-Q)}{\min(b_1, b_2)} \right\rceil, \left\lceil \frac{Q(Q-1)}{2b_0} \right\rceil\right). \tag{3.12}$$

The total lower bound is thus

$$\tau_{MNS}(G) = \tau_1 + \tau_2. \tag{3.13}$$

3.5.2 Many-to-Many Communications with *R*⊂*T*

Similarly as in case from Figure 3.10a for broadcasting we have

$$\tau_{PNB}(G) = \max(\lceil \log_{k+1} N \rceil, \lceil (P-1)/k \rceil) \tag{3.14}$$

and for scatter

$$\tau_{\mathrm{PNS}} = \max\left(\left\lceil \frac{(P-N)N}{b_1} \right\rceil, \left\lceil \frac{N^2}{2b_0} \right\rceil\right), \tag{3.15}$$

where b_1 is the number of channels from T to R and b_0 is a bisection width of sub-network R.

3.5.3 Many-to-Many Communications with *T*⊂*R*

A similar reasoning as in case from Figure 3.10b gives

$$\tau_{MPB}(G) = \max(\lceil \log_{k+1} P \rceil, M/k) \tag{3.16}$$

and

$$\tau_{\mathrm{MPS}} = \max\left(\left\lceil \frac{M(P-M)}{b_1} \right\rceil, \left\lceil \frac{M^2}{2b_0} \right\rceil\right), \tag{3.17}$$

where b_1 is the number of channels from T to R and b_0 is a bisection width of sub-network T.

3.5.4 More Complicated Many-to-Many Communications

In cases where $| T \cup R | \leq P$, we cannot use the bisection width any longer. The same situation happens if transmitters and receivers are agglomerated in several clusters separated by clusters of silent nodes.

If $\lambda_{CC}(l)$ is the load of link l in CC (i.e. the number of messages using link l), then the lower bound is given as

$$\tau_{MNX} = \max \lambda_{CC}(l) \tag{3.18}$$

over all links l.

3.5.5 Example Calculation of Lower Bounds

Now, we will try to estimate the lower bound for the broadcast and scatter communication. The spatial distribution of transmitters and receivers on the 4×4 2D mesh is illustrated in Figure 3.11.

There are M = 9 transmitting nodes, N = 11 receiving nodes, and Q = 4 nodes in intersection $T \cap R$. According to eq. (3.7),

$$\tau_{MNB}(G) = \max(\lceil \log_{4+1} 11 \rceil, \lceil 9/4 \rceil) = \max(\lceil 1.48 \rceil, \lceil 2.25 \rceil) = 3 \text{ steps.}$$

The complexity of scatter communication will be estimated by means of bisections $b_1 = 4$, $b_2 = 6$, and $b_0 = 1$. Substituting these parameters into eq. (3.13) we will get

$$\tau_{\mathrm{MNS}}(G) = \tau_1 + \tau_2 = \max(5, 5) + \max(7, 6) = 12 \text{ steps.}$$

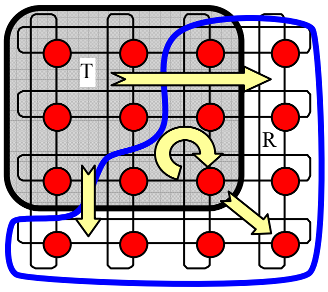


Figure 3.11. Nine-to-eleven broadcast/scatter communication.

4 Scheduling of Wormhole Collective Communications

The design of efficient routing algorithms for collective communication is the key issue in message-passing parallel computers or networks [89], [111], [129]. As we could see in the previous chapter, the lower bound on the time complexity of various CCs can be mathematically derived from the network parameters and the spatial distribution of communicating processes. Unfortunately, for irregular topologies (such as meshes, AMPs, random graphs, etc.) and/or many-to-many CCs only an approximation of the exact lower bound can be obtained because of varying node's degree, and the spatial distribution of processes on the underlying topology. Some hints on how to overcome this problem have been given in section 3.5.

Moreover, as far as this author knows, the concrete forms of congestion-free schedules for particular CC patterns, and for the minimal number of steps (time slots) derived from the lower bounds are unknown even for many common topologies! This is the main motivation behind this work. The goal of this thesis is to design a tool producing (near) optimal communication schedules to an arbitrary topology and arbitrary scatter/broadcast based communication service with an acceptable time complexity.

In recent years, many projects have addressed the design of efficient collective communication algorithms for wormhole-routed systems. Some of them will be briefly described in section 4.2.

Since time complexities of CCs are highly dependent on the underlying topology, many researchers have focused their efforts at optimization of the present topologies and their modification to gain any reduction of complexity of a particular CC or to improve any other network parameters. Recent studies have demonstrated the significant role that communication architecture plays in determining a parallel systems' overall performance. The following subsection briefly presents several interesting network topologies with some special characteristics mainly targeted to newly booming networks on chips (NoC) [98].

4.1 Interconnection Networks

Nowadays, there are many assorted interconnection network topologies intended for the general purpose multiprocessors like clusters of workstations, blade systems, HPC (High Performance Computers), MPP (Massive Parallel Processors), Network on Chips (NoC), etc. Furthermore, completely new networks for specific parallel applications can be created using, for example, combinatorial or evolutionary algorithms.

4.1.1 The Most Popular Interconnection Networks

The simplest interconnection topology often used in parallel systems is a *ring* topology [39]. It has been very popular due to its trivial manufacturing and still good performance for small systems up to dozens of nodes. At the present time, there are many application of this topology mainly in NoC systems like IBM Cell [91] or ATI Ring bus [1] systems.

Another famous networks topology is a multidimensional *hypercube* [170] having many attractive properties inducing regularity, symmetry, small logarithmic diameter, strong connectivity, recursive construction, partitionability, and relatively small link complexity. Efforts to improve some of these properties have lead to the design of many hypercube variants. They include twisted cubes [48], cube connected cycles [162], generalized hypercubes [20], banyan-hypercubes [227], and binary orthogonal multiprocessors [90].

Very popular networks are also based on a *mesh* [76], and *torus* [23] topology. Mesh topology has gained more consideration by designers due to its simplicity of manufacturing where only local and very short links are necessary to interconnect the network. The main problem with the mesh topology consists in its long diameter that results in the communication latency and a lack of regularity and symmetry. Torus topology has been proposed to reduce the latency of the mesh and to keep its simplicity. The only difference between torus and mesh topology is that the switches on the edges are connected to the switches on the opposite edges through wrap-around channels. By these wrap-around channels the torus becomes node symmetrical. Although the torus architecture reduces the network diameter, the long wrap-around connections can result in excessive delay. Some variations of the mesh and torus topologies have been discovered which further improve some of their properties (e.g. higher dimensional meshes and torus (3-D mesh) and Midimew [118], etc).

The *Minimal Distance Mesh* with wrap-around links (Midimew topology) reaches the lowest diameter and average distance of any regular and symmetric $d = 4$ graph and can be obtained from 2D-torus by changing the arrangement of wrap-around links (see Figure 4.1). Reduced values of D and d_a translate to improved performance in synthetic as well as real loads [1].

The increase the efficiency of communication networks often leads to a decrease in its reliability. Networks with *multistage topologies* can offer a small diameter, large bisection width and a large number of redundant paths,

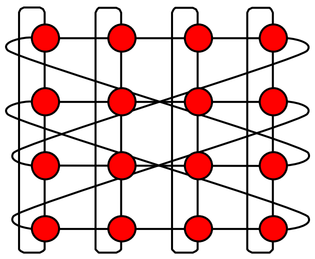


Figure 4.1. An example of a Midimew topology.

but they are hard to construct because of the complex wiring structure. Examples of the most popular multistage networks, which have been widely used recently, include the Butterfly network [213], the Clos network [34] or Benes network [16].

4.1.2 Optimal Diameter-Degree Networks

The *Optimal Diameter-Degree* (ODD) networks [140] are a highly promising area of interconnection network, however, not very deeply examined. The size of an ODD network is equal or close to upper bounds known for the given node degree and diameter. Simply said, as many nodes *P* as possible are connected by a regular network with a uniform node degree *d* (a *d*-regular network), with an inter-node distance up to *D*. Such systems are more compact than others and can support faster communications too.

The upper bounds on the number of *P* nodes with degree $d > 2$ that can be connected into an undirected graph, shortly graphs (4.1), or directed graph, shortly digraphs (4.2), and with diameter $D \geq 1$ are known as *Moore bounds* [134]:

$$P \leq 1 + d + d(d-1) + ... + d(d-1)^{D-1} = \frac{d(d-1)^D - 2}{d-2}, \tag{4.1}$$

$$P \leq 1 + d + d^2 + ... + d^D = \frac{d^{D+1} - 1}{d-1}. \tag{4.2}$$

A regular graph of degree *d* and diameter *D* whose number of vertices equals the above upper bound (4.1) is a Moore graph. If we exclude fully connected graphs with $D = 1$, there exists only a few such graphs:

- D=2: $d = 3$, $P = 10$ (Petersen graph) [88]
- D=2: $d = 7$, $P = 50$ (Hoffman-Singleton graph) [17]

and no others with the possible exception d = 57 (which is still undecided). There are no Moore graphs with $D \geq 3$ and no Moore digraphs either (disregarding trivial cases $D = 1$ or $d = 1$).

Whereas all above Moore graphs have the length of the shortest cycle (girth) five, the even girth (6, 8 and 12 only) can also be considered which leads to the worse upper bound

$$P \leq 2\sum_{i=0}^{D-1}(d-1)^i \tag{4.3}$$

and to generalized Moore graphs. A few examples follow:

- D= 2: $d = 3$, $P = 6$ (utility graph) [221]
- D= 3: $d = 3$, $P = 14$ (Heawood graph) [220]
- D= 4: $d = 3$, $P = 30$ (Levi graph) [121]

It is an open problem if there are infinitely many generalized Moore graphs of each degree.

Since there are not many known Moore graphs, it is of great interest to find graphs which for a given diameter D and maximum degree d have a number of vertices as close as possible to the Moore bound. The largest known graphs and digraphs are presented in [140] and [134]. The more systematic approach has been used to design "almost" Moore graphs that miss the upper bound by a small number [134] or whose number of nodes approximates the upper bounds asymptotically. The best networks in the latter case are based on Kautz digraphs with $P = d^D + d^{D-1}$ nodes. In Table 4.1, all three upper bounds (4.1) - (4.3), the largest known graphs, Kautz digraphs, and generalized Moore graphs, all with degree $d = 3$ are stated. It turns out that the largest known digraphs are Kautz digraphs and that digraphs (4.2) are potentially much larger than graphs (4.1).

The most interesting OOD topologies recently used in real multicomputers, are shown in Figure 4.2. They are Heawood graph with fourteen nodes, Petersen graph with ten nodes and Kautz digraph with twelve nodes.

4.1.3 Novel Network Architectures

In order to alleviate known bottlenecks of commonly used networks and the nescience of ODD networks for an arbitrary size, degree and diameter values, many engineers tried to design general-purpose networks by analyzing graph theory or using combinatorial optimization methods [28].

One of the novel architectures with an arbitrary size is the *K-Ring* topology [112]. Intuitively, the K-Ring topology can be seen as a graph built using $K > 0$ rings where each ring goes over all the nodes in a different order. The value K is called the dimension. For a given size and a given dimension there are many different K-Rings.

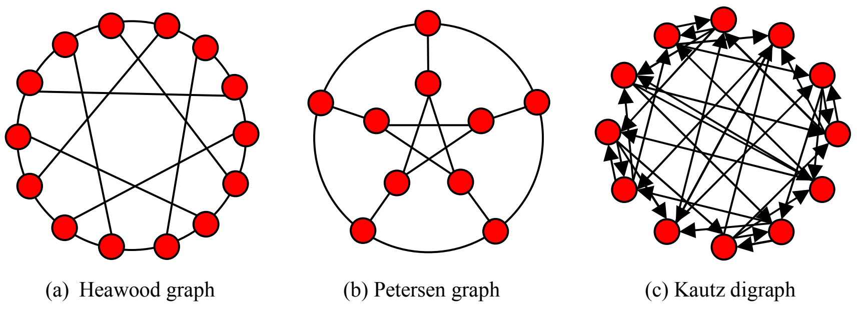


Figure 4.2. Interesting examples of almost optimal diameter-degree networks.

Table 4.1. The size of graphs and digraphs with degree d = 3 and diameter D. (* Denotes a generalized Moore graph; digraphs are in bold)

D	**2**	**3**	**4**	**5**	**6**	**7**
Bound (4.1)	10	22	46	94	190	382
Bound (4.2)	**13**	**40**	**121**	**364**	**1093**	**3280**
Bound (4.3)	6	14	30	62	126	254
Largest known	10	20	38	70	132	192
Kautz	**12**	**36**	**108**	**324**	**972**	**2916**
Moore	10	14*	30*	-	-	-

K-Rings are still not well known. There is especially almost no analytic formula for the computing of their characteristics. Because of their flexibility it is possible to build a lot of apparently different K-Rings, but we are still not able to determine which of them correspond to the same graphs. K-Rings are Caley graphs [154] and they belong to the General Corodal Rings (GCR) [19]. Figure 4.3 shows two examples of K-Rings with dimension K=2 and size 15.

Spidergon (also referred to as octagon) is also a novel on-chip interconnection network suitable for the aggressive on-chip communication demands of SoCs in several application domains and also for networking SoCs [102] (see Figure 4.4a). In this architecture, eight nodes are connected by an octagonal ring and four diagonal links. The distance between any two node processors are no more than two hops (through one intermediate node) within the local ring. The spidergon network is scalable. If one node processor is used as the bridge node, more spidergon networks can be cascaded together, as shown in Figure 4.4b. This scaling strategy based on bridge nodes connecting adjacent Spidergon has a drawback of a very low bisection width B_C and therefore a poor performance in all-to-all traffic. Another scaling strategy extends the spidergon to the multidimensional space by linking corresponding nodes of several spidergons, or simply increasing the ring length [176]. The last way how to scale Spidergon is to add more nodes along the ring and connect them with diagonal links.

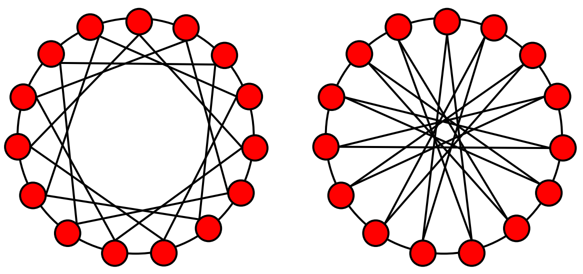


Figure 4.3. An Example of two different K-Rings of size 15 and $K = 2$.

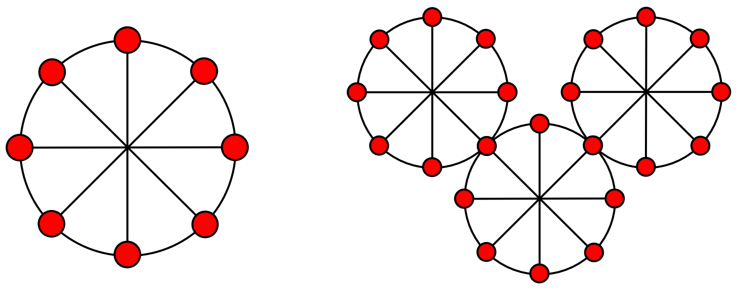

(a) general spidergon network (b) scalability of spidergon network

Figure 4.4. Two examples of spidergon networks

The *AMP topology* is a result of genetic graph optimization [28]. A Minimum Path (AMP) [29] configuration is constructed so that the network diameter and the average inter-node distance is minimized. This principle is maintained even at the expense of the loss of regularity in the system. The AMP networks have been found for node count P = 5, 8, 12, 13, 14, 32, 36, 40, 53, 64, 128, 256. The 8-node AMP network topology is depicted in Figure 4.5. The SC node denotes a system controller (host computer) that sends input data to processing nodes and collects results. Each processing node can communicate simultaneously on four bi-directional full duplex links.

The new area in interconnection network is represented by polymorphic on-chip networks [106]. These networks can be configured prior to or during the application runtime, in order to have the topology and buffering of arbitrary network designs. Simply said, if the structure of communication is known, the network is possible to reconfigure in runtime in order to afford as a good condition as the upcoming communication schedule can exploit.

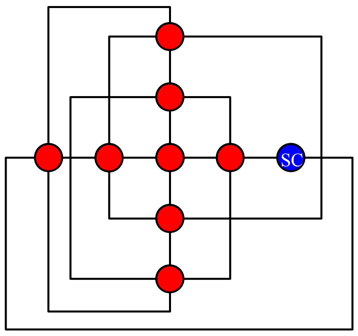


Figure 4.5. The 8-node AMP topology.

4.2 Scheduling of Collective Communications

Over the last several years, many methods have been described for implementing various CC patterns. Among the most common algorithms belong binary/binomial and other tree-based algorithms and the pairwise exchange-based algorithms [1], [120], [174] such as recursive-doubling and halving. Many topology-specific collective communication algorithms have been discussed. Paper [201] focuses on SMP-based cluster hierarchy to implement collective communication operations. Paper [174] advocates automatically tuned CCs based on network topology. The primary focus in these efforts is on other aspects (deadlock freedom, simple routing) of the implementation rather than exploitation the network concurrency at the algorithmic level. A few other papers have used the network concurrency to optimize CCs. Another possible direction is to look at generalizations of routing problems, such as scheduling problems. There is a wide range of scheduling models. One example is a shop scheduling [183] or exploitation of the concurrency in modern networks [200].

Although many collective communication algorithms have been designed for specific topologies, several researchers have taken a fundamentally different approach. Their idea is to design some families of parameterized algorithms that can be tuned to perform well on different architectures under various system conditions. For example, the postal model [15] (similar to sending a batch of letters through the postal service at one time) demonstrates the ability for a sending node to transmit multiple messages before the receiving node has received the first one. But for short messages, predominating in cache-coherent NoC, this model is unusable due to a dominating overhead of star-up latency over message transportation delay.

If we restrict the communication model only to a non-combining (also referred to as direct) model, we can decompose any collective communication into a set *COMM* of pair-wise communications (transfers, messages, paths)

$$x_{i,j} = \{c_1, c_2, c_3, \ldots, c_L\}, x_{i,j} \in \text{COMM}, \quad (4.4)$$

where c_{i} are unidirectional channels along the path from the source to the destination node. In the following sections, basic models for the systematic design of CCs in multiport wormhole-routed networks will be described. Moreover, some of the best known algorithms intended for particular communication patterns will be presented.

4.2.1 OAS Scheduling

For both OAS and AAS, the set of message transfers *COMM* can be determined in advance. This pretty feature enables to pack *COMM* into a minimum number of groups so that there is no conflict within a group. If this condition is satisfied, the groups are compatible. Compatibility relation γ on set *COMM* can thus be defined:

$$x_{i,j}\, \gamma\, x_{k,l} \equiv \exists! c_e \, \{c_e \in x_{i,j} \text{ and } c_e \in x_{k,l}\}, x_{i,j}, x_{k,l} \in \text{COMM}. \quad (4.5)$$

This relation defines a cover of the *COMM set* by maximum-size compatibility classes. All the message transfers in one compatibility class can realize transmission simultaneously, and therefore each such a group can be scheduled in one communication step. Obviously we would like to find a minimum number of compatibility classes still covering set *COMM*. The final step is to transform this minimum cover of *COMM* to blocks by eliminating messages transfers in more than one class and possibly simultaneously balancing the size of classes. Notice that a message transfer can travel along just one of many possible paths between source and destination pair.

The most frequently implemented OAS execution scheme is based on the *Sequential Tree* (ST), also known as the separate addressing algorithm [147], [13]. In this algorithm, the root sends a separate message to each of the other nodes participating in the scatter. The deadlock-free condition is ensured using minimal spanning tree of the underlying topology.

4.2.2 AAS Scheduling

In contrast to OAS, there is not much known about optimal AAS schedules for non-combination model, although it has been extensively studied so far. From the theoretical point of view, exact solutions of the AAS problem can be obtained by analyzing of compatibility classes as in the previous case. This task can be performed by *MILP method* (Mixed Integer Linear Programming) [141], but very long solutions are required for network sizes of practical interest. The

AAS scheduling can also be formulated as a graph colouring problem [99], [55]. Elements of *COMM* can be represented by graph nodes and incompatibility relation (two message transfers sharing a channel) by graph edges. A minimum number of colours needed to colour the graph gives the optimum number of compatibility classes (communication steps); nodes with the same colour belong to one compatibility class. MILP, as well as exact or heuristic graph colouring, yield only a suboptimal solution. The reason is the existence of multiple minimum paths for most source-destination pairs; it is not clear which minimum paths should be selected for message transfers (the set *COMM*). On the other hand, inclusion of all of them could produce more compatibility classes than necessary, aside from complex removal of redundant elements. Another approach, recursive division of set *COMM* described in [55], is supposed to be exact, but can be used only for a one-port model.

From the practical point of view, some research is developed on the cluster-based systems [192]. However, most of this research is designed to handle the topological constraints of the underlying networks, such as tori [116], [142], [208], meshes [191], [189], hypercubes [179], and multistage networks [30].

Young-Joo and Shin [189] proposed an interesting method with linear time complexity but only for combining model and targeted only to tori and meshes. Paper [57] extends the method and proposes the conceptually simple and symmetrical AAS algorithm for every message and every node so that it can be easily implemented.

One of few direct algorithms (non-combining model) targeted to hypercubes [180] or meshes [196] achieving the lower bound on time complexity is a *complete exchange* algorithm. Its idea is very simple: there are $P-1$ communication step; in each step $s = 1, 2, \ldots, P-1$, node i sends message $M_{i,j}$ to node $j = i$ xor k and receives message $M_{j,i}$ from node j. Thus, all communication occurs directly between the original source node and the final destination node of that message. For communication between non-adjacent nodes only the routers are required to relay the message at the intermediate nodes on the path.

Figure 4.6 shows the communication steps of the algorithm for the 2D (2×4) mesh using dimension ordered routing. Notice that in steps 2, 3, 6 and 7, messages transfers require some communication channels simultaneously, leading to channel contention! Since contention must not occur in our model of communication for NoC, the proposed algorithm is also inapplicable. Another idea on how to design a fast suboptimal and contention-free algorithm is a sequential combination of one-to-all techniques. Tseng [209] used a *gather-then-scatter* technique and enforced shortest paths in routing messages to achieve an asymptotically optimal number of communication steps. The proposed scheme consists of a sequence of gathering phases followed by a sequence of scattering phases. In the beginning, all nodes will join the communication. After each gathering phase, the blocks are concentrated into a smaller number of nodes. On the contrary, blocs are distributed to more nodes after each scattering phase. At the end, each block is guaranteed to arrive at its destination.

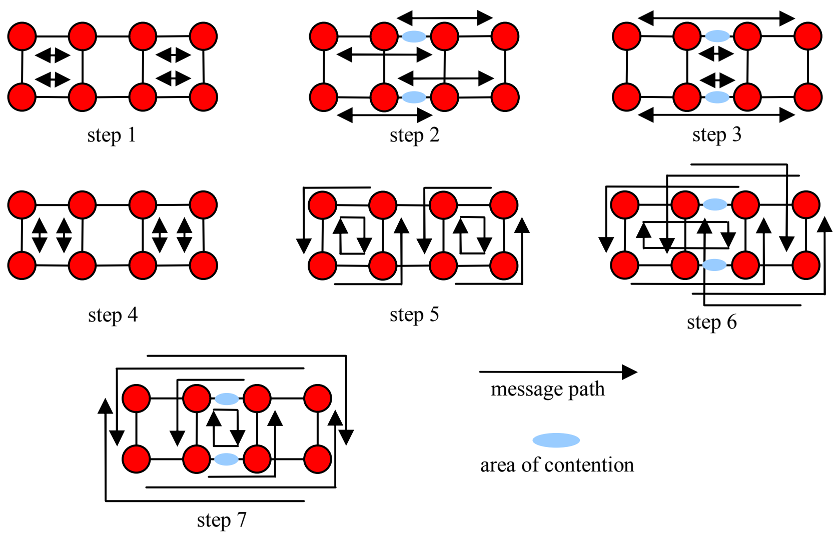


Figure 4.6. Direct pairwise exchange on 2D mesh [196]. The AAS communication takes 7 communication steps, some of which show contention of links.

4.2.3 OAB Scheduling

The problem with broadcasts is that the set *COMM* cannot be determined in advance since the informed nodes can become distributors for further transfers. For this reason, many novel techniques have been developed and used. We only concern with the basic principles on the OAB communication in this subsection.

Dominating sets. The *Extended Dominating Node* (*EDN*) [207] model is based on the notion of dominating sets in graph theory. A *dominating set D* of a graph *G* is a set of vertices in *G* such that every vertex in *G* is either in *D* or is adjacent to at least one vertex in *D* [58]. By exploiting the distance-insensitivity of wormhole routing, the EDN model extends the concept of domination to include sets of nodes reachable in a single message-passing step under a given unicast routing algorithm. The key issue in applying the EDN model to the development of collective operations lies in finding such regular and recursive patterns of dominating nodes that can pass messages to other sets of nodes while avoiding channel contention.

Figure 4.7 illustrates the concept of the node domination on a 2D torus. It presents a set of five dominating nodes (blue colored) for the 5×5 torus [206]. As illustrated with arrows, these five nodes can distribute a message to 20 remaining nodes in a single step by sending it to appropriate neighbors. The basic idea is to build recursively a hierarchy of extended dominating nodes, where level-*i* EDNs are informed from all EDNs of levels $j < i$.

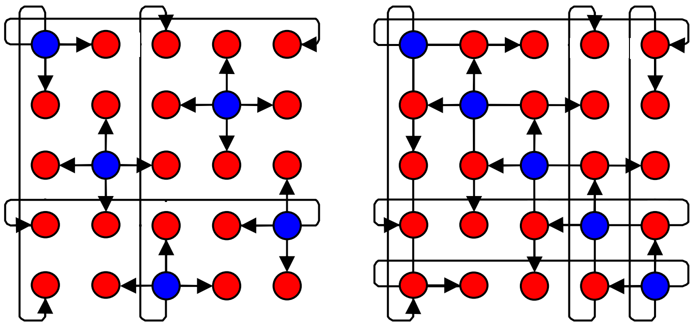


Figure 4.7. An illustration of (a) dominating sets and (b) extended dominating sets in the 5×5 torus. The dominating nodes are shown in blue.

Dimensional broadcast trees. The *dimensional broadcast tree* algorithm was described by Barnett in [10]. It exploits a *recursive doubling* process within every mesh/torus dimension. Figure 4.8 illustrates this algorithm's operation in the 4×4 2D mesh. In each message-passing step, every node holding a copy of the message is responsible for a part of a row or column. The node divides its part in half and sends a copy of the message to the node in the other half that occupies the same relative position. The algorithm thus takes advantage of the pipelining effect of wormhole routing while avoiding channel contention. The idea combining a dimensional broadcast tree and recursive doubling targeted to torus topology was presented in [153], [160].

Double-tree algorithm. The *Double Tree* (DT) broadcast algorithm of McKinley and Trefftz [128] is designed for multiple port wormhole hypercubes. The authors have demonstrated the advantages of this algorithm over the *Spanning Binomial Tree* (SBT) [10] algorithm using a commercial system. The DT algorithm divides the hypercube into two parts by sending a message from the source, say *S*, to its bitwise complement (the opposite node) *S*'. This message is called a diagonal message. In the next step, *S* and *S*' become the roots of partial spanning binomial trees as implied in Figure 4.9. As a result, this algorithm broadcasts a message in $\lceil P/2 \rceil$ routing steps and is more efficient than the SBT algorithm. In addition, it does not require reconstructing the message at every destination like in the case of *Transmission SubGraph* (TSG) [229] that is targeted for combinating hypercubes.

HO-KAO algorithm. The DT algorithm is really approximately twice as fast on real machines as the standard SBT algorithm, but it is optimal only for dimensions $d \leq 6$. An algorithm which is optimal up to a small multiplicative constant for $d \geq 7$ is called *Ho-Kao algorithm* (HKA) [84], [85]. It is based on the following idea: divide recursively *d*-dimensional hypercube into sufficiently small subcubes of nearly equal size and apply DT inside these subcubes.

Figure 4.10 demonstrates this idea for $d = 7$. HKA assumes the e-cube routing by checking the bits left-to-right and uses ascending dimension-simple paths

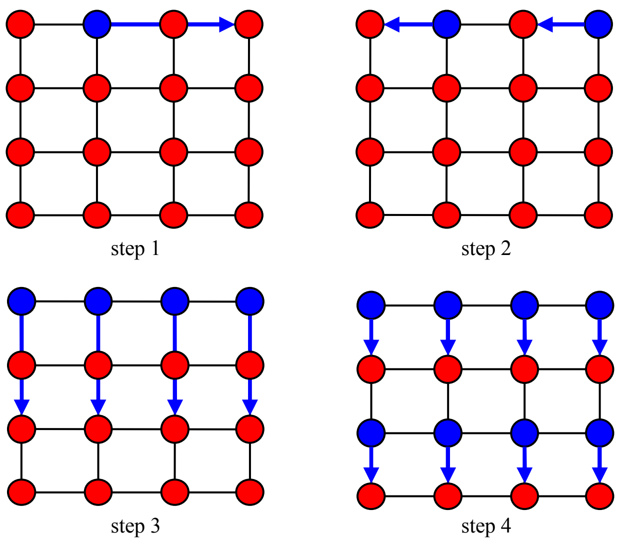


Figure 4.8. A generic schema of dimension ordered recursive doubling used in OAB on one-port WH meshes.

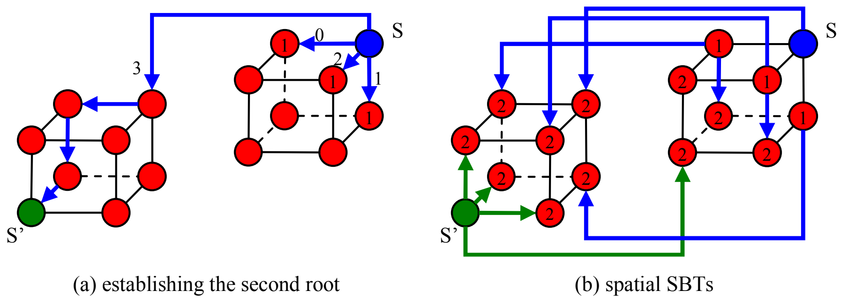


Figure 4.9.An example of 2-round OAB on the all-port 4D WH hypercube.

Given a sequence of dimension numbers $0 \le i_1 <\ldots < i_k \le d - 1$, $k \le d - 1$, an ascending dimension-simple path is any path $u_{0,\ldots,}\ u_k$ of nodes in d-dimensional hypercube so that u_j is obtained from $u_{j\text{-}1}$ by complementing bit i_j. It follows that all d paths $u_0 \rightarrow u_j$ are pairwise node-disjoint if e-cube WH routing is used, so that u_0 can send k packets to all these destinations simultaneously. HKA constructs an ascending dimension-simple path $u_0,\ldots, u_n$ so that d-dimensional cube can be partitioned into $d + 1$ disjoint subcubes S_i of equal or nearly equal size, so that subcube S_i contains u_i.

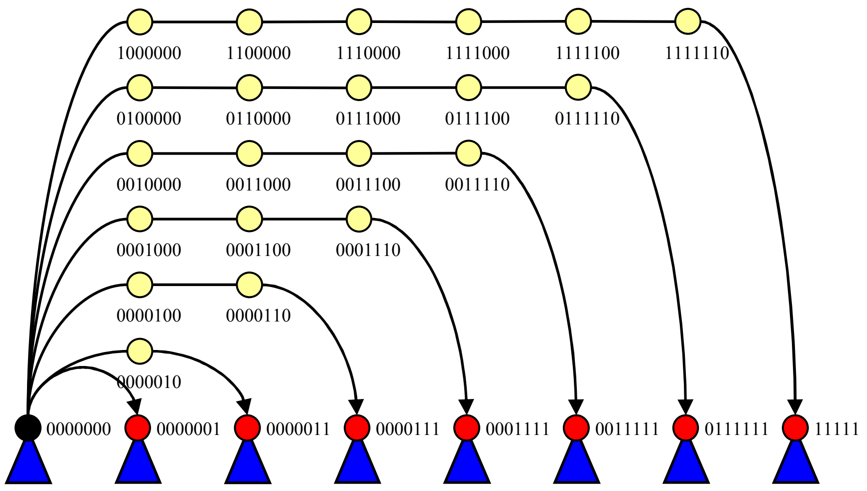


Figure 4.10. The optimal HO-KAO algorithm on the all-port 7D hypercube dividing the hypercube into seven 6D subcubes.

Ho-Kao algorithm finishes in $O(N/\log(N+1))$ wormhole routing steps and is optimal to within a multiplicative constant. Furthermore, the algorithm can be expressed by a few simple recursive functions.

4.2.4 AAB Scheduling

Finally, the basic ideas of AAB schedules will be presented in this subsection. Jung [100] proposes a theorem that a necessary and sufficient condition for an all-to-all broadcast in a k-ary 2-dimensional torus to be optimal is that the number of its data exchange steps between neighbor nodes is equal to its diameter. The obtained algorithms broadcast data under the constraints of the shortest paths routing and links load balancing without multiple receptions of identical data. The used approach, from its combinatorial optimization characteristic, has allowed it to reveal the existence of several families of optimal algorithms. It has been proven that there is a family of solutions which infer more efficient data switching processes at each node. These solutions are also easier to implement too. Unfortunately, at present, the technique is not applicable on other regular or even irregular networks.

