Bo Yang

Towards Optimal Application Mapping for Energy-Efficient Many-Core Platforms

Turku Centre *for* Computer Science

TUCS Dissertations
No 167, December 2013

Towards Optimal Application Mapping for Energy-Efficient Many-Core Platforms

Bo Yang

To be presented, with the permission of the Faculty of Mathematics and Natural Science of the University of Turku, for public criticism in Auditorium Beta on December 9th, 2013, at 12 noon.

University of Turku
Department of Information Technology
20014 Turku, Finland

2013

Supervisors

Associate Processor Juha Plosila
Department of Information Technology
University of Turku
20014 Turku, Finland

Senior Researcher Tero Säntti
BID Technology, University of Turku
20014 Turku, Finland

Senior Researcher Liang Guang
Department of Information Technology
University of Turku
20014 Turku, Finland

Reviewers

Professor Timo D. Hämäläinen
Department of Pervasive Systems
Tampere University of Technology
Korkeakoulunkatu 1, FIN-33720 Tampere
Finland

Associate Professor Zhonghai Lu
Department of Electronic Systems
Royal Institute of Technology (KTH)
Isafjordsgatan 39, Kista, 164 40, Stockholm
Sweden

Opponent

Associate Professor Jesús Barba Romero
School of Computer Science
University of Castilla-La Mancha
Edificio Fermín Caballero
13071 Ciudad Real, Spain

ISBN 978-952-12-2982-4
ISSN 1239-1883

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin OriginalityCheck service.

Abstract

As massively parallel computing on many-core platforms has become the state-of-the-art, application mapping is widely researched as an important step to minimize the energy consumption and improve the system performance. Conventional algorithms are prohibitively expensive for the application mapping problem, if even possible. To find the optimal application mapping, the frameworks and algorithms which optimize the allocation of large amount of computation and communication resources on many-core platforms, have to be innovated. This thesis proposes systematic frameworks for single-/multi-application mapping, and innovates on conventional and evolutionary algorithms for energy minimization on many-core platforms.

Firstly, an application mapping framework which integrates both computation and communication resource allocations is proposed. In this three-stage framework, IP selection aims to minimize the energy consumed for the computation of tasks. Tile assignment and communication mapping focus on the minimization of the energy consumed for the inter-core communication. In addition to investigating and formulating the individual operation of each individual stage, the framework emphasizes on the interaction of the three stages. With each individual stage addressing a particular resource allocation problem, the collective effort of the three stages results in an overall optimized mapping from the system's point of view.

For the three stages in the framework, tree-model and simulated annealing (SA) based algorithms are proposed in this thesis. By utilizing the tree-model of a network-on-chip (NoC) in the mapping, the tree-model based algorithm is time- and energy-efficient. To improve the optimality of the mapping solutions, two SA-based algorithms are developed for application mapping. Compared to the generic SA algorithm, the first one, the parameter-optimized SA (POSA) algorithm, utilizes a set of optimized parameters which speeds up the convergence to the global optimal mappings. Furthermore, the t_k-SA algorithm starts the annealing from an already optimized initial solution, with the appropriate initial temperature. The quantitative evaluations show that both SA-based algorithms need significantly fewer iterations to converge to the optimal mappings, without loss

of mapping quality.

While the previous framework and algorithms address the mapping problem of a single application, a novel framework for mapping multiple applications adaptively with unbounded or bounded number of cores is presented. Consisting of two steps, application mapping and task mapping, the proposed mapping method finds an area on a many-core NoC for each application and then maps all tasks of the application into the area. A weighted node average distance (WNAD) based algorithm is proposed for application mapping. The proposed t_k-SA algorithm is applied to the task mapping in the case of unbounded number of cores. A task-sequence based (TSB) algorithm is proposed for the task mapping in the case of bounded number of cores. The quantitative comparisons show that the WNAD+t_k-SA mapping algorithms achieves the lowest communication energy consumption in all combinations of application mapping and task mapping algorithms evaluated.

Acknowledgements

First and foremost, I would like to express my sincerest gratitude to my supervisors, Prof. Juha Plosila and Dr Tero Säntti. Prof. Juha Plosila has supported me throughout my doctoral study, which allows me to explore promising research topics freely and flexibly. With his solid knowledge and high-level guidance, I have been able to tackle challenging issues and make progresses in my research. Dr Tero Säntti gave me valuable instructions on how to carry out high quality research as soon as I began my study. I am grateful to his comments and criticism on my research works and the thesis.

I owe a very important debt to my best friend and my youngest supervisor, Dr Liang Guang. He has inspired and encouraged me in the whole course of my PhD program. We have worked together on many research tasks and he has contributed so much to my research. In addition, Dr Liang Guang has provided invaluable help and effort on finalizing and refining my thesis. He is a nice fella that I can always learn from, not only in the academic, by also in the real life.

I also wish to thank external experts for reviewing and being opponent for my thesis defence: Prof. xxx, Prof. Timo D. Hämäläinen and Prof. Zhonghai Lu. Thanks to their detailed reviews and constructive comments, the thesis has been modified and improved.

I would like to thank my colleagues, Thomas Canhao, Xu and Alexander Wei, Yin. We have cooperated in several research issues. I have been motivated by their professionals in corresponding fields. Besides, I am grateful for the generous help and friendliness from my colleagues, Pasi Liljeberg, Teijo Lehtonen, Rajeev Kumar Kanth and Khalid Latif.

I would like to acknowledge the financial and academic support from Academy of Finland and Turku Center for Computer Science (TUCS).

Last but not least, I would like to express my deepest appreciation to my wife and daughters. With their endless love and support, I have never lost my direction and courage to go forward in my life. They are also the source of my pleasure and happiness which help me to break down the technical barriers in my research, as well as the physical difficulties in our life.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ALU	Arithmetic Logic Unit
APU	Accelerated Processing Unit
BB	Branch-and-Bound
BM	Bounded Mapping
CCG	Core Communication Graph
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DSM	Distributed Shared Memory
DVFS	Dynamic Voltage and Frequency Scaling
EA	Evolutionary Algorithm
FLOPS	Floating Point Operation per Second
GA	Genetic Algorithm
GI	Greedy Incremental
GPU	Graphics Processing Unit
ILP	Integer Linear Programming
IP	Intellectual Property
IPR	IP Repository
ISA	Instruction Set Arxhitecture
LP	Linear Programming
MER	Maximal Empty Rectangle

MIC	Many Integrated Cores
MIN	Multistage Interconnection Network
MIPS	Microprocessors with Interlocked Pipeline Stages
MPI	Message Passing Interface
NAD	Nodes Average Distance
NoC	Network-on-Chip
NTG	Network Topology Graph
POSA	Parameter-Optimized SA
SA	Simulated Annealing
SCC	Single-Chip Cloud Computer
SMT	Symmetric Multithreading
TG	Task Graph
TSB	Task-Sequence Based
UM	Unbounded Mapping
WCA	Weighted Communication of an Application
WNAD	Weighted NAD

Chapter 1

Introduction

1.1 Parallel Computing on Many-Core Processors

1.1.1 Limitations of Single-Core Architecture

Since the first generation of microprocessors was invented in 1970s, the computer architects and the chip manufacturers have strived to provide new microprocessors with higher performance and extended functionality. Historically, for the processors with single central processing unit (CPU), called single-core processors or uniprocessors, there have been two major approaches to increase the computation capacity of the processors [1]. One is pipelining of individual instructions. In pipelining, the execution of an instruction is decomposed into stages, e.g., instruction fetch, instruction decode, operand fetch, execute and store result. The basic idea of pipelining is to keep all stages occupied by executing multiple instructions in a pipelined way. Although the execution time of one instruction is not decreased, the throughput of the processor is increased. The decomposed small stages also allow a faster clock rate of the processor, which again contributes to a higher throughput and is an obvious sign of increased performance to normal customers. Another approach is the adoption of superscalar architecture. Instead of using only one arithmetic logic unit (ALU), multiple ALUs are implemented in the superscalar processors. Sets of instructions of applications are checked by the compiler and independent instructions are executed simultaneously on the group of parallel ALUs, often out-of-order with respect to the original sequence. The approach is enabled by the increased number of transistors on chip provided by new technologies according to Moore's law [4]. With these approaches, we have witnessed an exponential performance improvement of single-core processors for about two decades, as shown in Figure 1.1.

However, the gains of the pipelining and superscalar architecture have reached their limits since about 2005. There are two outstanding reasons

which limit the continuous development of pipelining and superscalar architecture in a single-core processor. The first reason is that there is an upper bound for the amount of instruction level parallelism which can be exploited in an application. This limitation results in diminishing returns of applying pipelining and superscalar techniques, which are dependent on the instruction level parallelism of applications [1]. The second reason is the power/energy concern which has been imposed by the increased density of transistors on the processors [5]. As the single-core processors were run at ever higher clock rates, the power required grew at a faster rate than the frequency. In addition, when the technology scaled down, the leakage power could not be ignored anymore due to its increased portion in the total power consumption. Reported in [6], the leakage power has accounted for more than 50 percent of the total power consumption starting from the 65 nm technology. The power/energy situation becomes even worse for the complex single-core processors with deeper pipelining and broader issue widths, since larger amounts of transistors were needed for implementing the additional logic. The power/energy issue directly resulted in serious heat dissipation problem and constrained the performance growth of conventional single-core processors, as the cooling technologies did not keep up with the increasing power and energy consumption. The above limitations have prevented the processor manufactures from delivering ever higher performance with single-core processor architectures.

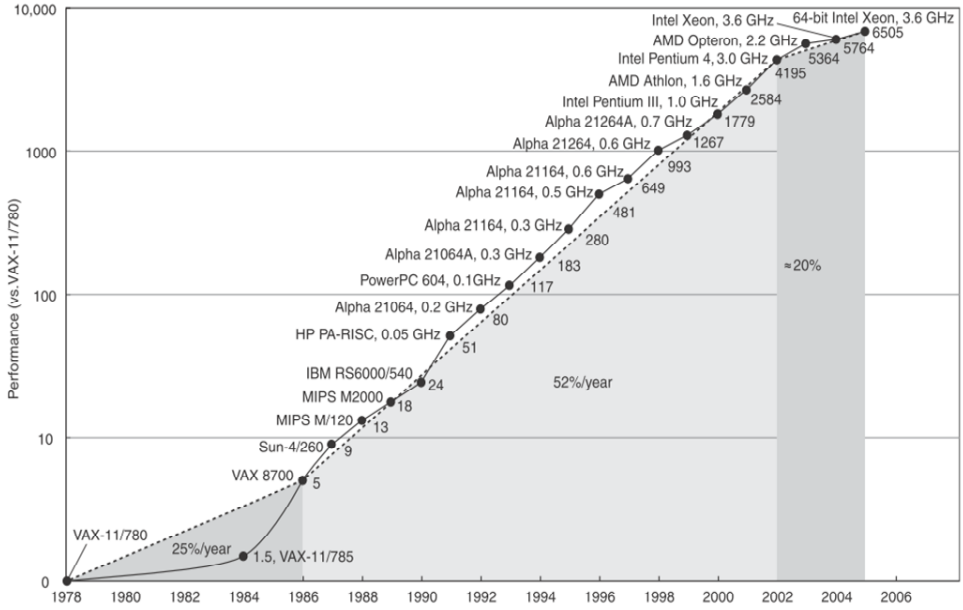


Figure 1.1: Performance Growth of Generations of Processors since 1978 [1]

1.1.2 Emergence of Many-Core Processors

To overcome the limitations of conventional single-core architectures and to fulfill the requirement of increasing computation capability from end users, multi-core processor architectures have been proposed by major chip vendors as a promising alternative to the single-core architectures. As the name implies, a multi-core processor integrates usually a moderate number of processing units, referred to as *cores*, into a single processor. All cores are organized according to a specific topology and connected through an interconnection medium such as bus, crossbar switch, or network-on-chip (NoC) [7]. The cores communicate with each other through the interconnections for data transfer, as well as for synchronization and coordination control (see Chapter 2). Figure 1.2 shows the architecture of the Intel Core i7 processor which can have up to 8 four-issue, out-of-order, two-way symmetric multithreading (SMT) cores. A three-level cache hierarchy is used for data exchange between cores and memories. With multi-core processors, the set of tasks can be distributed to multiple cores and processed in parallel. With a lighter workload, each core has not to be as complex and powerful as its counterpart in the single-core processors. Hence, each core does not have to run at the clock rate of the single-core processors, which significantly decreases the power consumption. As demonstrated in [8], a decrease of 20% in the clock frequency can reduce the dynamic power consumption by 50%. Compared to a single-core processor in which only the instruction level parallelism is supported, explicit parallelism like thread-level and data-level parallelism, can be efficiently exploited and employed by multi-core processors. These two major features of multi-core processors alleviate the limited instruction level parallelism and increased power concerns in single-core architectures. Moreover, compared to single-core processors, multi-core architectures provide flexibility of dynamically adjusting the supply voltage, clock frequency and workload on individual cores in order to optimize the system performance and reduce power/energy consumption [9]. Multi-core processors are the inevitable evolution of conventional uniprocessors and are taking over the world of computers.

The mainstream multi-core processors on the market are general-purpose multi-core processors, for example, Intel Nehalem and Core i7, AMD Phenom, IBM POWER7, Oracle SPARC T3 and ARM Cortex A15 ([2] [10] [11]). These processors target at a wide variety of applications which are usually found in desktops and servers. To reach the objective, a relatively small number of rather complex and powerful cores are employed. The cores in such a general-purpose multi-core processor typically inherit features that have been well developed in previous generations of uniprocessors, such as standard instruction set architecture (ISA) (e.g., CISC x86 ISA in Intel processors), wide-issue and out-of-order execution, multithreading and mul-

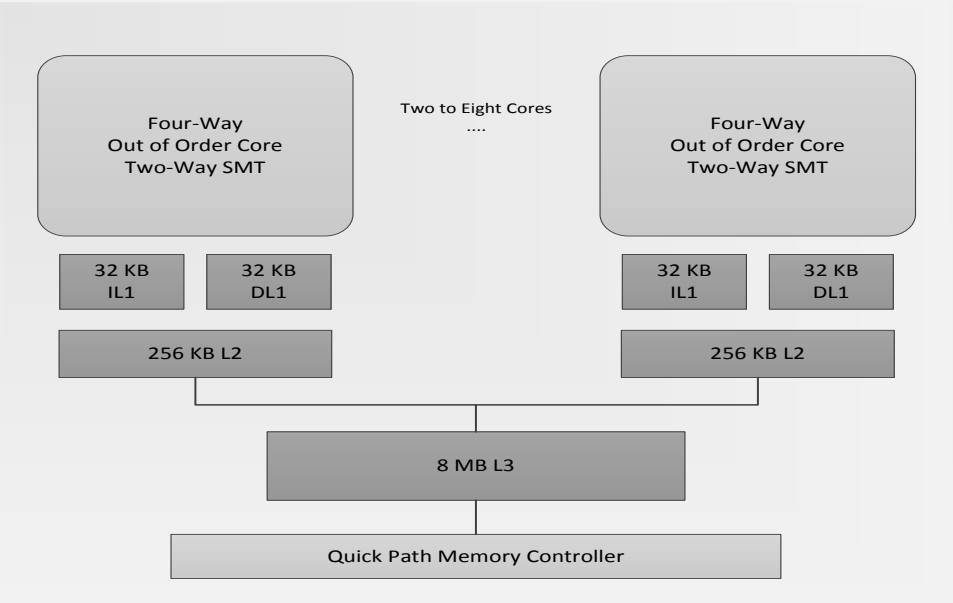


Figure 1.2: The Architecture of Intel Core i7 Processor [2]

tilevel cache hierarchy. The replication of this kind of general-purpose cores not only increases the computing capability of the multi-core processors, but also simplifies task and thread management.

Based on the same architectural innovation, an alternative parallel processor design approach is to integrate a larger number of smaller and simpler cores for more fine-grained parallel applications. This trend results in the so-called many-core processors, in which usually tens to thousands cores are employed. In contrast with cores in the multi-core processors, the cores in many-core processors are less complicated, but more specified for particular application domains like graphics processing and scientific computing. The examples of many-core processors include 80-tile Intel TeraFLOPS [12], Tilera's Tile GX family [13] and NVIDIA graphics processing units (GPUs) (up to 3072 cores in Nvidia's Fermi GPUs) [14]. As the future of high-performance processors, many-core processors have drawn great attention for the major chip manufactures. The ongoing projects of developing more powerful many-core processors include, for instance, single-chip cloud computer (SCC) and many integrated cores (MIC) architecture in Intel, microprocessors with interlocked pipeline stages (MIPS) in SiCortex, and CSX700 in ClearSpeed [11].

With respect to the heterogeneity of processing cores, the multi-core or many-core processors can be classified into homogeneous and heterogeneous processors. A processor consisting of a number of identical cores is termed as *homogeneous* processor. Conversely, the processor integrating various types

of cores, either in complexity or functionality, is called a *heterogeneous* processor. A heterogeneous processor could be a hybrid of general-purpose cores and high-performance cores like the accelerated processing units (APUs) of AMD Fusion [15] and Intel Sand Bridge[16] which integrate a multi-core CPU and a GPU on the same chip. A heterogeneous processor can contain processing elements such as DSPs, reconfigurable FPGA blocks and other intellectual property (IP) cores. For the sake of generality, in this thesis, we refer to the processing elements in a heterogeneous multi- or many-core processor as IP cores. With the heterogeneity of IP cores, heterogeneous processors are more capable of improving system throughput, reducing power consumption and exploiting parallel computing than homogeneous processors [17] [18].

1.1.3 Massively Parallel Computing

Although it was introduced already in the middle of 1960s, parallel computing did not become so popular before the emergence of multi- and many-core processors. Parallel computing used to be too expensive and difficult to use for normal organizations and end users, and was mainly limited to scientific computing in research units and governments. Nowadays, a desktop computer with a multi-core processor can be seen as a parallel computing system. More powerful parallel computing systems targeting more complicated and difficult applications, or massively parallel computing, have become the norm of the present computing systems. Massively parallel computing has been employed in a variety of fields including complicated simulations, energy analysis, computer-aided design and computer-aided manufacturing, graphics processing, life science, finance, and data mining. Data-intensity is the most significant feature of applications in these fields. Consequently, the computation for solving such problems is time- and energy-consuming. Take the climate simulation presented in [19] as an example. Using a high-resolution 3D model for partitioning the globe, it needs 10^{20} floating point operations to complete the climate simulation over a certain weather cycle. For such amount of operations, it will take 31 centuries for a single-core processor, at a rate of 10^9 floating point operation per second (FLOPS), to complete the simulation. Clearly, it is impractical to use a single-core processor to do jobs as heavy as the climate simulation. Massively parallel computing based on the multi- and many-core processors is a more efficient way to solve such kind of extremely data-intensive problems.

The implementation of massively parallel computing is a two-sided problem [19]. One side is the algorithms which represent the specific applications to be realized. The other side is the parallel computing platforms which provide the processing resources to the application, and can be computers with multi- or many-core processors, or cluster(s) of computers. To run an ap-

plication on a parallel computing platform, i.e., to move from the algorithm side to the platform side, a systematic design flow should be followed. In this flow, the issues including algorithm analysis, algorithm parallelization, parallel programming, communication identification, scheduling and mapping, have to be dealt with.

1.2 Communication-Centric Parallel Computing

To run an application in a parallel paradigm, the application which was originally represented by a sequential program, has to be implemented by a team of tasks which can be distributed and executed simultaneously on different cores. To generate the same outputs as the original sequential program, the concurrent executions of all tasks need to be coordinated so that the dependencies among them can be fulfilled and they can work in the correct order. The dependence between tasks can be control dependency or data dependency, or more precisely, classified into flow dependency, antidependency and output dependency [20]. If one task has dependency on one or more tasks, we say that the task is dependent on those tasks. Flow dependency happens when one task produces output that is used by another task. To meet this kind of data dependency, the resulted output (data) has to be transferred from the first task to the second task. Antidependence occurs when one task uses a variable as its input which should be written by another task later. An earlier write operation of the second task will make the first task produce a wrong output. A wrong output could also be generated while two tasks write to the same variable out of order. This scenario comes from the violation of the output dependency. To deal with the antidependency and output dependency, synchronization control information has to be exchanged among tasks or between tasks and the operation system. The transfer of data and the exchange of synchronization information introduce another important issue, communication, in the parallel computing systems on multi-core or many-core platforms. There are two typical methods for dealing with the communication problem on a multi- or many-core platform: shared variables or message passing. The method applicable to a specific platform is dependent on the memory structure of the platform. The communication methods will be discussed in more detail in Chapter 2.

The communication has great impact, not only on the overall performance of a parallel computing system, but also the power/energy consumption. A task cannot start its operation until the tasks that it is dependent on have already completed their executions and, for those with data dependency (or flow dependency), the resulted data have been received. A delay on the starting time of a task due to dependencies will in turn postpone the execution of tasks which are dependent on this task and increase the overall

execution time of an application. For some latency-sensitive applications, like game or streaming applications, the delay can become unacceptable. In addition, in modern technology, the energy consumption of communication is significant compared to that of computation. For example, transferring 32-bit data across a 10 mm chip consumes 17 pJ in 50 nm CMOS, while a 32-bit ALU operation only consumes 0.3 pJ. The great impact on performance and power/energy consumption makes communication analysis and optimization integral parts of a parallel computing system design. In other words, the parallel computing design on many-core platforms has become communication-centric, instead of computation-centric as it was in the uniprocessor era [21] [11].

Based on the communication-centric design paradigm, a general design flow of implementing parallel computing of an application on multi-core or many-core platforms is presented in Figure 1.3. The design starts from a sequential program which has been used for representing the application. The target platform, on which the parallelized program will be executed, is also given at the beginning. The first stage of the design flow is *algorithm design*. It begins with analyzing the sequential program to find which parts of the program can be run in parallel. Based on the analysis, the original program is decomposed into a set of program fragments, each of them is described by sequential code and called a *task*. The dependencies between tasks are also analyzed and represented by, for example, a directed acyclic graph (DAG) [22]. The size of a task (in terms of the number of instructions) is called *granularity*. The tasks with a large size have a coarse-grained granularity, while the tasks with a small size are fine-grained. The terms granularity, coarse and fine are relative in the description of an application. The granularity of tasks determines the amount of computation to complete the task, as well as the intra-task communication. More fine-grained tasks can increase the degree of parallelism, but also results in a larger amount of intra-task communication which implies more scheduling and mapping overhead. Decomposition of application into tasks needs to find a good compromise between the number of tasks and their granularity. The decomposition is also determined by the inherent parallelism property of an application.

