Spring 2010

Processor allocator for chip multiprocessors

Dawid Maksymilian Zydek
University of Nevada Las Vegas

PROCESSOR ALLOCATOR FOR CHIP MULTIPROCESSORS

by

Dawid Maksymilian Zydek

Master of Science
Wrocław University of Technology, Poland
2005

Postgraduate Certificate
Coventry University, UK
2005

A dissertation submitted in partial fulfillment
of the requirements for the

Doctor of Philosophy in Electrical Engineering
Department of Electrical and Computer Engineering
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
May 2010

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

THE GRADUATE COLLEGE

We recommend that the dissertation prepared under our supervision by

Dawid Maksymilian Zydek

entitled

Processor Allocator for Chip Multiprocessors

be accepted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Engineering
Electrical Engineering

Henry Selvaraj, Committee Chair

Emma Regentova, Committee Member

Rama Venkat, Committee Member

Mei Yang, Committee Member

Laxmi Gewali, Graduate Faculty Representative

Ronald Smith, Ph. D., Vice President for Research and Graduate Studies
and Dean of the Graduate College

May 2010

ABSTRACT

Processor Allocator for Chip Multiprocessors

by

Dawid Maksymilian Zydek

Dr. Henry Selvaraj, Examination Committee Chair
Professor of Electrical and Computer Engineering
University of Nevada, Las Vegas

Chip MultiProcessor (CMP) architectures consisting of many cores connected through Network-on-Chip (NoC) are becoming main computing platforms for research and computer centers, and in the future for commercial solutions.

In order to effectively use CMPs, operating system is an important factor and it should support a multiuser environment in which many parallel jobs are executed simultaneously. It is done by the processor management system of the operating system, which consists of two components: Job Scheduler (JS) and Processor Allocator (PA). The JS is responsible for job scheduling that deals with selection of the next job to be executed, while the task of the PA is processor allocation that selects a set of processors for the job selected by the JS.

In this thesis, the PA architecture for the NoC-based CMP is explored. The idea of the PA hardware implementation and its integration on one die together with processing elements of CMP is presented. Such an approach requires the PA to be fast as well as area and energy efficient, because it is only a small component of the CMP.

The architecture of hardware version of a PA is presented. The main factor of the structure is a type of processor allocation algorithm, employed inside. Thus, all important

allocation techniques are intensively investigated and new schemes are proposed. All of them are compared using experimentation system.

The PA driven by the described allocation techniques is synthesized on FPGA and crucial energy and area consumption together with performance parameters are extracted.

The proposed CMP uses NoC as interconnection architecture. Therefore, all main NoC structures are studied and tested. Most important parameters such as topology, flow control and routing algorithms are presented and discussed. For the proposed NoC structures, an energy model is proposed and described.

Finally, the synthesized PAs and NoCs are evaluated in a simulation system, where NoC-based CMP is created. The experimental environment took into consideration energy and traffic balance characteristics. As a result, the most efficient PA and NoC for CMP are presented.

ACKNOWLEDGMENTS

This dissertation completes a 3-year period in my live – three years of my Ph.D. studies in the USA. I have to admit that it was a very difficult time of my live. However, I have never been alone.

First of all, I would like to express my deepest gratitude to my Ph.D. advisor and doctoral advisory committee chair Dr. Henry Selvaraj. He has helped me in every aspect of my studies and life in the USA. Dr. Selvaraj recommended me Ph.D. studies at UNLV, during our conversations at a conference in Coventry. He helped me with paper work and my first steps in the USA and at UNLV. As my advisor, he always had time to help me, to give me advice or just talk with me without any reservation, no matter how busy he was. Besides professional activities, Dr. Selvaraj helped me a lot in my private life – he introduced me to Polish Americans in Las Vegas, he helped me with my transportation and accommodation problems, and in many other issues. There is just lack of words to express my thankfulness to him. Dr. Selvaraj, I will never forget help, kindness and positive energy that I have experienced from you.

I would like to thank the members of my advisory committee (in alphabetical order): Dr. Laxmi Gewali for invaluable action during the last period of my studies, Dr. Emma Regentova for valuable remarks, Dr. Rama Venkat for totally different approach and observations (it will bring a lot of research in near future) and Dr. Mei Yang for precious suggestions. Thank all of you very much for your professionalism and positive attitude. I was really lucky to have had the opportunity to work with such a dream committee members.

It is not only for me that the time in the USA was not easy. I would like to thank my wife Monika for patience and support. Thank you that you are always with me, that you support me and that you have endured these difficult three years in the USA.

I would also like to thank my mother Małgorzata, father Rudolf and uncle Ludwik for continuous and strong support not only during the past three years, but in my whole life. I am so happy that I have you and that I can always relay on you.

This thesis also includes ideas developed during my M.Sc. studies at Wrocław University of Technology. I would like to thank Dr. Leszek Koszałka and Dr. Iwona Poźniak-Koszałka for inspiration and support. Dr. Leszek Koszałka, in his class developed my research interests – without him this thesis would not have been written, because of I would not have been a researcher. Also, I would like to thank my friend Grzegorz Chmaj for inspiration and motivation.

As I have written at the beginning, I have never been alone. I met in Las Vegas very good and positive Polish Americans and the helpful attitude and positive energy experienced from them will never be forgotten. Thus, I would like thank: Iwona and Zdzich Podzorski, Robert and Asia Kopacz, Szymon Kopacz, Edyta and Krzysiek Jankowski. Thank you all very much – your attitude inspired me and changed my opinion about people.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

1D	1-Dimension
2D	2-Dimension
3D	3-Dimension
ADJ	ADJacency
ALM	Adaptive Logic Module
ALUT	Adaptive LUT
AS	Adaptive Scan
BF	Best Fit
BLAT	Busy List Allocation for Torus
BMAT	Bit Map Allocation for Torus
CCC	Cube Connected Cycles
CFL	Compacting Free List
CMP	Chip MultiProcessor
DOR	Dimension Order Routing
EVC	Express Virtual Channel
FCFS	First Come First Serve
FF	First Fit
FL	Free List
FPGA	Field Programmable Gate Array
FS	Frame Sliding
FSL	Free Submesh List
HDL	Hardware Description Language

IAS	Improved Adaptive Scan
IC	Integrated Circuit
IFF	Improved First Fit
IQA	Improved Quick Allocation
ISBA	Improved Stack-Based Allocation
JS	Job Scheduler
LB	Load Balanced
LUT	Look Up Table
NoC	Network-on-Chip
NVC	Normal Virtual Channel
OS	Operating System
OSI	Open System Interconnection
PA	Processor Allocator
PE	Processing Element
QA	Quick Allocation
SAT	Sorting Allocation for Torus
SBA	Stack-Based Allocation
SBAT	Stack Based Allocation for Torus
SoC	System-on-Chip
ULSI	Ultra-Large-Scale Integration
VC	Virtual Channel
VHDL	Very high speed integrated circuit HDL
VLSI	Very-Large-Scale Integration

LIST OF SYMBOLS

A	processor allocation algorithm
B	size/length of busy list
B_J	set of candidate blocks
$B[w, h]$	busy array of mesh/torus $w \times h$
β	busy subgrid
C_J	coverage set with respect to job J
C_T	coverage array
Ch	set of channels in network
$Ch(N_x,N_y)$	cut of network N
c	column number in mesh/torus
ch	channel
E_1	denoted by t_a – allocation time
E_2	denoted by t_s – simulation time
E_3	denoted by L – system load
E_4	denoted by F_e – external fragmentation
E_5	denoted by $R_{VC<x, y>}$ – VC count
E_6	denoted by $R_{EVC<x, y>}$ – EVC count
E_7	denoted by $T_{R_{VC}}$ – total VC count
E_8	denoted by $T_{R_{EVC}}$ – total EVC count
$E_{B_{bit}}$	buffering energy
E_{bit}	energy consumed in sending one bit of data
$E_{bit}^{t_i,t_j}$	average energy consumption of sending one bit of data from tile t_i to t_j

E_c	energy consumed in one cycle
$E_{L_{bit}}$	energy consumed on physical channels
$E_{L_{bit}}^{2D-mesh}$	energy consumed on physical channels between tiles for 2D-mesh
$E_{L_{bit}}^{2D-torus}$	energy consumed on physical channels between tiles for 2D-torus
$E_{R_{bit}}$	bit energy in router
$E_{R_{bit}}^{EVC}$	energy of bit traversing through express virtual channel
$E_{R_{bit}}^{NVC}$	energy of bit traversing through normal virtual channel
$E_{S_{bit}}$	energy consumed by switch arbitration
$E_{VC_{bit}}$	energy consumed by virtual channel arbitration
$E_{W_{bit}}$	energy consumed by interconnection wires inside switching fabric
F	size/length of free list
F_e	external fragmentation
F_{max}	average maximum frequency
f_{max}	maximum frequency
h	vertical length of mesh/torus
I_J	initial candidate base block
$J(p, q)$	job requesting subgrid $p \times q$
L	system load
l_{wire}	length of physical channel
$M(w, h)$	2D-mesh
m	address width of h
N	number of nodes

N_a	total number of processors in the system
N_f	number of free processors
N_{hops}	number of physical channels traversed by a packet between tile t_i and t_j
N_{hops}^{EVC}	number of express virtual channels traversed by a packet between t_i and t_j
N_{hops}^{NVC}	number of virtual channels traversed by a packet between t_i and t_j
n	address width of w
$\xi_{\beta,J}$	coverage of busy subgrid β with respect to job J
P	power dissipation
P_1	number of jobs in queue
P_2	range of numbers for the size of each job $J(p, q)$
P_3	range of numbers for execution time of each job in queue
P_4	size of w and h
P_{G1}	denoted by w – horizontal size of mesh/torus
P_{G2}	denoted by h – vertical size of mesh/torus
P_{G3}	evaluated topology
P_{L1}	number of jobs created by generator
P_{L2}	maximum size of generated job $J(p, q)$
P_{L3}	maximum execution time for job J
P_{L4}	type of distribution
P_{L5}	processor allocation algorithm
P_{L6}	address $<x, y>$ of PA
P_{L7}	flow control
P_{L8}	routing algorithm

P_{L9}	number of misroutes (misroute value)
p	horizontal length of subgrid
q	vertical length of subgrid
$R_{EVC<x, y>}$	EVC count
R_J	reject area of submesh with respect to job J
$R_{VC<x, y>}$	VC count
r	row number in mesh/torus
$S(p, q)$	subgrid in mesh/torus
$T_{R_{EVC}}$	total EVC count
$T_{R_{VC}}$	total VC count
$T(w, h)$	2D-torus
t_a	allocation time
t_s	simulation time
w	horizontal length of mesh/torus
x_b	horizontal coordinate of base
x_e	horizontal coordinate of end
x_s	horizontal coordinate of sink
y_b	vertical coordinate of base
y_e	vertical coordinate of end
y_s	vertical coordinate of sink
Z	set of all processors

CHAPTER 1

INTRODUCTION

In the quest of faster computers, the Integrated Circuits (ICs) have come a long way of development since their appearance in the late 1950s. In 1965, Intel co-founder Gordon Moore made a prediction known as “Moore’s Law” which stated that the number of transistors that could fit in an IC was increasing exponentially, doubling every two years [87]. This observation has continued to the present day, where silicon chips contain thousands of millions of transistors in 45nm technology (e.g. Intel® Xeon® Processor). International Technology Roadmap for Semiconductors expects, that by 2016 a single chip will contain multi-billion transistors in 25nm technology with frequencies around 35GHz [55]. This increasing chip density allows integration of more components onto a single die, instead of using several chips to implement a given system. Such combination of several components onto single chip is known as System-on-Chip (SoC).

The SoC design methodology gives the possibility to integrate a complete system with its peripherals, interfaces, storage and processing elements into a single package that offers higher performance, lower energy consumption and lower general system cost. Moreover, the current high integration level achieved by following Moore’s Law allows placing multiple processors on a single chip to form parallel computing system that is know as Chip MultiProcessor (CMP). The advantage of CMPs is replacement of several processors by single one with many cores, but energy consumption and needed space are reduced while throughput is the same or even higher. Also CMPs ensure better fault tolerance because a defect in one processor does not make the entire chip useless. Examples of CMPs with small number of cores are the Sony Emotion Engine [91], the

IBM Cell processor [93], dual-core or quad-core processors by AMD [5] or Intel [54]. More cores can be found in Cell Broadband Engine Architecture (CBEA) [51] developed by IBM, Sony and Toshiba (9 cores), TILE-Gx [106] developed by Tilera (100 cores) or Intel Teraflop [110] designed by Intel (80 cores).

There are two types of CMP: homogeneous – where all cores on the die are the same, and heterogeneous – where the cores differ in their characteristics, e.g. size of cache, clock frequency, etc. In this work, the author considers homogenous CMP.

For systems with only few cores integrated on the chip, the processor performance is restricted mainly by the processing elements (cores). In this case, system optimization focuses on processing elements – their internal frequency, architecture, etc. As the number of cores becomes larger, the system performance begins to be affected by on-chip interconnect – the bus structures widely used at present become a limiting factor for performance, space and energy consumption [35], [97]. The main factors influencing bus constrains are bus arbitration bottleneck and bandwidth limitation. The first factor means that with increasing number of bus hosts the arbitration delay will be increasing. In the case of bandwidth limitation, bus structure is shared by all components attached to the bus in a time division manner. Therefore the bandwidth decreases with the increasing number of components. In 1999, in order to overcome the disadvantages of bus structures, the idea of replacement of shared-medium approach with network based architecture was proposed [10], [35], [56], [112]. The Network-on-Chip (NoC) concept is based on ideas from computer networks, where each component of a SoC is a node of the on-chip network.

Beside the good on-chip interconnection system, the efficient use of cores that are available on a CMP is also an important factor in high performance multi-core processors. On-chip processors can corporate among themselves in two ways. They may be used to run separate jobs or they can serve to execute one application (parallel processing). Problem of partitioning the job into tasks that can be assigned to processors is called task assignment [79]. The result of task assignment process is a job that consists of tasks that can be executed simultaneously, but each task on a separate core of CMP. The decision, which job is assigned to which processors, is made by the processor management system of the Operating System (OS), which consists of two components: Job Scheduler (JS) and Processor Allocator (PA). The JS is responsible for job scheduling that deals with selection of the next job to be executed, while the task of the PA is processor allocation that selects a set of processors for the job selected by the JS (Fig. 1-1).

The work presented in this thesis deals with the processor allocator for chip multiprocessors using network on chip as the on-chip interconnection system.

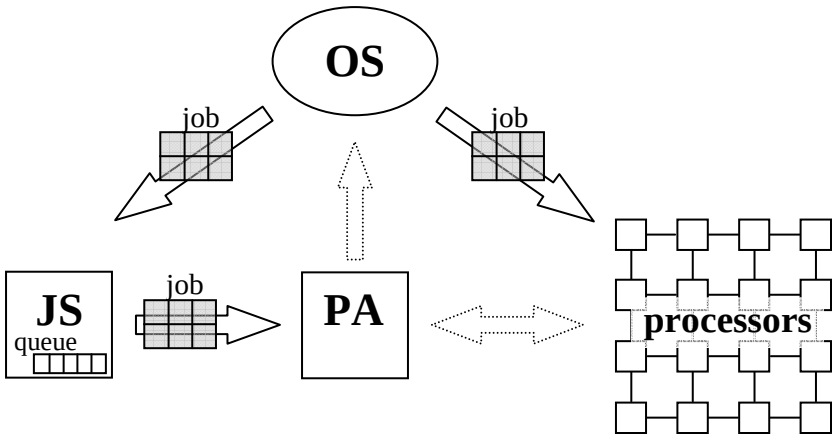


Figure 1-1. Block scheme of processor management system.

1.1. Network on Chip Overview

At the beginning, Network-on-Chip (NoC) research focused on combining ideas from computer networks and parallel computing. However NoCs and regular networks are designed for different environments and such parameters as energy, power and area consumption are unique to NoC. Also, on-chip networks are characterized by locality and determinism, that differentiate them from traditional networks [10]. Therefore, instead of using 7-layer OSI reference model [123] known from computer networks, 4-layer approach was adopted by NoC research community [12], [84]:

1. System Level: Covers applications and the network architecture. At this level, network details are hidden.
2. Network Adapter: Distinguishes the Processing Element (PE) from the NoC and takes care of end-to-end flow control (between source and final destination). At this level, packets are formed based on messages originating from the PEs.
3. Network: Consists of routers connected by channels, creating the topology. Flow control between routers (node-to-node) is defined at this level.
4. Link Level: Faces the physical connections between routers. In this level, packets are broken into flow control units – flits.

In this work, the author covers problem placed in levels two and three of the above model.

The main task of NoC is providing the communication infrastructure for PEs. Fig. 1-2 shows two-dimensional mesh network (2D-mesh), which is the most popular NoC architecture. PEs are developed separately as single blocks and NoC is formed to connect

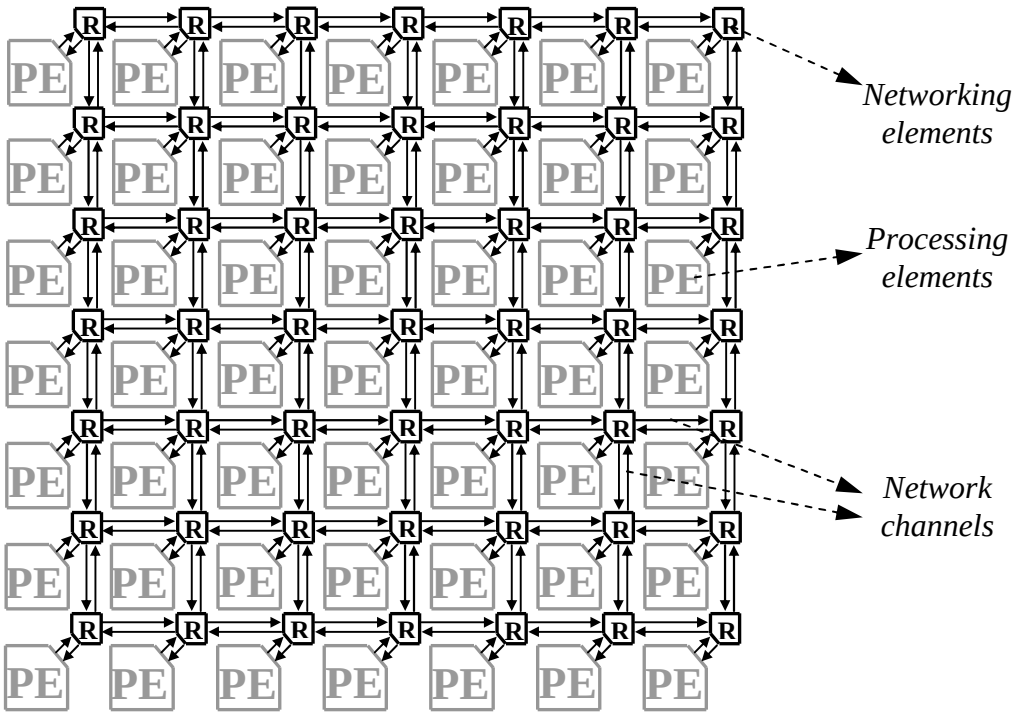


Figure 1-2. An example of NoC architecture (7×7 2D-mesh topology).

the blocks as nodes in the network. The scalable and configurable NoC can be adjusted to the needs of different network traffic. A PE and Router (R) are connected by direct link, the width of which is the same as the width of network channels. Each router is connected to four other neighboring routers (in the Fig. 1-2 for 2D-mesh topology, edge routers have 2 or 3 neighboring routers) through input and output channels and it is responsible for routing messages between PEs. A channel contains two one-directional point-to-point buses and it connects two routers or router with a PE [56], [76].

While designing an efficient NoC, the required performance has to be compromised to meet many design parameters, which unfortunately is not so trivial. The main NoC parameters are [34], [41]:

- The topology – determines the arrangement of nodes (routers) and channels. It relates to the physical structure of the network,
- The routing – determines the path for a packet from a source node (router) to a destination. A route (or path) is an ordered set of channels between source and destination nodes,
- Flow control – deals with the allocation of resources to packets along their route. The most important resources in NoCs are the channels and the buffers.

All above mentioned criteria have a large impact on the final performance and cost of the system. They are also not independent of each other, e.g. the choice of topology has an impact on available routing algorithms.

Efficient NoC architectures for CMPs are presented and studied in Chapter 2 of this work.

1.2. Processor Allocation Problem

CMPs consist of many parallel PEs with distributed off-chip memory. The PE node consists of a processor and cache memory. Jobs created to run on these systems are parallel programs contain many tasks that communicate with each other. Each of the tasks runs on a separate PE (core), so the incoming job specifies number of PEs needed to execute the job. The selection of the subset of PEs required for a given job is called processor allocation problem [24], [101], [127] and it is done by Processor Allocator (PA) [128].

Efficient processor allocation should be characterized by:

1. High utilization: Processor allocation must provide maximal resource utilization. It is done by minimizing any kind of fragmentation that maximizes the use of all processors.
2. Low Overhead: Allocation techniques must be fast and deliver low overhead (all allocation requests have to be processed at run-time).
3. Scalability: Algorithms must support systems with thousands of nodes without any bottleneck.

Beside a well designed PA, the system utilization parameter in a CMP system can be improved by a better job scheduling process. Job scheduling is done by Job Scheduler (JS) that deals with the selection of the job to be executed next. The natural FCFS (First Come First Serve) scheduling policy has "blocking" nature – larger job for which allocation algorithm could not find enough PEs may block smaller jobs in the queue, for which sufficient PEs are available. That limitation of FCFS policy was reason for researching different, more efficient scheduling strategies such as backfilling [88], gang [43], lazy scheduling [86], scan [72] or other [17], [19], [37], [70], [85], [119].

As it was mentioned earlier, in a CMP an incoming job is described by the number of PEs required. The JS selects the next job from the system queue for execution using a proper scheduling policy. For a job selected by the scheduler, the PA tries to find the required number of free PEs. If such a number of free PEs are not available, then the JS handles the job according to implemented policy (e.g. the scheduler waits until some PEs are released or a smaller job is sent from the queue to the PA) – Fig. 1-1. The jobs are

allocated in such a manner that at a given moment, a PE processes a task of one job only, and that if PEs are allocated, they process tasks of the job until completion [127], [128].

In this work, the author focuses on a PA structure for CMPs and the processor allocation algorithms. As job scheduling policy, the FCFS strategy is assumed.

There are two major processor allocation strategies: Contiguous and Non-contiguous [9], [100], [101]. In the contiguous processor allocation, the PEs allocated to a job are physically adjacent and have the same topology like NoC. In non-contiguous allocation strategy, job can be executed on multiple disjoint PEs that allows dividing a job rather than waiting until required number of adjacent PEs becomes available. But it can generate global traffic, which we would like to avoid in NoC [34], [41], [128]. This is the reason why the author has chosen to focus on contiguous strategy that is considered in this work – Chapter 3.

1.3. Objective and Scope of Research

The work presented in this thesis deals with modern CMPs and issues related to processor allocation problem for CMPs. The work focuses on the following topics:

1. Designing efficient NoC architecture for future homogenous CMPs.
2. Developing processor allocation algorithms for mesh-based CMPs.
3. Designing processor allocation algorithms for torus-based CMPs.
4. Examining the proposed allocation techniques by creating experimentation system.
5. Designing a CMP with a PA integrated on the same die.
6. Designing a PA and synthesizing it for an FPGA device.
7. Examining the proposed NoC-based CMPs with integrated PA by creating simulation environment.

1.4. Dissertation Overview

This chapter introduces a reader to the subject and scope of the thesis. In Chapter 2, the review of all important NoC solutions is presented. Such aspects as network topology, flow control and routing algorithm are explained and investigated. For the presented problem, a suitable NoC architecture is chosen and an appropriate energy model is proposed.

All important processor allocation techniques are described in Chapter 3. They are based on topologies chosen in the previous chapter. Besides reviewing existing algorithms, new algorithms and improvements on the existing ones are proposed. At the end of this chapter, the presented schemes are compared and the best approaches are shown.

In Chapter 4, the main idea of this thesis – integration of the PA on the same chip as CMP – is presented. The architecture of the proposed the PA is described. The PA is synthesized and synthesis results are presented and discussed. Based on the results, an energy estimation is performed.

The performance of the PA for NoC-based CMP is analyzed is Chapter 5. The most suitable PAs and NoCs from previous chapters are collected and compared in CMP simulation environment. Such aspects as energy and performance of the PA and NoC are investigated and analyzed.

Chapter 6 gives conclusions and possibilities for future development.

1.5. Self-Citations

The following publications have contributed to the material presented in this work:

- *Hardware Implementation of Processor Allocation Schemes for Mesh-Based Chip Multiprocessors* [126]: A hardware implementation of the PA for NoC with mesh topologies has been proposed. Allocation algorithms for mesh topologies have been presented together with their synthesis results.
- *Fast and Efficient Processor Allocation Algorithm for Torus-Based Chip Multiprocessors* [125]: Processor allocation algorithms for torus topology have been proposed. Algorithms are compared with their mesh counterparts.
- *Memory Utilization of Processor Allocator for NoC-based Chip Multiprocessors with Mesh Topology* [128]: Aspects of the memory utilization of the PA based on the mesh topology are explored.
- *Processor Allocation Problem for NoC-based Chip Multiprocessors* [127]: The conception of integration the PA on the same die as NoC-based CMP has been presented.
- *Review of Packet Switching Technologies for Future NoC* [129]: Novel NoC architectures have been reviewed and future directions of the NoC development have been presented.
- *Algorithms and Experimentation System of Unicast, Multicast and Broadcast Transmission for Optical Switches* [124]: The methodology of the building efficient and realistic experimentation systems has been presented and tested for algorithms for optical switches.

CHAPTER 2

NOC-BASED CHIP-MULTIPROCESSOR

CMP architectures with NoC are currently the most advanced multiprocessor architectures. They differ from standard CMPs with few cores on the same die, where regular busses are used as interconnects [91], [93]. Also they differ from older multiprocessor systems, where PEs were placed on one main board and they were connected by buses (or network) on the board, not on the chip [21], [81]. Inside the chip as it takes place in the CMPs, interconnects and processing elements are significantly closer than in off-chip networks. It implies better latency and bandwidth performance, but such properties like power, area and cost restrictions become crucial. Also, the complexity of designing efficient and scalable on-chip communication solutions will be increasing together with the number of PEs integrated on a single die. To provide scalability and effective use of resources available in ULSI technology, a tiled architecture is proposed [105] (Fig. 2-1). The PEs are connected by a NoC that replaces

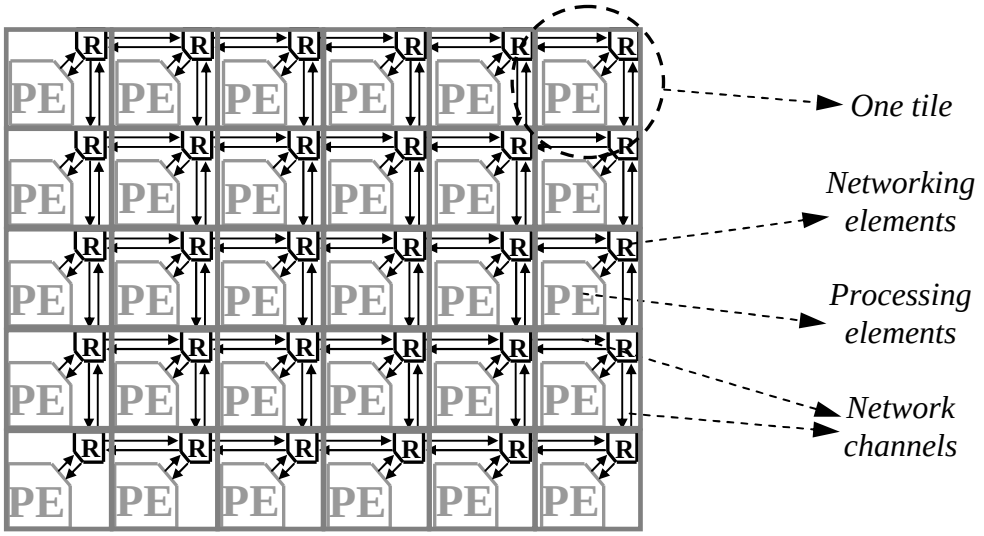


Figure 2-1. Tiled CMP (6×5 2D-mesh topology).

the traditional on-chip buses. The chip area is divided into square tiles. Each tile contains networking elements (router, PE network interface, network channel) and PEs (processor, cache memory, etc.). Each tile is connected to a network Router (R) by PE network interface. Communication among tiles is done by sending messages over the network using tiles' network interfaces (routers). The NoC serves as a global communication infrastructure, which can be optimized. On-chip network supports parallel communication, such that the required high bandwidth for CMPs can be realized. As it was mentioned earlier, the described architecture is homogenous – the PEs in every tile are the same (the same structure, computation, power, etc.). Every single PE is able to perform the required computation and return a result for a given set of input parameters.

2.1. Network Topology

Topology is one of the main properties that characterize NoC. It is also the first step in designing a network due to dependency of routing algorithm and flow control on the topology. Topology describes the layout and structure of the node and links on the chip. In this work, the author considers direct networks, where every node in the network is both a PE and router.

2.1.1. Definitions

Definition 2-1: A degree of a network node is the number of channels connected to the node. A network is degree regular, when all the nodes in the network have the same degree.

Definition 2-2: A path in the network is an ordered set of channels $\{ch_0, ch_1, \ldots, ch_{n-1}\}$, such that $ch_i \in Ch$ for $i \in [0, n\text{-}1]$ and $dest(ch_{i-1}) = sour(ch_i)$ for $i \in [1, n\text{-}1]$, where: Ch –

set of channels in the network, $dest(ch_i)$ – is defined as the destination node of the channel ch_i, $sour(ch_i)$ – is defined as the source node of the channel ch_i.

Definition 2-3: The length or hop count of a path is the number of channels traversed by the path. A hop is the unit of network distances.

Definition 2-4: A minimal path between two nodes is a path with the smallest hop count connecting these two nodes.

Definition 2-5: A diameter of a network is the largest, minimal hop count over all pairs of nodes in the network.

Definition 2-6: A cut of a network, $Ch(N_1, N_2)$, is a set of channels that partitions the set of all nodes into two disjoint sets, N_1 and N_2. Each element of $Ch(N_1, N_2)$ is a channel with a source in N_1 and destination in N_2, or vice versa.

Definition 2-7: A bisection of a network is a cut that partitions the entire network nearly in half, such that $|N_2| \leq |N_1| \leq |N_2| + 1$, where $|N_2|$ – number of nodes in set N_2, $|N_1|$ - number of nodes in set N_1.

Definition 2-8: The bisection bandwidth of a network is the minimum bandwidth over all bisections of the network.

Definition 2-9: The latency of the network is the time required by a packet to traverse the network. Besides a topology, latency depends also on routing, flow control and the design of router.

Definition 2-10: The throughput of a network is the data rate acceptable by the network. A throughput is expressed in bits per second [*bps*]. Similar to latency, throughput depends on the topology, flow control and implemented routing.

2.1.2. Characteristic of desired network

The degree of a node decides the number of ports in router. Thus, a node's degree has impact on the complexity (power, energy and area consumption) of router. Smaller the value of the degree, lesser the cost, while its homogeneity leads to uniform routers. It implies desire of small and fixed degree [34], [35], [41].

The diameter of a network characterizes the distance between nodes, that has direct impact on the latency of the network. Lower latency gives shorter distances, that implies a need for smaller network diameter [27], [34], [41].

The topology has significant influence on flow control and routing algorithms. Simple and regular topology reduces the complexity of routing algorithm [34], [41], [89].

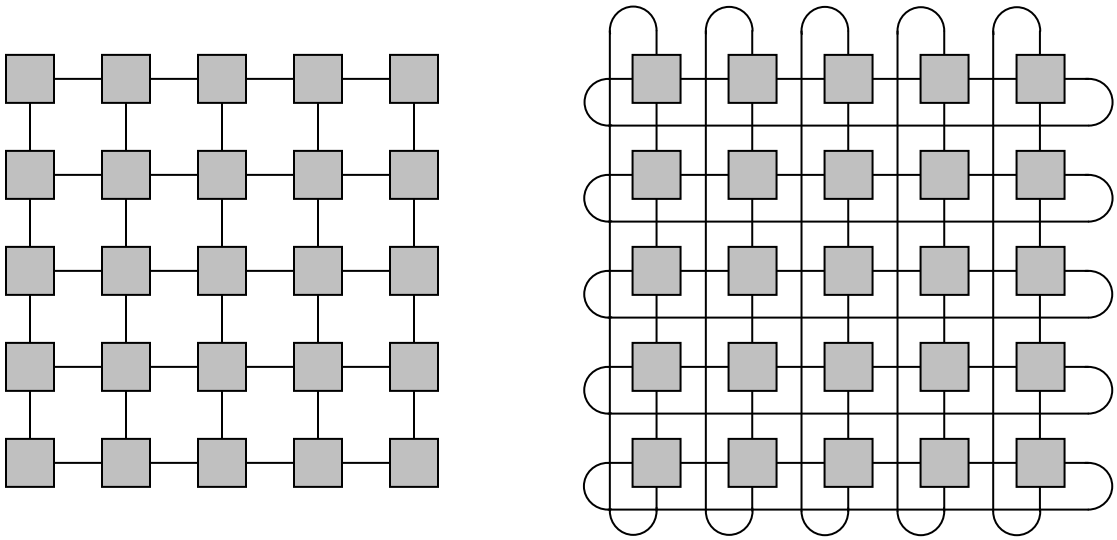
An optimal topology is also characterized by its path diversity. Multiple minimal paths among nodes reduce the impact of defects in manufacturing process. Path diversity also allows balancing the load across channels and makes the network more tolerant to faulty channels and nodes [34], [41], [50].

Whatever the topology, the network needs to be laid out in a die. Due to poor progress of "3D stacking" that is still under research, a 2D die is considered. This implies that for networks with dimension higher than 2D, topological adjacency does not lead to spatial adjacency. Thus higher dimensional topologies have negative impact on the wire delay and the wiring density. These are possible reasons for the necessity of long wires. However, implementation of long wires can affect the operating frequency and power consumption. Furthermore, tiles can have more channels crossing at least one of their edges, that can force a channel width to be less than required by architecture [57].

Besides on-chip embedding issues, low dimensional networks represent also better performance in comparison to their higher dimensional counterparts. At the same bisection, the low dimensional topologies provide lower latency and higher throughput. Moreover, topologies of many dimensions require more and longer wires [29].

2.1.3. 2D-mesh (*k*-ary 2-mesh) topology

2D-mesh (*k*-ary 2-mesh) topology [34], [41], [82] meets all properties described in Section 2.1.2. It is also an obvious and natural choice for tiled architecture due to its close match with the physical layout of the die. A *k*-ary 2-mesh consist of $N=k^2$ nodes arranged in a 2-dimensional mesh with *k* nodes along the two dimensions. Every node in the middle of a mesh is connected to four neighboring nodes, while nodes on the edges of mesh are connected to two or three neighbors, that makes a mesh network degree slightly irregular (Fig. 2-2a). Nodes are addressed by 2-digit radix-*k* address $\{a_1, a_0\}$, where a_1 and a_0 represent the node position in the first and second dimension respectively. The



(a) 5-ary 2-mesh (b) 5-ary 2-cube

Figure 2-2. Mesh (a) and torus (b) networks.

diameter of 2D-meshes is 2(k-1) hops. The bisection is $2N / k$, that can offer wider channels and higher channel bandwidth for a given bisection density.

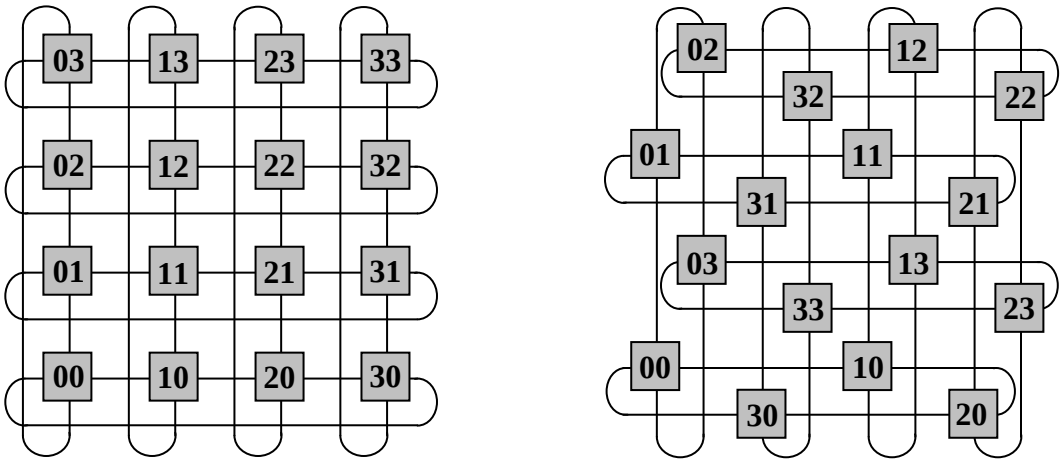
2D-meshes are characterized by uniformly short wires that allow high speed operation without repeaters. Minimal paths between nodes from logical point of view are also physically minimal, that allows exploiting local traffic. Path diversity is good as it ensures better reliability and load balance.

One of the main drawbacks of k-ary 2-mesh is lack of the same available bandwidth for every node – the bandwidth available to nodes at corners and edges is less while these nodes have a higher average distance from other nodes.

2.1.4. 2D-torus (k-ary 2-cube) topology

2D-torus (k-ary 2-cube) topology [33], [34], [41], similar to earlier described 2D-mesh, fits very well to the characteristic described in Section 2.1.2. A 2D-torus is achieved by enriching a 2D-mesh with additional channels that connect the external nodes in each row and column (Fig. 2-2b). Similar to 2D-mesh, k-ary 2-cube has regular physical arrangements that make its well suited for on-chip layout. A k-ary 2-cube consists of $N=k^2$ nodes arranged in a 2-dimensional torus with k nodes along the two dimensions. Every node is connected to four neighboring nodes – a torus network is degree regular. Wraparound channels added in 2D-toruses doubling their bisection to $4N / k$ and decreasing their diameter to $2\lfloor k/2 \rfloor$ hops. They also provide edge-symmetry that helps to improve load balance and reliability of a network.

Wraparound channels are reasons for the asymmetric channel lengths in a direct physical mapping of the 2D-torus (Fig. 2-3a). The wraparound channel has to be long enough to span the length of all k nodes. A long wraparound link can increase



(a) 4-ary 2-cube (b) Folded 4-ary 2-cube

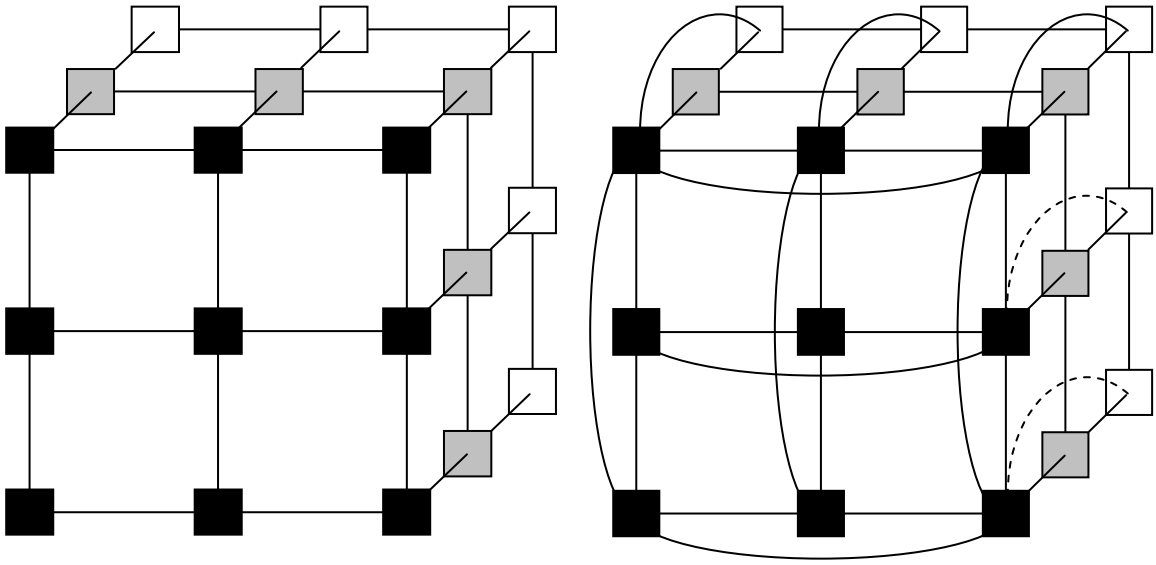
Figure 2-3. Folding toruses to avoid wraparound channels.

propagation delay that brings negatively impact on latency. Also it can require repeaters and decrease operating frequency of all links. All these problems can be avoided by folding the torus as shown in Fig. 2-3b. The folding keeps the topology untouched but eliminates the wraparound channels at the expense of doubling the length of the other channels, however such a doubled channel possesses acceptable latency characteristic and does not require repeaters.