For regular topologies the *Time-Arc-Disjoint Trees* (TADT) [212] tree technique could be utilized. For AAB each of the P input packets has to be delivered individually to all nodes. Every node u of network G is the root of a broadcast spanning tree $B(u)$. Each arc of $B(u)$ is labeled with the number of the round in which the broadcast packet passes this arc. An arc with label i is said to be of level i. The height of $B(u)$, $h(B(u))$, is the highest arc label in $B(u)$. Two trees $B(u)$ and $B(v)$ are said to be time-arc-disjoint if for any i ∈ {0,…, min($h(B(u))$,

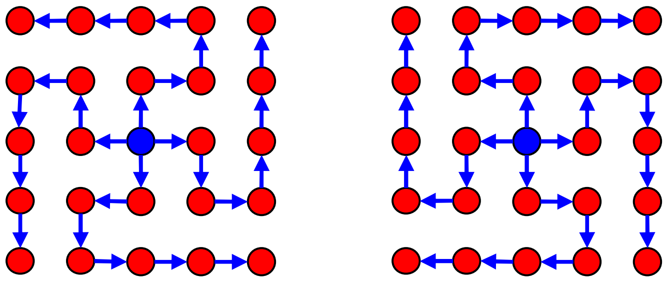


Figure 4.11. Examples of two different Time-Arc-Disjoint Trees for AAB communications rooted in the blue node.

$h(B(v))) - 1\}$, the sets of arcs at level i in $B(u)$ and $B(v)$ are disjoint. Two such Time-Arc-Disjoint trees, rooted in the same node, are shown Figure 4.11

The motivation behind these definitions follows from the lock-step assumption: all the nodes start their broadcasts at the same time and the broadcasts proceed synchronously along the trees with the same speed. At round i, only the arcs at level i are active in all the trees. If broadcast trees are time-arc-disjoint, then all these parallel broadcasts are pairwise link-contention-free.

Under these assumptions, the problem of designing an optimal AAB algorithm reduces the problem of designing TADTs [18] rooted in every node of the network! Here, we can nicely demonstrate how important the symmetry of the underlying topology for solving such a problem is. If the topology is vertex-symmetric, we may have a chance to find a generic structure of a TADT, independent of the location of its root. This problem has been solved for 2-D and 3-D tori, 2-D meshes, hypercubes, and some other Cayley graphs [212].

4.3 Conclusions and Outstanding Problems

This subsection summarizes the knowledge about scheduling of the collective communications (designing the communication algorithms). In section 4.1, we described the problem of choosing an underlying network that has great impact on time complexity of designed CC schedules, on the one hand, and also on the expensiveness of their design, on the other hand. From the lower bound, mathematically derived or estimated from interconnection topology, and the knowledge of the frequency of particular communications, a suitable topology can be selected and employed. If an optimized application has other special requirements on the underlying topology, some of combinatorial optimization techniques, like genetic algorithm [28], could be utilized to produce an application-specific topology.

The area of network design covers many outstanding problems like:

- How to generate easily implementable but also very powerful networks with low energy consumption?
- How to ensure high fault-tolerance?
- How to estimate the lower bounds for irregular networks with non-constants node degree?
- Can dynamic reconfiguration decrease the latency and increase the throughput of a network, and if it can, when should it be done?

If a suitable topology has been chosen and implemented, the routing strategy for CC has to be determined. Section 4.2 introduced the basic concepts of CCs scheduling and briefly described the state of the methods in this area. This issue can be concluded by making several statements:

- The OAS problem can be solved very simply implementing the ST tree.
- On the contrary, in case of AAS, all the theoretical and empirical techniques fail. Of course, the MIPL [141] or graph coloring [55] can be used, but the optimal schedule does not have to be discovered because of deterministic routing limitation or optimization time constraint.
- The problem with broadcasts is that the set *COMM* is not known in advance. Optimal algorithms reaching the theoretical lower bound of steps are not known even for familiar hypercubes of higher dimensions ($d >$ 7). However, for more complex topologies than simple meshes/tori or low dimensional hypercubes, this is still an outstanding problem.
- For AAB, the number of steps is limited by k messages that can be absorbed by any node in one step. Therefore, we can inform only adjacent nodes in one step and still develop optimum scheduling. The task is easier in symmetric networks: it is sufficient to find the TADT, which translated to all source nodes, creates no conflicts in any step of AAB communication. However, for asymmetric, irregular (non-constant k) networks, no similar systematic approaches exist so far.
- Many of the described methods are restricted by other limitations. We can mention, for example, port-model limitation, deterministic routing, slim nodes, regularity of underlying network, etc.

Up to now, we have considered only one-to-all or all-to-all communications. What about many-to-many communications? This area is completely unexplored, perhaps except for one-to-many broadcast more often referred to as multicast. The importance of many-to-many communication expresses in huge parallel systems with thousands of processing nodes, where many tasks are executed simultaneously and a task utilizes only a subset of these nodes. There are many opened problems related to many-to-many CC scheduling:

- Many-to-many communication patterns are insolvable by most of proposed techniques because from the wormhole point-of-view they transform the topology into an irregular form.

- Only nodes belonging to a given task can participate in a CC schedule and help, for example, with the distribution of messages; other nodes in the topology are not aware of it.
- Any fault in NoC transforms the topology into an irregular form, thus it is not solvable by presented techniques.
- Any fat topology transforms the both one-to-all and all-to-all CCs into the many-to-many communication patterns.
- A separated task closely related to this problem is how to spatially distribute processes onto nodes to ensure the best performance.

Finally, as we could see, the systematic approach, applicable to an arbitrary topology to obtain an optimal or near optimal schedule for any scatter or broadcast CC pattern does not exist. One of the reasons is a huge complexity of the scheduling problem proved to be NP-complete [31]. For this reason a novel technique, based on evolutionary algorithms will be developed and implemented to alleviate all presented restrictions and bottlenecks.

5

Evolutionary Algorithms

Since the 1960s there has been an increasing interest in the metaphor of imitating the evolution of living beings to develop powerful algorithms for difficult optimization problems. The algorithms belong to the class of stochastic algorithms named *evolutionary algorithm* (EA). The best known algorithms in this class include *genetic algorithms* (GA), developed by Holland [87]; *evolutionary strategies* (ES), developed by Rechenberg [164] and Schwefel [175]; *evolutionary programming* (EP), developed by Fogel [52]; and *genetic programming* (GP) developed by Koza [107]. There are also many hybrid versions that incorporate various features of the foregoing paradigms.

This chapter deals with detailed description of principles of Genetic Algorithms (GA) [66] and their modifications called *Estimation of Distribution Algorithms* (EDA) [137], [117] that help us to solve our optimization problem of deadlock-free scheduling of collective communications. The chapter also presents basic requirements and hints on how to design successful evolutionary algorithms that produce sufficient solutions of the problem.

An optimization problem solved by EA can be defined by specifying (1) a set of all potential solutions to the problem and (2) a measure to evaluate the performance of each candidate solution with respect to the objective. The goal is to find a solution or a set of solutions that perform best with respect to the specified performance measure. For example, in a maximum satisfiability (MAXSAT [78]) problem, each candidate solution represents an interpretation of all propositions (a list of truth values of all variables or propositions) and the quality of a solution can be defined as the number of satisfied clauses using the interpretation encoded by the solution. In the design of an aircraft wing, a solution can be represented by a set of parameters specifying the shape of the wing and the performance of each parameter set can be determined by an experiment in a wind tunnel. In the design of an algorithm for playing chess, a solution can be represented by a set of condition action rules, and the performance of each such set can be defined as the portion of games won against other competitor strategies.

In EA, there is no information about the relation between the semantics of solutions and the performance measure. The only way of learning something about this relation is to sample new candidate solutions and evaluate them. The task of finding the best solution to an EA optimization problem is extremely difficult. In order to illustrate the difficulty of EA optimization, imagine you are asked to implement a program in an unknown programming language, given only the syntax of the language and a procedure that evaluates how good each valid program is.

The following section introduces EAs as one of the approaches to *black-box optimization*, where the search for an optimum is driven by ideas inspired by the Darwinian survival of the fittest and the Mendelian inheritance of parental traits.

5.1 Genetic Algorithms

Genetic algorithms (GA) [135], first proposed by Holland in 1975 [87], approach black-box optimization by evolving a population of candidate solutions using operators inspired by natural evolution and genetics to solve problems in a wide variety of domains.

The genetic algorithm maintains a population of *individuals*. Each individual (string, chromosome) represents a potential solution to the problem at hand. Each individual is evaluated to give some measure of its *fitness*. Some individuals undergo stochastic transformation by means of genetic operations to form new individuals. There are two types of transformation: (1) *mutation*, which creates new individuals by making small changes in a single individual and (2) *crossover*, which creates new individuals by combining parts from two individuals. New individuals, called *offspring*, are then evaluated. A new population is formed by selecting the fitter individuals from the *parent population* and the *offspring population*. After several *generations*, the algorithm converges to the best individual, which hopefully represents an optimal or suboptimal solution to the problem. A general structure of the genetic algorithm is shown in Figure 5.1.

There are two important issues with respect to search strategies: *exploiting* the best solution and *exploring* the search space [21]. The genetic algorithms provide a directed random search in complex landscapes. Genetic operators perform essentially a blind search; selection operators hopefully direct the genetic search towards the desirable area of solution space. One general principle for developing an implementation of genetic algorithms for a particular real-world problem is to make a good balance between exploration and exploitation of the search space.

In general, solving a problem by genetic algorithms requires treating five basic issues, as summarized by Michalewicz [133]:

(1) Determine a genetic representation of solutions to the problem.
(2) Propose a way to create an initial population of solutions.
(3) Design an evaluation function rating solutions in terms of their fitness.
(4) Implement genetic operators altering the genetic composition of offspring during reproduction.
(5) Adjust values for the control parameters of genetic algorithms.

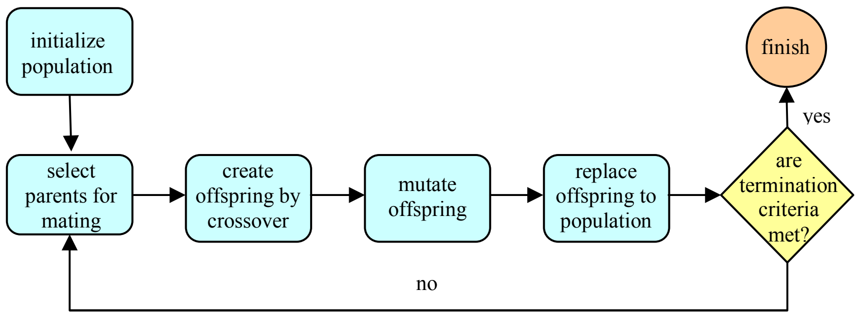

initialize population
finish
yes
select parents for mating
create offspring by crossover
mutate offspring
replace offspring to population
are termination criteria met?
no

Figure 5.1. Flowchart of a standard genetic algorithm.

In order to achieve a good balance between exploration and exploitation of the search space, all the issues must be examined carefully. Additional heuristics should be incorporated in the algorithm to enhance the performance. The most important parts of EA design will be further reviewed in the next subsections.

5.1.1 Encoding Issue

How to encode a solution of the problem into a chromosome (individual) is a key issue when using genetic algorithms. The issue has been investigated from many aspects, such as mapping characters from phenotype space to genotype space when solutions are decoded into individuals and metamorphosis properties when individuals are manipulated by genetic operators.

Classification of Encodings. In Holland's work, encoding is carried out using binary strings [87]. Binary encoding for function optimization problems is known to have severe drawbacks due to the existence of Hamming cliffs; pairs of encodings having a large Hamming distance while belonging to points of minimal distance in phenotype space [124]. For example, the pair 0111 and 1000 belongs to neighboring points in phenotype space (points of minimal Euclidean distance) but have maximum Hamming distance in genotype space. In order to cross the Hamming cliff, all bits have to be changed simultaneously. The probability that crossover and mutation will occur can be very small. In this sense, the binary code does not preserve the locality of points in the phenotype. For this reason the Gray code is often used [73].

For many problems in the industrial engineering world, it is nearly impossible to represent their solutions with binary encoding. During the last twenty years, various encoding methods have been created for particular problems to provide effective implementation of genetic algorithms. Depending on which kind of symbol is used as the *alleles* (values) of a *gene* (parts of a chromosome/parameters of a solution), the encoding methods can be classified as follows:

- Binary and integer encoding
- Real-number encoding
- Literal permutation encoding
- General data structure encoding

Integer and real-number encodings are best used for function optimization problems. It has been widely confirmed that integer and real-number encoding performs better than binary or Gray encoding for function optimizations and constrained optimizations [50], [133], [219]. Integer or literal permutation encoding is best used for combinatorial optimization problems. Since the essence of combinatorial optimization problems is the search for the best permutation or combination of items subject to constraints, literal permutation encoding can be the best way to solve this type of problem. For more complex real-world problems, an appropriate data structure is suggested as the allele of a gene, in order to capture the nature of the problem. In such cases, a gene may be an *n*-ary or more complex data structure.

In most practices, one-dimensional encoding is used. However, many real-world problems require solutions for multidimensional structures. It is natural to use a multidimensional encoding method to represent those solutions. For example, Vignaux and Michalewicz used an allocation matrix as an encoding for the transportation problem [217]. Cohoon and Paris used two-dimensional encoding for the VLSI circuit placement problem [35]. Anderson, Jones and Rayan used a two/dimensional grid type of encoding [5]. Moon and Kim used a two-dimensional encoding for graph problems [136]. Ono, Yamamura and Kobayashi used a job-sequence matrix as an encoding for job-shop scheduling problems [150]. A general discussion of multidimensional encoding and crossover was given by Bui and Moon [27] who argued that to fit solutions of multidimensional problems into one-dimensional encoding entails losing a considerable amount of the information contained in the multidimensional structure.

Infeasibility and Illegality. Genetic algorithms work on two types of spaces alternatively: coding space and solution space, or in other words, *genotype space* and *phenotype space*. Genetic operators work on genotype space, and evaluation and selection work on phenotype space. Natural selection is the link between chromosomes and the performance of decoded solutions. The mapping from genotype space to phenotype space has a considerable influence on the performance of a genetic algorithm. One outstanding problem associated with mapping is that some individuals correspond to infeasible solutions to a given problem. This problem may become very severe for constrained optimization problems and combinatorial optimization problems.

We need to distinguish between two basic concepts: infeasibility and illegality as shown in Figure 5.2. These are often misused in the literature. Infeasibility refers to the phenomenon that a solution decoded from chromosome lies outside the feasible region of a given problem; illegality refers to the phenomenon that a chromosome does not represent a valid solution to a given problem.

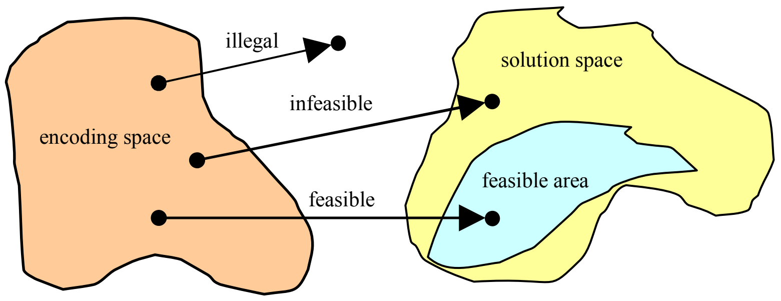


Figure 5.2. Simple demonstration of infeasibility and illegality.

The infeasibility of chromosome originates from the nature of the constrained optimization problem [168]. Whichever technique is used, conventional methods or genetic algorithms, it must handle the constraints. For many optimization problems, the feasible region can be represented as a system of equalities or inequalities. For such cases, the penalty method can be used to handle infeasible chromosomes [59], [132]. In constrained optimization problems, the optimum typically occurs at the boundary between the feasible and infeasible areas. The penalty approach will force the genetic search to approach the optimum from both sides of the feasible and infeasible regions.

The illegality of chromosomes originates from the nature of encoding techniques. For many combinatorial optimization problems, problem-specific encodings are used, and such encodings usually yield illegal offspring by a simple one-point crossover operation [87]. As an illegal chromosome cannot be decoded to a solution, the penalty techniques are inapplicable to this situation. Repair (restoration) techniques are usually adopted to convert an illegal chromosome to a legal one. For example, the well-known PMX operator [65] is essentially a two-point crossover for permutation representations, together with a repair procedure to resolve the illegitimacy caused by a simple two-cutpoint crossover. Orvosh and Davis [151] have shown that for many combinatorial optimization problems, it is relatively easy to repair an infeasible or illegal chromosome, and the repair strategy does indeed surpass other strategies, such as the rejecting or penalizing strategy.

Properties of Encodings. When a new encoding method is given, it is usually necessary to examine whether we can set up an effective genetic search using the encoding. Several principles have been proposed to evaluate an encoding [182]:

- *Nonredundancy*. The mapping between encodings and solutions should be 1-to-1. In other cases, GA will either waste time while searching (*n*-to-1) or another procedure performed on phenotype space to determine one solution among many possible ones has to be used (1-to-*n*).

- *Legality*. Any permutation of an encoding has to correspond to a solution. This property guarantees that most existing genetic operations can easily be applied to the encoding.
- *Completeness*. Any solution has a corresponding encoding. This property guarantees that any point in the solution space is accessible for a genetic search.
- *Lamarckian property*. The meaning of alleles for a gene should not be context dependent. The Lamarckian property for encoding concerns the issue of whether or not one chromosome can pass on its merits to future populations through common genetic operations [33].
- *Causality*. Small variations on the genotype space due to mutation should imply small variations in the phenotype space. The property is focused on the conservation of neighborhood structures; that is, for the successful introduction of new information by mutation, the mutation operator should preserve the neighborhood structure in the corresponding phenotype space.

5.1.2 Genetic Operators

Genetic operators are responsible for the generating of new candidate solutions. Together with a selection mechanism they make an engine of GA and explore the search space. In GA, accumulated information is exploited by the selection mechanism, while new regions of the search space are explored by means of genetic operators.

In conventional GAs, the crossover operator is used as the principal operator and the performance of a genetic system is heavily dependent on it. The mutation operator which produces spontaneous random changes in various chromosomes is used as a background operator.

Crossover. There are some basic ideas, how crossover can produce two offspring from two parents. We can include these common types:

- One-point, *n*-point and uniform crossover
- Arithmetic/intermediate crossover
- Permutation-based crossovers

One-point crossover was the original recombination operator proposed in [87] and examined in [43]. It works by choosing a randomly placed cut-point that splits both parents at this point and creates the two children by exchanging the tails (see Figure 5.3). One-point crossover can easily be generalized to *n-point crossover*, where the representation is broken down into more than two segments of contiguous genes, and then the offspring are generated by taking alternative segments from the two parents. The previous two operators worked by dividing the parents into a number of sections of contiguous genes and reassembling them to produce offspring. In contrast to this, *uniform crossover* [194]

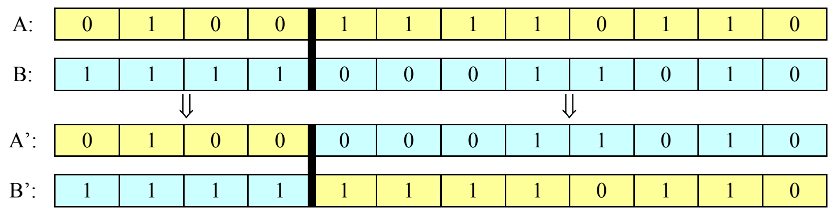

A: 0 1 0 0 1 1 1 1 0 1 1 0
B: 1 1 1 1 0 0 0 1 1 0 1 0
A': 0 1 0 0 0 0 0 1 1 0 1 0
B': 1 1 1 1 1 1 1 1 0 1 1 0

Figure 5.3. The schema of one-point crossover.

works by treating each gene independently and making a random choice as to which parent it should be inherited from.

There are two options for recombining two floating-point parents: (1) using an analogous operator to those used for binary chromosomes, or (2) using an operator that, in each gene position, creates a new gene value (allele) in the offspring that lies between those of the parents. In this way, crossover is able to create new gene material, but it has the disadvantage that as a result of the averaging process, the range of the allele values in the population for each gene is reduced [133].

Permutation-based representations present particular difficulties for the design of recombination operators, since it is not generally possible simply to exchange substrings between parents and still maintain the permutation property. *Partially Mapped Crossover* (PMX) was first proposed by Goldberg and Linge as a recombination operator for the Travelling Salesman Problem (TSP) in [65], and has become one of the most widely used operators for adjacency-type problems. The basic idea of PMX can be generalized as a combination of a two-point crossover with permutation preservation. First, PMX copies the segment between two cut-points from the first parent. Second, it completes the chromosome by looking for unused values from the second parent in order to preserve the permutation. The *Order Crossover* (OX) [40] was designed by Davis for order-based permutation problems. It begins in a similar fashion to PMX, but the intention is to transmit information about relative order from the second parent. Another mutation operator is *Cycle Crossover* (CX) [149] which is concerned with preserving as much information as possible about the absolute position in which elements occur.

Mutation. Many variants of a mutation operator have been also introduced during the last decades. The most important ones cover:

- Bitwise mutation of binary representations
- Random resetting and creep mutation based on integer representations
- Uniform and nonuniform mutation with a fixed distribution
- Permutation based mutation operators

Although a few other schemes have been occasionally used, the most common *bit-flip mutation* operator used for binary encodings considers each gene separately and allows each bit to flip with a small probability. The actual number of values changed is thus not fixed, but depends on the encoding length [87].

For integers encodings, there are two principal forms of mutation, and both mutate each gene independently within a user defined probability. The *random resetting* operator extends the bit-flipping so that a new value is chosen at random from the set of permissible values in each position. This is the most suitable operator to use when the genes encode for cardinal attributes, since all other gene values are equally likely to be chosen. *Creep mutation* was designed for ordinal attributes and works by adding a small (positive or negative) value to each gene within a defined probability. Usually these values are sampled randomly for each position, from a distribution that is symmetric around zero, and is more likely generated to small changes than large ones [60].

For floating-point representation, it is normal to ignore the discretisation imposed by hardware and consider the allele values as coming from a continuous rather that a discrete distribution. For a uniform operator the new values of a gene are drawn uniformly randomly from interval [$\mathrm{x}_{\min}$, $x_{\max}$] analogously to bit-flip or random resetting. Perhaps the most common form of nonuniform mutation used with floating point representation takes a form analogous to the creep mutation for integers. The operator adds to the current gene value an amount drawn randomly from a Gaussian distribution with a mean of zero and a user-specified standard deviation, and then curtailing the resulting value to the range [$\mathrm{x}_{\min}$, $x_{\max}$], [145].

For permutation representations, it is no longer possible to consider each gene independently, but instead finding legal mutations in a matter of moving alleles around in the genome. The most common ordered proposed operators, firstly published in [193], include swap, insert, scramble, and inversion mutation. The *swap operator* swaps two randomly picked genes in the chromosome. *Insert operator* works by picking two alleles at random and moving one so that it is next to the other, shuffling among the others to make room. *Scramble mutation* randomly chooses some subset of values within a chromosome and scrambles them. *Inversion mutation* works by randomly selecting two positions in the string and reverse the order in which the values appear between those positions.

The proper choice of genetic operator is problem (encoding) dependent. Our endeavour should be to design such operators that perform a good mixing of genetic material and prevent the creation of illegal solutions. Other hits on how to design genetic operators or choose the proper ones, can be found in the theory of GA proposed by Holland [87] and refined by Goldberg [67] in their building-block hypotheses; or in a convergence controlled variation hypothesis given by Eshelman, Mathias, and Schaffer [49].

Cheng and Gen suggests another approach for designing genetic operators [32]. For the two genetic operators, crossover and mutation, one is used to perform a random search to try to explore the area beyond a local optimum, and the other is used to perform a local search to try to find an improved solution. The genetic search then possesses two types of search abilities. With this approach, the mutation operator will play the same important role as that of crossover operator in a genetic search.

5.1.3 Selection and Replacement Mechanisms

The principle behind genetic algorithms is essentially the Darwinian natural selection. *Selection* provides the driving force in a genetic algorithm. With too much force, a genetic search will terminate prematurely; with too little force, evolutionary progress will be slower than necessary. Typically, a lower selection pressure is indicated at the start of a genetic search in favour of a wide exploration of the search space, while a higher selection pressure is recommended at the end to narrow the search space. The selection directs the genetic search towards promising regions in the search space. On the other hand, the replacement mechanism is responsible for replacing whole or part of the current population by newly created offsprings. During the past two decades, many selection/ replacement methods have been proposed, examined and compared. The most common types are as follows [77]:

- Roulette wheel selection
- $(\mu + \lambda)$ and (μ, λ)- selection
- Tournament selection
- Steady-state reproduction
- Ranking and scaling
- Sharing

Roulette wheel selection was proposed by Holland [87]. The basic idea is to determine selection probability or survival probability for each chromosome proportional to the fitness value. Then a model roulette wheel can be made displaying these probabilities. The selection process is based on spinning the wheel a number of times equal to population size, each time selecting a single chromosome for the new population. The wheel features the selection method as a stochastic sampling procedure. Baker proposed a stochastic universal sampling method [7] that uses a single wheel spin.

In contrast with proportional selection, $(\mu + \lambda)$-selection, and (μ, λ)-selection as proposed by Bäck are deterministic procedures that select the best chromosomes from parents and offspring [6]. Note that both methods prohibit selection of duplicate chromosomes from the population, so many researchers prefer to use this method to deal with combinatorial optimization problems. *Truncation* and *block selection* are also deterministic procedures that rank all individuals according to their fitness and select the best as parents [197]. *Elitist selection* is generally used as supplementary to proportional selection to preserve the best chromosome in the new generation if it has not been selected through a proportional selection process.

Generational replacement, replacing an entire set of parents by their offspring, can be viewed as another version of the deterministic approach. The *steady-state reproduction* of Whitley [226] and Syswerda [194] belongs to this class, in which the *n* worst parents are replaced by offspring (*n* is the number of offspring).

Another type of selection procedure contains random and deterministic features simultaneously. A typical example is the *tournament selection* of Goldberg, Korb, and Deb [70]. This method randomly chooses a set of chromosomes and picks out the best chromosome for reproduction. The number of chromosomes in the set is called the *tournament size*. A common tournament size is 2; this is called a *binary tournament*. Stochastic tournament selection was suggested by Wetzel [225]. In this method, selection probabilities are calculated normally and successive pairs of chromosomes are drawn using roulette wheel selection. After drawing a pair, the chromosome with higher fitness is inserted in the new population. The process continues until the population is full.

In the *proportional selection* procedure, the selection probability of an individual is proportional to its fitness. This simple scheme exhibits some undesirable properties. For example, in early generations, there is a tendency for a few superchromosomes to dominate the selection process; in later generations, when the population has largely been converted, competition among chromosomes is less strong and random search behavior will emerge.

The scaling and ranking mechanisms are proposed to mitigate these problems. The *scaling method* maps raw objective function values to positive real values, and the survival probability for each chromosome is determined according to these values. Fitness scaling has a two fold intention: (1) to maintain a reasonable differential between relative fitness ratings of chromosomes; and (2) to prevent a too-rapid takeover by some superchromosomes to meet the requirement to limit competition early but to stimulate it later.

Since De Jong's work, use of scaling objective functions has become widely accepted, and several scaling mechanisms have been proposed. According to the type of function used to transform the raw fitness into scaled fitness, scaling methods can be classified as linear scaling [75], sigma truncation [53], power law scaling [63], logarithmic scaling [75], etc.

For most scaling methods, scaling parameters are problem dependent. Fitness ranking has an effect similar to that of fitness scaling but avoids the need for extra scaling parameters [165]. Baker introduced the notion of ranking selection with genetic algorithms to overcome the scaling problems of the direct fitness-based approach [69]. The ranking method ignores the actual object function values; instead, it uses a ranking of chromosomes to determine survival probability. The idea is straightforward: sort the population from best to worst and assign the selection probability of each chromosome according to the ranking but not its raw fitness. Two methods are in common use: linear ranking and exponential ranking.

Sharing techniques, introduced by Goldberg and Richardson [68] for multimodal function optimization, are used to maintain the diversity of population. A sharing function is a way of determining the degradation of and individual's fitness due to a neighbor at some distance. With the degradation, the reproduction probability of individuals in a crowd peak is restrained while other individuals are encouraged to give offspring.

5.1.4 Creation of Initial Population

Initialization of the first population is kept simple in most EA applications and seeded by randomly generated individuals. In principle, problem-specific heuristics can be used in this step aiming at an initial population with higher fitness. Whether or not this is worth the extra computational effort very much depends on the application at hand.

5.1.5 Fitness Function

The role of the fitness function is to represent the requirements in order to adapt to them. It forms the basis for selection, and thereby it facilitates improvements. More accurately, it defines what improvement means. Technically, it is a function or procedure that assigns a quality measure to genotypes (e.g. in traveling salesman problem[65], the fitness function computes the length of route stored in a chromosome).

5.1.6 Values for the Parameters of GA

The behavior of genetic algorithms is characterized by a balance between exploitation and exploration in the search space. The balance is strongly affected by strategy parameters such as population size, maximum generation, crossover, and mutation ratio. How to choose a value for each parameter and how to find the values efficiently are very important, promising areas of research into genetic algorithms [74], [41]. Fixed parameters are used in most applications of GAs. The parameter values are determined using a set-and-test approach. However, the number of possible parameters and their different values means that this is a very time-consuming activity.

The technical drawbacks to parameter tuning based on experimentation can be summarized as follows:

- Parameters are not independent. Nevertheless, trying all different combination systematically is practically impossible.
- The process of parameter tuning is time consuming, even if parameters are optimized one by one, regardless of their interactions.
- For a given problem the selected parameter values are not necessarily optimal, even if the effort made for setting them has been significant.

During the history of EAs considerable effort has been spent on finding parameter values that were good for a number for test problems. A well-known example is that of [43], determining recommended values for basic parameters.

5.2 Estimation of Distribution Algorithms

The behavior of the evolutionary computation algorithms introduced in the previous section depends on several parameters associated with them (types of crossover and mutation operators, crossover and mutation ratios, population size, rate of generational reproduction, number of generations, etc.). If a researcher does not have experience in using this type of approach for the resolution of a concrete optimization problem, then the choice of suitable values for the parameters is itself converted into an optimization problem, as was shown by Grefenstette [74]. This reason, together with the fact that prediction of the movements of the populations in the search space is extremely difficult, has motivated the birth of a type of algorithms known as *Estimation of Distribution Algorithms* (EDA). EDAs were introduced in the field of evolutionary computation for the first time by Mühlenbein and Paaß [137], and some similar approaches can be found in the book by Zhigljavsky [228].

Holland [87] already recognized that to take into account interacting variables (genes) would be beneficial to genetic algorithms. This unexploited source of knowledge was called *linkage information*. Following this idea, in other approaches developed by different authors [70], [101], and [11] simple genetic algorithms were extended to process building blocks.

In EDAs, there are neither crossover nor mutation operators. Instead, the new population of individuals is sampled from a probability distribution which is estimated from a database containing selected individuals from the previous generation. Whereas in evolutionary computation heuristic, the interrelations (building blocks) between the different variables representing the individuals are kept implicitly in mind, in EDAs the interrelations are explicitly expressed through the joint probability distribution associated with the individuals selected at each iteration. In fact, estimation of the joint probability distribution associated with the database containing these selected individuals constitutes the bottleneck of this new heuristic. As estimation of distribution algorithms replaces standard recombination operators by building and sampling a probabilistic model, EDAs are also called as *Probabilistic Model Building Genetic Algorithms* (PMBGA) [158] and *Iterated Density Estimation Algorithms* (IDEAs) [24].

Since, there are a lot of different ways on how to estimate a probabilistic model at the required level of complexity, EDAs have been classified by this property into three basic classes which include:

- EDAs with no interactions between genes
- EDAs with pairwise interaction between genes
- EDAs with multivariate interactions.

5.2.1 EDA without Interactions

The **Population Based Incremental Learning** (PBIL) [8] replaces the population of candidate solutions by a *probability vector* (p_1, p_2, ..., p_n), where p_i denotes the probability of a 1 in the *i*-th position of solution strings (of the *i*-th gene). Each p_i is initially set to 0.5 which corresponds to a uniform distribution over the set of all solutions. In each iteration, PBIL generates s candidate solutions according to the current probability vector where $s \geq 2$ denotes the selection pressure. Each value is generated independently of its context (remaining bits) and thus no interactions are considered (see Figure 5.4).

The best solution from the generated set of s solutions is then used to update the probability-vector entries using

$$p_i = p_i + \lambda(x_i - p_i)\,, \tag{5.1}$$

where $\lambda \in (0,1)$ is the learning rate (say 0.02), and x_i is the *i*-th bit (gene) of the best solution. PBIL was also referred to as the *Hill Climbing with Learning* (HCwL) [115] and the *Incremental Univariate Marginal Distribution Algorithm* (IUMDA) [139]. Theoretical analyses of PBIL can be found in [115] and [71].

The **Compact Genetic Algorithm** (cGA) [80] eliminates the gap between PBIL and traditional GAs. Although cGA uses a probability vector instead of a population, updates of the probability vector correspond to replacing one candidate solution by another one using a population of size N and shuffling the resulting population using the population-wise uniform crossover [155].

Unlike PBIL and cGA, the **Univariate Marginal Distribution Algorithm** (UMDA) [137] maintains a population of solutions. A probability vector is computed using the selected population of promising solutions and new solutions are generated by sampling the probability vector. The new solutions replace the old ones and the process is repeated until termination criteria are met. UMDA is therefore equivalent to a GA with probabilistic uniform crossover (see sect. 5.1.2) Although UMDA uses a probabilistic model as an intermediate step between the original and new populations, unlike PBIL and cGA, the performance and dynamics of PBIL, cGA, and UMDA are similar.

PBIL, cGA, and UMDA can solve problems decomposable into subproblems of order one in a linear or quadratic number of fitness evaluations. However, if decomposition into single-bit subproblems misleads the decision making away from the optimum, these algorithms scale up poorly with problem size. For example, PBIL, cGA and UMDA require exponentially many evaluations until reliable convergence for additively separable traps.

5.2.2 EDA with Pairwise Interactions

EDAs with pairwise probabilistic models can encode dependencies in the form of a chain, a tree, or a forest. As these models move beyond the assumption of

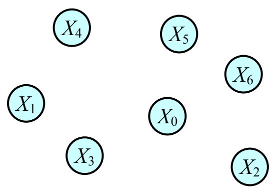


Figure 5.4. A graphical model with no interactions displayed as a Bayesian network.

variable independence, they represent the first step toward EDAs capable of solving decomposable problems of bounded difficulty in a scalable manner.