The set of decomposed tasks is then programmed using an appropriate programming language. In the *parallel programming* stage, the features of the target platform have to be taken into account. This is because the multi- and many-core platforms from different vendors might be built on different parallel architectures and memory organizations (see Section 2). Thus, they might only support particular communication methods and programming languages. For example, for the many-core processors with distributed memory, the message passing method has to be used for inter-core data transfer, synchronization and coordination. In this case, the language-

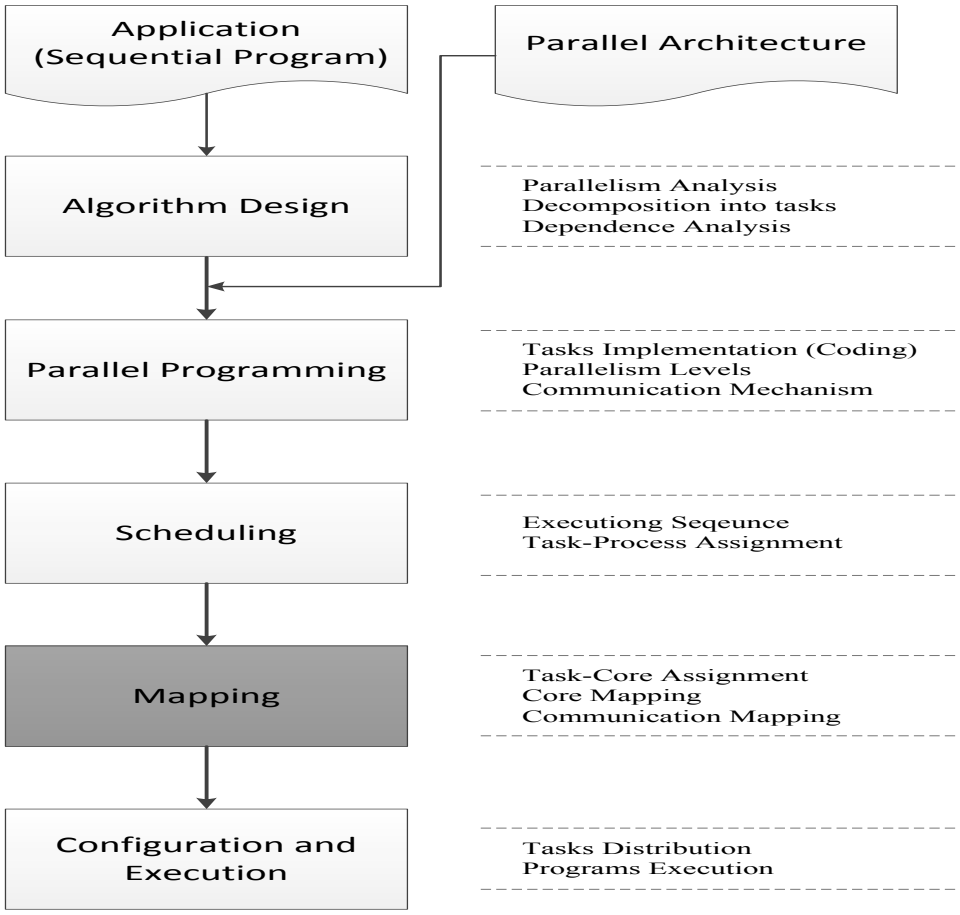


Figure 1.3: The Design Process of Parallel Computing on Many-Core Platforms

independent communication protocol, MPI (message passing interface), is currently the dominant model for programming. After an application has been programmed as a group of parallel programs, the communication volumes between the communicating tasks (programs) can be obtained by static or dynamic program analysis/profling, as introduced in [23] [24]. The communication volume information can be added into the dependency graph (e.g., a DAG) generated in the previous algorithm design stage. The updated dependency graph is an important representation of the application, which will be used in the following scheduling and mapping stages for producing an optimized design.

Figure 1.4 is an illustration of the design process presented here with the parallel programming stage omitted. The programs of tasks need to be assigned to processes or threads which can be run on a processor. This

assignment is called *scheduling*. The scheduling also produces the execution sequence of the given set of tasks to make sure that the antidependency and output dependency constraints are fulfilled. If the number of tasks is larger than that of available processes or threads, multiple tasks have to be assigned to one same process or thread. In this case, scheduling can take the granularity of each task into account and find a load-balanced scheduling solution.

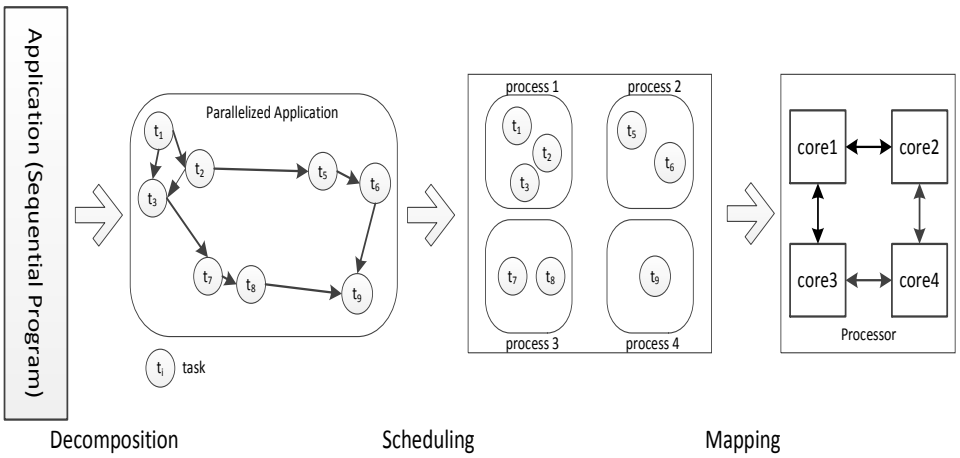


Figure 1.4: An Illustration of The Design Process

With the task-process assignment solution obtained in the scheduling stage, the *mapping* (or application mapping) stage maps processes or threads onto physical cores on a platform. Similarly to scheduling, multiple processes or threads can be mapped onto one same core if the number of processes or threads exceeds that of cores. Inter-task communication is the major issue which is taken care in the mapping stage. The typical objectives of mapping an application on a many-core platform include minimizing the communication power/energy consumption, decreasing the bandwidth requirement and reducing the contention and congestion situation on the communication networks. In addition, the workload on cores can be balanced by the application mapping as well.

After the optimized scheduling and mapping solutions have been found, the last stage of the design is to load the set of parallel programs to their mapped cores on the target multi- or many-core platform. Based on the scheduling carried out by the operating system, the set of tasks execute concurrently and complete the jobs of the application.

1.3 Application Mapping

This thesis focuses on the application mapping stage. Basically, the application mapping is a resource allocation problem. It decides how to allocate processing cores (computation resources) to tasks, and how to distribute inter-task communications on the interconnection network (communication resources) [25]. The solution of computation resource allocation determines on which core a task will be executed, which in turn determines the execution time and energy consumption of executing the task. The time and energy consumption of computation might vary significantly over different task-core mapping solutions, especially on heterogeneous many-core platforms. In addition, assuming that the cores are connected by a network architecture, e.g., a NoC, the task-core mapping also determines how far (in terms of hops) the data has to be transferred between each pair of communicating tasks. More hops result in higher communication energy consumption. Regarding the communication resource allocation, the distribution of inter-task communication on the interconnection network determines the contention and congestion situations in the network. High contention and congestion in the network will delay the communication and degrade the performance of the running applications. High congestion in the network also increases the communication energy consumption since data has to be kept longer in the communication network.

The application mapping is a resource allocation problem between a group of cores (resources) and a set of tasks (jobs), consisting of computation and communication resource allocation. The purpose of the application mapping is to obtain an optimized solution which can improve the system performance and minimize the cost such as energy/power consumption. This objective can be achieved by individual optimization of either computation resource allocation or communication resource allocation, or by the combination of both. However, this kind of resource allocation problem is categorized as the *combinational optimization* problem which consists of finding an optimal solution from a finite set of candidate solutions. It is an instance of the constrained quadratic assignment problem and is known as being NP-hard [26].

To solve such an important but difficult problem, extensive works have been carried out for developing efficient mapping algorithms [27, 28, 29, 30, 31, 32]. Based on the study of the existing works, we observed that:

1. Minimizing the communication power/energy consumption is one of the major objectives that have been addressed in existing works. This is due to, on one hand, an ever larger number of transistors on processor chips, which has resulted in increasing power/energy concerns. On the other hand, a tighter power/energy consumption constraint is

imposed by modern computers or hand-held devices because of cooling problems and/or battery capacity limitations, conflicting with the fact that more power and energy are required by increasingly complex parallel applications.

2. Performance and complexity of a mapping algorithm have to be traded off according to the pre-defined objectives and constraints imposed by hardware and software. More complex algorithms, e.g., genetic algorithm [32] or linear programming [29], can find near-optimal or optimal mapping solutions, but the required computation effort is huge or prohibitively expensive for large-size problems. On the contrary, simpler algorithms like greedy heuristic search [28], can find a solution with comparably less computation and runtime, but the solution could not be guaranteed to be globally optimal or near-optimal. Algorithms with better trade-off between performance and complexity need to be developed.

3. The majority of the existing works do individual optimization on either computation resource allocation, or communication resource allocation. A systematic method which combines the optimization process on both issues has been hardly investigated.

4. Most existing works aim to solve the single-application mapping problem in which only one application is mapped onto a many-core platform. As future many-core processors will provide abundant processing cores, the focus should be shifted from single-application mapping to multi-application mapping where multiple applications are mapped and executed on a many-core platform. This introduces another level of parallelism, the application-level parallelism.

The first observation shows that power/energy consumption should be the major concern when we develop new application mapping algorithms. According to the second observation, the mapping algorithms to be developed should ensure the optimality of the mapping solutions while keeping the complexity of the algorithm as low as possible. The third and fourth observations point out the potential fields in which we should explore to pursue high-performance and innovative application mapping methods for parallel computing on future many-core platforms.

1.4 Thesis Objectives

Motivated by aforementioned observations, this thesis targets: (1) identifying the general mapping process which incorporates both computation and communication resource allocation for finding an overall optimal mapping

solution; (2) developing novel mapping algorithms applied to computation and communication resource allocation problems. More precisely, the objectives of this thesis include:

1. To build a general application mapping framework for heterogeneous multi-core or many-core platforms, in which the specific mapping stages for computation and communication resource allocation are identified and integrated in a systematic way, in order to produce globally optimal mappings. The formulation of each stage and the cooperation mechanism among them will be investigated.

2. To develop mapping algorithms which are used for each specific mapping stage. The objective of the algorithms is to optimize the specific design interests in terms of performance or cost metrics (e.g., energy consumption, bandwidth requirement and execution time). As the exhaustive search and linear programming methods are too time-consuming, or even prohibitively expensive for large-size mapping problems, algorithms based on search heuristics (e.g., simulated annealing and genetic algorithm) will be particularly investigated. How to trade off the complexity and performance is the major consideration in developing new mapping algorithms.

3. To establish a methodology for mapping multiple applications onto many-core platforms. The objective of the multi-application mapping algorithms is to minimize the resource competition between applications, while keeping the design metrics of an individual application as good as those achieved by single-application mapping.

1.5 Thesis Contributions

To reach the objectives, extensive investigations of the application mapping problem on a heterogeneous many-core platform have been conducted. Based on the investigations, a generic framework of application mapping is presented. Three high-performance algorithms are developed with the purpose of improving the solution quality while decreasing the algorithm complexity. Utilizing the algorithm developed, a methodology of mapping multiple applications is proposed. The main contributions of this thesis are:

1. A three-stage process of application mapping is introduced for heterogeneous many-core platforms, which is composed of three consecutive stages, i.e., IP selection, tile assignment and communication mapping. The three-stage process provides a framework of how to solve the computation and communication resource allocation problems in a systematic way. IP selection selects one type of IP core available on the target

platform for each task. Tile assignment decides which tile (or node) is assigned for the execution of a task, provided that the type of the IP core on this tile is the same that was selected for the task in the IP selection stage. When the tile assignment has been done, communication mapping allocates one routing path on the communication network for each inter-tile communication. The functions of the three stages are formulated respectively.

2. Six categories of mapping algorithms which are broadly used in the three sub-stages of application mapping, including exhaustive search, branch-and-bound, greedy incremental search, simulated annealing, genetic algorithm, and integer linear programming, are analyzed and compared in terms of search space size, solution quality, complexity and problem scope. The study of the algorithms provides insights into the complexity and performance of each type of algorithm. Based on the study, we were inspired to put effort on developing optimized simulated annealing algorithm for high-performance application mapping, due to its better trade-off between solution quality and time complexity.

3. Three algorithms are proposed for application mapping, with different complexity and performance trade-offs. The first one is a time-efficient task-core mapping algorithm which utilizes a tree-model of a NoC to speed up the mapping process. Two simulated annealing (SA) based algorithms are proposed for the purpose of finding the globally optimal mapping solutions. To reduce the time complexity of the generic SA algorithm, the Nelder-Mead simplex method is applied in one algorithm to generate the optimized parameters for the SA algorithm. In another algorithm, called t_k-SA, the generic SA algorithm is accelerated by starting the annealing process from a lower initial temperature t_k together with an optimized initial mapping solution. The experiments show that both algorithms reduce the random partition of the generic SA algorithm and speed up the convergence to the final mapping solutions, without loss of mapping quality. The algorithms were initially published in my previous publications [33] [34] [35].

4. A novel methodology for mapping multiple applications adaptively with unbounded or bounded number of cores is presented. Composed of application mapping and task mapping, the proposed two-step mapping methodology maps multiple applications on different regions in a many-core NoC so that each application has no or negligible interference with other applications. The proposed t_k-SA algorithm is applied to the task mapping in the case of unbounded number of cores. The task-sequence based (TSB) algorithm is proposed for the task mapping

in the case of bounded number of cores. Without the limitations of the algorithms applied for application mapping and task mapping, the innovative mapping methodology provides a framework for mapping multiple applications on many-core platforms. The proposed methodology was partially presented in my previous publications [36] [37] [38].

1.6 List of Publications

The work presented in the thesis is based on the following publications:

1. Bo Yang, Liang Guang, Tero Säntti and Juha Plosila, Survey of Application Mapping Methods for Networked Many-Core Systems. Scalable Computing (Springer), accepted for publication.

2. Bo Yang, Liang Guang, Tero Säntti and Juha Plosila, Mapping Multiple Applications with Unbounded and Bounded Number of Cores on Many-Core Networks-on-Chip. Microprocessors and Microsystems (Elsevier), Volume 37, Pages 460-471, 2013. [38]

3. Bo Yang, Liang Guang, Tero Säntti and Juha Plosila, t_k-SA: Accelerated Simulated Annealing Algorithm for Application Mapping on Networks-on-Chip. Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference (GECCO'2012), 1191-1198, ACM, 2012. [35]

4. Bo Yang, Liang Guang, Tero Säntti and Juha Plosila, Parameter-Optimized Simulated Annealing for Application Mapping on Networks-on-Chip. Learning and Intelligent Optimization, LNCS, 307-322, Springer Berlin Heidelberg, 2012. [34]

5. Bo Yang, Liang Guang, Thomas Canhao Xu, Alexander Wei Yin, Tero Säntti and Juha Plosila, Multi-Application Multi-step Mapping Method for Many-Core Network-on-Chips. Proceeding of NORCHIP (2010), 1-6, IEEE, 2010. [36]

6. Bo Yang, Liang Guang, Thomas Canhao Xu, Tero Säntti and Juha Plosila, Multi-application mapping algorithm for Network-on-Chip platforms. Proceeding of IEEE 26th Convention of Electrical and Electronics Engineers in Israel (IEEEI), 540-544, IEEE, 2010. [37]

7. Bo Yang, Liang Guang, Tero Säntti and Juha Plosila, Tree-model based mapping for energy-efficient and low-latency Network-on-Chip. Proceeding of Design and Diagnostics of Electronic Circuits and Systems (DDECS), 189-192, IEEE, 2010. [33]

1.7 Organization of Thesis

The thesis is composed of 8 chapters. Chapter 2 discusses the typical energy-efficient design methods for many-core platforms. Existing works addressing the application mapping problem are reviewed as well. A three-stage application mapping process is presented in Chapter 3, including the formulation of each stage, and the comparison between the set of representative algorithms. Chapter 4 proposes an energy- and latency-efficient algorithm for mapping an application onto a 2D mesh NoC. A tree-model of a NoC is introduced and employed in the application mapping. Two accelerated simulated annealing algorithms are proposed for application mapping in Chapters 5 and 6. In Chapter 5, a set of parameters utilized in the SA algorithm is produced by the proposed method which significantly speeds up the SA algorithm. Alternatively, an optimized initial mapping and corresponding temperature are applied in Chapter 6 to accelerate the generic SA algorithm. Chapter 7 presents a methodology for mapping multiple application simultaneously on a many-core platform. The proposed adaptive methodology addresses mapping with both unbounded and bounded number of cores. Chapter 8 concludes the thesis and discusses envisioned future work for application mapping on many-core platforms.

Chapter 2

Energy-Efficient Many-Core Platforms

Energy-efficiency has become a primary concern for massively parallel computing on many-core platforms [39]. On one hand, although the energy/power consumption of individual cores can be decreased with technology scaling, the aggregate energy consumption and power density scale up when the number of integrated cores increases. Many-core platforms are energy-hungry due to the large amount of concurrent computations and communications [40] [41]. According to the prediction from ITRS [42], the power consumption of consumer devices will grow from today's 100 W to more than 200 W in 2015. On the other hand, the energy/power budget is becoming tighter and tighter due to increasing heat dissipation concerns. As demonstrated in [43], a modern multi-core mobile phone is required to support 100 $GOPS$ (10^{11} operations per second) with 1 W power budget. Hence, energy-efficient design on many-core platforms has drawn significant attention in literature. This chapter discusses the typical parallel architectures of many-core platforms. The emphasis is put on the impacts that the architectures and interconnection networks impose on the energy consumption in a many-core platform. Then, the typical energy-efficient design methods are presented. Of them, application mapping is one method which provides great potential to achieve energy-efficiency at the system level.

2.1 Parallel Architectures of Many-Core Platform

To explore the potential of massively parallel computing with multiple or many IP cores, various forms of parallel architectures have been developed. A parallel architecture is characterized by the nature and the number of IP cores, the arrangement of IP cores and memory modules, the nature and types of interconnection among IP cores and memory modules, the com-

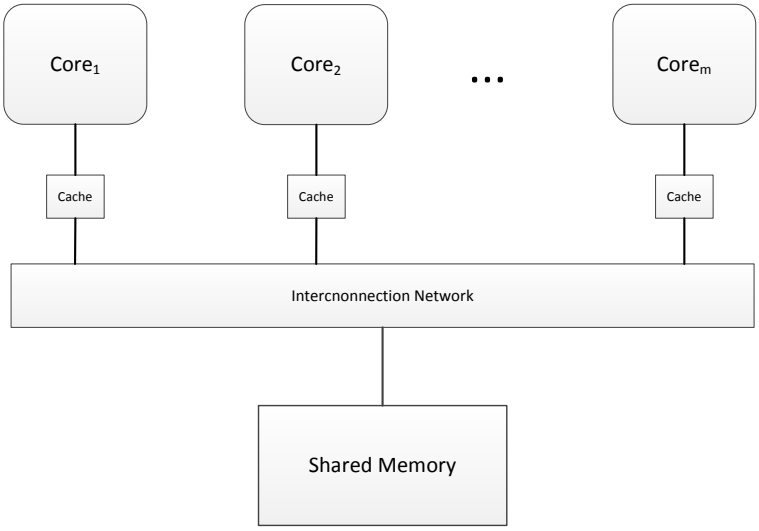
munication mechanism among IP cores and between IP cores and memory modules, and the applicable programming model [44]. Due to these characteristics, parallel architectures can be classified from different perspectives. In this section, we are going to introduce two typical parallel architectures which are classified based on the communication mechanism and programming model. They are *shared address space* and *message passing* architectures.

2.1.1 Shared Address Space

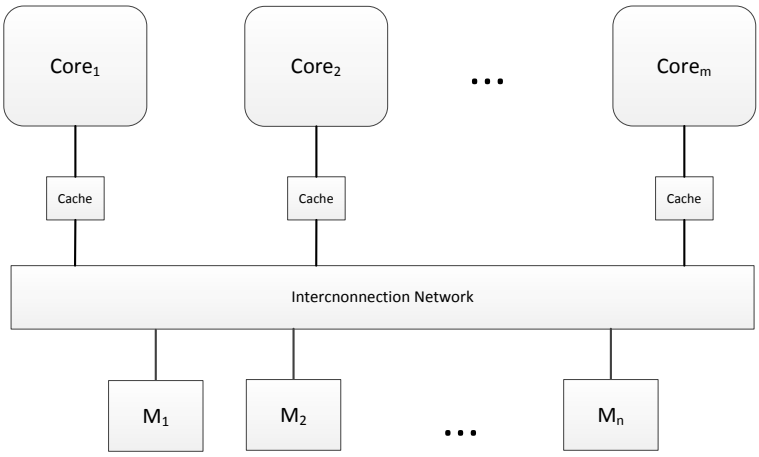
As the name implies, in a shared address space many-core platform, the memory space is accessible to all IP cores. Figure 2.1 shows the general abstraction of a shared address space architecture [45]. The shared memory can be one physical memory module (global memory) as shown in Figure 2.1a, or be composed of a set of memory modules as shown in Figure 2.1b. In the latter case, the whole address space is distributed into the set of memory modules. This architecture is also called a *distributed shared memory (DSM)* system. In the shared address space architecture, each core can address and access any location in the shared memory. A result of one IP core is written to the shared memory and read later by another IP core. Inter-core communication is realized by writing and reading shared variables which are stored in the shared memory. To alleviate the impact of slow memory access, a memory hierarchy is usually used in modern multi- or many-core processors [46]. Several levels of faster and relatively smaller caches are employed between the IP cores and the main memory. Frequently accessed data or instructions are pre-fetched and stored in caches and accessed directly by the IP cores at runtime. An access to the main memory occurs only when the required data has not been found in any of the cache levels.

Most modern desktop and server multiprocessors employ the shared address space architecture. Recalling Intel Core i7 shown in Figure 1.2 (Section 1.1.2) which uses the Intel's Nehalem microarchitecture, each core has private L1 and L2 caches, and a L3 cache shared with other cores. All cores are connected through a ring and access the local system memory through the integrated memory controller [47]. Similar architectures are applied to the latest AMD Opteron 6000 series [48], Oracle's Sparc T3 [49] and IBM Power 7 [50].

The main advantage of the shared address space architecture is that it facilitates parallelization of legacy sequential programs where all variables are defined in the same address space. From the programmers' point of view, the same set of variables is accessible to multiple IP cores in a parallel program as was accessible to a single core in the sequential implementation of the program. Therefore, programs require a minimum of restructuring for parallel computing on a many-core platform. Programming for the shared



(a) Implementation with One Physical Memory



(b) Implementation with Memory Modules

Figure 2.1: Shared Address Space Architecture

address space architecture is also easier to understand due to the similarity of operating systems programming and general multiprogramming. The programming language can be an extension of existing languages with directives or libraries supporting parallel operations. For example, OpenMP API (application program interface) provides a collection of compiler directives, library routines and environmental variables for the programming of shared memory systems. The compiler directives can be used for Fortran, C and C++ for tasking constructs, work-sharing constructs and synchronization constructs [51].

There are several challenges in designing systems based on the shared

address space architecture. Firstly, from the programming perspective, since race conditions occur when multiple threads or tasks operate on the same part of a program or variables (called *critical section*) simultaneously, keeping these operations in order is critical to ensure the correctness of the program. This leads to a complex synchronization problem [44] [52]. Explicit synchronization has to be implemented in programs manually by programmers. From the hardware perspective, although the utilization of caches efficiently hide the memory access latency, it introduces the cache coherence problem in a shared address space system [53] [45]. With multiple copies of a data item in multiple caches, a complex cache coherence protocol has to be employed to keep the multiple copies consistent whenever one of them is modified [44] [1] [47]. The protocol overhead per core increases with the number of cores, leading to a "coherence wall" [54]. Another issue of the shared address space architecture is the scalability. This is due to the limitation of the coherence wall and the performance bottleneck caused by the shared memory in general. In addition, the address space of the share memory cannot scale up infinitely.