2.1.5. Other network topologies

Besides *k*-ary 2-mesh and *k*-ary 2-cube, many different network topologies have been proposed for use in CMPs, such as: high dimensional *k*-ary *n*-cube and *k*-ary *n*-mesh [13], [29], [34], [41], fat tree [8], [90], hierarchical ring [15], [49], cube connected cycles [58], [95] and others [8], [34], [41], [64], [65], [98]. However they do not meet all requirements stated in Section 2.1.2.

Higher dimensional meshes and toruses can be constructed by iteratively adding dimensions (Fig. 2-4). They show higher bisection and path diversity, however in case of

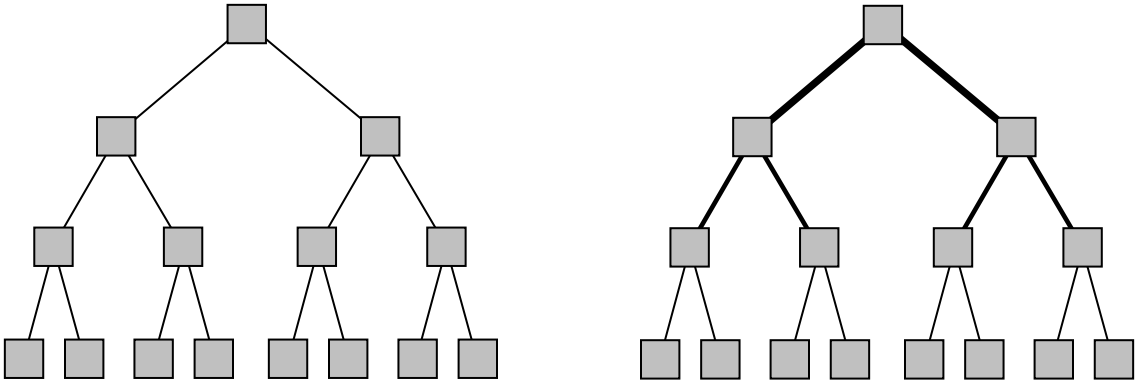


(a) 3-ary 3-mesh (b) 3-ary 3-cube

Figure 2-4. A 3D-mesh (a) and 3D-torus (b) networks with 3^3 nodes.

using them for CMP, they have drawbacks. Higher degree of these networks causes routers to be larger due to an increased number of buffers and complexity, thus area and power budgets are increased. Moreover, higher-dimensional meshes and toruses must be mapped on 2D die, however topological adjacency does not fit spatial adjacency. It can lead to longer wires and larger area requirements.

A tree topology creates the network in the form of a tree. A tree consists of a single root at the top level of the hierarchy, which is connected to nodes immediately below the level of root. The simplest tree network is the binary tree, where each node in the tree is connected with two nodes in the level immediately below the node – children (Fig. 2-5a).

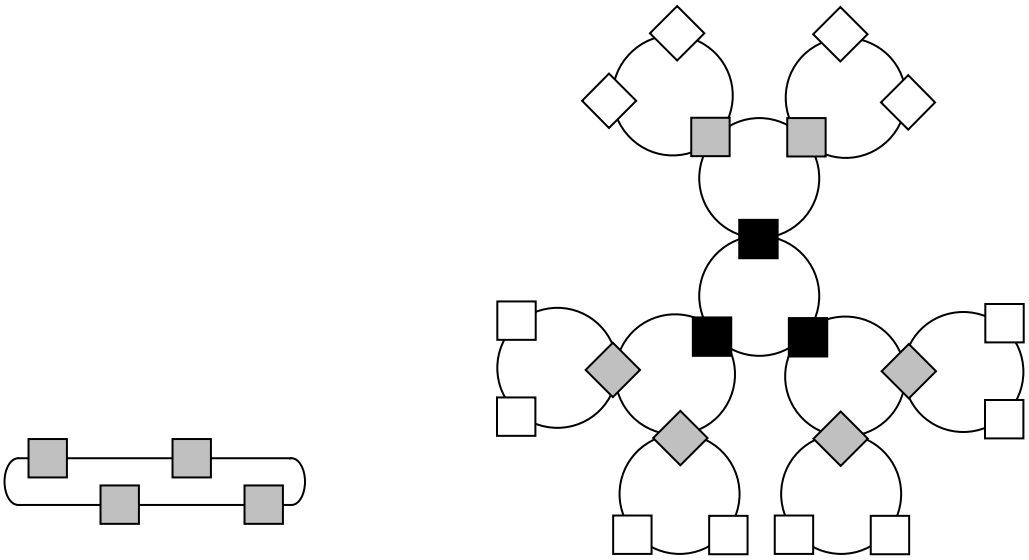


(a) A binary tree (b) A fat tree

Figure 2-5. Examples of tree networks: A binary tree (a) and a fat tree(b).

A tree can be created recursively by connecting each node to several children nodes until the desired depth is reached. Tree networks have a smaller diameter, but their bisection is only 1 – there is only one path available between any pair of nodes. It leads also to lack of path diversity and weak fault tolerance. Also the channels at the lowest levels can become system bottlenecks – besides traffic from their level, they have to serve also the higher levels. These problems can be eliminated by designing fat tree topologies, by creating “fatter” links on the lowest levels of tree (Fig. 2-5b). While fat trees overcome the tree performance problem, they introduce irregularity in the physical design of the routers and do not improve significantly the fault tolerance.

In ring topologies each node is attached along the same path. The path has a shape of a ring. The unidirectional ring is the simplest form of point-to-point interconnection. The bidirectional ring is 1D-torus that is characterized by its simplicity and low bisection bandwidth (Fig. 2-6a). The simplicity of the ring reduces the complexity at each node that reduces buffer, area and energy requirements. The main drawbacks of the ring are high



(a) A bidirectional ring (1D-torus) (b) A hierarchical ring of depth 3

Figure 2-6. Examples of ring networks: Bidirectional ring (a) and hierarchical ring (b).

average hop count for larger number of nodes and lack of path diversity. The ring is also not fault tolerant. The structure contains local rings connected by a central ring calls hierarchical ring (Fig. 2-6b). The local rings can be treated as the children of the central ring. Similar to trees, the depth of the network grows logarithmically as a function of the local rings and the total number of nodes. In comparison to the ring topology, hierarchical rings can offer better fault isolation. If more than one global ring per local ring is implemented, we can get also higher bisection bandwidth and some path diversity. However, still the fault tolerance and path diversity are poor. Moreover, hierarchical rings are susceptible to performance bottlenecks.

The n-dimensional Cube Connected Cycles (CCC) topology is derived from the 2-ary *n*-mesh (or 2-ary *n*-cube) with each node replaced by a ring of size *n* (Fig. 2-7). Thus, an *n*-dimensional CCC has $n2^n$ nodes. The CCC has node degree for all routers equal *n*,

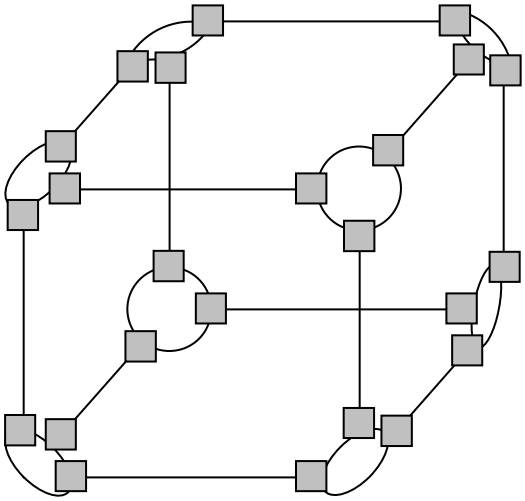


Figure 2-7. 3-dimensional cube connected cycles with 24 nodes.

good bisection bandwidth and acceptable latency. Drawbacks of the CCC are long wires that consume large chip area, lack of the same bandwidth for every node and lack of scalability – adding nodes to the network requires placing a node to each cycle in the cube.

2.2. Flow Control

In a CMP, PEs exchange data in the form of messages. The size of messages may differ, depending on the application and network architecture. Messages are divided into one or more packets. Packets are units of information for network and they have a restricted maximum length. A packet may be further divided into flow control units – flits. A flit is the smallest unit of information recognized by the flow control. Finally, a flit can be divided into one or more physical transfer units – phits, which can be transferred across a physical channel in a single clock cycle. Practically in many designs flit is equivalent to phit [12], [102].

Flow control describes allocation of NoC's resources (channel bandwidth, buffer capacity and control state) for packets traversing the network. Flow control policy decides if packet should be dropped, blocked in place, buffered or rerouted. A well designed flow control allocates these resources effectively in order to get good bandwidth and low latency. There are two approaches to flow control:

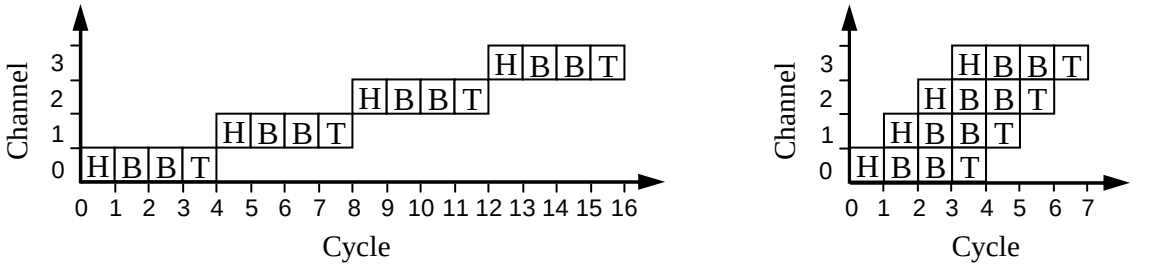
- Problem of resource allocation – resources have to be assigned to each packet that traverses network (e.g. routing algorithm determines, which resources are allocated to packets),
- Problem of resolving contention – when an outgoing channel is requested by packets arriving on different inputs, flow control mechanism has to allocate this channel to one packet and do something with others, e.g. block or drop.

The flow control can be classified as a buffered or bufferless. Bufferless approach is the simplest form of flow control that does not use buffers for storing packet in the routers. As it takes place in circuit switching [18], a physical path from source to destination node is reserved before first packet is sent. The time of setting up the path is called setup time, similarly the time needed to release the path is called tear-down time. Circuit switching exhibits high throughput because the bandwidth is guaranteed due to the reserved resources, however the disadvantage is poor network utilization during the setup and tear-down times, when data is not transmitted. Buffered approach is more complex, however it is more efficient form of flow control [34], [41], [62], thus the rest of this paper contains only buffered techniques. Buffering can be done in units of packets

(store-and-forward [28] and cut-through [62] flow control) or flits (wormhole [89], virtual-channel [30] or express-virtual-channel [75] flow control).

2.2.1. Store-and-forward, cut-through and wormhole flow control

Store-and-forward method sends packets towards their destination node without establishing a path before, like it took place in the circuit switching. Each packet flows through network independently, possibly along different paths – the paths are assigned dynamically. Each packet must be stored in whole at each node and then forwarded to a selected adjacent node (hence the name store-and-forward). Each router must provide enough storage space to buffer at least one packet. The selection of next node on the path is made by routing algorithm. Each packet needs to carry addressing bits (header) with information regarding its destination. Since resources are not allocated ahead, there is a possibility that two or more packets would like to get access to the same resources at the same time (contention). In such case, one packet has to be granted the resource before another packet. The delay caused by contention varies and depends on the level of traffic in the network. Network channels are also not allocated to specific source-destination path, thus each network channel is shared by many paths. In the Fig. 2-8a a time-space



(a) Store-and-forward (b) Cut-through and wormhole

Figure 2-8. Time-space diagram with 4-flit packet sent over 4-hop path with no congestion.

diagram for a store-and-forward flow control is presented. The entire 4-flit packet is forwarded over the 4-hop path without contention. The entire packet (all 4 flits: H – head flit, B – two body flits and T – tail flit) is sent over one channel before it can proceed to the next channel. The main drawback of store-and-forward flow control is its very high latency – the packet has to be completely received at one node before it can begin moving to the next node, that causes serialization latency.

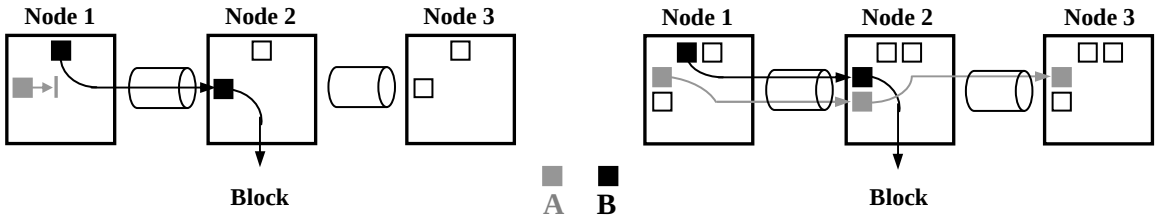
The latency penalty of store-and-forward method is eliminated in cut-through flow control. A packet is forwarded as soon as the head flit is received without waiting for the entire packet to be acquired (Fig. 2-8b). Similarly like it is in store-and-forward technique, cut-through reserves buffer space in unit of packets, so that in case of blocking the whole packet has to be buffered.

By allocating buffers in unit of packets, cut-through and store-and-forward make very inefficient use of buffer space. The buffer has to be large enough to store at least one packet and the size of the packets is limited by the storage space. We can get much more effective use of storage by allocating buffers in units of flits – and this is an idea of wormhole flow control. The packet is forwarded as soon as its head flit is received (like cut-through – Fig. 2-8b), but buffer space is allocated only for several flits instead of for the entire packet. The head flit governs the route and along its advance along the specified route, the remaining flits follow in a pipeline fashion. If the head flit meets a busy channel, it is blocked until the channel becomes available. Rather than buffering the remaining flits (whole packet) by removing them from the network (like it is in cut-through), the wormhole flow control can buffer only part of the packet (few flits). The remaining flits are blocked and they are kept in the buffers over multiple routers along

the path. In this way one input channel of all involved routers is occupied, that in overall bill gives multiple occupied channels along the path. The channel is released when the last (tail) flit has been transmitted through the channel. The advantage of wormhole flow control is that large packet buffers at each node are eliminated by a small flit buffers. This saving of area is crucial for the NoC implementation, because buffers in routers are the major area consumers [129]. Furthermore, the packet length in wormhole does not depend on the buffer size. Also in the absence of contention, wormhole technique makes network latency insensitive to path length. However in case of contention, saturation throughput is lower in comparison to cut-through.

2.2.2. Virtual-channel flow control

Virtual-channel flow control eliminates the drawback of lower saturation throughput in wormhole technique, while preserving its advantages such as small required buffer space and the packet length independence on the buffer size. Virtual-channel method creates logically separate channels that share the same physical channel. It leads to the possibility of existing flits of many packets in the channel. Practically, Virtual Channels (VCs) are flit buffers associated with a single physical channel. By introducing VCs, packets are forwarded in the network over them and that separates allocation of buffers from allocation of channels. A blocked packet blocks only the VC of a physical channel, but the other VCs can still use the physical channel. It is illustrated in the Fig. 2-9, where virtual-channel flow is compared with wormhole flow control. Flits of the packet *B* are in buffer in the nodes 1 and 2, and they are stuck in (due to contention) node 2. In Fig. 2-9a, where flow control is directed by wormhole technique, the packet *A* that is sent to node 3 is blocked behind packet *B.* In the same situation but for virtual-channel flow control



(a) Wormhole (b) Virtual-channel

Figure 2-9. Comparison of virtual-channel with wormhole flow control.

(Fig. 2-9b), the packet *A* can pass packet *B* and proceed to node 3. The physical channel is blocked only when all its VCs are blocked, however probability of such a situation is lower than the probability of blocking in wormhole technique. In this way, utilization of physical channels and network throughput is higher.

The benefits of the virtual-channel technique result in area, power and latency overhead due to the cost of a more complicated control and buffer implementation. The virtual-channel has to allocate a VC and then physical channel for the VC, while the previous flow controls allocate only physical channels for packets. It leads to an additional stage of arbitration and allocation in the routers. However beside improved performance, VCs deliver the other advantages, such as:

- Avoiding deadlocks by adding VCs to links and choosing routing schemes properly, so that cycles in the resource dependency graph can be broken [32] (this issue is discussed also in the Section 2.3 of this work),
- Optimizing wire utilization by sharing physical channels by many logical channels [102],

- Providing quality-of-service by allowing high priority packets move before those of lower priority or by providing required service level on dedicated VCs [44].

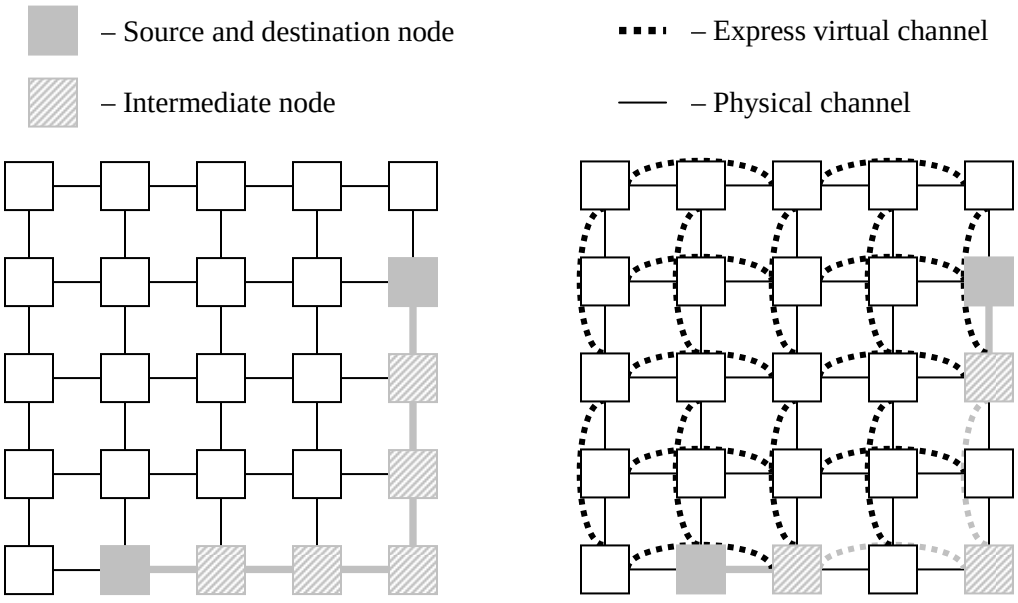
A virtual-channel router realization is pipelined – in the router a packet has to pass through several pipeline stages till resources are allocated to it. The way of implementing VCs requires careful analysis. In high traffic rates under uniform traffic, increasing number of virtual channels per physical channel can raise performance, but e.g. in case of hotspot traffic, assigning deeper buffers to less number of virtual channels gives better results [99].

2.2.3. Express-virtual-channel flow control

To overcome some of the earlier mentioned limitations of the virtual-channel technique, novel, express-virtual-channel flow control was introduced. The key idea of express-virtual-channel technique is to provide virtual express lanes in the network, which allow bypassing intermediate routers by skipping the router pipeline. It is realized by introducing express buffers that define Express Virtual Channels (EVCs). By implementing EVCs, packets can be virtually bypassed through intermediate nodes. The virtual bypassing in a router forwards EVC flits as soon as they reach the router without any buffering and arbitration, that significantly reduces packet latency and router energy consumption.

The express-virtual-channel flow control consists of two kinds of virtual channels at each port of a router: 1-hop Normal Virtual Channels (NVCs) that are the regular virtual channels and *k*-hop EVCs that carry flits *k*-hops at a time. The head flit is allowed to choose either a NVC or an EVC depending on their availability and path of the flit. When

transmission is over a *k*-hop EVC, the flit is allowed to bypass the router pipeline at the next intermediate *k*-1 nodes. In the Fig. 2-10, 2-hop EVCs in 5-ary 2-mesh topology are



(a) 5-ary 2-mesh

(b) 5-ary 2-mesh with express virtual channels

Figure 2-10. Transmission in regular structure (a) and in structure with express virtual channels (b).

shown and compared with NVCs solution. In the regular (NVCs) case (Fig. 2-10a) seven nodes are fully involved in transmission, while in structure with EVCs (Fig. 2-10b) number of fully involved nodes is reduced to five (for two nodes router pipeline is bypassed). EVCs are designed statistically at each router and they are prioritized over normal virtual channels. Beside lower latency (up to 84%) and better throughput (up to 23%), EVCs reduce router switching activity (packet is going through less number of

routers), limit number of buffers, and reduce contention that makes this solution energy (up to 38%) and area efficient [71].

The analysis presented in the Chapter 2.2 led the author to choose virtual-channel and its express channel extension as flow control for the presented work.

2.3. Routing

Routing is the procedure of selecting a path from a source node to a destination node in a particular topology. Besides topology and flow control, a routing algorithm is also an important factor in performance of the NoC. A good routing algorithm balances the load in the network channels, routes paths as fast as possible and is still able to work in the presence of faults. It should also be easily implemented in the hardware.

Routing can be classified in several ways. Based on place where the routing decision is made, we can distinguish between source and distributed routing [41]. In source routing, the entire path is selected before the packet is sent. The routing information must be carried by each packet that increases the packet size. Moreover, the chosen path cannot be changed after the packet has left the source. Second class of routing – distributed routing – is mostly used by direct networks. In this approach, routing decision for the packet is made in each router. Routers decide whether the packet should be delivered to the local PE or forwarded to a neighboring router. In order to reduce the network latency, a routing decision has to be made as quick as possible.

Routing can also be classified as [34]:

- Deterministic – always chooses the same path between a pair of nodes, even if there are multiple paths,

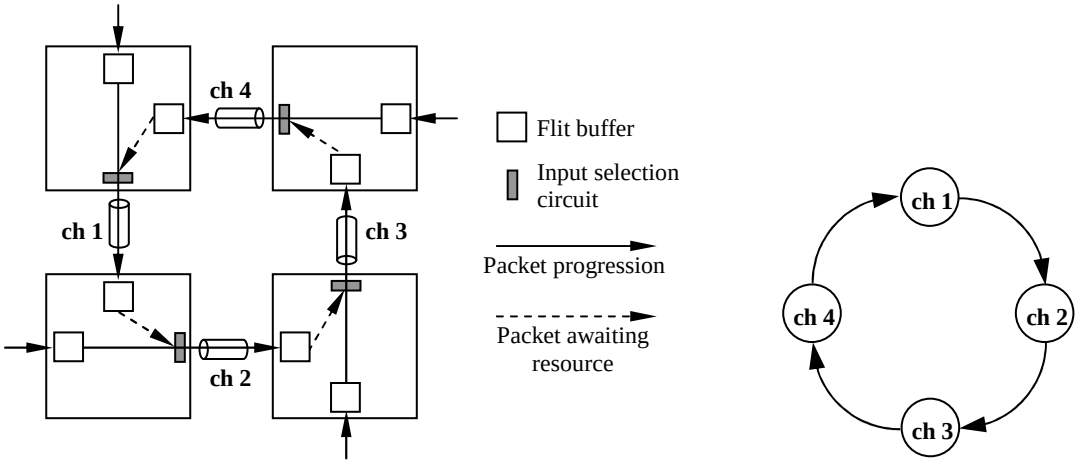
- Oblivious – routes packets without considering the network's state (deterministic algorithms are a subset of oblivious),
- Adaptive – uses information about the network's state (e.g. channel load information, length of queues for resources, etc.) to make routing decisions.

If the path selected by a routing algorithm is the shortest path between the source and destination, the algorithm is said to be minimal. Using a minimal routing algorithm, every traversed channel brings the packet closer to the destination. In a non-minimal routing, the chosen path can be longer, that allows reacting to current network condition.

An important property of routing algorithms is freedom from deadlocks and livelocks. Deadlock occurs, when packets are waiting for each other in the cycle. Livelock is caused by packet, which proceeds through network indefinitely and never arrives at its destination.

2.3.1. Deadlocks and livelocks

Deadlock [94] is a scenario, where packet delivery is postponed indefinitely, thus a set of packets can be blocked forever in the network. It can happen when packets are allowed to hold some resources (usually buffers or channels) while requesting others. Such a situation is presented in Fig. 2-11a, where channel deadlock involving four routers and four packets. Each packet (arrow) traverses one router straight and enters a second router, where it wants to turn left. In order to make a turn, each packet has to wait (dashed arrow) for the requested channel to become free. However the requested channels will become free only when some of the packets advance and release their channels, that will never occur. Thus, the four packets are deadlocked and will never



(a) Circular resource dependency (b) Resource dependency graph

Figure 2-11. An example of channel deadlock involving four packets.

make progress. In order to analyze deadlocks, resource dependency graph was proposed [32]. The channel dependence graph for a direct network and routing algorithm is a directed graph, where nodes represent resources and edges correspond to resource dependencies. For the situation presented in Fig. 2-11a, the resource dependency graph is shown in Fig. 2-11b. In that example, nodes are unidirectional channels (ch 1 to ch 4) and their output edges direct to the resources requested by the current resource owner, in our example the resource owners are packets. If a cycle appears in channel dependency graph, then we deal with deadlock situation.

Two approaches are used in NoC to eliminate deadlocks in routing algorithms (they are discussed later in the Sections 2.3.2 and 2.3.3):

- Deadlock avoidance – design routing algorithms in such a way, that deadlocks will never occur,

- Deadlock recovery – deadlocks are allowed, but mechanisms how to recover from them are designed.

In contrast to deadlock, livelock does not stop a packet, but rather stops its progress toward destination. Livelock occurs when the routing of a packet never leads the packet to its destination. Livelock is possible only for adaptive and non-minimal routing. If there is no limit of the maximum number of times a packet can choose non-minimal path, the packet can remain in the network indefinitely. One approach to livelock avoiding is to implement in the packet a field indicating its progress. It can be the number of times a packet has been routed through non-minimal path. Once the specified value of non-minimal progress reaches a threshold (often called misroute value), only a minimal path can be chosen. Another approach to livelock avoidance is age-based priority filed in the packet. When a conflict between packets occurs, a packet with higher priority (older) is privileged [34].

2.3.2. Deadlock avoidance routing

In order to avoid a deadlock, cycles in the resource dependency graph must be eliminated. One approach to eliminating cycles is by forcing a partial order of the resources and then ensuring that a packet is allocated resources in ascending order [32], [78], [83]. In this way, a cycle can not occur, because any cycle must contain at least one higher-ordered resource holder waiting for a lower-ordered resource, but by the ordered allocation it is not allowed. Deadlock avoidance by resource ordering for virtual-channel flow control was proposed in [32]. Cycles are broken by ordering each physical channel along a cycle into a group of virtual channels. Each group of VCs shares a physical

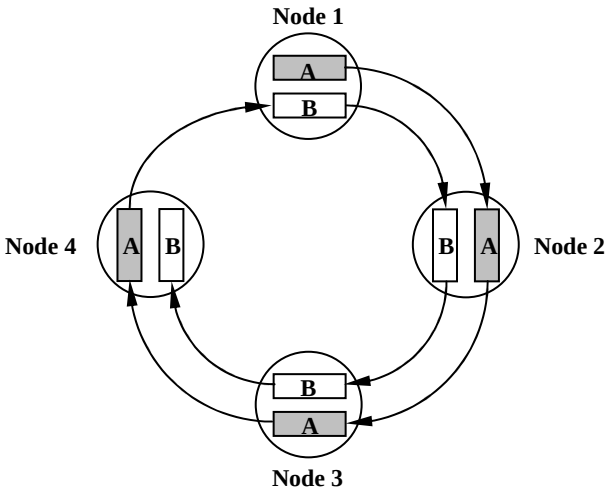


Figure 2-12. Deadlock avoidance by resource ordering.

channel. It is shown in Fig. 2.12. In each node the VCs are duplicated and divided into two classes: A and B. A new packet is injected to the network (Node 1) and is assigned to the VCs of class A. When the packet crosses the border between Node 4 and Node 1, it can use VCs only from class B and cyclic dependencies between resources are avoided. The concept of ordering VCs allowed the authors in [32] to offer deterministic, deadlock-free Dimension Order Routing (DOR) algorithm where e-cube [104] algorithm was extended. In the DOR, each packet is routed in one dimension till its arrival at the proper coordinate in that dimension. Afterwards, the packet proceeds to the next dimension, etc. By enforcing order on the dimensions traversed, deadlock-free routing is guaranteed. E.g. for *k*-ary 2-mesh, each node has address (*x*, *y*) in the mesh. The DOR sends packets first along the dimension *X* (or *Y*) and then along the *Y* (or *X*) dimension.

The DOR sends every packet from source to destination over exactly the same path. A path diversity offered by topology is ignored. Similarly, load balancing and reliability

is very weak. These issues were addressed in [109], where oblivious Valiant's algorithm was proposed. In Valiant scheme, a packet sent from a source to a destination is first sent from the source to a randomly chosen intermediate node and then from that node to the destination. It reduces the load of any traffic pattern. However the good performance given by randomization provides decreased locality. Better load balance can be also achieved by randomizing the order of dimensions in which packet is traversed [34]. At each node either for DOR or Valiant routings *x*-first or *y*-first direction is randomly chosen. Such enhanced algorithms will be named in this work as DOR Load Balanced (DOR-LB) and Valiant Load Balanced (Valiant-LB). DOR-LB provides minimal, load balanced oblivious routing and preserves locality. Both algorithms, of course are recommended to be implemented with VCs in order to ensure deadlock freedom and better channel utilization.

Adding a VC to a physical channel is less expensive than adding a new physical channel, however it is not free. It involves adding buffer space and control logic and also reduces bandwidth of the VCs, because they already share the physical channel. An advantage of adding VCs is that they can support highly adaptive routing algorithms. That concept is used in [78], where a minimal adaptive, the linder-harden algorithm is presented. The idea of virtual interconnection networks that provide adaptability, deadlock freedom and fault tolerance is introduced. Each physical channel is shared by many VCs, whose number depends on how many virtual networks are needed. The VCs can be divided into several groups or virtual networks. When the packet is blocked in a virtual network, it can be forwarded using another virtual network. If minimal routing is not required, deadlock freedom adaptive routing can be provided using fewer additional

VCs. Such a solution is proposed in [31], where two, static and dynamic non-minimal routing algorithms were presented. Both techniques allow the packets to take a longer path if the shortest path is not available. A disadvantage of the resource ordering is increased number of network resources by the implementation of resource classes and providing proper ordering. Furthermore, an imbalance of using divided resources can be observed.

Another way to avoid cycles is restrictions of routing algorithm. A general framework for restricting routing algorithms in *k*-ary *n*-meshes and *k*-ary *n*-cubes networks is the turn model [47], which provides a systematic approach to the development of adaptive, minimal and non-minimal routing algorithms, without adding VCs. A cycle in channel dependency graph occurs because the packet routes contain turns that form the cycle. To design adaptive routing algorithms for *k*-ary *n*-meshes and *k*-ary *n*-cubes, we have to:

- Partition the channels according to the direction in which they route packets,
- Identify the possible turns from one direction to another, ignoring 0-degree and 180 degree turns,
- Recognize the simple cycles these turns can generate,
- Prohibit one turn in each cycle to prevent deadlock,
- For *k*-ary *n*-cubes, include as many turns as possible from a set of wraparound channels,
- Add 0-degree and 180-degree turns that are needed for non-minimal routing algorithms, without reintroducing cycles.

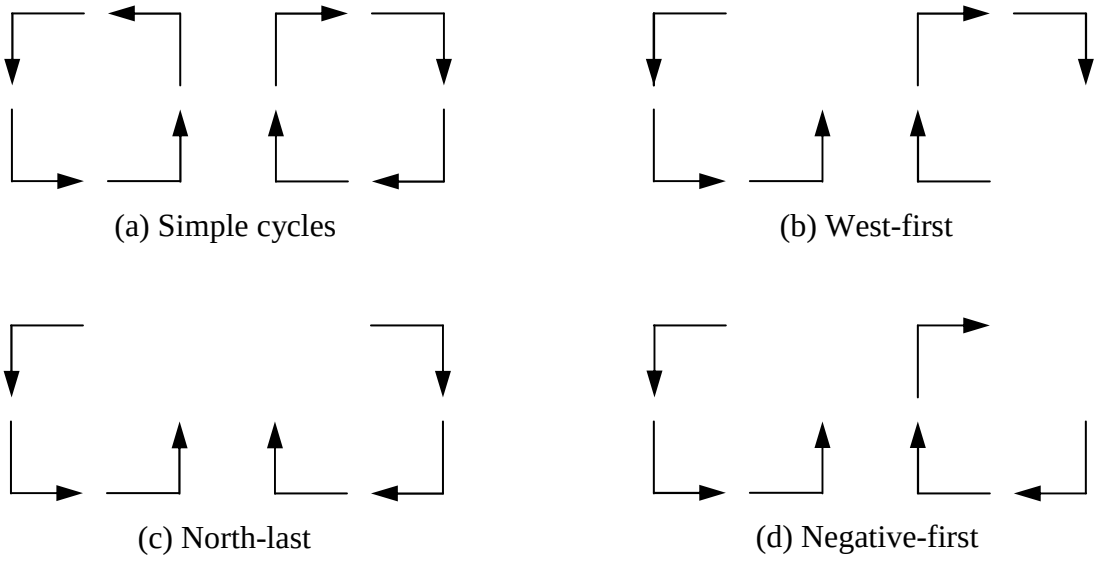


Figure 2-13. The turn model for k-ary 2-mesh network.

Fig. 2-13a shows eight possible turns in a 2D-mesh network and two simple cycles that can be created by combining these turns. As it can be seen, in order to avoid a cycle, at least one turn of each of these two cycles must be eliminated. Together with the turn model, authors in [47] have proposed three deadlock-free routing algorithms constructed by elimination of a turn (Fig. 2-13b, c and d). When the south-to-west turn is eliminated, the west-first algorithm is created (Fig. 2-13b). A packet must take all of its west hops before moving to any other direction. After turning from the west, it can turn in any other direction except west. By removing north-to-east turn, the north-last scheme is generated (Fig. 2-13c). In this technique, a packet can turn everywhere except north. Once a packet turns north it must follow only the north direction. In the negative-first routing algorithm (Fig. 2-13d), the east-to-south turn is excluded. First, a packet must proceed completely in the negative directions (south and west). Afterwards, it changes to the positive

directions (north and east) and stays there until it reaches its destination. The proposed algorithms are deadlock-free, livelock-free, minimal or non-minimal.

The earlier mentioned the DOR routing can also be considered as an example of the turn model (Fig. 2-14). The vertical-to-horizontal turn is eliminated and the DOR is

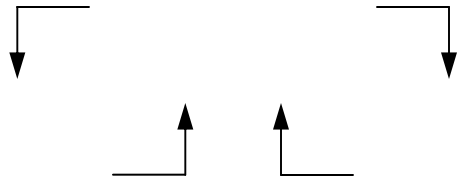


Figure 2-14. The Dimension Order Routing (*DOR*) as restricted version of the turn model.

generated. A packet is first routed completely in the *x* direction, before moving to the *y* direction. After turning from the *x*, it can traverse only the *y* dimension. While the turn model eliminates only one turn in a simple cycle, the DOR excludes two turns, that makes its more restrictive than the turn model.

The turn model and DOR restrict the turns that reduce the path diversity. In case of the DOR, the path diversity is reduced to zero. Moreover, in case of the turn model, the employed limitations do not allow routing messages along any of the shortest paths in the network and algorithms based on that model are partially adaptive (DOR is deterministic). Better partial adaptiveness is delivered by odd-even turn model [23], that is evaluation of the turn model. Instead of prohibiting turns, the odd-even turn scheme is based on restricting the locations, where certain turns can be taken so that a circular wait can never occur. Based on that model, the ROUTE routing algorithm was proposed.

Very popular approach to create routing algorithms is hybrid solution, which combine splitting network resources (VCs) with restricting the paths for packet. Actually, designing of deadlock-free, fully adaptive routing algorithms without virtual channels or redundant physical channels is not possible [34]. A non-minimal planar adaptive routing algorithm [22] restricts the routing freedom to two dimensions at a time and requires three VCs per physical channel. Similarly, authors of the turn model designed fully adaptive non-minimal routing algorithms double-y and mad-y [46] by adding an extra VC. Hybrid method is also used to create fully adaptive minimal routing techniques. Such algorithms are e.g. PFNF [108], algorithm proposed by Duato [40], 3P [103] or mesh_route [14] and they require only two VCs per physical channel to ensure deadlock freedom.

2.3.3. Deadlock recovery routing

Deadlock recovery routing is a totally different approach to dealing with deadlock. It is based on the assumption that deadlocks are generally infrequent [68]. So, instead of avoiding deadlocks, we can also let them appear and recover from them. Such a solution is interesting when we can not afford additional resources or lower the performance associated with deadlock avoidance. Deadlock recovery algorithms contain two major phases: detection and recovery. In the detection phase, a deadlock situation has to be recognized. The detection is usually realized using timeout counters that are placed in each network resource. The counters are reset when data is sent through resource (lack of deadlock). In case, if the counter achieves a given threshold, we treat it as deadlock and recovery phase has to be started. The recovery part can be regressive or progressive.

In regressive recovery, deadlocked packets are removed from the network. Such method is used in Compressionless routing [68]. If a packet is dropped, the sender has to be notified about it. Compressionless routing realizes this by resetting the sender's counter after each new flit is accepted into network. If the counter reaches a threshold value before the last flit is sent to the network, deadlock situation occurs and packet is removed from the network. If the last flit is injected to the network, the source in compressionless routing is ensured that the head flit has already reached destination, so, the packet has already allocated VCs in whole path and it can not be blocked. However the packet has to be long enough in order to ensure, that when the head flit reaches destination at least one flit still remains in the source. If the packet is not long enough, it is extended by appending empty flits.

The progressive recovery approach eliminates deadlock situation without removing deadlocked packet from the network. That method is used in Disha recovery scheme [6], where recovery from deadlock is performed by a single additional flit buffer at each node, which is the minimum required for a progressive recovery scheme. This special escape buffer is a special input buffer shared in each router (a floating VC) and it is used only when deadlock is detected. The escape buffers form kind of free path. Once such a deadlock situation occurs, one of the packets in the cycle is switched to that free path and routed minimally along this path until destination. By reaching destination, the dependency cycle is broken. The Disha scheme does not waste network resources by sending and removing packets, like it is in compressionless routing. Thus it has potentially higher performance. The Disha scheme was improved in [7], where instead of using sequential recovery, concurrent solution was proposed.

Deadlock recovery techniques are solutions to eliminate the need of additional resources to avoid deadlock. They are useful only if deadlocks occur not so often. In another case, the overhead produced by deadlock detection and recovery would degrade the overall performance. The drawback of regressive recovery can be eliminated by progressive solution, where in deadlock case removing and sending again a packet is not necessary. However efficient implementation of Disha requires an additional central buffer [41]. Moreover, for long packets recovering from deadlock is not so fast, that leads other deadlocked packets to wait for escape buffer. It increases latency and reduces throughput.

The list of routing algorithms proposed in the literature is almost infinite. However the most researched and developed are deadlock avoidance routing techniques. Also present and future scaling allows for greater flexibility and resource possibilities on the chip. Thus the hybrid, deadlock avoidance techniques are placed on the top of available routing solutions.

2.4. Proposed energy model

Energy consumption of a VLSI system becomes one of the most important costs. It is related to design aspects such as thermal and power constrains. A model of power consumption of network routers was proposed in [116]. The bit energy in router ($E_{R_{bit}}$) is defined as the dynamic energy consumed while traversing one bit of data through the router:

$$E_{R_{bit}} = E_{S_{bit}} + E_{B_{bit}} + E_{W_{bit}} \tag{2-1}$$

, where $E_{S_{bit}}$ is energy consumed by the switch arbitration, $E_{B_{bit}}$ is buffering energy (buffer write and read energy) and $E_{W_{bit}}$ is energy consumed by interconnection wires inside the switching fabric (energy required to traverse the crossbar switch). Besides energy consumed by router, we have to also consider energy consumed on the physical channels between tiles ($E_{L_{bit}}$). Thus, the average energy consumed in sending one bit of data from a tile to its neighboring tile can be calculated as:

$$E_{bit} = E_{R_{bit}} + E_{L_{bit}} \tag{2-2}$$

Consequently, the average energy consumption of sending one bit of data from tile t_i to tile t_j is:

$$E_{bit}^{t_i,t_j} = E_{R_{bit}}(N_{hops} + 1) + E_{L_{bit}} N_{hops} \tag{2-3}$$

, where N_{hops} is the number of channels traversed by a packet between tile t_i and t_j.