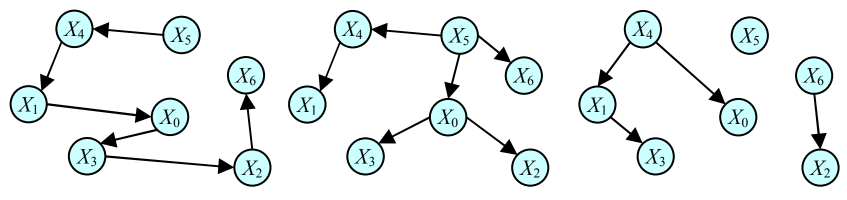
The **Mutual-Information-Maximizing Input Clustering** (MIMIC) algorithm [42] uses a chain distribution (Figure 5.5a) specified by (1) an ordering of string positions (variables); (2) a probability of a 1 in the first position of the chain; and (3) conditional probabilities of every other position given the value in the previous position in the chain. A chain probabilistic model encodes the probability distribution where all positions, except for the first position of the chain, are conditionally dependent on the previous position in the chain.

After selecting promising solutions and computing marginal and conditional probabilities, MIMIC uses a greedy algorithm to maximize mutual information between the adjacent positions in the chain. In this fashion the Kullback-Liebler divergence [109] between the chain and actual distributions is to be minimized.

The **Combining Optimizers with Mutual Information Trees** algorithm COMMIT, first proposed by Baluja and Davies [9], uses dependency trees (Figure 5.5b) to model promising solutions. Like in PBIL [8], the population is replaced by a probability vector, but in this case the probability vector contains all pairwise probabilities. The probabilities are initialized to 0.25. Each iteration adjusts the probability vector according to new promising solutions acquired on the fly. A dependency tree encodes the probability distribution where every variable except for the root is conditioned on the variable's parent in the tree. A variant of Prim's algorithm for finding the minimum spanning tree [163] can be used to construct an optimal tree distribution.

The **Bivariate Marginal Distribution Algorithm** (BMDA) [159] uses a forest distribution (a set of mutually independent dependency trees, see Figure 5.5c). This class of models is even more general than the class of dependency trees, because any forest that contains two or more disjointed trees cannot be generally represented by a tree. As a measure used to determine whether to connect two variables, BMDA uses Pearson's chi-square test [126]. This measure is also used to discriminate the remaining dependencies in order to construct the final model. In order to learn a model, BMDA uses a variant of Prim's algorithm [163].

Pairwise models capture some interactions in a problem with a reasonable computational overhead. EDAs with pairwise probabilistic models can identify, propagate, and juxtapose partial solutions of order two, and therefore they work



(a) MIMICS (b) COMMIT (c) BMDA

Figure 5.5. Graphical models with pairwise interactions in form of (a) chain, (b) tree, or (c) forest displayed as Bayesian networks.

well on problems decomposable into subproblems of order at mostly two [9], [24], [159]. Nonetheless, capturing only some pairwise interactions has still been shown to be insufficient for solving all decomposable problems of bounded difficulty scalably [9], [159]. That is why EDAs research has pursued more complex models discussed in the next section.

5.2.3 EDA with Multivariate Interactions

This section is an overview of EDAs with models that can encode multivariate interactions. Using general multivariate models has brought powerful algorithms capable of solving problems of bounded difficulty quickly, accurately, and reliably. On the other hand, learning distributions with multivariate interactions necessitates more complex model-learning algorithms that require significant computational time and still do not guarantee global optimality of the resulting model. Nonetheless, many difficult problems are intractable using simple models and the use of complex models and algorithms is warranted.

The **Factorized Distribution Algorithm** (FDA) [138] uses a fixed factorized distribution throughout the whole computation. The model is allowed to contain multivariate marginal and conditional probabilities, but FDA only learns the probabilities, not the structure (dependencies and independencies). In order to solve a problem using FDA, we must first decompose the problem and then factorize the decomposition.

The **Extended Compact Genetic Algorithm** (ECGA) [79] uses a marginal product model (MPM) that partitions the variables into several partitions which are processed as independent variables in UMDA (see Figure 5.6a). Each partition is treated as a single variable and different partitions are considered to be mutually independent.

In order to decide between alternative MPMs, ECGA uses a variant of the *Minimum Description Length* (MDL) metric [166] which favors models that allow higher compression of data (in this case, the selected set of promising solutions). More specifically, the *Bayesian Information Criterion* (BIC) [177] is used.

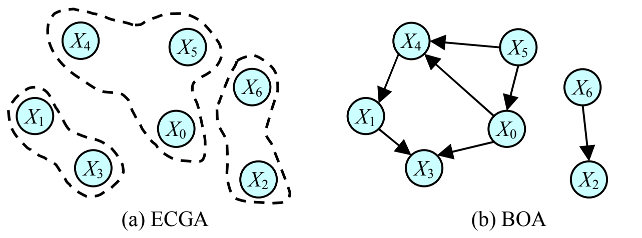


Figure 5.6. Graphical models with multivariate interactions displayed as a Bayesian network.

CGA provides robust and scalable solutions for problems that can be decomposed into independent subproblems of bounded order (separable problems) [173]. However, many real-world problems contain overlapping dependencies which cannot be accurately modeled by dividing the variables into disjoined partitions. This can result in poor performance of ECGA in relation to these problems.

The **Bayesian Optimization Algorithm** (BOA) [156] builds a Bayesian network for the population of promising solutions (Figure 5.6b) and samples the built network to generate new candidate solutions. Initially, BOA used the *Bayesian-Dirichlet* (BD) [36] metric subject to a maximum model-complexity constraint to discriminate competing models, but other metrics have been analyzed in later work. In all variants of BOA, the model is constructed by a greedy algorithm that iteratively adds a new dependency in the model that maximizes the model quality. Other elementary graph operators – such as edge removals and reversals – can be incorporated, but edge additions are the most important. The construction is terminated when no more improvement is possible. The greedy algorithm which is used to learn a model in BOA is similar to the one used in ECGA. However, Bayesian networks can encode more complex dependencies and independencies than models used in ECGA. Therefore, BOA is also applicable to problems with overlapping dependencies.

BOA uses an equivalent class of models as FDA; however, BOA learns both the structure and the probabilities of the model. Although BOA does not require problem-specific knowledge in advance, prior information about the problem can be incorporated using Bayesian statistics, and the relative influence of prior information and the population of promising solutions can be tuned by the user. An interesting study on incorporating prior problem specific information to improve the performance of BOA in graph partitioning and multiprocessor tasks scheduling can be found in [178].

EDAs that use models capable of covering multivariate interactions can solve a wide range of problems in a scalable manner; promising results were reported on two-dimensional Ising spin-glass systems [157], graph partitioning [178], telecommunication network optimization [167], silicon cluster optimization [173], and scheduling [100].

6 Evolutionary Design of Collective Communications

As we could see in previous chapters, a systematic approach applicable to an arbitrary topology to obtain an optimal or almost optimal schedule for any scatter or broadcast CC pattern does not exist. For this reason, a novel evolutionary based technique will be proposed.

Therefore, the hypothesis of the thesis can be formulated based on information and experiences collected in chapter 5: *Evolutionary design is able to produce optimal or near optimal communication schedules comparable or even better than which have been obtained by a conventional design for the networks sizes of interest. Moreover, evolutionary design can reduce many drawbacks of present techniques and invent still unknown schedules for an arbitrary topology and scatter/broadcast communication patterns.*

This chapter deals with the description of the main components of the technique, their design and implementation. The input data structure containing all necessary information about a solved task will be introduced, and the preprocessing of them will be described in the first subsection. Suitable encodings and their properties will be investigated in the second subsection. The third subsection will continue with a definition of the fitness function. The next subsection will describe acceleration and restoration heuristic to reduce execution time and increase success rate of proposed algorithms. The topic of the last subsection is an introduction of the heuristic mutation operator.

6.1 Input Data Structure and Preprocessing

The description of a network topology and other necessary information about a particular CC are specified in an input data file. The structure of the file is as follows:

- Any text/row beginning with the hash (#) symbol is considered to be a comment.

- The first uncommented row determines the number of nodes in a particular topology and the maximal number of output links per node (maximal node degree).
- Any other row describes one node. A row contains the node index; its operation mode within the CC (transmitter, receiver, both functions, switch); and a list of the node's neighbors, where two nodes are considered to be neighbors only if they are connected by a simple direct link.

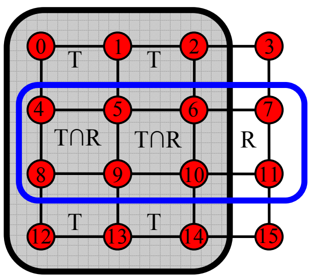
Figure 6.1 illustrates the 4×4 2D mesh with specified sets of transceivers and receivers that are performing many-to-many communication with each other. The topology contains 16 nodes; six of them are operating only as transmitters, two of them are operating only as receivers and six nodes are executing both functions. Let us note that any receiver, including nodes operating in both modes, can accept messages from any transmitter and vice versa. The topology also contains two nodes not participating in the illustrated CC, and working only as switches (i.e. messages can be transported via them, but no message can be consumed, or created and distributed from them).

On the right side of Figure 6.1, there is an input file dump. The comments are shown in italics for lucidity. The number of nodes is set to 16 (4×4 nodes). The maximal number of links is set to 4, since no node has got more than 4 neighbors (east, south, west, and north). Then, the lists of nodes and their neighbors follow. The node operation mode is specified by one of these four symbols $\{T, R, B, N\}$.

6.1.1 Input Data Preprocessing

After the input file is loaded, the data have to be preprocessed. In the first phase, the preprocessor divides the set of all nodes V^* into a set of transmitters T and a set of receivers R. Thereafter, a set of terminal nodes V is determined as the union $T \cup R$. Finally, all the sets are ordered based on the node index.

The preprocessor generates all the shortest paths (the set $R_{x,y}$) between all transmitter-receiver pairs and saves them into a specific data structure in the operating memory in the second phase. The algorithm [62] is inspired by the breadth-first search algorithms (BFS). BFS is based on the searching of a graph where a transmitter 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 transmitter 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 have been 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 receiver node. Receiver leaves' indices confirm the existence of searched paths, which are then stored as sequences of incident node indices.



```
# 16 nodes 4x4 mesh topology

# number of nodes and
# maximal number of links per node
  16    4

# Node operation mode
# - T = Transmitter
# - R = Receiver
# - B = Transmitter and Receiver
# - N = Switch

# node index, operation mode,
  neighbors
   0   T    1    4
   1   T    2    5    0
   2   T    3    6    1
   3   N         7    2
   4   B    5    8         0
   5   B    6    9    4    1
   6   B    7   10    5    2
   7   R        11    6    3
   8   B    9   12         4
   9   B   10   13    8    5
  10   B   11   14    9    6
  11   R        15   10    7
  12   T   13              8
  13   T   14        12    9
  14   T   15        13   10
  15   N             14   11
```

Figure 6.1. Description of an input file containing the 4×4 mesh topology with a many-to-many communication pattern.

A sample tree constructed while searching for the shortest paths from node no. 0 to node no. 6 in the 4×4 2D mesh topology is shown in Figure 6.2. Three paths were actually found in the tree: 0-1-2-6, 0-1-5-6 and 0-4-5-6. If one-to-all communication is being scheduled, only paths from a single transmitter node to all other receiver nodes are searched for. On the other hand, for optimization of many-to-many and all-to-all communication, all paths between every communicating pair of transmitter-receiver nodes are considered.

In certain cases when the target topology has nonuniform numbers of links per node (irregular topologies), it can happen that an optimal routing schedule cannot be constructed from a set of the shortest paths only. The usage of the shortest paths only could cause heavy utilization of some links but the rare utilization of the others which can prevent the finding of an optimal solution. In

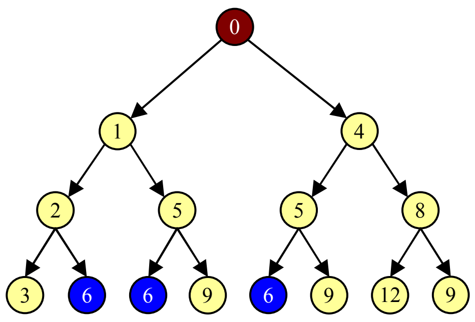


Figure 6.2. Construction of the shortest paths list from node no. 0 to node no. 6 in the 4×4 2D mesh topology.

order to avoid this problem, the algorithm must consider not only the shortest paths but also paths whose length may be longer. This approach can only be used for small topologies, because in cases of the large topologies the searching space of possible paths increases dramatically. For example, let us study 4×4 2D Mesh, all-to-all communication pattern, and the paths that are elongated by 2 (1 makes no sense in this topology) according to the shortest paths; whole number of paths is increased from 744 to 2784, for more complex topologies with 36 processors and more, rising to tens of thousands. Consequently, optimization algorithms are not able to search an optimal solution for more complex topologies by non-minimal routing.

6.2 Scatter Encoding

This chapter describes fundamental principles of chosen encoding of candidate routing schedules. As broadcast and scatter are completely different communication services, candidate solutions are encoded in separate ways. First, the definition and features of the scatter encoding will be introduced and examined.

6.2.1 Definition of the Scatter Encoding

Consider a scatter based CC communication between *M* transmitters from set *T* and *N* receivers from set *R* (e.g. from Figure 6.1).

(1) The CC can be defined as a set *COMM* of pair-wise transfers *src*, *dst* originating in $src \in T$ and terminating in $dst \in R$, where $src \neq dst$

$$COMM = \{comm_{src,dst} : src \in T, dst \in R, src \neq dst\}. \tag{6.1}$$

(2) A direct encoding can be designed for the scatter-based communication schedules (i.e. an exact description of the schedule is stored in a chromosome). A chromosome can be formalized as n-tuple of genes:

$$chromosome = \begin{pmatrix} gene_{0,0} & \dots & gene_{0,N-1} \\ \dots & \dots & \dots \\ gene_{M-1,0} & \dots & gene_{M-1,N-1} \end{pmatrix}, \tag{6.2}$$

where M is the number of transmitters and N is the number of receivers while n is the total number of genes. Notice that

$$M, N \le P \text{ and } n = M \cdot N. \tag{6.3}$$

(3) A gene $gene_{i,j}$ represents a single message transfer from the transmitter (source) $x_i \in T$ to the receiver (destination) $x_j \in R$, where $x_i \neq x_j$. The source and the destination are identified by the genes indexes i and j. A gene is an ordered couple:

$$\begin{aligned} &gene_{i,j} = (l_{i,j}, s_{i,j}), 0 \le i < M, 0 \le j < N, i \neq j \\ &l_{i,j} \in R_{i,j} \\ &0 \le s_{i,j} < Steps \end{aligned} \tag{6.4}$$

The first component $l_{i,j}$ represents a chosen path from the transmitter x_i to the receiver x_j stored in the set $R_{i,j}$. The second component $s_{i,j}$ determines a selected time slot (communication step) of the transfer between x_i and x_j. The total number of time slots (communication steps) is given by the predefined parameter *Steps*.

(4) The (shortest) path has been defined in section 2.2.4 as an ordered set of unidirectional channels

$$l_{i,j} = \{c_1, c_2, c_3, \dots, c_L\}, \tag{6.5}$$

where $src(c_1) = x_i$ and $dst(c_L) = x_j$.

(5) Next, consider a set *GENOME* containing all the genes included in a *chromosome*:

$$GENOME = \{gene_{i,j} : 0 \le i < M \text{ and } 0 \le j < N, i \neq j\}. \tag{6.6}$$

(6) Finally, we can define a bijective mapping f from set *GENOME* into set *COMM* meaning that each gene corresponds to a unique pair-wise transfer and also vice versa:

$$\begin{aligned} &f : gene_{i,j} \in GENOME \mapsto comm_{src,dst} \in COMM \\ &\text{iff } x_i = src, x_j = dst. \end{aligned} \tag{6.7}$$

6.2.2 Features of the Scatter Encoding

The basic advantage of the scatter encoding complies with most of the requirements defined in the section 5.1.1:

- *Nonredundancy*. The encoding has satisfied the condition of nonredundancy since all its instances can be transformed into unique communication schedules (see eq. 6.7). Any variation of genotype leads to a different schedule.
- *Legality*. The encoding also satisfies the condition of legality (any chromosome can be transformed into a communication schedule). If the genes values (alleles) respect the eq. 6.4, there is no way to create an illegal solution of the scatter schedule.
- *Completeness*. Because the chromosomes directly encode the communication schedules, any possible schedule can be stored in a chromosome.
- *Lamarckian property.* The Lamarckian property is satisfied considering that all point-to-point transfers are independent, and this implies that all alleles are context independent.
- *Causality*. The causality condition is not fully satisfied since even a small variation in a chromosome can produce a large difference in the quality of the decoded schedule (phenotype). The first gene component satisfies the condition since two paths from $R_{i,j}$ indexed by close values differ only in a small number of links. Hence, a swap of the paths will produce nearly the same schedule. On the other hand, a shift of a pair-wise transfer to a different time slot can rapidly modify channels' utilization producing hot-spots, and dramatically influencing the quality of the schedule.

6.2.3 Graphic Visualization of a Scatter Chromosome

The presented encoding covers all possible instances of *M*-to-*N* scatter chromosome. Figure 6.3 illustrates an example of the most complex all-to-all scatter chromosome on the 4×4 2D mesh.

Particular genes are separated by the blue lines. The first component $l_{i,j}$ is emphasized by a yellow background and $s_{i,j}$ by a blue background. The index i corresponds to the particular row (transmitter) and the index j to the particular column (receiver). Genes situated on the main diagonal are excluded from the chromosome since these communications do not have to be performed (source = destination).

M-to-*N* CCs are composed of only a subset of all possible pair-wise transfers. Some nodes are operating only as transmitters or receivers, while other nodes can be executing both functions, or can be excluded from communication performing only switching tasks.

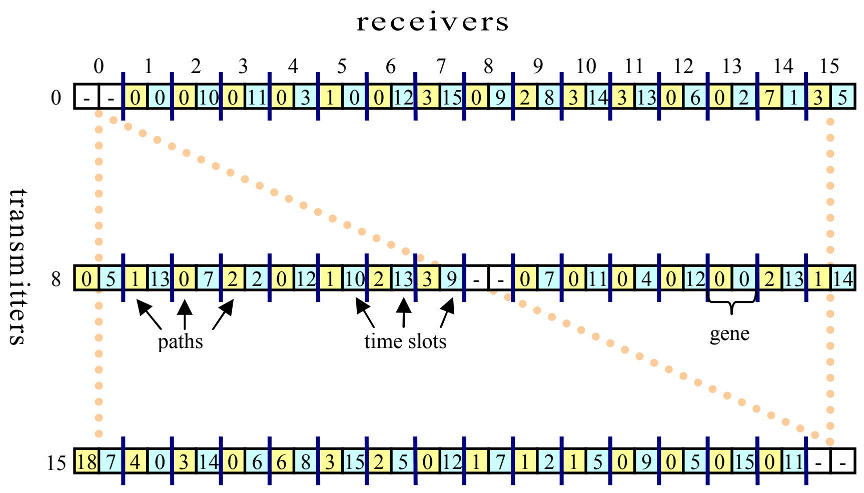


Figure 6.3. The structure of an AAS chromosome.

There is an example of MNS encoding based on a communication scheme presented in Figure 6.1 and shown in Figure 6.4. Because nodes no. 3, 7, 11, and 15 are not transmitting in this CC, their rows have been excluded from the chromosome (illustrated by hyphens). The columns representing nodes no. 0, 1, 2, 3, 12, 13, 14 and 15 have also been excluded from the genome since they are not operating as receivers.

6.3 Broadcast Encoding

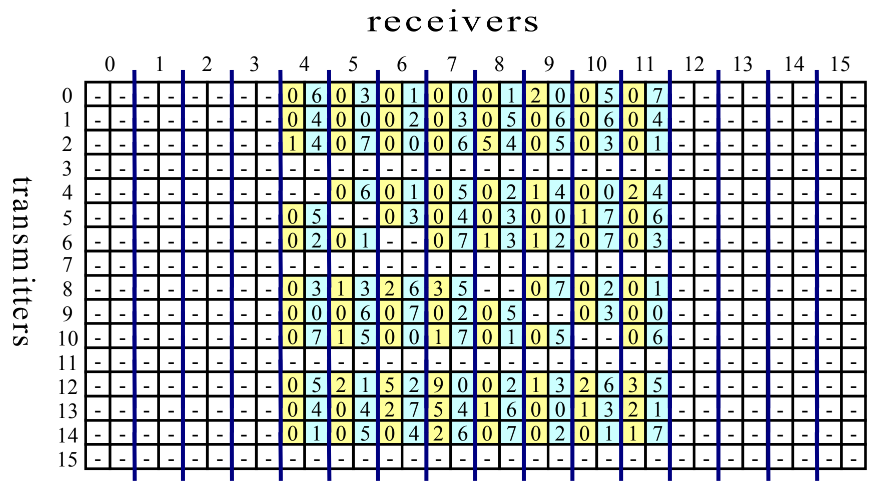
This section introduces the definition of the broadcast encoding. Its features are examined and graphic representation is illustrated in the following subsections.

6.3.1 Definition of the Broadcast Encoding

Consider a broadcast based CC communication between *M* transmitters from set *T* and *N* receivers from set *R* (e.g. from Figure 6.1). As previously discussed in section 4.2.3, the *COMM* set cannot be constructed in advance for broadcast based CCs, since each node already informed could also become a distributor of the broadcast message.

(1) Therefore, the definition of CC is based on a set of messages *MSG* that have to be delivered during a given CC to each destination. Let

$$MSG = \{msg_{src,dst} : src \in T, dst \in R, src \neq dst\} \tag{6.8}$$


receivers
transmitters

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	- -	- -	- -	- -	0 6	0 3	0 1	0 0	0 1	2 0	0 5	0 7	- -	- -	- -	- -
1	- -	- -	- -	- -	0 4	0 0	0 2	0 3	0 5	0 6	0 6	0 4	- -	- -	- -	- -
2	- -	- -	- -	- -	1 4	0 7	0 0	0 6	5 4	0 5	0 3	0 1	- -	- -	- -	- -
3	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
4	- -	- -	- -	- -	- -	0 6	0 1	0 5	0 2	1 4	0 0	2 4	- -	- -	- -	- -
5	- -	- -	- -	- -	0 5	- -	0 3	0 4	0 3	0 0	1 7	0 6	- -	- -	- -	- -
6	- -	- -	- -	- -	0 2	0 1	- -	0 7	1 3	1 2	0 7	0 3	- -	- -	- -	- -
7	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
8	- -	- -	- -	- -	0 3	1 3	2 6	3 5	- -	0 7	0 2	0 1	- -	- -	- -	- -
9	- -	- -	- -	- -	0 0	0 6	0 7	0 2	0 5	- -	0 3	0 0	- -	- -	- -	- -
10	- -	- -	- -	- -	0 7	1 5	0 0	1 7	0 1	0 5	- -	0 6	- -	- -	- -	- -
11	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
12	- -	- -	- -	- -	0 5	2 1	5 2	9 0	0 2	1 3	2 6	3 5	- -	- -	- -	- -
13	- -	- -	- -	- -	0 4	0 4	2 7	5 4	1 6	0 0	1 3	2 1	- -	- -	- -	- -
14	- -	- -	- -	- -	0 1	0 5	0 4	2 6	0 7	0 2	0 1	1 7	- -	- -	- -	- -
15	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -

Figure 6.4. The structure of MNS chromosome following the distribution of transmitters and receivers illustrated in Figure 6.1.

be a set of messages originating in transmitters $src \in T$, transported through the network via intermediate distributors $d \in V$, and consumed by receivers $dst \in R$. Notice, the distributor of the message transfer is not known in advance.

(2) A direct encoding has been designed to store broadcast-based communication schedules. A broadcast CC schedule is represented by the chromosome in the form of n-tuple of genes

$$chromosome = \begin{pmatrix} gene_{0,0} & \dots & gene_{0,N-1} \\ \dots & \dots & \dots \\ gene_{M-1,0} & \dots & gene_{M-1,N-1} \end{pmatrix}. \tag{6.9}$$

where M is the number of transmitters and N is the number of receivers while n is the total number of genes. Notice that

$$M,\ N \leq P \text{ and } n = M{\cdot}N. \tag{6.10}$$

(3) A gene $gene_{i,j}$ determines the way in which a receiver $x_j \in R$ obtains the broadcast message $msg_{i,j}$ from the transmitter $x_i \in T$ via a distributor $d_{i,j} \in V$. The producer and consumer of the message are identified by the genes indexes i and j. Individual genes are represented by an ordered triplet:

$$
\begin{aligned}
& gene_{i,j} = (d_{i,j}, l_{i,j}, s_{i,j}),\ 0 \le i < M,\ 0 \le j < N, i \ne j \\
& d_{i,j} \in D_{i,s_{i,j}} \subset V \\
& l_{i,j} \in R_{d_{i,j},j} \\
& 0 \le s_{i,j} < Steps
\end{aligned}
\tag{6.11}
$$

The first component of the gene selects a distributor $d_{i,j}$ of the message $msg_{i,j}$. Besides the transmitters, the distributor can be any node from $D_{i,s_{i,j}}$, and thus not only the transmitter. Set $D_{i,s_{i,j}}$ includes all nodes informed during all of $s_{i,j} - 1$ steps; see extended domination set theory [207] or section 4.2.3. The second component $l_{i,j}$ represents a chosen path between the distributor $d_{i,j}$ and a receiver x_j from the set $R_{d_{i,j},j}$. Analogously, the last component $s_{i,j}$ determines a selected time slot of the transfer between $d_{i,j}$ and x_j. The total number of time slots (communication steps) is given by the predefined parameter *Steps*.

(4) The (shortest) path was defined in the same way as in the section 2.2.4 as an ordered set of unidirectional channels.

(5) Next, consider a set *GENOME* containing all genes included in a *chromosome*:

$$GENOME = \{gene_{i,j} : 0 \le i < M \text{ and } 0 \le j < N, i \ne j\}. \tag{6.12}$$

(6) Finally, we can define a bijective mapping *f* from set *GENOME* into set *MSG*, thus each gene corresponds to a unique *src-dst* transfer of the message and also vice versa:

$$
\begin{aligned}
& f : gene_{i,j} \in GENOME \mapsto msg_{src,dst} \in MSG \\
& \text{iff } x_i = src,\ x_j = dst.
\end{aligned}
\tag{6.13}
$$

6.3.2 Features of the Broadcast Encoding

The basic characteristics of broadcast encoding are summarized below:

- *Nonredundancy*. The encoding has satisfied the condition of nonredundancy since all its instances can be transformed into a unique communication schedule, eq. 6.13. Any variation of genotype leads to a different schedule.
- *Legality*. The legality of all instances of OAB chromosomes can be simply guaranteed using restoration heuristic (see section 6.5.3 and the eq. 6.11). This correction of broadcast chromosomes follows the theory of domination sets [207]. The distributor in step *s* can become only a node informed during some of previous time steps {0,..., *s*-1}. Thus,

all nodes/genes violating this condition are corrected after any modification of the chromosome.

- *Completeness*. Because the chromosome directly encodes the communication schedule, any possible schedule can be stored in a chromosome.
- *Lamarckian* property. The Lamarckian property is satisfied considering that all pair-wise transfers are independent which implies that all alleles are context independent. On the other hand, the values within the scope of a gene are naturally context dependent, so that a selected path between a distributor and destination is dependent on the selected distributor. Fortunately, genetic operators can be simply modified to prevent linkage disruption by handling only whole genes.
- *Causality*. The causality condition is not fully satisfied since even a small variation in a chromosome can produce a large difference in the quality of a decoded schedule (phenotype). The reason is the same as in the case of OAS, moreover it is aggravated by the selection of distributors.

6.3.3 Graphic Visualization of a Broadcast Chromosome

The presented encoding covers all possible instances of *M*-to-*N* broadcast chromosome. Figure 6.5 illustrates an example of the most complex all-to-all broadcast chromosome on the 4×4 2D mesh. Particular genes are separated by the blue lines. The first component $d_{i,j}$ is emphasized by a green background, the second component $l_{i,j}$ is emphasized by a yellow background and $s_{i,j}$ by a blue background. The index i corresponds to the particular row (transmitter), and the index j to the particular column (receiver). Genes situated on the main diagonal are excluded from the chromosome since these communications do not have to be performed (source = destination).

M-to-*N* broadcasts are composed of only a subset of all possible point-to-point communications. Some nodes are operating only as transmitters or receivers, some others can be executing both functions, or can be excluded from communication performing only switching tasks (see 6.2.3).

6.4 Fitness Function Definition

The purpose of the fitness function is a quality measurement of candidate solutions. The candidate solutions encode communication schedules for a particular CC. The goal of the evolution is to design valid schedules with a predefined time complexity (number of communication steps in our approach). The necessary condition for all valid schedules is their *conflict-freedom*. Two communications are said to be in a conflict if and only if they share the same channel in the same time slot. In order to be able to implement evolved schedules into a parallel system, the conflicts of shared resources have to be prevented.

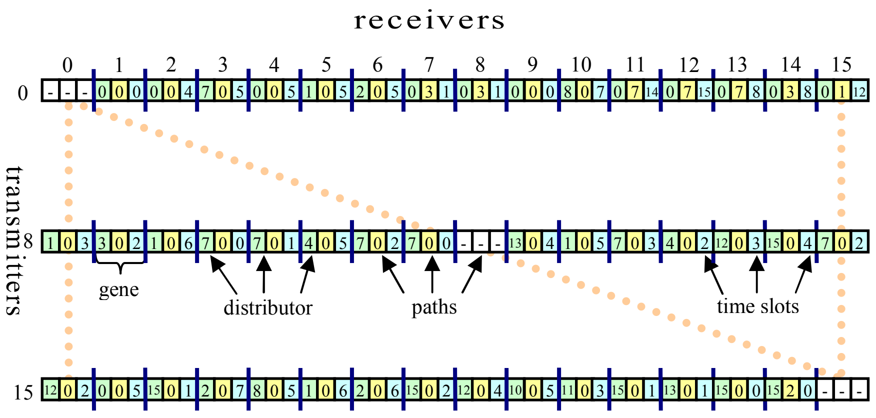


Figure 6.5. The structure of an AAB chromosome.

Consequently, the main idea of the fitness function is based on the conflict-free condition violations. All conflicts within a chromosome are discovered and their count is calculated. The quality of the chromosome is then determined from the number of conflicts.

The valid communication schedule for a given number of communication steps must be conflict-free. Valid schedules are either optimal (the number of steps equals the lower bound) or suboptimal. Evolution of a valid schedule for the given number of steps is finished up as soon as fitness (number of conflicts) drops to zero. If it does not do so in a reasonable time, the prescribed number of steps has to be one step increased and lunched again.

6.4.1 Formal Definition of the Fitness Function

This section proposes a formal description of the fitness function. The definition is the same for scatter and broadcast CC. For this purpose, we recall the introduced definitions of the encodings from sections 6.2.1 and 6.3.1.

(1) Let *SS* (Same Slot/Step) be a binary relation on the set *GENOME*. Let $a, b \in COMM$ (scatter) or $a, b \in MSG$ (broadcast) message transfers be represented by $gene_{i,j}$ and $gene_{k,l}$, then

$$gene_{i,j} SS\ gene_{k,l} \text{ iff } s_{i,j} = s_{k,l}. \tag{6.14}$$

Thus, two transfers are in relation *SS* if and only if they are being executed during the same time slot.

Now, we show that *SS* is an equivalence relation:

(a) SS is reflexive, since no transfer can be performed in more than one time slot.

(b) SS is symmetric considering $s_{i,j} = s_{k,l}$, then $s_{k,l} = s_{i,j}$

(c) SS is transitive. Let *a*, *b*, *c* be elements of *GENOME*. Whenever *a SS b* and *b SS c*, then also *a SS c* (*a* is being executed within the same slot as *c* whenever *a* is being executed within the same slot as *b* and *b* is being executed within the same slot as *c*).

Thus SS is an equivalence relation.

(2) The equivalence relation *SS* induces the partition *GENOME/SS*. Each equivalence class $[g_s]$ includes all transfers performed in the same time slot *s*.

(3) Let $E_{a,b}$ be a set of all shared channels between two transfers *a*, *b* represented by genes $gene_{i,j}$, $gene_{k,l} \in GENOME$ utilizes the paths $l_{i,j}$ and $l_{k,l}$, then

$$E_{a,b} = l_{i,j} \cap l_{k,l} \,. \tag{6.15}$$

The number of conflicts between *a* and *b* can be obtained as the cardinality of the set $E_{a,b}$.

(4) Define a multiset E_s including channels shared by all transfers within a given time slot, then

$$E_s = \bigcup_{a,b \in [g_s], a \neq b} E_{a,b} \,. \tag{6.16}$$

(5) The multiset *E,* covering all shared channels within the whole CC, can be obtained by a union over all equivalence classes. Thus

$$E = \bigcup_{[g_s] \in GENOME/SS} E_s \,. \tag{6.17}$$

(6) The total number of conflicts can be obtained as the cardinality of multiset *E*. Thus

$$Fitness = |E| \,. \tag{6.18}$$

6.4.2 Graphic Visualization of the Fitness Function

Figure 6.6 illustrates an example of two point-to-point communications forming a part of a CC. The (a) variant shows transfers 0→11 (the blue one) and 4→7 (the green one). These transfers will cause a conflict on the channel c = (5, 6), because of occupying the channel at the same time, which is naturally not possible. Contrary, the (b) variant does not invoke any violations of the conflict-free condition. Both transfers 0→11 and 13→7 are not sharing a channel.