2.1.2 Message Passing

Message passing is another type of communication mechanism and programming model which are usually employed in distributed memory architectures. As shown in Figure 2.2, in a distributed memory architecture, a private memory is allocated to each core for storing local program and data. The local memory can be accessed by the local IP core directly. When an IP core needs data in a remote memory, message passing is performed for transferring the required data from a private memory to another. To alleviate the long memory access latency, similar to the shared address space architecture, caches are also used between the IP cores and their local memory modules. Compared to implicit inter-core communication by reading or writing shared variables in shared address space architectures, communication in message passing architectures is realized by explicit *send* and *receive* operations. The send operation specifies a process of sending data in the local memory of an IP core to a remote memory accessed by another IP core. Correspondingly, the receive operation specifies a process of fetching data from a remote memory to the local memory. Depending on whether the send and the receive operations need to be synchronized for completing a data transfer, message passing can be classified as synchronous and asynchronous message passing [44]. In the synchronous paradigm, the data transfer can only be performed after a handshaking coordination between the source and destination cores. In the asynchronous paradigm, the data transfer, especially for a send operation, can start immediately without coordination with the destination core.

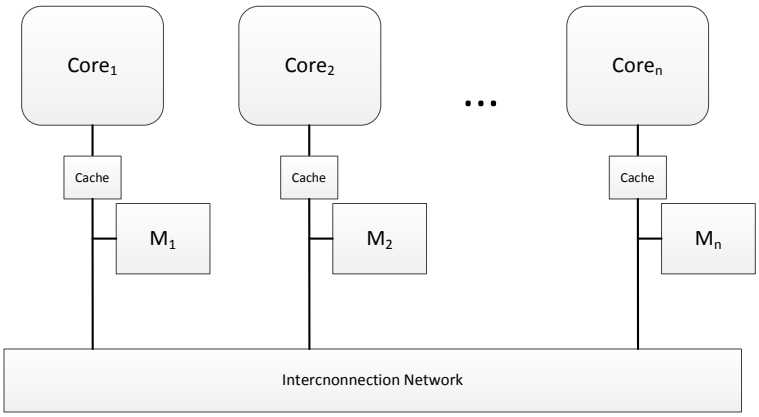


Figure 2.2: Message Passing Architecture with Distributed Memory Modules

To accomplish message passing, appropriate communication protocols have to be employed in both synchronous and asynchronous paradigms. These protocols can be performed by the IP cores. In this case, the computation on an IP core has to be blocked due to a send or receive operation (blocking send or receive). The blocking send and receive operations waste the computation resources on the platform. Therefore, the trend in modern message passing architectures is to decouple the communication operation from the computation operation. There are two typical approaches for the decomposition. One is to add a memory controller to perform the data transfer between the local and remote memories [55]. An IP core can continue its computation after issuing a communication call to the memory controller. The memory controller performs the data transfer independently. Another approach is to implement dedicated routers in the interconnection network. The IP cores are connected to the routers through network interfaces. The routers are connected by a specific topology and form an interconnection network for inter-core communication. The data to be transferred is first injected into the network by the local router, and then forwarded to the destination core by a set of routers according to a specific routing algorithm. This approach leads to the promising network-on-chip (NoC) structure for many-core platforms, which will be discussed in Section 2.2. Utilizations of either memory controllers or routers enables overlapping of computation and communication and improves thereby system performance.

The message passing architecture has been widely used in classical parallel computing systems such as client/server systems and computer clusters. With the emergence of multi- and many-core processors, the message passing architecture has recently drawn great attention due to its better scalability [56] [57] [58]. Take the prototype processor presented in [57] as an example, 48 Pentium^{TM} IA-32 cores are distributed into 24 tiles which

are arranged in a 6×4 2D-mesh NoC with 2 cores per tile. A five-port router is embedded in each tile to connect with the 4 neighboring routers and one local core. The 48 cores communicate over the NoC using the message passing architecture, which removes the dependence on hardware maintained cache coherence while remaining in a constrained power budget. From the architectural point of view, message passing architectures provide better scalability due to the modular way in which the IP cores and memory modules are organized, facilitating integration of hundreds to thousands of IP cores in future many-core platforms. From the programming point of view, the explicit send and receive operations used for inter-core communication eliminate the complex synchronization and coordination problem presented in the shared address space architectures. The cooperative send and receive operations have implicitly a built-in synchronization feature. In addition, message passing programming is easier for validation and more portable than shared memory programming, which makes it more efficient for programming on multi- and many-core platforms [54]. The most popular communication library for message passing architectures is MPI (message passing interface) which provides a communication interface to , for example, Fortran, C and C++ [59].

2.2 Interconnection Network

2.2.1 Network Characteristics

In both aforementioned parallel architectures, interconnection network is a key component for providing efficient and reliable communication between IP cores and memory modules (as shown in Figure 2.1 and 2.2). An interconnection network is characterized mainly by *bandwidth*, *latency*, *concurrency*, *diameter*, *connectivity* and *scalability* [52].

Bandwidth: Bandwidth is usually specified by the bisection bandwidth of an interconnection network, in terms of the maximum number of bytes per second that the network can transport across the section through which the network is partitioned into two equal halves.

Latency: Latency is the time consumed from the transmission of data into the network to its reception at its destination. The latency for transferring one message is usually composed of software overhead, channel delay, switching or routing delay, and contention time.

Concurrency: Concurrency of a network is measured by the number of connections which can be supported concurrently in the network. Concurrency is an important characteristic for a parallel architecture. A larger concurrency leads to higher throughput and smaller latency

since a larger number of inter-core communication events can perform simultaneously.

Diameter: Diameter is the longest path between any two nodes on the network. It is usually quantified by the hops between nodes at the farthest ends of the network. The diameter is used for estimating the longest transfer delay on the network.

Connectivity: Connectivity is quantified by the node or edge degree, which is the number of nodes or links that need to fail to disconnect the network. A higher connectivity implies better reliability and fault-tolerance.

Scalability: Scalability is the potential that a network can expand in a modular fashion. The performances of systems based on the network increase approximately at the same rate as the number of IP cores scales up.

The above characteristics of an interconnection network have great impact on the performance and energy-consumption of a parallel architecture. They are determined by the network topology, routing and switching techniques that are employed. For each of them, there are different design options with various characteristics. The following sections present typical design methods for interconnection networks. For more detail on interconnection networks, see [53] and [45].

2.2.2 Network Topology

The topology of an interconnection network defines how the IP cores and memory modules on a parallel platform are connected with each other. Interconnection networks are distinguished as direct and indirect networks [60]. In direct networks, each node is a terminal node which can be an IP core, a memory module or a combination of them, as shown in Figure 2.1 and 2.2. A terminal node acts as a source or destination node for packets. Packets are forwarded directly between terminal nodes. The connection cannot be changed during the execution time. In indirect networks, a node could be a terminal node or a switch which forwards packets from input ports to output ports. In contrast to direct networks, the packets between source and destination terminal nodes are forwarded indirectly by means of switch nodes. The connection between nodes can be dynamically configured to meet the changing communication requirements of applications.

Direct Networks

The typical direct networks include mesh, ring, tree and cube network. The topologies of them are shown in Figure 2.3. The fully-connected network is

shown here as an extreme case of direct networks, in which each terminal node is directly connected to all other terminal nodes. The linear array is a one-dimensional mesh which connects N nodes using $N-1$ links on a line. The two-dimensional mesh is constructed similarly as the linear array on both their x and y axes. In a 2D mesh, a node is usually denoted by its x and y coordinates on the horizontal and vertical direction respectively. The 2D mesh is the most popular network topology because of its regularity, scalability and routing simplicity. A 2D torus is a variant of the 2D mesh. The difference between a torus and a mesh is that in the torus the boundary nodes are also directly connected both row-wise and column-wise. A ring topology is obtained by connecting the two end nodes in a linear array. A tree topology connects nodes in a parent-child way. Figure 2.3e shows a binary tree topology in which each interior node has one parent node and two child nodes. $k-ary\ n-cube$ is the most general form of the cube networks. Meshes, rings and hypercube networks are topologically isomorphic to the $k-ary\ n-cube$ networks.

Each kind of topology has particular characteristics, as well as different performance and cost. A trade-off between topologies has to be made in the design by taking these aspects of each type of topology into account, with respect to the requirements of target applications. As reported in [52], given a constant number of nodes, higher dimension networks have better scalability and lower latency, but they utilize more wires, pins and larger switches. This leads to higher cost in several aspects, including heat dissipation, yield, testing, etc., compared to low-dimensional networks [61]. Hence, the low-dimensional networks, such as 2D meshes, are favored due to their facilitated implementation, simplicity of the routing strategy and network scalability.

Indirect Networks

The direct networks presented above are suitable for applications which can exploit regular, fixed topologies. In contrast, the indirect interconnection networks perform better for applications with variable and irregular communication patterns. The connection between two terminal nodes can be dynamically established by reconfiguring the intermediate switches. This provides great flexibility to meet various communication patterns from different applications. There are some significant disadvantages for dynamic networks as well, including low concurrency and scalability. The representative indirect networks are buses, crossbar networks and multistage networks, as shown in Figure 2.4.

Bus is the simplest indirect network. In the bus scheme, all terminal nodes are connected by a shared set of wires. A crossbar network provides a full connection between N nodes. As shown in Figure 2.4b, the total

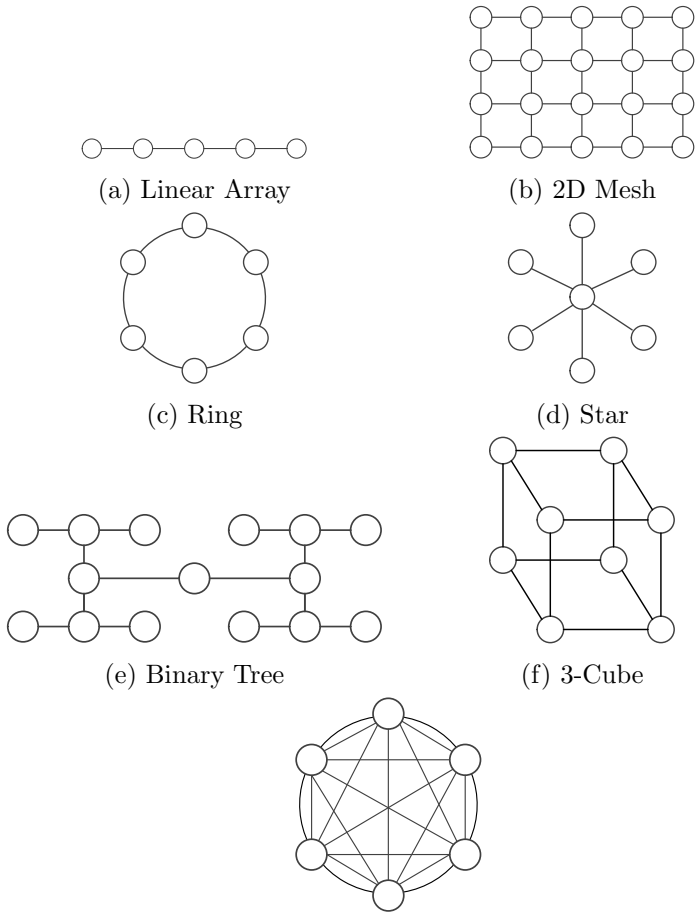
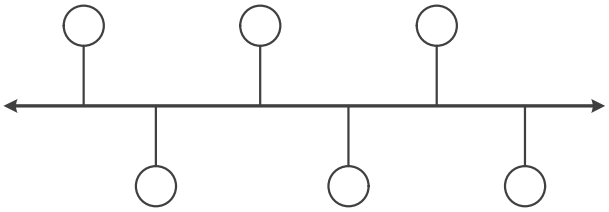


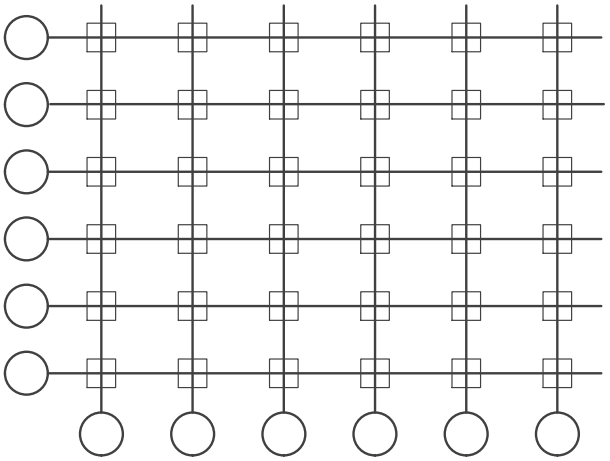
Figure 2.3: Direct Interconnection Networks

number of N^2 switches are required to set up the connection between any two nodes on the network. Multistage interconnection networks (MINs) make use the 2×2 or larger crossbar switches as the building blocks to develop fully connected network between nodes. A general MIN consists of multiple stages each of which consists of a set of crossbar switches. A MIN is a compromise between a low-performance but simple bus and a high-performance but costly crossbar network.

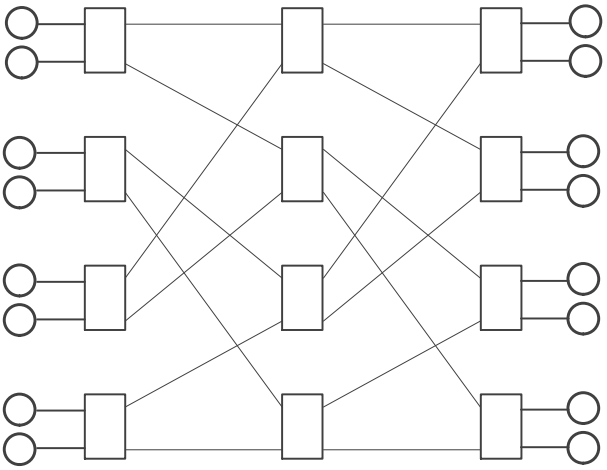
The buses are efficient and simple only for connecting a small number of IP cores. The lack of parallelism inhibits the scalability of buses for large size systems. In contrast, crossbar networks allow all permutations of connections in constant time and support more concurrent communication, but the cost of switches required is excessive for large size systems. Multistage interconnection networks are compromise solutions between low



(a) Bus Network



(b) Crossbar Network



(c) Multistage Omega Network

Figure 2.4: Dynamic Interconnection Networks

cost/performance buses and high cost/performance crossbar networks. Since less switches are utilized, a possible problem of multistage interconnection networks is the occurrence of hot spots in the network.

2.2.3 Routing and Switching

Direct and indirect networks provide communication paths between nodes. The message sent from a source node has to traverse a sequence of intermediate terminal nodes or switches before it finally reaches its destination node. To perform inter-core transactions, specific *routing* and *switching* techniques have to be utilized to regulate and coordinate the communication events on the network.

Routing Algorithms

In certain kind of interconnection networks, such as meshes, there are multiple paths and routes from one node to another node. A routing algorithm determines on which path the message from node A to node B should be transferred. The purpose of a routing algorithm is to guide the transfer at each router on the path so that the message can finally reach its destination. A routing algorithm also needs to handle the problems of deadlock and livelock. The typical routing algorithms include the dimension-order routing, source-based routing, table-driven routing and turn model routing [45].

There are two major classification schemes used for characterizing the routing algorithms. Based on the length of the selected routing path, the routing algorithms are classified into *minimal* and *non-minimal* routing algorithms. A minimal routing algorithm always chooses a routing path with the shortest possible length between two nodes. One example of the minimal routing algorithm is the dimension-order XY routing algorithm. On the other hand, a non-minimal routing algorithm may not always select the shortest routing path with the purpose of, for example, avoiding the congestion on the network. The routing algorithms can also be categorized as *deterministic* and *adaptive* routing. With deterministic algorithms, the routing path is deterministic and unique for a pair of communicating nodes. The path selection is only depending on the source and destination nodes, without taking other transmissions in the network into consideration. The deterministic algorithms are simple, but they can lead to unbalanced load and more contentions in the network. In contrast, adaptive routing algorithms can select one routing path from several alternative paths according to the traffic situation in the network. As a result, the communication traffic in the network can be balanced and the contention situation is alleviated by the adaptive routing algorithms. They also provide better fault-tolerance than the deterministic algorithms. Naturally, the adaptive algorithms are more complex than the deterministic algorithms. Based on whether the minimal-path strategy is applied or not, the dynamic algorithms are further classified into minimal and non-minimal adaptive algorithms. Examples of deterministic routing algorithms are dimension-order XY routing

and source-based routing. Turn model routing algorithms are examples of dynamic routing [62] [63].

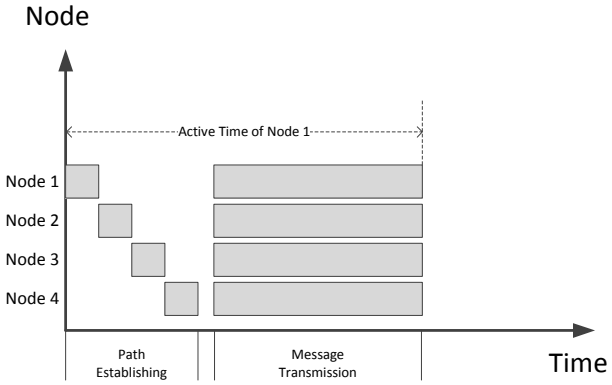
The problems very often encountered in routing are deadlock and livelock. Deadlock occurs when two data transmissions hold a resource that is required by both for the next step operation. Deadlock might happen whenever several transmissions have cyclic dependence on resources. Livelock is the situation in which a message keeps moving around the network but can never reach its destination. It is usually the result of using an adaptive routing algorithm so that a message is not guaranteed to reach its destination deterministically. Turn model routing algorithms such as west-first and north-last routing algorithms are used to avoid deadlock [62]. The deterministic dimension-order XY routing is free of livelock.

Switching Techniques

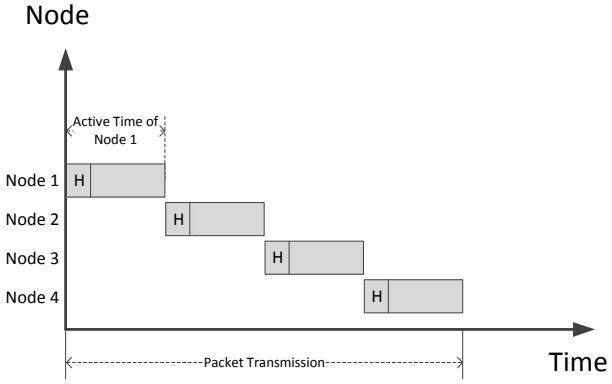
The switch strategy determines how a message is forwarded at a switch or a router along a rouging path between a source and a destination node. More particularly, the switching determines how to move data from an input channel to an output channel at a switch or a router. Circuit switching and packet switching are the typical strategies used in modern interconnection networks [60]. The different switching techniques are illustrated in Figure 2.5

In circuit switching (Figure 2.5a), the routing path from a source node to a destination node is established and exclusively occupied by the message transmission between the nodes. The routing path is released when the whole transmission is completed. A message is transferred on the path in terms of pieces of data which are called *physical units (phits)*. The routing path is established by using short probe messages traveling along the path, and is released by a message trailer or by an acknowledgement message from the receiver. The circuit switching can provide high speed and large bandwidth for message transmission, but it decreases the utilization efficiency of the physical links in the network, due to the exclusive occupation of the whole routing path during a message transmission.

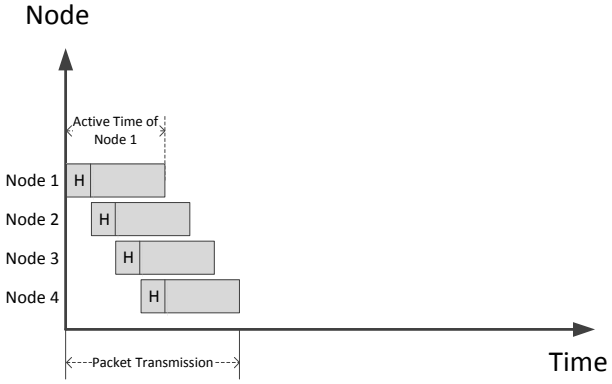
In packet switching, the transmission of a message is performed at the packet granularity. The message to be transferred is split into a sequence of packets which can be transferred independently to the destination node. To do so, a packet is generally composed of the header, the data and the trailer sections. The routing and control information are contained in the header. Each packet is sent separately to the destination node according to the routing information in the header. The conventional technique used in packet switching is the store-and-forward (Figure 2.5b). In the store-and-forward switching, an entire packet has to be received at a switch or a router before it can be forwarded to next switch or router. Compared to the circuit



(a) Circuit Switching



(b) Store-and-Forward Packet Switching



(c) Cut-Through Packet Switching

Figure 2.5: Illustration of Circuit and Packet Switching

switching, packet switching improves the utilization rate of links due to the adoption of the shorter packets.

If the sequence of packets of one message is transmitted along the same routing path, the pipelining technique can be used so that the set of packets

can use the links on the routing path in an overlapping way. The total transmission time of the whole message is therefore decreased. The pipelining techniques can be further used in the transmission of an individual packet, which leads to the cut-through switching and the wormhole switching [60] [21]. In cut-through switching (Figure 2.5c), the routing path of a packet is established by checking the phits of the packet header. Then, the rest of the packet will be transferred in a pipelining way on the routing path. A channel allocated to a packet can be freed as soon as the trailer of the packet has been transmitted through the channel. In the wormhole routing, the packet is organized as a sequence of *flow control units (flits)*, which can be as large as the packet header. Similar to the cut-through switching, the header flit is used to establish the routing path and the rest flits are transferred along the path in a pipelining way. The major difference from the cut-through switching is, if the transmission of a packet is blocked on a switch, only few flits are stored in the switch. The rest of the flits are stored in preceding switches on the path. Consequently, smaller buffers are required in switches or routers. The disadvantage of the wormhole switching is that a blocked packet transfer will occupy the buffer space of switches or routers on the path. Various latencies of the different packet switching algorithms are illustrated in Figure 2.5, where the transmission of one packet, from node 1 to node 4, is demonstrated with each aforementioned algorithm.

2.2.4 Network-on-Chip

Nework-on-chip has been proposed as a evolutionary architecture to integrate very large number of IP cores since last decade [64] [7] [65]. The NoC-based chip multiprocessors (CMPs) have been developed both in academia and industry [65] [57] [66] [65]. The development of NoC architecture has been driven by the growing demand on parallel computing and increasing popularity of many-core processors. Due to the poor scalability, the interconnection networks such as rings, buses, crossbars and MINs are not performing as efficiently on many-core processors as they are on multi-core processors. NoCs were proposed as an alternative for these conventional interconnection networks and targets to the systems consisting of tens to thousands of IP cores and memory modules. Instead of using the dedicated wires, the numerous IP cores and memory modules in a NoC are connected to a on-chip network which routes packets between them. Compared to their predecessors, NoC architectures have several advantages [67] [64]. The modularity is the key advantage of NoCs which supports scalability from the ground up, especially in terms of performance. The structured networks are more predictable and provide more potential to explore the power/performance design space and more flexibility to support specific applications. The distributed nature of NoC infrastructures also improve the

reliability of the systems due to the modular organization and the utilization of segmented connections, so that a local fault cannot break the whole network.

Structure of NoC

Figure 2.6 shows an example of a 4×4 2D mesh NoC with 16 nodes. The interconnection network of the NoC is composed of a set of routers and point-to-point links connecting the routers. An IP core is connected to a router through the network interface (NI). The NI acts as a translator which packetizes the data to be transferred on the network, and unpackets the packets received from the network. On the 2D mesh network, a router is directly connected to up to four neighboring routers via the point-to-point links. The data is sent to and received from the network for a IP core by the local router. Each router also works as an intermediate transporter which forwards the incoming packets to one of its neighboring routers according to the specific routing algorithm. Packet switching is generally used in NoCs. Both deterministic and adaptive routing algorithms can be applied to a NoC. For 2D mesh NoCs, the dimension-order routing algorithms, for example, XY routing, is often used. With XY routing, a node on the 2D mesh NoC is referenced by its x and y coordinates. From a source node, a packet is transferred along one dimension at a time until it finally reach its destination node. For a packet transmission from node A (x_a, y_a) to node B (x_b, y_b), the minimal length of the routing path equals to $|x_b - x_a| + |y_b - y_a|$. NoC interconnection is applicable to both message passing and shared address space parallel architectures.