In the equation 2-3 the $E_{R_{bit}}$ and $E_{L_{bit}}$ are constants for a given design. In [113] authors presented performance analysis of wires. The estimated $E_{L_{bit}}$ in [*pJ/bit*] for NoC is:

$$E_{L_{bit}} = 0.39 + 0.12 l_{wire} \tag{2-4}$$

, where l_{wire} is the length of a physical channel in [*mm*].

The energy model assumed in this thesis is proposed for two topologies chosen in the Chapter 2.1: *k*-ary 2-mesh and *k*-ary 2-cube. In the 2D-mesh, the assumed length of the physical channel equals length of the PE edge. That length reported in [110] is 1.5 [*mm*], so for the 2D-mesh we can assume l_{wire} = 1.5 [*mm*]. For 2D-torus folded version is considered, that doubles length of channels in comparison to 2D-mesh, thus for *k*-ary 2-cube l_{wire} = 3 [*mm*]. Finally, the energy consumed on the physical channels between tiles

in [*pJ/bit*] is $E_{L_{bit}}^{2D-mesh} = 0.57$ for *k*-ary 2-mesh and $E_{L_{bit}}^{2D-torus} = 0.75$ for *k*-ary 2-cube topology.

The research in [113] is extended in [115], where gate level power simulation of VC router proposed in [60] is performed. The router routes packets according to DOR scheme. The amount of energy required for a single bit to pass the router is $E_{R_{bit}} = 0.98$ [*pJ/bit*].

In [67] authors present implementation of router for routing algorithms based on VCs: DOR and fully adaptive hybrid algorithm. The presented results show, that average energy per packet for both DOR and adaptive algorithms under uniform and transpose traffic patterns is similar. Thus, for both DOR and adaptive routing techniques router energy $E_{R_{bit}} = 0.98$ [*pJ/bit*].

For NoCs based on virtual channel flow control, the $E_{R_{bit}}$ can be expressed as [75]:

$$E_{R_{bit}} = E_{S_{bit}} + E_{B_{bit}} + E_{W_{bit}} + E_{VC_{bit}} \tag{2-5}$$

, where $E_{VC_{bit}}$ is energy consumed by VC arbitration. In the express-virtual-channel flow control, packet traveling EVC is able to bypass the router pipeline of intermediate nodes without buffering, VC and switch arbitration. Thus, it saves $E_{B_{bit}}$, $E_{VC_{bit}}$ and $E_{S_{bit}}$, that reduces $E_{R_{bit}}$. Synthesis results presented in [60] show that energy required to traverse the crossbar switch $E_{W_{bit}}$ is 24% of all energy consumed by router. Thus, energy of bit traversing through EVC is 24% of energy of bit traversing through NVC:

$$E_{R_{bit}}^{EVC} = 0.24 E_{R_{bit}}^{NVC} = 0.23\,[pJ/bit] \tag{2-6}$$

Thus finally, for all considered routing algorithms (DOR, DOR-LB, Valiant, Valiant-LB and hybrid adaptive), the average energy consumption of sending one bit of data from tile t_i to tile t_j can be expressed by:

$$\text{For } k\text{-ary 2-mesh: } E_{bit}^{t_i,t_j} = 0.98(N_{hops}^{NVC}+1) + 0.23N_{hops}^{EVC} + 0.57N_{hops} \quad (2\text{-}7)$$

$$\text{For } k\text{-ary 2-cube: } E_{bit}^{t_i,t_j} = 0.98(N_{hops}^{NVC}+1) + 0.23N_{hops}^{EVC} + 0.75N_{hops} \quad (2\text{-}8)$$

, where N_{hops}^{NVC} is the number of NVCs traversed by a packet between tile t_i and t_j, N_{hops}^{EVC} is the number of EVCs traversed by a packet between tile t_i and t_j, and N_{hops} is the number of physical channels (*number of EVCs* + *number of NVCs* – 1) traversed by a packet between tile t_i and t_j.

Model of energy described in equations 2-7 and 2-8 is used in this thesis. It provides an efficient approximation for considered level of abstraction for the NoC architectures presented in this work. Similar models have been used in other works, e.g. [42], [53], [60]. The experimental results performed in [42] showed, that an average error between the considered model and other models is equal only 4.2%.

CHAPTER 3

PROCESSOR ALLOCATION

Modern CMPs are expected to support a multiuser environment in which many parallel jobs are executed simultaneously. Besides job scheduling, an efficient processor allocation is critical component to such CMPs. Job scheduling is done by Job Scheduler (JS) that is responsible for selecting the next job to be executed. Processor allocation is done by Processor Allocator (PA) that deals with selection of a set of PEs for the job selected by the JS (Fig. 1-1).

In a 2D-mesh and 2D-torus topology based CMP, an incoming job is described by the size of the subgrid it requires. The JS selects the next job from the system queue for execution using a proper scheduling policy. For a job selected by the scheduler, the PA tries to find an available subgrid. If such a free subgrid does not exist, then the scheduler handles the job according to the implemented policy (e.g. the JS waits until a submesh is released or a smaller job is sent from the queue to the PA). The jobs are allocated in such a manner that they do not overlap with each other, and if they are allocated, they run until completion. There are two major processor allocation strategies: Contiguous and Non-contiguous. As explained in the Chapter 1.2, this dissertation addresses contiguous approach.

The contiguous scheme has a tendency for external fragmentation, which occurs when there are enough free processors for a particular job, but it is not allocated due to lack of contiguously or different topology as NoC. It leads to a critical property of processor allocation algorithm – subgrid recognition ability (ability to find free subgrids for incoming jobs). If an algorithm can always find an available subgrid (if it exists) for

an incoming request, we say that the algorithm has complete subgrid recognition ability. However, such implementations with recognition completeness increase the complexity of the processor allocator [117]. Another important property of allocation algorithm is its speed. A good solution is supposed to have complete subgrid recognition ability, with low allocation overhead.

3.1. Definitions and Notations

3.1.1. *k*-ary 2-mesh

A *k*-ary 2-mesh (2D-mesh) topology, denoted by $M(w, h)$, consists of $w \times h$ nodes arranged in a $w \times h$ 2D grid. Each node in the mesh refers to a PE. The node in column c and row r is identified by address $<c, r>$, where $0 \leq c < w$ and $0 \leq r < h$. A non-boundary node $<c, r>$ is connected by direct communication channel to its neighboring nodes, $<c \pm 1, r>$ and $<c, r \pm 1>$. A boundary node has two or three neighboring nodes depending on its location within the entire mesh.

Definition 3-1: A 2D submesh $S(p, q)$ in the mesh $M(w, h)$ is a subgrid $M(p, q)$ such that $1 \leq p \leq w$ and $1 \leq q \leq h$. A job requesting a submesh $p \times q$ is denoted by $J(p, q)$. A submesh S is identified by its base (lower left node) and end (upper right node) and is denoted as $S[<x_b, y_b> <x_e, y_e>]$.

Definition 3-2: A node is busy if it has been allocated to a job. A busy submesh β is a submesh where all of its nodes have been allocated to jobs.

Definition 3-3: A node is free if it is not allocated to any job. A submesh is free when all of its nodes are available (are free).

Definition 3-4: A busy array of a mesh $M(w, h)$ is a bit map $B[w, h]$, in which element $B[c, r]$ has a value 1 or 0 if node $<c, r>$ is busy or free respectively.

Definition 3-5: A busy list is a set of all busy submeshes in the system. Similarly, a free list is a set of all free submeshes in the system.

Definition 3-6: The coverage of a busy submesh β with respect to a job J is denoted by $\xi_{\beta,J}$ and it is a set of processors such that use of any node in $\xi_{\beta,J}$ as the base of free submesh for the allocation of J will cause the job J to overlap with β. The coverage set with respect to J is denoted by C_J and it is the set of the coverages of all busy submeshes.

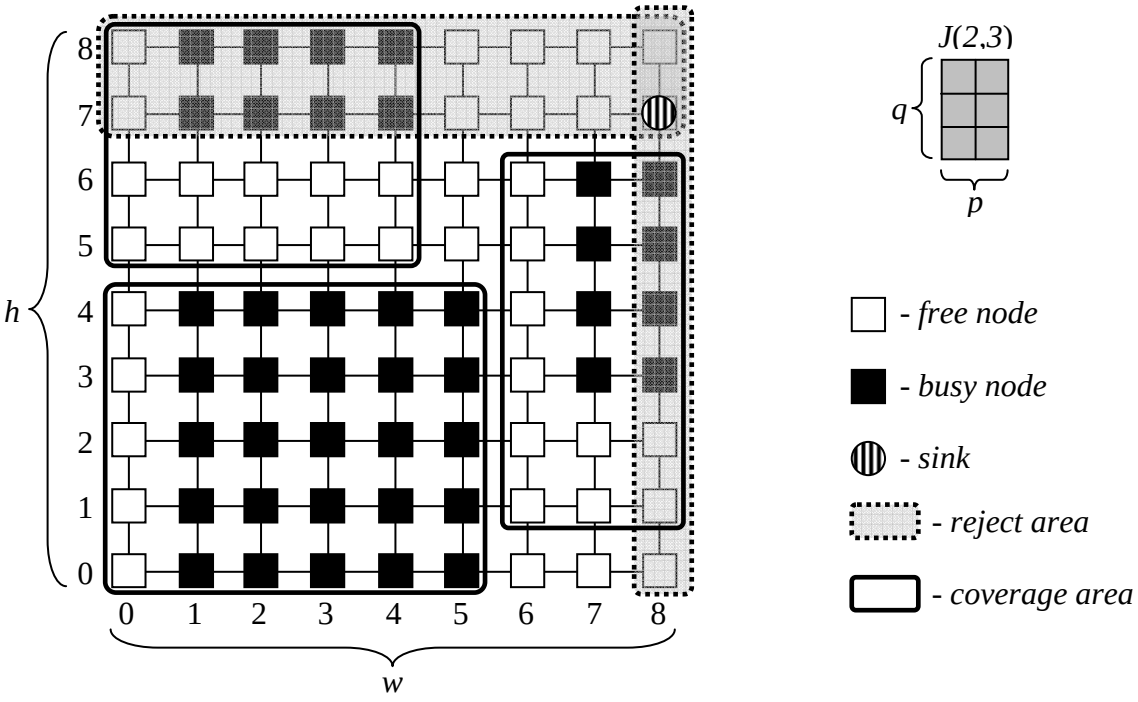
Definition 3-7: The reject area of submesh with respect to a job J, denoted by R_J, is a set of processors such that use of any node in R_J as the base of free submesh for the allocation of J will cause the job J cross the boundary of the mesh.

Definition 3-8: The sink of the reject area is the processor with coordinates $<w - p + 1, h - q + 1>$.

Definition 3-9: A base block with respect to a job J is a submesh whose nodes can be used as base for free submeshes to allocate job J. A set of disjoint base blocks is called the base set.

Definition 3-10: External fragmentation is the ratio of the number of free processors to the total number of processors in the mesh, when the allocation of incoming task fails but there is sufficient number of free processors.

As an example, a mesh $M(9, 9)$, busy and free nodes, sink, coverage areas, reject areas, busy array, busy and free lists with respect to $J(2, 3)$ are presented in Fig. 3-1. The busy nodes are marked using black color while free using white color. The reject area is denoted by the shaded region with doted edges. The coverages of busy submeshes β_1=[<1, 0> <5, 4>], β_2=[<7, 3> <8, 6>] and β_3=[<1, 7> <4, 8>] are $\xi_{\beta1,J}$=[<0, 0> <5, 4>],



Busy list:

<1, 0> <5, 4>
<7, 3> <8, 6>
<1, 7> <4, 8>

Free list:

<0, 0> <0, 8>
<6, 0> <8, 2>
<6, 3> <6, 6>
<1, 5> <5, 6>
<5, 7> <8, 8>

$$B[9, 9] = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3-1. A mesh $M(9, 9)$, busy and free nodes, sink, coverage areas, reject areas, busy array, busy list and free list.

$\xi_{\beta 2,J}$=[<6, 1> <8, 6>] and $\xi_{\beta 3,J}$=[<0, 5> <4, 8>] respectively. The coverage set $C_J = \xi_{\beta 1,J} \cup \xi_{\beta 2,J} \cup \xi_{\beta 3,J}$. The reject area R_J = [<0, 7> <8, 8>] ∪ [<8, 0> <8, 8>] and its sink has coordinates <8, 7>. The base set is [<5, 5> <5, 6>] ∪ [<6, 0> <7, 0>]. Both internal and external fragmentations in the presented case are equal to zero. For a given job J, $C_J \cup R_J$

represents the set of processors, which can not be the base of the free submeshes. Thus the base set for J is $Z - C_J - R_J$, where Z refers to the set of all processors in the system. It is important to note, that the base set with respect to $J(p, q)$ is different from this with respect to $J(q, p)$, when $p \neq q$.

3.1.2. *k*-ary 2-cube

A k-ary 2-cube (2D-torus) topology, denoted by $T(w, h)$, consists of $w \times h$ nodes arranged in a $w \times h$ 2D grid. Each node in the torus refers to a PE. The node in column c and row r is identified by address $<c, r>$, where $0 \leq c < w$ and $0 \leq r < h$. A node $<c, r>$ is connected by direct communication channel to its neighboring nodes $<c \pm 1, r>$ and $<c, r \pm 1>$, where in case if:

- $c = -1$, $c \leftarrow w - 1$,
- $r = -1$, $r \leftarrow h - 1$,
- $c = w$, $c \leftarrow 0$,
- $r = h$, $r \leftarrow 0$.

Thus each node has four neighboring nodes.

Definition 3-11: A 2D subtorus $S(p, q)$ in the torus $T(w, h)$ is a subgrid $T(p, q)$ such that $1 \leq p \leq w$ and $1 \leq q \leq h$. A job requesting a subtorus $p \times q$ is denoted by $J(p, q)$. A subtorus S is identified by its base (lower left node) and end (upper right node) and is denoted as $S[<x_b, y_b> <x_e, y_e>]$. In contrast to the 2D-mesh topology, in toruses x_b can be greater than x_e, similarly, y_b can be greater than y_e, however the base still remains as lower left corner with end on the upper right node (from a job point of view).

Definition 3-12: A node is busy if it has been allocated to a job. A busy subtorus β is a subtorus, where all of its nodes have been allocated to jobs.

Definition 3-13: A node is free if it is not allocated to any job. A subtorus is free when all of its nodes are available (are free).

Definition 3-14: A busy array of a torus $T(w, h)$ is a bit map $B[w, h]$, in which element $B[c, r]$ has a value 1 or 0 if node $<c, r>$ is busy or free respectively.

Definition 3-15: A busy list is a set of all busy subtoruses in the system. Similarly, a free list is a set of all free subtoruses in the system.

Definition 3-16: The coverage of a busy subtorus β with respect to a job J is denoted by $\xi_{\beta,J}$ and it is a set of processors such that use of any node in $\xi_{\beta,J}$ as the base of free subtorus for the allocation of J will cause the job J to be overlapped with β. The coverage set with respect to J is denoted by C_J and it is the set of the coverages of all busy subtoruses.

Definition 3-17: A base block with respect to a job J is a subtorus whose nodes can be used as base for free subtoruses to allocate job J. A set of disjoint base blocks is called the base set.

Definition 3-18: External fragmentation is the ratio of the number of free processors to the total number of processors in the torus, when the allocation of incoming task fails but there is sufficient number of free processors.

It is assumed in this dissertation that jobs are always placed in the torus network from left to right in horizontal direction and from bottom to top in vertical direction. It is illustrated in Fig. 3-2. The base of the job J_1 is <3, 1> and its end is <0, 2>. The

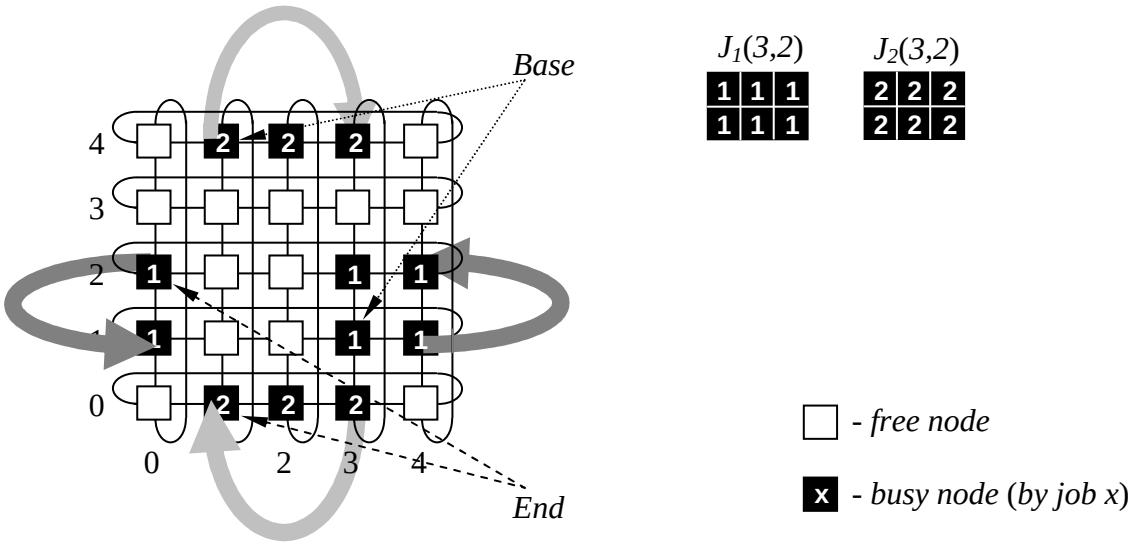
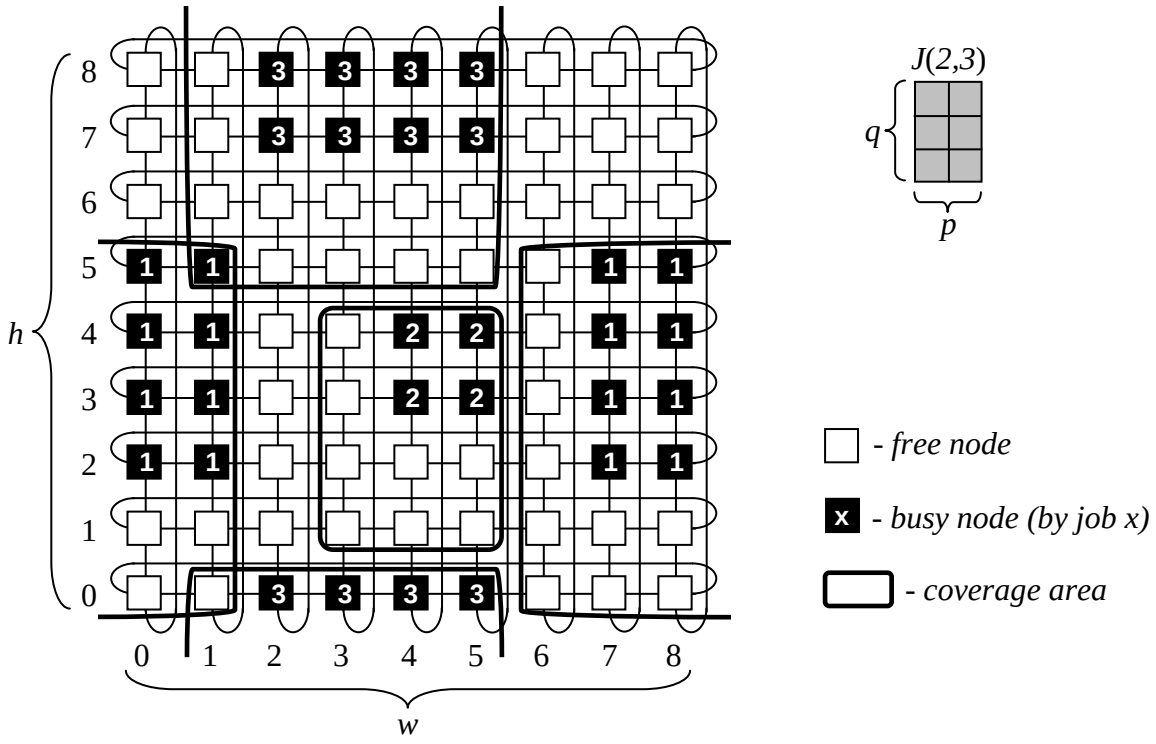


Figure 3-2. Location and orientation of jobs using wraparound channels on a torus network.

beginning of the J_1 is on the right hand side of the torus, but its end is on the left hand side of the network, while the job is placed in left-right fashion. Similarly for job J_2, the base and the end are <1, 4> and <3, 0> respectively. The beginning of J_2 is on the top of the system, but its end is on the bottom of torus, however the job is allocated in bottom-up way.

As illustration of the given definitions, a torus $T(9, 9)$, busy and free nodes, coverage areas, busy array, busy and free lists with respect to $J(2, 3)$ are presented in Fig. 3-3. The busy nodes are marked black while the free are marked white. The busy subtoruses are β_1=[<7, 2> <1, 5>], β_2=[<4, 3> <5, 4>] and β_3=[<2, 7> <5, 0>]. The subtorus β_2 could also be a submesh for the corresponding 2D-mesh topology (the job J_2 does not use wraparound channels). The coverages for given subtoruses are $\xi_{\beta 1,J}$=[<6, 0> <1, 5>], $\xi_{\beta 2,J}$=[<3, 1> <5, 4>] and $\xi_{\beta 3,J}$=[<1, 5> <5, 0>] respectively. The coverage set $C_J = \xi_{\beta 1,J}$



Busy list:

<7, 2> <1, 5>
<4, 3> <5, 4>
<2, 7> <5, 0>

$$B[9,9] = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Free list:

<6, 6> <1, 1>
<2, 5> <5, 6>
<2, 1> <5, 2>
<2, 3> <3, 4>
<6, 2> <6, 5>

Figure 3-3. A torus *T*(9, 9), busy and free nodes, coverage areas, busy array, busy list and free list.

$\cup\ \xi_{\beta 2,J} \cup\ \xi_{\beta 3,J}$. The base set is [<6, 6> <0, 8>] ∪[<2, 1> <2, 4>]. Both internal and external fragmentations in the presented case are equal to zero.

For a given job J, C_J represents the set of processors, which can not be the base of the free subtoruses (for meshes it is $C_J \cup R_J$). Thus the base set for J is $Z - C_J$ (for a 2D-mesh topology it is $Z - C_J - R_J$), where Z refers to the set of all processors in the system. In the case of k-ary 2-cube NoCs, we do not have reject area R_J (Def. 3-7). By introduction wraparound channels in 2D-torus, it is not possible to get a job that would cross boundary of the torus, thus R_J does not exist. Similarly, a 2D-torus topology does not have a sink (Def. 3-8). For a k-ary 2-cube the base set with respect to $J(p, q)$ is different from this with respect to $J(q, p)$, when $p \neq q$, as was the for a k-ary 2-mesh.

3.2. Analysis of Allocation Algorithms

Increasing the efficiency of allocation algorithms for 2D-meshes was subject of a lot of research. However, for torus networks, authors have focused on high dimensional k-ary n-cubes [20], [26], [38], [66], [96], [118] and specifically, allocation problem for CMPs in a 2D-torus topology is not addressed. In this section, analysis of proposed processor allocation techniques for k-ary 2-mesh topologies is presented.

3.2.1. Busy Array (Bit Map) Approach

The solution with a bit map representing the allocation status of processors in the mesh is presented in [122]. Based on the idea, two allocation schemes are presented: the First Fit (FF) and the Best Fit (BF). With respect to an incoming job, the busy array B (without considering reject area R_J) is scanned in to create a coverage array C_T, which is a bit map representing the coverage set. In order to form the C_T in an efficient way, each coverage $\xi_{\beta,J}$ is divided into three regions: job coverage, left coverage and bottom coverage (Fig. 3-4). Then, two scans are necessary:

1. All rows from right to left (determining a job and left coverages).

2. All columns (left to right) from top to bottom (creating a bottom coverage).

The FF strategy returns the first available node that does not belong to the coverage set, while the BF selects the base which has the maximum number of busy neighbors.

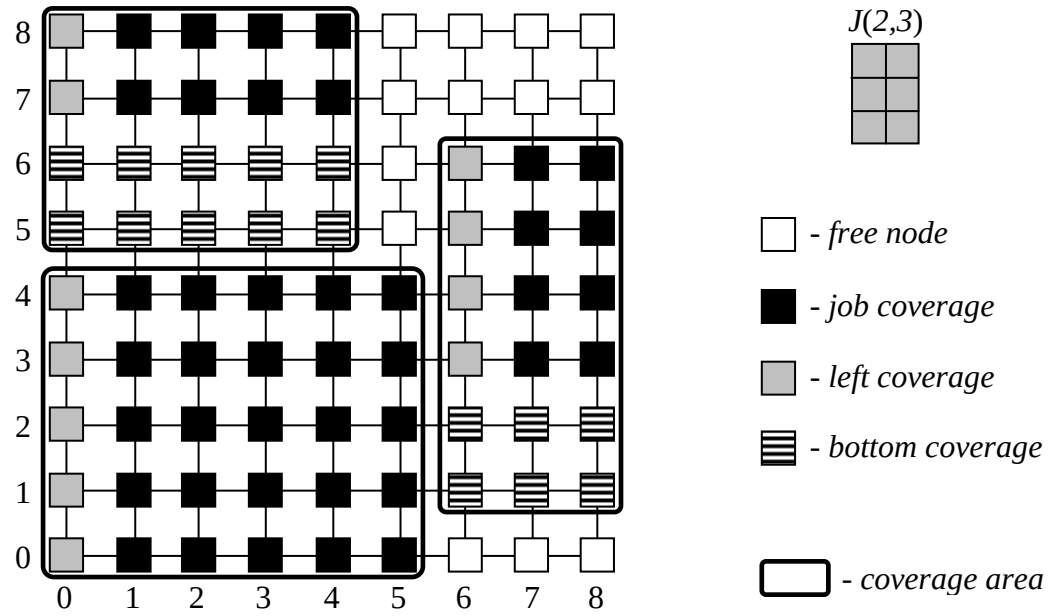


Figure 3-4. Coverages and their three regions with respect to *J*(*2, 3*). Reject area is not shown.

Thus, in each case of the BF strategy, the bit map is scanned two times while for the FF in some cases second scan can be interrupted (when the first base node is found, scanning is interrupted and the node is returned as base). Simulation experiments show that system utilization and external fragmentation are almost the same for both schemes. However the FF algorithm is more efficient and simple, that led to the selection of FF in practical implementations [122].

With the FF solution, achieved time complexity is *O*(*wh*) for allocation. The drawback of the algorithm is lack of recognition completeness, which is the result of considering only the fixed orientation of tasks.

FF Algorithm

1. calculate sink with respect to *J*
2. **for** each row in *B* **do**
   for each node in considered row from right to left **do**
   - create job and left coverages in C_T with respect to *J*

3. **for** each column from left to right in C_T without R_J **do**
   for each node in considered column from top to bottom **do**
   - create bottom coverage in C_T with respect to *J*

   if node can be base **then**
   return node as base for *J* (success)
   exit
4. **return** fail (job is not allocated)
   exit

Deallocation of processors in the systems with a busy array reduces to clearing all the elements in bit map *B*. It is also done in $O(wh)$.

3.2.2. Implementation of Busy List

The replacement of bit map by a list with busy subgrids is proposed in [24], [25], [36], [39], [69], [73], [117] and [120]. As in the case of bit map, each technique with busy list is based on the fact that for any job *J* none of the nodes inside $C_J \cup R_J$ can serve as the base. So, for each *J*, creating C_J and R_J is a common first step for algorithms with busy list. Because of the fact that allocation status of processors is maintained using busy list, C_J is also in a list form and it is constructed by scanning the busy list, and for each β in the list $\xi_{\beta,J}$ is constructed. For a given $\beta=[<x_b, y_b> <x_e, y_e>]$, its coverage with respect to $J(p, q)$ is $\xi_{\beta,J}=[<x_c, y_c> <x_e, y_e>]$, where $x_c=max(0, x_b - p + 1)$ and $y_c=max(0, y_b - q + 1)$. R_J is found by calculating the sink.

Finding the node that is not in $C_J \cup R_J$ and the strategy of choosing one node (if many) is the next step that differs from the above mentioned solutions. The first strategy with busy list was Frame Sliding (FS) [25]. The FS for the given job $J(p, q)$ maintains a

frame of size $p \times q$, which is identified by lower left corner. The frame slides through the mesh, starting from the lowest left available node. When nodes in the currently examined frame are not available, the frame slides over the mesh by taking horizontal and vertical steps equal to the width and height of the frame respectively. The frame concept in the FS is applied by checking if lower left corner of the frame is not in $C_J \cup R_J$ (the coverage list is searched for each node under consideration) and if it fails (lower left corner is in $C_J \cup R_J$) by moving the corner according to the size of the frame. With the FS solution, the achieved complexity is $O(whB)$, where B is the length of the busy list. The FS technique does not have complete submesh recognition ability due to two factors:

1. Fixed strides of frame.
2. Fixed orientation of tasks.

The advanced version of the FS algorithm with recognition completeness is proposed in [39]. The Adaptive Scan (AS) strategy for every requested job J goes through the

AS Algorithm

1. calculate sink with respect to J
2. create C_J with respect to J based on busy list
3. **for** each row in $M(w, h)$ without R_J **do**
   for each node in considered row without R_J **do**
   for each element in C_J **do**
   if node belongs to considered element **then**
   - go to next node

   else if last element in C_J **then**
   return node as base for J (success)
   exit
4. **if** orientation of job was not changed **then**
   - $J(p, q) \leftarrow J(q, p)$
   - go to step 1

   else
   return fail (job is not allocated)
   exit

mesh, node by node, taking horizontal and vertical steps respectively, that differentiates it from the FS. Each node, belonging to $C_J \cup R_J$ is tested by going through the whole coverage list. Membership to $C_J \cup R_J$ causes the algorithm to check another node while in the other case, node can be a base for a given *J*. Recognition completeness is achieved through node by node verification and by considering two job orientations: if allocation of *J*(*p, q*) fails, then *J*(*q, p*) possibility is checked. Similar to FS, the complexity of AS is *O*(*whB*).

Another development in FS and AS strategies is presented in the Quick Allocation (QA) algorithm [120]. In the QA scheme individual checking of the nodes is replaced by testing only each row in the mesh. It was achieved by introducing a one dimensional array called *last_covered*, which remembers the horizontal coordinate (*x*-coordinate) of the rightmost covered node for each row. Thus, instead of going through each node in the row, it is enough to check, if horizontal coordinate in the *last_covered* for the node under

QA Algorithm

1. calculate sink with respect to *J*
2. create C_J with respect to *J* based on busy list
3. sort C_J in the increasing order of x_c
4. **for** each row in *M(w, h)* without R_J **do**
   - *last_covered* in considered row ← -1
   - **for** each element in C_J **do**
     - update *last_covered* to rightmost covered node in considered row
     - **if** value of *last_covered* + 1 is in R_J **then**
       - go to next row
     - **else if** last element in C_J **then**
       - **return** value of *last_covered* + 1 as base for *J* (success)
       - **exit**
5. **if** orientation of job was not changed **then**
   - *J(p, q)* ← *J(q, p)*
   - go to step 1

   else
   - **return** fail (job is not allocated)
   - **exit**

consideration is not in R_J. If so, the *last_covered* value for the row is the base for the requested job *J*. A key issue here is the method of creating and updating the *last_covered* array. It is done by going through the coverage list and analyzing the horizontal coordinates of $\xi_{\beta,J}$. In order to make this analysis possible in a single pass of the coverage list, it is necessary to sort coverages $\xi_{\beta,J}$=[<x_c, y_c> <x_e, y_e>] in the increasing order of x_c. The QA technique is recognition complete and according to computer simulations [117], [120] is faster and more efficient than the earlier described algorithms based on busy list. The complexity of the QA solution is $O(hB)$.

All the algorithms so far presented find the base for the job by maintaining a busy list and scanning the nodes or rows of the mesh. The ADJacency (ADJ) strategy [36] initiates another approach, which eliminates the need for scanning. For requested job *J*, the ADJ compares the corners of *J* with the corners of each $\xi_{\beta,J}$ in the coverage list. If a submesh *J* overlaps with some of the allocated subgrids, the submesh *J* slides along its boundary. It adjoins the busy submeshes as much as possible, that reduces the chances of external fragmentation. The ADJ is recognition complete and as computer simulations showed, only the QA scheme from former busy list algorithms is faster than the ADJ [36], [120]. It is important to note that the ADJ is the first technique, where the allocation time does not depend directly on the size of the mesh, but it varies along with the size of the busy list (we do not have to scan the mesh structure, we search the busy list). The complexity of the ADJ is $O(B^3)$.

The idea from ADJ of getting rid of scanning the nodes or rows and considering only the coordinates of submeshes was the foundation for creating the innovative allocation scheme – the Stack-Based Allocation (SBA) algorithm [117]. The SBA uses coordinate

calculations and spatial subtractions in order to find a base. To increase the efficiency of the algorithm, main steps of the scheme are performed using a stack. For a requested job J, the initial candidate base block I_J is determined and it gives the first set of candidate blocks B_J : I_J=[<0, 0> <x_s – 1 , y_s – 1>], where <x_s, y_s> is the sink. The coverages $\xi_{\beta,J}$ in C_J in form of a busy list are then spatially subtracted from the B_J : if the first coverage $\xi_{\beta,J}$ interests with any block in B_J, then the $\xi_{\beta,J}$ is subtracted from the intersecting block of B_J, that gives a new block (or blocks) for B_J (the B_J is updated). Afterwards, the next $\xi_{\beta,J}$ is checked (if it intersects with any block in the updated B_J) and if so, it is subtracted from B_J , and so on. This continues until all $\xi_{\beta,J}$ in C_J are considered. If B_J is not empty after subtracting all $\xi_{\beta,J}$, any node from B_J can be a base for J. The key idea of the algorithm that makes it very efficient is to implement B_J as a stack. A candidate block on the top of the stack is always compared with next $\xi_{\beta,J}$ to see if they intersect with each other. When new blocks for B_J are generated from a spatial subtraction, they are pushed onto stack, replacing the top element. Each block on the stack has a pointer to the next $\xi_{\beta,J}$ that the block should be compared with. When a block on the stack with a null pointer appears on top of the stack, the desired base block is obtained. If the stack becomes empty, then the allocation fails.

The complexity of the SBA was $O(B^2)$. But it is proved in [121], that the actual time complexity is $O(B^3)$.

Due to very interesting approach to processor allocation problem, the SBA technique is a subject of very intensive research [24], [69], [73]. The algorithm was improved in [24], where the Improved Stack-Based (ISBA) algorithm is proposed. Both, the SBA and ISBA algorithms use manipulation of job orientation to obtain complete submesh

recognition ability. However, when job $J(p, q)$ has both p and q sizes equal ($p = q$) there is no need to change job orientation – because it causes the algorithm to execute two times with the same job $J(q, q)$. This drawback is eliminated in the ISBA.

In systems with busy list, deallocation of a job requires removal of an element from the busy list. It can be done in constant time by implementing pointers from allocated jobs to corresponding elements in the busy list. Thus, the time complexity of deallocation is $O(1)$.

ISBA Algorithm

1. calculate sink with respect to J
2. create C_J with respect to J based on busy list
3. create initial block I_J
4. assign pointer to first available $\xi_{\beta,J}$ for I_J
5. push I_J onto the stack
6. **while** the stack is not empty **do**
   - **if** pointer to $\xi_{\beta,J}$ in block on the top = null **then**
     - **return** block from top as base for J (success)
     - **exit**
   - **else** $k \leftarrow$ pointer to $\xi_{\beta,J}$ in block on the top
   - **if** block on the top intersects with $C_J[k]$ **then**
     - pop up block from the top of the stack
     - perform spatial subtraction $C_J[k]$ from popped up block
     - **for** each candidate block got from spatial subtraction **do**
       - assign pointer to $C_J[k+1]$ for candidate block
       - push candidate block on the stack
   - **else**
     - pointer to $\xi_{\beta,J}$ for top block $\leftarrow$ next $\xi_{\beta,J}$ in C_J
7. **if** orientation of job was not changed **and** $p \neq q$ **then**
   - $J(p, q) \leftarrow J(q, p)$
   - go to step 1

   else
   - **return** fail (job is not allocated)
   - **exit**

3.2.3. Solution with Free List

Another approach to the processor allocation problem is not keeping in memory information about subgrids that are busy, but maintaining a list with free subgrids (keep

list of processors that can be allocated for a requested job) [1], [63], [80]. The allocation seems to be simple and fast. To allocate a job, the free list is scanned to find a free submesh, which is large enough to accommodate the job. Problem appears while deallocating, when a subgrid is released and can be joined with another, that would create greater subgrid (in this way we could accept bigger job if requested). Such an expansion is necessary in order to preserve recognition completeness of the scheme.

The Free List (FL) algorithm [80] maintains a free list, where subgrids can overlap and they are sorted in non decreasing order of their size. So, for a requested job *J*, the free list is searched from the beginning for the first free candidate submesh whose size is equal or larger than *J*. Both orientations of *J* are checked. In order to reduce external fragmentation, four corners of the candidate submesh are considered and one with the highest boundary value is actually allocated (boundary value is calculated similar to the ADJ [36]). The deallocation process involves expansion of free submesh with free subgrids in the free list. This operation is hard and as it is reported in [1], the FL technique can miss sometimes the biggest submeshes that cause the scheme to lack recognition completeness. The allocation and deallocation complexity of the FL is $O(F^2)$, where F is the length of a free list.

The drawbacks of the FL are addressed in [63]. The Free Submesh List (FSL) strategy maintains two lists: the free list with no overlapping free submeshes and the busy list with busy subgrids. For an allocation request, the FSL scans the free list (it is ordered in the no increasing order) and generates a sublist with candidates of the desired size. Then, the algorithm evaluates candidates by using the reservation factor that decreases external fragmentation. In the deallocation process, the requested job *J* is removed from

the busy list and the free list is updated to one initial free submesh I_F=[<0, 0> <w – 1 , h – 1>]. Afterwards, based on the I_F and busy list, new free subgrids are generated. In this way, the largest possible submeshes are always on the free list and their expansion is not required. Furthermore, the FSL scheme is recognition complete. Drawbacks of the solution are the necessity of two lists (busy and free) and higher time complexities: $O(F^2)$ for allocation and $O(F^3)$ for deallocation.

Disadvantages in both the free list techniques, lack of recognition completeness in the FL and high complexity in the FSL case were foundations for creating the newest scheme – the Compacting Free List (CFL) algorithm [1]. The CFL maintains unordered list of possibly overlapping free submeshes. For requested job J the first free subgrid that is large enough to accommodate J is selected. Both orientations of J are considered and also four corners of aspirant subgrid are tested as the base – one with the highest boundary is chosen. If the J was smaller than a candidate submesh, it is subtracted from the submesh and results are added to the head of list. Also, the allocated submesh is subtracted from the remaining free submeshes that overlap with it – results are put on the head of list. Deallocation is divided into two phases: in the first, the deallocated submesh is expanded into elements in the free list, in the second the members of the free list are expanded into subgrids expanded in phase one. Together with two types of proposed expansions, the CFL does not miss sometimes maximal free subgrids, what took place in the FL, thus the CFL is recognition complete. The time complexity of the CFL is linear for both allocation and deallocation, and equals $O(F)$.

All presented techniques with implementation of free list are complicated in both allocation and deallocation phases. Computer simulation results show that even the best

of free list strategies, the CFL scheme with linear time complexity, is not better than previous busy array and busy list strategies [1]. Necessity of frequent expansion, scanning and sorting (the FL) make solutions with list of free subgrids not efficient and less attractive in comparison to the others. In this dissertation, free list techniques are neither considered nor researched.

3.3. Proposed Allocation Algorithms for *k*-ary 2-mesh

The FF and BF algorithms suffer from lack of recognition completeness. Let us consider the case: a free submesh $w \times h$ is available, where $w \neq h$. So, the number of available processors is $w \times h$. Then allocation of job $J(p, q)$ is requested, where $p = h$ and $q = w$, that implies $p \neq q$. The number of requested processors is equal to number of free processors ($p \times q = w \times h$). Let us consider two possible cases:

1. $w > h$: it implies, that $q > p$, so $q > h$. As the free submesh height h is smaller than the height of the requested job q, algorithm will return false (job is not allocated).
2. $w < h$: it implies, that $q < p$, so $w < p$. As the free submesh width w is smaller than the width of the requested job p, algorithm will return false (job is not allocated).