The illustration of the fitness function concept is provided in Figure 6.7. On the left side of the figure, there are depicted two different AAS chromosomes with marked pair-wise transfers from Figure 6.6. The sources of the messages

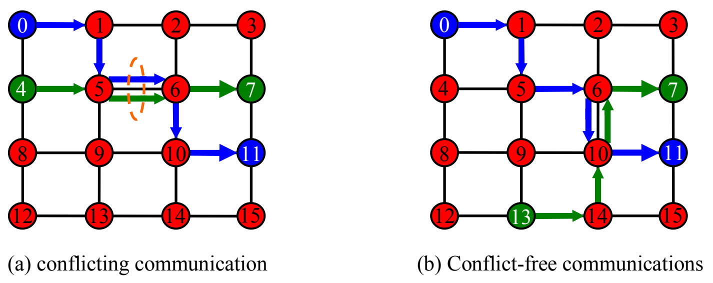

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
(a) conflicting communication
(b) Conflict-free communications

Figure 6.6. Two point-to-point communications illustrating the conflict-free condition.

transfers are determined by the row indices. The destination of the messages transfers are determined by the column indices.

First, the genes are categorized using the $s_{i,j}$ component. Then, all paths stored in the genes in the same category are mutually compared. The $l_{i,j}$ component of a gene is a pointer into the set $R_{i,j}$ maintaining all paths between the source-destination pairs. In the first case illustrated in Figure 6.7, the paths contain the same substring of length 2, hence one conflict is identified. In the second case, the paths do not share any substring. Notice, that two paths can stash more than one conflict.

6.5 Acceleration and Restoration Heuristics

This section introduces acceleration heuristics intended for speeding-up the optimization process. The acceleration heuristics are executed before the fitness function evaluation or inside the mutation operator (see section 6.6). The section also presents a restoration heuristic aimed at prevention of illegal solutions in the case of broadcast CC. The restoration heuristic is executed just before the fitness function evaluation.

6.5.1 Port Based Acceleration Heuristic

Port based acceleration heuristic improves optimization speed, taking into account the search space restrictions due to a limited message injection and acceptance capability of network nodes. Since no node can inject more than k messages into the network in a time slot (k-port model, see section 3.1), the acceleration heuristic checks this condition in the whole chromosome and redesigns ports' utilizations in all time slots.

Figure 6.8 shows node congestion. Four messages are waiting for transmission here, however only three of them can be dispatched since the node has been equipped by three output and three local ports. The fourth message has to wait for another time slot.

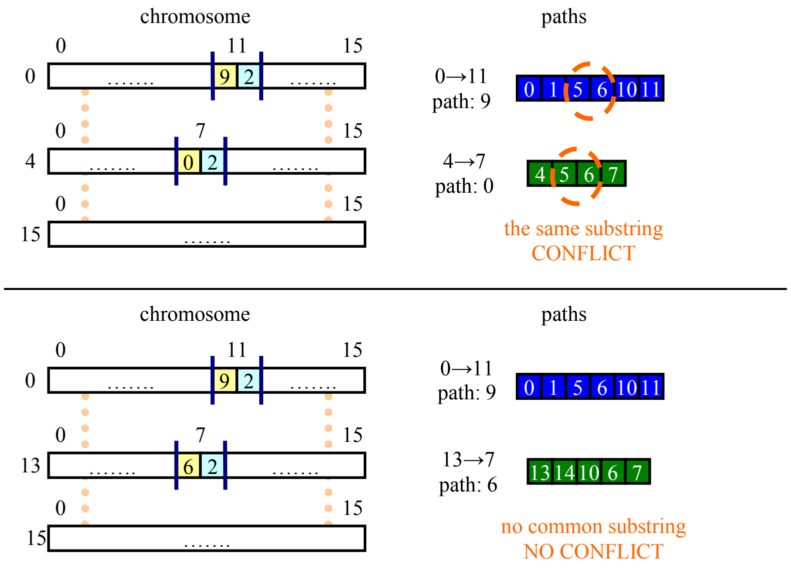


Figure 6.7. Illustration of the fitness function evaluation concept.

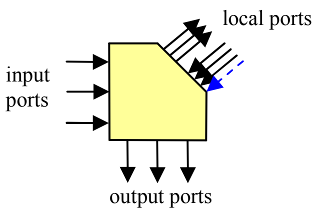


Figure 6.8. Illustration of port number limitation.

The implementation of the heuristic is based on a node utilization histogram (see Figure 6.9). First, the numbers of messages dispatched in particular time slots are counted for all transmitters. Then all the messages that cannot be transmitted due to overload are shifted to such time slots where there are free output channels.

The same check is also made on input channels to avoid situations a node is consuming more than k messages. This heuristic performs a local search on candidate solutions based on space domain so that conflicts caused by port-model limitation can be avoided in advance. This leads to the significant reduction of congestions speeding-up of the search because the evolution can only concern on conflict appearing along the paths.

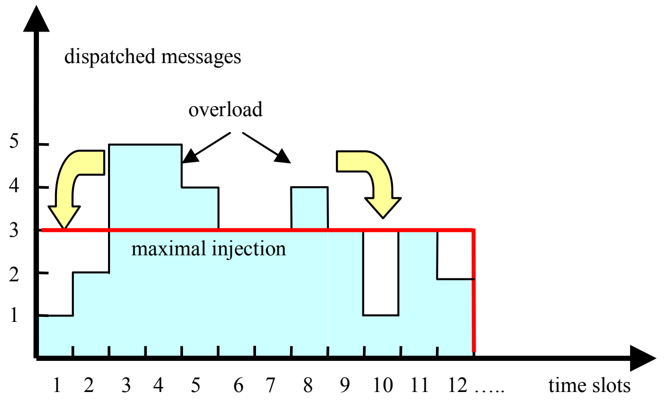


Figure 6.9. The concept of port based heuristic using a channel utilization histogram.

6.5.2 Transfer Swap Heuristic

The transfer swap heuristic swaps associated time slot of two transfers originating in the same transmitter. Figure 6.10 illustrates one possible situation. Consider two different time slots s_i and s_j and four different message transfers. The heuristic works as follows: (1) one transfer from the same transmitter in each time slots is randomly selected (e.g. 0→11from slot s_i, and 0→3 from slot s_j); (2) Their time slots are mutually exchanged, so that the transfer 0→11 will be executed in slot s_j, and the transfer 0→3 in slot s_i.

This heuristic performs a local search on a candidate solution based on the time domain. The conflicting transfers are rescheduled in spite of their timing to reduce total congestion.

6.5.3 Restoration of Broadcast Chromosomes

Considering that the broadcast chromosomes can get illegal during genetic manipulations (crossover, mutation, model sampling, etc.), the legality condition given by dominating sets, eq. 6.11, has to be verified.

The restoration (a repair of the broadcast tree) proceeds in subsequent communication steps.

First, the domination set for a particular time slot $D_{i,s_{i,j}}$ and transmitter $x_i \in T$ is determined. This set includes all nodes informed by message msg_i in all of the previous steps, which is

$$D_{i,s_i} = \{x_i\} \cup D_{i,s_1} \cup \ldots \cup D_{i,s_{i,j}-1}. \tag{6.19}$$

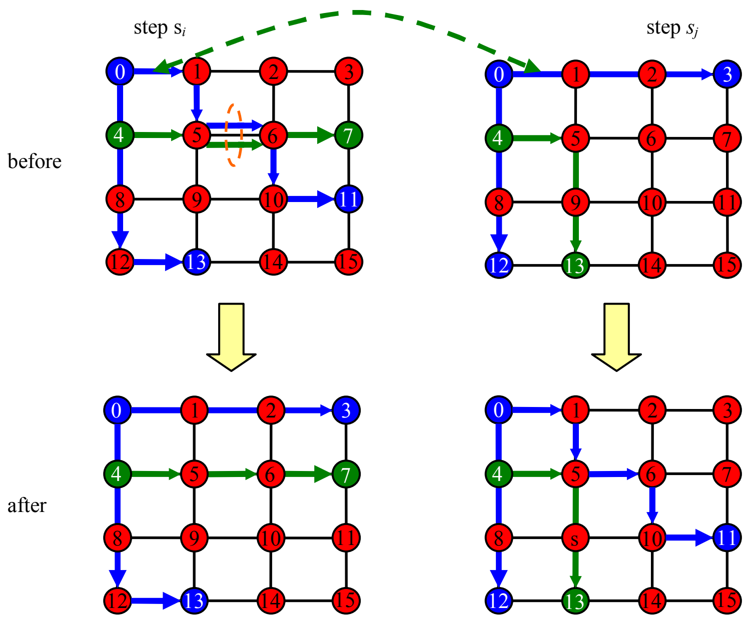


Figure 6.10. The concept of the transfer swap heuristic.

For all receivers $x_j \in R$ in a given step $s_{i,j}$, the condition $d_{i,j} \in D_{i,s_{i,j}}$ is checked.

If $d_{i,j}$ violates the condition of being a member of $D_{i,s_{i,j}}$, then $d_{i,j}$ is changed to be an element of $D_{i,s_{i,j}}$. The choice is based on node ports' utilization histogram (see 6.5.1).

A modification of the distributor node $d_{i,j}$ has an impact naturally on utilized channels $l_{i,j}$, and set $R_{d_{i,j},j}$. Hence, the original path is replaced by a newly chosen one from a list of exploitable paths between the new distributor-destination pair.

Figure 6.11 presents the idea of the restoration heuristic. There is an invalid broadcast communication on the left side of the figure, variant (a). The informed nodes (elements of $D_{i,s_{i,j}}$) are highlighted by a blue background. Notice that, node no. 11 is receiving the message from node no. 2. Naturally, this is not possible, since node no. 2 has not obtained the message yet. Therefore, the source of the transfer is exchanged for node no. 13 (one of the blue ones, see the right side of the figure). Consequently, the used path is also changed, variant (b).

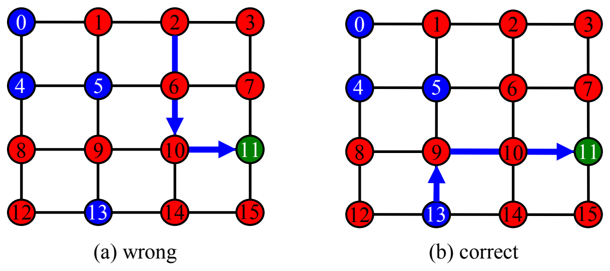


Figure 6.11. Restoration of the broadcast tree. The informed nodes are shown in blue. The destination node is shown in green.

6.5.4 Initial Population Building Heuristic

The search space of broadcast based schedules is much more restricted than the search space of scatter based ones. The initial population of scatter schedules can be simply generated by a uniform random generator, thus all gene values are randomly selected from corresponding definition scopes (see eq. 6.4).

However, this technique mostly leads to illegal solutions in the case of broadcast communication. Therefore, a specific heuristic for the initial population creation has been developed. The heuristic injects good building blocks into the initial population. The idea follows these steps for each gene:

(1) $d_{i,j}$ is selected using a uniform random generator from the set V.

(2) $l_{j,i}$ is selected using a uniform random generator from the set $R_{d_{i,j},j}$.

(3) $s_{i,j}$ is set initially to an undefined value.

(4) By selecting correct time slots, the restoration heuristic produces legal broadcast schedules that violate the conflict-free condition in much fewer cases.

6.5.5 Restricted Surrounding Heuristic

The heuristic is based on the search space pruning and it is incorporated into the restoration heuristic. If for a given topology and MNB pattern, this formula ($(M-1)/k > \log_k N$) is valid, then MNB can be performed as a controlled flood (i.e. all processors send their message only to uniformed neighbors). In each step, messages are propagated in waves through the interconnection network. This feature of the interconnection network and MNB communication can be employed with an advantage to prune the search space. The set of possible receivers of the broadcast message in a slot can be restricted only to nodes within a given radius $r \in <1, D/2>$, where D is a network diameter. This restriction leads to a massive reduction of possible engaged shortest paths.

The suitable radius value is chosen according to the ratio between the numbers of transmitters M and receivers N. The suitable radius r for OAB ($M = 1$, $N = P$) is $D/2$ considering that many optimal broadcast algorithms inform the furthermost node first (see section 4.2.3). On the other hand, the suitable radius value for AAB ($M = P$, $N = P$) is $r = 1$, [203]. Generally, the lower values of radius lead to faster convergence, but in some cases it is necessary to choose larger values ($D/2$ in the case of OAB communication) to ensure retrieval of a purposeful schedule.

As a consequence of the reception restriction, the restoration heuristic has to be modified. In a process of building the dominating set, it can happen that there is no node that could inform a selected receiver in a given time slot. In this case, the communication has to be postponed to the later time slot, where at least one possible source of broadcast message has already existed. In some cases this postponement can cause a number of communication steps which is in excess of the requested maximum. This situation is handled by a penalization function. The amount of penalty is given by the sum of all pair-wise transfers running over the prescribed maximum number of steps. Finally, this value is added to the conflict count computed by the fitness function.

Figure 6.12 illustrates the surrounding of node no. 1 with $r = 1$. This node can accept the messages only from the nodes belonging to its surrounding and also, the node can distribute the messages only to nodes lying in its surrounding.

6.6 Mutation Operator

The mutation is primarily intended to bring new genetic material into the population of chromosomes. Typically, very simple mutation operators are used (see section 5.1.2). As we know, with an increasing distance between a source-destination pair, the number of possible paths exponentially goes up. Therefore, it is necessary to employ a high-quality strategy responsible for testing possible paths and their organization into time slots.

The mutation operator exploits possibilities of some of the proposed heuristics. The mutation operator follows this procedure:

(1) The port based heuristic is executed. This step prevents conflicts on input and output channels of terminal nodes.

(2) There are two ways how to select a mutated gene. The gene is selected with uniform probability in the first half of instances in order to be able to escape from suboptimal solutions. Otherwise, we select only from genes that are causing conflicts to guide the search to better regions of the search space.

(3) One component of the selected gene is selected to be mutated.

    (a) Path mutation: a new path between source and destination is chosen using Gaussian distribution with the mean value equal to the current path index, and with the variance value equal to

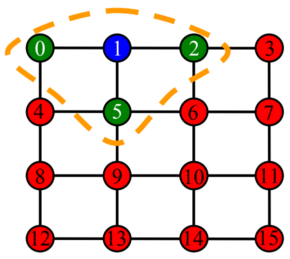


Figure 6.12. Illustration of the surrounding of the node no. 1 with $r = 1$.

the one third of the total number of paths between source-destination pairs. It ensures that similar paths to the current will be generated with higher probability, and on the other hand, any path can still be selected.

(b) Slot mutation: first, the transfer swap heuristic is executed. If it does not improve the quality of the solution, the time slot is generated using a uniform random generator.

(c) For broadcast solutions, there is a possibility to mutate the distributor of the message. The distributor is selected using a uniform random generator from all terminal nodes participating in the CC. Naturally, the restoration heuristic has to be applied before the fitness function is executed.

7 Optimization Tools and Parameters Adjustments

Combinatorial search and optimization techniques in general look for a solution to a problem among many potential solutions. For many search and optimization problems, exhaustive search is not feasible and some form of directed search is undertaken instead. In addition, rather than only the best (optimal) solution, a good non-optimal solution is often sought.

The basic components of many search and optimization techniques are very similar. The definition of basic components like: an evaluation function, an encoding and a global data structure is often expected. The previous chapter proposed the fundamental concepts of the most frequently utilized components independent of a searching technique. Consequently, the problem of CC scheduling can be solved by many common techniques like evolutionary algorithms [135], hill climbing [115], tabu search [165], simulated annealing [21], particle swarm optimization [104], ant colonies [46], and many others.

This chapter deals with the selection of the most suitable optimization tool for our task. We restricted ourselves to investigate the behaviour of *Standard Genetic Algorithm* (SGA) [66], *Mixed Bayesian Optimization Algorithm* (MBOA) [148] and *Univariate Marginal Distribution Algorithm* (UMDA) [137]. These three algorithms are the typical representatives of evolutionary and estimation of distribution algorithms. The optimal parameters of the selected algorithms are going to be established, and the performance of the algorithms is going to be measured and mutually compared in the following subsections. The victorious algorithm will be employed to check the quality of the proposed method and to design the optimal schedules on many topologies of interest.

7.1 Optimization Tools

This subsection introduces all the selected optimization tools, describes all their operators and suitable parameter setups.

7.1.1 Standard Genetic Algorithm

Standard Genetic Algorithm (SGA) [66], detailed in section 5.1.1, is probably the most frequently used evolutionary algorithm. In order to complete the algorithm design, some operators have had to be established:

(1) *Selection mechanism*. Two parents are selected from the parent population by a binary tournament operator [70].

(2) *Crossover*. One-point crossover with allowed crossover points only on the boundaries of the genes is used [87].

(3) *Mutation*. The mutation operator proposed in section 6.6 is employed.

(4) *Replacement strategy*. The steady-state reproduction is used [226]. The parent population is sorted according to its fitness after every generation. The worst two thirds are replaced by newly generated offspring. The higher replacement value ensures reasonable diversity level and prevents an early stuck in local optima.

In order to execute the algorithm, the values of several parameters will be adjusted experimentally. The most important ones cover [74]:

(1) *Population size*. The recommended values lie between 50 and c. 200.

(2) *Crossover probability*. The recommended values lie in (0.7, 0.9).

(3) *Mutation probability*. The recommended values lie in (0.001, 0.1).

(4) *Number of generations*. The number of generation strongly depends on the population size and the optimized task complexity.

The SGA algorithm has been implemented using GAlib [218] library and compiled by GNU C++ under a Linux operating system.

7.1.2 Mixed Bayesian Optimization Algorithm

Mixed Bayesian Optimization Algorithm (MBOA) algorithm was created by Jiri Ocenasek in [148]. MBOA is based on a Bayesian Optimization Algorithm (BOA). The probabilistic model of MBOA is a set of binary decision trees/ graphs. MBOA also differs from BOA in the heterogeneous model parameters. The decision trees can also be created for continuous or mixed domains. Although, the mutation operator is not frequently used in BOA algorithms because it could disturb convergence of the probabilistic model, its employing can generate new solutions from unexplored places of a search space and keep population diversity.

In order to complete the algorithm design, some operators have had to be established:

(1) *Selection mechanism*. The parent population is selected using a binary tournament operator [70].

(2) *Mutation*. The mutation operator proposed in section 6.6 is employed to incorporate a local search technique into an MBOA algorithm.

(3) *Replacement strategy*. The steady-state principle is used [226]. The parent population is sorted according to its fitness after every generation. The worst two thirds are replaced by a newly generated offspring. The higher replacement value ensures a reasonable diversity level and prevents an early stuck in local optima.

In order to execute the algorithm, the values of several parameters will be adjusted experimentally. The most important ones cover [148]:

(1) *Population size*. The recommended values are between 50 and several hundreds with respect to the character of the solved task.

(2) *Parent population size*. The size of the parent population is usually 50% of the population size. The probability model is estimated from the selected parent population.

(3) *Model building frequency*. The model is constructed in all generations.

(4) *Mutation probability*. If it is ever used, the recommended values lie (0.001, 0.1) [155].

(5) *Number of generations*. The number of generations strongly depends on the population size and the optimized task complexity.

MBOA has been implemented and compiled by standard GNU C++ under a Linux operating system.

7.1.3 Univarite Marginal Distribution Algorithm

A Univariate Marginal Distribution Algorithm (UMDA) lies on the frontier between classic genetic algorithms and estimation of distribution algorithms [137]. It does not cover any dependencies between variables (genes). For all components of the genes, histograms are created. Then, the frequencies of all values of the genes are calculated. In contrast to SGA, the whole parent population is used to learn the model and produce offspring (not only two individuals).

In order to complete the algorithm design, some operators have had to be established:

(1) *Selection mechanism*. The parent population is selected using a binary tournament operator [70].

(2) *Mutation*. The mutation operator proposed in section 6.6 is employed to incorporate a local search technique into an UMDA algorithm.

(3) *Replacement strategy*. The steady-state principle is used. The parent population is sorted according to its fitness after every generation. The worst two thirds are replaced by a newly generated offspring. The higher replacement value ensures a reasonable diversity level and prevents an early stuck in local optima.

In order to execute the algorithm, the values of several parameters will be adjusted experimentally. The most important ones cover [137]:

(1) *Population size*. The recommended values are between 50 and c. 200.

(2) *Parent population size*. The size of the parent population is usually 50% of the population size. The probability model is estimated from the selected parent population.

(3) *Model building frequency*. The model is constructed in all generations.

(4) *Mutation probability*. The recommended values lie between 0.001 and 0.1, in case it is ever used.

(5) *Number of generations*. The number of generations strongly depends on the population size and the optimized task complexity.

UMDA has been implemented and compiled by standard GNU C++ under a Linux operating system.

7.2 Experimental Comparison of Solution Quality

The first experimental work deals with the quality of obtained solutions. The goal of these measurements was the establishment of the most suitable parameters for all selected optimization tools to produce the most quality solutions. Two sufficiently difficult benchmark tasks were chosen; AAS and AAB patterns on the 4D hypercube.

An AAS communication pattern was chosen since it is the most complex variant of MNS communication. Similarly, AAB is the most complex variation of the MNB communication if the surrounding size is not restricted. The surrounding size was not restricted to get the benchmark more difficult in these experiments. The 4D hypercube with 16 nodes was selected due to its symmetry and known optimal solutions for both communication patterns. The corresponding AAS and AAB chromosome lengths were 512 and 786 gene components.

The numbers of communication steps were knowingly set one step below the lower bound for both communication patterns. It made it possible to compare the algorithms only by the number of conflicts after a finite number of generations. That prevented the algorithm finishing before a given number of generations had been evolved due to a global optima discovery.

Thirty experimental runs were executed for all the parameters setups. All graphs show the average values of the fitness function (conflicts counts) from all 30 runs after 500,000 generations.

7.2.1 Solution Quality Produced by SGA

In order to obtain a high-quality optimization tool based on SGA, three crucial parameters had to be established: the population size, the mutation probability, and the crossover probability. The tested values of parameters were as follows:

- Population size: 50, 100 and 150 individuals
- Mutation probability: 0.0, 0.2, 0.5, 0.7, 0.9
- Crossover probability: 0.0, 0.2, 0.5, 0.7, 0.9

Since the mutation operator also incorporates the local search techniques, the values of the mutation probability are significantly higher than in related literature.

Figure 7.1 shows the influence of the population size and the mutation and crossover probabilities on the best evolved solution quality. The results of the AAS benchmark are situated in the left column of the figure; cases (a), (b), and (c). The results of the AAB benchmark are situated in the right column of the figure; cases (d), (e), and (f).

The rows match particular population sizes: 50 individuals for cases (a) and (d); 100 individuals for cases (b) and (e); and 150 individuals for cases (c) and (f). The x-axis of the graphs represents growing mutation probability while the y-axis of the graph corresponds to growing crossover probability. Finally, the z-axis evaluates the solutions quality in terms of the conflicts count. The best parameter setup is highlighted by a red oval.

Figure 7.1 reveals several very important tendencies common for both benchmarks:

- *Higher population size does not lead to significant improvement of the solution quality!* It can be seen all graphs for both benchmarks are very similar. Actually, the higher population size slightly decreases the solutions quality. It can be caused by the slower convergence of SGA; however 500,000 generations appear to be sufficient to do it.
- *The crossover probability has only a small impact on the solution quality!* The mixture of genetic material performed by the crossover operator does not produce good building blocks (see building block theory [67]). It is probably due to the great number of potential source-destination paths.
- *The evolution is controlled mainly by the mutation operator!* The mutation operator improved by local search techniques is responsible for the solution quality improvement. With higher mutation probability, the number of conflicts significantly decreases. For example, if the mutation is not employed, the designed schedules contain about 200 conflicts. On the other hand, if 90% of all solutions are mutated, the obtained solutions attacks the 7-8 conflicts count level for AAS benchmark.

The following conclusions can be deduced from Figure 7.1:

- An appropriate population size lies between 50 and 100 individuals.
- Reasonable crossover probability for this population size is approximately 0.5 - 0.7.
- It is always better to mutate as many individuals as possible, so that the most favorable value is around 0.9.

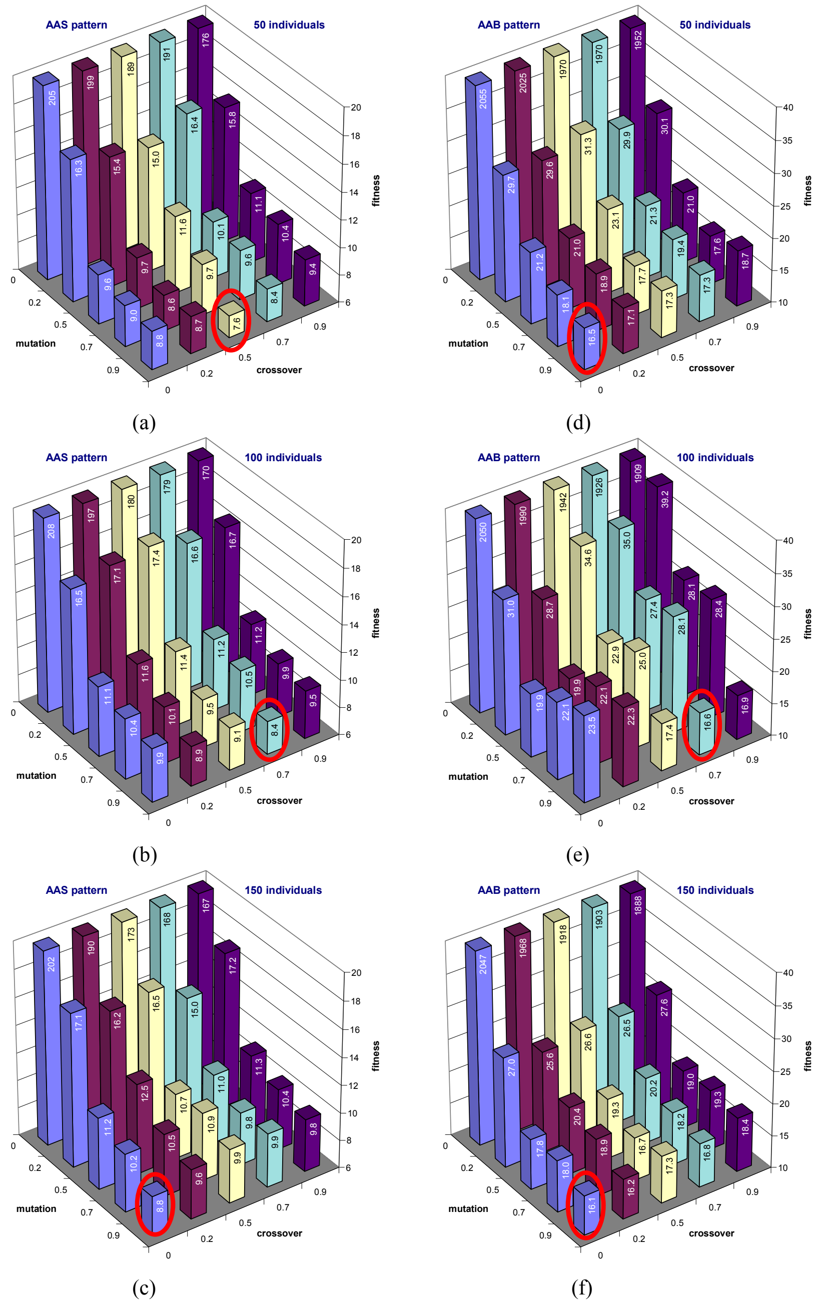


Figure 7.1. The influence of the population size, the mutation and crossover ratio on the solution quality: (a) - (c) AAS benchmark; (d) - (f) AAB benchmark; SGA algorithm.

7.2.2 Solution Quality Produced by MBOA

In order to obtain a high-quality optimization tool based on MBOA, we have had to establish two crucial parameters: the population size and the mutation probability. The tested values of parameters were set as follows:

- Population size: 50, 100 and 150 individuals
- Mutation probability: 0.0, 0.2, 0.5, 0.7, 0.9

Since the mutation operator incorporates also the local search techniques, the values of the mutation probability are significantly higher than in related literature. The maximal population size was limited due to the time complexity of the algorithm (see section 7.3).

Figure 7.2 shows the influence of the population size and the mutation probability on the solution quality. The results of the AAS benchmark are situated in the left column of the figure: cases (a) and (b). The results of the AAB benchmark are situated in the right column of the figure: cases (c) and (d). The first row shows the influence of the mutation probability in the whole gamut of the mutation probabilities. As the mutation affects the solution quality dramatically, the graphs are detailed in the second row. The x-axis of the graphs represents growing mutation probability. Three different bars representing the results achieved with different population sizes are depicted for each mutation probability. The y-axis evaluates the solutions quality in terms of the conflicts count. Finally, the best parameter setup is highlighted by a red oval.

Several interesting tendencies can be observed from Figure 7.2:

- *Higher population slightly improves the solution quality!* This statement is almost true in the whole mutation probability gamut for the AAS benchmark. Only in a few cases, some oscillations can be observed. Contrary, the opposite tendency can be observed for the AAB benchmark. The higher the population size is, the worse the solutions are produced in most cases. The fitness function values for 50 and 100 individuals are very closed indeed.
- *The evolution is controlled mainly by the mutation operator!* The mutation influences the quality dominantly. Disabling the mutation operator leads to very poor solutions. Higher values of the mutation probability bring only a minority improvement, but even a small innovation can bring us a high benefit.

Analogous to the previous subsection, some statements can be concluded from Figure 7.2:

- An appropriate population size lies between 100 and 150 individuals.
- Reasonable mutation probability is approximately 0.5 - 0.7.

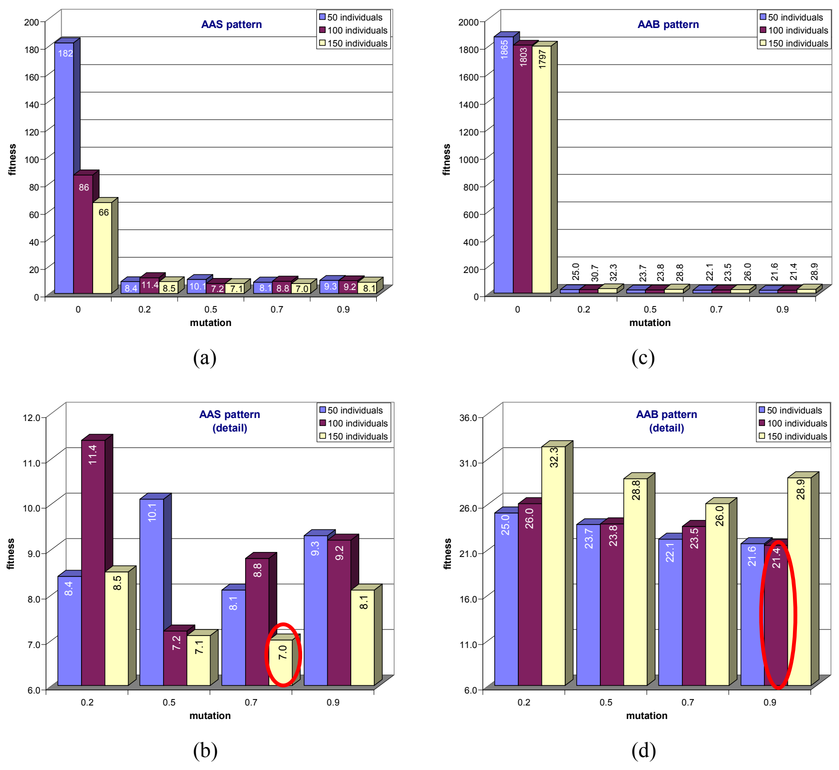


Figure 7.2. The influence of the population size and the mutation probability on the solution quality: (a) - (b) AAS benchmark; (c) - (d) AAB benchmark; MBOA algorithm.

7.2.3 Solution Quality Produced by UMDA

Finally, the quality of solutions produced by an UMDA algorithm was investigated. As in the previous cases, it was necessary to use suitable parameters: the population size and the mutation probability. The tested values of the parameters were set as follows:

- Population size: 50, 100, and 150 individuals
- Mutation probability: 0.0, 0.2, 0.5, 0.7, 0.9

Since the mutation operator also incorporates the local search techniques, the values of the mutation probability are significantly higher than in related literature.

Figure 7.3 shows the influence of the population size and the mutation probability on the solution quality. The AAS results are situated in the left column and the AAB results in the right column again. The details are presented in the second row. The x-axis represents growing mutation probability. Three different columns representing the results achieved with different population sizes are depicted for each mutation probability. The y-axis evaluates the solutions quality. Finally, the best parameter setup is highlighted by a red oval.

Very similar tendencies to the MBOA algorithm can be observed from Figure 7.3:

- *The population size has only a small influence on the solution quality!* As accrue from the figure, higher population size sometimes improves the solution quality, but sometimes does not. From this point of view, it is better to use a smaller population due to lower computation expensiveness.
- *The evolution is controlled mainly by the mutation operator!* If the mutation operator is not employed, the quality of the produced solutions is poor. The quality of the solutions dramatically appreciates incorporating the mutation operator into the evolution process. It is very interesting that even small mutation probability improves the quality many times. The higher values of the mutation probability only bring relatively small additional improvements.

Analogous to the previous subsections, some statements can be concluded from Figure 7.3:

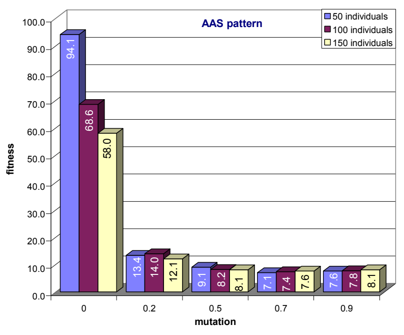
- An appropriate population size lies between 50 and 100 individuals.
- Reasonable mutation probability is approximately 0.7 - 0.9.

7.2.4 Summary

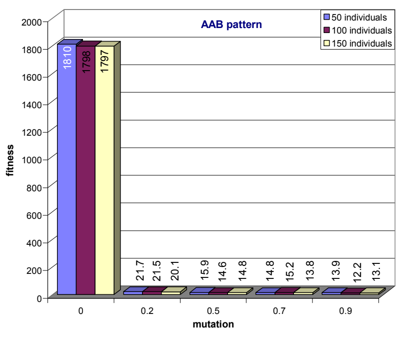
This subsection summarizes the quality of the solution produced by the proposed optimization tools. Figure 7.4 compares the achieved solution quality with the best parameters setups for AAS and AAB benchmarks and all the tools.