2D mesh NoC is one of the most popular NoC structures due to its regularity and modularity which in turn simplify the routing on the network and facilitate the implementation. 2D mesh NoC has been implemented in the Intel 80-core TeraScale processor [12] and the 48-core SCC processor [57], as well as the 64-core Tilera processor [65]. Extended from 2D implementations, 3D NoCs have recently been proposed, either by stacking multiple layers of 2D mesh NoCs [68], or by placing interconnection network and IP cores on separate layers [69]. Besides 2D mesh, other topologies like torus and hypercube can also be applied to NoCs.

Most significant characteristic of NoC architectures is that it decomposes the computation and the communication. Thanks to the utilization of routers, an IP core needs not to participate or to be blocked by a message transmission so that the computation and communication operation can overlap each other. A node consisting of an IP core and a router operates as a lightweight processor and can efficiently communicate with other nodes. Clearly, the distributed and modular structure of NoCs provides better scalability for constructing high-performance many-core platforms.

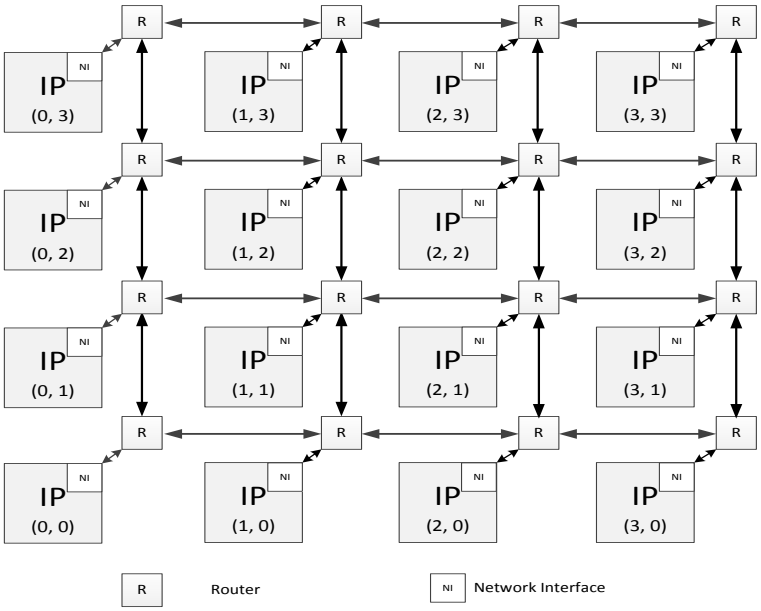


Figure 2.6: An Example of 4x4 2D Mesh NoC

Besides scalability, the NoC structure provides more opportunity and flexibility to explore energy-efficient design on many-core platforms.

2.3 Energy-Efficient Design Methods

2.3.1 Energy Consumption

In a many-core platform, energy is consumed for running the IP cores, memories, I/Os and interconnection network. Energy consumption is composed of the dynamic and static parts. Dynamic energy is consumed by the switching of capacitances, while static energy is consumed by subthreshold and gate leakage [70]. As long as the IP cores and the interconnection network are powered on, there is static energy consumption. Therefore, the static energy consumption is more platform-dependent, which is determined by factors like supply voltage, body bias voltage and temperature of the platform, as well as the active time of the IP cores and the interconnection network (Equation 2.1, derived from [71] [72]). Dynamic energy is consumed when the IP cores are processing data and the interconnection network is transmitting data. The dynamic energy consumed for switching transitions in a router or a link is proportional to the capacitance and the square of the supply voltage (Equation 2.2 [73]). Dynamic energy consumption is the direct result of the computation and the communication operations. Take a NoC-based system as an example, a sequence of packets has to be forwarded

through a set of routers along the routing path until the packets reach the destination node. During the transmission, energy is dissipated inside each router and on each link. As defined in Equation 2.3 [73], the total dynamic energy consumption for transmitting the sequence of packets is the sum of the router and link traversing energy over the routing path.

$$\begin{aligned} E_{sta} &= \int_{t=t_0}^{t=t_1} P_{sta}\,\mathrm{d}t \\ P_{sta} &= K_1 \times V_{DD} \times 10^{-K_2 \times V_{th} T} + |V_{bs} \times I_j| \end{aligned} \tag{2.1}$$

$$E_{dyn-tran} \propto C_L \times V_{DD}^2 \tag{2.2}$$

$$E_{dyn-NoC} = \sum^{allpackets} (E_{dyn-router} + E_{dyn-link}) \times H \tag{2.3}$$

In the above equations, V_{DD}, V_{th} and V_{bs} are the supply voltage the threshold and the bias voltage respectively. K_1 and K_2 are two experimentally determined parameters. T is the temperature. I_j is the junction leakage. H is the hop count of a packet traversing from its source node to its destination node. C_L is the load capacitance.

Based on the origins of static and dynamic energy consumption, and the design flexibility provided by the NoC architecture, a diversity of methods have be proposed to achieve energy-efficiency on a NoC-based many-core platform. To classify, these methods follow two main directions. One group of them focuses on how to optimize the platform-specific parameters in Equation 2.1 and 2.2 which affect the static and dynamic energy consumption at the architecture level. The methods in this group include, for instance, adaptive voltage and frequency scaling, power gating and clock gating. The other direction aims to optimizing the allocation of the IP cores and communication channels in order to minimize the energy consumed for data processing and transmission. An important and efficient method on this direction is the application mapping. As we can see from Equation 2.3, given the dynamic energy consumption of the routers and the links, $E_{dyn-router}$ and $E_{dyn-link}$ respectively, the dynamic energy consumption on a NoC can be reduced by minimizing the length of the routing path, H. The minimization of routing path can be achieved by the application mapping method.

The typical energy-efficient design methods employed on NoC-based platforms are explained in the following sections.

2.3.2 Adaptive Voltage and Frequency Scaling

Dynamic voltage and frequency scaling (DVFS) is a widely used technique for energy efficient design [74] [75] [76]. DVFS originates from the fact

that it is inefficient and unreasonable for all IP cores and links on a many-core platform to operate at a uniform supply/threshold voltage levels and a single frequency because the workloads on them may vary significantly over time. Various demands on processing and communication can be fulfilled with different voltage levels and frequencies. DVFS is enabled by the NoC architectures where each node can work in a globally asynchronous locally synchronous (GALS) way. DVFS is usually implemented by monitoring the run-time workloads on IP cores and links, based on which the appropriate voltage levels and frequencies are set. For example, in [74], the processorsŕuntime statistics and an online learning algorithm are utilized for determining the appropriate voltage level and frequency at any given point of time. It is reported that the maximum energy saving of 49% can be achieved. In [75], the utilization ratio of links is monitored as an indicator of the workload. When the utilization of a link is too high, a higher frequency can be set to reduce the workload. If there is low utilization of a link, the frequency can be decreased to save energy.

DVFS can be implemented at various granularities: chip-wide, region-wide and per-core [73]. Chip-wide DVFS adapts multiple voltage levels and frequencies to the whole platform corresponding to various workloads from different applications. Chip-wide DVFS is the basic form with minimal overhead and complexity. Since the workloads on different regions of a many-core platform have spatial locality, it is more reasonable to employ DVFS per region instead of per chip. One approach of region-wide DVFS is to deploy several voltage and frequency islands on the platform, each of them consisting of multiple IP cores supplied by the same voltage level and frequency [76]. The voltage level and frequency of each island is independent of the chip-level voltage level and frequency. An island can be allocated to an application with appropriate processing and communication requirements so that the energy consumption can be minimized. The most fine-grained DVFS is to utilize scalable voltage levels and frequency for each IP core. This design paradigm is adopted on a 176-core platform [66], in which the voltage level and frequency of each core can be adjusted by a local DVFS controller. The fine-grained DVFS can provide high performance and design flexibility, but with more area and timing overhead.

2.3.3 Power Gating

With technology scaling, the static power, which used to be minimal in earlier CMOS technologies, is becoming the dominating part of power consumption in sub-90 nm CMOS technologies [70]. Meanwhile, with the increasing number of IP cores on a many-core platform, it is more likely that some portion of the platform, including the IP cores and the interconnection network, might be idle temporally. This makes it possible to employ

the power gating techniques, whereby transistors are used to disconnect the power of the idle portion of the platform, to reduce the power and energy consumption.

Many techniques have been developed for power gating [77]. For NoC based platforms, most efforts have been put on the interconnection network because the buffers in the routers and the repeaters on the links are the major contributors of the static power [78]. For example, in [79], the proposed NoC consists of an always-on network and a number of configurable on/off channels. The always-on network ensures the basic connectivity and functionality of the interconnection network. There are much fewer channels in the always-on network than a normal NoC so that the static power is greatly reduced. The on/off channels can be dynamically powered on or off, depending on the local traffic load. In [80], the links will be powered off if there is no incoming packets in the next few cycles. The power gating is based on the prediction of future traffic load, which is performed by using information obtained from the look-ahead routing of packets in the neighboring routers.

2.3.4 Clock Gating

Another traditional technique for energy-efficient design is clock gating. In clock gating, the energy is saved by shutting off clock of the logic block of a device when there is no work to be done. Clock gating is an efficient way to reduce the power consumption on the clock net, which accounts for 20% to 40% of the total dynamic power consumption [81]. The clock gating can be employed at the system level as well as at the combinational and sequential logic levels [81].

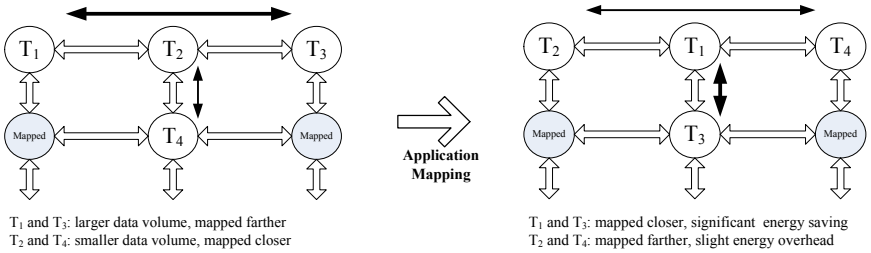
In [82], a two-level clock gating is proposed for NoCs. At first level, the fine-grained clock gating is applied to router components including input FIFOs and matrix arbiters. A the upper level, a whole router can be gated when it is inactive. The clock-gating of a router is determined at the runtime by signals which indicate if the router has any buffered data or blocked output virtual channels.

2.3.5 Application Mapping

On a many-core platform, application mapping determines how to map a set of concurrent tasks onto a group of IP cores provided by the platform, with respect to specific design interests and constraints. While aforementioned methods achieve energy-efficiency at the infrastructure level, application mapping provides an overall system design method which takes the characteristics of both platforms and applications into account. Application mapping is classified as an outstanding design problem and important design

dimension on many-core platforms, which plays significant role in determining the energy consumption and performance of the overall system [83] [84]. Research focusing on the application mapping problem has been extensively performed from various perspectives [27, 85, 86, 31, 30, 41, 36, 87].

Different task-core mapping solutions will result in different performances and energy consumption characteristics for the same set of tasks. Figure 2.7 gives a simple example of two mapping solutions for tasks T_1 to T_4. Among them, the data volume between tasks T_1 and T_3 is larger than that between tasks T_2 and T_4. Without considering the inter-task data volume, tasks T_1 and T_3 might be mapped farther than tasks T_2 and T_4, as shown in Figure 2.7a. Clearly, this is not energy-efficient because larger amount of data between tasks T_1 and T_3 has to be transmitted on a longer routing path (two hops). Figure 2.7b gives an optimized mapping solution whereby the data volume is taken into account so that T_1 and T_3 are mapped on two neighboring nodes. The energy saving from communication between tasks T_1 and T_3 results in the overall energy decrease even though there is a slight increase between tasks T_2 and T_4. Simple as the example is, Figure 2.7 illustrates the basic principle of energy-efficient application mapping.



(a) Random Mapping (b) Energy-Efficient Mapping

Figure 2.7: A Basic Principle of Energy-Efficient Application Mapping

As presented in Section 1.3, application mapping is indeed a resource allocation problem. On a many-core platform, the resources that can be allocated to execute one or more applications include the IP cores and the communication channels in the interconnection network, or computation and communication resources, respectively. For the sake of generality, considering heterogeneous many-core platforms, application mapping can achieve energy-efficiency by the following approaches:

1. To select an appropriate type of IP core for each task so that the energy consumed by the group of IP cores for executing the tasks is minimized. This task-core mapping is called *IP selection* in this thesis. IP selection is enabled by the heterogeneity of IP cores which means that each core has various performance and energy consumption characteristics for a set of tasks. For example, each IP core presented in [88]

has various features such as energy consumption, execution time and memory consumption for individual tasks. IP selection is taken into account in order to minimize the computation energy consumption. In [85], IP selection is performed to minimize the communication loads between cores with the consideration of that tasks mapped on different IP cores generate communication loads. In this case, IP selection is also an approach of reducing the communication energy consumption.

2. To assign a tile (or node) on the many-core platform to map the IP cores that have been selected in the IP selection stage. Tile assignment aims at minimizing the inter-core communication energy consumption since the distance between two communicating tiles affects the energy consumed for transmitting data between two tasks on the tiles. This core-tile mapping is called *tile assignment* in this thesis. [3] presents a branch-and-bound algorithm for mapping a set of IP cores on a NoC architecture so that the communication energy consumption is minimized. In [89], a clustering technique is applied to the core-to-node mapping in order to minimize the communication power consumption on 2D mesh NoCs.

3. To find a specific routing path for communication between IP cores that have been mapped onto tiles in the tile assignment stage. The allocation of routing path to inter-core communication is called *communication mapping* in this thesis. Interconnection network may provide multiple routing path between two tiles. Communication mapping can be performed, for instance, to minimize the communication energy consumption [85], or to alleviate the contention and congestion on the links so that the packet latency can be decreased [90] [91].

The IP selection, tile assignment and communication mapping explore the design space for minimizing the energy consumption from different perspectives. For an application mapping problem, these three approaches can be performed systematically to obtain a more optimized mapping solution than each of them can achieve individually. Generally, the three stages proceed sequentially, i.e., IP selection at first, then tile assignment, and communication mapping at last. But in some cases, it is unnecessary to perform all three stages. For example, on NoCs employing deterministic routing algorithms, communication mapping is not applicable since the routing path between any two tiles is determined. In addition, to achieve an overall optimal mapping, the three stages may need to iterate with feedback from each other stages.

IP selection, tile assignment and communication mapping are all featured as the *combinatorial optimization* problems which consist of finding an optimal solution from a finite set of candidate solutions. Each of them is

an instance of the constrained quadratic assignment problem which is NP-hard [26]. The search space in each problem increases factorially with the number of tasks and IP cores on a many-core platform. For such problems, deterministic methods, e.g., exhaustive search, which exhaustively explore the search space, can find the optimal solution. But, it is prohibitively expensive, if not impossible, for the practical size of problems. Therefore, constructive or transformative heuristics are widely employed for application mapping on many-core platforms.

The three-stage application mapping approach will be presented in detail in Chapter 3, in terms of the formulation of general application mapping, the objective and constraint functions, and the representative mapping methods.

2.4 Chapter Summary

This chapter introduced the state-of-the-art shared address space and message passing parallel architectures. The emphasis was put on the communication mechanism applied on each type of architecture because it has great impact on energy consumption of a many-core platform. As an important component which affects the energy consumption and performance of a many-core platform, the interconnection networks were discussed in terms of topologies, routing algorithms and switching techniques. Of them, network-on-chip was presented as a promising interconnection structure. Typical energy-efficient design methods were explained. Application mapping was presented as an energy-efficient design method at the system level.

Chapter 3

Application Mapping

Generally speaking, the role of application mapping is to decide how to distribute a set of tasks on a set of cores so that the particular metrics of design interest can be optimized. There are two inputs to this problem: application and NoC platform. An application is composed of a set of tasks which can execute in parallel. There are inter-task communications representing data or control dependencies between tasks. A NoC provides the computation and communication resources for completing the application. For a heterogeneous NoC, on-chip processing elements (PEs) could consist of CPUs, DSPs, video processors, FPGAs, embedded memory blocks and other IP cores (as shown in Figure 3.1). Due to the increasing number of integrated cores on the many-core platforms, as well as the increasing complexity of massively parallel computing, application mapping plays more and more important role in improving the system performance and decreasing the energy consumption [83] [84]. As shown in [92], the application mapping generated by the optimized SA algorithm can save up to 66.32% of energy consumption compared to traditional SA algorithm. To find such kinds of optimal or near-optimal mapping solutions, optimization algorithms need to be applied to application mapping problem.

This chapter presents the general three-stage application mapping framework. The mapping stages are formulated with classical application and NoC models. The energy model of a many-core NoC is derived which is used as one major objective of the energy-efficient mapping algorithms developed in this thesis. The representative mapping methods that have been applied to application mapping are studied and compared in order to clarify the directions we should explore to develop more efficient mapping algorithms in this thesis.

3.1 Three-Stage Application Mapping

As mentioned in Chapter 2, we define the application mapping on a heterogeneous many-core platform as a three-stage resource allocation problem. The typical three-stage application mapping is depicted in Figure 3.1. In a heterogeneous system, a task could have multiple implementations with different performances and costs, corresponding to different type of IP cores provided in the many-core NoC. Therefore, the first stage of application mapping is to select one type of IP core to implement a particular task, called *IP selection*. The second stage is called *tile assignment*. The tile assignment decides which tile on a NoC should be assigned to a selected IP core. This stage performs the physical placement of the selected IP cores. To efficiently distribute the communication traffic on the communication network, the third stage, *communication mapping*, is performed after the previous two stages, to find a particular routing path for each pair of communicating cores (tasks).

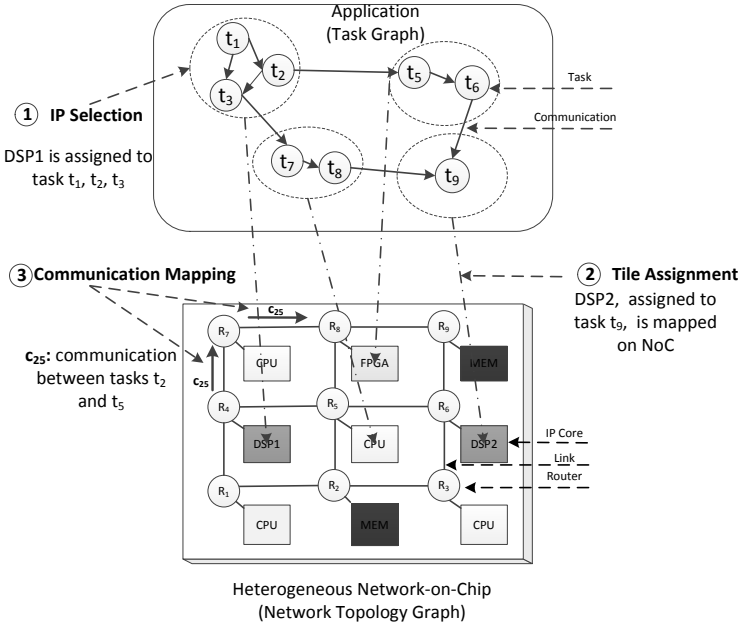


Figure 3.1: An Overview of Multi-Stage Application Mapping on Heterogeneous NoC

Based on the time when the mapping is performed, application mapping can be categorized into static and dynamic mapping. Static mapping is performed at design time, while dynamic mapping is performed at runtime. Static mapping can utilize full information about the application and the NoC, including the topology of the application, the volume/rate of inter-task communications, and the characteristics of the NoC. In contrast to

static mapping, the information used in dynamic mapping is limited to what can be collected at runtime. As a result, superior mappings can be found by static mapping with a global view of the system, compared to those found by dynamic mapping with a local view of the system [93]. This thesis emphasizes on the static mapping on NoC-based many-core platforms.

3.2 NoC and Application Model

To solve the application mapping problem, we first need to define the NoC and the application. The most widely used approach is graph model. The abstracted graph can provide the information associated with the NoC and the application, which are used in the evaluation of the objective functions and constraints in application mapping.

3.2.1 NoC Model

A heterogeneous NoC is characterized by communication network and IP repository provided by the NoC. As discussed in Section 2.2.4, 2D mesh NoC is the most popular NoC structure due to its regularity and modularity. In this thesis, 2D mesh topology is assumed for the formulation of the NoC. The deterministic XY routing is adopted which is a minimal path routing algorithm and is free of deadlock and livelock. The wormhole routing is employed due to its low cost (the buffer capacity can be less than the length of the packet) and low latency (the router can start forwarding the first flit of a packet without waiting for the tail).

Communication Network

As shown in Figure 3.1, the communication network is modeled as a network topology graph (NTG) . A NTG is a directed graph $< R, L >$, where $R = \{r_1, r_2, \ldots, r_n\}$ is the set of vertices, representing the set of n routers available on the NoC. Each router connects to one IP core as its processing element. $L = \{(r_i, r_j) | r_i, r_j \in R\}$ is the set of edges, representing the set of links between routers. $l_{ij} = \{(r_i, r_j)\}$ denotes the link between two adjacent routers r_i and r_j. R and L have the following properties:

1. $\forall r_i \in R$
   (a) $PE(r_i)$ gives the type of IP core connected to the router r_i.
   (b) E_{bit} gives the energy consumed for transferring one bit of data through the router [3].

2. $\forall l_{ij} \in L$

   (a) $bw(l_{ij})$ gives the maximal bandwidth of link l_{ij}.

   (b) $E_{L_{bit}}$ gives the energy consumed for transferring one bit of data on the link l_{ij} [3].

Generally, a router and the IP core connected to the router are termed as a tile or a node, which is usually labeled by its coordinates on the NoC. For example, in Figure 3.1 the bottom-left and top-right nodes could be labeled as $(0, 0)$ and $(2, 2)$ respectively.

IP Repository

IP repository is a library of IP cores provided by a NoC. For example, in the Embedded System Synthesis Benchmarks Suite (E3S) [94], 17 processors are provided in the IP repository. Each of them has different features in terms of price, energy and power consumption, and so on. Each processor also consumes various amount of energy and time for executing different types of tasks. Similar to E3S, we define the IP repository as $IPR = IP_1, IP_2, \ldots, IP_q$, where each IP_i is one type of IP core and has the following properties:

1. $IPType$ gives the type of the IP core (e.g., CPU, DSP, MEM and FPGA shown in 3.1), which defines the category of tasks that IP_i can execute.

2. ET_i gives the set of execution times corresponding to tasks that IP core IP_i can execute, e.g., task time in [94].

3. EC_i gives the set of energy consumptions corresponding to tasks that IP core IP_i can execute, e.g., task power in [94].

4. $w(IP_i)$ gives the number of IP_i type of cores available on the NoC and,

$$\sum_{i=1}^{q} w(IP_i) = n \tag{3.1}$$

   which means that the number of IP cores equals to that of routers on the NoC, assuming that one core is only connected to one router.

3.2.2 Application Model

The widely used models for defining an application are task graph [31] [95] and core communication graph [3] [89] [87].