Theorem 3.1: Considering both orientations of job for FF and BF schemes, $J(p, q)$ and $J(q, p)$, guarantees the complete submesh recognition.

Proof: For requested job $J(p, q)$, let us consider two possible cases:

1. $w > h$: it implies, that $q > p$, so $q > h$. As the height of the free submesh h is smaller than the height of the requested job q, algorithm will return false. After changing orientation we have $J(q, p)$, that implies $p > q$, so $h > q$ and job is allocated.

2. $w < h$: it implies, that $q < p$, so $w < p$. As the width of the free submesh w is smaller than the width of the requested job p, algorithm will return false. After changing orientation we have $J(q, p)$, that implies $p < q$, so $p < w$ and job is allocated.

□

Each algorithm presented is characterized by its runtime overhead. In systems that support multitasking and multiprogramming, the PA can be employed many times before a large enough subgrid is found. Every call of allocation procedure in the PA generates additional allocation overhead, which we want to minimize. As it was mentioned earlier, both job orientations are considered in modern algorithms in order to get recognition completeness. In cases where both orientations are considered, schemes run two times: for $J(p, q)$ and $J(q, p)$. However, when job $J(p, q)$ has both p and q parameters equal (p=q) algorithm is executed two times with the same job.

Theorem 3.2: If the algorithm is recognition complete for any job $J(p, q)$, then in case when $p = q$ considering only one orientation guarantees recognition completeness.

Proof: In case of allocation failure for $J(p, q)$, algorithms consider another orientation $J(q, p)$. If the other orientation fails, there is no large enough submesh to accommodate J and the algorithms return false. When $p = q$, we consider job $J(q, q)$. If after first orientation $J(q, q)$ allocation fails, changing orientation on $J(q, q)$ will change nothing, so, failure after first orientation guarantees lack of submesh large enough to accommodate J.

□

By applying theorem 3.2 to allocation algorithms, we are eliminating redundant runtime and reducing allocation overhead.

3.3.1. The Improved First Fit (IFF) Algorithm

As it is mentioned in Section 3.2.1, the FF algorithm performs better than the BF. This work proposes a new IFF algorithm, which is an extension of the FF technique by applying theorems 3.1 and 3.2. The last step of the FF is modified in IFF:

IFF Algorithm

1. }
2. } Steps like for the FF algorithm
3. }
4. **if** orientation of job was not changed **and** $p \neq q$ **then**
   - $J(p, q) \leftarrow J(q, p)$
   - go to step 1

   else
   return fail (job is not allocated)
   exit

By theorems 3.1 and 3.2, the proposed IFF algorithm is recognition complete.

3.3.2. The Improved Adaptive Scan (IAS) Algorithm

The AS algorithm is recognition complete as it considers two orientations of job $J(p, q)$. It considers both orientations in each case, also, when $p = q$. The proposed new IAS algorithm is the AS with the application of theorem 3.2. The last step of the AS is modified in IAS:

IAS Algorithm

1. }
2. } Steps like for the AS algorithm
3. }
4. **if** orientation of job was not changed **and** $p \neq q$ **then**
   - $J(p, q) \leftarrow J(q, p)$
   - go to step 1

   else
   return fail (job is not allocated)
   exit

By theorem 3.2, the IAS is recognition complete.

3.3.3. The Improved Quick Allocation (IQA) Algorithm

The QA technique is also not optimal due to changing job $J(p, q)$ orientation always, even if $p = q$. The proposed IQA is a modified version by applying theorem 3.2 to the last step of the QA:

IQA Algorithm

1. }
2. } Steps like for the QA algorithm
3. }
4. }
5. **if** orientation of job was not changed **and** $p \neq q$ **then**
   - $J(p, q) \leftarrow J(q, p)$
   - go to step 1

   else
   return fail (job is not allocated)
   exit

By theorem 3.2, the IQA is recognition complete.

3.4. Allocation Algorithms for *k*-ary 2-cube

The detailed description of a 2D-torus topology is presented in Section 2.1.4. Generally, a 2D-torus topology is characterized by good path diversity, better load balance and reliability in comparison to 2D-meshes. It is achieved by enriching a 2D-mesh with additional, wraparound channels that connect the external nodes in each row and column. Moreover, in the case of processor allocation, 2D-toruses give better flexibility and possibility of finding free processor for the job.

In Fig. 3-5 a torus network $T(5, 5)$ is shown, where allocation of the job $J_3(4, 1)$ is requested. On the torus, two jobs J_1 and J_2 are already allocated. It is important to notice,

that for equivalent 2D-mesh network $M(5, 5)$, we would have three allocated jobs: J_{1a}, J_{1b} and J_2. Because of the extra channels added in torus topology, separate jobs J_{1a} and J_{1b} in the mesh can be treated as one job J_1 in the torus. Moreover, the torus network in Fig. 3-5 allows us to allocate job J_3, that would not be possible in the case of equivalent mesh $M(5, 5)$. A base for the job J_3 on the given torus could be node <3, 1> (its end is <1, 1>) or <3, 0> (its end is <1, 0>).

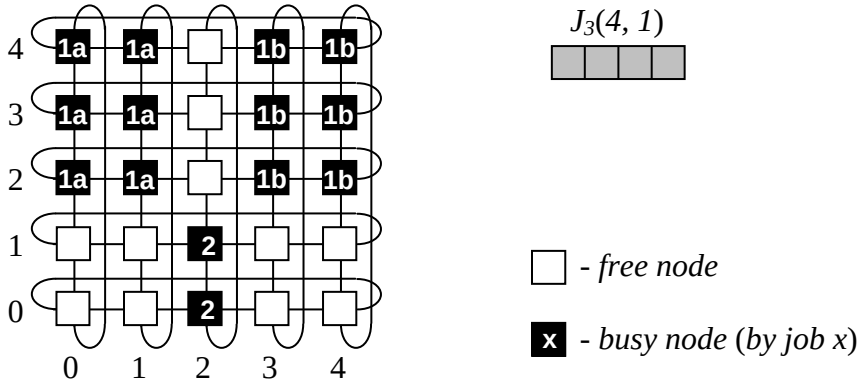


Figure 3-5. A torus $T(5, 5)$ with two allocated jobs $J_1(4, 3)$ and $J_2(1, 2)$, and one job in a queue $J_3(4, 1)$.

3.4.1. Construction of Coverage Set

All processor allocation techniques based on busy array and busy list, create coverage set in one of the first steps.

For algorithms based on busy array, coverages are created by scanning busy array B. In this case, the methodology of creating coverage set C_J for 2D-meshes and 2D-toruses is similar.

In the schemes using a busy list for processor allocation, coverages are determined for each busy subtorus and they depend on the addresses of base $<x_b, y_b>$ and end $<x_e, y_e>$ of

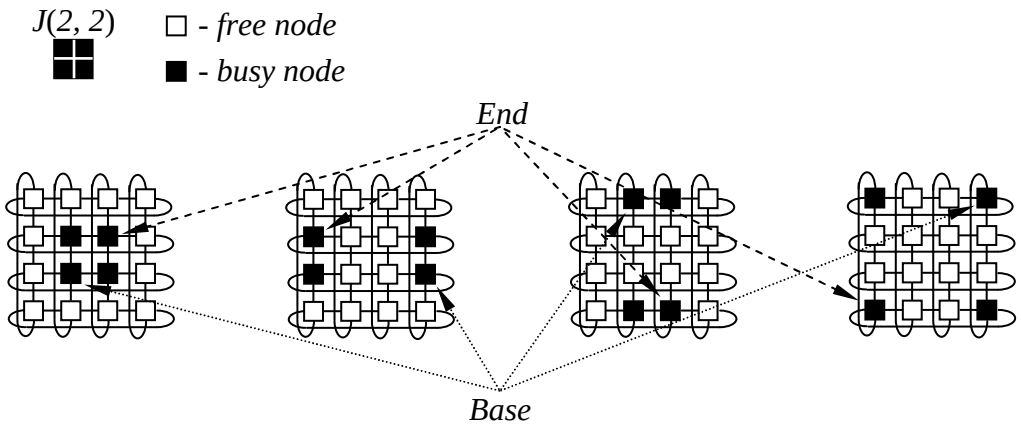


Figure 3-6. Four different cases of a job $J(2,2)$ allocation for a torus network: (a) Regular case known from 2D-mesh networks, where $x_b \leq x_e$ and $y_b \leq y_e$, (b) $x_b > x_e$ and $y_b \leq y_e$, (c) $x_b \leq x_e$ and $y_b > y_e$, (d) $x_b > x_e$ and $y_b > y_e$.

the subgrid. For *k*-ary 2-mesh topology, we had only one possible case of addresses, where $x_b \leq x_e$ and $y_b \leq y_e$ (Fig. 3-6a). In *k*-ary 2-cube topology, we have four possible cases – all of them are shown in Fig. 3-6. For each presented instance, coverages need to be determined in a different way. For a given β=[<x_b, y_b> <x_e, y_e>], its coverage with respect to $J(p, q)$ is $\xi_{\beta,J}$=[<x_1, y_1> <x_2, y_2>], where x_1, y_1, x_2 and y_2 are determined according to the Construction of Coverage algorithm.

3.4.2. Bit Map Allocation for Torus (BMAT) Algorithm

The general idea of the BMAT algorithm is based on the approach used in IFF algorithm. With respect to an incoming job, the busy array *B* is scanned to create a coverage array C_T in the form of bit map.

Each coverage $\xi_{\beta,J}$ is divided into three regions: job coverage, left coverage and bottom coverage (Fig. 3-7) that allows creating a C_T in an efficient way. Then in the worst case, two inspections through a C_T are required:

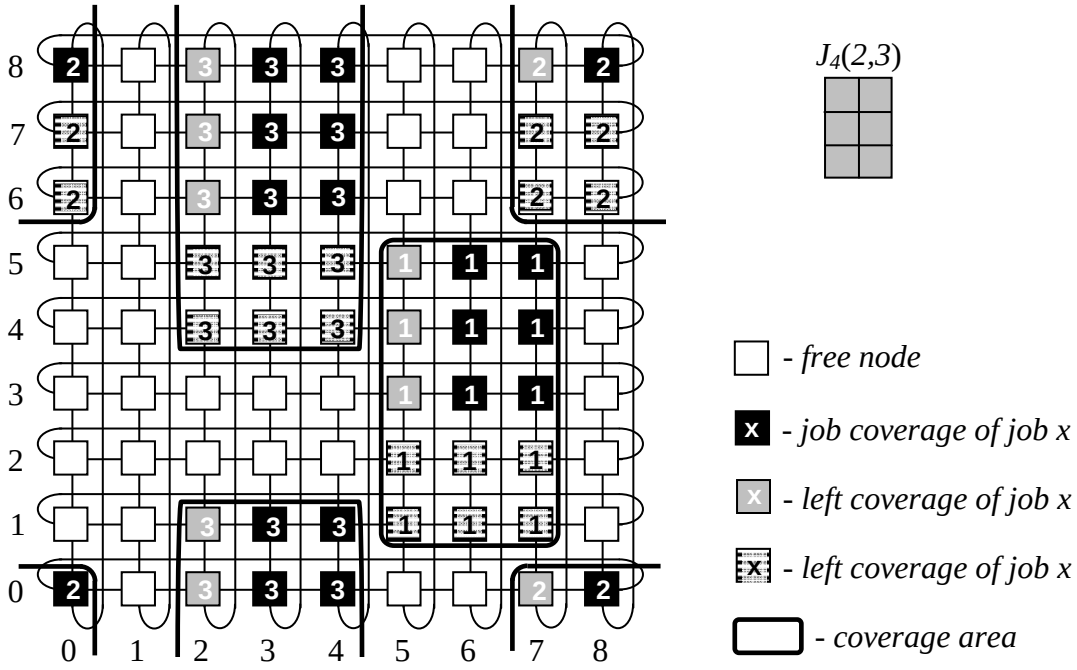


Figure 3-7. Coverages of jobs $J_1(2, 3)$, $J_2(2, 2)$ and $J_3(2, 5)$ with their three regions, with respect to $J_4(2, 3)$.

1. All rows from right to left, each row two times (determining the left coverages of a job).
2. All columns from top to bottom, each column two times (creating a bottom coverage).

The BMAT technique is recognition complete by manipulating the job orientation. If for a given $J(p, q)$ the allocation fails and $p \neq q$, the scheme will change the orientation of the job and then $J(q, p)$ possibility is checked. When both attempts fail, the allocation of the job fails.

Construction of Coverage

1. $x_1 \leftarrow x_b - p + 1$
2. $y_1 \leftarrow y_b - q + 1$
3. $x_2 \leftarrow x_e$
4. $y_2 \leftarrow y_e$
5. **if** $x_b \leq x_e$ **then**
   - **if** $x_1 < 0$ **then**
     - $x_1 \leftarrow w + x_1$
     - **if** $x_1 \leq x_e + 1$ **then**
       - $x_1 \leftarrow 0$
       - $x_2 \leftarrow w - 1$
   - **else**
     - **if** $x_1 \leq x_e + 1$ **then**
       - $x_1 \leftarrow 0$
       - $x_2 = w - 1$
6. **if** $y_b \leq y_e$ **then**
   - **if** $y_1 < 0$ **then**
     - $y_1 \leftarrow h + y_1$
     - **if** $y_1 \leq y_e + 1$ **then**
       - $y_1 \leftarrow 0$
       - $y_2 \leftarrow h - 1$
   - **else**
     - **if** $y_1 \leq y_e + 1$ **then**
       - $y_1 \leftarrow 0$
       - $y_2 = h - 1$

BMAT Algorithm

1. **for** each row in B **do**
   - *iteration* $\leftarrow 0$
   - **while** *iteration* < 2 **do**
     - **for** each node in considered row from right to left **do**
       - create job and left coverages in C_T with respect to J
       - *iteration* $\leftarrow$ *iteration* $+ 1$
2. **for** each column from left to right in C_T **do**
   - *iteration* $\leftarrow 0$
   - **while** *iteration* < 2 **do**
     - **for** each node in considered column from top to bottom **do**
       - create bottom coverage in C_T with respect to J
       - **if** node can be base **then**
         - **return** node as base for J (success)
         - **exit**
     - *iteration* $\leftarrow$ *iteration* $+ 1$
3. **if** orientation of job was not changed **and** $p \neq q$ **then**
   - $J(p, q) \leftarrow J(q, p)$
   - go to step 1
   - **else**
     - **return** fail (job is not allocated)
     - **exit**

Theorem 3.3: The time complexity of the BMAT algorithm is $O(wh)$ for both, allocation and deallocation.

Proof: In step 1 of the algorithm, bit map B is scanned row by row from left to right and each row has to be scanned two times, that gives $O(2wh) = O(wh)$. In step 2, in the worst case, all columns in the coverage array C_T are scanned from top to bottom, two times each, that gives also $O(2wh) = O(wh)$. In step 3, in the worst case, the above operations are repeated one more time, so finally the procedure runs in $O(wh)$.

Deallocation of processors in the BMAT reduces to clearing elements in the bit map B, that can be done also in $O(wh)$.

□

3.4.3. Busy List Allocation for Torus (BLAT) Algorithm

The BLAT technique is based on the strategy employed in the IAS scheme. For an incoming job $J(p, q)$, the BLAT scans a busy list and creates coverage set C_J, which is also in a list form. Both busy and coverage lists contain coordinates of each β and $\xi_{\beta,J}$ respectively. When C_J is created, each node is tested for membership in C_J that is done by inspecting the whole C_J for every node. Node which is not in the C_J can be a base for the given J, in other case, the algorithm checks another node. The BLAT scheme is recognition complete.

Theorem 3.4: The time complexity for allocation in the BLAT algorithm is $O(whB)$, where B is the size of busy list – number of busy submeshes. Deallocation is done in $O(1)$.

Proof: Step 1 goes through busy list and creates C_J. It is done in $O(B)$. The most expensive is step 2. In the worst case, the scheme goes node by node through all rows and

BLAT Algorithm

1. create C_J with respect to J based on busy list
2. **for** each row in $T(w, h)$ **do**
   for each node in considered row **do**
   for each element in C_J **do**
   if node belongs to considered element **then**
   - go to next node

   else if last element in C_J **then**
   return node as base for J (success)
   exit
3. **if** orientation of job was not changed **and** $p \neq q$ **then**
   - $J(p, q) \leftarrow J(q, p)$
   - go to step 1

   else
   return fail (job is not allocated)
   exit

columns and checks C_J for the considered node. It can be done in $O(whB)$. The procedure of checking if the node under consideration is an element of C_J is complicated for torus networks. However, it can be designed by using groups of conditional expressions (Section 3.4.1), that takes $O(1)$. Step 3, in the worst case repeats the above operations one more time, so the final time complexity of the BLAT is $O(whB)$.

Deallocation of nodes in the BLAT reduces to removal of an element from the busy list. It can be done in $O(1)$, by implementing pointers from allocated jobs to corresponding elements in the busy list.

□

3.4.4. Sorting Allocation for Torus (SAT) Algorithm

The SAT strategy has its origin in the IQA technique. Based on C_J, a one dimensional array *last_covered* is created. The array remembers the horizontal coordinate (x-coordinate) of the right most covered node for each row. It limits scanning of each row

and column only to searching the *last_covered* array. The SAT requires all $\xi_{\beta,J}$=[<x_1, y_1> <x_2, y_2>] in the C_J to be sorted in increasing order of less *x*-coordinate. The SAT technique is characterized by recognition completeness.

Theorem 3.5: The time complexity for allocation in the SAT algorithm is $O(hB)$, where B is the size of busy list – number of busy submeshes. Deallocation is done in $O(1)$.

Proof: Similarly like for the BLAT, the step 1 is done in $O(B)$. In the step 2, finding the less *x*-coordinate takes $O(1)$ and sorting C_J takes $O(B \log_2 B)$. In step 3, for each row in a torus, *last_covered* array is created, that takes $O(Bh)$. Step 4, in the worst case, repeats the above operations one more time. Steps 2 and 3 are most expensive in the algorithm. However, for measurable size of torus, h dominates $\log_2 B$, so $O(hB)$ is the time complexity of the allocation for the SAT.

Deallocation reduces to removing the subtorus from a busy list, so it is $O(1)$. □

SAT Algorithm

1. create C_J with respect to J based on busy list
2. sort C_J in the increasing order of less x-coordinate
3. **for** each row in *T(w, h)* **do**
   - *last_covered* in considered row ← -1
   - **for** each element in C_J **do**
     - update *last_covered* to rightmost covered node in considered row
     - **if** value of *last_covered* + 1 ≥ *w* **then**
       - go to next row
     - **else if** last element in C_J **then**
       - **return** value of *last_covered* + 1 as base for *J* (success)
       - **exit**
4. **if** orientation of job was not changed **and** $p \neq q$ **then**
   - *J(p, q)* ← *J(q, p)*
   - go to step 1

   else
   - **return** fail (job is not allocated)
   - **exit**

3.4.5. Stack-Based Allocation for Torus (SBAT) Algorithm

The SBAT algorithm is an enhanced version of the ISBA scheme with support for torus networks. The SBAT also uses coordinate calculations, spatial subtractions and stack in order to find a base. In comparison to the ISBA, the SBAT needs more complex parts, such as: creating C_J, checking intersection and performing spatial subtraction between subtoruses. These extensions were necessary due to torus properties described in Section 3.4.1. The SBAT technique is recognition complete.

Theorem 3.6: The time complexity for allocation in the SBAT algorithm is $O(B^3)$, where B is the size of the busy list – number of busy submeshes. Deallocation is done in $O(1)$.

Proof: Time complexity of allocation for the SBA algorithms was subject of intensive research [117], [121]. Based on that research, complexity of the SBA is $O(B^3)$. We show that complexity of parts developed by us is $O(1)$, that does not have an impact on the final complexity of algorithm $O(B^3)$.

Step 1 is extended due to specific properties of torus described in Section 3.4.1. However, C_J can be created by scanning a busy list and applying conditional expressions. It is done in $O(B)$. In step 5, checking intersection and performing spatial subtraction are the most critical parts, which have also been developed by us. Checking the intersection for torus can be done in $O(1)$. Similarly, spatial subtraction can be also done in $O(1)$ by using bunches of condition operations. The stack operation is described using a state diagram as in [121]. It takes $O(B^3)$. Thus, $O(B^3)$ is the time complexity of the allocation for SBAT.

Deallocation reduces to removing the subtorus from a busy list, so it is $O(1)$.

□

SBAT Algorithm

1. create C_J with respect to J based on busy list
2. create initial block I_J
3. assign pointer to first available $\xi_{\beta,J}$ for I_J
4. push I_J onto the stack
5. **while** the stack is not empty **do**
   - **if** pointer to $\xi_{\beta,J}$ in block on the top = null **then**
     - **return** block from top as base for J (success)
     - **exit**
   - **else** $k \leftarrow$ pointer to $\xi_{\beta,J}$ in block on the top
   - **if** block on the top intersects with $C_J[k]$ **then**
     - pop up block from the top of the stack
     - perform spatial subtraction $C_J[k]$ from popped up block
     - **for** each candidate block got from spatial subtraction **do**
       - assign pointer to $C_J[k+1]$ for candidate block
       - push candidate block on the stack
   - **else**
     - pointer to $\xi_{\beta,J}$ for top block $\leftarrow$ next $\xi_{\beta,J}$ in C_J
6. **if** orientation of job was not changed **and** $p \neq q$ **then**
   - $J(p, q) \leftarrow J(q, p)$
   - go to step 1

   else
   - **return** fail (job is not allocated)
   - **exit**

3.5. Evaluation of Allocation Performance

In order to evaluate the performance of the presented algorithms, an experimentation system was created and extensive computer simulations were conducted. Eight presented algorithms have been compared, namely: IFF, IAS, IQA, ISBA, BMAT, BLAT, SAT and SBAT.

3.5.1. Experimentation System

During investigations, the following criteria for algorithms quality evaluation were used:

- Allocation time t_a – defined as time needed to allocate the given job J. It contains time needed to find the base for a J and time for allocating the job (the allocation for a

bit map reduces to updating a busy array *B*, for a busy list the allocation is done by placing busy subgrid on the busy list),

- Simulation time t_s – defined as total time of simulation. It is the time needed to run simulation – allocate and process all given jobs,
- System load *L* – defined as the ratio of the number of busy processors (N_b) to the total number of processors available in the system (N_a):

$$L = \frac{N_b}{N_a} * 100\% \tag{3-1}$$

- External Fragmentation F_e – defined as the ratio of the number of free processors (N_f) to the total number of processors in the system (N_a), when the allocation of incoming task fails but there is sufficient number of free processors:

$$F_e = \frac{N_f}{N_a} * 100\% \tag{3-2}$$

The investigations of the considered algorithms were performed on the Intel Pentium 4 machine (2 × 3GHz processor) with 2 GB of RAM. Experimentation system was developed in C.

The logical structure of the input-output system is described by the relation *E = R (A, P)* and presented in Fig. 3-8. The elements of the system are:

- Controlled input *A*: processor allocation algorithm being an element of the set {IFF, IAS, IQA, ISBA, BMAT, BLAT, SAT, SBAT},
- Problem parameters:
  - P_1: number of jobs in the queue (the queue is organized in FCFS fashion),
  - P_2: the range of uniformly distributed pseudorandom numbers for the size of each job *J*(*p, q*) in the queue (range of *p* and *q*),

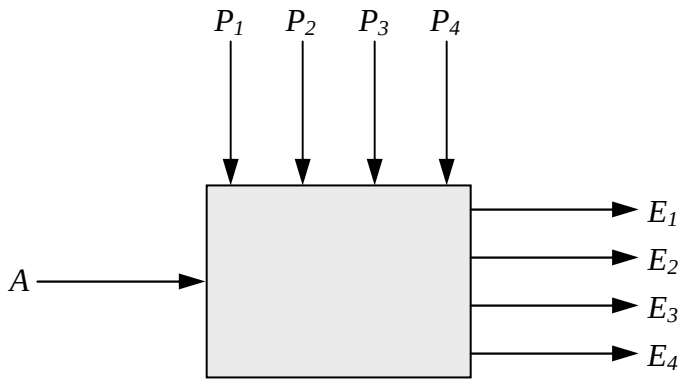


Figure 3-8. Block-diagram of the defined input-output system.

- P_3: the range of uniformly distributed pseudorandom numbers for execution time of each job in the queue,
- P_4: size of torus $T(w, h)$ (or mesh $M(w, h)$) – size of w and h.

- Outputs:
  - E_1**:** denoted by t_a – allocation time,
  - E_2**:** denoted by t_s – simulation time,
  - E_3**:** denoted by L – system load,
  - E_4**:** denoted by F_e – external fragmentation.

3.5.2. Analysis of Results

Using simulation system described in the previous section, two experiments were conducted. Experiment 1 was focused on comparing the allocation time for the considered techniques. The impact of more advanced structure of torus topology on allocation speed was investigated as well. For IFF, IAS, IQA and ISBA, mesh $M(w, h)$ was considered. Similarly, for the BMAT, BLAT, SAT and SBAT, torus $T(w, h)$ was considered. The parameter P_4 was equal: 5×5, 10×8, 10×10, 15×10, 20×20, 30×30,

40×40, 100×100, 500×500, 600×600, 700×700, 800×800, 900×900 and 1000×1000. In the experiment 1 for each of the P_4:

- P_1 was calculated according to the rule:

if $w * h < 1000$ **then**
- $P_1 \leftarrow 1000$

else
- $P_1 \leftarrow w * h$

- P_2: $(1 \div \lceil 0.4 * w \rceil) \times (1 \div \lceil 0.4 * h \rceil)$,
- P_3: $1 \div 1000$ [ms].

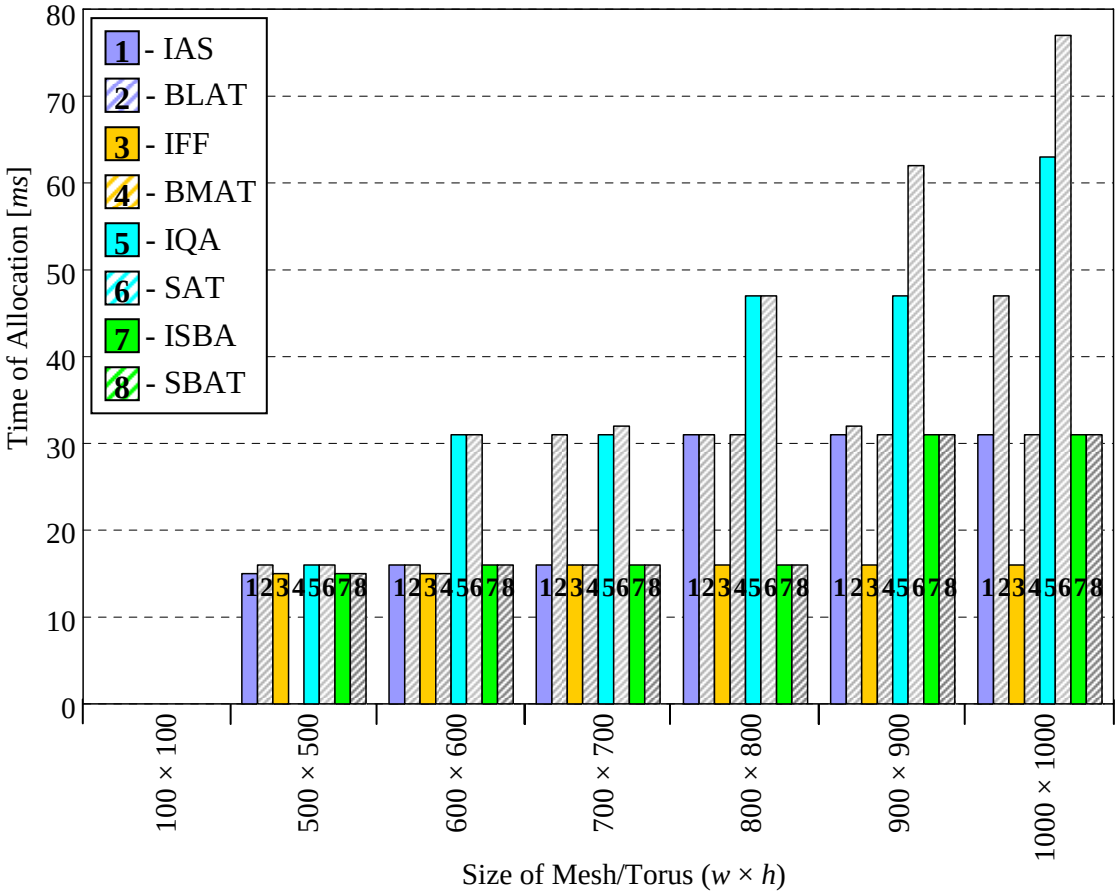


Figure 3-9. Median allocation time in function of size of grid.

Experiment 1 was run in such a way that for the first set of input parameters $P_1 \div P_4$, all considered algorithms were investigated. After that, the set of input parameters was changed and again for this new set all the algorithms were examined, etc. Results of experiment 1 are presented in Fig. 3-9 and 3-10, where median allocation time and average allocation time, respectively, for each algorithm are presented. As we can notice, allocation times for all toruses/meshes smaller than 100 × 100 are the same for all considered schemes and they are oscillating around 0[*ms*]. In Fig. 3-9, we observe that more complex structure of algorithms for torus does not have significant impact on allocation time. We can notice slight differences, however, they can be neglected due to

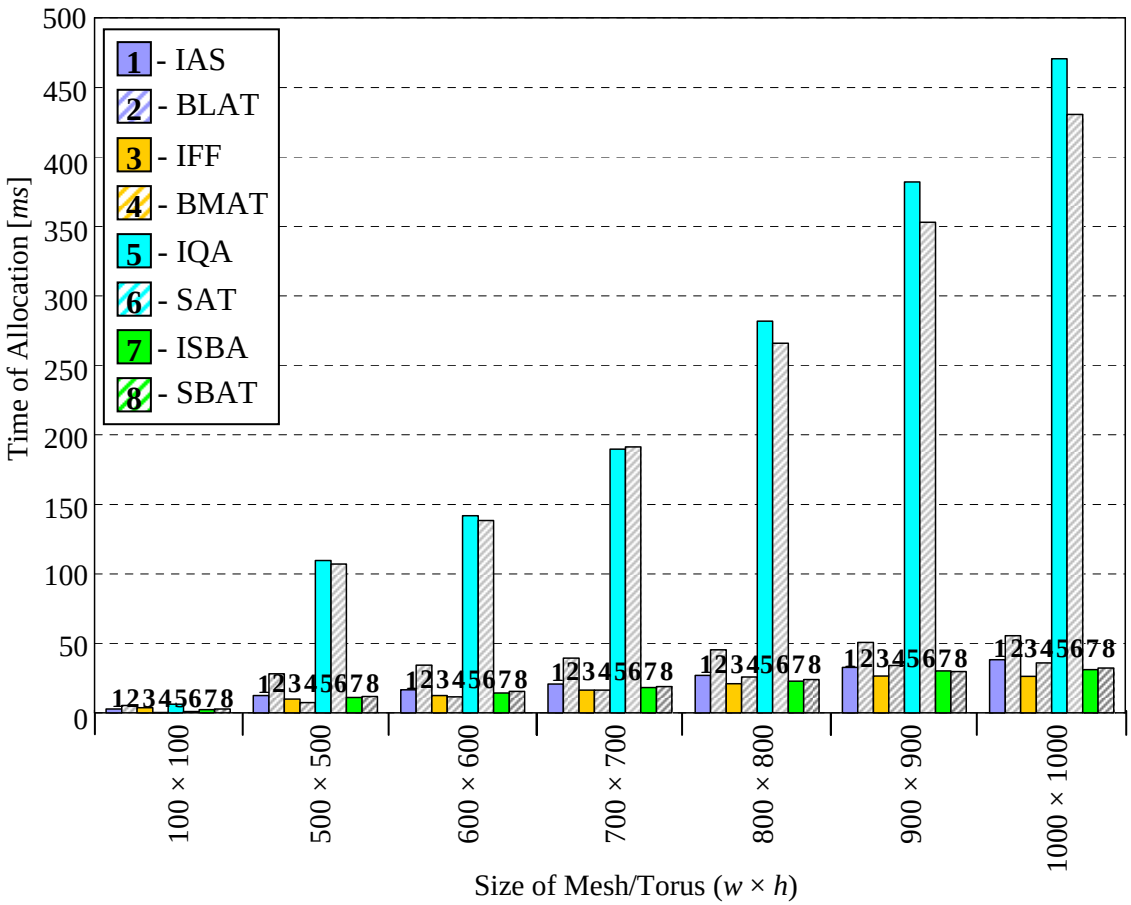


Figure 3-10. Average allocation time in function of size of grid.

simulation error – experiments were done in the multitasking operating system, that can cause little measurement errors. Similarly, average allocation time presented in Fig. 3-10 also confirms that algorithms for toruses are not slower.

Comparison between each techniques reveals, that schemes where sorting is implemented (the IQA and the SAT) have significantly worse allocation time than other techniques, for toruses/meshes greater then 500 × 500. Bottleneck is the sorting operation required by the algorithms. The outcomes of the IQA are in contradiction to earlier research [117], [120], where speed of the algorithm was one of the best. During investigations several $O(B \log_2 B)$ sorting techniques were used, including merge sort and heap sort, but in all cases results were similar. The ambiguity in speed among techniques with sorting appears due to quality of experiments: All previous experiments, were done for networks less than 100 × 100 (32 × 32 in [117] and 80 × 80 in [120]). In our experiment, the IQA performed well, also for sizes less than 100 × 100 that confirmed earlier outcomes.

The best results are achieved for the BMAT and SBAT algorithms (torus) and for the IFF and ISBA schemes (mesh). These techniques are very fast for any size of torus/mesh.

Experiment 2 investigates the performance of CMP for each considered algorithm. Experiment 2 also finds what type of topology for CMP is actually more efficient (regardless of the used allocation algorithm) and what processor allocation algorithm ensures the best system performance. For IFF, IAS, IQA and ISBA, mesh $M(w, h)$ is considered. Similarly, for BMAT, BLAT, SAT and SBAT, torus $T(w, h)$ is considered. For the experiment, three sets of problem parameters are defined:

- Set 1: P_1: 10000,

P_2:

- (1 ÷ 5) × (1 ÷ 5),
- (1 ÷ 10) × (1 ÷ 10),
- **(**1 ÷ 15) × (1 ÷ 15),
- (1 ÷ 20) × (1 ÷ 20),

P_3: 1 ÷ 1000 [ms],

P_4: 50 × 50,

- Set 2: P_1: 10000,

  P_2:

  - (1 ÷ 10) × (1 ÷ 10),
  - **(**1 ÷ 15) × (1 ÷ 15),
  - (1 ÷ 20) × (1 ÷ 20),

  P_3: 1 ÷ 1000 [ms],

  P_4: 75 × 75,

- Set 3: P_1: 10000,

  P_2:

  - (1 ÷ 10) × (1 ÷ 10),
  - **(**1 ÷ 15) × (1 ÷ 15),
  - (1 ÷ 20) × (1 ÷ 20),

  P_3: 1 ÷ 1000 [ms],

  P_4: 100 × 100.

In experiment 2, for first set for each P_2 (four in set 1), all proposed algorithms are examined. After that, the next set was applied and again for each P_2 all the algorithms are examined, etc.

Outcomes of experiment 2 are shown in Fig. 3-11 to 3-16. Regardless of the allocation technique used, all criteria investigated in the experiment ($E_2 \div E_4$) for the smallest considered system (P_4: 50 × 50) are better for torus topology. This advantage of torus over the mesh grows together with the size of the requested jobs (the difference between torus and mesh is most visible for P_2: 1 ÷ 20). For the largest considered system (P_4: 100 × 100), the advantage of *k*-ary 2-cube topology is evident for large jobs, however for smaller ones (P_2: 1 ÷ 15, P_2: 1 ÷ 10), the advantage of torus is not so

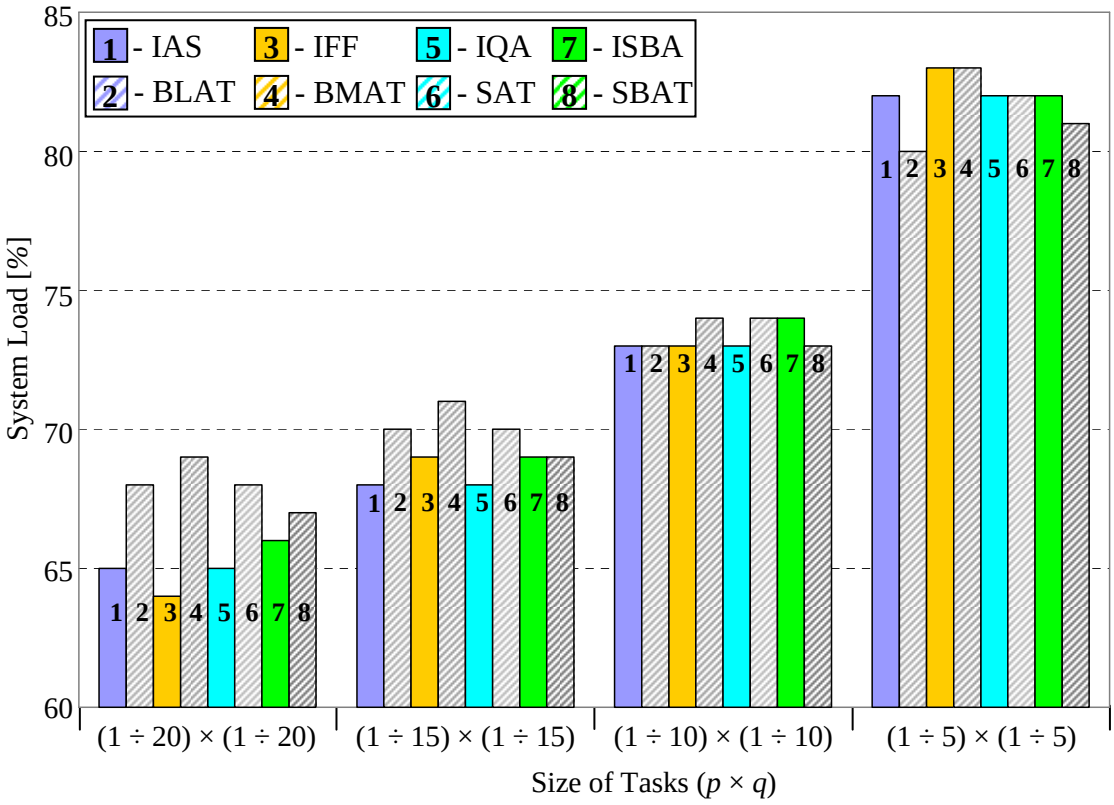


Figure 3-11. Median of system load for torus/mesh 50 × 50.

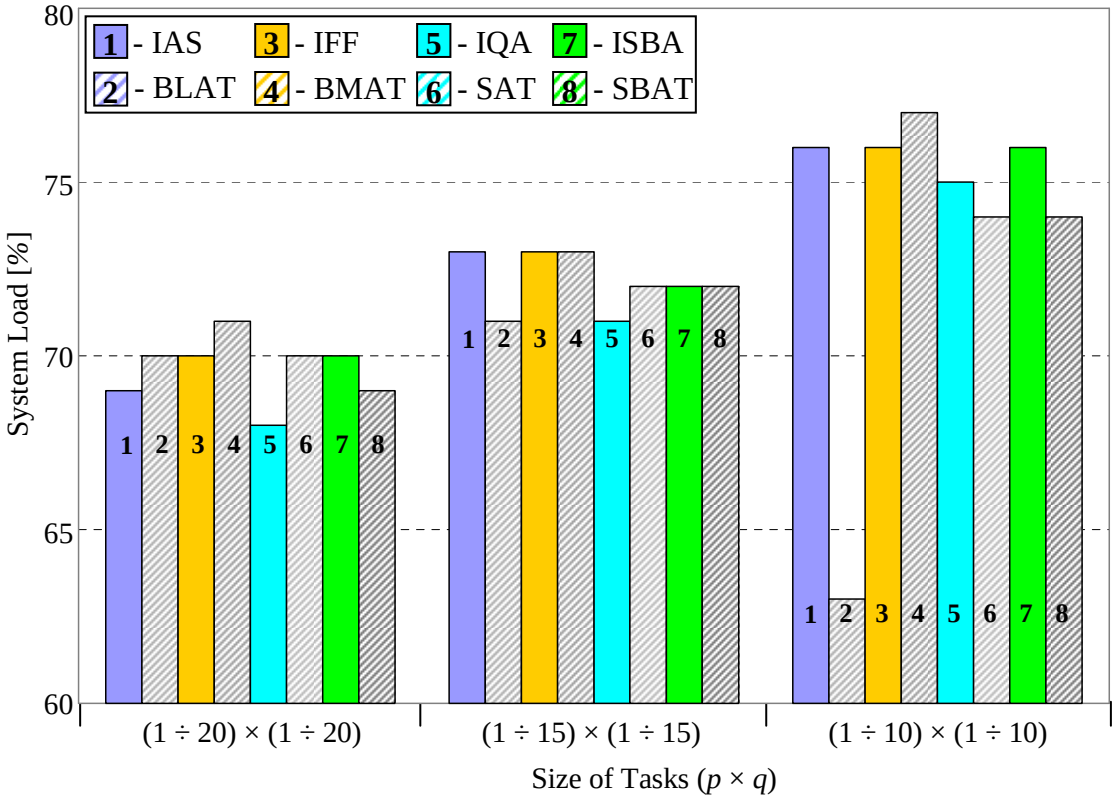


Figure 3-12. Median of system load for torus/mesh 100 × 100.

obvious. System load (Fig. 3-11 and 3-12) for all considered P_4 is higher for torus topology, when allocations of larger jobs (P_2: 1 ÷ 20, 1 ÷ 15) are requested. But in some instances for smaller jobs (P_2: 1 ÷ 10, 1 ÷ 5), the system load for mesh topology is higher. External fragmentation (Fig. 3-13 and 3-14) for P_4: 50 × 50 is lower in torus case, but for P_4: 75 × 75, 100 × 100 several times the mesh behaves better. Simulation time (Fig. 3-15 and 3-16) also is better for toruses with P_4: 50 × 50, but few results for P_4: 75 × 75, 100 × 100 reveal that allocation and execution of jobs on the torus take longer time than on the mesh.