The best solutions for the AAS benchmark are produced by MBOA. Let us note that solutions produced by UMDA achieve very similar quality. SGA produces slightly worse solutions than both other optimization tools. The best solutions for the AAB benchmark are produced by UMDA. There are many more differences in solution quality between the investigated tools. SGA produces solutions with approximately 30% worse quality than UMDA. The worst solutions are produced by the MBOA algorithm with about twice as worse quality.

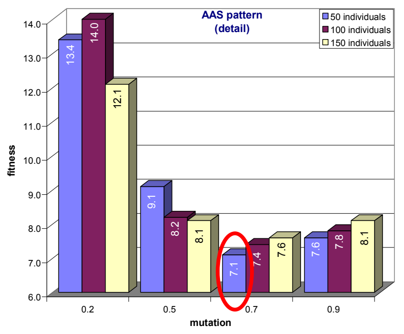
For all these reasons, UMDA can be declared the best optimization tool for both tested benchmarks.



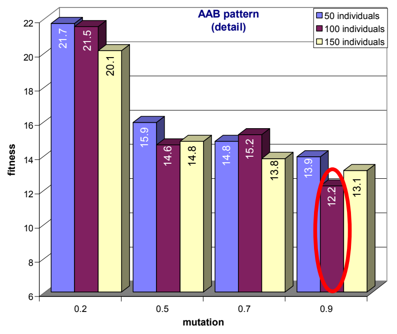
(a)



(c)

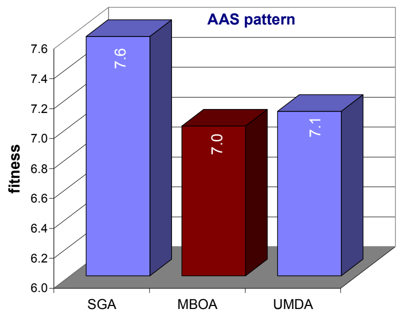


(b)

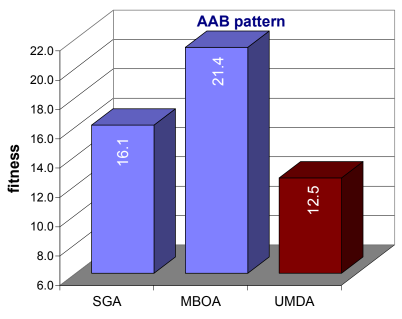


(d)

Figure 7.3. The influence of the population size and the mutation probability on the solution quality: (a) - (b) AAS benchmark; (c) - (d) AAB benchmark; UMDA algorithm.



(a) AAS benchmark



(b) AAB benchmark

Figure 7.4. The comparison of achieved solution quality by different optimization tools.

7.3 Experimental Comparison of Tool Speed

The second experimental work deals with the optimization speed of the proposed optimizations tools. The goal of these measurements was to estimate the time complexity of proposed algorithms and select the fastest one. The same benchmarks as in section 7.2 were used.

The execution times of 500,000 generations of all three optimization tools were measured for both AAB and AAS benchmarks. Then, the measured times were converted into the *generation time.* The generation time is the time that one generation takes to be executed. Thirty experimental runs were executed for all the parameter setups. All graphs show the average values of the generation time in milliseconds over all 30 runs. All results were measured on IBM Blade servers equipped with 2x dualcore AMD Opteron 275 processors and supplied by 4GB DDR2 RAM at 800 MHz.

In order to be able to compare all three optimization tools, the parameter values have to be set as close as possible with respect to the quality of produced solutions.

The following parameters values were used as follows:

- Population size: 50, 100, and 150 individuals
- Mutation probability: 0.9
- Crossover probability: 0.9

Figure 7.5 shows the generation time in milliseconds measured for all optimization tools. The results of the AAS benchmark are situated in the left column of the figure: cases (a) and (b) while the results of the AAB benchmark are situated in the right column of the figure: cases (c) and (d). The x-axis represents the increasing population size, the y-axis evaluates the generation time in ms.

First, we can take note of the incorporable higher generation time of the MBOA tool. MBOA is nearly a hundred times as slow as SGA and UMDA. MBOA exploits a very complex probabilistic model to be able to solve hard real-world tasks. It has actually achieved the best solution quality for the AAS benchmark. Unfortunately, the creation phase of such a complex model consumes too much time.

The second row of Figure 7.5 details the remaining optimization tools. It is evident that a simple genetic algorithm is approximately about 30-40% faster than UMDA. Although, UMDA only creates a simple probabilistic model, this phase still consumes more time than a simple one-point crossover. Nevertheless, the differences between SGA and UMDA go down for the AAB benchmark.

Finally, if we compare AAS and AAB generation times, we can observe a marked growth. The growth is caused by a longer genome in the case of AAB (three components per gene against two components per gene in AAS), and also more complex fitness function (restoration heuristic).

Owing to the results of this experimental study, SGA can be declared the fastest optimization tool for both tested benchmark.

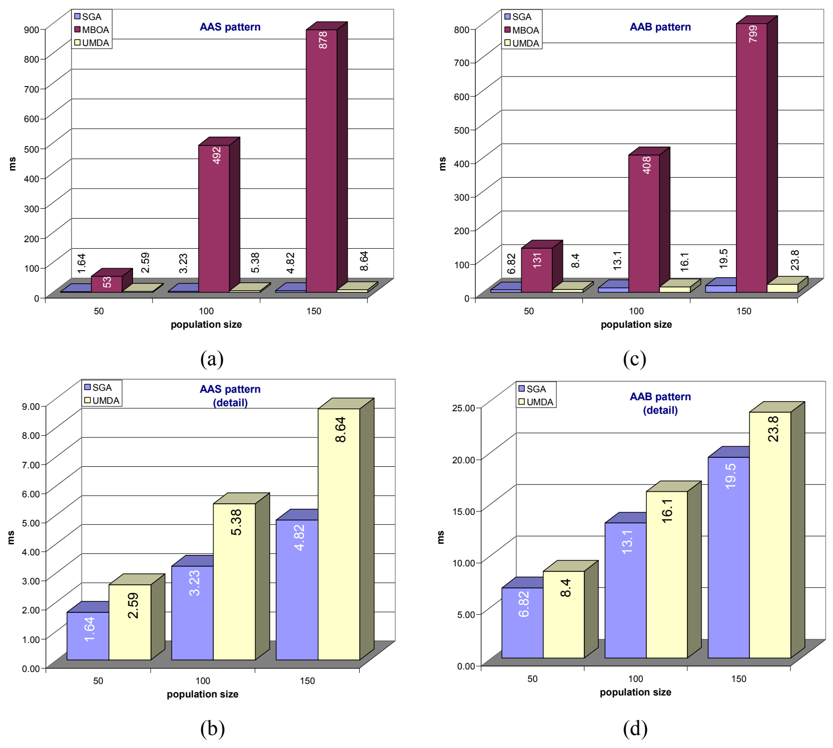


Figure 7.5. The performance of investigated optimization tools with the optimal parameter setups: (a) and (b) AAS benchmark; (c) and (d) AAB benchmark.

7.4 Experimental Comparison of Tool Scalability

This subsection deals with the scalability that indicates the ability to handle growing amounts of work in a graceful manner. Two different characteristics of the proposed optimization tools were investigated. Firstly, we focused on the number of fitness evaluations needed to reach the global optimum (NE). Secondly, we focused on the success rate representing the percentage occurrence of the global optimum in ten performed runs (SR). The global optimum is represented by a conflict-free (fitness function equals to zero) communication schedule reaching the lower bound on the number of communication steps.

Both these characteristics were examined using our familiar hypercubes. Five different instances of the hypercube topology with 8, 16, 32, 64 and 128 nodes served these tasks. All of the four basic communication patterns were used to examine the behavior of the tools for this time. Only ten experimental runs were executed because of an excessive time complexity in order of days for some experimental setups. Appropriate parameters were set up according to tips described in subsection 7.2:

- Population size: 100 individuals
- Mutation probability: 0.9
- Crossover probability: 0.9

7.4.1 Tool Scalability of Scatter Based Communications

Figure 7.6 presents the growth of the NE characteristic in dependence on the hypercube size. The figure shows the results for the OAS benchmark, case (a); and for the AAS benchmark, case (b). The x-axis of the graphs represents growing hypercube size while the y-axis evaluates the needed number of fitness function evaluations to reach the global optimum for a particular benchmark. The NE characteristics are completed with the SR ones shown in Table 7.1 and Table 7.2. The configurations of the hypercubes, where some tool discovered the global optima in all experimental runs, are highlighted in red.

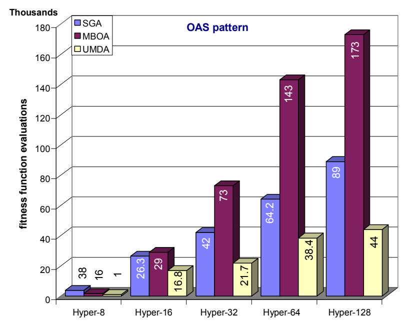
All proposed tools scaled well for the OAS pattern. Only a linear dependence of NE on the topology size was observed. Furthermore, the values of NE characteristic were very favorable and would allow us to solve more complex topologies. SE characteristics also promised a high yield of the optimization process (most runs would find the global optima).

Looking carefully through both characteristics in Figure 7.6a and Table 7.1, some conclusions valid for the OAS communication pattern can be made:

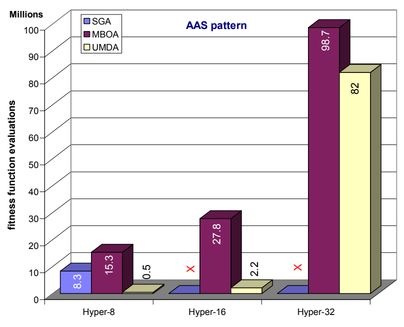
- SGA needed twice as small NE as MBOA, but the success rate (SR) was the lowest of all the tools. The 100% success rate was reached only for the smallest hypercube.
- MBOA needed the most NE (it was the slowest), but reached very good SR, comparable to UMDA.
- UMDA conducted best in both of the characteristics. It was the faster tool, and what is more important, it also attained the highest SR of all the tools.

Optimization of AAS pattern is a much more difficult task. Let us note that expanding the hypercube twice doubles the number of pair-wise communication of the OAS pattern. But what will happen in the case of AAS? The number of pair-wise communication increases four times! That is why the proposed tools attained the values of about a million evaluations.

More than quadratics dependence of NE on the hypercube size can be observed. The SGA had not been able to reach the optima in all cases. Consequently, the corresponding bars in Figure 7.6b were crossed. Furthermore, no tool could reliably solve bigger instances of hypercube than a 32-node one. Also SR statistics are worse than in the case of OAS. SR characteristic notify us that the search for the optimal AAS schedules on larger topologies has not been very effective. Practically 90% of the experimental runs have got stuck in a local



(a) OAS benchmark



(b) AAS benchmark

Figure 7.6. The scalability of investigated optimization tools - the numbers of fitness functions evaluations to reach global optima of scatter CCs with optimal parameter setups.

optimum (i.e. a few conflicts remained in the schedule). It makes us to increase the minimal number of communication steps and thereby simplify the task, but at the cost of performance lost in the designed schedule.

Some conclusion valid for the AAS pattern can be made from Figure 7.6b and Table 7.2

- SGA is not a suitable tool for this communication pattern. It has been able to solve only the smallest hypercube, and moreover with very low reliability.
- MBOA reached better SR values. However, NE values were very high, the dependency on the hypercube size was nearly quadratic.
- UMDA conducted best in both of the characteristics again. But the NE grew nearly exponential (2.2M evaluations for hyper-16 against 82M evaluations for hyper-32). The larger topologies might be solvable by this tool.

7.4.2 Tool Scalability of Broadcast Based Communications

Figure 7.7 presents the growth of the NE characteristic in dependence on the hypercube size for broadcast patterns. The figure shows the results for the OAB benchmark case (a); and for the AAB benchmark case (b) in the same way as in the previous section. The NE characteristics are completed by the SR ones shown in Table 7.3 and Table 7.4. The configurations of the hypercubes, where some tool discovered the global optima in all experimental runs, are highlighted in red.

Table 7.1. The success rate with increasing benchmark instances; OAS benchmark.

	Hyper-8	Hyper-16	Hyper-32	Hyper-64	Hyper-128
SGA	**100%**	80%	70%	70%	70%
MBOA	**100%**	**100%**	**100%**	90%	80%
UMDA	**100%**	**100%**	**100%**	**100%**	90%

Table 7.2. The success rate with increasing benchmark instances; AAS benchmark.

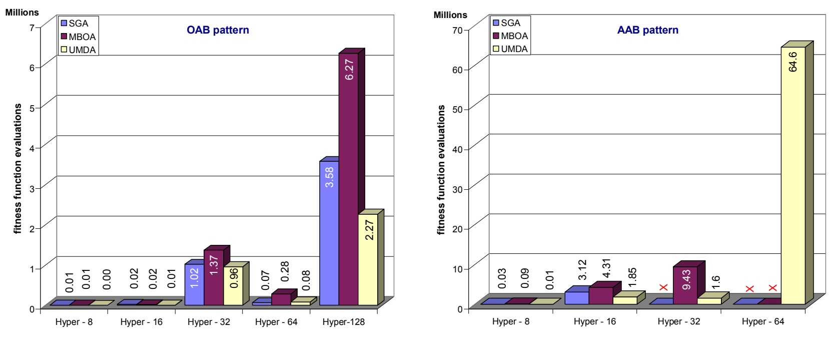
	Hyper-8	Hyper-16	Hyper-32	Hyper-64	Hyper-128
SGA	10%	-	-	-	-
MBOA	60%	30%	10%	-	-
UMDA	**100%**	40%	10%	-	-

The scalability of OAB pattern was a little worse than in the case of OAS, but still remained on the favourable level. NE characteristics commonly attained the values up to millions against several hundred of thousand. That was caused by a one third longer chromosome and more complex communication patterns. There is an interesting drop of NE for 64-node hypercube (hyper-64) in the figure. It is related to a rounding up function used in the formulas of the lower bounds. This phenomenon is explained later in section 7.4.3. SR characteristics also look good. All the tools have been able to produce optimal solution with a sufficient probability.

Looking carefully at both characteristics shown in Figure 7.7a and Table 7.3, some conclusions valid for the OAB communication pattern can be made:

- SGA, MBOA and UMDA achieved similar NE up to hyper-64. Bigger differences could be observed as late as the 128-node hypercube. SGA can be said to have achieved moderate NE with a slightly lower success rate.
- MBOA needed the most NE (it was the slowest), but reached very good SR comparable to UMDA on the other hand.
- UMDA conducted best in both of the characteristics again. It was the fastest tool (three times as fast as MBOA), and what is more important, it also attained the highest SR of all the tools, however, the SR values were very closed.

NE characteristics for the AAB pattern embody an exponential growth. Both MBOA and UMDA were able to find demanded solutions up to 32-node hypercube in a relatively short time. Unfortunately, MBOA reached only a 10% success rate for this topology. Moreover, SGA was not able to solve the bench-mark at all. On the other hand, UMDA broke this limit and also solved a 64-node hypercube but with a very low SR.



(a) OAB benchmark (b) AAB benchmark

Figure 7.7. The scalability of investigated optimization tools - the numbers of fitness functions evaluations to reach global optima of broadcast CCs with optimal parameter setups.

Some conclusions valid for the AAB pattern can be made from Figure 7.7b and Table 7.4:

- SGA is not a suitable tool for this communication pattern because it has been able to solve only two smallest hypercubes, and moreover with very low reliability.
- MBOA reached better SR values but with a higher NE. Unfortunately, it completely failed at solving the biggest hypercube.
- UMDA conducted best in both of the characteristics again. But the NE grows exponential (160k evaluations for hyper-32 against 64.6M evaluations for hyper-64). A drop of NE is present between a 16 and 32-node hypercube instances.

7.4.3 Summary

This subsection summarizes scalability of proposed optimization tools. Two different characteristics have been used to evaluate the scalability in terms of the number of evaluations needed to reach the global optimum and the success rate of its discovery. The values of these characteristics have been tabulated for all the proposed CC patterns.

Generally, the higher SR was observed at one-to-all patterns, where it was not a problem to solve the hypercubes up to 128 nodes. On the other hand, the limits of the proposed tools were found near the 32/64 node boundary in cases of all-to-all patterns.

Using real generational times presented in section 0 for the optimal population size, we can estimate the real time complexity for a particular optimization tool, a given topology and communication pattern. For simplicity, one example

Table 7.3. The success rate with increasing benchmark instances; OAB benchmark.

	Hyper-8	Hyper-16	Hyper-32	Hyper-64	Hyper-128
SGA	**100%**	**100%**	50%	50%	30%
MBOA	**100%**	**100%**	**100%**	70%	50%
UMDA	**100%**	**100%**	**100%**	90%	60%

Table 7.4. The success rate with increasing benchmark instances; AAB benchmark.

	Hyper-8	Hyper-16	Hyper-32	Hyper-64	Hyper-128
SGA	70%	20%	-	-	-
MBOA	**100%**	50%	10%	-	-
UMDA	**100%**	80%	40%	10%	-

will be introduced. Consider the AAB pattern, hyper-16 topology and UMDA optimization tool. The evolution process will take approximately:

$$t_{AAB}(Hyper-16) = \frac{NE}{pop\_size} generation\_time = \frac{1.85 \cdot 10^6}{100} \cdot 8.4 \cdot 10^{-3} = 155.4s \quad (7.1)$$

Note there is only a 40% probability to obtain the global optimum.

Finally, we should explain an interesting gap occurred in the NE characteristics. Let us compare the NE characteristics for the OAB communication pattern, Figure 7.7a. It is related to a rounding up function used in the formulas of the lower bound. While for hyper-32 the unrounded lower bound on the number of steps is 1.93, for hyper-64 it is 2.13. The time complexity of an optimal communication schedule can not exceed two communication steps in the first case, whereas it can be split into three steps in the second case. By comparing unrounded and rounded time complexities we can make a conclusion that in the case of a hyper-64 topology, much more interconnection links remain unused and the evolutionary algorithm has more space to find the optimal schedule. The same abnormality can be seen on many other topologies and CC patterns.

Taking into accounts all the mentioned pros and cons, the UMDA tool can be declared as the best scaling tool.

7.5 Conclusions

The goal of this section has been to search for the most suitable optimization tool. We have examined in depth the qualities of Standard Genetic Algorithm (SGA) [66], Mixed Bayesian Optimization Algorithm (MBOA) [148] and Univariate Marginal Distribution Algorithm (UMDA) [137].

Firstly, the suitable parameter values of all the optimization tools were searched for. Based on quality of obtained solutions the best parameters values were chosen. Consequently, all the tools were mutually compared and UMDA was found out to produce the best results. Afterwards, the generational time of the proposed tools with suitable parameters was compared, and after that, the time complexity was estimated. From these results, SGA was deducted to be the fastest of all. Lastly, the scalability of the proposed tools was examined. From these results, UMDA was concluded to scale best.

Finally, we can conclude this section and acclaim the winner. The UMDA algorithm seems to be the most effective and powerful optimization tool for our task. For all these reasons, it will be used for all further experiments.

8
Experimental Results of the Quest for High Quality Schedules

This chapter introduces the results of the proposed technique of evolutionary design of collective communication on wormhole networks. The key requirements for the schedules have been described in chapter 4. The main ideas of the proposed technique have been spelled out in chapter 6. The previous chapter has solved the issue of a suitable optimization tool selection and appropriate parameters setups. In this chapter, we are going to investigate the skills of the proposed technique implemented into UMDA [139] on various interconnection topologies, including several common topologies, optimal diameter degree topologies, some special topologies, multistage and fat topologies. We are also going to briefly discuss the results achieved on faulty topologies and with many-to-many communication patterns.

The goal of this chapter is to verify the quality of the technique and to determine the attainable upper bounds on various interconnection networks. Moreover, we would like to prove that our evolutionary based technique is able to reinvent well known collective communication algorithms like recursive doubling [128], Ho-Kao [84], etc. As we could see in section 4.2, there are many interconnection topologies with unknown optimal schedules for some CC patterns. Actually, a systematic approach optimally executing given CCs in asymmetric, irregular, fat or faulty topologies has not exist so far. Consequently, this chapter demonstrates the most interesting newly invented communication schedules for some promising interconnection networks. The achieved results (the upper bounds of time complexity in terms of numbers of communication steps) are also well-arranged in appendix.

8.1 Common Interconnection Topologies

This section introduces the results of the proposed technique implemented into the UMDA algorithm on the interconnection topologies often employed to build parallel computers. In this section, hypercubes, rings, meshes and tori are focused on. Further, the upper bounds of CC schedules produced by evolution are compared to theoretical lower bounds and the most interesting evolved schedules are presented here.

8.1.1 Hypercubes

Hypercubes are regular direct topologies with known optimal scheduling, except for OAB, for any hypercube size. Table 8.1 summarizes the time complexity of evolved schedules. From this table, the upper bounds (found by evolution) are evident to meet the mathematically derived lower bounds in most cases. A single integer represents both the lower and the upper identical bounds reached by UMDA. Otherwise, two integers in one cell separated by a slash indicate that the lower bound (a smaller integer) has not been reached. Finally, empty cells (illustrated by hyphens) denote the instances of the problem that have not been tested yet because of their time complexity.

OAS schedules reached the lower bounds in all cases. OAB schedules were equaled the best known algorithms like double tree and Ho-Kao. As the optimal schedule reaching 3 steps for a 256-node hypercube is not known up to now, the evolution equaled only the best known suboptimal solution of 4 steps; however the scheduling with 3 steps had been almost successful. Only one conflict remained. Unfortunately, this conflict was not possible to remove even after days of optimization. UMDA also found optimal schedules for AAB up to 64-node and AAS up to 32-node instances. Regarding AAS and 64-node hypercube, the lower bound had to be increase by 3 steps to obtain a conflict-free schedule at least. Let us note the chromosome size for a 64-node hypercube and AAS was 8192 genes with dozens of their possible values!

We will now present the optimal schedules only for the 8-node hypercube because of being limited by their size and complexity. Table 8.2 presents the optimal schedules for OAS and OAB. OAS communication can theoretically be scheduled in three time steps (slots), which has been proven by UMDA reinventing the conflict-free sequential tree (ST) [147]. Looking at the OAB schedule, a modified double tree can be found in this schedule. The message is sent to three nodes at different distances in the fist step. Consequently, node no. 4 helps the root with the broadcast. In an original double tree algorithm [128], the message is sent to the furthermost node (here node no. 7 at the distance of 3). In our case, the root of the second tree is chosen in the distance of 2.

The optimal AAS schedule cannot be executed in less than four communication steps (see Table 8.3). In each step, every node can distribute up to three distinct messages ($d = 3$). In spite of studying this schedule carefully, we have not found any system or regularity of the schedule. We can only note the schedule is formed by pairwise communications that do not cause any conflict.

In contrast, the AAB schedule illustrated in Table 8.4 embodies a strongly regular structure. The communication is guided by a controlled flood [203]. The schedule is formed by neighbor-to-neighbor transfers; each distributing the message only to uniformed adjacent nodes.

Table 8.1. Time complexity of the evolved schedules for four basic CC patterns on all-port hypercubes.

Topology	OAS	AAS	OAB	AAB
Hypercube-8	3	4	2	3
Hypercube-16	4	9	2	4
Hypercube-32	7	16	2	7
Hypercube-64	11	35/32	3	11
Hypercube-128	19	-	3	-
Hypercube-256	32	-	4/3	-

Table 8.2. Optimal OAS and OAB schedules on the 8-node all-port hypercube.

OAS		
step	**from→to**	**path**
1	0→4	0,4
	0→5	0,1,5
	0→6	0,2,6
2	0→1	0,1
	0→2	0,2
3	0→3	0,1,3
	0→7	0,4,5,7

OAB		
step	**from (origin)→to**	**path**
1	0→1	0,1
	0→3	0,2,3
	0→4	0,4
2	0→2	0,2
	0→5	0,1,5
	4 (0)→6	4,6
	4 (0)→7	4,5,7

Table 8.3. Optimal AAS schedule on the 8-node all-port hypercube.

source	time step/from → to (path)			
	step 1	**step 2**	**step 3**	**step 4**
0	0→3 (0,2,3)	0→1 (0,1)	0→2 (0,2)	0→6 (0,2,6)
		0→5 (0,4,5)	0-4 (0,4)	0→7 (0,4,5,7)
1	1→0 (1,0)	1→6 (1,5,7,6)	1→2 (1,3,2)	1→3 (1,3)
		1→7 (1,3,7)	1→4 (1,5,4)	1→5 (1,5)
2	2→4 (2,0,4)	2→0 (2,0)	2→5 (2,6,7,5)	2→1 (2,0,1)
		2→6 (2,6)	2→7 (2,3,7)	2→3 (2,3)
3	3→4 (3,7,6,4)	3→0 (3,1,0)		3→1 (3,1)
	3→5 (3,1,5)			3→2 (3,2)
	3→6 (3,2,6)			3→7 (3,7)
4	4→1 (4,0,1)	4→3 (4,0,2,3)	4→5 (4,5)	4→0 (4,0)
	4→7 (4,5,7)		4→6 (4,6)	4→2 (4,6,2)
5	5→3 (5,1,3)	5→2 (5,4,6,2)	5→1 (5,1)	5→0 (5,1,0)
	5→6 (5,4,6)		5→7 (5,7)	5→4 (5,4)
6	6→2 (6,2)	6→4 (6,4)	6→0 (6,2,0)	6→5 (6,7,5)
	6→3 (6,7,3)	6→7 (6,7)	6→1 (6,4,0,1)	
7	7→5 (7,5)	7→1 (7,5,1)	7→0 (7,3,1,0)	7→3 (7,3)
		7→2 (7,3,2)	7→6 (7,6)	7→4 (7,6,4)

Table 8.4. Optimal AAB schedule on the 8-node all-port hypercube.

source	**time step/from → to (path)**		
	step 1	**step 2**	**step 3**
0	0→1 (0,1)	1→3 (1,3)	6→7 (6,7)
	0→2 (0,2)	4→5 (4,5)	
	0→4 (0,4)	4→6 (4,6)	
1	1→0 (1,0)	3→2 (3,2)	4→6 (4,6)
	1→3 (1,3)	5→4 (5,4)	
	1→5 (1,5)	5→7 (5,7)	
2	2→0 (2,0)	0→1 (0,1)	7→5 (7,5)
	2→3 (2,3)	0→4 (0,4)	
	2→6 (2,6)	3→7 (3,7)	
3	3→1 (3,1)	1→0 (1,0)	0→4 (0,4)
	3→2 (3,2)	1→5 (1,5)	
	3→7 (3,7)	2→6 (2,6)	
4	4→0 (4,0)	5→1 (5,1)	7→3 (7,3)
	4→5 (4,5)	0→2 (0,2)	
	4→6 (4,6)	6→7 (6,7)	
5	5→1 (5,1)	4→0 (4,0)	3→2 (3,2)
	5→4 (5,4)	7→3 (7,3)	
	5→7 (5,7)	7→6 (7,6)	
6	6→2 (6,2)	2→0 (2,0)	5→1 (5,1)
	6→4 (6,4)	2→3 (2,3)	
	6→7 (6,7)	7→5 (7,5)	
7	7→3 (7,3)	3→1 (3,1)	2→0 (2,0)
	7→5 (7,5)	6→2 (6,2)	
	7→6 (7,6)	6→4 (6,4)	

8.1.2 Rings

The bidirectional ring topology, though very simple, is not free from the routing deadlock because the channel dependency graph is not acyclic [47]. This can be seen on a unidirectional as well as a bidirectional ring. The problem can be solved by the introduction of virtual channels [47] and by implementing rules on channel usage. We assume that these rules are adhered to automatically in all our CC schedules and thus the deadlock is avoided.

Table 8.5 summarizes the upper bounds $\tau^{CC}(G)$ reached by UMDA on unidirectional and bidirectional rings of sizes 8, 16, 24 and 32 nodes. From the table, the lower bounds $\tau_{CC}(G)$ and the upper bounds $\tau^{CC}(G)$ are evident to be mostly identical. The only exception is AAS communication in larger networks, where the lower bounds are apparently too tight. In fact, the obtained numerical results have led us to improve of theoretical lower bounds for AAS communication. The lower bound for AAS and WH networks in wide use has been formulated in [134], [39] as:

$$\tau_{AAS}(G) = \left\lceil \frac{P^2}{2B_C} \right\rceil, \tag{8.1}$$

Table 8.5. Time complexity of the evolved schedules on rings.

Topology	OAS	AAS	OAB	AAB
Ring-bi-8	4	8	2	4
Ring-bi-16	8	34/32	3	8
Ring-bi-24	12	78/72	4/3	12
Ring-bi-32	16	140/128	4	16
Ring-uni-8	7	28	3	7
Ring-uni-16	15	128/120	4	15
Ring-uni-24	23	300/276	5	23
Ring-uni-32	31	550/496	5	31

This bound have been improved to this form:

$$\tau_{AAS}(G) = \max\left(\left\lceil \frac{P^2/(2+2\Delta)}{B_C} \right\rceil, \left\lceil \frac{\sum}{Pd} \right\rceil\right). \tag{8.2}$$

For orthogonal topologies such as hypercubes, the correction is not needed as Δ=0 and two expressions in which the maximum is sought for are equivalent. However, for other networks the inclusion of the correction is essential and may change the bound dramatically (e.g. for the 8-node unidirectional ring the lower bound has changed from 16 to 28 steps!), see section 3.4.2.

OAS algorithm spreading customized messages to every partner is trivial because the source has to inject one (d = 1) or two (d = 2) messages at a time into the ring and the lower bound clearly applies. A sample OAS schedule on a unidirectional 8-node ring is illustrated in Table 8.6. The optimal OAB algorithm reaching the lower bound recursively doubles (k = 1) or triples (k = 2) the number of informed nodes in each step. The ring is split into two halves (k = 1) or three thirds (k = 2) and then the source sends a message in its image in the other half or images in other thirds. The same schedule has been reinvented by UMDA exactly for the unidirectional 8-node ring, see Table 8.6.

The AAS communication can be implemented as (P–1) permutations, deadlock-free with virtual channels, but not without link congestions (conflicts). Communication time thus cannot be estimated. We have therefore tried to find an AAS communication schedule organized into congestion-free steps with the usage of UMDA. Though in most cases the optimal solution has not been found, even a suboptimal solution could prevent contentions and in addition improve the performance of a parallel algorithm.

There is a sample solution of an optimal AAS schedule for the 8-node bidirectional ring shown in Table 8.7. For simplicity the path descriptions were omitted. The number of steps reaches the lower bound of 8 steps and 56 messages get distributed. In each step some nodes use one or all (two) their ports while other nodes are idle. For example, in step 1, node no. 4 is communicating simultaneously on both external channels; nodes no. 0, 2, 3 and 7 are communicating on a single channel; and nodes no. 1, 5 and 6 are idle. It is important to note that all 16 channels keep busy in all the steps.

Table 8.6. Optimal OAS and OAB schedules on the 8-node unidirectional ring.

OAS		
step	**from→to**	**path**
1	0→4	0,1,2,3,4
2	0→3	0,1,2,3
3	0→6	0,1,2,3,4,5,6
4	0→5	0,1,2,3,4,5
5	0→7	0,1,2,3,4,5,6,7
6	0→2	0,1,2
7	0→1	0,1

OAB		
step	**from (origin)→to**	**path**
1	0→4	0,1,2,3,4
2	0→2	0,31,2
	4 (0)→6	4,5,6
3	0→1	0,1
	2 (0)→3	2,3
	4 (0)→5	4,5
	6 (0)→7	6,7

Table 8.7. Optimal AAS and AAB schedules on the 8-node bidirectional all-port ring.

AAS								
source	**from → to**							
	step 1	**step 2**	**step 3**	**step 4**	**step 5**	**step 6**	**step 7**	**step 8**
0	0→4	0→3	0→1		0→2		0→7	
		0→5			0→6			
1		1→0	1→2	1→6		1→3		1→4
			1→5					1→7
2	2→0	2→1	2→6	2→3	2→5	2→7	2→4	
3	3→4	3→5		3→1		3→2	3→0	
				3→6		3→7		
4	4→2				4→0		4→3	4→1
	4→7						4→5	4→6
5		5→2	5→1		5→0	5→3	5→7	
		5→6			5→4			
6		6→0	6→7	6→2	6→5		6→4	6→1
				6→3				
7	7→3		7→0			7→1	7→2	7→4
						7→5	7→6	

AAB				
source	**from → to**			
	step 1	**step 2**	**step 3**	**step 4**
0	0→1	1→2	2→3	5→4
	0→7	7→6	6→5	
1	1→0	2→3	3→4	6→5
	1→2	0→7	7→6	
2	2→1	1→0	4→5	5→6
	2→3	3→4	0→7	
3	3→2	2→1	1→0	6→7
	3→4	4→5	5→6	
4	4→3	3→2	2→1	1→0
	4→5	5→6	6→7	
5	5→4	4→3	7→0	0→1
	5→6	6→7	3→2	
6	6→5	7→0	0→1	1→2
	6→7	5→4	4→3	
7	7→0	0→1	1→2	4→3
	7→6	6→5	5→4	

AAB communication around a unidirectional ring has also a straightforward solution: all processors just send their messages in one direction around the ring and the communication proceeds by pipelining in *P*–1 steps. All the channels are used in all steps. Regarding a bidirectional ring, every node sends its messages into two branches of a primitive broadcast tree and they do one hop in each step. It is easily seen that these broadcast trees of all the nodes are time-arc disjoint (i.e. no channel is used more than once in a single step) [18]. Thus the lower bound can again be reached. In this case, UMDA has not been able to reinvent this algorithm (see Table 8.7).