Task Graph

A task graph (TG) is a graphical abstraction of an application, which presents the characteristics of an application. Figure 3.2 is an example of a TG of a simplified MP3 decoder in [96]. A TG is a weighted directed graph $< T, C >$, where $T = \{t_1, t_2, \ldots, t_m\}$ represents the set of m tasks of an application, corresponding to the set of TG vertices, and $C = \{c_{ij} = (t_i, t_j) | t_i, t_j \in T\}$ represents the communication between tasks, corresponding to the set of TG edges. For a communication $c_{ij} = (t_i, t_j)$, task t_i is called the *parent task* of task t_j. Conversely, task t_j is the *child task* of task t_i. A task without parent task is called an *entry task* and a task without child task, an *exit task*. The direction of an edge c_{ij} presents either the data or the control dependency between tasks t_i and t_j. The sets T and C have the following properties:

1. $\forall t_i \in T$
   (a) *id* gives the unique name of task t_i, which also defines the functionality of task t_i.
   (b) *parent* gives the set of tasks that are parent tasks of task t_i, in other words, the tasks that t_i is dependent on.
   (c) *child* gives the set of tasks that are child tasks of task t_i, in other words, the tasks that are dependent on task t_i.
   (d) IM is the set of implementations of task t_i. Each $IM_{ij} \in IM$ corresponds to the implementation of task t_i on jth type of IP core.
2. $\forall c_{ij} \in C$
   (a) vol_{ij} gives the data volume of the communication c_{ij}.
   (b) bw_{ij} gives the bandwidth requirement of the communication c_{ij}.
   (c) la_{ij} gives the maximal latency constraint of the communication c_{ij}.

The examples of task t_0 and communication c_{01} shown in Figure 3.2 illustrate how to define T and C in a TG of an application. A TG and related properties can be derived by static or dynamic program analysis/profiling. As the details of the process are beyond the scope of the thesis, interested readers can refer to [23] and [24] for detailed explanation.

Core Communication Graph (CCG)

After each task is assigned to an IP core in the IP selection stage, a task graph is transformed into a core communication graph. In contrast to the

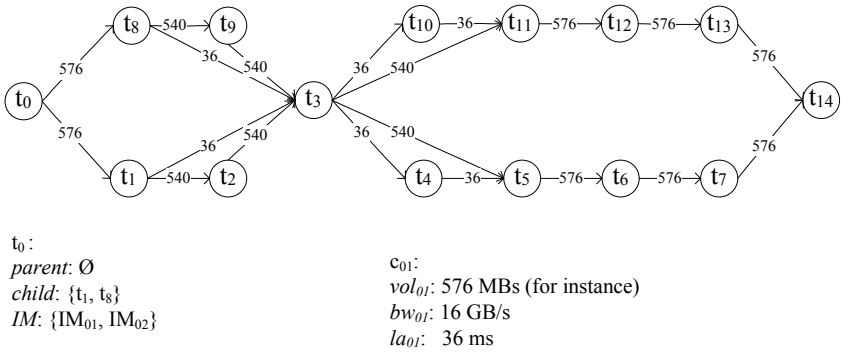


Figure 3.2: A Task Graph of MP3 Decoder

original TG, a CCG presents the communication between cores selected for an application. Figure 3.3 is the transformed CCG of the TG in Figure 3.2. In a CCG, the tasks assigned to the same IP core are clustered as one vertex (depicted by the dotted circle). That is, in Figure 3.3, 9 cores are selected for the 14 tasks in Figure 3.2. A CCG is a weighted directed graph $< Cor, Com >$, where $Cor = \{cor_1, cor_2, \ldots, cor_p\}$ represents the set of p IP cores selected for the application ($1 \leq p \leq \min\{m, n\}$), corresponding to the set of CCG vertices, and $Com = \{com_{ij} = (cor_i, cor_j) | cor_i, cor_j \in Cor\}$ denotes the communication between cores, corresponding to the set of CCG edges. The sets Cor and Com have the following properties:

1. $\forall cor_i \in Cor$
   (a) $IPType$ gives the type of the core cor_i, which is defined in the repository IPR.
   (b) T_i is the set of tasks assigned to the core cor_i.
   (c) $\forall t_j \in T_i$, ET_{ij} gives the execution time of task t_j on the core cor_i.
   (d) $\forall t_j \in T_i$, EC_{ij} gives the energy consumption for executing task t_j on core cor_i.
2. $\forall com_{ij} \in Com$
   (a) $vol(com_{ij})$ gives the aggregate data volume between cores cor_i and cor_j, and $vol(com_{ij}) = \sum vol(c_{uv})\,(\forall t_u \in T_i, \forall t_v \in T_j)$.
   (b) $bw(com_{ij})$ gives the worst case bandwidth requirement between cores cor_i and cor_j, and $bw(com_{ij}) = \sum bw(c_{uv})\,(\forall t_u \in T_i, \forall t_v \in T_j)$

While a TG represents only application features, a CCG is based on application and IP core features and provides a representation of the application mapped on NoC cores.

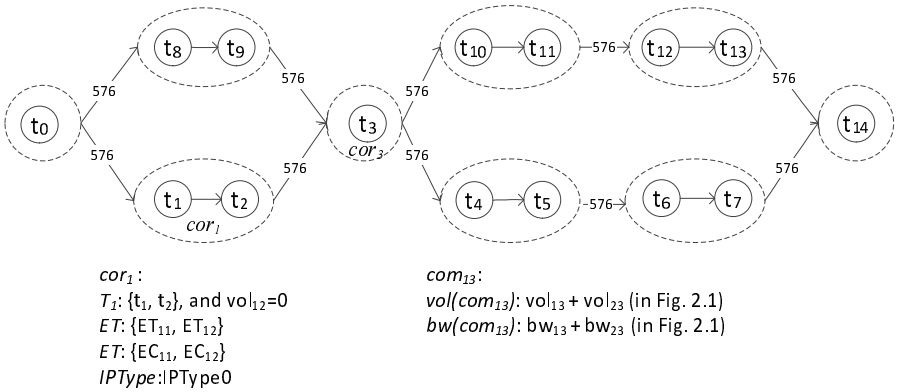


Figure 3.3: An Example CCG of the TG in Figure 3.2

3.3 Formulation of Application Mapping Problem

In traditional systems with multiple IP cores, application mapping, also called task mapping, mainly refers to how to map each task on a specific IP core. Both the category and the number of integrated IP cores are limited. On a NoC-based many-core system, as shown in Figure 3.1, various types of IP cores are provided, and a number of same type of IP cores may be distributed on different tiles on the NoC. Moreover, there can be several routing paths between two tiles. Hence, two additional approaches, tile assignment and communication mapping, can be applied to explore the potential provided by the many-core NoC architectures and to find more optimized mapping solution for a many-core system. Altogether, a globally optimal mapping can be obtained by performing IP selection, tile assignment and communication mapping.

3.3.1 Framework of Three-Stage Application Mapping

The framework of the general application mapping is depicted in Figure 3.4. Prior to the application mapping, the application and NoC architecture have been modeled by task graph and network topology graph (with IP repository IPR) respectively. Network topology graph also defines design constraints, e.g., the available bandwidth on communication links. The first stage of application mapping is IP selection which selects one type of IP core in IPR for each task. The output of IP selection is the core communication graph (CCG). Thereafter, the tile assignment decides on which tile (node) each selected core in the CCG should be placed. The assignment of one core in CCG might be constrained to particular tiles due to the heterogeneity of IP cores on a NoC, as shown in Figure 3.4. When the tile assignment is done,

the last stage, the communication mapping, is to allocate one routing path on the communication network for each communication in the CCG.

At each stage, a candidate mapping is evaluated with respect to a set of pre-defined objective functions and constraints. The mapping process iterates with the feedback from the evaluations until the best mapping solution is found. The feedback can come from the current stage (called internal feedback), or from the other stages (called external feedback). For example, if no mapping solution could satisfy the bandwidth constraint in the communication mapping step, an external feedback would be sent to the tile assignment and a new IP core to tile mapping solution has to be created for the next round of communication mapping, even though it is worse than previous tile assignment solution. This means, to find a globally optimal solution with respect to the particular objectives and constraints, IP selection, tile assignment and communication mapping stages have to operate jointly. Finally, a mapping solution satisfying all pre-defined design objectives and constraints will be obtained after the three-stage mapping.

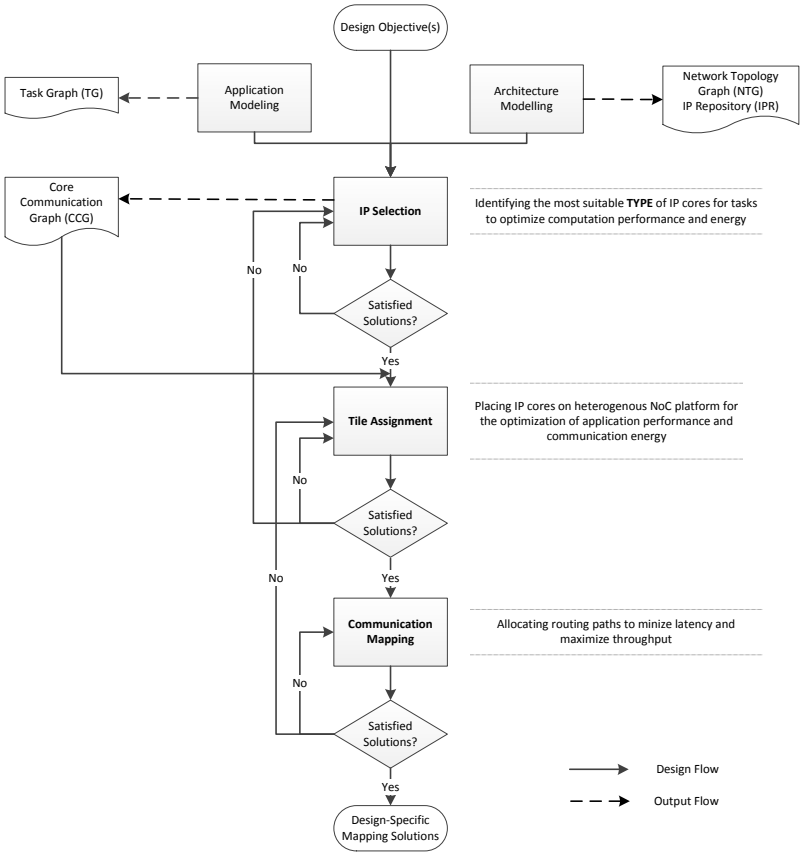


Figure 3.4: The Work Flow of Application Mapping

The following sections formulate the problem at each stage. The mapping function for each stage is defined.

3.3.2 IP Selection

Given a TG and an IPR, the role of IP selection is to select one type of IP core to a task. The IP selection function is defined as:

$$IPSelect : T \rightarrow IPR, IPSelect(t_i) = IP_j, \forall t_i \in T, \exists IP_j \in IPR \quad (3.2)$$

subject to:

$$\forall t_i \in T, \sum_{\forall IP_j \in IPR} A_{ij} = 1 \quad (3.3)$$

and

$$\forall IP_j \in IPR, \sum_{\forall t_u \in T} \sum_{\forall t_v \in T} A_{uj} A_{vj} B_{uv}/2 \leq w(IP_j) \quad (3.4)$$

where

$$A_{ij} = \begin{cases} 1 & IP_j \text{ type of IP core is assigned to } t_i \\ 0 & \text{otherwise} \end{cases}$$

and

$$B_{uv} = \begin{cases} 1 & \text{tasks } t_u \text{ and } t_v \text{ are assigned to different cores in CCG} \\ 0 & \text{otherwise} \end{cases}$$

Equation 3.3 restricts that a task can only be assigned to one core, while Equation 3.4 gives the constraint on the availability of a certain type of IP core, which means the number of IP_j type of core selected for an application cannot exceeds that are provided by the NoC. Arrays $A : T \times IPR$ and $B : T \times T$ are the results of the IP selection stage. Using arrays A and B, a TG is transformed into a CCG.

3.3.3 Tile Assignment

Given a CCG and a NTG, tile assignment is to decide on which tile a selected core in the CCG should be placed on the NoC. The tile assignment function is defined as:

$$TileAssign : Cor \rightarrow R, TileAssign(cor_i) = r_j, \forall cor_i \in Cor, \exists r_j \in R \quad (3.5)$$

subject to

$$\begin{aligned} \forall cor_i \in Cor, \sum_{\forall r_j \in R} D_{ij} = 1 \\ \forall r_j \in R, \sum_{\forall cor_i \in Cor} D_{ij} = 1 \end{aligned} \quad (3.6)$$

and

$$cor_i.IPType = PE(r_j).IPType \tag{3.7}$$

where

$$D_{ij} = \begin{cases} 1 & cor_i \text{ is mapped on } r_j \\ 0 & \text{otherwise} \end{cases}$$

Equation 3.6 ensures that only one tile on the NoC is assigned to a core in the CCG. Equation 3.7 guarantees that a selected core in CCG is mapped on a tile with same type of IP core. The tile assignment stage results in array $D : Cor \times R$.

3.3.4 Communication Mapping

After IP selection and tile assignment stages, communication mapping, also known as routing path allocation, chooses one specific path for each communication between communicating cores. Communication mapping is an efficient way to balance the communication and reduce the potential contentions and congestions on the communication network, which in turn decrease the average/maximal packet latency and save energy for buffering the packets on the routers [29] [91]. The communication mapping function is defined as follows:

$$ComMap : Com \rightarrow L, ComMap(com_{ij}) = \Omega_{ij}, \forall com_{ij} \in Com \tag{3.8}$$

where Ω_{ij} is a routing path on the NoC from the source router r_i (connected to core $TileAssign^{-1}(r_i)$ in CCG) to the destination router r_j (connected to core $TileAssign^{-1}(r_j)$ in CCG). Ω_{ij} is composed of a nonempty sequence of links $l_0, l_1, \ldots, l_k$ and

1. $d(l_i) = s(l_{i+1})\,(0 \leq i \leq k)$,
2. $s(l_0) = r_i$, $d(l_k) = r_j$.

Subject to

$$\sum bw_{l_u}^{com_{ij}} \leq bw_{l_u}, \forall com_{ij} \in Com, \forall l_u \in L \tag{3.9}$$

where the left part of the inequality is the bandwidth requirement on the link l_u, and $bw_{l_u}^{com_{ij}}$ is obtained by the following equation:

$$bw_{l_u}^{com_{ij}} = \begin{cases} bw_{com_{ij}} & \text{if } l_u \in \Omega_{ij} \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

Equation 3.9 defines the bandwidth constraint of each link on the NoC. The aggregate bandwidth requirement could not exceed the bandwidth available on a link.

Note that, the communication mapping is not applicable to NoCs where runtime adaptive routing is employed. In addition, the minimal path routing is preferable in the communication mapping. The minimal path routing chooses the shortest path between two communicating cores, which is able to minimize the communication energy consumption. Take a 2D mesh NoC as an example, the set of shortest paths lie in the rectangle formed by the source and destination nodes. The length of the shortest path is the Manhattan distance between two nodes, i.e., $|x_d - x_s| + |y_d - y_s|$ ((x_s, y_s) and (x_d, y_d) are the coordinates of the source and destination nodes respectively). Equation 3.10 gives the bandwidth requirement on link l_u imposed by the communication com_{ij}, where $bw_{com_{ij}}$ is defined in Section 3.2.2.

3.4 Energy Consumption Model

Energy consumption has drawn significant attention in application mapping [86] [88] [3] . A major goal of application mapping is to minimize the energy consumption of executing an application. The energy consumption is contributed by both communication network and processing cores, corresponding to the communication and computation energy consumption respectively. The energy consumption is formulated as:

$$E = E_{comm} + E_{comp} \tag{3.11}$$

where E_{comm} and E_{comp} represent the communication and computation energy consumption respectively. Both E_{comm} and E_{comp} are composed of *dynamic* and *static* energy consumption. Dynamic energy consumption is proportional to switching activity, which happens when data is being processed on the IP cores or transferred on the NoC. Static energy consumption mainly originates from subthreshold leakage current and is dependent on the application execution time, the number of transistors and the temperature [71] [72].

A classical dynamic communication energy consumption model of NoCs, based on the bit energy consumption E_{bit}, was proposed in [3] as follows:

$$E_{bit} = E_{B_{bit}} + E_{S_{bit}} + E_{L_{bit}} \tag{3.12}$$

where $E_{B_{bit}}$, $E_{S_{bit}}$ and $E_{L_{bit}}$ are the energy consumed for transferring one bit data by the buffer, switch, and link at a router respectively.

The dynamic energy consumed by communication com_{ij} in a CCG is proportional to E_{bit}, the data volume $vol(com_{ij})$ and the length of the routing path $\gamma_{ij} = |\Omega_{ij}|$. Equation 3.13 gives the dynamic energy consumption of the communication com_{ij}.

$$E_{com_{ij}} = vol(com_{ij}) \times (\gamma_{ij} \times (E_{B_{bit}} + E_{S_{bit}}) + (\gamma_{ij} - 1) \times E_{L_{bit}}) \tag{3.13}$$

And the dynamic communication energy consumption, $E_{comm-dy}$, is the total energy consumed by all communications in a CCG.

$$E_{comm-dy} = \sum_{com_{ij} \in Com} E_{com_{ij}} \tag{3.14}$$

A static communication energy consumption model was proposed in [97] as follows:

$$E_{comm-st} = n \times P_{router} \times t_{exec} \tag{3.15}$$

where n is the number of routers on the NoC, P_{router} is the static power consumption of each router, which takes the temperature into account. t_{exec} is the execution time of the application. The total communication energy consumption is computed as:

$$E_{comm} = E_{comm-dy} + E_{comm-st} \tag{3.16}$$

Using the application and NoC models presented in Section 3.2, and the IP selection result $A : T \times IPR$ in Section 3.3.2, the dynamic computation energy consumption $E_{comp-dy}$ is calculated by:

$$E_{comp-dy} = \sum_{IP_j \in IPR} \sum_{t_i \in T} EC_{ij} \times A_{ij} \tag{3.17}$$

and the static computation energy consumption can be calculated similarly as the static communication energy consumption in Equation 3.15:

$$E_{comp-st} = \sum_{i=1}^{n} P_{cor_i} \times t_{exec} \tag{3.18}$$

where P_{cor_i} is the static power consumption of IP core $cor_i \in Cor$.

The total computation energy consumption is computed as:

$$E_{comp} = E_{comp-dy} + E_{comp-st} \tag{3.19}$$

Equation 3.14 and 3.19 are used to evaluate the qualities of mapping solutions found by the algorithms developed in this thesis.

3.5 Mapping Methods

3.5.1 Overview of Mapping Methods

Recalling the formulations of IP selection, tile assignment and communication mapping, we know that all resource allocation problems dealt with the three stages are the *combinatorial optimization* problem, which consists of finding an optimal solution from a finite set of candidate solutions. Each

of them is an instance of constrained quadratic assignment problem, which is known to be NP-hard [98]. The search space in each problem increases factorially with the application and NoC sizes. For such problems, deterministic methods, e.g., exhaustive search and Branch-and-Bound search, which exhaustively explore the search space, are guaranteed to find the optimal solution. But, they are not affordable for the practical size of problems due to the prohibitively expensive computing. Therefore, constructive or transformative heuristics are widely used to solve the mapping problems with large number of tasks and NoC nodes.

Heuristics are pseudo-random search and optimization techniques which perform the exploration of search space based on learned experience. Compared to exhaustive search, heuristics can provide near-optimal solution with reduced complexity and affordable computation. Heuristics incrementally improve the quality of mappings over iterations or generations, until the number of iterations (generations) reaches to a pre-defined limit or the criteria of termination are met. At each iteration (generation), current mappings are reconstructed and evaluated for finding the better solutions compared to the best ones that have been found so far. Widely used heuristics for application mapping include greedy incremental, simulated annealing and genetic algorithm.

Another method to solve the application mapping problem is to use mathematical programming. Linear programming (LP) is an example of mathematical programming. Using mathematical programming, it is able to find the set of parameters which can minimize or maximize the objective functions under a set of constraints. In practice, integer linear programming (ILP) is more applicable for application mapping problem where the variables in objective functions and constraints are integer.

The following sections discuss the representative algorithms which have been applied to application mapping problems. Following the illustrations of the algorithms, an comparison between algorithms is performed in terms of search space size, solution quality, complexity and problem scope. The comparison clarifies the directions that will be explored in this thesis in order to develop high-performance energy-efficient application mapping algorithms.

3.5.2 Exhaustive Search (ES)

ES is the simplest and most straightforward method for application mapping problem. To find the optimal solution in the entire search space, all candidate solutions are exhaustively enumerated and evaluated with respect to the objective functions. Take the IP selection problem as an example, for a CCG with p cores and a NTG with n routers ($p \leq n$), there are in total $n!/(n-p)!$ candidate solutions, which incurs an $o(n!)$ complexity for the ES algorithm. Considering this factorially increased complexity, in practice, ES

is only applicable to the mapping problems with small size. The experiment in [97] shows that, for all small NoCs (up to 3×4 or 2×5), it is possible to find the optimal mapping by ES. For larger ones (8×8 and 10×10), it is impossible to find the optimal mappings by ES within a reasonable computation time.

3.5.3 Branch-and-Bound Search

Branch-and-Bound (BB) search deals with optimization problems over a search space that can be represented by the leaves of a search tree. Each intermediate node represents a partial solution and a leaf node, a complete solution. The search tree is traversed in a specific order, and the score of the best leaf found so far is kept as a bound B. If the most achievable score among the descendant leaf nodes of an intermediate node is worse than B, the search tree is pruned at that node, i.e., its subtree (a branch) will not be searched any more. Otherwise, the child nodes of current node will be generated and evaluated in the next round. The algorithm continues until all non-pruned branches have been enumerated and the best leaf is found.

In [25] [26] [3], BB algorithm is applied to tile assignment and communication mapping in order to minimize the communication energy consumption subject to the specified design constrains. The mono-objective BB algorithm proposed in [26] is extended to a Pareto-based BB (PBBB) algorithm in [95] to perform multi-objective exploration of the mapping space. Even though BB algorithm is possible to skip unnecessary evaluations by branch-pruning, it is difficult to predict the exact effect. In the worst case, the complexity of BB is as high as ES. In [99], the authors use an efficient BB mapping algorithm to minimize the execution time. The BB algorithm is optimized by using greedy heuristic for the initial solution and bounds so that the convergence is speeded up. The proposed mapping algorithm in [100] tries to reduce the search space of BB algorithm by binding the tasks with large communication volume together and mapping them preferably on adjacent nodes in the NoC. The binding technique also aims to decrease the communication energy consumption.

The example of search tree for mapping a four-core application (represented by a CCG) onto a 2×2 NoC, derived from [3], is shown in Figure 3.5.

The root node corresponds to the state that no IP core has been mapped on the NoC. Intermediate node, for example, "03XX", represents that cores cor_0 and cor_3 have been mapped to routers r_0 and r_1 respectively, while cor_2 and cor_1 have not yet been mapped. The leaf node "0321" means that four cores, cor_0, cor_3, cor_2 and cor_1 have been mapped to routers r_0 to r_3 respectively. Two bounds, the upper bound cost (UBC) and the lowest bound cost (LBC) are employed. UBC of a node is defined as a value that is

no less than the minimum energy consumption of its legal, descendant leaf nodes. LBC of a node is defined as the lowest energy consumption that its descendant leaf nodes can possibly achieve. The detail of the BB algorithm using LBC and UBC techniques can be found in [3]. The experiment shows that for applications with 9 and 36 cores, compared to a general simulated annealing algorithm, the speedup achieved by the BB algorithm is 45 and 127 times respectively, while the energy consumptions are reduced for most benchmarks.