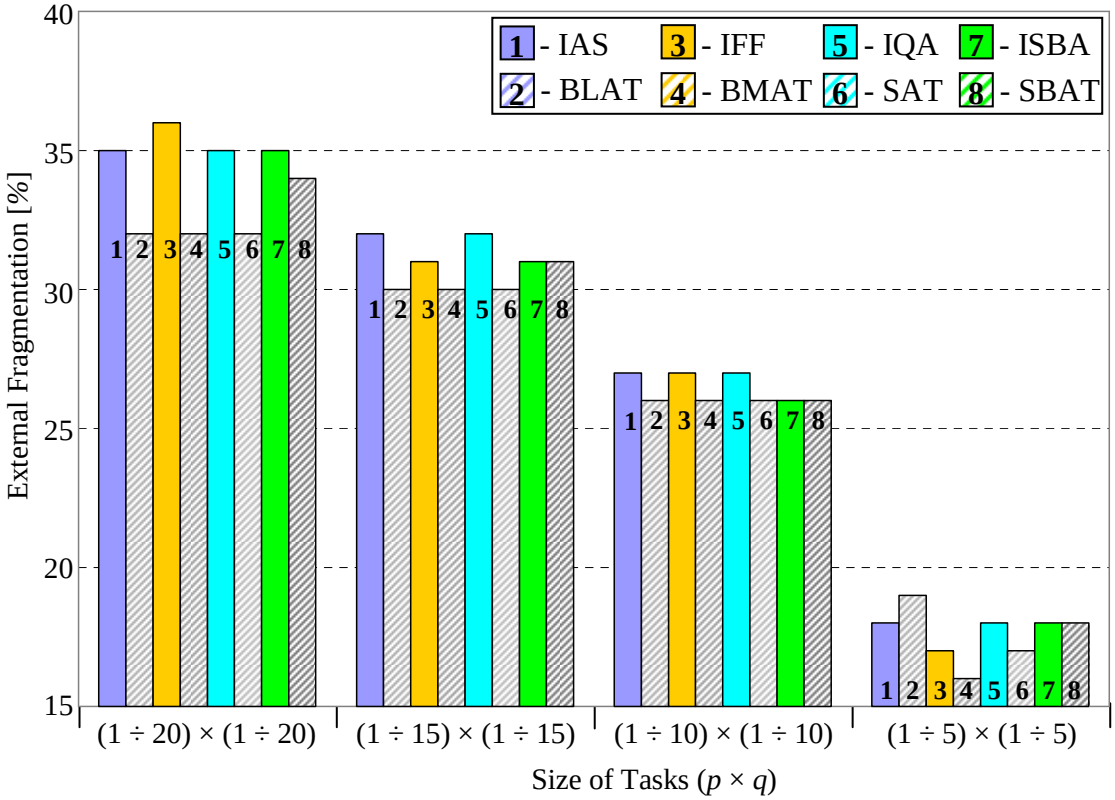


Figure 3-13. Median of external fragmentation for torus/mesh 50 × 50.

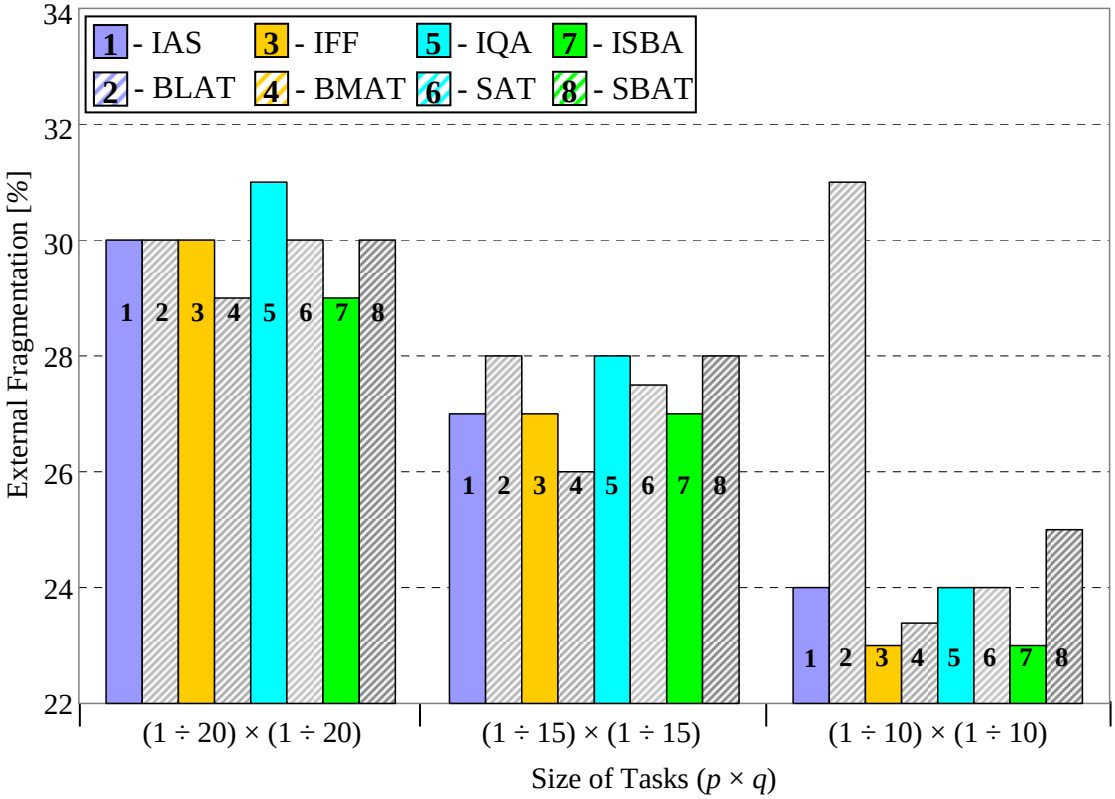


Figure 3-14. Median of external fragmentation for torus/mesh 100 × 100.

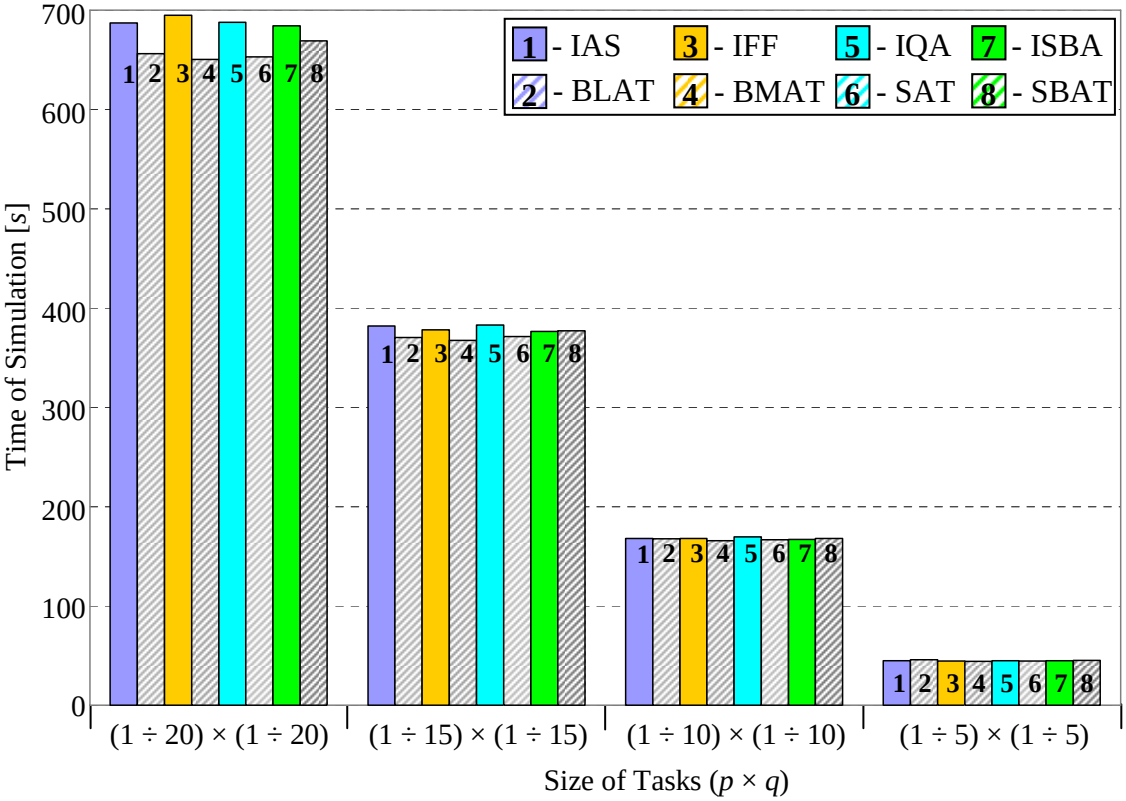


Figure 3-15. Median of time of simulation for torus/mesh 50 × 50.

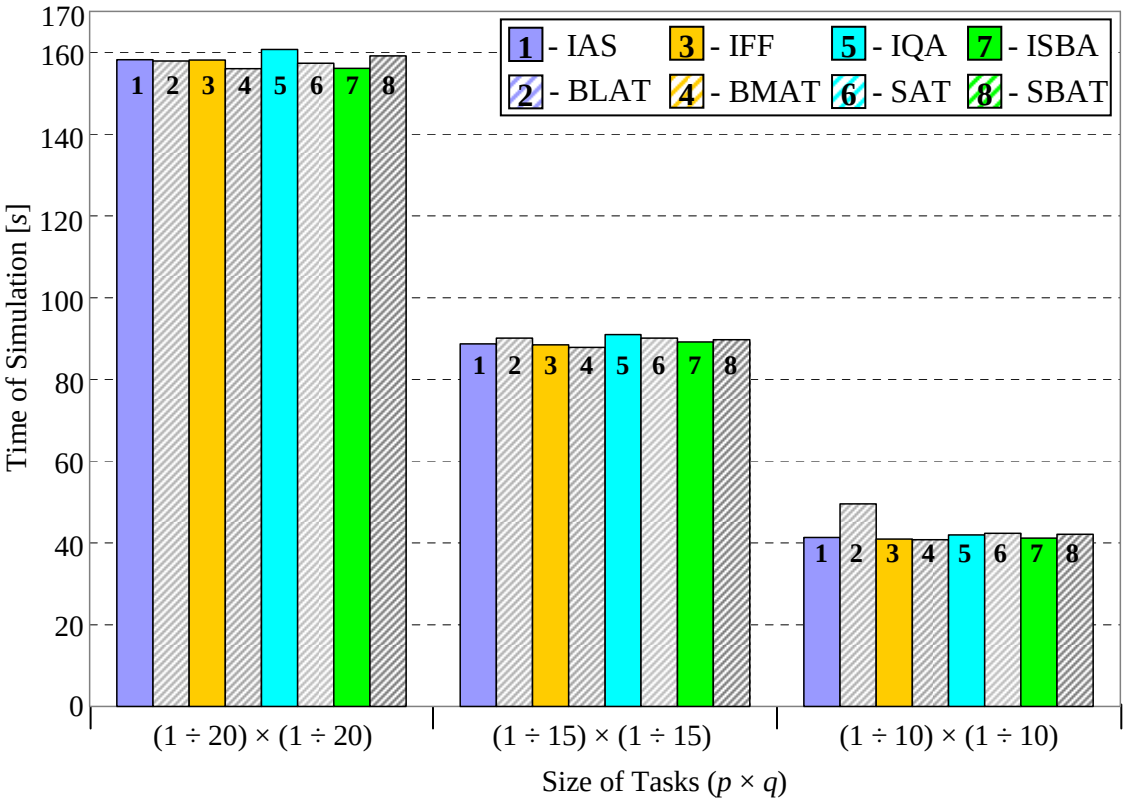


Figure 3-16. Median of time of simulation for torus/mesh 100 × 100.

As we could expect, regardless of the used algorithm, torus topology achieves better indices for very busy systems, where larger jobs are requested (larger proportionally to the size of the system). It is due to higher flexibility of processor allocation offered by *k*-ary 2-cube topology.

From all the proposed processor allocation techniques, the BMAT algorithm outperforms the others. It achieves the best results in almost all our investigations (Fig. 3-11 to 3-16). Only for P_4: 100 × 100 and P_2: 1÷10 output E_4 (external fragmentation) is slightly worse for the BMAT – the IFF and the ISBA achieve better results (Fig. 3-14). For mesh network, the IFF and the ISBA deserve our attention.

3.5.3. Conclusions

The investigations of the allocation time achieved by algorithms reveal that additional operations necessary for torus topology do not have any impact on the allocation speed.

The fastest algorithm for 2D-torus is the BMAT, for 2D-mesh it is the IFF.

Examination of CMP performance does not provide clear answer to what topology ensures better results, regardless of used allocation technique. 2D-torus topology ensures better utilization for systems, where allocation of larger jobs is requested. For smaller jobs, 2D-mesh topology can provide the same efficiency like 2D-torus, moreover, in some cases 2D-mesh can be even better than 2D-torus.

Among all the investigated algorithms, the fast BMAT offers the best results for all the considered parameters so, torus-based CMP with the PA driven by the BMAT is characterized by high system load and low external fragmentation. Allocation and execution of all jobs took the BMAT technique the lowest time. For mesh-based CMP the IFF algorithm, which is also one of the fastest, offers the best system utilization. The

outcomes achieved by BMAT and IFF techniques place the algorithms based on bit map on the top of all allocation schemes.

The presented results of allocation performance show that torus and BMAT algorithm as clear choice for topology and processor allocation scheme for a PA, in order to achieve good PEs utilization and performance. If implementation of a system is limited to mesh topology, the IFF scheme also remains as a reasonable solution.

CHAPTER 4

CMP WITH INTEGRATED PROCESSOR ALLOCATOR

The growing requirements for higher-speed systems force the system designers to propose hardware implementation of systems or their components. Technology scaling causes that the size and the speed of FPGAs (Field-Programmable Gate Array) have been drastically improving. Even small FPGA systems show much better performance than microprocessors in many application areas. Compared to software-based implementations, hardware implementations in FPGA can achieve excellent performance. Hardware implementation of many advanced algorithms has been proposed in literature, e.g. Fast Fourier Transform [92], JPEG compression [107], MPEG-4 compression [74], etc.

As it is described in Chapter 1 of this dissertation, the processor management system of an Operating System (OS) is responsible for supporting a multiuser environment in which many parallel jobs are executed simultaneously. In order to get high performance and efficiency, two components of the processor management system, i.e. the PA and the JS, are proposed to be implemented in hardware and placed as tile on the same die as PEs in CMP (Fig. 4-1). Job scheduler receives request and places job in queue, which is controlled in a particular way (e.g. FCFS fashion). The scheduled job is moved to a processor allocator, which assigns the job to available processors according to one of the algorithms described in Chapter 3. As decided by the processor allocator, processors are reserved and operands are sent to processors to process them. After execution, processors return the results and a “release” message is sent to the allocator, which updates the status of processors. Operands and results are sent through I/O port, which for simplicity are not

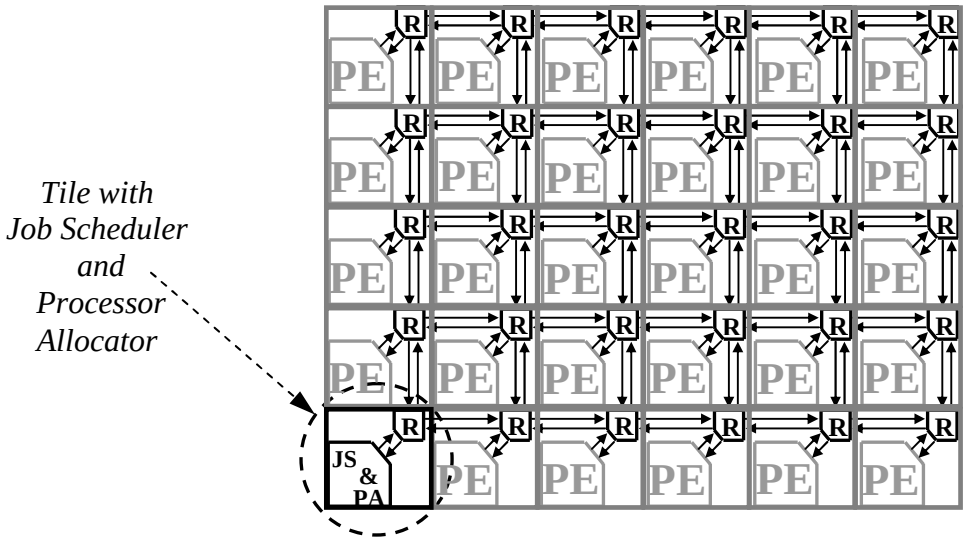


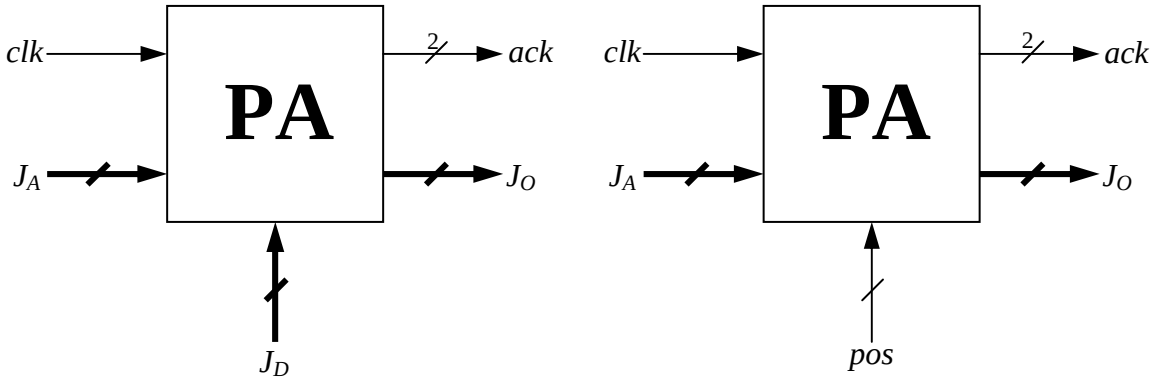
Figure 4-1. A CMP with integrated a JS and PA.

shown in the Fig. 4-1. All data (control messages, operands and results) are sent by the implemented NoC. The allocator and the scheduler are implemented using the same networking elements (R) like PEs and therefore can use the same for communication.

4.1. Architecture of PA

A NoC is designed in such a way so as to provide low latency and high data throughput. Such a well performing NoC consumes chip space, resources, energy, power and heat. Thus even with the positive prospect of transistors scaling in CMP design process, we have to deal with on-chip resource utilization. The idea of integrating PA onto CMP makes it imperative to design the architecture carefully and optimally.

Internal architecture of a PA may vary with the implemented algorithm that leads also to different I/O structures. The I/O organization for designs with busy array is presented in the Fig. 4-2a while for busy list in the Fig. 4-2b. In both cases, signals *clk* and *ack* are the same. The *clk* is the clock signal (the JS is triggered by the same signal), the *ack* is 2-



(a) Busy array (b) Busy list

Figure 4-2. PA structure for list and busy array implementations.

bit output signal that is an acknowledgement informing the JS about the status of given job. The *ack* can have two meanings:

1. Job is allocated, values “00” or “11”,
2. Job is not allocated, value “01” or “10”.

The width is set at 2 bits allowing possible future extensions. Input and output signals *J* have the same width and structure, but they differ depending on the technique used.

In the case of bit map, to allocate a job we need to have its width and height and for deallocation we additionally need the coordinates of the base. Each job also needs to have its ID in order to be recognized by OS:

$$J \rightarrow \text{“<job\_number><}x_b\text{><}y_b\text{><}p\text{><}q\text{>”}$$

Width of <*job_number*> depends on OS while <x_b>, <y_b>, <*p*> and <*q*> depend on size of mesh/torus *w* and *h*. For requested job, OS sends on the bus J_A signal, where <*job_number*> is next job number, <x_b> and <y_b> are empty (have value -1), <*p*> and

<*q*> are size of the job. If the job is not allocated, “01” signal is sent on the *ack* and signal J_O is empty (or ignored). In case of allocation, *ack* becomes “00” and J_O signal is sent to OS, where: <*job_number*> is the number from J_A, <x_b> and <y_b> are coordinates of the base found by the PA, <*p*> and <*q*> are size of the job. If a job is executed and deallocation is requested, J_D signal is asserted with values assigned as in the case of J_O.

In list schemes, for the purpose of allocation, we also need the width and height of a job and for deallocation we need only its position on the busy list:

$$J \rightarrow \text{“<}job\_number\text{><}p\text{><}q\text{><}position\text{>”}$$

Signals <*job_number*>, <*p*>, <*q*> are defined like earlier, width <*position*> depends on the length of the busy list i.e., on the number of processors and therefore, it is $w \times h$. For allocation, OS asserts J_A similar to the busy array case, but <*position*> is empty (have value -1). In the case of success (allocation) *ack* becomes “00” and J_O signal is sent to OS, where: <*job_number*> is the number from the J_A, <*p*> and <*q*> are then coordinates of the base found by the PA, <*position*> is the position of the job on the busy list. If deallocation is required, *pos* signal is asserted with the value of job location in the busy list.

4.2. PA’s Memory Structure and Logic Utilization Aspects

4.2.1. Memory Structure on the Chip

The algorithms with busy array (IFF and BMAT) maintain two busy arrays:

1. Mesh/torus bit map – with allocation status for each node,
2. Coverage bit map – created for each incoming job and informing which node can be the base for the task.

Array size depends on the size of mesh/torus, thus it is $w \times h$. Because it is a bit map, the size of each cell is 1 [*bit*], so, finally the size of required memory is: $2 \times w \times h \times 1$ [*bits*].

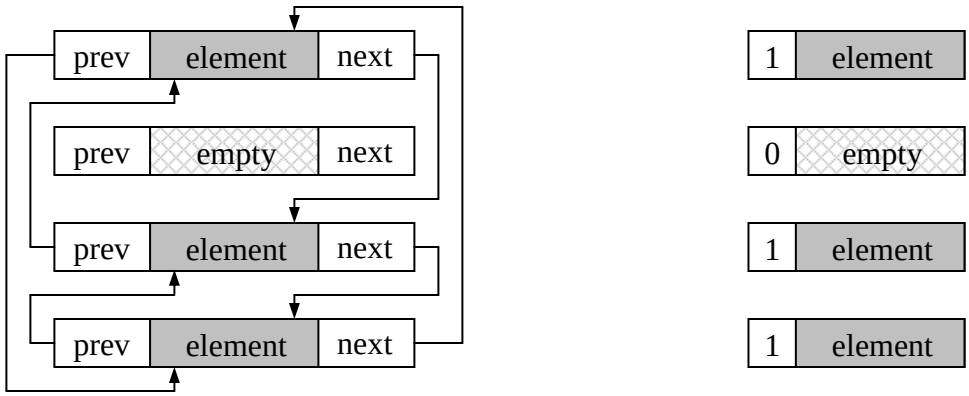
Schemes based on busy lists (IAS, IQA, ISBA, BLAT, SAT and SBAT) need two lists:

1. Busy list – list with coordinates of busy subgrids,
2. Coverage list – created for each requested job that contains the coordinates of coverages.

Assuming that in the worst case each node hosts one separate job implies separate busy subgrid for each node, thus the length of busy and coverage list is $w \times h$.

It is important to note that the list contains coordinates. The size of each coordinate depends on horizontal and vertical size of mesh/torus w and h respectively. To code addresses of nodes using natural binary code, n and m bits are needed for w and h sizes respectively, where: $2^n \geq w$ -1 and $2^m \geq h$ -1. As an example, a mesh with w = 12 and h = 8 is considered. In this case, addresses of nodes between <0, 0> and <11, 7> have to be stored in a memory. Using natural binary code, n = 4 and m = 3 bits are needed to store one horizontal and vertical coordinate respectively.

List structure itself becomes a sensitive factor. In the Fig. 4-3 two list structures are shown. In case (a), beside subgrid coordinates, each element includes pointers to previous and next element. Such a solution ensures very effective memory utilization from data structure point of view. Also scanning of the list is limited only to valid memory entries – cells without busy submesh are not read. Size of each cell in this case consists of:



(a) Standard list (b) Bit list

Figure 4-3. 3-element list structures: standard list and list with validation bit position.

1. $n + m$ bits for both pointers *prev* and *next*,
2. $n + m$ bits for lower left and upper right corner of subgrid.

Size of the whole busy list is $w \times h$ times each entry. As two lists are required for the algorithms, we have: $2 \times w \times h \times (4 \times n + m)$ [*bits*]. Case (b) in Fig. 4-3 presents memory minimized structure, where a validation bit is associated with each element of the list. If element includes busy subgrid, the bit is set to one, while for empty entries, the bit is set to zero. In this case each memory cell has to be scanned, but only one bit is checked in order to get information about validation of the memory cell. Size of each cell contains one validation bit and $n + m$ bits for lower left and upper right corner of subgrid, thus for two busy lists we have: $2 \times w \times h \times (1 + 2 \times n + m)$ [*bits*].

In Fig. 4-4, size of memory as a function of mesh/torus size for the three discussed solutions is plotted. Solution with bit map outperforms the remaining approaches in terms of memory usage. Also as we could expect that the list implemented with validation bit

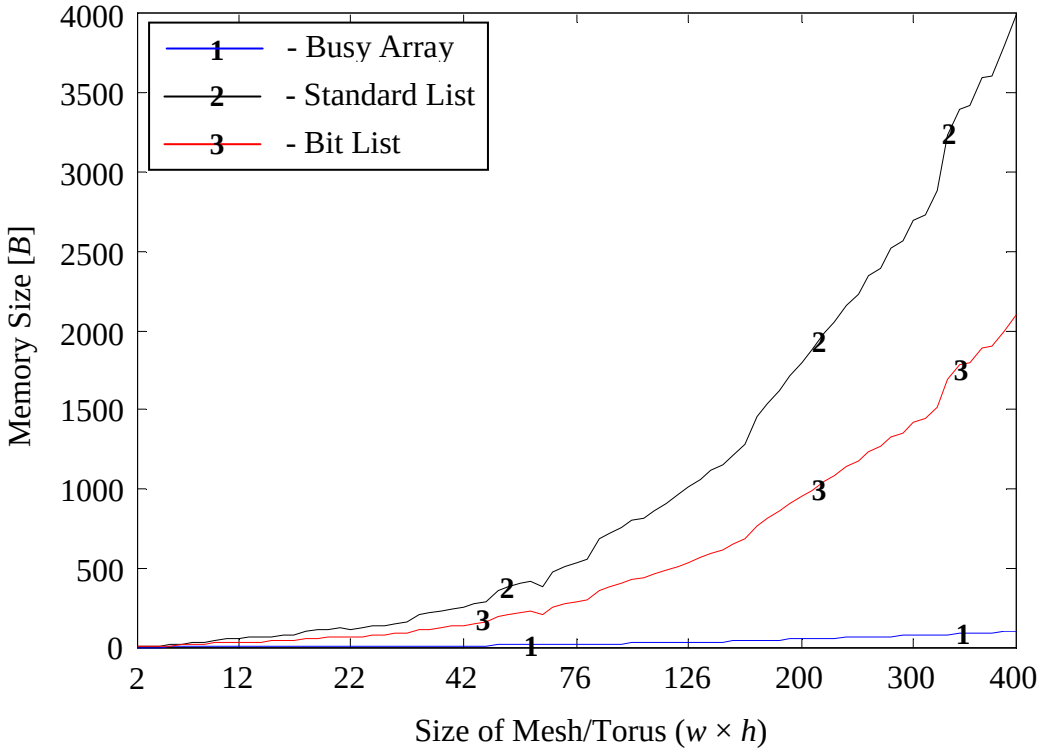


Figure 4-4. Size of memory using in the system based on size of mesh/torus.

takes less space than the standard list. In all the considered cases the difference becomes more significant with increasing size of mesh/torus.

4.2.2. Analysis of Algorithms in Terms of Logic Utilization

The solutions with busy array are based on scanning a bit map, cell by cell. According to the content of a cell (few conditions are used), the technique updates a coverage array, which is also a bit map. As each cell has a bit size, there is not a lot of logic involved in scanning/updating. Also there is almost no arithmetic operation, thus the busy array solutions are not heavy logic consumer. It is necessary to differentiate between solutions for mesh (IFF) and torus (BMAT). The BMAT is more advanced, thus it is expected to involve more logic.

The situation with busy list structure is more complex. Two lists (busy and coverage) are involved, they are scanned, compared and spatial subtractions are performed on the elements of the lists, which is logic consuming. Comparison and condition operations are executed for wider memory cells, which automatically leads to additional logic overheads. Moreover, the solutions for a torus network contain more complex sections, e.g., creating C_J or checking intersections. For IQA and SAT schemes, sorting of elements are also required. So, in this case allocation operation is very difficult. Deallocation is reduced to removing the element from busy list, which is very fast in the case of bit list, but for the standard list we need to reassign pointers of previous and next element, what is not so complex.

Preliminary analysis indicates a cause of concern about possible synthesis problems in list based techniques, especially for torus topologies.

4.3. Synthesis of PA

4.3.1. Overview

The PA structure along with all the presented algorithms including the IFF, IAS, IQA, ISBA, BMAT, BLAT, SAT and SBAT have been implemented in VHDL and simulated using Active-HDL tool by Aldec Inc. After performing logic level simulation for correctness verification, the PA was synthesized using Altera's Quartus II software.

Synthesis of the PA was done for the device EP3SL150F780C2 from Stratix III family [3], [4]. The Logic Array Block of the EP3SL150 is composed of 56800 basic building blocks known as Adaptive Logic Modules (ALMs) that consist of combinational logic, two registers, and two adders. The combinational portion has eight inputs and includes a Look-Up Table (LUT) that can be divided between two Adaptive LUTs

(ALUTs). The EP3SL150F780C2 has 113600 ALUTs and registers, and 488 pins. An entire ALM is needed to implement an arbitrary 6-input function, but because it has eight inputs to the combinational logic block, one ALM can implement various combinations of two functions. Internal clock speed of EP3SL150 is up to 600 MHz.

IFF and BMAT algorithms are synthesizable smoothly and synthesis is possible even for large instances, especially for the IFF technique, where the synthesis result shows that there is still a lot of spare capacity in the device (EP3SL150). For the other algorithms that are busy-list based, only the IAS and BLAT schemes were synthesizable. Moreover, synthesis of these techniques was possible only for the bit list approach, while for the standard list synthesis of the algorithms was not achievable.

For the IQA, ISBA, SAT and SBAT algorithms, synthesis for both, the standard and bit list ended with an error. In the case of IQA and SAT, the bottleneck is in sorting operation. In order to preserve $O(hB)$ time complexity, sorting algorithms need to have $O(B\ log_2 B)$. Due to very difficult synthesis of recursive functions [45], instead of using the merge sort algorithm, the heap sort technique [52] that is not so recursive was tested, but even in this case synthesis was not possible. The ISBA and SBAT algorithms have implemented two lists and stack. All these structures are used intensively by many scans, subtractions and updates, which in conjunction with the necessity of using long signals among these structures, make the technique not synthesizable.

4.3.2. Synthesis Results

The synthesis of the IFF, IAS, BMAT and BLAT algorithms has been successfully performed. The synthesizable instances for IAS, BMAT and BLAT are limited to size 8 × 8 (64 PEs), 16 × 10 (160 PEs) and 8 × 7 (56 PEs), respectively. For larger systems, the

PA driven by these techniques are not synthesizable. The IFF is fully synthesizable. The instances considered in this work are between 2 × 2 (4 PEs) and 20 × 20 (400 PEs). However, the IFF is very well scalable and additional experiments for greater meshes can be performed.

In Fig. 4-5 and 4-6, the maximum frequency for the PA based on IFF, IAS, BMAT and BLAT designs is shown. Detailed results are listed in Table 4-1 as well. The plots display results at conditions, where voltage is 1100 [mV] and temperatures are 85 [$^{\circ}C$] and 0 [$^{\circ}C$] in Fig. 4-5 and 4-6, respectively. As we can see, for smaller instances, the maximum frequency f_{max} for the PA with the IFF algorithm is more than four times higher than for PA with other schemes. IFF is two times faster for the size 8 × 8. For grids greater than 8 × 8, f_{max} of the IFF decreases to reach the last synthesized instance

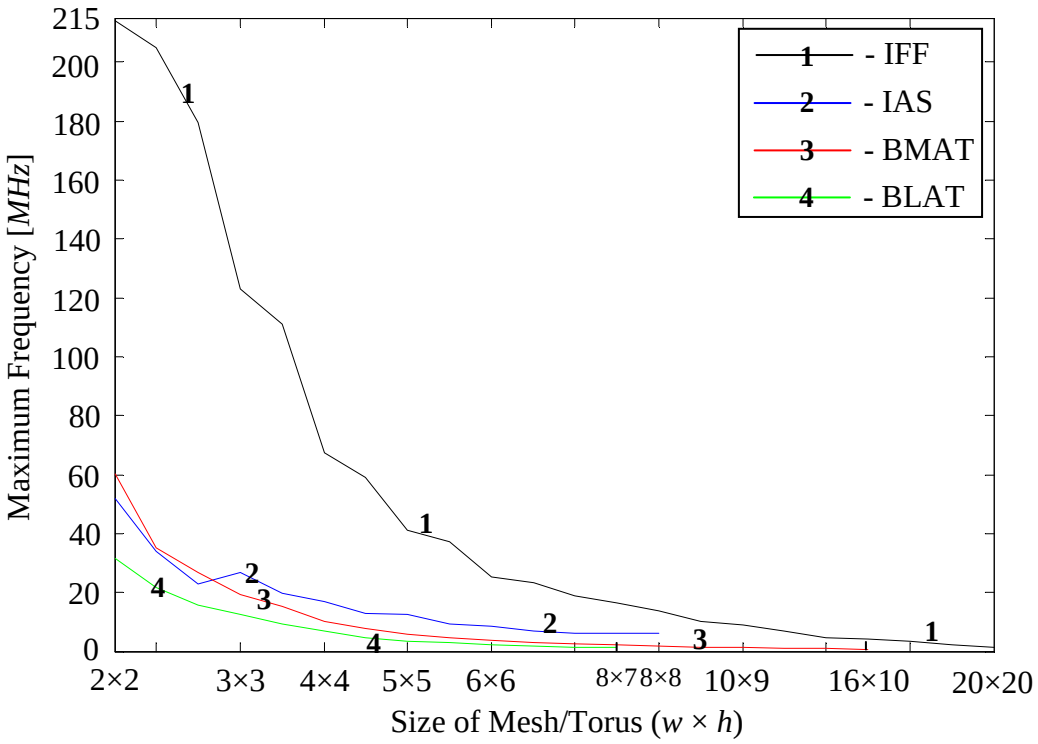


Figure 4-5. Maximum frequency f_{max} at voltage 1100 [mV] and temperature 85 [$^{\circ}C$].

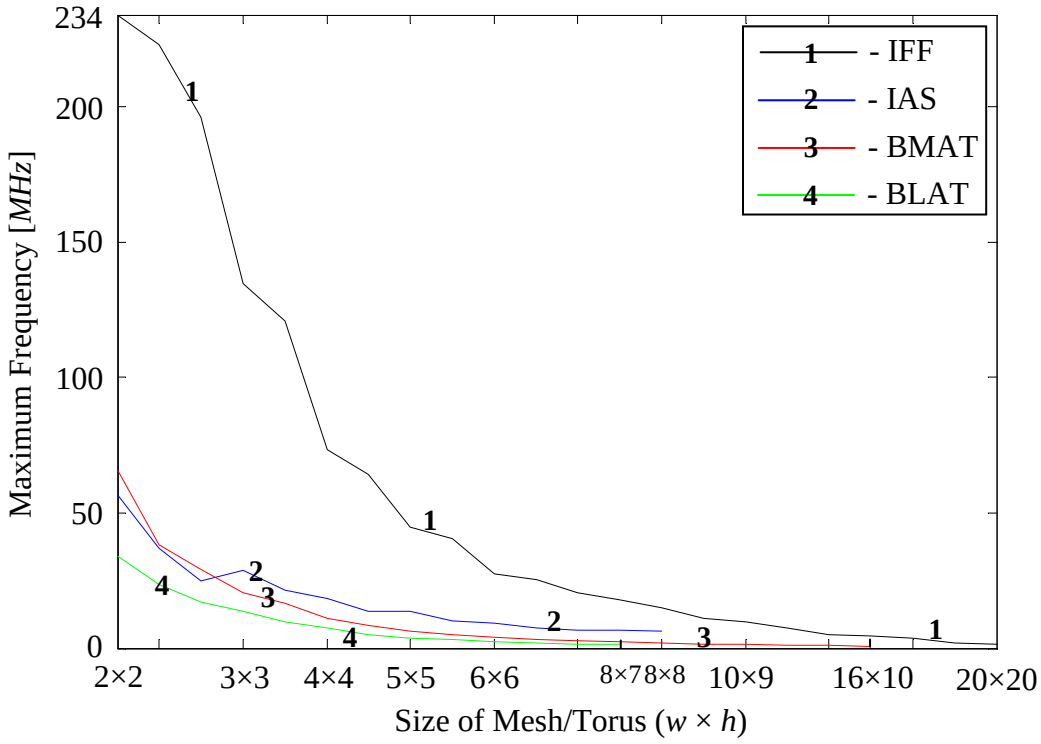


Figure 4-6. Maximum frequency f_{max} at voltage 1100 [mV] and temperature 0 [$°C$].

(20 × 20) 1.55 MHz, while other techniques are not implementable. Also, for all considered cases, the advantage of the schemes for 2D-mesh topology can be noticed – achievable frequencies are higher. As far as algorithms are concerned, the algorithms with busy array achieved better outcomes. The PA driven by BMAT technique for 2D-torus achieved almost the same results like the IAS for 2D-mesh.

Figures 4-7, 4-8 and 4-9 show percentage of logic gate utilization on the chip, number of combinational ALUTs and number of dedicated registers used, respectively. These parameters (also in Table 4-2) have tremendous impact on the design – they have direct impact on energy, power and area consumption by the PA, and also the whole CMP. From the plots we can see huge advantage of the IFF scheme over the rest. Similarly, the

Table 4-1. Maximum frequencies of a PA in the EP3SL150F780C2 device.