8.1.3 Meshes and Tori

Meshes and tori are also very popular topologies. Meshes can be easily manufactured on a chip due to local interconnections only, however they have other disadvantages. The main one is a lack of node symmetry and regularity as the node degree is not constant. While the corner nodes can only use 2 external links, the nodes on the boundary are equipped with 3 links and internal nodes even with 4 links. Therefore, one-to-all schedules for all-port meshes ($k = d$) will need more or less steps, respectively. This is a big difference in comparison to node symmetric tori networks containing wrap-around links.

As far as 2D meshes are concerned, the dimension-ordered deterministic routing (first in x, then in y dimension) on meshes and tori is known to be deadlock-free [131]. A certain degree of adaptivness can be obtained by more relaxed routing, such as North-last or West-first strategy [64].

Table 8.8 confronts the upper bounds achieved by evolution with the mathematically derived ones. There are several square instances of meshes and tori with sizes from 3×3 nodes up to 10×10 nodes in the table. Two cases of rectangular topologies with dimensions 3×4 and 4×8 have also been investigated. Three different values for OAB and OAS are provided in the case of meshes, reflecting three chosen root node positions (left corner, the centre of the upper edge, the centre of the mesh).

During the search for the optimum schedule, it may be necessary to include not only multiple minimum paths, but sometimes even non-minimum ones! Figure 8.1 shows one example - OAS communication in the mesh topology. In order to reach the minimum number of communication steps (the lower bound is 5 steps), 3 messages must be injected into a network in every step by the source node. The last step requires non-minimum routing.

OAB in 1-port 2D-meshes is a relatively easy task: recursive doubling (as on the ring) is done first in x dimension and as soon as all nodes in row x are informed, OAB in all columns is done simultaneously, again by recursive doubling (see Figure 4.8). On the other hand, the development of an optimal OAB algorithm in all-port 2D-meshes is difficult because the lower bound is pretty tight. In order to achieve it, we need an algorithm in which every node, once informed, must find in every subsequent step four uninformed nodes and deliver them the message, so that globally all used paths are link-disjoined. Since meshes, unlike tori, are not node-symmetric, there are no elegant algorithms for them. Here again we have to resort to evolutionary optimization that have produced successful optimal schedules up to 10×10 mesh and 7×7 torus. Surveying the upper bounds presented in Table 8.8, we learn that OAS and OAB do not pose a challenge for UMDA. Therefore, the evolution is able to produce novel, maybe up to now unknown, schedules.

The similar situation is in the design of TADT trees useful for OAS and AAB communications. They are known for square tori, but in the other cases their construction is difficult due to the lack of symmetry. On the contrary, the evolution was very successful here. Meshes up to 8×8 and tori up to 6×6 nodes

Table 8.8. Time complexity of the evolved schedules for 2D meshes.

Topology	OAS	AAS	OAB	AAB
Mesh-3x3	4,3,2	6	2,2,2	4
Mesh-3x4	6,4,3	12	3,2,2	6
Mesh-4x4	8,6,4	16	3,2,2	8
Mesh-5x5	12,8,6	32	3,3,3,	12
Mesh-4x8	16,11,8	64	3,3,3	16
Mesh-6x6	18,12,9	56/54	4,3,3	18
Mesh-7x7	24,16,12	-	4,4,3	24
Mesh-8x8	32,22,16	-	4,4,3	32
Mesh-9x9	40,27,20	-	4,4,4	-
Mesh-10x10	50,33,25	-	5,5,5	-
Torus-3x3	2	4	2	3
Torus-3x4	3	6	2	3
Torus-4x4	4	9/8	2	4
Torus-5x5	6	16/15	2	6
Torus-4x8	8	33/32	3	11/8
Torus-6x6	9	30/27	3	9
Torus-7x7	12	-	3	13/12
Torus-8x8	16	-	3	18/16
Torus-9x9	20	-	4/3	-
Torus-10x10	25	-	4/3	-

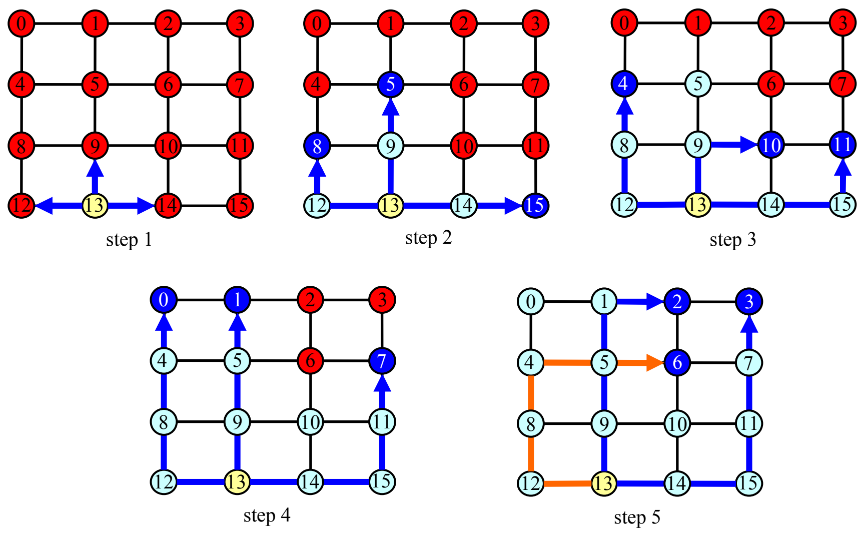


Figure 8.1. Optimal OAS schedule exploiting a non-minimal routing in the fifth step.

were solved without any loss of performance in the schedules. In the case of 7×7 mesh, only one step worse solution was obtained. Moreover, UMDA could elegantly solve rectangular topologies, like mesh-4×8, too.

Finally, the conflict-free AAS schedules are not known even for square tori. AAS schedules were hard nuts to crack, especially in cases of tori. The optimal schedules were designed up to 4×8 (32 node) mesh and only up to 4×3 tori. Rectangular meshes and tori needed more relaxed lower bounds, because due to different *x* and *y* dimensions not all-port could be effectively used.

Many of evolved schedules embodied a one step higher time complexity. The evolution on very large topologies has not been done because the execution time of evolution could simply exceed several days.

Due to the exceeding complexity of the evolved schedules we will not show all-to-all schedules but only one interesting OAB example, see Table 8.9. The root (node no. 0) found two partners in the first step, nodes no. 6 and 13. These nodes located other partners and distributed the message to the other seven nodes in the second steps. Finally, all remaining nodes were informed in the last step.

8.2 Optimal Diameter-Degree Topologies

The evolution has been also applied to several optimal diameter-degree topologies that either already found the commercial application (such as scalable Kautz networks, [188]) or are potential candidates for NoCs (like non-scalable Petersen (P = 10) [88], Heawood (P = 14) [220] or Levi (P = 30) [121]) networks. The potential of the ODD networks has been discussed in section 4.1.2 briefly.

As far as this author knows, performance of collective communications on such networks has not been studied as of yet. The reason may be that until recently [188] these networks have not been used in commercial systems. Consequently, any designed schedule improves the performance of CCs and pushes our knowledge in this area a leap forward.

The results of the UMDA optimization tool are summarized in Table 8.10. Two integers in one cell separated by a slash indicate that the lower bound (a smaller integer) has not been reached whereas a single integer represents both the lower and the upper identical bounds reached by an EA. An asterisk (*) indicates the fact that a non-minimum routing had to be used; otherwise the minimum routing was used everywhere else. From the table below it can be concluded that except for the AAS pattern, the lower bounds have been reached, and therefore cannot be improved any more. However, the upper bounds of AAS are fairly close to the optimum. This is good news for many NoC designers because: (1) ODD networks enable us to make highly compact systems, and (2) we can make them pretty fast now.

In order to show an example of the optimal schedule, the unidirectional oriented Kautz-12 network has been chosen. The resulting schedules are presented

Table 8.9. Optimal OAB schedule on the 4×4 mesh; node 0 which is situated in the left corner is the root.

step 0		step 1		step 2	
from→to	**path**	**from→to**	**path**	**from→to**	**path**
0→6	0,1,2,6	0→9	0,1,5,9	0→1	0,1
0→13	0,4,8,12,13	0→12	0,4,8,12	6→7	6,7
		6→10	6,10	10→15	10,11,15
		6→11	6,7,11	11→3	11,7,3
		13→2	13,9,10,6,2	13→8	13,12,8
		13→4	13,12,8,4	14→5	14,13,9,5
		13→14	13,14		

Table 8.10. Time complexity of the evolved schedules for ODD networks.

Topology	OAS	AAS	OAB	AAB
Petersen-10	3	5	2	3
Kautz-12	4	7	2	4
Heawood-14	5	10/9	2	5
Levi-30	10	31/28	3	10
Kautz-36	12*	34/31	3	12

only for the most complex all-to-all communication patterns in Table 8.11. Some empty slots in the AAS table show that not all links are used in every step of AAS.

The broadcasting sub-trees have been chosen to illustrate the AAB schedule. There are three columns (sub-trees) in the AAB table, because all nodes are equipped by three output channels, hence they can distribute three messages simultaneously (via nodes no. 3, 4 and 5 in the case of source no. 0). The notations 0-3-1 and 0-3-2 denote the following: the message was delivered to node no. 3 in the first communication steps. Then, the message was received by nodes no. 1 and 2 from node no. 3 in the second communication step. Another example, the notation 0-4-x-x-9 denotes the following statements: Node no. 4 was informed by node no. 0 in the first step; then the message did not move two steps; and finally, the message was delivered to node no. 9 by node no. 4 in the fourth communication step.

Let us note that the presented solutions are not unique, several solutions have been found for both AAS and AAB patterns.

8.3 Novel Network Architectures

This section explores the quality of evolved CC schedules for some novel interconnection networks. First, the small K-ring [112] and Midimew topologies with 8 nodes (see section 4.1.3) were examined. This section is mainly targeted to spidergon [102] (also called octagon) and AMP topology [28]. Spidergon is

Table 8.11. Optimal AAS schedule in 7 steps and AAB schedule in 4 steps on the oriented Kautz-12 network.

AAS								AAB			
source	from→to							source	three subtrees broadcasting source message		
	step 1	step 2	step 3	step 4	step 5	step 6	step 7		tree 1	tree 2	tree 3
0	0→9	0→2	0→3	0→6	0→1		0→8	0	0-3-1	0-4-x-x-9	0-5-x-6
		0→4	0→7	0→B					0-3-2	0-4-x-x-A	0-5-x-x-7
		0→5	0→A							0-4-x-x-B	0-5-x-x-8
1	1→7	1→2	1→0	1→3	1→6	1→4		1	1-6-3	1-8-A	1-x-7-0
	1→9	1→A	1→5			1→B			1-6-4	1-8-x-9	1-x-7-x-2
			1→8						1-6-x-x-5	1-8-x-B	
2	2→7	2→1	2→4	2→0	2→5	2→6		2	2-9-8	2-A-5	2-B-1
		2→3	2→9		2→B	2→A			2-9-x-6	2-A-x-4	2-B-x-0
		2→8							2-9-x-7	2-A-x-x-3	
3		3→1	3→0	3→8	3→5	3→2	3→6	3	3-0-4	3-1-8	3-2-9
			3→B		3→9	3→4	3→A		3-0-5	3-1-x-6	3-2-A
						3→7				3-1-x-7	3-2-B
4	4→A	4→0	4→1		4→3	4→2	4→7	4	4-9-6	4-A-x-3	4-B-x-x-0
		4→6	4→8		4→B	4→5			4-9-7	4-A-x-x-5	4-B-x-x-1
						4→9			4-9-x-8		4-B-x-x-2
5	5→4	5→0	5→A	5→7	5→2	5→6	5→1	5	5-7-2	5-x-6-x-3	5-x-8-A
	5→B			5→8		5→9	5→3		5-7-x-x-0	5-x-6-x-4	5-x-8-x-9
									5-7-x-x-1		5-x-8-x-B
6	6→2	6→0	6→1	6→A	6→8	6→3	6→B	6	6-3-x-0	6-4-9	6-x-5-7
	6→5	6→4			6→9	6→7			6-3-x-1	6-4-x-A	6-x-5-8
									6-3-x-2	6-4-x-B	
7	7→0	7→6		7→3	7→8	7→5	7→2	7	7-0-x-4	7-2-x-9	7-x-1-x-6
	7→1			7→9		7→B	7→4		7-0-x-5	7-2-x-A	7-x-1-x-8
	7→A								7-0-x-x-3	7-2-x-B	
8		8→2	8→9	8→1	8→4	8→3	8→0	8	8-9-x-x-6	8-A-3-x-0	8-x-B-1
			8→B	8→7			8→5		8-9-x-x-7	8-A-x-5	8-x-B-2
				8→A			8→6			8-A-x-x-4	
9	9→6	9→7	9→2	9→5	9→0	9→1		9	9-6-x-3	9-7-0	9-8-x-x-A
	9→A		9→4	9→8	9→3				9-6-x-4	9-7-x-1	
					9→B				9-6-x-5	9-7-x-2-B	
A	A→1	A→8	A→3	A→2		A→4	A→0	A	A-3-0	A-4-B	A-5-7
	A→7		A→6	A→5					A-3-x-x-1	A-4-x-9-8	A-5-x-x-6
	A→B			A→9					A-3-x-x-2		
B	B→5		B→0	B→A	B→4	B→1	B→8	B	B-1-6	B-x-0-3	B-x-2-x-9
	B→6		B→2		B→7	B→3	B→9		B-1-x-8	B-x-0-x-4	B-x-2-x-A
									B-1-x-x-7	B-x-0-x-5	

also a novel on-chip network architecture suitable for the aggressive on-chip communication demands of SoCs in several application domains and also for networking SoCs [102] (see Figure 4.4a). As a ring, it is also not free from deadlock and virtual channels have to be used. The AMP topology is a result of genetic graph optimization [28]. A Minimum Path (AMP) [29] configuration is constructed so that the network diameter and the average inter-node distance is minimized. Let us note that AMP is an asymmetric topology.

Table 8.12 compares the time complexity of evolved schedules for particular communication patterns. From the table, UMDA can be concluded to have been able to design optimal schedules for both OAS and OAB in all instances of the proposed topologies. The obtained schedules for AAB on spidergon also reached the lower bounds. In the case of AAS, UMDA tried to approach the lower bounds as close as possible. In some cases it was verified that the difference between lower and upper (reached) bounds was due to the minimum routing strategy used in our approach. Inclusion of the non-minimum routing would have led to an enormous increase of possible source-destination paths and, therefore, was not explored. However, in some small networks the analysis of the last remaining conflict in fitness function revealed that it could have been eliminated if non-minimum routing had been used.

Collective communications on the generic 8-node symmetric spidergon network are easy. One-to-all communications are done the same way for every source node. OAS can be clearly done in three steps and OAB needs two steps. In order to implement AAB, we have to use such a broadcasting tree that is time-arc-disjoint (TADT) and can be used by all nodes simultaneously without creating conflict. The same tree as well as for store-and-forward switching can be used, restricting communication in each step to only between neighbors. The most complex AAS communication is not performed the same way by all nodes - there is no analogy to the TADT. The optimum AAS schedule is given in Table 8.13.

After checking of AAB results on AMP topology, it could be concluded that UMDA does not offer very good results (e.g. two steps worse schedule for 23 nodes). This problem has been hardly studied. It originates in the asymmetrical nature of AMP where there are some bottlenecks and hotspots that make it impossible to design an optimal schedule. Consequently, the lower bound cannot ever be found. Let us note that this can be a problem of more topologies, not only AMP. The theoretical lower bounds tell us only the lowest possible time complexity of the schedule for a particular CC and topology, but they do not tell us anything about whether or not it is possible to reach them! On the other hand, UMDA surprisingly designed optimal AAS schedules for all AMP instances.

Since AMP is an asymmetrical network, no systematic approach to schedule CC exists so the best ever known schedules are presented. The optimal schedules for AAB and AAS have been shown in Figure 3.7 and Figure 3.9 in section 3.4.1 and 3.4.2, respectively.

8.4 Multistage Topologies

This section is oriented to multistage interconnection networks, also called MINs [195], [108]. MINs are indirect networks that means the interconnection networks are composed of several processing nodes (processors) and intermediate switches. More formally, MIN is a succession of stages of switching elements (SEs) and interconnection wires connecting *P* processing (terminal) nodes. SEs in the most general architecture are themselves interconnection

Table 8.12. Time complexity of the evolved schedules for some novel networks.

Topology	OAS	AAS	OAB	AAB
K-Ring-8	2	3	2	2
Midimew-8	2	3	2	2
Spidergon-6	2	3	2	2
Spidergon-8	3	4	2	3
Spidergon-12	4	9	2	4
Spidergon-16	5	17/16	2	5
Spidergon-20	7	26/25	3	7
Spidergon-24	8	37/36	3	8
Spidergon-28	9	51/49	3	9
Spidergon-32	11	70/64	3	11
Spidergon-36	12	91/81	3	12
Spidergon-64	21	-	3	24
AMP-8	2	3	2	2
AMP-23	6	14	2	8/6
AMP-32	8	22	3	8
AMP-42	11	31	3	14/11
AMP-53	13	46	3	14/13

Table 8.13. Optimal AAS schedule on the 8-node all-port spidergon.

source	time step/from → to (path)			
	step 1	step 2	step 3	step 4
0	0→3 (0,7,3)	0→2 (0,1,2)	0→1 (0,1)	0→4 (0,4)
		0→7 (0,7)		0→5 (0,1,5)
				0→6 (0,7,6)
1	1→4 (1,0,4)	1→0 (1,0)	1→2 (1,2)	1→3 (1,2,3)
	1→6 (1,5,6)		1→5 (1,5)	
			1→7 (1,0,7)	
2	2→1 (2,1)	2→5 (2,6,5)	2→4 (2,3,4)	2→0 (2,1,0)
	2→3 (2,3)		2→6 (2,6)	
	2→7 (2,6,7)			
3	3→0 (3,4,0)	3→1 (3,2,1)	3→2 (3,2)	3→5 (3,4,5)
		3→4 (3,4)		3→6 (3,2,6)
				3→7 (3,7)
4	4→2 (4,3,2)	4→1 (4,5,1)	4→0 (4,0)	4→3 (4,3)
	4→5 (4,5)	4→7 (4,3,7)	4→6 (4,5,6)	
5	5→2 (5,1,2)	5→4 (5,4)	5→3 (5,4,3)	5→0 (5,4,0)
		5→7 (5,6,7)		5→1 (5,1)
				5→6 (5,6)
6	6→4 (6,5,4)	6→3 (6,2,3)	6→0 (6,7,0)	6→2 (6,2)
			6→1 (6,2,1)	6→5 (6,5)
				6→7 (6,7)
7	7→1 (7,0,1)	7→4 (7,0,4)	7→5 (7,6,5)	7→0 (7,0)
	7→2 (7,6,2)	7→3 (7,3)		
		7→6 (7,6)		

networks of small sizes. If P is the number of terminal nodes (the MIN's degree) and k is the SE's degree (the number of input/output ports), the minimum number of switches in a stage must be P/k.

The interconnection pattern or patterns between MIN's stages can be represented mathematically by a set of functions. Examples of such topologies cover an unidirectional Omega [119] (a perfect-shuffle permutation) and Butterfly [102] (butterfly permutation) networks. Figure 8.2 presents the 8-node Omega and Butterfly networks. Terminal nodes are shown in green and switching elements in red boxes. In both cases, eight input-output ports are interconnected by three stages of 2×2 switches. It is easy to see that a single pass through the three stages allows any input port to reach any output port.

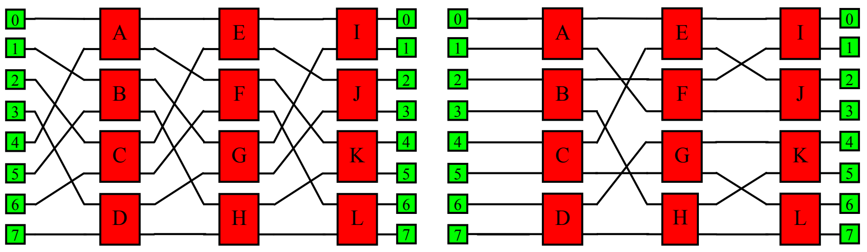
The main disadvantage of permutation-based MINs is their zero fault-tolerance and high blocking probability. In order to alleviate the bottleneck consisting in only a single path between an input-output pair, the multipath Clos network has been proposed [210]. Here, each network input-output pair can be connected by a path via an arbitrary middle stage. The basic version of a Clos network consists of three SE stages, as shown in Figure 8.3a. Clos networks of more than three stages emerge by substituting again the middle stage SEs by Clos network.

The MINs described so far have unidirectional network links, but bidirectional forms are easily derived as two MINs back-to-back, folded on one another (see Figure 8.3b). The overlapping unidirectional links run in different directions, thus forming bidirectional links, and the overlapping switches merge into a single switch with twice the ports (i.e. 4×4 switch). A representative of this class is a Fat-tree [120] topology which originates in the two folded Butterfly network. Unlike traditional trees in computer science, fat trees resemble real trees because they get thicker near the root.

Let us carefully look at investigated MINs. There are two different types of nodes in these networks. We have to realize that SE nodes can only pass on the message from their input to output and are not able to consume, create or modify the message. Only terminal nodes can do this. The situation is similar to many-to-many communication patterns, when only terminal nodes (the green ones in Figure 8.3) are involved in communication so that the chromosomes include only terminal nodes (see Figure 6.4)

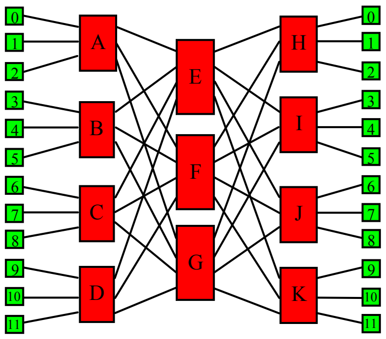
The UMDA has been applied to several MINs that already found the commercial application such as Omega, Butterfly and Clos networks. The bidirectional MINs were represented by binary and fat trees where terminal nodes were placed only in leaves. This study was completed by a full binary tree, where every node represents one processing node.

First, we verified the ability of the UMDA to discover optimal communication schedules for unidirectional MINs (see Table 8.14). Two integers in one cell separated by a slash indicate that the lower bound (a smaller integer) has not been reached, whereas a single integer represents both the lower and the upper identical bounds reached by the UMDA. Obtained schedules for 8-node Omega

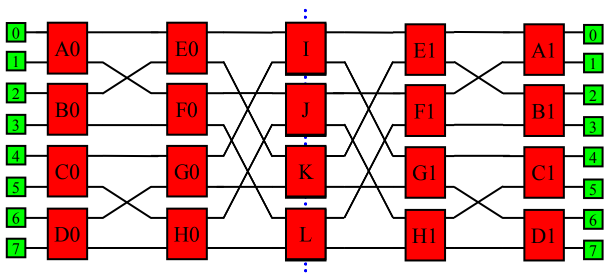


(a) Omega network (b) Butterfly network

Figure 8.2. Illustration of the 8-node Omega and Butterfly networks.



(a) Clos network



(b) unfolded Butterfly network (Fat tree)

Figure 8.3. General form of the Clos network and Fat tree topology.

and Butterfly have met the theoretical lower bound for all classes of collective communications, and thus cannot be improved anymore. The limit of simultaneously executable transfers was reached by 12-node and 16-node topologies. For the successful accomplishment of all-to-all communications, UMDA had to add one additional communication step to the theoretically derived value. The Clos network embodies the same problem leading also in one step addition.

Second, we investigated the ability of the proposed UMDA to discover optimal communication schedules for bidirectional MINs represented by the binary (B-Tree), fat (Fat-Tree), and full binary tree (Full-Tree). Binary trees represent suitable interconnection networks for chip multiprocessor because they need only a very simple link arrangement on a 2D silicon chip. However, as we can see in Table 8.14, their performance rapidly decreases with the number of connected processing nodes. More importantly, the proposed UMDA is able to find optimal communication schedules for most tested binary trees and investigated communication patterns. As can be seen from Table 8.14, the lower bounds for the AAB pattern are too tight. Consequently, it has not been possible to find optimal solutions. Moreover, there is no certainty that it will be even possible to reach them because the controlled flood does not work for multistage networks.

The fat tree topology eliminates the bottleneck of the narrowing bandwidth towards the root. The height of the tree remains the same, but the number of bidirectional links proportionally increases. In this case, UMDA created optimal schedules for fat trees with 4, 8, 16, and 32 leaves, except for AAS on the 32-leave fat tree. Here, only suboptimal solutions with one step worse time complexity were searched.

Finally, the experimental work was completed with the full binary tree, where all switching elements integrate a processing unit too. Since the full binary tree is an asymmetrical topology, several different situations depending on a level of a source node (from a leave to the root) have been investigated. In all cases, the theoretical lower bounds were reached.

An example of a designed optimal AAS schedule for the 8-leave fat-tree topology is shown in Table 8.15. There is a complete path from source to destination via a turning node (sub-tree root) in every cell. The first row in a cell represents the movement up from the source to the sub-tree root. The second row represents the movement down from the sub-tree root to the destination node. Let us note that since interconnection links are bidirectional, the messages do not have to go through all intermediate stages. Finally, it should be mentioned that the presented communication schedule is not unique: several optimal schedules can be found for a given CC.

8.5 Fat Topologies

One way on how to increase the number of interconnected terminal nodes and not make the network more complex is to use a concept of fat topologies. The idea is very simple: all terminal nodes are replaced by multiprocessor nodes, so that more than one CPU core is connected by the shared router to the interconnection network. For instance, we can divide 12 CPU cores in spidergon network among 12, 6 or 4 multiprocessor nodes with 1, 2 or 3 CPU per a node,

Table 8.14. Time complexity of the evolved schedules for unidirectional and bidirectional MINs.

Topology	OAS	AAS	OAB	AAB
Omega-8	7	7	3	8
Omega-16	15	16/15	4	16/15
Butterfly-8	7	7	3	7
Butterfly-16	15	16/15	4	16/15
Clos-8	11	12/11	4	12/11
Clos-16	15	16/15	4	16/15
B-Tree-4	3	4	2	3
B-Tree-8	7	16	3	8/7
B-Tree-16	15	64	4	20/15
B-Tree-32	31	256	5	64/31
Fat-Tree-4	3	3	2	3
Fat-Tree-8	7	7	3	7
Fat-Tree-16	15	15	4	15
Fat-Tree-32	31	32/31	5	31
Full-Tree-7	6,4,3	12	3,2,2	7
Full-Tree-15	14,12,8,7	56	3,3,3,3	15
Full-Tree-31	30,28,24,16,15	240	4,4,4,4,4	31
Full-Tree-63	62,60,56,50,48,32	992	5,5,5,5,5,5	64

Table 8.15. Optimal AAS schedule on the 8-node fat tree.

source	path [from, via, turning point (sub-tree root), via, to]						
	step 1	step 2	step 3	step 4	step 5	step 6	step 7
0	0,A,F B,2	0,A,E,I G, D,7	0,A,F,L H,C,5	7,D,G,K E,B,3	0,A,F,L H,D,6	0,A,E B,3	0,A 1
1	1,A,E,K G,C,5	1,A,F B,3	1,A,E,K G,D,6	1,A,E,K G,D,7	1,A,E,I G,C,4	1,A 0	1,A,E B,2
2	2,B,F,L H,D,6	2,B,E,K G,C,5	2,B,E,I G,C,4	2,B,E A,1	2,B 3	2,B,F,L H,D,7	2,B,E A,0
3	3,B,E,I G,D,7	3,B,F,J H,D,6	3,B,F A,1	3,B,F A,0	3,B 2	3,B,E,I G,C,5	3,B,F,L H,C,4
4	4,C,G,K E,B,3	4,C,H,J F,A,1	4,C,H D,7	4,C,H D,6	4,C,H,L F,A,0	4,C,H,J F,B,2	4,C 5
5	5,C 4	5,C,G,K E,B,2	5,C,G,I E,A,0	5,C,G,I E,B,3	5,C,G,I E,A,1	5,C,G D,6	5,C,G D,7
6	6,D,H,J F,A,1	6,D,G,I E,A,0	6,D,H,L F,B,2	6,D,G C,5	6,D 7	6,D,H C,4	6,D,H,L F,B,3
7	7,D,G,I E,A,0	7,D,H C,4	7,D,G,K E,B,3	7,D,H,L F,B,2	7,D,H C,5	7,D,G,K E,A,1	7,D 6

respectively. In this case, only a one-port model is usually assumed. As microelectronic technology is turning to multicore and multi-threaded processors for more performance and less power consumption, networks interconnecting such fat nodes are of interest (e.g. fat hypercube has been recently used in SGI Origin 3000 machine [185]). Also, Opteron processors produced by AMD are equipped by HyperTransport links on the chip ready for a fat cube connection [103]. Fat nodes (8 CPUs per node) have been used in a Swiss-T1 cluster with a K-ring network [56].

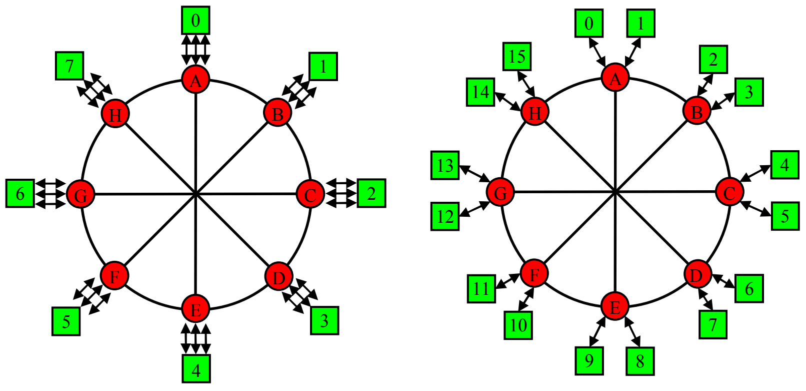
Figure 8.4 illustrates the way the network gets fat. The direct all-port slim spidergon is transformed into the indirect one-port fat spidergon with the same structure of switching elements (routers), but with twice as many interconnected terminal nodes. The technique of fat interconnection networks has several advantages over traditional networks: (1) it makes some small networks more scalable, even though the interconnection graph of a network is not scalable at all (e.g. OOD networks, see section 4.1.2) or even partially scalable (spidergon, hypercube, AMP); (2) it provides in many cases cheaper network implementation in terms of hardware cost and is more often suitable for networking systems on a chip; (3) the performance in collective communications is only a little worse than that of base networks, but it can be controlled by a multiplicity of links (see the fat tree topology, Figure 8.3b) and by overlapping local and global communications.

Table 8.16 summarizes the upper bounds attained on a 2-fat one-port fat hypercube and 2-fat, 3-fat and 4-fat spidergons. The term 2-fat implies two CPUs connected to a switching element. The notation 2-fat-hypercube-8 represents a 3D hypercube with 8 switching elements connected by three input/output links and 16 terminal nodes, two per each switch. The construction of an input file for the evolution algorithm is straightforward. A slim topology is taken, all terminal nodes are replaced by switches changing the operation mode from *T*, *R* or *B* to *N*. Two new terminal nodes are then connected to the original one by adding edges to and from it (see section 6.1). The technique is actually the same to the case of indirect topologies and the many-to-many communications.

It should be noted that neither lower bounds for one-port nor for all-port model apply here. The reason is that we cannot assign external network ports of a node explicitly to internal cores. Let us also note that the optimal schedules are not known for the fat spidergon networks so far. Accordingly, Table 8.16 presents the best know upper bounds of CCs for the fat spidergon networks found by evolution. The upper bounds are close to the double of the basic topologies - sometimes better sometimes worse, compare Table 8.16 with Table 8.12 and Table 8.1.

An example of a designed optimal AAB on the 2-fat-hypercube-4 is shown in Table 8.17. There is shown the complete path from source terminal node via some switches to the destination terminal node in every cell. The terminal nodes are denoted by numbers from 0 to 7, switching nodes are denoted by letters from *A* to *D*. It can be seen that the schedule follows the technique of a modified controlled flood (i.e. no message is transported further than 3 hops).

Let us note that for fat topologies and AAB, the controlled flood mechanism has to be modified because the switch nodes are not able to maintain any message, but only interconnect their inputs to outputs and, therefore, cannot be used as distributors. Consequently, the modified flood mechanism should take into account only the terminal nodes. In the case of direct slim topologies, the



(a) all-port slim spidergon (b) one-port fat spidergon

Figure 8.4. Illustration of the all-port slim 8-node spidergon and one-port 2-fat 8-node spidergon.

Table 8.16. Time complexity of the evolved schedules for one-port bidirectional fat hypercubes and spidergons with 2 terminal nodes per switch.

Topology	OAS	AAS	OAB	AAB
2-fat-hypercube-4	7	8	3	7
2-fat-hypercube-8	15	17	4	15
2-fat-hypercube-16	31	36	6	33
2-fat-hypercube-32	63	-	7	-
2-fat-spidergon-6	11	12	4	12
2-fat-spidergon-8	15	17	4	16
2-fat-spidergon-10	19	25	5	20
2-fat-spidergon-12	23	37	5	24
2-fat-spidergon-14	27	50	5	29
2-fat-spidergon-16	31	66	6	37
2-fat-spidergon-18	35	86	6	42
3-fat-spidergon-4	11	11	4	11
3-fat-spidergon-8	23	38	5	24
3-fat-spidergon-12	35	82	6	42
4-fat-spidergon-4	15	16	4	16
4-fat-spidergon-6	23	38	5	24
4-fat-spidergon-8	31	64	6	34

nearest terminal nodes are situated at a distance of 1 (see Figure 8.4a). However, in the case of indirect fat topologies, two switches are put between them. Finally, the nearest terminal nodes not connected to the same switch are situated at a distance of 3 (see the Figure 8.4b). Subsequently, the value of radius *r* has to be increased to a value of at least three to be able to reach a terminal node connected to a neighbor switch (see section 6.5.5).