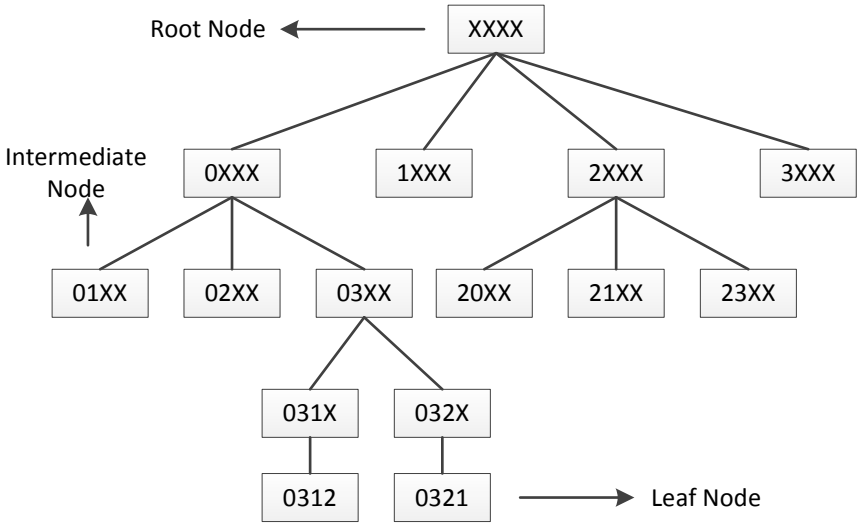


Figure 3.5: An Example of Search Tree in [3]

3.5.4 Greedy Incremental Search

Greedy Incremental (GI) algorithm is an intuitive approach for solving the combinatorial optimization problems. Starting from an empty solution $X = \emptyset$, GI algorithm repeatedly adds one element x to X from a set of candidate elements E so that the best improvement of the objective function can be achieved by the solution $X \cup x$. The process continues until $E = \emptyset$ [101]. Decision at each step is made based on improving local or current state without considering global optimization. And the previous decision will not be reconsidered in the future steps.

GI algorithm is employed in [28] for the tile assignment and the minimum-path routing. Starting from an initial mapping, the GI algorithm iteratively improve the mapping solution in a way of swapping two tasks at one time. Only when improvement is measured, can a swap be accepted. Similar to [28], GI is applied in [30] as one kind of energy-efficient tile assignment algorithm. GI algorithm is composed of two nested loops. Starting from an initial mapping, the external loop chooses a node on the NoC as the pivot

of evaluation, and the internal loop iteratively swaps the IP core mapped on this node with those on other nodes. The swapped nodes that lead to the largest saving on energy consumption (same model as Equation 3.13), are returned by the internal loop and the swap is performed by the external loop. The external loop then proceeds to find the next pivot by discarding the previously evaluated nodes. The algorithm stops when there is no pivot available.

According to the comparison in [30], the advantage of GI algorithm is its faster speed compared to other algorithms. It can obtain a local optimum by performing the pair-wise swapping $O(n^2)$ times for an n-node NoC. The time complexity of the GI algorithm is therefore $O(n^2)$. The quality of the final mapping solution is not guaranteed to be the global optimum, and is greatly affected by the initial mapping solution.

3.5.5 Simulated Annealing (SA)

SA algorithm originates from the annealing of solids in the field of metallurgy, a way involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. While heated at high temperature, atoms with high energy become more active and are able to move away from their initial positions which represents a state of the local minimum in terms of internal energy. Then, new state with changed internal energy is created due to these moves. While the temperature is falling down, the material will finally reach to the frozen state with minimal internal energy. The slow cooling gives atoms more chances to move to a state with lower internal energy.

The pseudo-code of the generic SA algorithm is shown in Algorithm 1. The symbols and definitions of the applied functions and parameters are listed in 3.1. Different implementation alternatives of these functions and parameters are summarized in [31]. Analogous to the process of physical annealing, the general SA algorithm proceeds by iteratively replacing the current solution with a new randomly generated solution. The moves from current solution to that with lower or equal cost (downhill moves) are always accepted, and those to higher cost (uphill moves) may be accepted with a changing probability which depends on the cost difference ($\Delta C > 0$) and a global parameter t (called the temperature). The temperature t gradually decreases during the annealing process. SA becomes greedy when the temperature t approaches to zero and terminates when the function $Terminate(i, R)$ becomes true. During the whole process, SA behaves like a pure randomizing algorithm at the beginning and a pure greedy algorithm at the end. This feature allows the SA algorithm to take advantage of both the randomizing algorithm (the ability of finding the global optima) and the greedy algorithm (a faster convergence).

Algorithm 1: General Simulated Annealing Algorithm

```
1  S ← S_0
2  C ← Cost(S_0)
3  S_best ← S
4  C_best ← C
5  R ← 0
6  for i ← 0 to ∞ do
7     T ← Temp(i, T_0)
8     S_new ← Move(S, T)
9     C_new ← Cost(S_new)
10    ΔC ← C_new − C
11    if ΔC < 0 or Accept(ΔC, T) then
12       if C_new < C_best then
13          S_best ← S_new
14          C_best ← C_new
15       end if
16       S ← S_new
17       C ← C_new
18       R ← 0
19    else
20       R ← R + 1
21       if Terminate(i, T_0, R) = True then
22          break
23       end if
24    end if
25 end for
26 return S_best and C_best
```

Compared to a pure greedy algorithm, e.g., the GI algorithm presented previously, the significant strength of SA is the ability to find the global optimum. This is achieved by accepting worse solutions, which enables the algorithm to escape from the local optimum. However, to obtain the global optimum, SA algorithm generally needs much more evaluations than pure greedy algorithms. The result in [3] shows that to map an integrated video/audio application, the solutions produced by SA are better than those by BB at the price of slower speed. Based on the observation in [102], SA can be seen as a variant of the $(1 + 1)$-EA (evolutionary algorithm). It is hard to give an exact complexity of SA, which depends on the parameters and functions applied. Mathematical tools [102] can be utilized to analyze the complexity of different SA implementations.

Table 3.1: Functions and Parameters for SA

Symbol	Definition
S	Mapping solution (S_0: initial solution)
$Cost(S)$	Cost function
$Temp(i, T_0)$	Temperature function, return the temperature at iteration i
$Move(S, T)$	Move function, return a neighboring mapping of S at temperature T
$Accept(\Delta C, T)$	Accept function, return True (to accept) or False (to reject) a worse solution
$Terminate(i, R, T_f)$	Termination function, return True (to terminate the SA) or False (to continue the annealing)
T_0	The initial temperature
T_f	The final temperature
R	The number of rejected solutions

3.5.6 Genetic Algorithm

Compared to the aforementioned ES, BB and SA algorithms, genetic algorithm (GA) is more suitable for multi-objective optimization problems. GA is a search heuristic that mimics the process of natural evolution. GA usually starts from a population of randomly generated individuals, where each individual presents a solution. In each generation, the fitness of every individual in the population is evaluated and associated with a rank. To generate a new generation, usually the top few percentages of individuals are kept. The rest of the population is composed of the offspring created by crossover and mutation operators. For the crossover operation, two parent individuals are selected. Offspring are created by exchanging parts of the selected individuals. The crossover may be single-point or multi-point. The mutation operator selects one parent individual and randomly change some of its properties to create offspring. The evolution repeats until a termination condition has reached. A termination condition may be that, for example, a satisfied solution is found, or a fixed number of generations have been created and evaluated. The solution quality and convergence time of GA are controlled by parameters including crossover rate, mutation rate, population size, etc [103].

For a multi-objective problem, it is less likely to find one single solution which can optimize all objectives at the same time. Instead, a set of solutions need to be produced by GA algorithm. In practice, the domination concept [104] with a ranking schema is applied to classify the set of solutions into so-called *Pareto fronts*. Solutions from the front ranked as one (*Pareto-optimal*

front) are equally good or better than solutions from *Pareto fronts* with higher rank.

GA requires a genetic representation of the solution domain, and a fitness function (objective function) to evaluate the solutions. We take the work in [105] as an example to illustrate the implementation of GA, where GA is used in the IP selection stage with the purpose of minimizing the total execution time and power consumption.

1. Representation. Each individual (chromosome) is represented by one-dimension array, and indexes of the array correspond to the tasks in T (from TG). Initially, a random type of IP core is selected from the IP repository and assigned to each element (gene) in the array.

2. Tournament selection, one-point crossover and simple mutation are used.

3. Fitness Functions

   (a) Execution Time.

   $$Time = \max_{p \in Paths(TG)} (\sum_{\forall t \in p} time_t) \tag{3.20}$$

   where $Paths(TG)$ provides all possible paths in a given TG and $time_t$ is the required execution time for task t on its selected IP core. In this function, the communication delay is not considered in calculating the execution time of an application.

   (b) Power Consumption.

   $$Power = \sum_{t \in TG} power_t \tag{3.21}$$

   where $power_t$ is the required power consumption for the execution of task t on its selected IP core.

Similar to SA, the complexity of GA is problem-specific and dependent on different fitness functions and operators. A GA employing crossover and mutation operators is a $(N + N)$-EA [102], whose complexity is not yet possible to be calculated. In [106], the authors claimed that GA takes more computation time than SA, because it compares a population of individuals at each generation, while in SA only a pair of solutions are compared at each iteration.

3.5.7 Integer Linear Programming (ILP)

Linear programming (LP) is a technique for the optimization problem with a linear objective function, subject to linear equality and linear inequality constraints. The major advantage of LP is that it allows different, and often conflicting design interests to be integrated and explored within a unified model. A LP problem is formulated by dependent variables in the objective functions and independent variables in the constraint functions. ILP is a subclass of LP where variables are forcibly constrained to be integer.

A 0-1 ILP formulation is presented in [107] for energy-efficient mapping. The experiments on several real applications show that, for applications with smaller amount of tasks and communications, the proposed ILP approach can find optimum results under the given CPU time limit in most cases. However, for the applications with more tasks and communications (e.g., application VOPD with 16 tasks and 20 communications), the ILP method fails to obtain the optimum mapping in the given time. The ILP formulation presented in [88] takes both the computation and communication energy consumption into account. Although the global optimal mapping can be found by the proposed ILP formulation, the computational complexity of the ILP grows rapidly as the problem complexity grows. To alleviate the computational complexity of ILP approach, a simulated annealing with timing adjustment (SA-TA) heuristic is proposed as well. As claimed, the SA-TA algorithm achieves near global optimal mapping even under tight timing constraints (less than 5% energy overhead using 1000 iterations).

We use the instance in [29] to illustrate the ILP method. The objective in [29] is to produce a contention aware mapping which can trade off the network latency (and implicitly, the throughput) with the communication energy consumption. The problem is formulated as an ILP problem as follows.

1. Variables

   (a) $MD_{r_s r_t}$: the Manhattan Distance from routers r_s to r_t.

   (b) $l_1, l_2, \ldots, l_K$: the set of K uni-directional segment links on the NoC, for each link l_k, $l_k^{r_s r_t}$ defines whether or not the link l_k is on the routing path from routers r_s to r_t, and

   $$l_k^{r_s r_t} = \begin{cases} 1, & \text{if } l_k \in \Omega_{st} \\ 0, & \text{otherwise} \end{cases}$$

   where Ω_{st} is a routing path from source router r_s to destination router r_t.

   (c) $m_{cor_i}^{r_s}$ gives the mapping result, i.e.,

   $$m_{cor_i}^{r_s} = \begin{cases} 1, & \text{core } cor_i \in Cor \text{ is mapped on the router } r_s \in R \\ 0, & \text{otherwise} \end{cases}$$

where $Core$ and R are the set of cores in CCG and routers in NTG respectively.

(d) $p_{cor_i cor_j}^{r_s r_t}$ shows the communication mapping result, i.e.,

$$p_{cor_i cor_j}^{r_s r_t} = \begin{cases} 1, & \text{communicating cores } cor_i \text{ and } cor_j \text{ are mapped on routers } r_s \text{ and } r_t \text{ respectively} \\ 0, & \text{otherwise} \end{cases}$$

(e) $z_{l_k}\_cor_i cor_j cor_m cor_n r_s r_t r_p r_q$ is used to represent the path-based contention.

$$z_{l_k}\_(cor_i cor_j cor_m cor_n r_s r_t r_p r_q) = \begin{cases} 1, & \text{link } l_k \text{ is shared by routing path from routers } r_s \text{ to } r_t \text{ and that from routers } r_p \text{ to } r_q, \text{ on which communicating tasks } cor_i, cor_j, cor_m \text{ and } cor_n \text{ are mapped respectively.} \\ 0, & \text{otherwise} \end{cases}$$

2. Objective Function. The objective is to minimize the weighted communication distance (and implicitly, communication energy consumption) and the path-based contention as well, i.e.,

$$\min \Big\{ \frac{(1-\alpha)}{\beta} \times \Big(\sum_{\forall (cor_i, cor_j) \in Cor} w_{cor_i, cor_j} \times \Big[\sum_{\forall r_s, r_t \in R} (MD_{r_s r_t} \times p_{cor_i cor_j}^{r_s r_t}) \Big] \Big) + \frac{\alpha}{\gamma} \times \sum_{\forall l_k} z_{l_k}\_(cor_i cor_j cor_m cor_n r_s r_t r_p r_q) \Big\} \tag{3.22}$$

3. Constraints.

(a) One-to-one core-to-router mapping: each task is exclusively mapped to one router.

$$\begin{aligned} &\forall r_s \in R, \sum_{\forall cor_i \in Cor} m_{cor_i}^{r_s} \leq 1 \\ &\forall cor_i \in Cor, \sum_{\forall r_s \in C} m_{cor_i}^{r_s} = 1 \\ &\forall cor_i \in Cor, \forall r_s \in R, 0 \geq m_{cor_i}^{r_s} \leq 1 \end{aligned} \tag{3.23}$$

(b) Communication path: there must be one communication path for any communicating cores that are mapped on the NoC.

$$\begin{aligned}&\forall(cor_i, cor_j) \in Cor\\&m^{r_s}_{cor_i} + m^{r_t}_{cor_j} - 1 \leq p^{r_s r_t}_{cor_i cor_j} \leq \frac{m^{r_s}_{cor_i} + m^{r_t}_{cor_j}}{2}\\&0 \leq p^{r_s r_t}_{cor_i cor_j} \leq 1\end{aligned} \tag{3.24}$$

(c) Bandwidth constraint: for each link l_k, the total bandwidth requirement for all communications passing through the link cannot exceed the bandwidth $bw(l_k)$.

$$\sum_{\forall r_s, r_t \in R} \sum_{\forall (cor_i, cor_j) \in Cor} bw_{cor_i, cor_j} \times l^{r_s r_t}_k \times p^{r_s r_t}_{cor_i cor_j} \leq bw(l_k) \tag{3.25}$$

(d) Path-based network contention count: the path-based contention happens when two communications contend for the same link.

$$\begin{aligned}&\forall(cor_i, cor_j), (cor_m, cor_n) \in Cor\\&p^{r_s r_t}_{cor_i cor_j} + p^{r_p r_q}_{cor_m cor_n} + l^{r_s r_t}_k + l^{r_p r_q}_k - 3 \leq z_{l_k\text{-}}(cor_i cor_j cor_m cor_n r_s r_t r_p r_q)\\&0 \leq z_{l_k\text{-}}(cor_i cor_j cor_m cor_n r_s r_t r_p r_q) \leq 1\end{aligned} \tag{3.26}$$

After the mapping problem is formulated, tools like $MATLAB$, $Xpress-MP$ [107] and $lp-solver$ [29] can be used to obtain the final mapping solution. Note that, in [29], the authors also use a LP approximation to relax the NP-hard ILP problem, which significantly decreases the run-time of the mapping algorithm (from hours to less than one second). Similarly, to overcome the unacceptable complexity of ILP in dealing with application mapping problems, an approach of quadratic programming (QP) formulation is proposed in [108]. Due to the decrease in the number of variables for the mapping problems with large size, the QP approach is, as reported in [108], at least 10 times faster than the original ILP formulation for the given 20 benchmarks.

3.5.8 Summary of Mapping Methods

Based on the study of the above methods, we can see that each of them has a compromise between quality, search space, complexity and the scope of the applicable problems. The comparison between these methods is summarized in Table 3.2.

ES and BB are able to find the global optimum at the price of highest computational complexity, as a result of the exhaustive search over the entire

Table 3.2: Comparison of Mapping Methods

Method	Category	Search Space	Quality	Complexity	Problem Scope
ES	Deterministic Search	Whole Space	Global Optimum	$o(n!)$	Single-Objective
BB	Deterministic Search	Whole Space (worst case)	Global Optimum	$o(n!)$ (worst case)	Single-Objective
GI	Heuristic Search	Small Search Space	Local optimum	$o(n^2)$	Single-Objective
SA	Constructive Heuristic Search	Medium Search Space	Global or Near Global Optimum	Problem-Specific	Single- and Multi-Objective
GA	Transformative Heuristic Search	Large Search Space	Global or Near Global Optimum	Problem-Specific	Single- and Multi-Objective
ILP	Mathematical Programming	Whole Space	Global Optimum	Class NP	Single- and Multi-Objective

search space. Similarly, ILP is a method in the complexity class NP-hard [29] for finding the global optimum. It is impractical to use ES, BB and ILP for mapping problems with large number of tasks and NoC nodes. GI is the simplest algorithm, but it can only converge to a local optimum. As a trade-off between complexity and quality, SA and GA can converge to the global optimum or sub-optimum in finite iterations (generations) by exploring larger search space than GI. The exact complexity of SA and GA depends on the parameters and functions employed. In addition, SA, GA and ILP are capable of solving both single- and multi-objective problems.

3.6 Chapter Summary

The framework of the general application mapping process, consisting of *IP selection*, *tile assignment* and *communication mapping* stages, was presented in this chapter. Each stage focuses on a particular sub-problem and a final mapping solution is obtained by the collective efforts of these three stages. The energy consumption model used for evaluate the quality of mappings was derived based on the existing works. The methods applied to the application mapping problem was studied. Since the application mapping is

a NP-hard optimization problem, heuristics are employed to find the near-optimal solutions. SA, GA and ILP methods can find better mappings than GI and BB with reasonable computational complexity. SA, GA and ILP are also capable of solving the multi-objective mapping problems. By utilizing optimized parameters and operators, SA and GA can be further accelerated for specific mapping problems.

Chapter 4

Tree-Model Based Mapping Heuristic

This chapter presents a lightweight mapping algorithm based on a tree model of a NoC. All the nodes on a NoC as well as the interconnections between them are abstracted as a extended tree structure. By mapping the selected tasks one by one on the tree starting from the root of the tree, the tree-model based mapping algorithm minimizes the energy consumption and packet latency. The utilization of the tree model makes the mapping a one-round procedure which significantly decreases the run-time of the algorithm. Besides time efficiency, the proposed mapping algorithm achieves better performance compared to the GI algorithm.

4.1 Objective Function

The algorithm is dedicated to the tile assignment stage. Here we assume that the given application has already been partitioned and implemented as a set of tasks, and each of them has been allocated to one type of IP core. The objective of tile assignment is to minimize the communication energy consumption and packet latency. We use the application and NoC models presented in Section 3.2 to describe the application and NoC respectively. The NoC is assumed to be a 2D mesh NoC employing the minimal routing algorithm and the wormhole switching (see Section 2.2.3).

4.1.1 Energy Model

As the emphasis is put on minimizing the dynamic communication energy consumption, Equation 3.14 defined in Section 3.4 is used as the energy consumption model.

$$
\begin{aligned}
E_{comm-dy} &= \sum_{com_{ij} \in Com} E_{com_{ij}} \\
&= \sum_{com_{ij} \in Com} vol(com_{ij}) \times (\gamma_{ij} \times (E_{B_{bit}} + E_{S_{bit}}) + (\gamma_{ij} - 1) \times E_{L_{bit}})
\end{aligned}
\tag{4.1}
$$

where $E_{B_{bit}}$, $E_{S_{bit}}$ and $E_{L_{bit}}$ are the energy consumed for transferring one bit data by the buffer, switch and link respectively, and $\gamma_{ij} = |\Omega_{ij}|$ is the length of a routing path Ω_{ij} from a source router r_i to a destination router r_j (see Section 3.3.4).

4.1.2 Delay Model

Besides the energy model, a delay model is developed to define the delay for transmitting a packet along a routing path Ω_{ij}. The delay of transferring one data from the source tile to the destination tile is affected not only by the length of communication path, but also by the network traffic situation. When the network is in congestion, all flits have to wait for a period of time at the router until resources are released by other transmissions. Because of the concurrent nature of network traffic, it is difficult to make an exact calculation of the delay at a specific router on the routing path which is resulted from the network congestion. Hence, we formulate the transmission delay using an average delay, D_{con}. The delay for transferring one flit from a source tile t_i (or router r_i) to a destination tile t_j (or router r_j) is then formulated as follows:

$$
D_{flit}^{t_i t_j} = \gamma_{ij} \times \left(D_{S_{flit}} + D_{con}\right) + (\gamma_{ij} - 1) \times D_{L_{flit}} \tag{4.2}
$$

where $D_{S_{flit}}$ represents the delay of switching one flit at a router. $D_{L_{flit}}$ represents the delay of forwarding one flit across a link. The total delay of communication communication c_{ij} from tile t_i to tile t_j is derived as:

$$
\begin{aligned}
D^{t_i t_j} = &\frac{vol_{ij}}{F} \times \gamma_{ij} \times \left(D_{S_{flit}} + D_{con}\right) \\
&+ \frac{vol_{ij}}{F} \times (\gamma_{ij} - 1) \times D_{L_{flit}}
\end{aligned}
\tag{4.3}
$$

where F is a constant and dependent on the specific NoC. F is dependent on the packet size and the switching technique utilized in the NoC.

It is shown in Equation 4.1 and 4.3 that, given the NoC-specific constants (i.e., $E_{B_{bit}}$, $E_{S_{bit}}$, $E_{L_{bit}}$, $D_{S_{flit}}$, D_{con} and $D_{L_{flit}}$), both the communication energy consumption and the delay are proportional to the product of the data volume vol_{ij} and the length of the routing path γ_{ij} between two communicating cores (tasks). Therefore, instead of directly using the energy model

and the delay model, we define the following weighted communication of an application (WCA) as the objective function.

Definition 1: The weighted communication of an application (WCA) is the sum of products of data volume vol_{ij} and the length of the routing path γ_{ij}, for all communicating tasks of the application which are mapped on a NoC.

$$WCA = \sum_{\forall c_{ij}} vol_{ij} \times \gamma_{ij} \tag{4.4}$$

Using the WCA, the different mappings can be relatively evaluated. According to Equation 4.1 and 4.3, a mapping with the minimal WCA will yield the minimized communication energy consumption and packet delay.

4.2 Tree-Model Based Mapping Algorithm

To reach the objective of minimizing the WCA, it is important for us to know the logical relationship between nodes on the NoC. We notice that for any topology, given a specific node, it is possible to abstract the connections with other nodes by traversing the whole network from the given node. Compared to linear data structure such as linked list and one dimensional array, tree structure can be traversed more flexibly and efficiently. Starting from a node on a tree, all its parents and children nodes can be traversed, which makes tree structure suitable for representing the many-to-many connections in a NoC. In practice, tree model has already been widely used in a broad range of applications including pattern classification [109], data searching [110] to parallel computing [111]. Therefore, we utilize the tree model to abstract the connections between nodes in a NoC.

4.2.1 Extended Tree Model of a NoC

To abstract a NoC into a tree, first the center point of the NoC is chosen as the root node of the tree, which has the shortest average distance to all other nodes in the NoC. The neighbors of the center point are put on the tree as the child nodes of the root node. Then, the same traversal iterates for each child node. The procedure continues until all nodes in the NoC are put onto the tree. The structure is called an extended tree since some child nodes may have more than one parent node.