Scheme	**IFF**		**IAS**		**BMAT**		**BLAT**	
Temperature	85 [°C]	0 [°C]	85 [°C]	0 [°C]	85 [°C]	0 [°C]	85 [°C]	0 [°C]
Size of Grid:	f_{max} [MHz]	f_{max} [MHz]	f_{max} [MHz]	f_{max} [MHz]	f_{max} [MHz]	f_{max} [MHz]	f_{max} [MHz]	f_{max} [MHz]
2×2	214.09	233.81	51.89	56.24	60.09	65.56	31.43	33.99
3×2	204.96	223.11	34.02	36.78	35.05	38.16	21.83	23.69
4×2	179.5	196.08	22.79	24.68	27.01	29.4	15.83	17.15
3×3	122.87	134.59	26.78	28.97	19.09	20.72	12.5	13.55
4×3	110.96	120.83	19.75	21.3	15.49	16.83	9.2	9.94
4×4	67.42	73.52	16.96	18.33	9.99	10.85	6.8	7.36
5×4	59.05	64.25	12.76	13.83	7.93	8.6	4.72	5.1
5×5	41.01	44.71	12.62	13.63	5.77	6.25	3.31	3.57
6×5	37.04	40.26	9.48	10.23	4.68	5.07	3.1	3.36
6×6	25.35	27.6	8.66	9.34	3.67	3.97	2.29	2.48
7×6	23.12	25.15	7.07	7.65	3.14	3.4	1.74	1.88
8×6	18.75	20.39	6.33	6.86	2.71	2.93	1.48	1.6
8×7	16.39	17.81	6.1	6.5	2.18	2.35	1.3	1.4
8×8	13.65	14.84	5.97	6.44	1.88	2.04	–	–
10×8	10.32	11.2	–	–	1.45	1.57	–	–
10×9	8.96	9.73	–	–	1.39	1.5	–	–
10×10	6.79	7.38	–	–	1.1	1.19	–	–
15×10	4.57	4.96	–	–	0.81	0.88	–	–
16×10	4.32	4.69	–	–	0.75	0.81	–	–
20×10	3.4	3.69	–	–	–	–	–	–
20×15	1.99	2.16	–	–	–	–	–	–
20×20	1.43	1.55	–	–	–	–	–	–

advantage of busy array techniques over the busy list algorithms can be noticed as well. For 8×7 grid, logic utilization for IAS and BLAT is 71% and 96%, respectively, while for the same instance the IFF and BMAT use only 2% and 19%, respectively. For combinational ALUTs, advantage of the IFF can be observed as well: for the 8×7 grid, IAS, BLAT and BMAT require 57174, 83516 and 18934 ALUTs, respectively, while IFF uses 1610. In the case of dedicated registers, for the 8×7 instance, IAS and BLAT need 3570 and 3602 registers, respectively, whilst IFF and BMAT employ only 135 registers each.

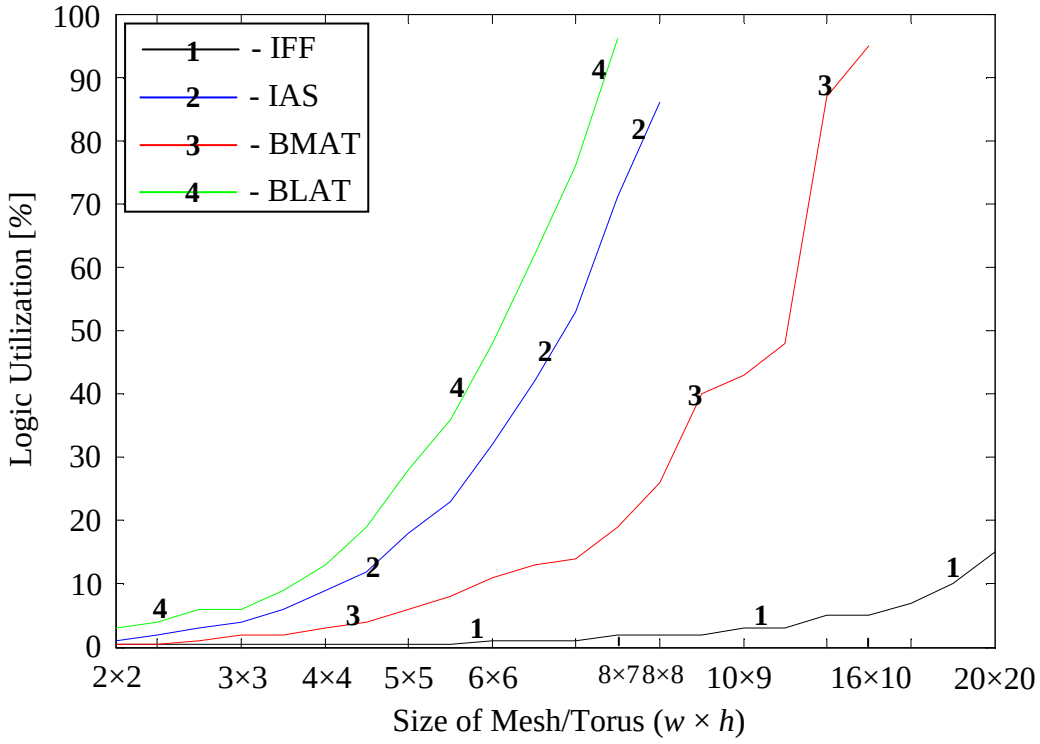


Figure 4-7. Logic utilization in the EP3SL150F780C2 device.

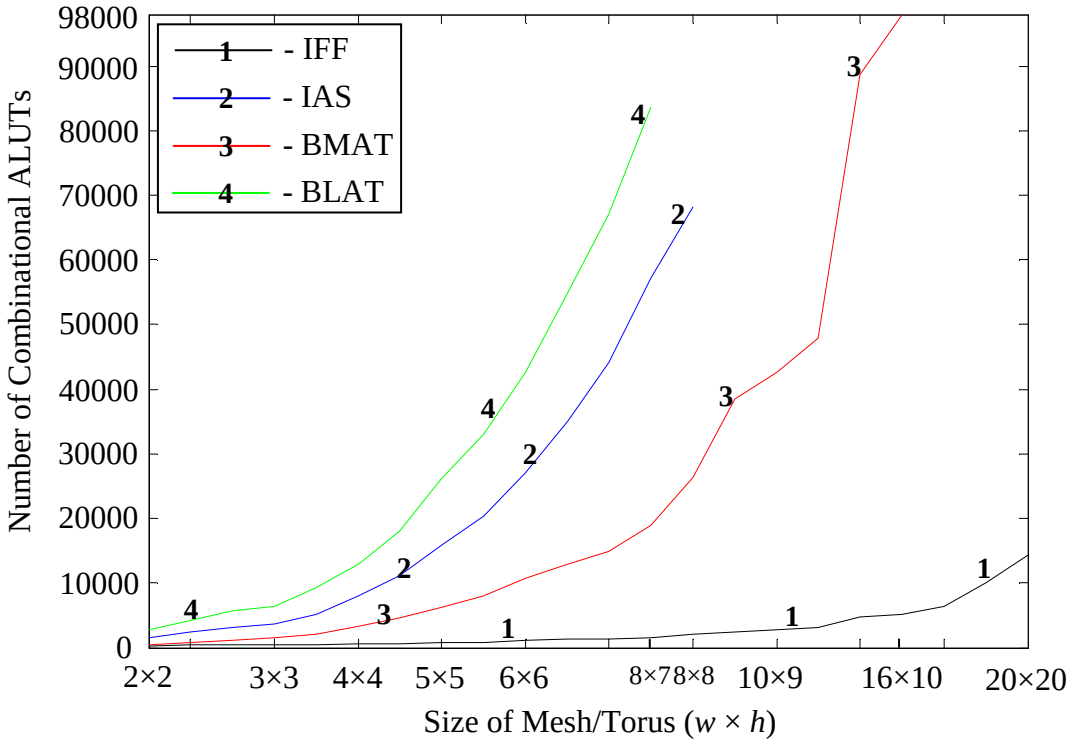


Figure 4-8. Number of combinational ALUTs used.

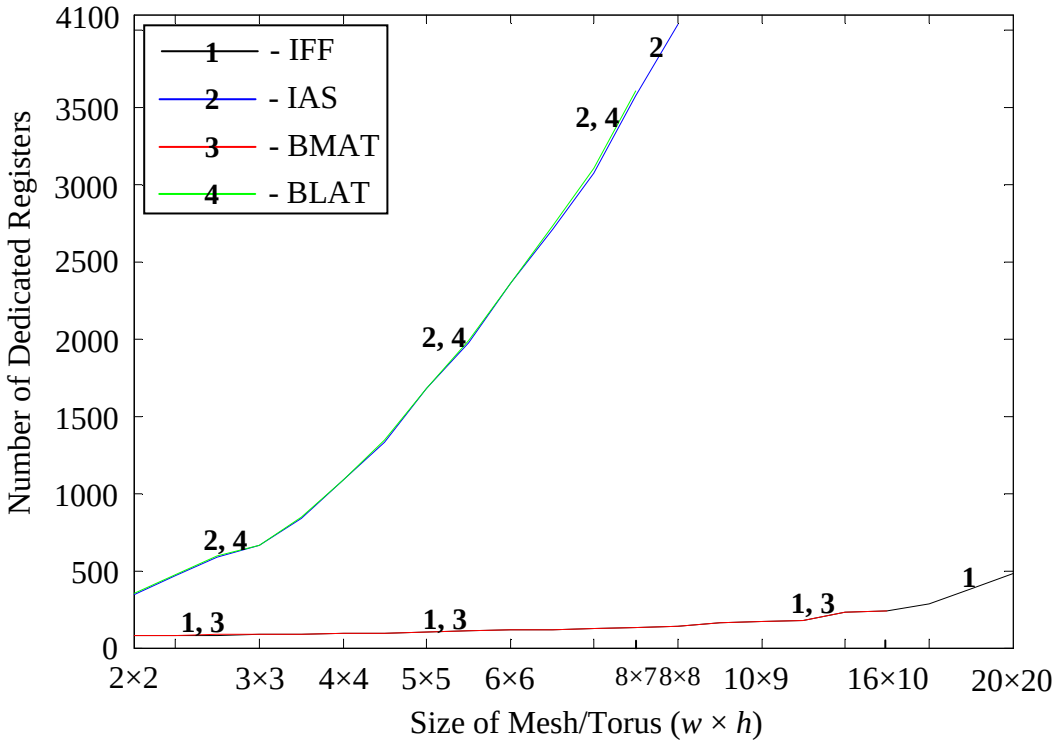


Figure 4-9. Number of dedicated registers used.

4.3.3. Energy Estimation

Since Stratix III is the target device, we need to deal with Stratix-specific features in power estimation. Altera provides a PowerPlay Early Estimator [2] that is based on a spreadsheet. The tool allows users to specify switching activities, f_{max}, usages of various components and other related information to estimate the total power of Altera's FPGA in early design stages. Because, as in the Chapter 2.4, we would like to get energy estimation for a PA, the average power dissipation P from spreadsheet is converted into energy consumed in a cycle E_c [16], according to the formula:

$$E_c = P \frac{1}{F_{\max}} [\mu J] \qquad (4\text{-}1)$$

, where F_{max} is the average maximum frequency of f_{max} at 0 [$°C$] and 85 [$°C$] in [MHz].

Table 4-2. Results of the PA synthesis for the EP3SL150F780C2 device.

Scheme	**IFF**			**IAS**			**BMAT**			**BLAT**		
Pins	163			125			163			125		
Size of Grid:	*Logic Util.[%]*	*ALUT*	*Dedic. Reg.*	*Logic Util.[%]*	*ALUT*	*Dedic. Reg.*	*Logic Util.[%]*	*ALUT*	*Dedic. Reg.*	*Logic Util.[%]*	*ALUT*	*Dedic. Reg.*
2×2	0.5	339	78	1	1490	347	0.5	459	79	3	2756	351
3×2	0.5	383	81	2	2432	465	0.5	882	82	4	4188	475
4×2	0.5	391	83	3	3166	585	1	1177	84	6	5644	594
3×3	0.5	443	85	4	3788	664	2	1585	86	6	6466	664
4×3	0.5	475	88	6	5236	835	2	2126	89	9	9259	844
4×4	0.5	615	93	9	8094	1085	3	3331	93	13	12905	1085
5×4	0.5	682	98	12	11105	1333	4	4544	98	19	18072	1347
5×5	0.5	818	103	18	15898	1679	6	6235	104	28	26135	1679
6×5	0.5	874	108	23	20359	1973	8	8074	109	36	32998	1989
6×6	1	1151	115	32	26993	2362	11	10778	115	48	42735	2362
7×6	1	1288	121	42	35137	2707	13	12969	121	62	55018	2734
8×6	1	1366	127	53	44025	3072	14	14881	127	76	67112	3106
8×7	2	1610	135	71	57174	3570	19	18934	135	96	83516	3602
8×8	2	1994	143	86	68292	4034	26	26281	143	–	–	–
10×8	2	2383	160	–	–	–	40	38502	160	–	–	–
10×9	3	2726	170	–	–	–	43	42592	171	–	–	–
10×10	3	3169	181	–	–	–	48	47837	181	–	–	–
15×10	5	4858	231	–	–	–	87	88710	231	–	–	–
16×10	5	5176	241	–	–	–	95	97906	241	–	–	–
20×10	7	6376	282	–	–	–	–	–	–	–	–	–
20×15	10	10017	382	–	–	–	–	–	–	–	–	–
20×20	15	14396	483	–	–	–	–	–	–	–	–	–

The final values are presented in Table 4-3. Similarly like earlier outcomes, the results show the significant advantage of busy array techniques, especially mesh-based IFF technique. The busy list schemes use between two and five times more energy in comparison to bit map solutions, that makes PAs driven by the IAS and BLAT not efficient. The amount of logic and dedicated registers used by these algorithms has huge impact on the maximum frequency and energy consumption.

Table 4-3. Power P and energy consumed in a cycle E_c for proposed the PAs.

Scheme	**IFF**		**IAS**		**BMAT**		**BLAT**	
Size of Grid:	*P [W]*	*E_c [nJ]*	*P [W]*	*E_c [J]*	*P [W]*	*E_c [J]*	*P [W]*	*E_c [J]*
2×2	0.6116136	2.7310275	0.6109082	11.299514	0.6116136	9.7351947	0.6114341	18,692574
3×2	0.6114821	2.8569259	0.6113026	17.268436	0.6118108	16.713859	0.6120258	26,890412
4×2	0.6116136	3.2569019	0.6115656	25.766404	0.6119423	21.696234	0.6126175	37,150849
3×3	0.6116136	4.751135	0.6118285	21.949006	0.6120738	30.74975	0.6129462	47,059212
4×3	0.6116136	5.277308	0.6124202	29.837771	0.6123367	37.892125	0.6140641	64,165524
4×4	0.6116793	8.6799964	0.6136038	34.774937	0.6127969	58.809687	0.6155766	86,945855
5×4	0.6117451	9.9228723	0.6148532	46.246951	0.6133229	74.207249	0.6176815	125,80072
5×5	0.6118108	14.274634	0.6168264	46.996295	0.6139804	102.1598	0.6210373	180,53409
6×5	0.6118108	15.829517	0.6186684	62.777108	0.6147695	126.10657	0.6238676	193,14787
6×6	0.6119423	23.113968	0.6213663	69.040705	0.6158875	161.22709	0.6278843	263,26387
7×6	0.612008	25.357698	0.6247235	84.88091	0.6168083	188.62639	0.6328913	349,66373
8×6	0.612008	31.272766	0.6284113	95.286013	0.6175976	219.00625	0.6379012	414,22157
8×7	0.6121395	35.797632	0.633814	100.6054	0.6192423	273.39617	0.6446295	477,50335
8×8	0.612271	42.981467	0.6383628	102.87878	0.6222693	317.48436	–	–
10×8	0.6124025	56.914728	–	–	0.6272728	415.41245	–	–
10×9	0.6125997	65.55374	–	–	0.6289852	435.28386	–	–
10×10	0.6127312	86.482878	–	–	0.6310932	551.17307	–	–
15×10	0.6134544	128.74174	–	–	0.6479754	766.83481	–	–
16×10	0.6135859	136.20109	–	–	0.6517388	835.56256	–	–
20×10	0.6140462	173.21472	–	–	–	–	–	–
20×15	0.6155587	296.65478	–	–	–	–	–	–
20×20	0.6173345	414.31847	–	–	–	–	–	–

4.4. Conclusions

The PA synthesis results confirm flexibility and advantages of bit map based solutions, while list based schemes turn out to be hardly synthesizable. From among all the considered list solutions, we have implemented only the IAS and BLAT, but it is only for limited instances – the largest synthesized mesh for the IAS was 8 × 8, for the BLAT the largest tours was 8 × 7. The IFF algorithm as engine of PA outperforms the IAS, BLAT and BMAT in every investigated aspect i.e. maximum frequency, logic utilization and energy consumption. However, the difference between IFF and BMAT is not so significant like in the other cases, which confirms the advantage of busy array solutions.

IAS and BLAT, as techniques for PA are slow and consume huge amount of energy and area.

The presented outcomes show, that for CMPs with integrated PA, only the bit map techniques, i.e. IFF and BMAT, give reasonable solutions. The other solutions characterize unfeasible synthesis (all list solutions without IAS and BLAT) or poor performance and huge resource exploitation (the IAS and BLAT). The mesh-based PA driven by the IFF scheme performs well and as a small component of the CMP is area and energy efficient. For torus-based solutions, only BMAT remains as an acceptable technique.

CHAPTER 5

CMP – EXPERIMENTATION SYSTEM

The goal of creating an experimentation system for CMP is to examine the NoC-based CMP with an integrated PA. The structure of the system allows testing of the PA driven by all processor allocation techniques described in this work (Chapter 3). In the testing environment, the PA can be connected with other nodes by the NoC, which was analyzed in the Chapter 2. The testing environment provides the possibility to test many NoC configurations. The experimentation system is characterized by its modular design that allows complex input-output tests for subsystems. The concept of the system, its design and implementation is done in such a way to make the experimental environment as close as possible to the real CMP.

5.1. Structure of the System

A physical structure of the experimentation system is based on the concept presented in Fig. 4-1. The logical modules of the system and information flow are presented in Fig. 5-1. We can distinguish the following modules:

- **Parameters** – contains global parameters, common for three modules: Data Generator, PA and NoC-based CMP. The global parameters are:
  - P_{G1}: denoted by w – horizontal size of the mesh/torus,
  - P_{G2}: denoted by h – vertical size of the mesh/torus,
  - P_{G3}: defines, which topology is evaluated. Two topologies are supported: 2D-mesh and 2D-torus.

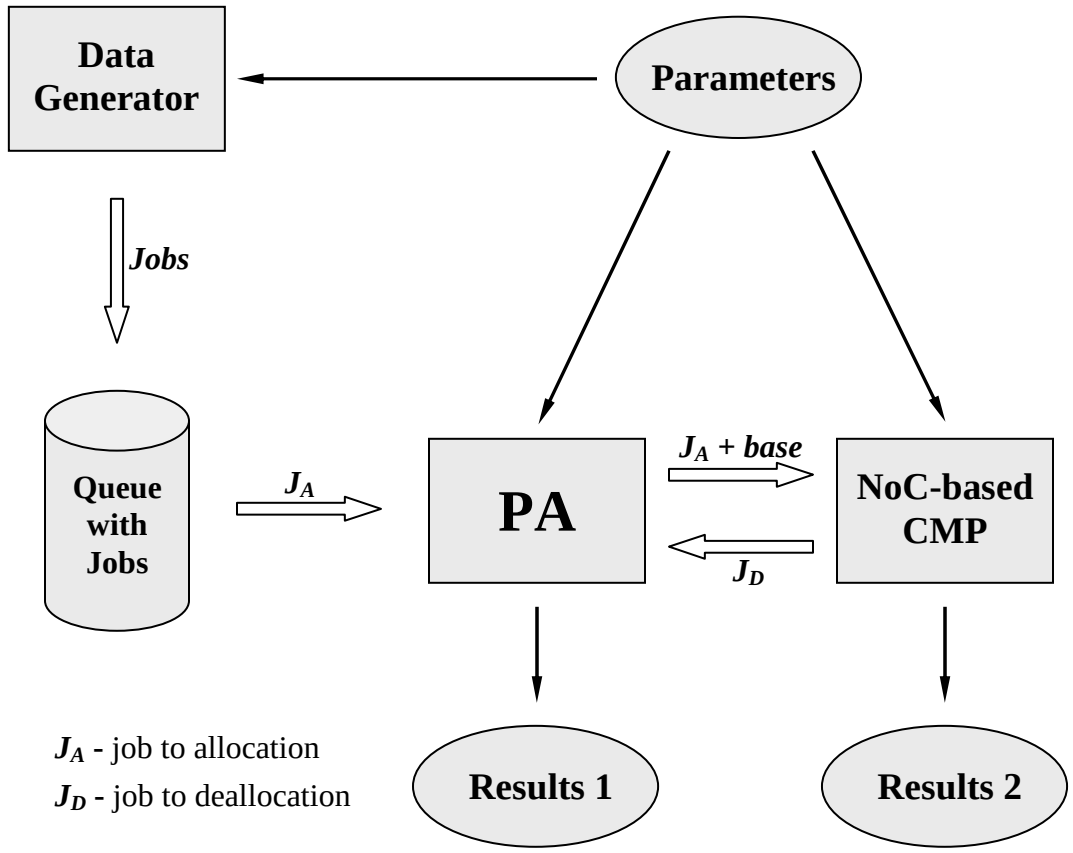


Figure 5-1. The logical organization of the experimentation system.

- **Results 1** – the allocation algorithms quality criteria are collected in the module Results 1. The evaluated criteria are presented in Chapter 3.5.1, and they are:
  - E_1**:** denoted by t_a – time of allocation,
  - E_2**:** denoted by t_s – simulation time,
  - E_3**:** denoted by L – system load,
  - E_4**:** denoted by F_e – external fragmentation.

- **Results 2** – in this module, NoC utilization is recorded. The assumed NoC utilization criteria are:

- E_5: denoted by $R_{VC<x, y>}$ – VC count, defined for virtual channels for each router in a network, where <*x*, *y*> is a network address of the router. The $R_{VC<x, y>}$ contains information, how many packets are transmitted through VCs in the router <*x*, *y*>, e.g. if twenty packets are transmitted through router with address <3, 3> using its VCs, the $R_{VC<3, 3>}$ is equal 20,
- E_6: denoted by $R_{EVC<x, y>}$ – EVC count, defined for express virtual channels for each router in a network, where <*x*, *y*> is a network address of the router. The $R_{EVC<x, y>}$ contains information, how many packets are transmitted through EVCs in the router <*x*, *y*>,
- E_7: denoted by $T_{R_{VC}}$ – total VC count, contains VC count value for all routers in the network, and it is defined as:

$$T_{R_{VC}} = \sum_{i=0}^{i=w-1} \sum_{j=0}^{j=h-1} R_{VC<i,j>} \tag{5-1}$$

- E_8: denoted by $T_{R_{EVC}}$ – total EVC count, contains EVC count value for all routers in the network, and it is defined as:

$$T_{R_{EVC}} = \sum_{i=0}^{i=w-1} \sum_{j=0}^{j=h-1} R_{EVC<i,j>} \tag{5-2}$$

- **Data Generator** – based on global (P_{G1} and P_{G2}) and local parameters, it generates the queue of tasks to allocate for the considered CMP. The logical structure of the Data Generator module is described by the relation *O = R (P)* and is presented in Fig. 5-2. The elements of the sub-system are:
  - Problem parameters:

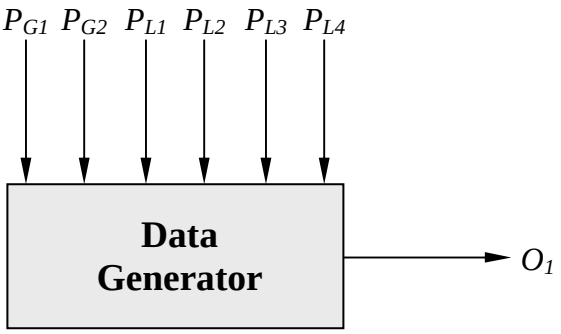


Figure 5-2. Block-diagram of the Data Generator module as input-output system.

- P_{G1}, P_{G2}: global parameters,
- P_{L1}: local parameter – number of jobs created by generator,
- P_{L2}: local parameter – maximum horizontal p and vertical q size of the generated job $J(p, q)$,
- P_{L3}: local parameter – maximum execution time for job J in PEs,
- P_{L4}: local parameter – type of distribution, an element of the set {normal, uniform}.

- Output O_1: queue of jobs. Each job is characterized by: job number, horizontal and vertical size and required execution time.

The generator can produce normally and uniformly distributed pseudo-random numbers. For the size of a job $J(p, q)$ and the execution time of the job, generated numbers are between 1 and local parameters P_{L2} and P_{L3}, respectively. The job number is assigned according to the job position in the queue (first job has number 1, second 2, etc.).

- **Queue with Jobs** – is the result of Data Generator module. The queue of jobs is generated once before experiment. The jobs in the queue are ordered in FCFS fashion and are passed on to the PA module as such.

- **PA** – processor allocator. The physical structure of the PA is presented in Chapter 4. The PA determines the base for a job supplied by the Queue module. The logical structure of the PA module is described by the relation *(O, E) = R (A, P)* and presented in Fig. 5-3. The elements of the sub-system are:

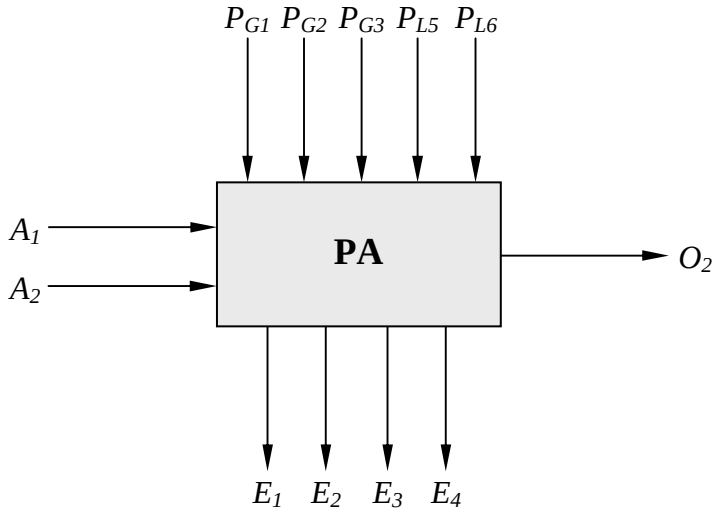


Figure 5-3. Block-diagram of the PA module as input-output system.

  - Inputs:
    - A_1: job to allocate, supplied by the module Queue. The job is characterized by: job number, horizontal and vertical size and required execution time,
    - A_2: job to deallocate, supplied by the module NoC-based CMP. The job is characterized by its size and a base (bit map case) or job number (busy list).

- Problem parameters:
  - P_{G1}, P_{G2}, P_{G3}: global parameters,
  - P_{L5}: local parameter – processor allocation algorithm used in order to determine a base, an element of the set {IFF, IAS, IQA, ISBA, BMAT, BLAT, SAT, SBAT},
  - P_{L6}: local parameter – address $<x, y>$ of the PA in the network. A node with address of the PA can not be a PE for jobs.
- Outputs:
  - O_2: job to allocate with a base determined by the PA, characterized by job number, size and execution time,
  - E_1, E_2, E_3, E_4: evaluation criteria described in the Results 1 module.

- **NoC-based CMP** – is the module simulating the NoC of the CMP. The physical structure was discussed in Chapter 2. The logical structure of the NoC-based CMP module is described by the relation $(O, E) = R (A, P)$ and presented in Fig. 5-4. The elements of the sub-system are:
  - Input A_3: is a job with a base. Based on the A_3, the module is able to determine, which PEs can be reserved for the job. Reservation is done by sending allocation packets from the PA to each processor required by the job,
  - Problem parameters:
    - P_{G1}, P_{G2}, P_{G3}: global parameters,
    - P_{L7}: local parameter – a flow control implemented in the network and number of nodes that can be virtually bypassed using EVCs. For virtual-channel flow

control, P_{L7} is set to 0 (zero nodes can be bypassed). Values greater than 0 mean that considered flow control is an express-virtual-channel with number of bypassed nodes specified by the parameter.

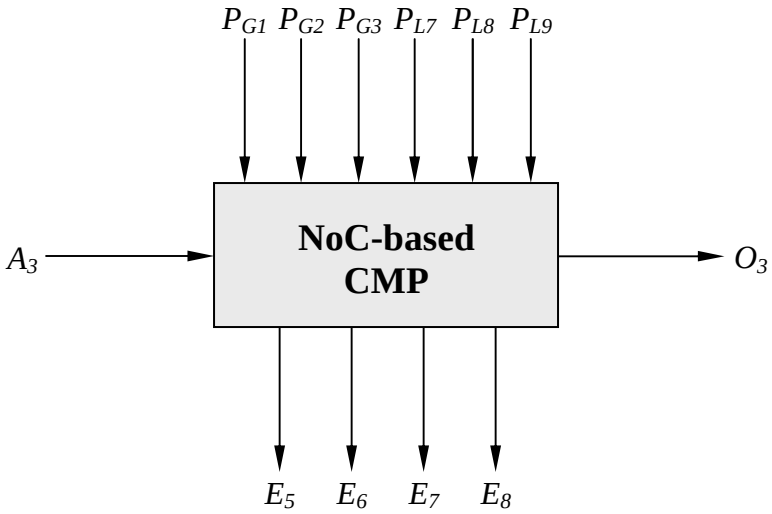


Figure 5-4. Block-diagram of the NoC-based CMP module as input-output system.

- P_{L8}: local parameter – a routing algorithm that determines the route of the packet, being an element of the set {DOR, Valiant, DOR-LB, Valiant-LB, Adaptive},
- P_{L9}: local parameter – indicates an allowed number of misroutes (misroute value). The parameter is valid only in case, when the P_{L8} is set to “Adaptive”.

- Outputs:
  - O_3: job to deallocate with its size and the base,
  - E_5, E_6, E_7, E_8: NoC utilization criteria described in the Results 2 module.

5.2. Implementation of the System and Simulation Methodology

All modules of the experimentation, except the Data Generator were developed in C. The Data Generator module was written in MatLab. The generator returns the queue of jobs as a text file. The queue is processed in the next step by the PA module. The results generated by the PA and NoC-based CMP modules are in the form of text and excel files, to make further data analysis easier.

The PA takes jobs from the Queue, job after job and tries to find free PEs in order to allocate the job. If such free PEs exist, the PEs are allocated for the job (the job is sent together with a base to the NoC-based module). If there is no free PEs, the PA waits until another job will release some PEs – jobs are processed in FCFS fashion. In order to find free PEs for a job, the PA module uses one of the allocation schemes, presented in Chapter 3 of this work. Thus, all mentioned processor allocation algorithms can be tested.

If there is enough free PEs for a job, the PEs can be allocated. An allocation process is done by the NoC-based CMP module. In order to allocate PEs, the allocation message has to be sent from a PA to each PE assigned to a job. It is assumed, that this message takes one flit, e.g. if job requires 6 processors, 6 flits have to be sent from a PA to all PEs assigned to the job. Similarly, if a job is done, a deallocation message (that takes also one flit) has to be sent from each involved PE to the PA. Thus, the PA is updated and the just-released PEs can accommodate another job. In this dissertation the goal is to analyze a PA and traffic generated by that PA, so, only allocation and deallocation messages are considered. In a real system, a lot of different packets are sent among nodes, e.g. just after allocation, messages with commands and operands have to be sent to all the relevant PEs. Similarly while processing, the PEs can exchange control messages and data among

themselves. When processing is done, result messages have to be sent as well. However, in this experimentation system, all messages other than allocation and deallocation message are passed over.

The messages are sent using NoC. In the described experimentation system, NoCs with the following parameters can be evaluated:

- Topologies: *k*-ary 2-mesh and *k*-ary 2-cube,
- Flow controls: virtual-channel and express-virtual-channel,
- Routing algorithms: DOR, Valiant, DOR-LB, Valiant-LB and Adaptive.

As it was mentioned at the beginning of this chapter, the module responsible for NoC was written in C language, which is sequential. It provided higher implementation flexibility over concurrent languages (like e.g. SystemC) without decreasing the quality of this dissertation. Many NoCs have already been implemented [11], [15], [48], [59], [61], [77], [111], [114], and all performance and implementation aspects have been described. Thus in this experimentation system, the exact analysis of the NoC parameters, like performance of routing algorithms, analysis of deadlock and livelock occurrences, timing issues, etc., are not included. Instead of focusing on parameters of NoC, the evaluation system performs detailed analysis of the PA. For the analysis of the PA and its impact on the NoC, the sequential simulation is convenient and adequate. The chosen implementation had an impact on implemented routing algorithms, where deadlock situation could not occur. The DOR, Valiant, DOR-LB and Valiant-LB routing techniques have been designed as described in Chapter 2.3.3. They are oblivious techniques, thus the livelock can not occur as well. The Adaptive routing technique

employed in the system uses historical data of network resource usage in order to route the packet. A physical channel that is used not so often has a priority over the channel used very intensively. As in earlier mentioned techniques, the deadlock situation in the Adaptive algorithm case can not occur. However, the livelock situation is possible. In order to avoid livelocks, the misroute value (which can be defined in the simulation system) is implemented.

The presented experimentation system allows exact analyzing of PA behavior, based on all algorithms presented in this work. Additionally, for the presented NoC solutions, investigation of the traffic generated by PA is also possible. Together with the energy model proposed in Chapter 2.4, an energy consumption of the traffic can be determined as well. The synthesis results presented in Chapter 4.3 provides the energy utilization by the PA based on IFF, IAS, BMAT and BLAT algorithms and for these algorithms complete energy analysis of the processor allocation for the NoC-based CMP can be performed.

5.3. Analysis of Results

The CMP simulator was run on the on Intel Pentium 4 machine (2 × 3GHz processor) with 2 GB of RAM. Due to energy and performance analysis presented in Chapters 3.5 and 4.3, the final experiments were performed for allocation techniques based on a bit map, i.e. the IFF and BMAT algorithms. The rest of allocation schemes were omitted because of their worse characteristics in comparison to the busy array solutions.

For the IFF algorithm, the free PEs allocated to a job are always adjacent like in the mesh topology. However, such adjacent PEs can also communicate among each other and

the PA using the torus topology. Thus, for the PA driven by the IFF algorithm, we consider two cases:

- IFF-mesh – where PEs adjacent like in the mesh communicate between each other using mesh-based NoC,
- IFF-torus – where PEs adjacent like in the mesh communicate between each other using torus-based NoC.

The BMAT technique finds free PEs for the requested job based on the torus topology, thus the neighboring PEs allocated to a job can be adjacent by using the wraparound channels. In order to exchange messages, the PEs could use a mesh-based NoC. However, in such case we could destroy locality of the PEs allocated to one job. Additionally, jobs allocated to PEs using wraparound channels would not be contiguous, if a mesh-based NoC would be used. In this dissertation, the contiguous allocation strategies are considered, thus for the PA based on the BMAT strategy, we consider only a torus-based NoCs as the communication medium.

5.3.1. Experiment 1: Misroute Value Estimation for Adaptive Routing

In the employed Adaptive routing algorithm, the misroute value is used to avoid the possibility of livelocks occurring. In experiment 1, the Adaptive routing algorithm is tuned by adjusting the misroute value, in order to ensure best energy-performance parameters. In experiment 1, four queues of jobs are generated using the Data Generator module with problem parameters:

- P_{G1} for queue 1, 2, 3 and 4: 10, 15, 20 and 30 respectively,
- P_{G2} for queue 1, 2, 3 and 4: 10, 10, 20 and 30 respectively,

- P_{L1}: 1000,
- P_{L2}: $\lceil 0.4 * w \rceil$ and $\lceil 0.4 * h \rceil$,
- P_{L3}: 500,
- P_{L4}: normal.

For each generated queue (according to global parameters P_{G1} and P_{G2}), the PA driven by the IFF and BMAT algorithms is employed. For PA configured like this, the Adaptive routing algorithm for several values of the misroute parameter is tested. During experiment, the PA and NoC-based CMP modules are configured with parameters:

- P_{G1} for queue 1, 2, 3 and 4: 10, 15, 20 and 30 respectively,
- P_{G2} for queue 1, 2, 3 and 4: 10, 10, 20 and 30 respectively,
- P_{G3}: {2D-mesh, 2D-torus},
- P_{L5}: {IFF, BMAT},
- P_{L6}: <0, 0>,
- P_{L7}: 0,
- P_{L8}: Adaptive,
- P_{L9}: {2, 3, 4, 5}.

The results of experiment 1 are presented in Fig. 5-5 to 5-7 (for 10 × 10 grids) and in Table 5-1. In the figures, the VC count $R_{VC<x,\ y>}$ of each router for the considered algorithms is shown. Table 5-1 contains the total energy consumed by network for all NoCs and allocation algorithms implemented in the experimentation system, as a function of the misroute value. As we can see in the figures, the traffic was better

balanced for torus-based NoCs (BMAT and IFF-torus cases – Fig. 5-5 and 5-7). The balancing of traffic has significant impact on the thermal aspects of chip utilization. If traffic has a good balance, routers used are spread across the whole chip. In such case, it

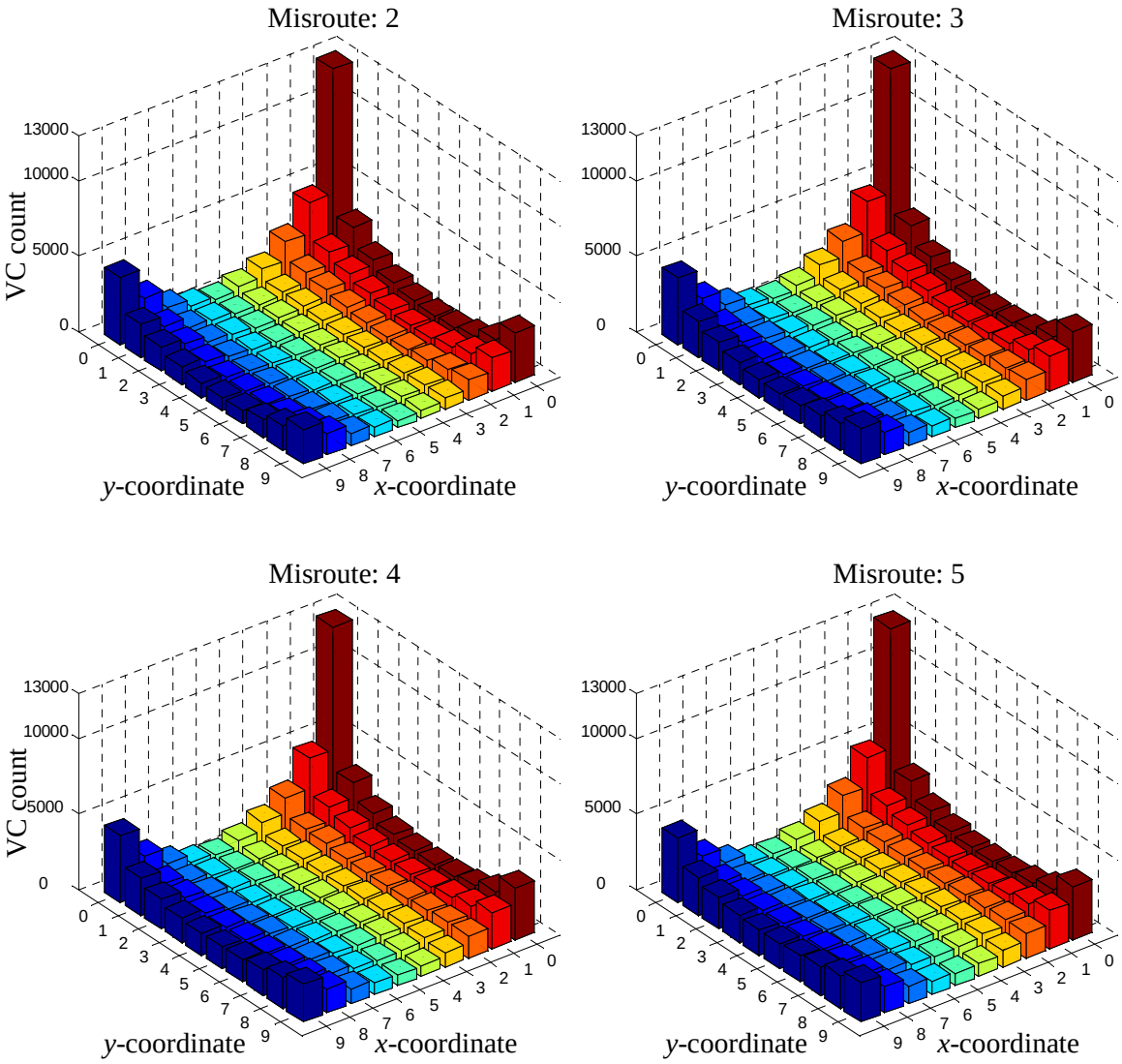


Figure 5-5. VC count of each router in the network for the BMAT algorithm, based on the misroute value.

is much easier to deal with heat dissipation than in the case, where there is weaker traffic balance (Fig. 5-6).