8.6 Faulty Topologies

Since SoC/NoC systems include many processor cores, interconnection links and other units, their failure rate is much higher than the failure rate of the single-processor system. Generally, there are two kinds of faults in NoC systems: faulty links and faulty nodes. The first one is a damaged link interconnecting two parts of a system. After the faulty link has been located, it has to be excluded from all routing algorithms (CC schedules). After new CCs have been rescheduled, the system is able to work properly with only little loss of performance. A node fault can be imagined as implying that all of that node's communication links are faulty. Of course, the network has to stay connected (i.e. at least one path for each source-destination pairs has to remain).

In this section, we will consider faults to be permanent (defects in manufacturing) as opposed to, say, transient, intermittent or even malicious faults. Our technique supposes that faulty links or nodes have been already detected, and the faulty region has been bordered. Conceptually, the faulty region may be considered as an island of faults in a sea of communication channels and nodes. In the same manner a ship is navigated around an island, it should be feasible to route a message around a faulty region (see Figure 8.5). That can be done using adaptive routing algorithms that re-route paths from source-destination pairs. However, these algorithms achieve only suboptimal results (i.e. possible faster CC schedules can exist, but they are not discovered, whereas only deterministic principles are used for rescheduling). In addition, many of the fault-tolerant adaptive routing algorithms are not deadlock free and this introduces additional delays and congestions. Moreover, networks in a faulty state are neither symmetrical nor regular, and analytic methods for scheduling do not exist. In order to relax all these restrictions, our evolutionary based technique has been applied for finding CCs schedules on faulty networks.

The proposed evolutionary technique has been applied to two networks that had already found the commercial application such as scalable 12-node Kautz networks [188] and the well known 4×4 2D-mesh.

As the Kautz network is known for its fault tolerance, performance degradation under a single link fault has been tested. A fault diameter of the Kautz12 network is D+2, meaning that among multiple links between any two nodes the longest path is 4. The network performance under a single link fault is given in Table 8.18 (with node no. 0 as the root of OAB and OAS), but the network could operate even under a double link fault. In any case, when the link fault has been detected, a new schedule can be created in 20 seconds on a single processor and then the cluster can continue with a lower performance (see Table 8.22).

The 2D meshes are very suitable for the interconnection network for System on Chips (SoC) because they need only a very simple link arrangement on a 2D silicon chip. For the 4×4 2D mesh, performance degradation under a single link fault and a single node fault were tested. Table 8.19 shows the performance degradation under a single link fault. The source node of OAB and OAS was appointed node no. 0. Any link fault in the 4×4 2D mesh increased the time

Table 8.17. Optimal AAB schedule on the 2-fat-hypercube-4.

source	**from→ via →to**						
	step 1	**step 2**	**step 3**	**step 4**	**step 5**	**step 6**	**step 7**
0	0,A,B,2	2,B,D,7	0,A,C,5	0,A,B,3	0,A,1	5,C,D,6	
					7,D,C,4		
1	1,A,C,4	4,C,D,6	6,D,7		1,A,B,3	7,D,B,2	3,B,A,0
						1,A,C,5	
2	2,B,D,7		2,B,A,1		4,C,5	2,B,A,0	2,B,3
			7,D,C,4		2,B,D,6		
3	3,B,A,0	0,A,C,4	3,B,2	2,B,A,1		3,B,D,7	4,C,5
							7,D,6
4	4,C,D,6	6,D,B,3	4,C,A,0	3,B,2		4,C,A,1	5,C,D,7
				4,C,5			
5	5,C,A,1	1,A,B,2	1,A,B,3	1,A,0	5,C,D,7		1,A,C,4
			5,C,D,6				
6	6,D,C,5	5,C,A,0		6,D,7		0,A,B,3	0,A,1
						6,D,C,4	6,D,B,2
7	7,D,B,3	3,B,A,1		7,D,C,4	3,B,A,0		
		7,D,C,5		5,C,D,6	6,D,B,2		

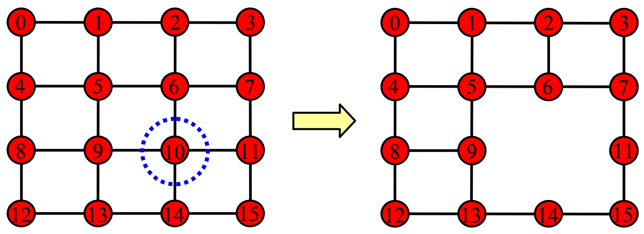


Figure 8.5. Isolation of faulty node no. 10.

overhead of AAS about 37%. The OAS and AAB communication would be delayed twice, but only in two cases, and OAB communication would not be influenced in any way. In all tested link failures, the optimal communication schedules were discovered for a given CC.

Finally, the performance degradation under a single node fault was tested (Table 8.20). A node fault can be thought of as implying that all of that node's communication links are faulty. From this table, the same performance degradation under a single node fault as under a single link fault can be observed.

As the communication schedules are too complicated to be simply demonstrated in the text form, they will be omitted this time around.

Table 8.18. Performance of the Kautz12 network with a single faulty link (in the number of steps).Reduced performance is in bold.

link	OAS	AAS	OAB	AAB
no fault	4	7	2	4
0→3	**6**	**9**	**3**	**6**
0→5	**6**	**9**	**3**	**6**
0→4	**6**	**9**	**3**	**6**
1→7	4	**9**	2	**6**
1→6	4	**9**	2	**6**
1→8	4	**9**	2	**6**
2→B	4	**9**	2	**6**
2→A	4	**9**	2	**6**
2→9	4	**9**	2	**6**
3→0	4	**9**	2	**6**
3→1	**5**	**9**	2	**6**
3→2	**5**	**9**	2	**6**
5→7	4	**9**	2	**6**
all other	4	**9**	2	**6**

Table 8.19. Performance of the 4×4 2D mesh network with a single faulty link (in the number of steps). Reduced performance is in bold.

link	OAS	AAS	OAB	AAB
no fault	8	16	3	8
0→1	**15**	**22**	3	**15**
0→4	**15**	**22**	3	**15**
all other	8	**22**	3	8

Table 8.20. Performance of the 4×4 2D mesh network with a single faulty node (in the number of steps). Reduced performance is in bold.

node	OAS	AAS	OAB	AAB
no fault	8	16	3	8
0	**15**	**22**	3	**15**
4	**15**	**22**	3	**15**
all other	8	**22**	3	8

8.7 Many-to-Many Communications

Besides one-to-all and all-to-all communication patterns, we can also find many communication patterns that are engaged only as a subset of nodes in. Typical occurrences of these CC patterns can be found in algorithms based on recursion, like divide and conquer [198]. For example, one big task is divided into several smaller ones. These smaller tasks can also be divided into subtasks and processed in a parallel fashion. The communication among a subset of nodes participating in a subtask can be globally seen as many-to-many (*M*-to-*N*) communication.

Another situation could be simply imagined. Let us suppose we keep at our disposition an extensive parallel cluster. Unfortunately, our task can effectively employ only a small part of the cluster. From the global point of view, all our all-to-all CCs will be transformed to *M*-to-*N* ones.

The importance of *M*-to-*N* communications also consists in their commonness. Many special patterns can be simply converted to them. Moreover, any CC executed on an indirect (fat, multistage) network can be actually seen as *M*-to-*N* communication, where switching nodes are excluded from the communication. *M*-to-*N* communications can also be beneficial for one-port networks. We can simply transform a direct all-port network to an indirect one-port one by putting switches between the terminal nodes and entry points of the network. It is now possible to handle the CC problem as *M*-to-*N* communications. Moreover, any combination of all-port and one-port nodes can be created.

We have investigated only a couple of possible *M*-to-*N* arrangements, but it can be simply demonstrated that the proposed technique is able to design *M*-to-*N* with any distribution of transmitters and receivers. The topologies of interest were the 8-node all-port hypercube and 8-node all-port spidergon. Four arrangements of transceivers and receivers were investigated for both topologies. In the case of a hypercube, we examined the communication within the same base (4 nodes) and between the bases (nodes on the lower base transmitted the messages to nodes on the upper base). Furthermore, we examined the situation when every node on the lower base sends the message to all other nodes, and finally, the nodes situated on the body diagonal forwarded the messages to the nodes on the lower base. In the case of a spidergon, we investigated these arrangements: within the left half of the spidergon; between left and right half; left half distributing the messages to all the nodes; and the nodes with odd indexes transmitting to the nodes with even indexes. The upper bounds of obtained schedules are summarized in Table 8.21.

The obtained upper bounds are identical to the estimated lower bounds for all the cases. It should be noted that the upper bounds lie between one-to-all and all-to-all bounds. Of course, for more complex topologies, the quality of schedules does not have to be identical to the estimated lower bounds, but in all cases it will be at least the same as in the case of all-to-all communication on a given interconnection network.

It is also important to remember that for most *M*-to-*N* communication, the non-minimal routing has to be used to exploit all of the potential of the interconnection network (all free links). Minimal routing would often lead to only suboptimal solutions. This situation can be simply observed (e.g. in the case of the 8-node spidergon and MNS communication within the left half). If only minimal routing is allowed, the communication takes 3 steps because links from the second half cannot be employed, especially, the links connecting both halves.

Table 8.21. Time complexity of the evolved schedules for the 8-node hypercube and spidergon for a few *M*-to-*N* CC patterns.

Topology	Pattern	MMS	MMB
Hypercube-8	Within the same base	2	2
	Between the bases	4	2
	Lower base to all	4	3
	diagonal to the lower base	2	2
Spidergon-8	Within the left half	2	2
	Left half to right half	3	3
	Left half to all	3	3
	Odd nodes to even nodes	3	2

On the other hand, allowing the non-minimal routing, the communication will save one step. This schedule is shown in Table 8.22. Notice the transfer from node no. 4 to node no. 6. The shortest path goes through node no. 5, but the channel between 5 and 6 is busy, so the longer path has had to be chosen (4, 0, 1, 2, 6) (for a better understanding see Figure 8.4a).

8.8 Summary

This section has analyzed the capabilities of a proposed evolutionary technique to design near optimal CC schedules on almost all types of interconnection networks at great length. It is seen from the results that for the networks of interest in this work, the obtained upper bounds are mostly close or equal to theoretical lower bounds. The only exception is AAS communication in larger networks, where the lower bounds are apparently too tight. In fact, the obtained numerical results have led us to an improvement of theoretical lower bounds for AAS communication (see section 3.4.2). This is a very significant contribution of this thesis.

Of course, the fact that the lower bound may not always be reached by the presented algorithm is to be expected because it may not be attainable in principle by any algorithm. Sometimes lower bounds can be obtained in schedules with non-minimum routing. Unfortunately, inclusion of the non-minimum routing would lead to an enormous increase of possible paths from sources to destinations and to the prohibitive computer memory and time requirements.

Table 8.23 shows average execution times of UMDA during 5 successful runs on several networks. OAS communication is relatively easy; a solution always takes less than one second. For OAB communication, the values are less than one second for simple network topologies. The longest execution time (hypercube-32) is about 41 seconds. Contrary, evolution of a suitable solution for all-to-all communication takes a much longer time, especially for the AAS communication. An exponential increase of the execution time with the network size can be observed. For the most complex topologies, it can easily reach

Table 8.22. Optimal MNS schedule within the left side of the 8-node all-port spidergon.

source	from, via, to	
	step 1	step 2
4	4,5	4,0,1,2,6
	4,0,7	
5	5,6	5,4
		5,6,7
6	6,5	6,2,1,0,4
	6,2,3,7	
7	7,3,4	
	7,0,1,5	
	7,6	

Table 8.23. Execution times of UMDA in seconds, minutes, hours and days (average values during 5 successful runs).

Topology	OAS	AAS	OAB	AAB
Ring -8	<1s	5m6s	<1s	<1s
Uni-Ring-8	<1s	57s	<1s	<1s
Spidergon-8	<1s	2s	<1s	<1s
Petersen-10	<1s	12s	<1s	2s
Kautz-12	<1s	23s	<1s	3s
Heawood-14	<1s	9m17s	<1s	5s
Spidergon-16	<1s	22m36s	3s	1m41s
Levi-30	<1s	1d6h	3s	2h2m
Hypercube-32	<1s	4d5h	41s	28m38s
Kautz-36	<1s	3d5h	20s	9h50m

impracticable values of days. The execution times as such delimitate the area of suitable topologies for the proposed evolutionary approach of CC scheduling.

Let us note that all experiments were realized in sequential manner on IBM Blade servers equipped with 2x dualcore AMD Opteron 275 processors and supplied by 4GB DDR2 RAM at 800 MHz.

The evolved schedules may serve for writing high-performance communication routines for application-specific networks on chip or for development of communication libraries in the case of general-purpose interconnection networks. All the upper bounds achieved by UMDA are summarized in Appendix. For implementation we need to establish synchronization between particular communication steps and upload the routing tables (taken from the schedule) to the switching elements and terminal nodes. The time complexity of real implementation can be then obtained by the usage of the model of wormhole communication (see section 3.1).

9 Conclusions

The importance of point-to-point interconnection networks steadily grows. Many real multiprocessor systems have been based on wormhole interconnection networks (e.g. Tilera Tile [199], Sicortex SC systems [184], IBM Cell processor [91]), and other systems will certainly follow. These networks are directly predestined to force out bus based interconnections from Systems on Chip (SoC) and Networks on Chip (NoC) of many scales due to their simple manufacturability, reliability, fault-tolerance, low latency and high throughput.

One of the key issues in this area is the performance of communications among nodes placed in such a topology. Since any parallel task requires communication and synchronization mechanisms, the communication latency should be minimized while the network throughput should be maximized. Most communication and synchronization mechanisms can be simply converted into collective communications (CC) based on scatter/gather and broadcast/reduce services, for example, the data distribution among nodes can be implemented by one-to-all scatter, the parallel algorithm control by one-to-all broadcast, results collection by all-to-one gather or reduce, barrier synchronization by all-to-all broadcast, etc.

As we could see in chapter 3, the time complexity of a particular collective communication pattern can be estimated from the network parameters. The lower bounds on time complexity can be expressed in terms of the number of synchronized communication steps (time slots). Although, the lower bounds can be obtained relatively easily, the design of such communication schedules meeting the lower bounds is still an outstanding problem because the schedules have to prevent link blocking, deadlock and livelock [37].

Many approaches that implement (near) optimal schedules of CC have been published [174], [128], [12]. Unfortunately, the authors have concentrated either on only a fixed topology, such as hypercube [85] or mesh [10], or have not been able to work with irregular topologies, many-to-many communication patterns, all-port model or non-determinist routing (see chapter 4). A universal approach has not existed up till now.

Therefore, the hypothesis of the thesis has been formulated based on information and experiences collected in chapter 5: *Evolutionary design is able to* *produce optimal or near optimal communication schedules comparable or even better than which have been obtained by a conventional design for the networks sizes of interest. Moreover, evolutionary design reduces many drawbacks of present techniques and invents still unknown schedules for an arbitrary topology and scatter/broadcast communication patterns.*

The design of the proposed evolutionary technique has been detailed in chapter 6. The chapter has described the encodings of schedules, fitness function evaluation and acceleration and restoration heuristics. The chapter has also provided the formal definition of the main components of the technique. Let us remind ourselves that the proposed method is not only restricted to evolutionary algorithms, but any other optimization technique can be employed.

As the evolutionary algorithms are highly sensitive to a suitable parameters setup, a comprehensive study has been accomplished in chapter 7. Three different evolutionary algorithms have been investigated, namely the Standard Genetic Algorithm (SGA) [66], Mixed Bayesian Optimization Algorithm (MBOA) [148] and Univariate Marginal Distribution Algorithm (UMDA) [137]. The results have shown that the most suitable one is the UMDA algorithm. The results have also shown that the evolution is controlled mainly by a mutation operator and that the sufficient population size is relatively small.

Chapter 8 has presented the achievements of the UMDA algorithm. The postulated hypothesis has been completely proved in this chapter. UMDA has reinvented many important optimal scheduling techniques like the recursive doubling [128] on hypercubes, rings, tori or TAD Tree [212] approach for tori and hypercubes. It has also equaled the Ho-Kao algorithm [84] and ST algorithm [147]. Moreover, many unknown optimal schedules have been invented namely for optimal diameter-degree networks, irregular meshes, fat topologies, indirect topologies, and many-to-many communication pattern. If it was not possible to design an optimal schedule, a suboptimal one was found in a reasonable time. All the results are also summarized in appendix.

Since the schedules are designed for the source-based routing technique [39], the wormhole switches [146] can remain without any change. The schedules can be uploaded into processing nodes that can exploit them to perform the communications as fast as possible. The designed schedules may also serve for writing high-performance communication functions for a concrete topology. Consequently, this function can be included into, for example, the well-known OpenMPI [92] library. Otherwise, when a promising topology has been introduced, the proposed technique could be simply applied to this topology and accelerate the communication on it.

The results of this technique has been also published at prestigious conferences in the area of evolutionary algorithms like ACM GECCO in 2007, 2008, 2009, and 2010 or ICES 2008, and in the area of networks on chip like IEEE ICN 2006, 2007, 2009, IEEE ICONS 2010 or IEEE PARALEC 2006. The main parts of the thesis have been accepted as a chapter of upcoming book "Autonomic Networking-on-Chip: Bio-inspired Specification, Development and Verification".

The core of this thesis has actually awarded Honorable Mention at Human-Competitive Competition, London, UK, 2007 and by Jan Hlavička Award for the Best Paper presented at workshop on Computer architectures and diagnostics PAD 2005.

9.1 Contributions of the Thesis

This section summarizes all notable contribution of the thesis and the proposed technique.

(1) The new method for scheduling of collective communication with the prevention of link blocking, deadlock and livelock not limited by a topology or a distribution of cooperating nodes has been proposed.

(2) The thesis has introduced the concept of the main components of most optimization tools. Hence, a lot of optimization tools based on classical artificial intelligence or heuristic methods like hill climbing [115], as well as stochastic methods like simulating annealing [21], particle swarm optimization [104], and artificial ant colonies [46] can be exploited.

(3) The integrity of the technique is given by its capabilities to design optimal or at least sub-optimal schedules for many kinds of network topologies and communication pattern that have never been solved so far. The technique can be applied to

   (a) Regular topologies (e.g. hypercube and torus).

   (b) Irregular topologies (e.g. mesh, AMP).

   (c) Multi-stage topologies (e.g. Omega, Clos).

   (d) Fat topologies of any type (e.g. fat tree, fat hypercube).

   (e) One-port, all-port, and even k-port model (by an additional limitation of port based heuristic).

   (f) Minimal and even non-minimal routing (only for small scale topologies).

   (g) Simplex, full duplex links and even half-duplex (by a simple modification of conflict counting function not to distinguish between channel directions).

   (h) Any distribution of cooperating nodes on the topology (one-to-all, many-to-many, all-to-all).

   (i) Faulty nodes and faulty links (an operational state can be restored after a link or a node fault).

   (j) And many others.

(4) Many optimal communication schedules have been reinvented with the help of the technique implemented into the UMDA algorithm. Let's remember the recursive doubling algorithm or time-arc disjoint broad-

cast trees at least. Very significant success has been in the invention of algorithms that equal the Ho-Kao algorithm.

(5) Many still unknown optimal schedules have been designed by the technique (e.g. for rectangular tori, meshes, spidergons, AMPs, ODD networks, etc.).

(6) The section 3.5 has provided a new look on the lower bounds for many-to-many CCs. They have been classified and estimated there for the first time ever.

(7) The results allow us to rectify the mathematical formula for the lower bound of all-to-all scatter, (see section 3.4.2).

(8) An assessment whether the theoretical lower bounds of CC times are reachable at all or how close we can get to them (multistage topologies, spidergon, etc.) has been done.

For all these reasons this technique was declared as human competitive. It means that produced results are comparable to them produced by an expert.

9.2 Future work

Although the proposed technique covers a large area of the scheduling of collective communication on wormhole networks, there are naturally many other possible extensions of the technique.

An introduction of the channel capacity (throughput) is one of many possible future extensions. Only the channels with the same capacity have been presumed in this work, but in real-world applications the channels can be hierarchized into more levels of different capacity (e.g. backbone links, inter-cluster links, local interconnection, etc). Accordingly, the capacity of the channels should be taken into account in the fitness function to allow more transfers at a time with respect to the channel capacity.

Only scatter or broadcast CCs have been investigated in this work because they are most frequently used. A few additional communication services could be seen in real-world applications. Let us note for example permutation and scan communications or application specific CCs created by a collection of point-to-point communications. Such a CC can perform any possible combination of messages transmissions. The input file structure would have to be modified to be able to handle a general CC. The source-destination pairs of all message transfers would have to be provided to the technique. A corresponding chromosome would proceed from many-to-many scatter ones.

For some networks, it could also be beneficial to use the combining model that was not used to keep the switches as simple as possible in this work. Message combining reduces the total number of messages, making each node sends fewer messages of a larger size. Reducing the number of steps start-ups can improve communication performance in the case of short messages when the start-up delays dominate in CC times. Unfortunately, the combination of messages completely changes the principles of the scheduling and such a new technique would have to be designed.

Another direction for future research could be targeted to an improvement of acceleration and restoration heuristics. Limitations of the technique have been observed mainly at AAS communications, so additional research in this area could be beneficial. Better results could also be obtained by employing a more sophisticated optimization tool, such as particle swarm optimization (PSO) that turns out to be a surprisingly good optimization tool.

References

[1] Advanced Micro Devices, Inc. *Radeon X1800 memory controller, Technology*, White Paper, 2005, URL <http://ati.amd.com/products/radeonx1k/whitepapers /X1800_Memory_Controller_Whitepaper.pdf>.

[2] Agarwal, P. and Breuer, M. A. Some theoretical aspect of algorithmic routing. In *Proceedings of the 14th conference on Design automation table of contents*, IEEE Press, Piscataway, NJ, USA, pages 23-31, 1997.

[3] Al-Hashimi, B. *System-on-chip: next generation electronics*. The Institution of Electrical Engineering, United Kingdom, ISBN-10:0863415520, 2006.

[4] AMD: Advanced Micro Devices. *AMD Direct Connect Architecture*, 2009, URL: <http://www.amd.com/us/products/technologies/direct-connect-architecture /Pages/direct-connect-architecture.aspx>.

[5] Anderson, C., Jones, K., and Ryan, J. A two-dimensional genetic algorithm for the ising problem. In. *Complex Systems*, vol. 5, pages 327-333, 1991.

[6] Bäck, T. *Proceedings of the 7th International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Francisco, 1997.

[7] Baker, J. Adaptive selection method for genetic algorithms. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, N.J., pages 100-111, 1987.

[8] Baluja, S. *Population-Based Incremental Learning: A method for Integrating Genetic Search Based Function Optimization and Competitive Learning*. Carnegie-Mellon Technical report, CMU-CS-94-163, 1994.

[9] Baluja, S. and Davies, S. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proceedings of the International Conference on Machine Learning*, pages 30-38, 1997.

[10] Bamettet, M., et al. *Broadcasting on Meshes with Worm-Hole Routing*, Technical report. TR-93-24, Dept. Computer Science, University of Texas at Austin, 1993.

[11] Bandyopadhyay, S., Kargupta, H., and Wang, G. Revising the gemga: Scalable evolutionary optimization through linkage learning. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, IEEE Press, pages 603-608, 1998.

[12] Banikazemi, M. and Panda, D. K. Can Scatter Communication Take Advantage of Multidestination Message Passing? In *Proceedings of International Conference on High Performance Computing*, Springer-Verlag Berlin Heidelberg, pages 204-211, 2000.

[13] Banikazeni, M. and Dhabaleskwar, P. K. *Efficient Scatter Communication in Wormhole k-ary n-cubes with Multidestination Message Passing*. Technical Report OSU-CISRC-9/96-TR46, 1996.

[14] Barnett, M., Shuler, L., van de Geijn, R., Gupta, S., Payne, D. G., and Watts, J. Interprocessor collective communication library (InterCom). In *Proceedings of the Scalable High Performance Computing Conference*, IEEE Computer Society Press, pages 357-364, 1994.

[15] Bar-Noy, A. and Kipnis, S. Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems. In *Proceedings of the 1992 Symphosium on Parallel algorithms and architectures*, ACM, New York, pages 13-22, 1992.

[16] Beneš, V. E. Mathematical Theory of Connecting Networks and Telephone Traffic, Academic Press, ISBN-10: 0120875500, 1965.

[17] Benson, C. T. and Losey, N. E. *On a Graph of Hoffman and Singleton.* Combinatorial Theory Series. B 11, pages 67-79, 1971.

[18] Bermond, J. C and Fraigniaud, P. Broatcasting and NP-completeness. In *Graph Theory Notws of New York*, pages 8-14, 1992.

[19] Bermond, J.-C. and Delorme C. Strategies for Interconnection Networks: Some Methods from Graph Theory. In *Journal of Parallel and Distributed Computing*, no. 3, pages 433-449, 1986.

[20] Bhuyan, L. N. and Agrawal, D. P. Generalized Hypercube and Hyperbus Structures for a Computer Network. In *IEEE Transaction on Computers* 33 (4), pages 323-333, 1984.

[21] Booker, L. Improving search in genetic algorithms. In Davis, L., editor, *Genetic Algorithms and Simulated Annealing,* Morgan Kaufmann Publishers, San Francisco, 1987.

[22] Borkar, S. iWrap: An integrated solution to high-speed parallel computing. In *Proceeding of Supercomputing'88*, pages 330-339, 1988.

[23] Bose, B., Broeg, B., Known Y., and Ashir, Y. Lee Distance and opological Properties of k-ary n-cubes. In *IEEE Transactions on Computers*, vol. 44, no. 8, pages 1021-1030, August 1995.

[24] Bosman, P. A. N. and Thierens, D. Linkage information processing in distribution estimation algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO-99), pages 60-67, 1999.

[25] Brindle, A. *Genetic algorithms for function optimization*. Ph.D. dissertation, University of Alberta-Edmondton, 1981.

[26] Buhrman, H., Hoepman, J., and Vitányi, P. Optimal routing tables. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, ACM Press, NY, USA, pages 134-142, 1996.

[27] Bui, T. N. and Moon B. R. On multi-dimensional encoding/crossover. In *Proceedings of the 6th International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Francisco, pages 49-56, 1996.

[28] Burgess C. J. and Chalmers, A. G. Genetic algorithms for generating minimum path configurations In *Microprocessors and Microsystems*, vol. 19, no. 1, February 1995.

[29] Burgess, C. J. and Chalmers, A. G. O*ptimum Transputer Configurations for Real Applications Requiring Global Communications*. Technical Report. UMI Order Number: CS-EXT-1995-044, University of Bristol, 1995.

[30] Chau, S. C. and Fu, A. W. C. An optical multistage interconnection network for optimal All-to-All personalized exchange. In *Proceedings of the fourth International Conference on Parallel and Distributed Computing, Applications and Technologies* (PDCAT'2003), pages 292-295, 2003.

[31] Chelius, G. and Fleury, E. *NP-Completeness of ad hoc multicast routing problems*, Research report no. 5665, Institut National de Recherche en Informatique et en Automatique, September 2005.

[32] Cheng, R., and Gen, M. Evolution program for resource constrained project scheduling problem. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, IEEE Press, Piscataway, NJ, pages 736-741, 1994.

[33] Cheng, R., Gen, M., and Tsujimura Y. A tutorial survey of job-shop scheduling problems using genetic algorithms: I. Representation. In *Computer and Industrial Engineering*, vol. 35, no. 2, 1999.

[34] Clos, C. A study of non-blocking switching networks. In *Bell System Technical Journal* 32 (5), pages 406-424, March 1953.

[35] Cohoon, P. and Paris, W. Genetic placement. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 422-425, 1986.

[36] Cooper, G. F. and Herskovits, E. H. A Bayesian method for the induction of probabilistic networks from data. In *Machine Learning*, vol. 9, pages 309-347, 1992.

[37] Dally W. J. and Seitz, C. L. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, vol. C-36, no. 5, pages 547-553, 1987.

[38] Dally, W. J. and Towles, B. Route packets, not wires: on-chip interconnection networks. In *Proceedings of the Design Automation Conference*, Las Vegas, NV, pages 684-689, June 2001.

[39] Dally, W. and Towles, B. *Principles and Practices of Interconnection Networks*. The Morgan Kaufmann Series in Computer Architecture and Design, Morgan Kaufman Publishers, 2004.

[40] Davis, L. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[41] Davis. L. Adapting operator probabilities in genetic algorithm. In *Proceedings of the 3rd international Conference on Genetic Algorithms*, Morgan Kaufman Publishers, San Francisco, pages 61-69, 1989.

[42] De Bonet, J. S., Isbell, C. L. and Viola., P. MIMIC: Finding optima by estimating probability densities. Advances. In *Neural Information Processing Systems*, vol. 9, The MIT Press, Cambridge, 1997.

[43] De Jong, K. A. An Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan, 1975.

[44] Dimopoulos, N. J., Chowdhury, M., Sivakumar, R., and Dimakopoulos, V. Routing in Hypercycles. Deadlock free and backtracking strategies. In *PARLE '92 Parallel Architectures and Languages Europe*, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, vol. 605, pages 973-974, 2006.

[45] Dolter, J. W, Ramanathan, P., and Shin, K. G. Performance Analysis of Virtual Cut-Through Switching in HARTS: A Hexagonal Mesh Multicomputer. In *IEEE Transactions on Computers*, vol. 40, no. 6, 1991.

[46] Dorigo, M. and Stützle, T. *Ant Colony Optimization*. The MIT Press, ISBN-10: 0-262-04219-3, 2004.

[47] Duato, J., Yalamanchili, S. *Interconnection Networks – An Engineering Approach*, Morgan Kaufman Publishers, Elsevier Science, 2003

[48] Esfahanian, A., Ni, L., Sagan, B. On enhancing hypercube multiprocessors. In *Proceedings of the 1988 International conference of Parallel Processing*, pages 86-89, 1988.

[49] Eshelman, L., Mathias, K., and Schaffer, J. Convergence controlled variation. In *Foundations of Genetic Algorithms*, vol.4, Morgan Kaufmann Publishers, San Francisco, 1997.

[50] Eshelman, L. and Schaffer, J. Real-Coded genetic algorithms and interval schemata. In *Foundations of Genetic Algorithms*, vol. 2, Morgan Kaufmann Publishers, San Francisco, pages187-202, 1993.

[51] Fischer, S., Vöcking, B. Adaptive routing with stale information. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, ACM, New York, NY, pages 276-283, 2005.

[52] Fogel, L., Owens, A., and Walsh, M. *Artificial Inteligence Through Simulated Evolution*, Wiley, New York, 1966.

[53] Forrest, S. *Documentation for prisoner's dilemma and norms programs that used the genetic algorithms*. Ph.D. dissertation, University of Michigan-Ant Arbor, 1985.

[54] Fox, G. C. Solving Problems on Concurrent Processors. In *General Techniques and Regular Problems*, vol. 1, Prentice Hall, Englewood Cliffs, NJ, 1988.

[55] Gabrielyan, E. and Hersch, R. D. Efficient Liquid Schedule Search Strategies for Collective Communications. In *Proceedings of the 12th IEEE International Conference on Network ICON 2004*, Singapore, vol. 2, pages 760-766, 2004.

[56] Gabrielyan, R. and Hersch D. Efficient Liquid Schedule Search Strategies for Collective Communications. In *Proceedings of he12th IEEE International Conference on Networks*, Singapore, Vol. 2, November 16-19, pages 760-766, 2004.

[57] Gang, L., Nai-jie, G., Kun, B., Kun, T., and Wan-li, D. Optimal All-to-All Personalized Communication in All-Port Tori. In *Proceedings of World Academy of Science, Engineering and Technology*, vol. 10, ISSN 1307-6884, December 2005.

[58] Garey, M. R. and Johnson, D. S. Computer and Intractability, A Guide to the Theory of NP-Completeness. Freeman, 1979.

[59] Gen, M. and Cheng, R. A survey of penalty techniques in genetic algorithms. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, IEEE Press, Piscataway, NJ, pages 804-809, 1996.

[60] Gen, M. and Cheng, R. Genetic algorithms and engineering optimization. In *Engineering in Design and Automation*, Wiley Series, ISBN 0-741-31531-1, 2000.

[61] Geoffray, P. and Hoefler, T. Adaptive Routing Strategies for Modern High Performance Networks. In *Proceedings of the 6th IEEE Symposium on High Performance Interconnects*, pages165-172, 2008.

[62] Gill, S. Parallel Programming. In: *The Computer Journal*, vol. 1, pages 2-10, April 1958.

[63] Gillies, A. *Machine learning procedures for generating image domain feature detectors*, Ph.D. dissertation, University of Michigan-Ant Arbor, 1985.

[64] Glass, C. J. and Ni. L. The turn model for adaptive routing. In *Symposium on Computer Architecture*, pages 278-287, 1992.

[65] Goldberg, D. E. and Linge, R., Alleles, loci, and the traveling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications*, Lawrence Erlbaum, Hillsdale, New Jersey, pages 154-159, 1985.

[66] Goldberg, D. E. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, MA, 1989.

[67] Goldberg, D. *Genetic Algorithm in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[68] Goldberg, D. and Richardson, J. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, N.J., pages 41-49, 1987.

[69] Goldberg, D. and Deb, K. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, San Francisco, pages 69-93, 1991.

[70] Goldberg, D., Korb, B., and Deb, K. Messy genetic algorithms: motivation, analysis and first results. In *Complex Systems*, vol. 3, pages 493-530, 1989.

[71] Gonzalez, C., Lozano, J., and Larrañaga, P. Analyzing the PBIL algorithm by means of discrete dynamical systems. In *Complex Systems*, vol. 4, no. 12, pages 465-479, 2001.

[72] Grama, A., Gupta, A., Karypis, G., and Kumar, V. *Introduction to Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc, Second edition, ISBN-10: 0-201-64865-2, 2003.

[73] Gray, F. *Pulse Code Communication*, U.S. Patent 2 632 056, 1953.

[74] Grefenstette, J. Optimization of control parameters for genetic algorithms. In *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1), pages 122-128, 1986.

[75] Grefenstette, J. and Baker, J. How genetic algorithms work: a critical look at implicit parallelism. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Francisco, pages 20-27, 1989.