The abstraction of the extended tree model of a 2D mesh NoC is presented in Algorithm 2. Although the algorithm is illustrated with a 2D mesh NoC, we note that the extended tree model and the abstraction algorithm are applicable to NoCs with any kinds of topologies, as long as the starting point is specified.

Algorithm 2: Abstraction Algorithm of Extended Tree Model

Input : A 2D mesh of size $x \times y$
Output: A extended tree structure representing the 2D mesh

1 Select the starting point whose coordinates is $\left(\frac{x}{2}, \frac{y}{2}\right)$;
2 Create the initial Mapping Tree (MT), MT = $\{t_k\}$, t_k is defined as the root of the tree;
3 Create set B = $\{t_k\}$;
4 Traverse the NoC from node $t_k \in B$ clockwise, denote the neighboring nodes of t_k in all directions, i.e., t_{kw}, t_{kn}, t_{ke} and t_{ks} respectively, as the child nodes of the node t_k; Conversely, denote the node t_k as the father node of nodes t_{kw}, t_{kn}, t_{ke} and t_{ks}.;
5 Create level set L = $\{t_{kw}, t_{kn}, t_{ke}, t_{ks}\}$, append L to MT, B = L and then empty L;
6 Traverse the NoC from each node $t_i \in B$ sequentially, find all children nodes of $t_i \in B$, append these children nodes to L if it is not in L yet. When all $t_i \in B$ has been accessed, append L to MT, B = L and then empty L;
7 Repeat 6 until all nodes of the NoC are appended in MT;

Figure 4.1 is an example where a 3×3 2D mesh NoC is abstracted into an extended tree using Algorithm 2. The traversal starts from the center point in order to:

1. Make the tree as flat as possible. As a result, the length of the routing path from the root to any leaf nodes can be minimized as much as possible.

2. Obtain a tree as symmetrical as possible so that each node located at the interior levels has the most number of child nodes. Consequently, it is possible to map more tasks communicating with a given task on its neighbor nodes to achieve smaller WCA.

The level of the extended tree implies some characteristics of the nodes at that level:

1. Resource quantity. If we refer to the child nodes of a node as its resources, the root node and the nodes at interior levels have more resources than those leaf nodes. In Figure 4.1 the resource quantity of nodes at the root, 1st and 2nd level is 4, 2 and 0 respectively.

2. Communication distance. Different levels on the extended tree imply different lengths of routing path from nodes located at that level to nodes at other levels. For example, in Figure 4.1 the average length

(in hops) from any node at the 2nd level to root node is 2, while that from a node at the 1st level is 1.

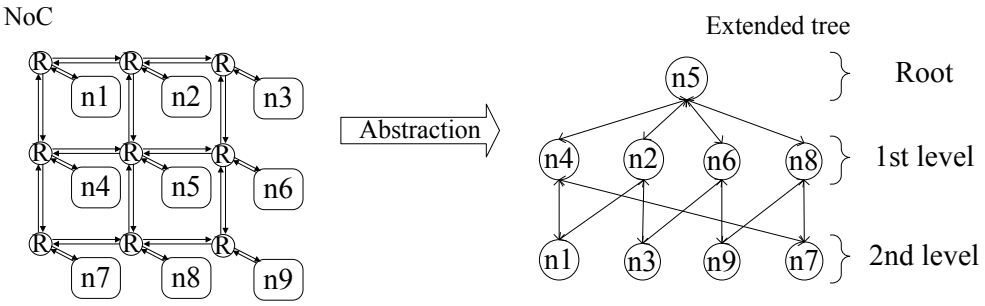


Figure 4.1: Extended Tree Abstraction of a 2D Mesh

Due to these two characteristics, it is reasonable for mapping algorithm to map a task on a node at a level as low as possible. In such a way, the newly mapped task gets more child nodes with shortest communication distance (1 hop) on the NoC, to map tasks that communicate with it. Additionally, this task can communicate with those tasks that have already been mapped on the tree through shorter communication paths. Both of them will contribute to the minimization of the WCA.

4.2.2 Mapping Tasks on Extended Tree

After the NoC is abstracted as an extended tree, we can use the tree to map the tasks of an application. The mapping algorithm takes advantage of the features implied by the levels on the tree and try to map a selected task on a node at lowest level.

To present the task mapping algorithm, we first introduce the following definitions that are used in the tree-model based task mapping algorithm.

Definition 2: A Partial Tree (PT) is a partial solution of mapping an application where a part of tasks in the task graph have been mapped on the extended tree of the NoC. The partial tree $PT =< T^{'}, C^{'} >$ is a subset of the TG. That is, $T^{'} \subseteq T$, $C^{'} \subseteq C$ and each $t_i \in T^{'}$ has already been mapped on the tree.

Definition 3: The Communication Volume (CV) of task $t_j \in T$ is the aggregate communication data volume between t_j and all its parent tasks and child tasks in T. The CV of t_j is calculated as follows:

$$CV_{t_j} = \sum_{\forall t_p \in T} \left(vol_{t_p,t_j}\right) + \sum_{\forall t_c \in T} \left(vol_{t_j,t_c}\right) \tag{4.5}$$

where t_p and t_c denote a parent and a child task of t_j respectively, and vol_{t_i,t_j} is the data volume between tasks t_i and t_j.

Definition 4: The Affinity to Partial Tree (APT) of a task $t_j \in T - T^{'}$ is the communication weight between t_j and its parent and child tasks in $T^{'}$. The APT of a task t_j is calculated as follows:

$$APT_{t_j} = \sum_{\forall t_p \in T^{'}} \left(vol_{t_p,t_j}\right) + \sum_{\forall t_c \in T^{'}} \left(vol_{t_j,t_c}\right) \tag{4.6}$$

where t_p and t_c denote a parent and a child task for t_j on the PT respectively.

Using these definitions, the tree-model based mapping algorithm is given in Algorithm 3.

Algorithm 3: Tree-Model-Based Task Mapping Algorithm

Input : TG, abstracted tree ET of a NoC
Output: Mapping solution M

```
1  Calculate CV for all tasks t_i ∈ T;
2  Select the task t_b with largest CV, map this node onto root node n_r
   of ET, create T' = {t_b} and M = {< n_r, t_b >}, remove t_b from T;
3  while T is not empty do
4      Calculate APT_{t_k} for all t_k ∈ T;
5      Select task t_j with largest APT_{t_j} as the task to be mapped;
6      Set the WCA_min the WCA of the PT WCA_PT;
7      for each node n_i in the T' do
8          for each child node n_c of the node n_i do
9              map task t_j on node n_c;
10             calculate the WCA of the PT WCA_PT;
11             if WCA_PT < WCA_min then
12                 WCA_min = WCA_PT;
13                 set the objective node b_obj the node n_c;
14     Map t_j onto n_obj, append pair < n_obj, t_j > into M;
15     Append t_j into T', remove t_j from T;
```

At the beginning of the mapping, the task with the largest CV in the TG is selected and mapped on the root node of the extended tree. Thereafter, the task with the largest APT is selected. The algorithm attempts to map the selected task on a free child node of those nodes on which one task has already been mapped. After each mapping trial, the WCA of the PT will be calculated and compared with the minimal one achieved so far in previous trials. Finally, the node which achieves the minimal WCA of the PT will be chosen to map the selected task. For an application composed of n tasks, the process iterates n times until all tasks are mapped on the nodes in the extended tree, which is less than $\frac{n\times(n-1)}{2}$ times in the case of

the GI algorithm presented in Section 3.5.4. The final mapping solution minimizes the WCA using the APT method and consequently reduces the energy consumption and packet delay. Figure 4.2 is an example of using the tree-model based algorithm to map an application on an extended tree of a 3×3 NoC.

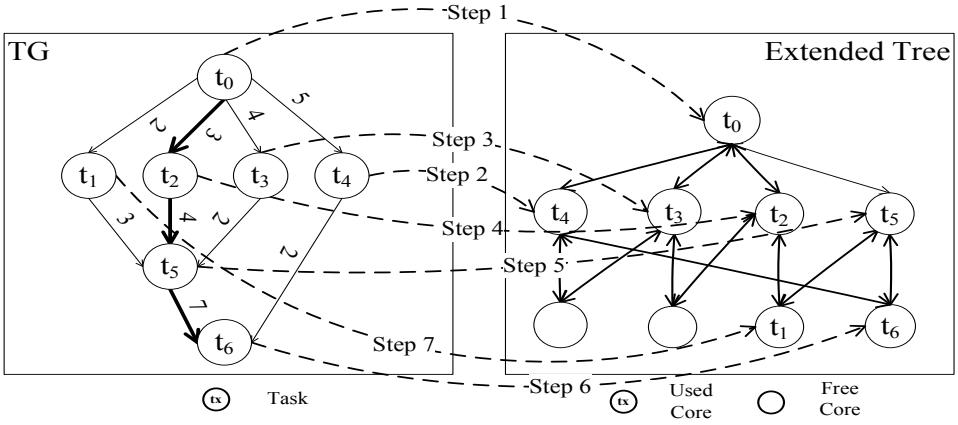


Figure 4.2: Task Mapping Using Extended Tree of a NoC

4.3 Experimental Evaluation

4.3.1 Experiment Setup

The efficiency of the tree-model based algorithm is compared with the GI algorithm in terms of run-time of algorithm, average network latency (ANL), energy consumption (E) and WCA, using the benchmark applications from the SPLASH-2 application suite [112].

Full system simulation is performed to gain further insight into the performance of mapping algorithms. A multiprocessor simulator [113] is used as our simulation platform to gather application traffic traces and original mapping. Links and routers are modeled accurately in the simulation. The wormhole router includes a routing computation unit, a virtual channel allocator, a switch allocator, a crossbar switch and four input buffers. X-Y deterministic routing is used to avoid deadlocks. Two applications are selected from the SPLASH-2 suite: Fast Fourier Transform (FFT) and Radix. The original traffic data of Radix is used to compare the run-time of the tree-model based and the GI algorithm with NoC size variation from 3×3 to 8×8.

Another cycle-accurate NoC simulator Noxim [114] is used to simulate the application traffic on the NoC and produce detailed results of both algorithms. The ANL, E and WCA under the original random mapping, tree-

model based and GI mapping algorithms are compared using experiment data gained from Noxim simulator.

4.3.2 Results Analysis

Both algorithm complexity and NoC size affect the run-time of algorithm. Figure 4.3 shows that a significant reduction of run-time is achieved by tree-model based algorithm. In the experiment, the only case that the GI is faster than the tree-model based algorithm occurs in the 3×3 NoC. In other cases, the run-time of the tree-model based mapping is only 10% of the GI on average. Run-time comparison shows that the speed-up of the tree-model based algorithm over the GI is up to 92 times when the NoC size grows from 4×4 to 8×8. Moreover, the run-time of tree-model base algorithm increases only slightly when NoC size increasing. The run-time of the tree-model based mapping in a 8×8 NoC is about 12 times longer than that in a 3×3 NoC, in comparison to 1504 times of the GI mapping. The speed-up achievement means that the tree-model based algorithm is more efficient and applicable if considering the increasing NoC size as well as the tighter run-time constraints in the real-time systems.

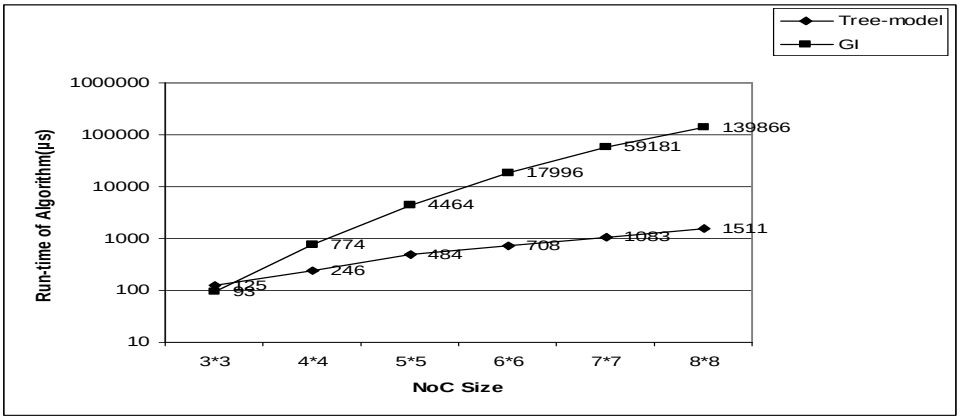


Figure 4.3: Comparison of Run-time of Tree-model Based and GI algorithm

We use a configuration of 5×5 mesh NoC with 25 cores which models a chip multiprocessor (CMP) for our experiments. Each PE has a core, a private L1 cache and a shared L2 cache bank. Memory controllers are connected to the top and bottom side of the chip. The static non-uniform Cache Architecture (NUCA) [115] is implemented in our memory/cache architecture, in which data are mapped to cache banks statically. Compared with the uniform cache architecture (UCA), which has been used in traditional commercial multi-core processors, NUCA has more flexible cache access latencies and thus improves the system performance. The reason is

that in UCA, the unified cache access latency is determined by the worst case wire delay. A two-level distributed cache coherence protocol known as MESI [116] is used in our memory hierarchy in which each L2 bank has its own directory. Four types of cache line status, namely Modified (M), Exclusive (E), Shared (S) and Invalid (I) have been implemented.

We evaluated ANL, E and WCA under different mapping algorithms. The detailed configurations of processor, cache and memory configurations can be found in Table 4.1. Results are presented for FFT and Radix workloads with 16 threads.

Table 4.1: System configuration parameters

Processor configuration	
Number of processors	25 in-order
Issue width	1
Cache configuration	
L1 cache	Private, split instruction and data cache, each cache is 16KB. 4-way associative, 64-bit line, 3-cycle latency.
L2 cache	Shared, unified 12.5MB (25 banks, each 512KB). 64-bit line, 6-cycle latency.
Cache coherence protocol	MESI
Cache hierarchy	Static NUCA
Memory configuration	
Size	4GB DRAM
Access latency	260 cycles
Controllers	10
Requests per processor	16 outstanding
Network configuration	
Routing algorithm	X-Y deterministic
Router scheme	Wormhole
Flit size	128 bits
Router and link latency	1 cycle

Figure 4.4 shows the results of ANL, E and WC on FFT and Radix in which three mapping strategies are evaluated. As anticipated, the original unoptimized mapping has the worst latency and energy values, due to the higher weighted communication. It is shown that in FFT traffic, compared with the original mapping, the network latency, energy and weighted communication of the tree-model mapping are reduced by 21.3%, 21.5% and

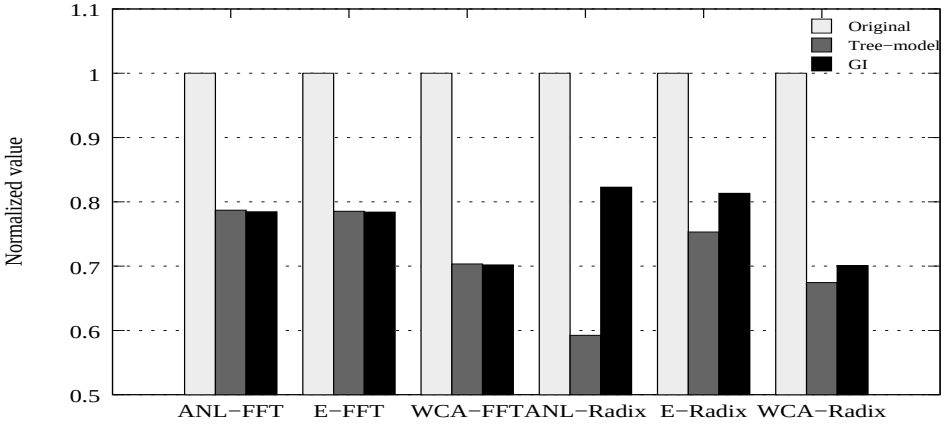


Figure 4.4: Comparison of normalized ANL, E and WCA for FFT and Radix benchmarks over different mapping strategies

29.7%, respectively. The GI algorithm shows similar results, with reduction of 21.5%, 21.6% and 29.8% respectively. The proposed tree-model based mapping has comparable performance with the GI, but its complexity is much lower (as shown in Figure 4.3). For Radix traffic, as shown in Figure 4.4, the GI and the tree-model mapping outperform the original mapping in all aspects. Comparing with the original mapping, the average network latency of the tree mapping and the GI are reduced by 40.8% and 17.3% respectively; energy consumption of the tree-model mapping and the GI are reduced by 24.7% and 18.7% respectively; weighted communication of the tree-model mapping and the GI are reduced by 32.6% and 29.9% respectively. It is noteworthy that tree-model mapping performed much better than the GI when the traffic becomes heavier in Radix than FFT application. The ANL reduction of the tree-model based algorithm is significantly greater than that of the GI because under the GI strategy the task with the largest CV is mapped on node 6 instead of node 12 which is the center node of the NoC. Therefore, the ANL is increased because of unbalanced traffic on network.

4.4 Chapter Summary

A novel mapping algorithm based on the extended tree model of NoC was presented and evaluated. Experimental results show that the proposed algorithm achieves lower energy consumption and average network delay. Compared to the GI algorithm, the tree-model based algorithm finds comparably better mappings using significantly less run-time, especially in the cases of large NoC sizes. The improvements on system performance as well as the reduction of algorithm run-time make the tree-model based algorithm a viable

mapping algorithm, which can be used solely as a energy- and time-efficient mapping algorithm, or jointly with other mapping algorithms to find higher quality mappings as we will presented in the following chapters.

Chapter 5

Simulated Annealing for Global Optimum

Compared to heuristics like branch-and-bound (BB), and greedy incremental (GI), the major advantage of simulated annealing algorithm is that it is able to find the globally optimum mapping. In addition, SA algorithm provides better trade-off between complexity and performance in comparison with the other algorithms presented in Chapter 3. SA has drawn more and more attention for application mapping problems on many-core NoCs [3] [87] [117]. As presented in Section 3.5.5, to apply SA algorithm, a set of parameters and functions has to be specified, including *initial temperature*, *final temperature*, *cooling ratio*, *temperature function*, *move function*, *accept function*, etc. These parameters and functions determine how closely and how quickly SA can converge to the global optimum of the objective function. However, there are no straightforward ways to specify these parameters. In previous works, these parameters are set either by using randomly selected values [3] [87], or parameters tailored to the particular problems [117]. Clearly, it is not certain if these random or tailored values can be generally applicable to all other problems. Additionally, the performance of SA algorithm cannot be guaranteed by randomly selected parameters.

For the parameters used in the SA algorithm, we argue that:

1. The parameters of SA are problem-specific. The annealing process of SA algorithm is controlled by the set of functions which are determined by the set of parameters. Various sets of parameters will result in different annealing processes and convergency to the optimal solution. The parameters have to be selected with respect to the specific problem.
2. The parameters of SA have a joint impact on the performance of SA. This means that these parameters should be selected systematically, instead of being set randomly and independently.

Based on these two considerations, a systematic method is necessary for generating the set of parameters of SA algorithm for the application mapping problem. In this chapter, we propose to use the *Nelder-Mead simplex method*, which is originally introduced in [118], to automatically generate the optimized parameters of SA algorithm. The generated parameters are applied to the SA algorithm to find the optimized mapping solution which achieves minimized communication energy consumption on a many-core NoC. Due to the utilization of the optimized parameters, the annealing process of the SA algorithm is significantly shortened while keeping good mapping quality.

5.1 Nelder-Mead Simplex Method

The purpose of the Nelder-Mead simplex method is to minimize the output of a function $f(p)$ with n variables $x_1, x_2, \ldots, x_n$. In this method, a number of $n+1$ points (solutions) $p_0, p_1, \ldots, p_n$ are originally selected and form the so-called simplex. The set of points is then used to generate a new and better point which will replace the worst point in current simplex and forms a new simplex. Each point of the simplex is a n-tuple with n variables, i.e., $p_k = \left(x_1^k, x_1^k, \ldots, x_n^k\right)$. The Nelder-Mead simplex method compares the $n+1$ values of the function $f(p)$ $(f(p_i)\,(0 \leq i \leq n))$, and replaces the point with largest value by the newly generated point. In each iteration, the replacement is realized by three operations: *reflection, expansion* and *contraction*. If the replacement fails through these three operations, all points forming the simplex are updated with new values to generate a new simplex. The general Nelder-Mead simplex method is presented in Algorithm 2.

As shown in Algorithm 2, the principle of the Nelder-Mead simplex method is, if $f(p_r) \leq f(p_{n-1})$, then the point p_n is replaced by its reflection point p_r. Thereafter, if $f(p_r) < f(p_0)$, the reflection point is expanded to the expansion point p_e and the point p_n is replaced by p_e. The procedure restarts when the expansion is done. In the case if $f(p_r) > f(p_{n-1})$, the contraction point p_c is generated. If $f(p_c) < min\,(f(p_r), f(p_n))$, the point p_n is replaced by contraction point p_c. Otherwise, all points in current simplex are updated by $p_j = (p_j + p_0)/2 (j = 0, 1, \ldots, n)$ and a new simplex is generated. The process iterates with the above operations.

By continuously replacing the point p_n with a point which achieves smaller $f(p)$, the value of the function $f(p)$ converges to the minimum. The process terminates when the function *stop*() becomes true. The state of function *stop*() can be determined by whether the value of function $f(p)$ has converged to a final value [118], or whether the points forming the simplex have already converged to a final point [119]. In this work, because we try to find the optimized parameters for the SA algorithm, we adopt the latter

Algorithm 4: Nelder-Mead Simplex Method for Minimizing $f(p)$

```
1 !ht
2 [118]
  Select the initial n + 1 points p_i (0 ≤ i ≤ n).
  while (!stop()) do
    Sort f(p_i) (0 ≤ i ≤ n) such that
    f(p_0) ≤ f(p_1) ≤ ··· ≤ f(p_{n-1}) ≤ f(p_n).
    Let p̄ = Σ_{i=0}^{n-1} p_i/n.
    Generate reflection point p_r = p̄ + α ∗ (p̄ − p_n).
    if f(p_r) ≤ f(p_{n-1})) then
      Replace p_n by p_r.
      Generate expansion point p_e = p̄ + β ∗ (p_r − p̄).
      if (f(p_r) < f(p_0)) ∧ (f(p_e) < f(p_r)) then
        Replace p_n by p_e.
      end if
    else
      Let f(p*) = min(f(p_r), f(p_n)).
      Generate contraction point p_c = p̄ + γ ∗ (p* − p̄).
      if f(p_c) ≤ f(p*) then
        Replace p_n by p_c.
      else
        Update p_j with (p_j + p_0)/2 for j = 0, 1, ..., n.
      end if
    end if
  end while
  Return the point p_0.
```

way to define the function *stop*(). More precisely, in Algorithm 2, *stop*() becomes true when $|x_i^k - x_j^k| \leq \varepsilon^k (i \neq j)$, for all i, j and k, where x_i^k and x_j^k are the kth element of point p_i and p_j respectively. Each element of vector ε, called *convergence degree* of variable x, is a predefined small positive value which determines the magnitude of the convergence.

In Algorithm 2, reflection coefficient α, expansion coefficient β and contraction coefficient γ give the factors with which the new simplex is generated by reflection, expansion and contraction operation respectively. These coefficients decide the speed of the convergence and the quality of the final point. In [118] and [119], two sets of α, β and γ were used. In this work, we evaluated both sets of values by applying them in the Nelder-Mead simplex method for the same set of benchmarks. The results show that both sets of parameters achieve comparable performance of the SA algorithm, but the Nelder-Mead simplex method using the coefficients in [119] can converge to the final point with 100 times less CPU time than that in [118]. Therefore, we use 1/3, 2.0 and 1.5 for α, β and γ respectively, which were used in [119].