Beside thermal aspects, total energy consumption is also important. NoC energy gathered in Table 5-1 is calculated according to formulas 2-7 and 2-8, presented in Chapter 2.3. During calculations it was assumed that the width of one flit is 32 bits, which is the most popular width used in research [102]. As we can see, the largest

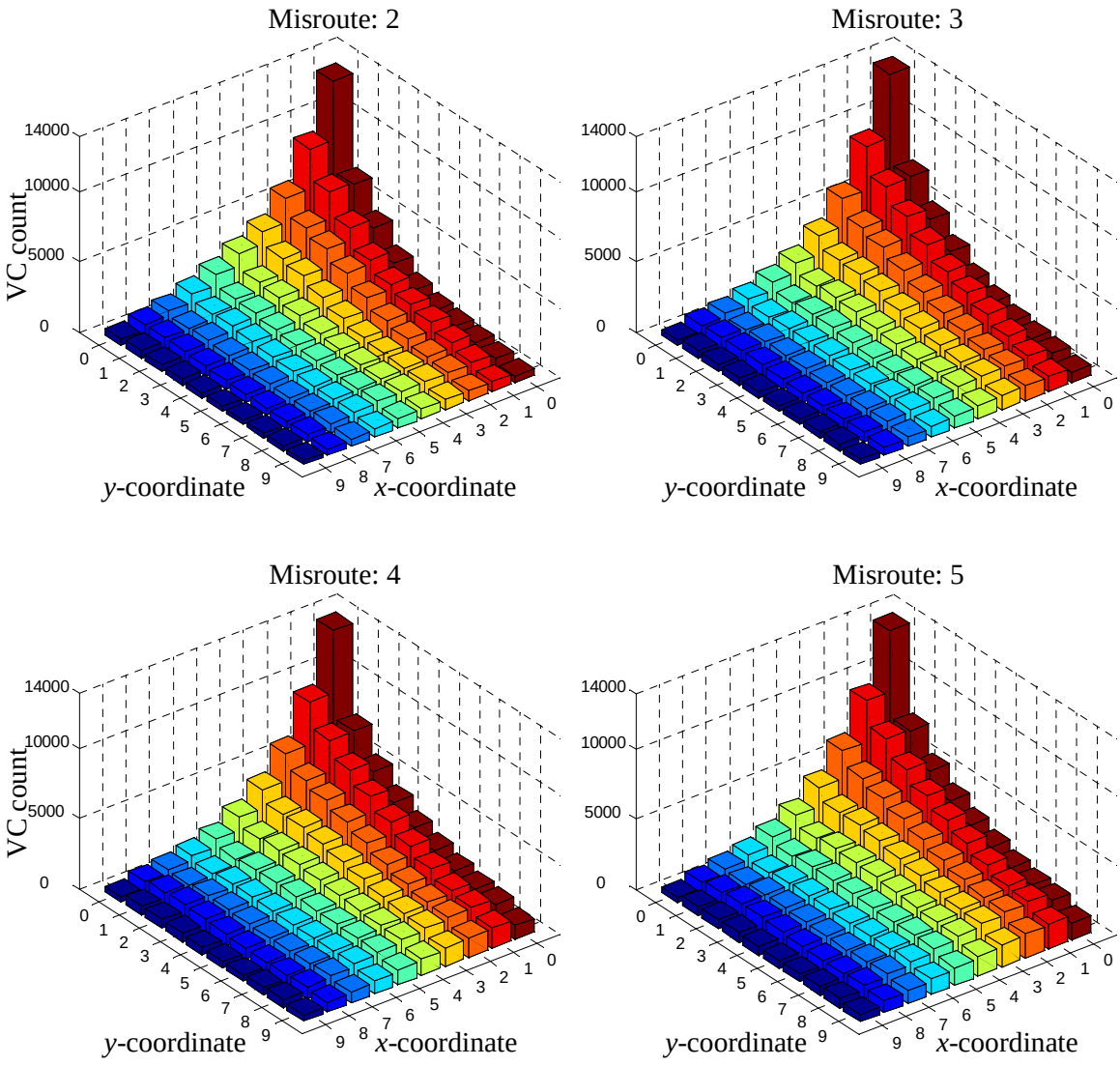


Figure 5-6. VC count of each router in the network in the IFF-mesh case, based on the misroute value.

amount of energy was consumed in mesh-based NoC case. Lack of wraparound channels in mesh topology makes longer routes that increases the number of routers used in transmission together with the amount of energy needed to send a message.

Choosing the misroute value is a trade-off between better traffic balance and total energy consumption. The results presented in the figures confirmed that in all cases, the best traffic balance is achieved with a misroute value of 5 (the highest value in

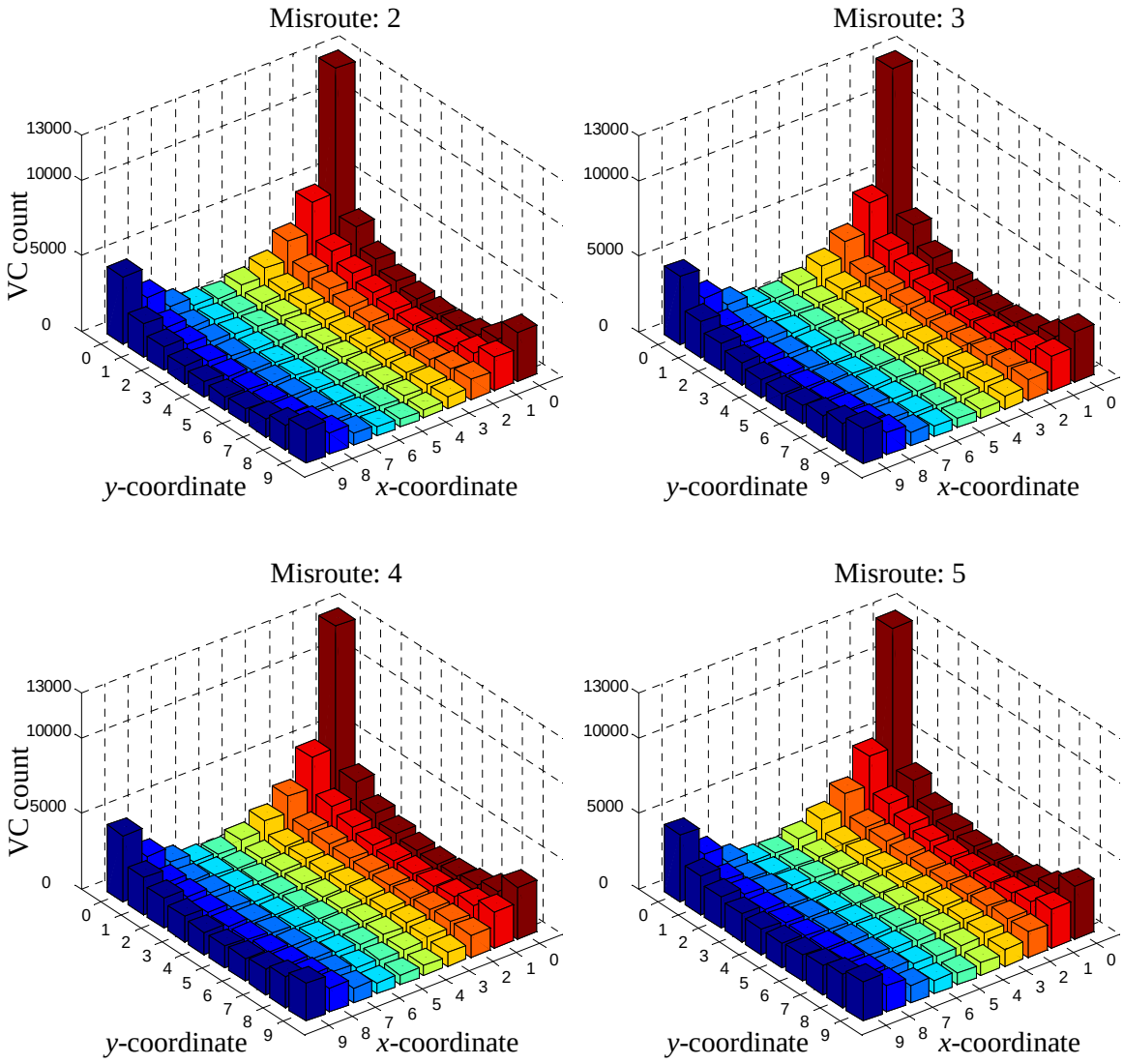


Figure 5-7. VC count of each router in the network in the IFF-torus case, based on the misroute value.

experiment). However, such a configuration consumes the largest amount of energy. Thus, for adaptive routing algorithms for NoC-based CMPs with integrated a PA, the misroute value should be chosen as high as possible to meet the available energy budget together with network delay and latency constrains.

Table 5-1. Energy used by NoC based on the Adaptive routing as a function of allocation algorithm, misroute value, and size of Mesh/Torus.

			NoC Energy [μJ]			
			Size of Mesh/Torus			
Algorithm	**NoC**	**Misroute**	***10 × 10***	***15 × 10***	***20 × 20***	***30 × 30***
IFF	Mesh	2	8.601366	14.316079	49.622252	149.724079
		3	9.657647	16.014383	54.284751	159.848927
		4	10.726626	17.476988	58.204341	167.528595
		5	11.716146	19.07629	62.676972	177.024614
IFF	Torus	2	6.568052	10.792685	35.011799	99.754544
		3	7.579369	12.311376	39.424877	108.87455
		4	8.55293	13.87751	43.825332	119.112497
		5	9.503572	15.352688	48.090045	127.697836
BMAT	Torus	2	6.53085	10.633746	34.561501	97.882933
		3	7.525448	12.156534	38.940477	107.983254
		4	8.53599	13.732688	43.321335	116.586198
		5	9.426732	15.184061	47.902486	125.530492

5.3.2. Experiment 2: Routing Algorithms Comparison

The experimentation system presented in Chapter 5.1 allows comparison of five routing algorithms: DOR, Valiant, DOR-LB, Valiant-LB and Adaptive. The objective of experiment 2 is to evaluate the routing techniques and determine, which routing technique and what PA used in the CMP provide the best energy-performance characteristic. In experiment 2, the job queue from experiment 1 is adopted, where the Data Generator module is configured as follows:

- P_{G1}: 10,
- P_{G2}: 10,
- P_{L1}: 1000,
- P_{L2}: $\lceil 0.4 * w \rceil$ and $\lceil 0.4 * h \rceil$,
- P_{L3}: 500,
- P_{L4}: normal.

Jobs from the queue are allocated by the PA configured with IFF and BMAT algorithms. The jobs are allocated to PEs using NoC driven by one of the five presented routings. NoC is tested for all described routings. For each routine, the same job queue is applied. In experiment 2, the PA and NoC-based CMP modules are configured as follows:

- P_{G1}: 10,
- P_{G2}: 10,
- P_{G3}: {2D-mesh, 2D-torus},
- P_{L5}: {IFF, BMAT},
- P_{L6}: <0, 0>,
- P_{L7}: 0,
- P_{L8}: {DOR, Valiant, DOR-LB, Valiant-LB, Adaptive},
- P_{L9}: {1, 2, 3, 4}.

The obtained energy results are collected in Table 5-2, where energy of PA, NoC and the total energy are presented. An Adaptive-M*x* algorithm in the table represents the

Adaptive algorithm with a maximum number of *x* misroutes. The presented energy results are calculated according to formulas 2-7 and 2-8, and based on the results of synthesis presented in Table 4-3. As in experiment 1, it is assumed that the width of one flit is 32 bits. As it can be noticed, PA with BMAT allocation strategy consumed

Table 5-2. Energy consumption of the PA, NoC and total CMP depending on the used routing algorithm.

	PA Energy [μJ]		**NoC Energy [μJ]**			**Total CMP Energy [μJ]**		
	BMAT	***IFF***	***BMAT***	***IFF-Mesh***	***IFF-Torus***	***BMAT***	***IFF-Mesh***	***IFF-Torus***
DOR	1633.1258	256.8541	4.2233	6.2805	4.2518	1637.3491	263.1346	261.1059
Valiant	1633.1258	256.8541	7.535	10.16	7.5778	1640.6608	267.0141	264.432
DOR-LB	1633.1258	256.8541	4.2233	6.2805	4.2518	1637.3491	263.1346	261.1059
Valiant-LB	1633.1258	256.8541	7.5727	10.1728	7.5859	1640.6985	267.0269	264.4400
Adaptive-M1	1633.1258	256.8541	5.3971	7.4152	5.4128	1638.5229	264.2694	262.2669
Adaptive-M2	1633.1258	256.8541	6.5308	8.6014	6.568	1639.6566	265.4555	263.4222
Adaptive-M3	1633.1258	256.8541	7.5254	9.6576	7.5793	1640.6512	266.5118	264.4335
Adaptive-M4	1633.1258	256.8541	8.536	10.7266	8.5529	1641.6618	267.5808	265.4071

significantly (6 times) more energy than IFF scheme. It is a price for wraparound channel recognition offered by BMAT. Among NoCs considered in the experiment, the NoC with the DOR routing technique achieves the lowest energy usage. It is not a surprise because DOR is the easiest possible algorithm and it routes packets through minimal paths. The Valiant algorithm requires a high amount of energy to route the traffic. Even the most advanced Adaptive routing algorithm with three misroutes allowed, achieves better energy performance. Similarly like in the experiment 1, NoC with mesh topology (IFF-Mesh case in the table) achieved the worst energy results for all routing schemes.

In Fig. 5-8 to 5-10, the VC count $R_{VC<x, y>}$ of each router with DOR, Valiant, DOR-LB, Valiant-LB and Adaptive routing techniques is shown. Adaptive routing is plotted with misroute values 1 and 3. In the first case, the Adaptive algorithm uses the smallest amount of energy, in the second, good traffic balance is achieved and required energy is still smaller than for the Valiant technique. Fig. 5-8 presents NoC, where the PA with BMAT allocation scheme is implemented, Fig. 5-9 and 5-10 have the PAs driven by IFF. In all the figures, the weak traffic balance for DOR and Valiant routings can be noticed. For the DOR, this disadvantage can be compensated by very low energy consumption (Table 5-2). However, the Valiant technique becomes completely unattractive. Low traffic balance provided by DOR and Valiant can be eliminated by load balance extension (DOR-LB and Valiant-LB cases in the figures). Both DOR-LB and Valiant-LB cases bring significant balance improvement, moreover, DOR-LB remains still minimal and still needs the smallest amount of energy. Among all the considered routing techniques, the Adaptive scheme is characterized by the best traffic balance and together with its energy performance becomes very attractive, especially in comparison to Valiant techniques.

Experiment 2 reveals the enormous advantages of IFF-based PA with torus-based NoC. The IFF-Torus approach is characterized by very good energy performance, moreover, torus topology of NoC ensures better traffic balance than mesh. This solution in conjunction with DOR-LB routing technique gives very good energy-balance characteristic. The BMAT-based PA demonstrates very high energy consumption that makes this solution less attractive.

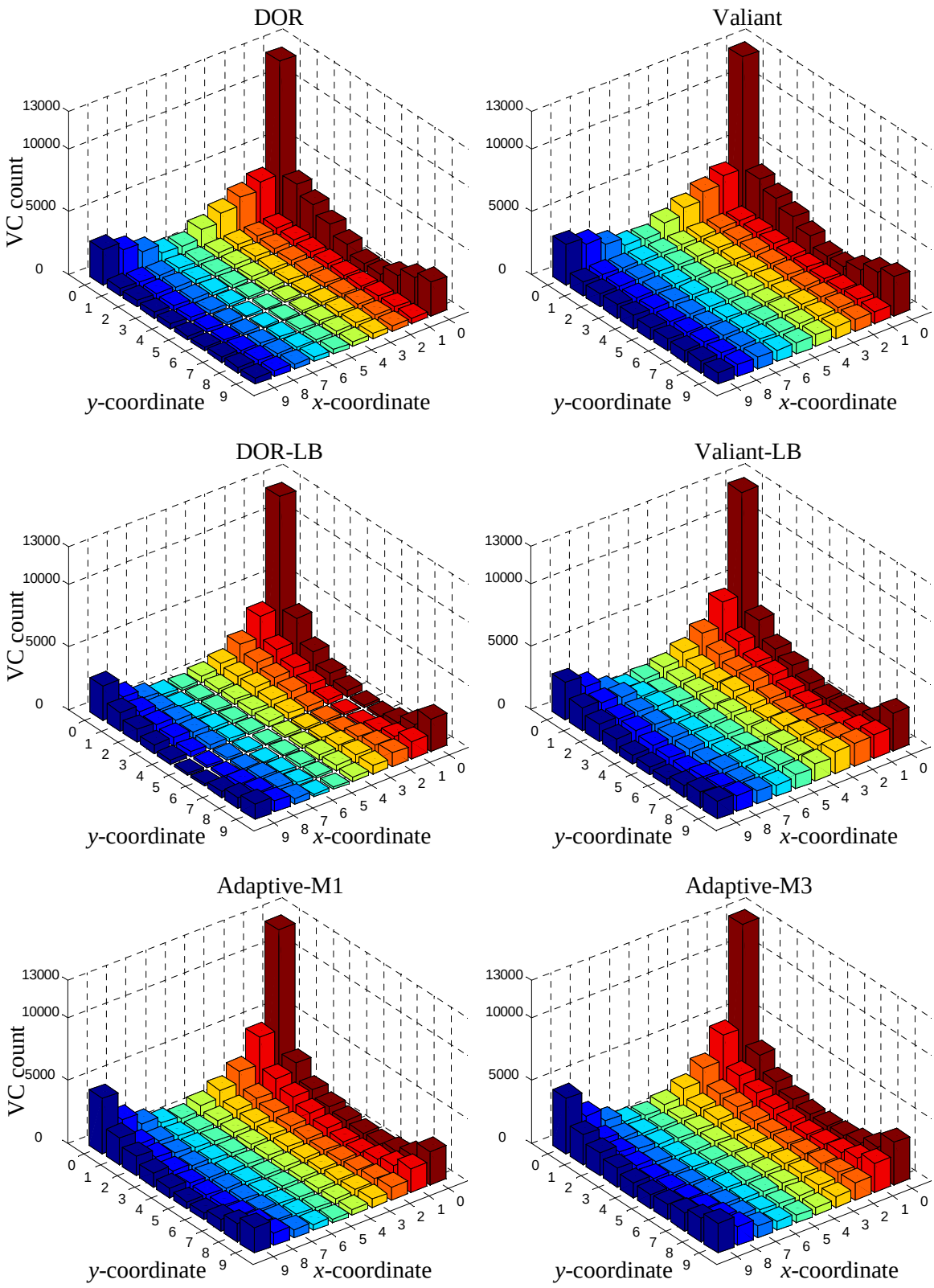


Figure 5-8. VC count of each router in the network with BMAT allocator for all considered routing algorithms.

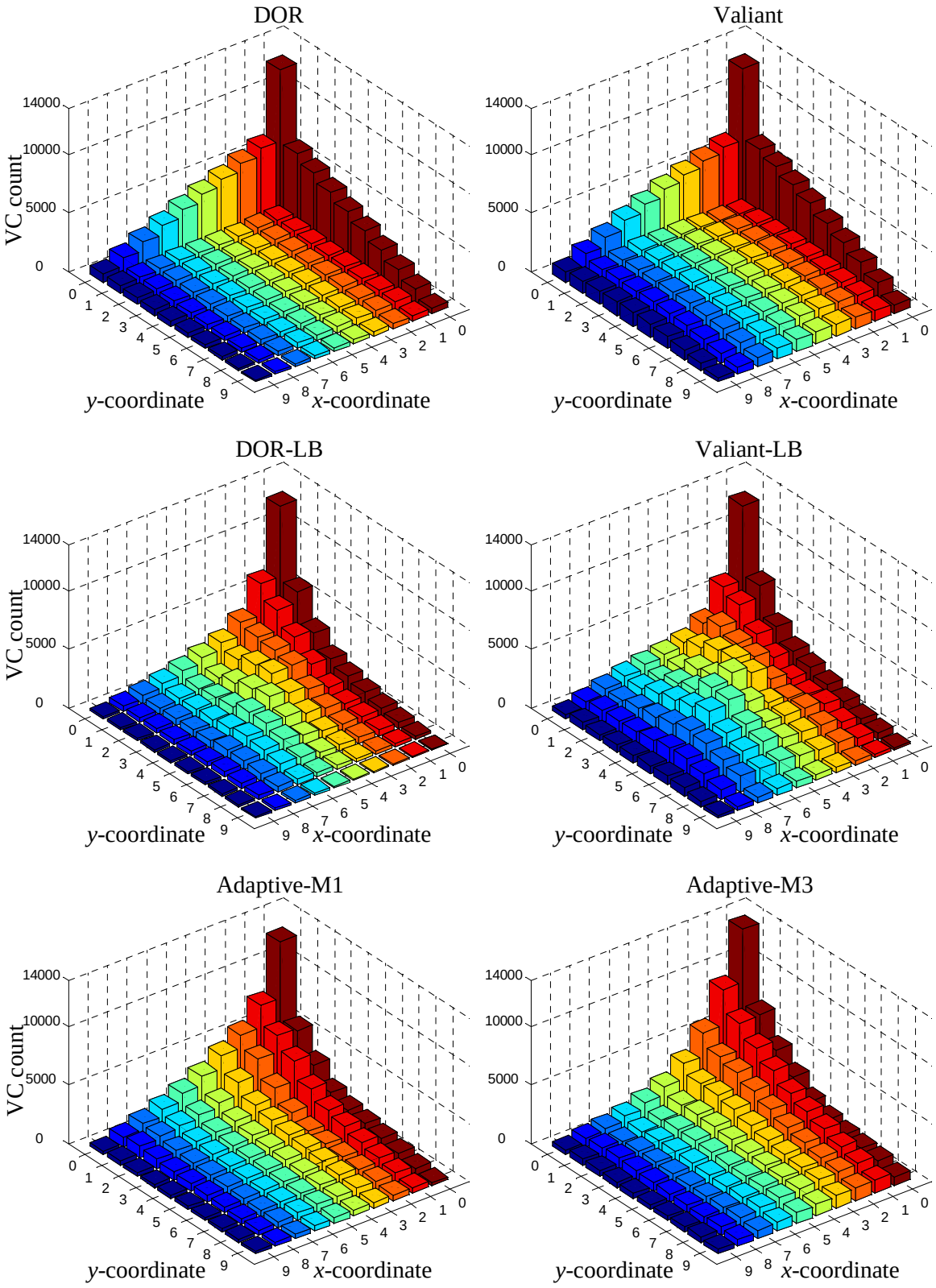


Figure 5-9. VC count of each router in the network in IFF-Mesh case for all considered routing algorithms.

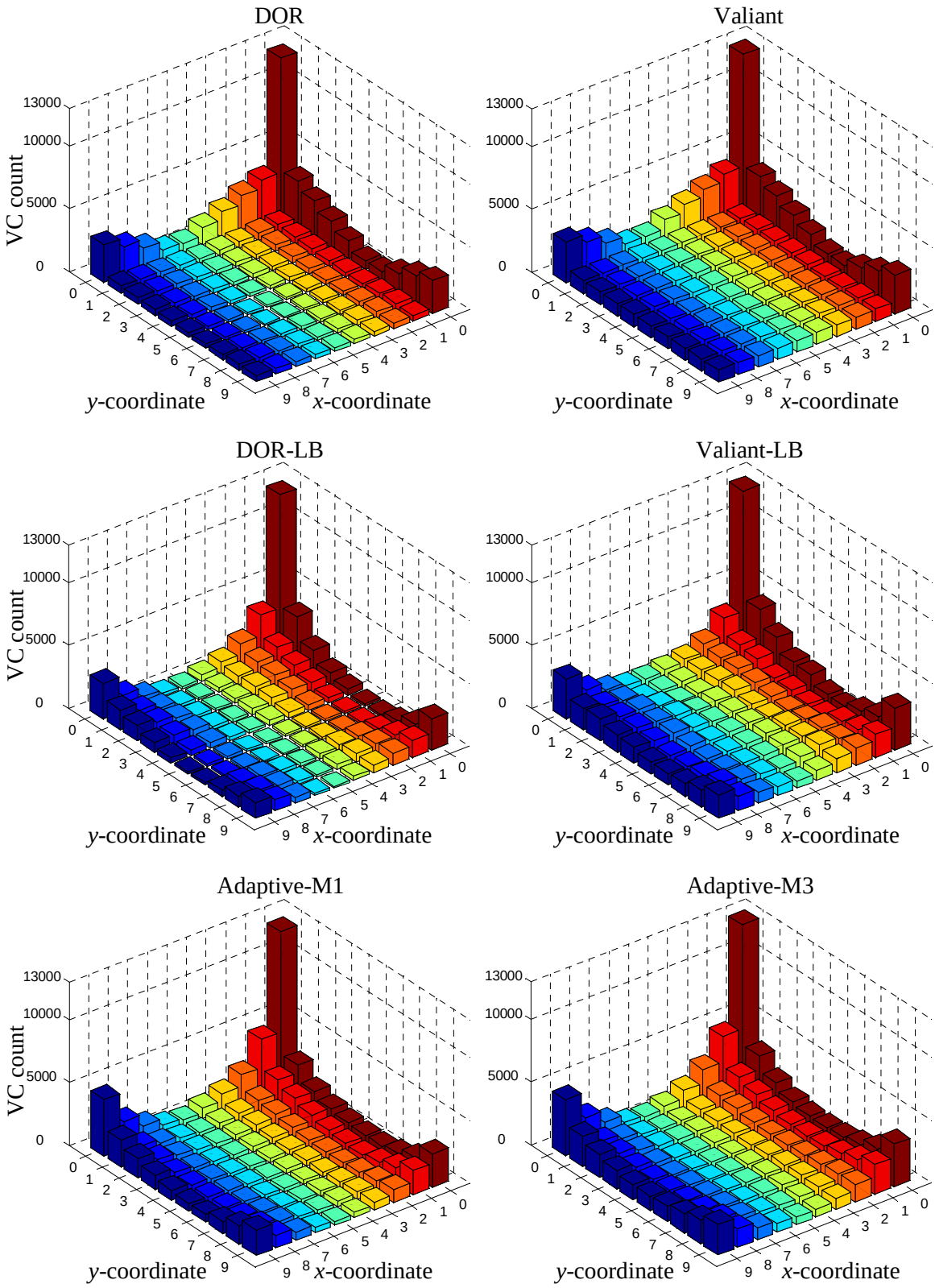


Figure 5-10. VC count of each router in the network in IFF-Torus case for all considered routing algorithms.

5.3.3. Experiment 3: Impact of Express Virtual Channels

In experiment 3, the express-virtual-channel flow control is implemented and compared with its virtual-channel counterpart. It is done by enriching routers in the experimentation systems by express buffers. As in previous experiments, the same queue of jobs has been taken, where Data Generator module is configured as follows:

- P_{G1}: 10,
- P_{G2}: 10,
- P_{L1}: 1000,
- P_{L2}: $\lceil 0.4 * w \rceil$ and $\lceil 0.4 * h \rceil$,
- P_{L3}: 500,
- P_{L4}: normal.

A PA with IFF and BMAT allocation algorithms is employed for the queue. For all combinations, a NoC with all five routing algorithms is investigated. For each routing technique, two flow control mechanisms (virtual-channel and express-virtual-channel) are implemented. In experiment 3, the PA and NoC-based CMP modules are configured as follows:

- P_{G1}: 10,
- P_{G2}: 10,
- P_{G3}: {2D-mesh, 2D-torus},
- P_{L5}: {IFF, BMAT},
- P_{L6}: <0, 0>,
- P_{L7}: {0, 1, 2, 3},

- P_{L8}: {DOR, Valiant, DOR-LB, Valiant-LB, Adaptive},
- P_{L9}: {1, 2, 3, 4}.

The results of experiment for 10 × 10 torus/mesh CMP are presented in Table 5-3 and Figures 5-11 to 5-15. An Adaptive-M*x* algorithm in the table represents the Adaptive algorithm, where the maximum number of misroutes is *x*. The length of EVC in the table and figures represents the parameter P_{L7} – number of routers that can be virtually bypassed by EVC. The presented energy results are calculated according to formulas 2-7 and 2-8. Similarly like in the previous experiments, it is assumed that the width of one flit is 32 bits.

As it can be noticed in Table 5-3, implementation of EVCs for all considered routing techniques decreases the amount of energy used. Especially for mesh-based NoC, the saving offered by express-virtual-channel flow control is significant. The best achieved results are for cases where the number of VC buffers used is reduced by employing more express buffers. Thus, if more express buffers are used, the gain in energy saving is higher. The number of express buffers used depends on the length of EVC, size of mesh/torus and size of jobs. Figures 5-11 to 5-15 present total VC and EVC count as a function of EVC length. The plots are obtained for 10 × 10 NoC. For the NoC under consideration, in almost all cases the higher EVC usage and energy saving is achieved for P_{L7} equals 2 – one express channel bypasses virtually two nodes. In few cases for IFF-Mesh instance (DOR, DOR-LB, Adaptive with misroute 1, 2 and 3), configuration with P_{L7} equals 3 delivered better results.

Table 5-3. Energy consumption by NoC with considered routing algorithms, based on flow control.

	Total VC Count $T_{R_{VC}}$			**Total EVC Count** $T_{R_{EVC}}$			**NoC Energy [μJ]**		
EVC's Length	***BMAT***	***IFF-Mesh***	***IFF-Torus***	***BMAT***	***IFF-Mesh***	***IFF-Torus***	***BMAT***	***IFF-Mesh***	***IFF-Torus***
	DOR								
0	76288	126622	76802	0	0	0	4.223335	6.280482	4.25179
1	58434	75732	58870	17854	50890	17932	3.794839	5.059122	3.821422
2	55752	66034	56290	20536	60588	20512	3.730471	4.82637	3.759502
3	61012	66016	61622	15276	60606	15180	3.856711	4.825938	3.88747
	Valiant								
0	136094	204088	136994	0	0	0	7.53419	10.122796	7.584019
1	101950	126466	102165	34344	79230	34123	6.72101	8.301033	6.725983
2	102924	117066	102858	33488	88212	33856	6.748088	8.064732	6.755974
3	114470	122448	114409	22158	82404	2142	7.031965	8.182994	6.400887
	DOR-LB								
0	76288	126622	76802	0	0	0	4.223335	6.280482	4.25179
1	58434	75732	58870	17854	50890	17932	3.794839	5.059122	3.821422
2	55752	66034	56290	20536	60588	20512	3.730471	4.82637	3.759502
3	61012	66016	61622	15276	60606	15180	3.856711	4.825938	3.88747
	Valiant-LB								
0	134642	200826	138684	0	0	0	7.453812	9.961001	7.677578
1	102251	126408	102017	33137	79300	33709	6.699823	8.299948	6.704807
2	103250	116626	101644	33432	89378	33176	6.764379	8.072758	6.667442
3	115564	123352	115014	21876	84048	20982	7.083686	8.269919	7.025202
	Adaptive-M1								
0	97490	151290	97774	0	0	0	5.397078	7.504015	5.4128
1	80400	100429	80500	18572	51177	18622	5.033393	6.291441	5.040497
2	75928	87972	76222	21994	61660	22124	4.893137	5.941938	4.91349
3	82870	87057	83369	14892	62199	14733	5.054727	5.910353	5.077366
	Adaptive-M2								
0	117970	173398	118638	0	0	0	6.530851	8.600572	6.567831
1	96793	114884	97360	21357	56614	21400	6.028247	7.147596	6.060985
2	93098	104994	93262	24112	64256	23964	5.910089	6.852687	5.914527
3	100861	102979	101193	16821	64773	16737	6.111203	6.765978	6.126948
	Adaptive-M3								
0	135936	194710	136910	0	0	0	7.525448	9.657647	7.579369
1	111871	131938	111927	23177	59162	23187	6.920041	8.058703	6.923454
2	107332	114890	107792	26018	69122	26016	6.757855	7.468098	6.783258
3	116254	115937	116863	18066	67101	17883	7.002402	7.468292	7.030378
	Adaptive-M4								
0	154190	216262	154496	0	0	0	8.53599	10.72663	8.55293
1	126598	145749	127220	25094	62713	25232	7.795444	8.834634	7.834206
2	121560	126822	121348	27860	71738	27702	7.603282	8.126895	7.586591
3	131499	125088	131832	18921	69828	18984	7.873178	7.991993	7.893589

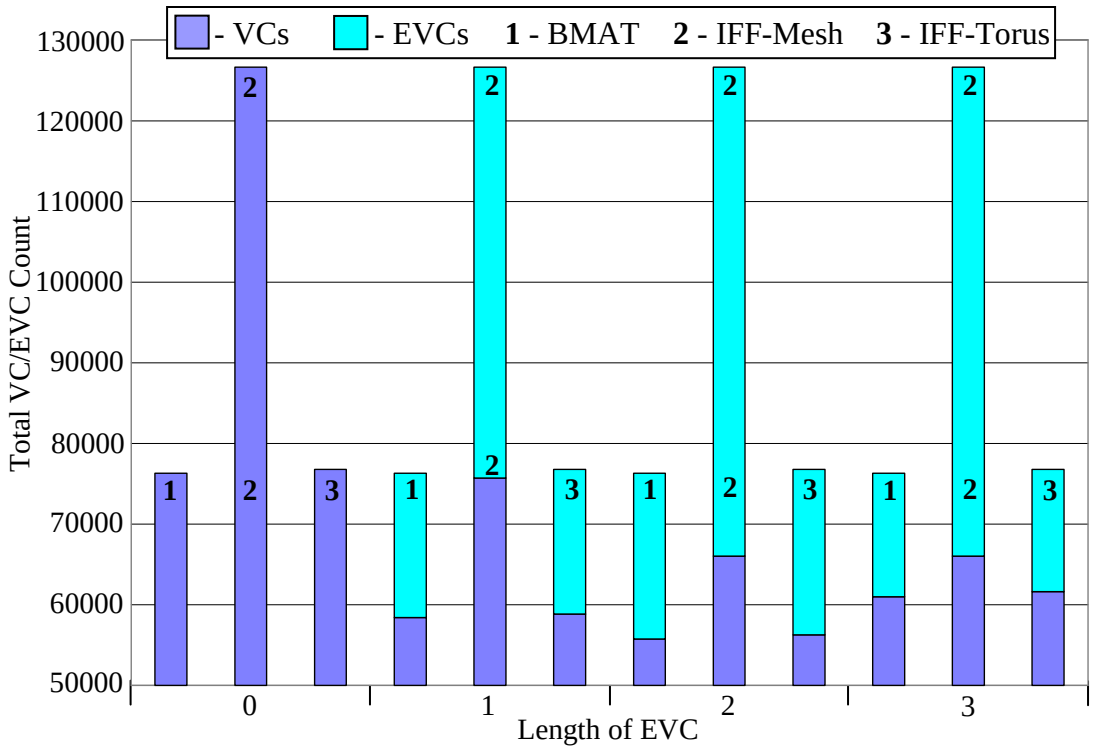


Figure 5-11. Total VC $T_{R_{VC}}$ and EVC $T_{R_{EVC}}$ count for DOR and DOR-LB routing algorithms, based on length of express virtual channel.

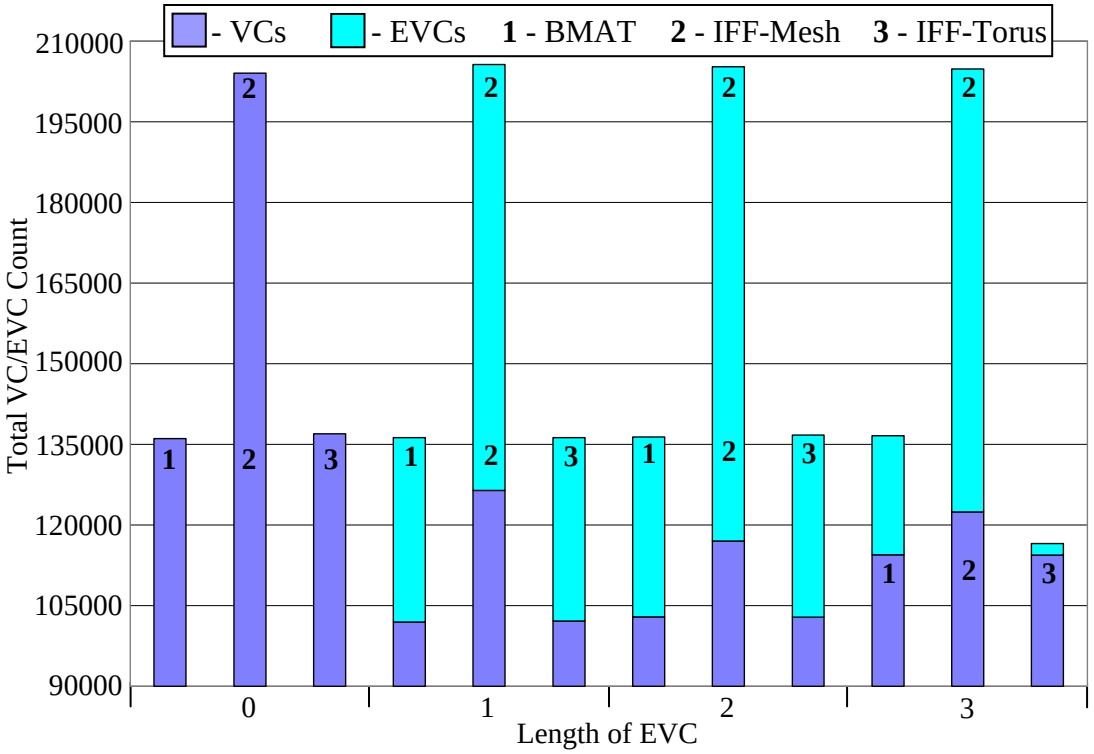


Figure 5-12. Total VC $T_{R_{VC}}$ and EVC $T_{R_{EVC}}$ count for Valiant routing algorithm, based on length of express virtual channel.

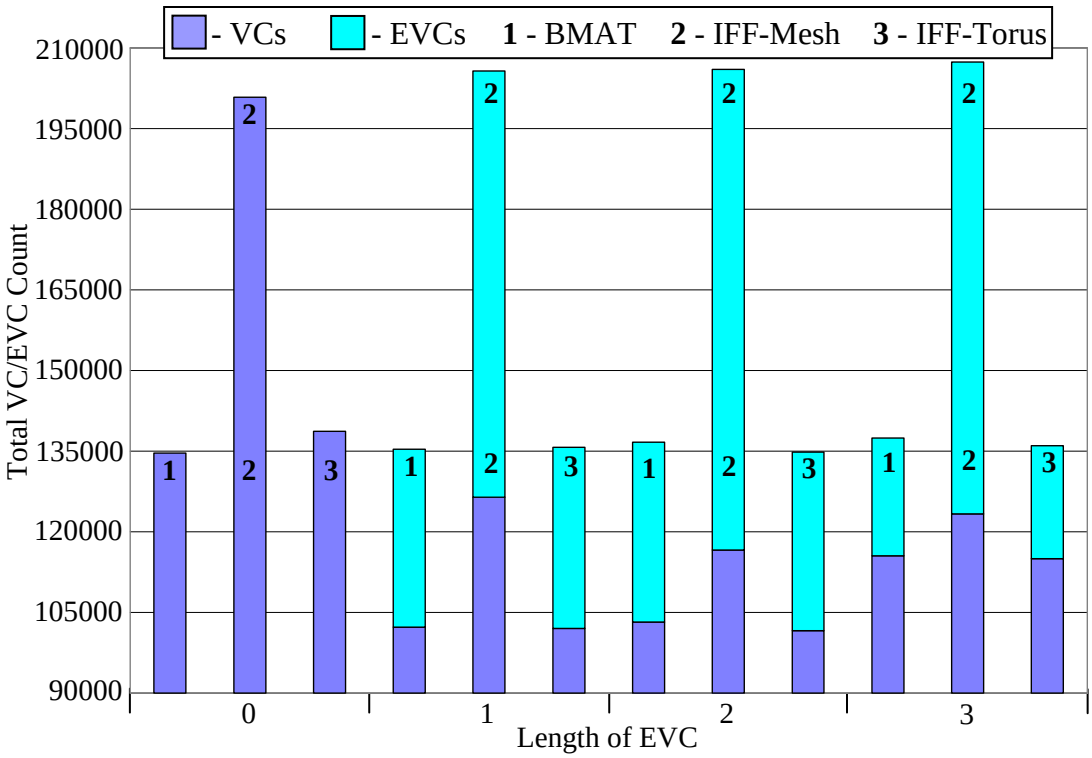


Figure 5-13. Total VC $T_{R_{VC}}$ and EVC $T_{R_{EVC}}$ count for Valiant-LB routing algorithm, based on length of express virtual channel.