[76] Groth, D. and Toby, S. *Network+ Study Guide, Fourth Edition*. Sybex, Inc., ISBN 0-7821-4406-3, 2005.

[77] Hancock, P. An empirical comparison of selection methods in evolutionary algorithms. In *Evolutionary Computing*, Springer-Verlag, Berlin, pages 80-95, 1994.

[78] HantaFo, Z. and Haiou, S. Exact Algorithms for MAX-SAT. In *Electronic Notes in Theoretical Computer Science,* no. 1, 2003.

[79] Harik, G. *Linkage learning via probabilistic modeling in the ECGA*. IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, Illinois, 1999.

[80] Harik, G. R., Lobo, F. G., and Goldberg, D. E. The compact genetic algorithm. In *IEEE Transactions on Evolutionary Computation*, pages 523-528, 1999.

[81] Hemani, A., Jantsch, A., Kumar, S., Postula, A., Oberg, J., Millberg, M., and Lindqvist, D. Network on a chip: An architecture for billion transistor era. In *IEEE NorChip*, 2000.

[82] Hennessy, J. L. and Patterson, D. A. *Computer Architecture - A Quantitative Approach*. 4th Edition, Morgan Kaufman Publishers, Inc., 2006.

[83] Hiranandani S., Kennedy, K., and Tseng, C. W. Compiling Fortran D for MIMD Distributed-Memory Machines. In *Communications of the ACM*, pages 66-80, 1992.

[84] Ho, C.-T. Optimal broadcasting on SIMD hypercubes without indirect addressing capability. In *Parallel and Distributed Computing*, vol. 13, no. 2, pages 246-255, October 1991.

[85] Ho, C.-T. and Raghunath, M. T. Efficient communication primitives on hypercubes. In Concurrency: *Practice and Experience*, no. 6, pages 427-457, 1992.

[86] Ho, R., Mai, K., and Horowitz, M. The future of wires. In *Proceedings of the IEEE*, vol. 89, issue 4, pages 490-504, 2001.

[87] Holland, J. H. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[88] Holton, D. A., Sheehan, J. *The Petersen Graph*, Cambridge University Press, ISBN 0-521-43594-3, 1993.

[89] Huang, C. C. and McKinley, P. K. Communication issues in parallel computing across ATM networks. In *IEEE Parallel and Distributed Technology*, vol. 4, pages 73-86, 1994.

[90] Hwang, K. and Kim, D. Generalization of orthogonal multiprocessor for massively parallel computation. In *Proceedings of the 2nd Frontiers MPC*, pages 391-398, October 1988.

[91] IBM Corporation. *The CELL processor at IBM research*, 2009, URL: <http://www.research.ibm.com/cell/>.

[92] Indiana University. *Open MPI: Open Source High Performance Computing*, 2009, URL: <http://www.open-mpi.org/>.

[93] Intel. *Intel Nehalem-EX processor*, 2009: URL: <http://www.intel.com/pressroom/archive/releases/20090526comp.htm>.

[94] Intel. *Tera-scale research program,* 2009. URL: <http://techresearch.intel.com/articles/Tera-Scale/1421.htm>.

[95] ISO OSI model. In ISO/IEC standard 7498-1:1994, URL: <http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip>.

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

[97] Jaffe, J. M. Distributed multi-destination routing: The constraints of local information. In *Proceedings of the first ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, ACM New York, NY, USA, pages 49-54, 1982.

[98] Jantsch, A., Tenhunen, H. (Eds.), *Networks on Chip,* Kluwer Academic Publishers, 2003.

[99] Joseph Culberson. *Graph Coloring Resources Page*, 2004, URL: <http://webdocs.cs.ualberta.ca/~joe/Coloring/>

[100] Jung, J. P. and Sakho, I. A Methodology for Devising Optimal All-port All-to-all Broadcast Algorithms in 2-Dimensional Tori. In *Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks (LCN'03)*, pages 558-566, 2003.

[101] Karguptam H. and Goldberg D, E. Search, blackbox optimization, and sample complexity. In *Foundations of Genetic Algorithms 4*, Morgan Kaufmann, San Mateo, CA, 1997.

[102] Karim, F. and Nguyen, A. An Interconnect Architecture for Networking Systems on Chips. In *IEEE Micro*, pages 36-45, 2002.

[103] Keltcher C. N. The AMD Opteron Processor for Multiprocessor Servers. In *IEEE Micro*, pages 66-76, 2003.

[104] Kennedy, J. and Eberhart, R. C. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, Piscataway, NJ., pages 1942-1948, 1995.

[105] Khonsari, A., Shahrabi, A., Ould-Khaoua, M., and Sarbazi-Azad, H. Performance comparison of deadlock recovery and deadlock avoidance routing algorithms in wormhole-switched networks. In *IEEE Proceedings Computers and Digital Techniques*, ISSN: 1350-2387, pages 97-106, 2003.

[106] Kim, M. M., Davis, J. D., Oskin, M., and Austin, T. Polymorphic On-Chip Networks. In *SIGARCH Compututer. Archititecture News*, vol. 36, issue 3, pages 101-112, 2008.

[107] Koza, J. R. *Genetic Programming*, MIT Press, Cambridge, MA, 1992.

[108] Kruskal, C. P. and Snir, M. The performance of multistage interconnection networks for multiprocessors. In *IEEE Transactions on Computers*, vol. 32, issue 12, pages 1091-1098, 1983.

[109] Kullback, S. and Leibler, R. A. On information and sufficiency. In *Annals of Mathematical Statistics*, vol. 22, pages 79-86, 1951.

[110] Kumar, S., Jantsch, A., Soininen, J.-P., Forsell, M., Millberg, M., Oberg, J., Tiensyrja, K., and Hemani, A. A network on chip architecture and design methodology. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI*, pages 105-112, 2002.

[111] Kumar, V., Grama, A., Gupta, A., and Karypis, G. *Introduction to parallel computing: design and analysis of algorithms*. Benjamin/Cummings Press, 1994.

[112] Kuonen, P. The K-Ring: a versatile model for the design of MIMD computer topology. In *Proceedings of the High-Performance Computing Conference* (HPC'99), San Diego, USA, pages 381-385, 1999.

[113] Kurose, J., F. and Ross, K., W. *Computer Networking: A Top-Down Approach,* Addison-Wesley, ISBN 0-321-49770-8, 2007.

[114] Kuszmaul, B. C. Fast, Deterministic Routing, on Hypercubes, Using Small Buffers. In *IEEE Transactions on Computers*, IEEE Computer Society Washington, DC, USA vol. 39, no. 11, pages 1390-1393, 1990.

[115] Kvasnicka, V., Pelikan, M. and Pospichal, J. Hill climbing with learning (An abstraction of genetic algorithm). In *Neural Network World*, vol. 6, pages 773–796, 1996.

[116] Lam, C. C, Huang, C. H., and Sadayappan, P. Optimal Algorithms for All-to-All Personalized Communication on Rings and Two Dimensional Tori. In. *Journal of Parallel and Distributed Computing*, no. 43, pages 3-13, 1997.

[117] Larrañaga P. and Loazano J. A. *Estimation of Distribution Algorithms, A New Tool for Evolutionary Computation*, Kluwer Academic Publishers, pages 59-100, 2002.

[118] Lau, C. M. and Chen, G. Optimal Layouts of Midimew Networks. In *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 9, pages 954-961, 1996.

[119] Lawrie, D. A. Access and alignment of data in an array processor. In *IEEE Transactions on Computers*., vol. 24, pages 1145-1155, 1975.

[120] Leiserson, C. E. Fat-Trees: Universal Networks for Hardware Efficient Supercomputing. In *IEEE Transactions on Computers*, vol. C-34, pages 892-901, 1985.

[121] Levi, F. W. *Finite geometrical systems*. Calcutta, 1942.

[122] Li, J. Aickelin, U. A Bayesian optimization algorithm for the nurse scheduling problem. In *Proceedings of the Congress on Evolutionary Computation (CEC-2003)*, pages 2149-2156, 2003.

[123] Liao, W-K. and King, Ch-T. Valved routing: efficient flow control for adaptive nonminimal routing in interconnection networks. In *IEEE Transactions on Computers*, vol. 44, no. 10, pages 1181-1193, 1995.

[124] Ludvig, J., Hesser, J. and Manner, R. Tracking the representation problem by stochastic averaging. In *Proceedings of the 7th International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Francisco, pages196-203, 1997.

[125] Mao, W., Chen, J., and Watson, W. One-to-all personalized communication in torus networks In *Proceedings of the 25th IASTED International Multi-Conference: parallel and distributed computing and networks*, ACTA Press, Anaheim, CA, pages 291-296, 2007.

[126] Marascuilo, L. A. and McSweeney, M. *Nonparametric and distribution free methods for the social sciences*. Brooks/Cole Publishing Company, CA, 1977.

[127] Matsutani, H., Koibuchi M., and Yamada, Y. Non-Minimal Routing Strategy for Application-Specific Networks-on-Chips. In *Proceedings of the 2005 International Conference on Parallel Processing Workshops*, IEEE Computer Society, Washington, DC, USA, pages 273-280, 2005.

[128] McKinley, P. K. andTrefftz, C. Efficient Broadcast in All-Port Wormhole-Routed Hypercubes. In *Proceedings of International Conference on Parallel Processing*, vol. 11, pages 288-291, 1993.

[129] McKinley, P. K., Tsai, Y. J., and Robinson, D. Collective communication in wormhole-routed massively parallel computers. In *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 2, pages184-190, 1996.

[130] McKinley, P. K., Xu, H., Esfahanian, A.-H., and Ni, L. M. Unicast-Based Multicast Communication in Wormhole-Routed Direct Networks. In *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 12, pages 1254–1265, 1994.

[131] Message Passing Interface Forum. *Document for Standard Message-Passing Interface*, Technical Report CS-93-214, University of Tennessee, 1993.

[132] Michalewicz Z. A survey of constraint handling techniques in evolutionary computation methods. In *Proceedings of the 4th Annual Conference on Evolutionary Programming*, MIT Press, Cambridge, MA, pages 135-155, 1995.

[133] Michalewiz, Z. *Genetic Algorithm + Data Structure = Evolution Programs*, 3rd edition, Springer-Verlag, New York, 1996.

[134] Miller, M. and Širáň, J. Moore graphs and beyond: A survey of the degree/diameter problem. In *The Electronic Journal of Combinatorics*, 2005.

[135] Mitchell, M. *An introduction to genetic algorithms*. MIT Press, Cambridge, MA, 1996.

[136] Moon, B. R. and Kim, C. K. A two-dimensional embedding of graphs for genetic algorithms. In *Proceedings of the 7th International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Francisco, pages 204-211, 1997.

[137] Mühlenbein, H. and Paaß, G. From recombination of genes to the estimation of distributions I. Binary parameters. In *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature – PPSN IV*, pages 178-187, 1996.

[138] Mühlenbein, H., Mahnig, T., Rodriguez, A. O. Schemata, distributions and graphical models in evolutionary optimization. In *Journal of Heuristics*, vol. 5, pages 215-247, 1998.

[139] Mühlenbein, H., The equation for response to selection and its use for prediction. In *Evolutionary Computation*, vol. 5, no. 3, pages 303–346, 1997.

[140] Munafo, R., *The Diameter-Degree Problem*, 2008, URL: <http://www.mrob.com /pub/math/ttl-problem.html>.

[141] Murty, K. *Linear and Combinatorial Programming*. John Wiley, New York, 1976.

[142] Naijie, G. U. Efficient Indirect All-to-All Personalized Communication on Rings and 2-D Tori. In *Journal of Computer Science and Technology*, vol. 16, no. 5, pages 480-483, 2001.

[143] Najaf-abadi, H. H. and Sarbazi-Azad, H. An empirical performance analysis of minimal and non-minimal routing in cube-based OTIS multicomputers. In *Journal of High Speed Networks*, IOS Press, vol. 16, no.2, pages 133-155, 2007.

[144] Nayebi, A., Shamaei, A., and Sarbazi-Azad, H. Improving a Fault-Tolerant Routing Algorithm Using Detailed Traffic Analysis. In *Proceedings of High Performance Computation Conference*, Springer-Verlag Berlin Heidelberg, pages 766-775, 2007.

[145] Neubauer, A. Adaptive non-uniform mutation for genetic algorithms. In *Computational Intelligence Theory and Applications*, LNSC,Springer Berlin / Heidelberg, pages24-34, 1997.

[146] Ni, L. M. and McKinley, P. K., A Survey of Wormhole Routing Techniques in Direct Networks. In *Computer*, vol. 26, no. 2, pages 62-76, 1993.

[147] Nupairoj, N. and Ni, L. M. *Benchmarking of Multicast Communication Services*. Technical Report MSU-CPS-ACS-103. Michigan State University, September 1995.

[148] Ocenasek, J. *Parallel Estimation of Distribution Algorithms*. PhD. Thesis, Faculty of Information Technology, Brno University of Technology, Brno, Czech Rep., 2002.

[149] Oliver, I. M., Smith, D. J., and Holland, J. A study of permutation crossover operators on the travelling salesman problem. In *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications*, Lawrence Erlbaum, Hillsdale, New Jersey, pages 224-230, 1987.

[150] Ono, I, Yananyra, M., and Kobayashi, S. A genetic algorithm for job-based order crossover. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, IEEE Press, Piscataway, NJ, pages 547-552, 1996.

[151] Orvosh, D. and Davis, L. Using a genetic algorithm to optimize problems with feasibility constraints. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, IEEE Press, Piscataway, NJ, pages 548-552, 1994.

[152] Pande, P. P., Grecu, C., Ivanov, A., Saleh, R., and De Micheli, G. Design, Synthesis, and Test of Networks on Chips. In *IEEE Design and Test of Computers*, vol. 22, no. 5, pages 404-413, 2005.

[153] Park, J.-Y. L., Lee, S.-K., and Choi, H.-A. *New algorithms for broadcasting in Meshes*. George Washington University, Technical Report GWUIIST-93-03, 1993.

[154] Pegg, Ed Jr., Rowland, T., Weisstein, E. W. Cayley Graph. In *MathWorld-A Wolfram Web Resource*, 2009, URL: <http://mathworld.wolfram.com/Cayley Graph.html>.

[155] Pelikan M. *Hierarchical Bayesian Optimization Algorithm*, Springer-Verlag, Berlin, pages 13-31, 2005.

[156] Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. BOA: The Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, vol. 1, Morgan Kaufmann Publishers, San Fransisco, CA, pp 525-532, 1999.

[157] Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. *Linkage problem, distribution estimation, and Bayesian networks*. IlliGAL Report No. 98013, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL. 1998.

[158] Pelikan, M., Goldberg, D. E., and Lobo, F. G. A survey of optimization by building and using probabilistic models. In *Computational Optimization and Applications*, vol. 21, pages 5-20, 2002.

[159] Pelikan, M. and Mühlenbein, H. The bivariate marginal distribution algorithm. In *Advances in Soft Computing-Engineering Design and Manufacturing*, Springer-Verlag, London, pages 521-535, 1999.

[160] Peters, J. and Syska, M. *Circuit-switched broadcasting in torus networks*. Simon Fraser University, Technical Report, CMPT TR 93-04, May 1993.

[161] Peterson, H., Sen, S., Chandrashekar, J., Gao, L., Guérin, R. A, and Zhang, Z-L. Message-Efficient Dissemination for Loop-Free Centralized Routing. In *ACM Computer Communication*, vol. 38, no. 3, pages 65-74, 2008.

[162] Preparata, F. P., and Vuillemin, J. The cube connected cycles: A versatile network for parallel computation. In *Communications of the ACM*, vol. 24, pages 300-309, May 1981.

[163] Prim, R. Shortest connection networks and some generalizations. In *Bell Systems Technical Journal*, vol. 36, pages 1389-1401, 1957.

[164] Rechenberg, I. Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Frommann-Holzboog, Stuttgart, Germany, 1973.

[165] Reeves, C. Diversity and diversification in genetic algorithms: some connection with tabu search. In *Artificial Neural Nets and Genetic Algorithms*, Springer-Verlag, New York, pages 344-451, 1993.

[166] Rissanen, J. J. Modelling by shortest data description. In *Automatica*, vol. 14, pages 465–471, 1978.

[167] Rothlauf, F., Goldberg, D. E., and Heinzl, A. *Bad codings and the utility of well-designed genetic algorithms*. IlliGAL Report No. 200007, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL. 2000.

[168] Russell, S. J. and Norving, P. *Artificial Intelligence: A Modern Approach*, 2nd edition, Prentice Hall, pages 137-160, 2002.

[169] Rytter W. Context-free recognition via shortest paths computation: a version of Valiant's algorithm. In *Theoretical Computer Science*, vol. 143, no. 2, pages 343-352, 1995.

[170] Saad, Y. and Schultz, M. H. Topological properties of hypercubes. In *IEEE Transactions on Computers*, vol. 37, pages 867-872, July 1988.

[171] Salinger, P. and Tvrdík, P. All-to-All Scatter in Kautz networks. In *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, vol. 1470, pages 1057-1061, 1998.

[172] Sarbazi-Azad, H. and Khonsari, A., Ould-Khaoua, M. Analysis of k-ary n-cubes with dimension-ordered routing. In The *IEEE/ACM International Symposium on Cluster Computing and the Grid*, Berlin-Brandenburg Academy of Sciences and Humanities, Berlin, Germany, pages 493-502, 2003.

[173] Sastry, K., *Efficient atomic cluster optimization using a hybrid extended compact genetic algorithm with seeded population*. IlliGAL Report No. 2001018, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL. 2001.

[174] Sathish, S. V., Graham, E. F., and Jack, D. *Automatically tuned collective communications*. Dallas, Texas, United States, IEEE Computer Society, 2000.

[175] Schewefel, H. *Evolution and Optimization Seeking*, Wiley, New York, 1995.

[176] Schmaltz, J. and Borrione, D. *A Generic On Chip Network Model*. Tima Lab. Research Report ISRN TIMA-RR-05/03-06-FR, 2005.

[177] Schwarz, G., Estimating the dimension of a model. In *The Annals of Statistics*, vol. 6, pages 461-464, 1978.

[178] Schwarz, J. and Ocenasek, J. Experimental study: Hypergraph partitioning based on the simple and advanced algorithms BMDA and BOA. In *Proceedings of the International Conference on Soft Computing*, pages 124–130, 1999.

[179] Scott, D. S. Efficient All-to-All Communication Patterns in Hypercube and Mesh Topologies. In *Proceeding of the Sixth Conference Distributed Memory Concurrent Computers*, Portland, OR, pages 398-403, 1991.

[180] Seidel, S. R. Circuit switched vs. store and forward solutions to symmetric communication problems. In *Proceedings of the 4th Conference on Hypercube Computers and Concurrent Applications*, pages 253-255, 1989.

[181] Seitz, C. L. The Cosmic Cube. In *Communication of ACM*, vol. 28, pages 22-33, January 1985.

[182] Sendhoff, B., Kreuts, M., and Seelen, W. A condition for the genotype-phenotype mapping: causality. In *Proceedings of the 7th International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Francisco, pages 354-361, 1997.

[183] Shmoys, D. B. and Stein, C. Improved approximation algorithms for shop scheduling problems. In *SIAM Journal on Computing*, pages 617-632, 1994.

[184] Sicortex, Inc. *The webpage of SiCortex SC5832 system*, 2009. URL: <http://www.sicortex.com/products/high_capability_system_sc5832>.

[185] Silicon Graphics International. *SGI Origin 3000 web page*, 2009 URL: <http://www.sgi.com/products/remarketed/origin3000/overview.html>.

[186] Sony Corporation. *The Playsation 3 web page*, 2009, URL: <http://www.us.playstation.com/PS3/Systems?ref=http%3A//www.sony.com/index.php>.

[187] Spencer, C. Circuit-Switched Structured Communications on Toroidal Meshes, Master thesis, Simon Fraser University, 1994.

[188] Stewart, L. C. and Gingold, D. *A New Generation of Cluster Interconnect*. White Paper, SiCortex Inc., 2006.

[189] Suh Y. J. and Shin, K. G. All-to-All Personalized Communication in Multi-dimensional Torus and Mesh Networks. In *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 1, pages 38-59, 2001.

[190] Sullivan H. and Bashkow T. R., A large scale, homogeneous, fully distributed parallel machine. In *Proceedings of the 4th International Symposium on Computer Architecture*, 1997.

[191] Sundar, N. S., Jayasimha, D. N., Panda, D. K., and Sadayappan, P. Hybrid Algorithms for Complete Exchange in 2D Meshes. In *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 12, pages 1201-1218, December 2001.

[192] Sur, S., Jin, H. W., and Panda, D. K. Efficient and scalable All-to-All personalized exchange for infiniband-based clusters. In *Proceedings of the 2004 International Conference on Parallel Processing*, pages 275-282, 2004.

[193] Syswerda, G. Schedule optimization using genetic algorithms. In *Handbook of genetic algorithms*, Van Nostrand Reinhold, pages 332-349, 1991.

[194] Syswerda, G. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Francisco, pages 2-9, 1989.

[195] Szymanski T. and Hamacher V. On the permutation capability of multistage interconnection networks. In *IEEE Transactions on Computers*, vol. 36, no. 7, pages 810-822, July 1987.

[196] Thakur, R. and Choudhary, A. All-to-all communication on mesh with wormhole routing. In *Proceedings of the 1994 International Parallel Processing Symposium*, pages 561 -565, 1994.

[197] Thierens, D. and Goldberg, D. Convergence models of genetic algorithm selection schemes. In *Parallel Problem Solving from Nature*, PPSN III, Springer-Verlag, Berlin, pages 119-129, 1994.

[198] Thomas, C. H., Charles, L. E., and Ronald, R. L. *Introduction to Algorithms*, MIT Press, 2000.

[199] Tilera Coprorpation. *The web page of Tile-Gx processor*, 2009, URL: <http://www.tilera.com/products/TILE-Gx.php>.

[200] Tipparaju, V. and Nieplocha, J. Optimizing All-to-All Collective Communication by Exploiting Concurrency in Modern Networks. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, IEEE Computer Society, Washington, DC, USA, ISBN 1-59593-061-2, 2005.

[201] Tipparaju, V., Nieplocha, J., and Panda, D. Fast collective operations using shared and remote memory access protocols on clusters. In *Proceedings of the international Parallel and Distributed Processing Symposium*, 2003.

[202] Top500.org. *The webpage of the Roadrunner*, 2009, URL: <http://www.top500.org/system/10377>.

[203] Topkis, D.M. All-to-All Broadcast by Flooding in Communications Networks. In *IEEE Transactions on Computers*, n. 9, vol. 38, 1330-1333, 1989.

[204] Toueg, S. Deadlock- and livelock-free packet switching networks. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, Los Angeles, CA, USA, pages 94-99, 1980.

[205] Towles, B. and Dally W. J. Worst-case traffic for oblivious routing functions. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, ACM Press, NY, USA, pages 1-8, 2002.

[206] Tsai, Y. and McKinley, P. K. A Broadcast Algorithm for All-Port Wormhole Routed Torus Networks. In *IEEE Transactions on Parallel and Distributed Systems*, vol.7, no.8, 1996.

[207] Tsai, Y. and McKinley, P. K. An Extended Dominating Node Approach to Broadcast and Global Combine in Multiport Wormhole-Routed Mesh Networks. In *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 1, 1997.

[208] Tseng, Y. C., Lin, T. H., Gupta, S. K. S., and Panda, D. K. Bandwidth-Optimal Complete Exchange on Wormhole- Routed 2D/3D torus Networks: A Diagonal-Propagation Approach. In *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 4, pages 380-396, April 1997.

[209] Tseng, Y. C., Ni, S. Y., and Sheu, J. P. Toward Optimal Complete Exchange on Wormhole-Routed Tori. In *IEEE Transaction on Computers*, vol. 48, no. 10, pages 1065-1082, October 1999.

[210] Tusch, D. and Hommel, G. Multicast routing in Clos networks. In: *Proceedings of 2004 Design, Analysis, and Simulation of Distributed Systems*, Arlington, pages 21-27, 2004.

[211] Tutte, W. T. *Graph Thory*, Cambridge university press, New York, ISBN 0-521-79489-9, 2001.

[212] Tvrdík P. *Parallel Systems and Algorithms*, Czech Technical University in Prague, Lecture Notes, Prague, p. 177, ISBN 80-01-01559-9, 1997.

[213] Tzeng, N. F. Reliable butterfly distributed-memory multiprocessors. In *IEEE Transactions on computers*, vol. 43, no. 9. pages 1004-1013, 1994.

[214] University of Tennessee. *Parallel Basic Linear Algebra Subprograms (PBLAS)* web page, 2009, URL <http://www.netlib.org/scalapack/pblas_qref.html>.

[215] US Patent 7039058. *Switched interconnection network with increased bandwidth and port count*, 2006, URL: <http://www.patentstorm.us/patents/7039058 /fulltext.html>.

[216] Valiant, L. General context-free recognition in less than cubic time, In *Journal of Computers and Systems*, vol. 10, no. 2, pages 308-315, 1975.

[217] Vignaux, G. A., Michalewicz Z. A genetic algorithm for the linear transportation problem. In *IEEE Transaction on Systems, Man and Cybernetics*, vol. 21, pages 445-452, 1991.

[218] Wall, M. *GAlib: Matthew's Genetic Algorithms Library*, Massachusetts Institute of Technology, 1999, URL <http://lancet.mit.edu/ga/>.

[219] Walters, G. A., and Smith, D. K. Evolutionary design algorithm for optimal layout of tree networks. In *Engineering Optimization*, vol. 24, pages 261-281, 1995.

[220] Weisstein, E. W. Heawood Graph. In *MathWorld - A Wolfram Web Resource,* 2009, URL: <http://mathworld.wolfram.com/HeawoodGraph.html>.

[221] Weisstein, E. W. Utility Graph. In *MathWorld - A Wolfram Web Resource*, 2009, URL: <http://mathworld.wolfram.com/UtilityGraph.html>.

[222] Weisstein, E., W. Hypercube. In *From MathWorld - A Wolfram Web Resource*. 2009, URL <http://mathworld.wolfram.com/Hypercube.html>.

[223] Weisstein, E., W. Torus. In *MathWorld - A Wolfram Web Resource*. 2009, URL: <http://mathworld.wolfram.com/Torus.html>.

[224] Weisstein, W. E. Fast Fourier Transform. In *MathWorld - A Wolfram Web Resource*. 2009, URL <http://mathworld.wolfram.com/FastFourierTransform .html>.

[225] Wetzel A. Evaluation of the effectiveness of genetic algorithms in combinatorial optimization, Technical report, University of Pittsburgh, 1983.

[226] Whitley, D. A different genetic algorithm. In *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, Denver, 1989.

[227] Youssef, A. S. and Narahari, B. Banyan-hypercube networks. In *IEEE Transactions on Parallel Distributed Systems*. vol. 1, no. 2, pages 160-169, April 1990.

[228] Zhigljavsky, A. *Theory of Global Random Search*, Kluwer Academic Publishers, 1991.

[229] Zhuang, X., Liberatore, V. A Recursion-Based Broadcast Paradigm in Wormhole Routed Networks. In *IEEE Transactions on parallel and distributed systems*, vol. 16, no. 11, November 2005.

Appendix

The following tables summarize the upper bounds $\tau^{CC}(G)$ in terms of numbers of communication steps achieved on investigated topologies using the UMDA algorithm. Two integers in one cell separated by a slash indicate that the lower bound (a smaller integer) has not been reached while a single integer represents both the lower and the upper identical bounds reached by UMDA. Empty cells (illustrated by hyphens) denote the cases that have not been tested for their time complexity yet

Topology	OAS	AAS	OAB	AAB
Hypercube-8	3	4	2	3
Hypercube-16	4	9	2	4
Hypercube-32	7	16	2	7
Hypercube-64	11	35/32	3	11
Hypercube-128	19	-	3	-
Hypercube-256	32	-	4/3	-
Ring-bi-8	4	8	2	4
Ring-bi-16	8	34/32	3	8
Ring-bi-24	12	78/72	4/3	12
Ring-bi-32	16	140/128	4	16
Ring-uni-8	7	28	3	7
Ring-uni-16	15	128/120	4	15
Ring-uni-24	23	300/276	5	23
Ring-uni-32	31	550/496	5	31
Mesh-3x3	4,3,2	6	2,2,2	4
Mesh-3x4	6,4,3	12	3,2,2	6
Mesh-4x4	8,6,4	16	3,2,2	8
Mesh-5x5	12,8,6	32	3,3,3,	12
Mesh-4x8	16,11,8	64	3,3,3	16
Mesh-6x6	18,12,9	56/54	4,3,3	18
Mesh-7x7	24,16,12	-	4,4,3	24
Mesh-8x8	32,22,16	-	4,4,3	32
Mesh-9x9	40,27,20	-	4,4,4	-
Mesh-10x10	50,33,25	-	5,5,5	-
Torus-3x3	2	4	2	3
Torus-3x4	3	6	2	3
Torus-4x4	4	9/8	2	4
Torus-5x5	6	16/15	2	6
Torus-4x8	8	33/32	3	11/8

Topology	OAS	AAS	OAB	AAB
Torus-6x6	9	30/27	3	9
Torus-7x7	12	-	3	13/12
Torus-8x8	16	-	3	18/16
Torus-9x9	20	-	4/3	-
Torus-10x10	25	-	4/3	-
Petersen-10	3	5	2	3
Kautz-12	4	7	2	4
Heawood-14	5	10/9	2	5
Levi-30	10	31/28	3	10
Kautz-36	12*	34/31	3	12
K-Ring-8	2	3	2	2
Midimew-8	2	3	2	2
Spidergon-6	2	3	2	2
Spidergon-8	3	4	2	3
Spidergon-12	4	9	2	4
Spidergon-16	5	17/16	2	5
Spidergon-20	7	26/25	3	7
Spidergon-24	8	37/36	3	8
Spidergon-28	9	51/49	3	9
Spidergon-32	11	70/64	3	11
Spidergon-36	12	91/81	3	12
Spidergon-64	21	-	3	24
AMP-8	2	3	2	2
AMP-23	6	14	2	8/6
AMP-32	8	22	3	8
AMP-42	11	31	3	14/11
AMP-53	13	46	3	14/13
Omega-8	7	7	3	8
Omega-16	15	16/15	4	16/15
Butterfly-8	7	7	3	7
Butterfly-16	15	16/15	4	16/15
Clos-8	11	12/11	4	12/11
Clos-16	15	16/15	4	16/15
B-Tree-4	3	4	2	3
B-Tree-8	7	16	3	8/7
B-Tree-16	15	64	4	20/15
B-Tree-32	31	256	5	64/31
Fat-Tree-4	3	3	2	3
Fat-Tree-8	7	7	3	7
Fat-Tree-16	15	15	4	15
Fat-Tree-32	31	32/31	5	31
Full-Tree-7	6,4,3	12	3,2,2	7
Full-Tree-15	14,12,8,7	56	3,3,3,3	15
Full-Tree-31	30,28,24,16,15	240	4,4,4,4,4	31
Full-Tree-63	62,60,56, 50,48,32	992	5,5,5,5,5,5	64
2-fat-hypercube-4	7	8	3	7
2-fat-hypercube-8	15	17	4	15
2-fat-hypercube-16	31	36	6	33
2-fat-hypercube-32	63	-	7	-
2-fat-spidergon-6	11	12	4	12
2-fat-spidergon-8	15	17	4	16
2-fat-spidergon-10	19	25	5	20

Topology	OAS	AAS	OAB	AAB
2-fat-spidergon-12	23	37	5	24
2-fat-spidergon-14	27	50	5	29
2-fat-spidergon-16	31	66	6	37
2-fat-spidergon-18	35	86	6	42
3-fat-spidergon-4	11	11	4	11
3-fat-spidergon-8	23	38	5	24
3-fat-spidergon-12	35	82	6	42
4-fat-spidergon-4	15	16	4	16
4-fat-spidergon-6	23	38	5	24
4-fat-spidergon-8	31	64	6	34

Performance of the Kauz-12 network with a single faulty link (reduced performance is in bold).

link	OAS	AAS	OAB	AAB
no fault	4	7	2	4
0→3	**6**	**9**	**3**	**6**
0→5	**6**	**9**	**3**	**6**
0→4	**6**	**9**	**3**	**6**
1→7	4	**9**	2	**6**
1→6	4	**9**	2	**6**
1→8	4	**9**	2	**6**
2→B	4	**9**	2	**6**
2→A	4	**9**	2	**6**
2→9	4	**9**	2	**6**
3→0	4	**9**	2	**6**
3→1	**5**	**9**	2	**6**
3→2	**5**	**9**	2	**6**
5→7	4	**9**	2	**6**
all other	4	**9**	2	**6**

Performance of the 4×4 2D mesh network with a single faulty link (reduced performance is in bold).

link	OAS	AAS	OAB	AAB
no fault	8	16	3	8
0→1	**15**	**22**	3	**15**
0→4	**15**	**22**	3	**15**
all other	8	**22**	3	8

Performance of the 4×4 2D mesh network with a single faulty node. (reduced performance is in bold).

node	OAS	AAS	OAB	AAB
no fault	8	16	3	8
0	**15**	**22**	3	**15**
4	**15**	**22**	3	**15**
all other	8	**22**	3	8

Time complexity of the evolved schedules for the 8-node hypercube and spidergon for a few *M*-to-*N* CC patterns.

Topology	Pattern	MMS	MMB
Hypercube-8	Within the same base	2	2
	Between the bases	4	2
	Lower base to all	4	3
	diagonal to the lower base	2	2
Spidergon-8	Within the left half	2	2
	Left half to right half	3	3
	Left half to all	3	3
	Odd nodes to even nodes	3	2

Název	Evolutionary Design of Collective Communications on Wormhole Network
Autor	Ing. Jiří Jaroš, Ph.D.
Recenzenti	Prof. Günther Raidl
	Prof. Ing. Jiří Šafařík, CSc.

Vydavatel	Vysoké učení technické v Brně
	Fakulta informačních technologií
Obálka	Mgr. Dagmar Hejduková
Tisk	MJ servis, spol s r. o.
Vyšlo	Brno, 2010
Vydání	první