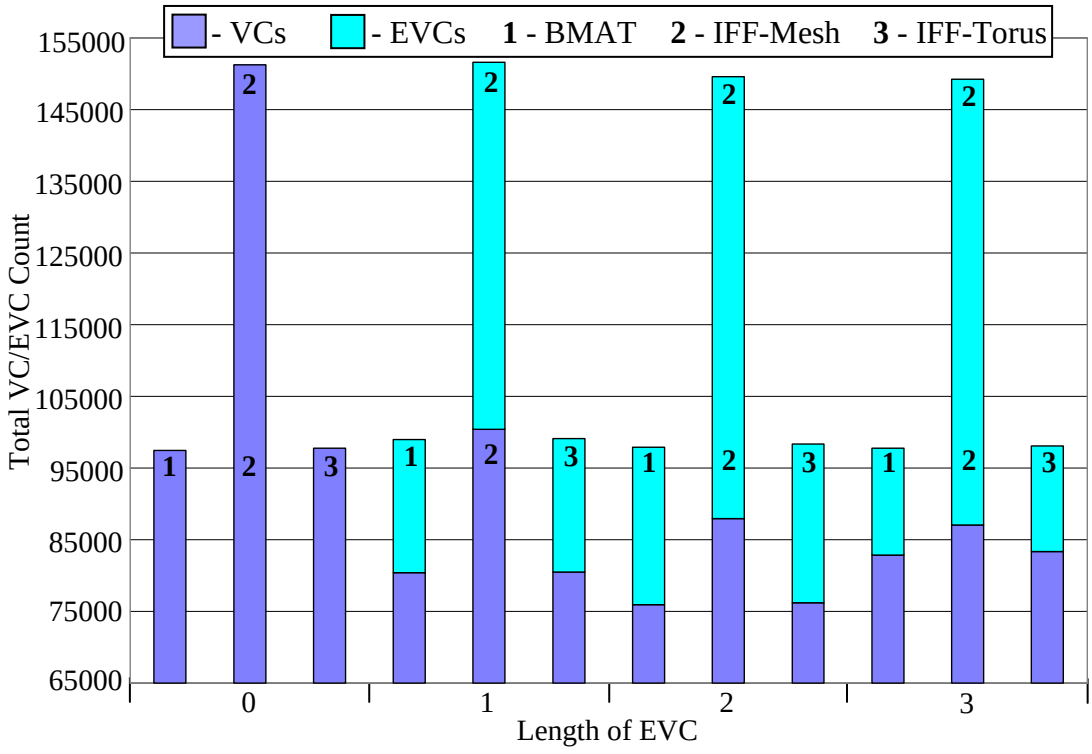


Figure 5-14. Total VC $T_{R_{VC}}$ and EVC $T_{R_{EVC}}$ count for Adaptive routing algorithm with misroute value 1, based on length of express virtual channel.

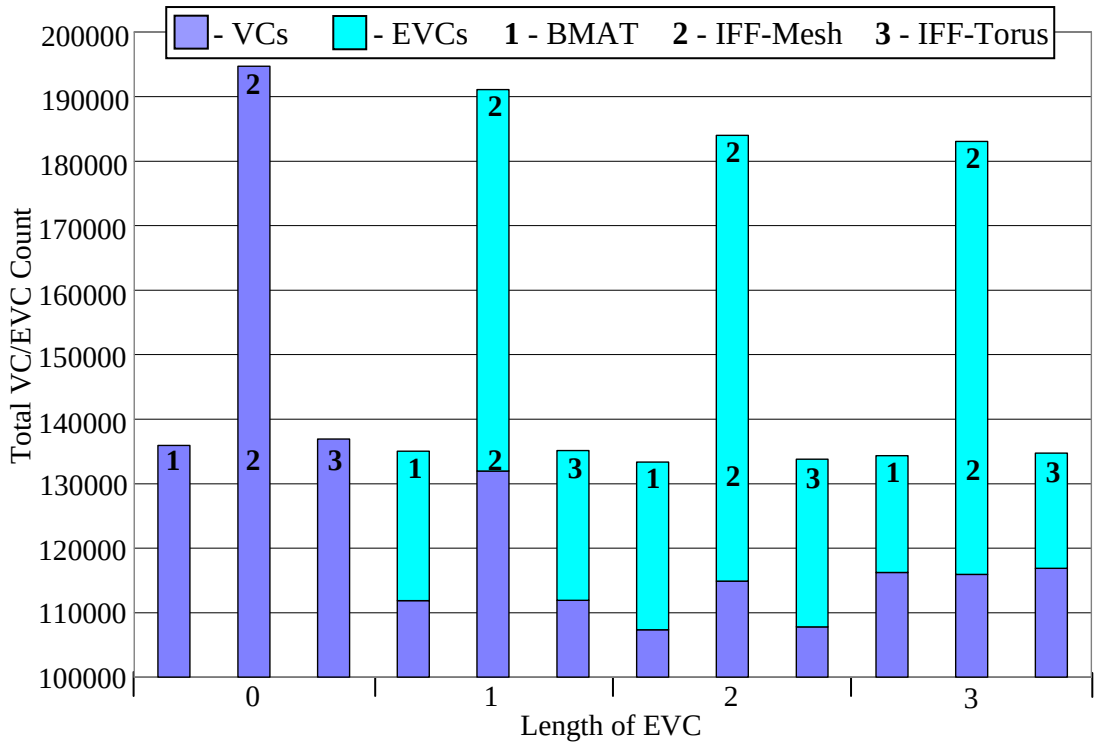


Figure 5-15. Total VC $T_{R_{VC}}$ and EVC $T_{R_{EVC}}$ count for Adaptive routing algorithm with misroute value 3, based on length of express virtual channel.

In all the investigated cases in experiment 3, implementation of express-virtual-channel flow control brings energy saving to make it the clear choice for modern CMPs. Length of EVCs had impact on the amount of energy saved and its choice has to be made individually for each system.

The advantage of EVCs can be observed especially for NoC with mesh topology, where the saving is the highest. However, even with significant benefits offered by express buffers in meshes, the torus topology provides better energy characteristic in all considered cases.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

The continuous development of semiconductor technology in the last four decades has enabled the integration of billions of transistors on a chip and increased the available frequencies. However, for a general purpose single processor, making use of all the available logic in one clock cycle is not possible any more. Moreover, increasing functionality has made centralized control much harder. These factors have led to multi-core designs, where multiple single processors are placed on a single chip, that creates parallel computing system – Chip Multiprocessor.

In the modern CMPs the global buses that interconnect the individual cores are replaced by Network-on-Chip. The NoCs allow concurrent communication of concurrently handled data packets. This increases the communication performance in comparison with bus architectures.

In order to effectively use CMPs, operating system is an important factor and it should support a multiuser environment in which many parallel jobs are executed simultaneously. It is done by the processor management system of the operating system, which consists of two components: Job Scheduler and Processor Allocator.

In this thesis, PA architecture for NoC-based CMPs has been proposed and examined. The PA is driven by processor allocation algorithms, which have been developed and analyzed using experimentation system. The hardware implementation of the PA has been presented and synthesized on the FPGA board. Synthesis results have been obtained showing PA performance characteristics and energy consumption. For the proposed PA that is a part of CMP, the efficient NoC structures have been studied and investigated in

experimentation environment. The presented analysis revealed the image of modern CMP with integrated processor management system.

In our detailed exploration of on-chip interconnection networks in Chapter 2, the topology of efficient NoCs is narrowed to 2-dimensional *k*-ary 2-meshes and *k*-ary 2-cubes. As flow control, virtual-channel and express-virtual-channel mechanisms have been chosen. Among tens of routing algorithms, the oblivious DOR and Valiant techniques with their load balance extensions have been considered. As adaptive technique, the hybrid, deadlock avoidance techniques have been suggested. For the proposed NoC architectures, the energy model has been created.

The suggested NoC topologies are efficient and have direct impact on processor allocation algorithms, which are deliberated in Chapter 3. For 2D-mesh topology, all important allocation techniques have been examined. Additionally, new allocation schemes have been proposed by eliminating drawbacks in former techniques. For 2D-torus network, extensions concerning wraparound channels have been proposed and four new allocation algorithms for torus have been created. In order to compare the efficiency of the considered allocation techniques, an evaluation system has been developed. The results reveal very good performance of allocation algorithms based on bit map. Thus the BMAT and IFF schemes have been chosen as allocation techniques for the PA.

The PA architecture and its synthesis are described in Chapter 4. The synthesis results and energy estimation disclosed the significant advantage of PA based on IFF allocation technique. Among all techniques considered, only busy array schemes are synthesizable, especially the IFF is very promising for future explorations.

The chosen NoC structures and PA architectures are combined together in Chapter 5, where an experimentation system for the NoC-based CMP with integrated the PA has been described. The outcomes confirmed the good results of PA driven by IFF algorithm. However, for NoC, the best load balance and energy consumption are achieved for the torus network. This led to idea of implementing the PA driven by IFF with torus topology of the NoC. This idea turned out to be the best. The IFF-based PA and torus-based NoC driven by DOR-LB routing with express-virtual-channel flow control delivered the best load balance and energy characteristic. If higher reliability and load balance are needed, the adaptive routing technique with carefully chosen parameters can also be a very good idea.

To summarize, the work presented in this thesis has shown the future directions for developing CMPs. It has been shown, that it is possible to implement the PA on the same die with PEs. The proposed PA structure with IFF algorithm is well scalable and can handle fast development of modern CMPs. Similarly, the proposed NoC for CMPs with the PA achieves very good energy-performance characteristic that is optimistic as well.

6.1. Future Work

The research described in this thesis covers first stages in the proposed approach. Since the processor allocation algorithms have been analyzed in depth, we believe that the PA architecture and its hardware implementation can be further developed. The area required by a PA can not be significantly reduced. However, optimization towards better operational frequency and performance could bring some improvements.

Current chip technology is well developed for 2D stacking. 3D stacking is still under research, however we believe, that sooner or later 3D solutions will become more

efficient. Thus, development and improvement of processor allocation solutions for higher dimensions could be one of the directions for future development.

The natural next step of this work is implementation of the proposed CMP with integrated a PA, NoC and PEs on one chip. At the first stage, an FPGA board seems to be an adequate and reasonable solution. All the three components are already separately researched and synthesized, thus additional work should focus on proper configuration and hardware implementation.

Once FPGA version is available, the performance analysis becomes necessary. The best way is to apply available applications to the implemented CMP and compare the results with single-core general processor and CMPs without any integrated processor management system. There are a lot of benchmarks available. However, a good idea is to employ popular applications, such as all kind of compressions algorithms, DSP functions, etc.

The last step of future work would be an ASIC implementation of the CMP, where the computational power of PEs could be much higher than in the case of FPGA board.

BIBLIOGRAPHY

[1] I. Ababneh, "An Efficient Free-List Submesh Allocation Scheme for Two-Dimensional Mesh-Connected Multicomputers," *Journal of Systems and Software*, vol. 79, no. 8, 2006, pp. 1168-1179.

[2] Altera Corporation, "PowerPlay Power Analysis," *Quatrus II 9.1 Handbook, Altera*, vol. 3, 2009.

[3] Altera Corporation, "Stratix III Device Family Overview," *Stratix III Device Handbook*, vol. 1, 2009, Chapter 1.

[4] Altera Corporation, "Stratix III FPGAs vs. Xilinx Virtex-5 Devices: Architecture and Performance Comparison," *White Paper, Altera*, 2007.

[5] AMD Multi-Core, AMD, 2009, http://multicore.amd.com/.

[6] K. V. Anjan, T. M. Pinkston, "An Efficient, Fully Adaptive Deadlock Recovery Scheme: Disha," *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, 1995, pp. 201-210.

[7] K. V. Anjan, T. M. Pinkston, J. Duato, "Generalized Theory for Deadlock-Free Adaptive Wormhole Routing and its Application to Disha Concurrent," *Proceedings of the 10th International Parallel Processing Symposium*, 1996, pp. 815-821.

[8] J. Balfour , W. J. Dally, "Design Tradeoffs for Tiled CMP On-Chip Networks," *Proceedings of the 20th Annual International Conference on Supercomputing*, 2006, pp. 187-198.

[9] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, Lewis M. Mackenzie, "Comparative Evaluation of the Non-Contiguous Processor Allocation Strategies Based on a Real Workload and a Stochastic Workload on Multicomputers," *International Conference on Parallel and Distributed Systems*, vol. 2, 2007, pp. 1-7.

[10] L. Benini, G. D. Micheli, "Networks on Chips: A New SoC Paradigm," *Computer*, vol. 35, no. 1, 2002, pp. 70-78.

[11] T. Bjerregaard, "The MANGO Clockless Network-on-Chip: Concepts and Implementation," *Ph.D. Thesis, Technical University of Denmark*, 2005.

[12] T. Bjerregaard, S. Mahadevan, "A Survey of Research and Practices of Network-on-Chip," *ACM Computing Surveys (CSUR)*, vol. 38, no. 1, 2006, article no. 1.

[13] B. Bose, B.Broeg, Y. Kwon, Y.Ashir, "Lee Distance and Topological Properties of k-ary n-cubes," *IEEE Transactions on Computers*, vol. 44, no. 8, 1995, pp. 1021-1030.

[14] Y. M. Boura, C. R. Das, "Efficient Fully Adaptive Wormhole Routing in n-Dimensional Meshes," *Proceedings of the 14th International Conference on Distributed Computing Systems*, 1994, pp. 589-596.

[15] S. Bourduas, "Modeling, Evaluation, and Implementation of Ring-Based Interconnects for Network-on-Chip," *Ph.D. Thesis, McGill University*, 2008.

[16] G. C. Cardarilli, A. D. Re, A. Nannarelli, M. Re, "Power Characterization of Digital Filters Implemented on FPGA," *IEEE International Symposium on Circuits and Systems ISCAS 2002*, vol. 5, 2002, pp. 801-804.

[17] C. - Y. Chang, P. Mohapatra, "An Integrated Processor Management Scheme for Mesh-Connected Multicomputer Systems," *Proceedings of the international Conference on Parallel Processing (ICPP)*, 1997, pp. 118-121.

[18] K. C. Chang, J. S. Shen, T. F. Chen, "Evaluation and Design Trade-Offs Between Circuit-Switched and Packet-Switched NoCs for Application-Specific SoCs," *Proceedings of the 43rd Annual Conference on Design Automation*, 2006, pp. 143-148.

[19] G. I. Chen, T. H Lai, "Scheduling Independent Jobs on Partitionable Hypercubes," Journal *of Parallel and Distributed Computing*, vol. 12, no. 1, 1991, pp. 74-78.

[20] H. - L. Chen, C. - T. King, "Dynamic Processor Allocation in Scalable Multiprocessors Using Boolean Algebra," *International Journal of Computer Mathematics*, vol. 67, no. 3 & 4, 1998, pp. 333–358.

[21] M. - C. Chiang, G. S. Sohi, "Evaluating Design Choices for Shared Bus Multiprocessors in a Throughput-Oriented Environment," *IEEE Transactions on Computers*, vol. 41, no. 3, 1992, pp. 297-317.

[22] A. A. Chien, J. H. Kim, "Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors," *Proceedings of the 19th Annual International Symposium on Computer Architecture*, 1992, pp. 268-277.

[23] G. - M. Chiu, "The Odd-Even Turn Model for Adaptive Routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, 2000, pp. 729-738.

[24] G. Chmaj, D. Zydek, L. Koszalka, "Comparison of Task Allocation Algorithms for Mesh-structured Systems," *In Computer Systems Engineering, Theory & Applications, Fourth Polish-British Workshop*, 2004, pp. 39-50.

[25] P. J. Chuang, N. F. Tzeng, "An Efficient Submesh Allocation Strategy for Mesh Computer Systems," *Proceedings of the 11th International Conference on Distributed Computing Systems*, 1991, pp. 256-263.

[26] P. - J. Chuang, C. - M. Wu, "An Efficient Recognition-Complete Processor Allocation Strategy for k-ary n-cube Multiprocessors," *IEEE Transaction on Parallel and Distributed Systems*, vol. 11, no. 5, 2000, pp. 485-490.

[27] W. J. Dally, "Express Cubes: Improving the Performance of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers*, vol. 40, no. 9, 1991, pp. 1016-1023.

[28] W. J. Dally, "Fine-Grain Message Passing Concurrent Computers," *Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications*, vol. 1, 1988, pp. 2-12.

[29] W. J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers*, vol. 39, no. 6, 1990, pp. 775-785.

[30] W. J. Dally, "Virtual-Channel Flow Control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, 1992, pp. 194-205.

[31] W. J. Dally, H. Aoki, "Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 4, 1993, pp. 466-475.

[32] W. J. Dally, C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, vol. 36, no. 5, 1987, pp. 547-553.

[33] W. J. Dally, C. L. Seitz, "The Torus Routing Chip," *Journal of Distributed Computing*, vol. 1, no. 4, 1986, pp. 187-196.

[34] W. J. Dally, B. Towles, "Principles and Practices of Interconnection Networks," *Morgan Kaufmann*, 2004.

[35] W. J. Dally, B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 684-689.

[36] D. Das Sharma, D. K. Pradhan, "A Fast and Efficient Strategy for Submesh Allocation in Mesh-Connected Parallel Computers," *Proceedings of the 5th IEEE Symposium on Parallel and Distributed Processing*, 1993, pp. 682-689.

[37] D. Das Sharma, D. K. Pradhan, "Job Scheduling in Mesh Multicomputers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 1, 1998, pp. 57-70.

[38] D. Das Sharma, D. K. Pradhan, "Processor Allocation in Hypercube Multicomputers: Fast and Efficient Strategies for Cubic and Noncubic Allocation," *IEEE Transaction on Parallel and Distributed Systems*, vol. 6, no. 10, 1995, pp. 1108-1122.

[39] J. Ding, L. N. Bhuyan, "An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected Systems," *Proceedings of the International Conference on Parallel Processing*, vol. 2, 1993, pp. 193-200.

[40] J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 12, 1993, pp. 1320-1331.

[41] J. Duato, S. Yalamanchili, L. Ni, "Interconnection Networks," *Morgan Kaufmann*, 2003.

[42] N. Eisley, L. – S. Peh, "High-Level Power Analysis for On-Chip Networks," Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, 2004, 104-115.

[43] D. G. Feitelson, M. A. Jette, "Improved Utilization and Responsiveness with Gang Scheduling," *Proceedings of the Job Scheduling Strategies for Parallel Processing, LNCS*, vol. 1291, 1997, pp. 238-261.

[44] F. Feliciian, S. B. Furber, "An Asynchronous On-Chip Network Router with Quality-of-Service (QoS) Support," *Proceedings of the IEEE International SOC Conference*, 2004, pp. 274-277.

[45] G. Ferizis, H. E. Gindy, "Mapping Recursive Functions to Reconfigurable Hardware," *International Conference on Field Programmable Logic and Applications, FPL '06*, 2006, pp. 1-6.

[46] C. J. Glass, L. M. Ni, "Maximally Fully Adaptive Routing in 2D Meshes," *Proceedings of the International Conference on Parallel Processing*, vol. 1, 1992, pp. 101-104.

[47] C. J. Glass, L. M. Ni, "The Turn Model for Adaptive Routing," *Proceedings of the 19th Annual International Symposium on Computer Architecture (ISCA)*, 1992, pp. 278-287.

[48] K. Goossens, J. Dielissen, A. Radulescu, "AEthereal Network on Chip: Concepts, Architectures, and Implementations," *IEEE Design & Test of Computers*, vol. 22, no. 5, 2005, pp. 414-421.

[49] R. Grindley, "The NUMAchine Multiprocessor," *Proceedings of the International Conference on Parallel Processing*, 2000, pp. 487-496.

[50] B. Grot, S. W. Keckler, "Scalable On-Chip Interconnect Topologies," *CMP-MSI: 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects*, 2008.

[51] M. Gschwind *et al.*, "Synergistic Processing in Cell's Multicore Architecture," *IEEE Micro*, vol. 26, no. 2, 2006, pp. 10-24.

[52] J. Harkins, T. El-Ghazawi, E. El-Araby, M. Huang, "Performance of Sorting Algorithms on the SRC 6 Reconfigurable Computer," *Proceedings of the International Conference on Field-Programmable Technology*, 2005, pp. 295-296.

[53] J. Hu, "Design Methodologies for Application Specific Networks-on-Chip," *Ph.D. Thesis, Carnegie Mellon University*, 2005

[54] Intel® Multi-Core Technology, Intel, 2009, http://www.intel.com/multi-core/.

[55] International Technology Roadmap for Semiconductors 2007 Edition-Executive Summary, *ITRS*, 2007, http://www.itrs.net/.

[56] A. Jantsch, H. Tenhunen, "Networks on Chip," *Kluwer Academic Publishers*, 2003.

[57] D. N. Jayasimha, B. Zafar, Y. Hoskote, "On-Chip Interconnection Networks: Why They are Different and How to Compare Them," *Intel*, 2006.

[58] S. Kambhatla, "Hypercube Vs Cube-Connected Cycles: A Topological Evaluation," *Proceedings of the 6th Distributed Memory Computing Conference*, 1991, pp. 703-706.

[59] H. Kariniemi, "On-line Reconfigurable Extended Generalized Fat Tree Network-on-Chip for Multiprocessor System-on-Chip Circuits," *Ph.D. Thesis, Tampere University of Technology*, 2006.

[60] N. Kavaldjiev, G. J. M. Smit, and P. G. Jansen, "A Virtual Channel Router for On-Chip Networks," *Proceedings of IEEE International SOC Conference*, 2004, pp. 289-293.

[61] N. K. Kavaldjiev, "A run-time reconfigurable Network-on-Chip for streaming DSP applications," *Ph.D. Thesis, University of Twente*, 2007.

[62] P. Kermani, L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, vol. 3, 1979, pp. 267-286.

[63] G. Kim, H. Yoon, "On Submesh Allocation for Mesh Multicomputers: A Best-Fit Allocation and a Virtual Submesh Allocation for Faulty Meshes," *IEEE Transaction on Parallel and Distributed Systems*, vol. 9, no. 2, 1998, pp. 175-185.

[64] J. Kim, J. Balfour, W. J. Dally, "Flattened Butterfly Topology for On-Chip Networks," *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 172-182.

[65] J. Kim, W. J. Dally, D. Abts, "Flattened Butterfly: A Cost-Efficient Topology for High-Radix Networks," *Proceedings of the 34th Annual International Symposium on Computer Architecture*, 2007, pp. 126-137.

[66] J. Kim, C. R. Das, W. Lin, "A Top-Down Processor Allocation Scheme for Hypercube Computers," *IEEE Transaction on Parallel and Distributed Systems*, vol. 2, no. 1, 1991, pp. 20-30.

[67] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, C.R. Das, "A Low Latency Router Supporting Adaptivity for On-Chip Interconnects," *Proceedings of the 42nd annual Design Automation Conference*, 2005, pp. 559-564.

[68] J. H. Kim, Z. Liu, A. A. Chien, "Compressionless Routing: A Framework for Adaptive and Fault-Tolerant Routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 3, 1997, pp. 229-244.

[69] L. Koszalka, "Static and Dynamic Allocation Algorithms in Mesh Structured Networks," *Proceedings of the 3rd International Conference on Distributed Computing and Internet Technology, LNCS*, vol. 4317, 2006, pp. 89-101.

[70] E. Krevat, J. G. Castanos, J. E. Moreira, "Job Scheduling for the BlueGene/L System," *Proceedings of the 8th International Euro-Par Conference Paderborn, LNCS*, vol. 2400, 2002, pp. 207-211.

[71] T. Krishna, A. Kumar, P. Chiang, M. Erez, L. - S. Peh, "NoC with Near-Ideal Express Virtual Channels Using Global-Line Communication," *Proceedings of the 2008 16th IEEE Symposium on High Performance Interconnects*, 2008, pp. 11-20.

[72] P. Krueger, T. - H. Lai, and V. A. Dixit-Radiya, "Job Scheduling Is More Important than Processor Allocation for Hypercube Computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 5, 1994, pp. 488-497.

[73] M. Kubiak, "Analiza Wydajnosci Dynamicznych Algorytmow Alokacji Zadan w Systemach Zorientowanych Siatkowo," *M.Sc. Thesis, Wrocław University of Technology*, 2006.

[74] A. Kulmala, O. Lehtoranta, T. D. Hamalainen, M. Hannikainen, "Scalable MPEG-4 Encoder on FPGA Multiprocessor SOC," *EURASIP Journal on Embedded Systems*, vol. 2006, no. 1, 2006, pp. 1-15.

[75] A. Kumar, L. - S. Peh, P. Kundu, N. K. Jha, "Express Virtual Channels: Towards the Ideal Interconnection Fabric," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, 2007, pp. 150-161.

[76] S. Kumar *et al.*, "A Network on Chip Architecture and Design Methodology," *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'02)*, 2002, pp.105-112.

[77] J. Liang, A. Laffely, S. Srinivasan, R. Tessier, "An Architecture and Compiler for Scalable On-Chip Communication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 7, 2004, pp. 711-726.

[78] D. H. Linder, J. C. Harden, "An Adaptive and Fault-Tolerant Wormhole Routing Strategy for k-ary n-cubes," *IEEE Transactions on Computers*, vol. 40, no. 1, 1991, pp. 2-12.

[79] J. W. S. Liu, "Real-Time Systems," *Prentice Hall*, 2000.

[80] T. Liu, W. - K. Huang, F. Lombardi, L. N. Bhuyan, "A Submesh Allocation Scheme for Mesh-Connected Multiprocessor Systems," *Proceedings of the 1995 International Conference on Parallel Processing*, vol. II, 1995, pp. 159–163.

[81] D. T. Marr, S. Natarajan, S. Thakkar, R. Zucker, "Multiprocessor Validation of the Pentium Pro," *Computer*, vol. 29, no. 11, 1996, pp. 47-53.

[82] P. Mazumder, "Evaluation of On-Chip Static Interconnection Networks," *IEEE Transactions on Computers*, vol. 36, no. 3, 1987, pp. 365-369.

[83] P. M. Merlin, P. J. Schweitzer, "Deadlock Avoidance in Store-and-Forward Networks--I: Store-and-Forward Deadlock," *IEEE Transactions on Communications*, vol. 28, no. 3, 1980, pp. 345-354.

[84] G. D. Micheli, L. Benini, "Networks on Chips: Technology and Tools," *Morgan Kaufman*, 2006.

[85] D. Min, M. W. Mutka, "Efficient Job Scheduling in a Mesh Multicomputer without discrimination against large jobs," *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing (SPDP '95)*, 1995, pp. 52-59.

[86] P. Mohapatra, C. Yu, C. R. Das, and J. Kim, "A Lazy Scheduling for Improving Hypercube Performance," *Proceedings of the 1993 International Conference on Parallel Processing (ICPP '93)*, vol. 1, 1993, pp. 110-117.

[87] G. E. Moore, "Cramming More Components onto Integrated Circuits," *Proceedings of the IEEE*, vol. 86, no. 1, 1998, pp. 82-85.

[88] A. W. Mu'alem, D. G. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, 2001, pp. 529-543.

[89] L. M. Ni, P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer*, vol. 26, no. 2, 1991, pp. 62-76.

[90] S. R. Ohring, M. Ibel, S. K. Das, M. J. Kumar, "On Generalized Fat Trees," *Proceedings of the 9th International Symposium on Parallel Processing (IPPS)*, 1995, pp. 37-44.

[91] M. Oka, M. Suzuoki, "Designing and Programming the Emotion Engine," *IEEE Micro*, vol. 19, no. 6, 1999, pp. 20-28.

[92] J. Palmer, B. Nelson, "A Parallel FFT Architecture for FPGAs," *Proceedings of the 14th International Conference on Field Programmable Logic and Application*, *LNCS*, vol. 3203, 2004, pp. 948-953.

[93] D. C. Pham *et al.*, "Overview of the Architecture, Circuit Design, and Physical Implementation of a First-Generation Cell Processor," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, 2006, pp. 179-196.

[94] T. M. Pinkston, S. Warnakulasuriya, "Characterization of Deadlocks in k-ary n-Cube Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 9, 1999, pp. 904-921.

[95] F. P. Preparata, J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *Communications of the ACM*, vol. 24, no. 5, 1981, pp. 300-309.

[96] W. Qiao, L. M. Ni, "Efficient Processor Allocation for 3D Tori," *Proceedings of the 9th International Symposium on Parallel Processing*, 1995, 466-471.

[97] V. Raghunathan, M. B. Srivastava, R. K. Gupta, "A survey of techniques for energy efficient on-chip communication," *Proceedings of the 40th annual Design Automation Conference*, 2003, pp. 900-905.

[98] D. A. Reed, D. C. Grunwald, "The Performance of Multicomputer Interconnection Networks," *Computer*, vol. 20, no. 6, 1987, pp. 63-73.

[99] M. Rezazad, H. Sarbazi-azad, "The Effect of Virtual Channel Organization On the Performance of Interconnection Networks," *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, work. 14, vol. 15, 2005.

[100] C. A. F. D. Rose, H.- U. Heiss, "Dynamic Processor Allocation in Large Mesh-Connected Multicomputers," *Proceedings of the 7th International Euro-Par Conference Manchester, LNCS*, vol. 2150, 2001, pp. 783-792.

[101] C. A. F. D. Rose, H. - U. Heiss, B. Linnert, "Distributed Dynamic Processor Allocation for Multicomputers," *Parallel Computing*, vol. 33, no. 3, 2007, pp. 145-158.

[102] E. Salminen, A. Kulmala, T. D. Hamalainen, "Survey of Network-on-Chip Proposals," *White Paper, OCP-IP*, 2008, pp. 1-13.

[103] C. - C. Su, K. G. Shin, "Adaptive Deadlock-Free Routing in Multicomputers Using Only One Extra Virtual Channel," *Proceedings of the 1993 International Conference on Parallel Processing*, vol. 1, 1993, pp. 227-231.

[104] H. Sullivan, T. R. Bashkow, "A Large Scale, Homogeneous, Fully Distributed Parallel Machine, I," *ACM SIGARCH Computer Architecture News*, vol. 5, no. 7, 1977, pp. 105-117.

[105] M. B. Taylor *et al.*, "The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs," *IEEE Micro*, vol. 22, no. 2, 2002, pp. 25-35.

[106] The TILE-Gx™ Processor Family, *Tilera*, 2009, http://www.tilera.com/products/processors.php.

[107] A. K. R. Toomu, "Pipelined Implementation of JPEG Image Compression Using VHDL," *M.Sc. Thesis, University of Nevada, Las Vegas*, 2008.

[108] J. Upadhyay, V. Varavithya, P. Mohapatra, "A Traffic-Balanced Adaptive Wormhole Routing Scheme for Two-Dimensional Meshes," *IEEE Transactions on Computers*, vol. 46, no. 2, 1997, pp. 190-197.

[109] L. G. Valiant, G. J. Brebner, "Universal Schemes for Parallel Communication," *Proceedings of the13th Annual ACM Symposium on Theory of Computing*, 1981, pp. 263-277.

[110] S. Vangal *et al.*, "An 80-Tile 1.28 TFLOPS Network-on-Chip in 65nm CMOS," *IEEE International Solid-State Circuits Conference (ISSCC 2007)*, 2007, pp. 98-589.

[111] D. Wiklund, "Development and Performance Evaluation of Networks on Chip," *Ph.D. Thesis, Linkoping University*, 2005.

[112] W. Wolf, A. A. Jerraya, "Multiprocessor systems-on-chips," *Morgan Kaufmann*, 2005.

[113] P. Wolkotte, G. Smit, J. Becker, "Energy Efficient NoC for Best Effort Communication", *Proceedings of the 15th International Conference on Field Programmable Logic and Applications*, 2005, pp. 197-202.

[114] P. T. Wolkotte, "Exploration within the Network-on-Chip Paradigm," *Ph.D. Thesis, University of Twente*, 2009.

[115] P. T. Wolkotte, G. J. M. Smit, N. Kavaldjiev, J. E. Becker, and J. Becker, "Energy Model of Networks-on-Chip and a Bus," *Proceedings of the 2005 International Symposium on System-on-Chip*, 2005, pp. 82-85.

[116] T. Ye, L. Benini, and G. De Micheli, "Analysis of Power Consumption on Switch Fabrics in Network Routers," *Proceedings of the 39th Annual Design Automation Conference*, 2002, pp. 524-529.

[117] B. S. Yoo, C. R. Das, "A Fast and Efficient Processor Allocation Scheme for Mesh-Connected Multicomputers," *IEEE Transaction on Computers*, vol. 51, no. 1, 2002, pp. 46-60.

[118] B. S. Yoo, C. R. Das, "A Fast and Efficient Processor Management Scheme for k-ary n-cubes," *Journal of Parallel and Distributed Computing*, vol. 55, no. 2, 1998, pp. 192-214.

[119] B. S. Yoo, C. R. Das, C. Yu, "Processor Management Techniques for Mesh-Connected Multiprocessors," *Proceedings of the International Conference on Parallel Processing (ICPP (2))*, 1995, pp. 105-112.

[120] S. - M. Yoo, H. Y. Youn, "An Efficient Task Allocation Scheme for Two-Dimensional Mesh-Connected Systems," *IEEE Transaction on Parallel and Distributed Systems*, vol. 8, no. 9, 1997, pp. 934-942.

[121] L. Zhang, "Comments on "A Fast and Efficient Processor Allocation Scheme for Mesh-Connected Multicomputers"," *IEEE Transaction on Computers*, vol. 52, no. 2, 2003, pp. 255-256.

[122] Y. Zhu, "Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers," *Journal of Parallel and Distributed Computing*, vol. 16, no. 4, 1992, pp. 328-337.

[123] H. Zimmermann, "OSI Reference Model – the ISO Model of Architecture for Open Systems Interconnection," *IEEE Transactions on Communications*, vol. 28, no. 4, 1980, pp. 425-432.

[124] D. Zydek, I. Pozniak-Koszalka, "Algorithms and Experimentation System of Unicast, Multicast and Broadcast Transmission for Optical Switches," *Proceedings of Eighteenth International Conference on Systems Engineering (ICSE 2006)*, IEE, IEEE, I MECH E., 2006, pp. 547-551.

[125] D. Zydek, H. Selvaraj, "Fast and Efficient Processor Allocation Algorithm for Torus-Based Chip Multiprocessors," *Journal of Computers & Electrical Engineering*, ISSN 0045-7906, submitted October 2009.

[126] D. Zydek, H. Selvaraj, "Hardware Implementation of Processor Allocation Schemes for Mesh-Based Chip Multiprocessors," *Journal of Microprocessors and Microsystems*, ISSN 0141-9331, vol. 34, no. 1, 2010, pp. 39-48.

[127] D. Zydek, H. Selvaraj, “Processor Allocation Problem for NoC-based Chip Multiprocessors,” *Proceedings of 6th International Conference on Information Technology: New Generations (ITNG 2009)*, 2009, pp. 96-101.

[128] D. Zydek, H. Selvaraj, L. Gewali, “Memory Utilization of Processor Allocator for NoC-based Chip Multiprocessors with Mesh Topology,” *Proceedings of Twentieth International Conference on Systems Engineering (ICSE 2009)*, 2009, pp. 497-502.

[129] D. Zydek, N. Shlayan, E. Regentova, H. Selvaraj, “Review of Packet Switching Technologies for Future NoC,” *Proceedings of 19th International Conference on Systems Engineering (ICSEng 2008)*, 2008, pp. 306-311.

VITA

Graduate College
University of Nevada, Las Vegas

Dawid Maksymilian Zydek

Degrees:

Postgraduate Certificate, Systems & Control, 2005
Coventry University, UK

Master of Science, Computer Science, 2005
Wrocław University of Technology, Poland

Special Honors and Awards:

Graduate and Professional Student Association Department's Representative, University of Nevada, Las Vegas, 2009

Recipient of Bally Technologies Scholarship, University of Nevada, Las Vegas, 2009

Recipient of Bally Technologies Scholarship, University of Nevada, Las Vegas, 2008

Member of Tau Beta Pi - The USA Engineering Honor Society, University of Nevada, Las Vegas, 2008

Best Paper Award, ICN 2007, Wrocław University of Technology, 2007

Honorable Member of Student Science Association SISK, Wrocław University of Technology, 2006

Chancellor's Award for the Best Graduate of Faculty of Electronics at Wrocław University of Technology (ranked among 921 graduated that year), 2005

Dean's Award for Special Achievements in Growth of Faculty of Electronics at Wrocław University of Technology, 2004

Best Paper and Presentation Award, 2nd Student Scientific Conference, Wrocław University of Technology, 2004

Dean's Award for Special Achievements in Growth of Faculty of Electronics at Wrocław University of Technology, 2003

Dean's Award for Special Achievements during Studies at Faculty of Electronics at Wrocław University of Technology, 2003

Refereed Journals Articles:

D. Zydek, H. Selvaraj, "Hardware Implementation of Processor Allocation Schemes for Mesh-Based Chip Multiprocessors," *Journal of Microprocessors and Microsystems*, ISSN 0141-9331, vol. 34, no. 1, 2010, pp. 39-48.

D. Zydek, H. Selvaraj, "Fast and Efficient Processor Allocation Algorithm for Torus-Based Chip Multiprocessors," *Journal of Computers & Electrical Engineering*, ISSN 0045-7906, submitted October 2009.

D. Zydek, I. Pozniak-Koszalka, L. Koszalka, K. J. Burnham, "Algorithms to Managing Unicast, Multicast and Broadcast Transmission for Optical Switches," *Lecture Notes in Computer Science*, vol. 5297, Springer Verlag, 2008, pp. 21-30.

Articles in Refereed Conference Proceedings:

D. Zydek, H. Selvaraj, L. Gewali, "Synthesis of Processor Allocator for Torus-Based Chip MultiProcessors," *Proceedings of 7th International Conference on Information Technology: New Generations (ITNG 2010)*, IEEE Computer Society Press, 2010, accepted February 2010.

D. Zydek, H. Selvaraj, L. Gewali, "Memory Utilization of Processor Allocator for NoC-based Chip Multiprocessors with Mesh Topology," *Proceedings of Twentieth International Conference on Systems Engineering (ICSE 2009)*, IEEE, I MECH E., 2009, pp. 497-502.

D. Zydek, H. Selvaraj, "Processor Allocation Problem for NoC-based Chip Multiprocessors," *Proceedings of 6th International Conference on Information Technology: New Generations (ITNG 2009)*, IEEE Computer Society Press, 2009, pp. 96-101.

D. Zydek, N. Shlayan, E. Regentova, H. Selvaraj, "Review of Packet Switching Technologies for Future NoC," *Proceedings of Nineteenth International Conference on Systems Engineering (ICSEng 2008)*, IEEE Computer Society Press, 2008, pp. 306-311.

I. Poźniak-Koszałka, D. Zydek, “Algorithms of Unicast, Multicast and Broadcast Transmission for Optical Switches,” *Proceedings of Sixth International Conference on Networking (ICN 2007)*, Eds: Cosmin Dini, Los Alamitos: IEEE Computer Society Press, 2007, pp. 95-101.

D. Zydek, I. Poźniak-Koszałka, “Algorithms and Experimentation System of Unicast, Multicast and Broadcast Transmission for Optical Switches,” *Proceedings of Eighteenth International Conference on Systems Engineering (ICSE 2006)*, IEE, IEEE, I MECH E., 2006, pp. 547-551.

D. Zydek, I. Poźniak-Koszałka, “Comparison and Simulation Research of Algorithms Applied to Optical Switching,” *In Computer Systems Engineering, Theory & Applications, Fifth Polish-British Workshop*, IEE Control and Automation Professional Network, 2006, pp. 180-192.

D. Zydek, I. Poźniak-Koszałka, “Unicast, Multicast and Broadcast Traffic Modeling for Optical Switching Technologies,” *Proceedings of Modeling and Simulation of Systems Conference (MOSIS 2006)*, Ostrava: MARQ, Acta MOSIS, no.107, 2006, pp. 216-222.

G. Chmaj, D. Zydek, L. Koszalka, “Comparison of Task Allocation Algorithms for Mesh-Structured Systems,” *In Computer Systems Engineering, Theory & Applications, Fourth Polish-British Workshop*, IEE Control and Automation Professional Network, 2004, pp. 39-50.

D. Zydek, I. Poźniak-Koszałka, “Architecture and Algorithms Used to Emulate Optical Switching,” *In Computer Systems Engineering, Theory & Applications, Fourth Polish-British Workshop*, IEE Control and Automation Professional Network, 2004, pp. 197-206.

Dissertation Title:

Processor Allocator for Chip Multiprocessors

Dissertation Examination Committee:

Chairperson, Henry Selvaraj, Ph.D.
Committee Member, Emma Regentova, Ph.D.
Committee Member, Rama Venkat, Ph.D.
Committee Member, Mei Yang, Ph.D.
Graduate Faculty Representative, Laxmi Gewali, Ph.D.