5.2 Parameter-Optimized Simulated Annealing

This section presents the SA algorithm which applies the parameters generated by the Nelder-Mead simplex method. The set of parameters and functions used for implementing the SA algorithm is defined first.

5.2.1 Parameters and Functions in SA

Cost Function

The SA algorithm is adopted in the tile assignment stage (Section 3.3.3). The objective of the mapping is to minimize the communication energy consumption on a many-core NoC. Similar to the mapping algorithm presented in Chapter 4, we use the weighted communication of an application (WCA) defined in Equation 4.4 as the objective function $Cost(S)$ in SA. As verified in Chapter 4, a mapping with the minimal WCA results in the minimized communication energy consumption.

$$WCA = \sum_{\forall c_{ij}} vol_{ij} \times \gamma_{ij} \tag{5.1}$$

where vol_{ij} and γ_{ij} are the data volume and the length of the routing path for the communication c_{ij}.

Annealing Schedule: $Temp(i)$ Function

The annealing schedule determines how the temperature is cooling down. At each step of annealing, a new temperature is generated by temperature

function $Temp(i)$. We choose the geometric annealing schedule presented in [31] as the temperature function.

$$Temp(i) = T_0 \times q^{\lfloor \frac{i}{L} \rfloor} \tag{5.2}$$

The new temperature is decided by the initial temperature T_0, the cooling ratio q, the accumulated number of iterations i and the number of iterations at each temperature L.

Number of Iterations L

The number of iterations at each temperature L is set as $M(N-1)$, where M and N are the number of cores in the CCG (Section 3.2.2) and that of tiles in the many-core NoC respectively.

Acceptance Function: $Accept(\Delta C, T)$

While an downhill move ($\Delta C < 0$) is always accepted, the function $Accept(\Delta C, T)$ determines whether a uphill move ($\Delta C > 0$) should be accepted or not at the temperature T. The normalized inverse exponential form is chosen to implement the acceptance function in this work.

$$\begin{aligned} Accept(\Delta C, T) = True \Leftrightarrow random() < p \\ p = \frac{1}{1 + \exp\left(\frac{\Delta C}{KC_0T}\right)} \end{aligned} \tag{5.3}$$

With this acceptance function, the possibility of accepting a uphill move, p, is less than 50%. On the basis of the original normalized inverse exponential form presented in [31], we add the normalizing ratio K in the acceptance function which works together with the initial cost C_0 to normalize the cost difference ΔC. This comes from the observation that using the original normalized inverse exponential form , in cases that the C_0 is huge, an accepting possibility close to 50% will be generated even for a very small ΔC at a very low temperature. This makes SA inefficient at the last rounds of lower temperatures.

Initial and Final Temperature

The acceptance function in (5.3) defines the relation between the accepting possibility p, cost difference ΔC and temperature T. Equation (5.3) can be solved with respect to T as follows:

$$T = \frac{\Delta C}{\ln(\frac{1}{p} - 1)} \tag{5.4}$$

If we define P_s the possibility of accepting the maximal ΔC at initial temperature T_0, and P_f the possibility of accepting the minimal ΔC at final temperature T_f, then the initial and final temperature can be calculated as follows:

$$T_0 = \frac{\Delta C_{max}}{\ln(\frac{1}{P_0} - 1)}, T_f = \frac{\Delta C_{min}}{\ln(\frac{1}{P_f} - 1)} \tag{5.5}$$

When T_0 and T_f are set manually using randomly selected values (the cases in [3] [87]), only a numerical range is given by T_0 and T_f, and there are no real-world meanings behind T_0 and T_f. In this work, by contrast, the usage of P_s and P_f is more meaningful and understandable for designers to choose the T_0 and T_f by Equation (5.5).

Move function:$Move(S, T)$

We use the random swapping as the move function. An IP core in the current mapping is randomly selected and the mapping tile of the IP core is swapped with that of another randomly selected IP core.

Termination function:$Terminate(i, R)$

We add one criteria $N_{\Delta C=0}$ into the termination function of coupled temperature and rejection threshold which is presented in [31], to determine the stopping condition of the SA algorithm.

$$\begin{aligned} Terminate(i, R) = True \Leftrightarrow (Temp(i) < T_f \wedge R \geq R_{max}) \\ \vee (N_{\Delta C=0} = Z) \end{aligned} \tag{5.6}$$

$N_{\Delta C=0}$ stands for the number of consecutive temperatures at which the lowest cost C_{best} has not been changed. Z is the maximal number of $N_{\Delta C=0}$ allowed in the SA algorithm. R is the number of consecutive rejections since last acceptance and R_{max} is the maximal number of rejections allowed in the SA algorithm. With this termination function, the annealing is stopped either when the temperature reaches to or below the final temperature and the moves in last R_{max} iterations are rejected, or in the last Z temperatures, no better solutions have been found. In this work, we set $R_{max} = L$ and $Z = 0.1N_T$, when N_T stands for the total number of temperatures from T_0 to T_f.

Initial Mapping

A random mapping in which each IP core is randomly mapped on a tile, is generated as the initial mapping.

Summary of Parameters

Table 5.1 summarizes the parameters and functions used in the SA algorithm in this work. We can see from Table 5.1, to apply the SA algorithm, the values of 6 parameters need to be specified: $q, K, P_s, P_f, \Delta C_{max}$ and ΔC_{min}. As long as these parameters are specified, other parameters such as T_0 and T_f can be decided. In these 6 parameters, the values of ΔC_{max} and ΔC_{min} can be obtained from a set of mapping trials generated from the original mapping using the move function (Section 5.3). The other 4 parameters, labeled "Nelder-Mead Simplex Method" in column "Value" in Table 5.1, are the most important parameters of SA in this work and they are going to be optimized by the Nelder-Mead simplex method presented in Section 5.1.

5.2.2 Parameter Optimization

To apply the Nelder-Mead simplex method for optimized values of q, K, P_s and P_f, we need to define the function $f(p)$ and specify the boundaries from which the parameter is chosen, and the convergence degree of each parameter.

Function $f(p)$

Since using various sets of q, K, P_s and P_f, the SA algorithm will produce different minimized values of WCA, we can define the output of the SA, i.e., the minimized WCA, as the function $f(p)$ of variables q, K, P_s and P_f. With this definition, it is possible to use the Nelder-Mead simplex method to find the final point of variables q, K, P_s and P_f which yield the minimum WCA by the SA algorithm.

Parameter Boundaries and Convergence Degrees

Contrary to the work in [118] where the variables is unbounded, the parameters q, K, P_s and P_f in our specific application mapping problem are bounded. Among them, the parameters P_s and P_f are theoretically in the range $(0.0, 0.50]$ according to Equation (5.3). For the SA algorithm, it is reasonable to set a high acceptance possibility at the initial temperature and a low acceptance possibility at the final temperature. In this work, we set the range of P_s and P_f by $[0.20, 0.49]$ and $(0.0, 0.10]$ respectively. And the convergence degrees ε_{P_s} and ε_{P_f} are set 0.01 and 0.005 respectively. The cooling ratio q can be in range $(0.0, 1.0)$. In this work, we set the range $[0.80, 0.99]$ for q. The convergence degree ε_q is set 0.005. The value of K is allowed in the range of $(0.0, 1.0]$ and the convergence degree ε_K is set 0.05.

At the beginning of the Nelder-Mead simplex method, 5 initial points are generated by choosing 4 elements, i.e, q, K, P_s and P_f, from their allowable range. During the process of the Nelder-Mead simplex method, whenever an

Table 5.1: Functions and Parameters for SA

Symbol	Definition	Value
S	Mapping solution (S_0: initial solution)	
$Cost(S)$	Cost function	WCA (Equation (5.1))
$Temp(i)$	Temperature function$i$	$T_0 \times q^{\lfloor \frac{i}{L} \rfloor}$
i	Accumulated number of iterations	
q	Geometric annealing schedule cooling ratio	Nelder-Mead Simplex Method
L	Number of iterations at each temperature	$M(N-1)$
N	Number of tiles in CCRG	
M	Number of tasks in CWG	
$Accept(\Delta C, T)$	Return accept (True) or reject (False) for a worse move	$random() < 1/(1+\exp\frac{\Delta C}{KC_0T})$
K	Normalizing ratio	Nelder-Mead Simplex Method
C_0	Initial cost	$Cost(S_0)$
T_0	Initial temperature	$\Delta C_{max}/\ln(\frac{1}{P_s}-1)$
T_f	Final temperature	$\Delta C_{min}/\ln(\frac{1}{P_f}-1)$
ΔC_{max}	The maximal ΔC at initial temperature T_0	Experiment
ΔC_{min}	The maximal ΔC at final temperature T_f	Experiment
P_0	The possibility of accepting the maximal ΔC at initial temperature T_0	Nelder-Mead Simple Method
P_f	The possibility of accepting the minimal ΔC at final temperature T_f	Nelder-Mead Simplex Method
$Move(S, T)$	Return a neighboring mapping of S	
$Terminate(i, R)$	Return terminate (True) or continue (False)	$Temp(i) < T_f \wedge R \geq R_{max} \vee N_{\Delta C=0} = Z$
R	Number of rejections	
R_{max}	Allowed maximal number of rejections	L
N_T	The total number of temperatures from T_0 to T_f	$\ln(\frac{T_0}{T_f})/\ln(q)$
Z	The allowed maximal number of temperatures with $\Delta C = 0$	$0.1N_T$

element of a point exceeds its boundary, the boundary value is used for the element. The function *stop*() becomes true and the process is terminated when these 5 points converge to one point.

5.2.3 Parameter-Optimized Simulated Annealing Algorithm

Algorithm 5: Parameter-Optimized Simulate Annealing

1 Define the boundaries and convergence degree ε for parameters q, K, P_s and P_f.
2 Obtain the final point of the Nelder-Mead simplex method, $p_{opt} = simplex()$.
3 Set $q_{opt} = p_{opt}.q, K_{opt} = p_{opt}.K$.
4 Set $P_{s_{opt}} = p_{opt}.P_s, P_{f_{opt}} = p_{opt}.P_f$.
5 Find the best solution by applying the final point to SA, $S_{best} = sa(q_{opt}, K_{opt}, P_{s_{opt}}, P_{f_{opt}})$.
6 Return S_{best}.

Applying the Nelder-Mead simplex method, we develop the parameter-optimized simulated annealing algorithm for application mapping problems on many-core NoCs. The proposed algorithm is described in Algorithm 5 where the functions $sa()$ and $simplex()$ correspond to Algorithm 1 and 2 respectively. After defining the boundaries and convergence degree for four target parameters, i.e., q, K, P_s and P_f, the optimized set of parameters is obtained by the Nelder-Mead simplex method. The mapping solution with minimized WCA is then found by running the SA algorithm with the optimized set of parameters.

Since there are no limitations on the variables and objective functions applied in the Nelder-Mead method, the Algorithm 5 is applicable for obtaining different sets of optimized parameters corresponding to different implementations of the SA algorithm. This makes the proposed method applicable for selecting optimized parameters in diverse application mapping problems.

5.3 Experimental Evaluation

To evaluate the efficiency of the proposed parameter-optimized simulated annealing (POSA) algorithm, we experiment POSA with a set of benchmarks and compare it against the implementation of the SA algorithm in [3].

5.3.1 Experiment Setup

The implementation of the SA algorithm in [3] is available in the NoCmap project [120]. In the NoCmap, the geometric annealing schedule is used and the q is set 0.9. T_0 is fixed to 100 and the final temperature T_f is unbounded. The objective of the NoCmap is to minimized the total communication energy consumption and the energy model presented in [3] is

adopted in the simulator. In this work, we also use the NoCmap simulator to obtain the communication energy consumption of the mappings generated by the POSA algorithm.

Four benchmark applications are selected for the comparison, including a video object plane decoder (VOPD) and a MPEG4 from SUNMAP [121], a multimedia systems application (MMS) [26] and a H.264 decoder (H264) [122]. The CCGs of these applications are derived from original descriptions in these works. The benchmarks and corresponding NoCs used in this work are summarized in Table 6.1.

The mapping of each benchmark is performed by the NoCmap and the POSA algorithms. The communication energy consumptions of both mappings are produced by the NoCmap simulator. For POSA, the average ΔC_{max} and ΔC_{min} are obtained from $5 * L$ move trials starting from the original random mapping, which are used to calculate the T_0 and T_f with given parameters P_0, P_f, C_0 and K. Both algorithms were executed on a Desktop PC with 3.0 GHz Intel Core2 Duo CPU and 8.0 GB of memory to evaluate the run-time overhead.

5.3.2 Results Analysis

Optimized Parameters

Table 5.2: Optimized Parameters of SA for Benchmarks

Benchmark	Cores	NoC	q	P_0	P_f	K
VOPD	16	4x4	0.91	0.44	0.05	0.72
MPEG4	12	3x4	0.95	0.34	0.05	0.36
MMS	25	5x5	0.94	0.36	0.05	0.62
H264	16	4x4	0.89	0.42	0.05	0.49

Applying the POSA algorithm (Algorithm 5), the optimized mapping solution with minimized WCA of each application is found. At the same time, the optimized parameters of the SA algorithm are obtained. Table 5.2 lists the optimized parameters of SA for mapping the four benchmarks. It is observed that, instead of using an identical set of parameters, to find an optimized mapping, different parameters should be used in SA for mapping different applications.

Iterations and Runtime

Table 5.3 shows the iterations (I_s) of NoCmap and POSA algorithm for finding the final mapping solution of each application. The column T_0 and

T_f are the initial and final temperature respectively. T_t refers to the temperature at which SA terminates. I_s is the number of iterations that the SA algorithm has run until the termination. Column *pct* presents the percentage of the iterations of POSA to that of NoCmap. We can see that, since optimized parameters are applied, a much lower initial temperature is set in POSA. As a result, POSA uses significantly smaller number of iterations which is on average less than 1% of that used in NoCmap, to get the final mapping.

Table 5.3: Iterations of SA for Benchmarks

Benchmark	T_0		T_f		T_t		I_s		
	NoCmap	POSA	NoCmap	POSA	NoCmap	POSA	NoCmap	POSA	pct
VOPD	100	2.69	-	1.35e-4	2.28e-6	9.88e-5	4.30e6	2.74e4	0.64%
MPEG4	100	1.90	-	1.26e-4	5.80e-7	8.38e-5	2.61e6	2.77e4	1.06%
MMS	100	1.36	-	1.26e-5	5.80e-7	1.25e-5	1.14e7	1.18e5	1.04%
H.264	100	3.11	-	1.94e-4	0.15	1.43e-4	1.61e6	1.94e4	1.02%

Table 5.4 shows the runtimes of SA in NoCmap and POSA (in seconds) and the speedup achieved by POSA. POSA is, on average, 1.41 times faster than NoCmap. Note that, the runtime of POSA includes the time consumed by the Nelder-Mead simplex method in which the SA is run more than hundred times. In terms of the runtime of a single run of SA, a large speedup is achieved by POSA due to less evaluating iterations. In Table 5.4, $POSA^{'}$ and *Speedup*2 represent the runtime of a single run of SA applying the set of optimized parameters, and the speedup over the NoCmap respectively. We can see that in POSA, the SA with optimized parameters is on average 237 times faster than that in NoCmap. This indicates the great impact of optimized parameters on the runtime of SA algorithm.

Table 5.4: Runtimes and Speedup for Benchmarks

Benchmark	NoCmap	POSA	Speedup1	POSA´	Speedup2
VOPD	31.69	15.50	2.04	0.087	364
MPEG4	15.74	9.67	1.63	0.059	267
MMS	171.74	181.75	0.94	1.17	147
H.264	12.34	11.90	1.04	0.072	171
Average	-	-	1.41	-	237

WCA and Energy Consumption

In this work, minimizing communication energy consumption on NoC is the objective of applying SA for the tile assignment. Figure 5.1 shows the WCA achieved by NoCmap and POSA for each application respectively. The

results of both algorithms vary slightly. The maximum of WCA variance is less than 4% in the case of application H.264.

As anticipated from the results of the minimized WCA, the communication energy consumptions achieved by NoCmap and POSA are almost same. Figure 5.2 shows that the maximal difference exists again in the case of application H.264, which is less than 2%. The comparable energy consumption verify the efficiency of the proposed POSA algorithm. Although a significantly smaller number of iterations is conducted, POSA still can find the optimized mapping solutions which are comparably good with that found by NoCmap in which more iterations have to be performed.

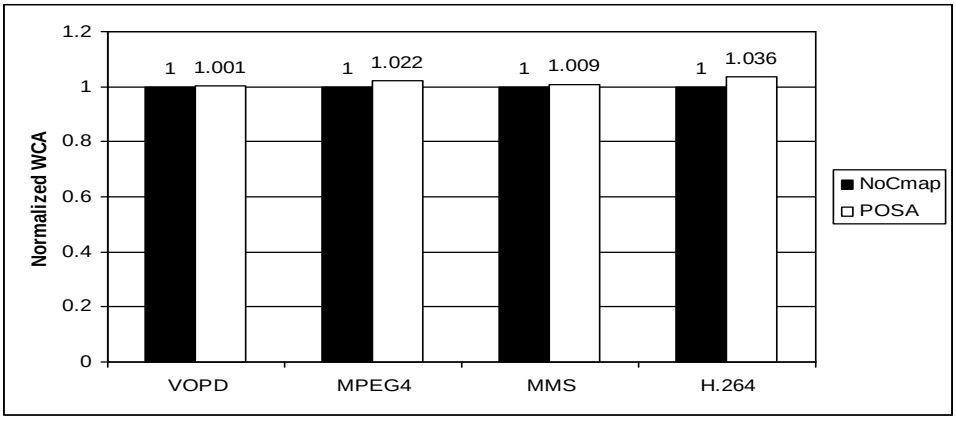


Figure 5.1: Comparison of WCA

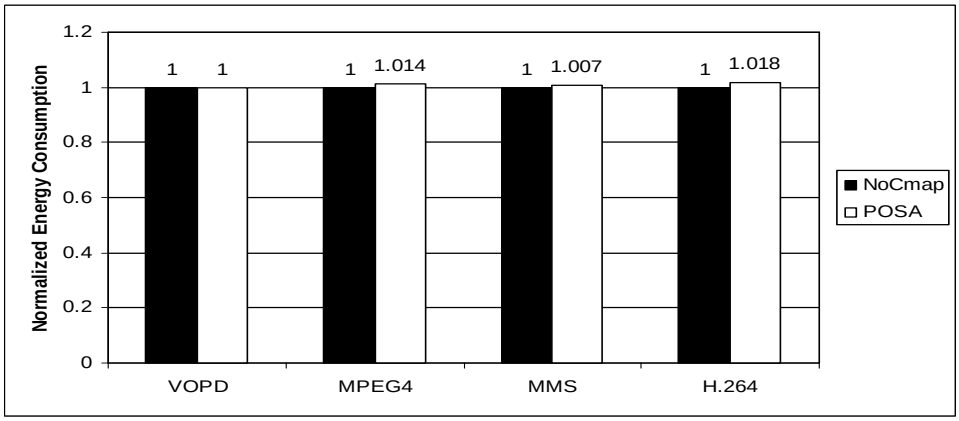


Figure 5.2: Evaluation of Energy Consumption

5.4 Chapter Summary

The set of parameters applied in the SA algorithm has great impact on the runtime and the quality of the final solution. A method to systematically select the parameters of the SA algorithm for the application mapping problem was proposed in this chapter. The Nelder-Mead simplex method, which is used to get the minimization of a function of n variables, was applied to find the optimized parameters of the SA algorithm. With the set of optimized parameters, less evaluation iterations are needed and the SA algorithm is accelerated. The experiment showed that the proposed POSA algorithm only uses on average less than 1% iterations of that used in the NoCmap algorithm to converge to the final optimized solution. An average speedup of 1.4 times is achieved by POSA over NoCmap. With the optimized parameters, the SA algorithm is 237 times faster than the NoCmap, while the comparably good mappings are still found. Besides the time- and energy-efficiency, the proposed POSA algorithm provides a way to flexibly select various number of parameters with respect to different cost functions for different kinds of mapping problems.

Chapter 6

Accelerating Simulated Annealing

Simulated Annealing algorithm is a promising method to solve application mapping problems on many-core NoCs. One major drawback of SA algorithm is low speed. As shown in Chapter 5, to obtain the global optimum, SA algorithm generally needs hundreds of thousands iterations. Since the annealing process of SA is controlled by a set of functions and parameters, to speed up the general SA algorithm, one feasible way is to apply optimized functions and parameters, instead of using generic or empiric ones. While Chapter 5 focuses on the method of selecting optimized parameters for SA algorithm, this chapter presents a method which shortens the annealing process of the SA algorithm by applying optimized initial mapping and temperature.

It is worth noting that generally the SA algorithm starts the annealing process from a randomly generated initial solution and a high initial temperature T_0 [3]. As a result, the SA algorithm behaves randomly at the beginning stage and lots of iterations are wasted without optimization. The random behavior of SA at high temperatures provides us an opportunity to accelerate SA algorithm. In [89], a cluster-based annealing method was proposed for the application mapping problem. Using the clustering technique, a better initial mapping is obtained and a faster convergence can be achieved. It shows up to 30% reduction of runtime without compromising the solution quality. In [87], instead of using a uniformly random probability when determining the core to be swapped, two probability distribution functions (PDFs) are applied to build a variable grain swapping move function. With the optimized move function, the SA algorithm in [87] is reported to obtain an average runtime speedup of 98.55% while keeping the quality of the mapping solution.

Based on the analysis of the general behavior of SA algorithm, the basic

idea of the accelerating method in this chapter is to start the SA algorithm from an optimized initial mapping obtained by the previously presented tree-model based heuristic in Chapter 4. Then, a corresponding lower initial temperature t_k is derived and applied in the SA algorithm to skip a portion of the random behavior at the beginning stages of a general SA algorithm. Consequently, the convergence to the optimal solution is accelerated without degrading the solution quality. Figure 6.1 shows the workflow of the proposed t_k-SA algorithm. The two highlighted processes are the main contributions of the proposed method.

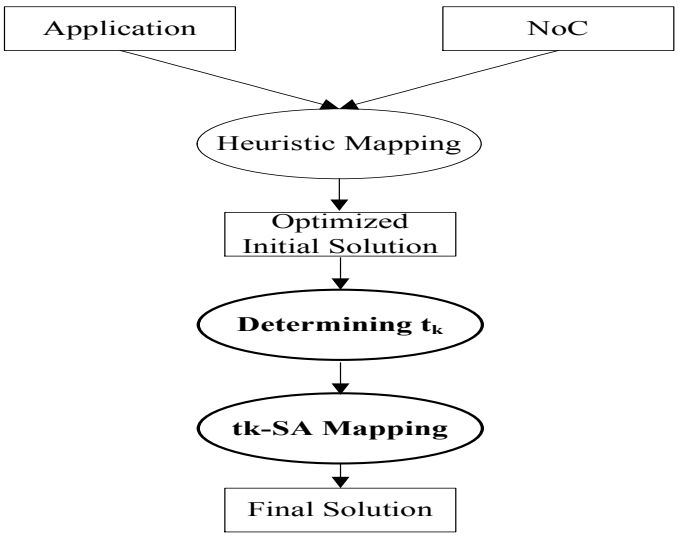


Figure 6.1: The Workflow of the t_k-SA Algorithm

6.1 Simulated Annealing

The SA algorithm uses the same set of parameters and functions as those in Section 5.2 except for the acceptance function.

6.1.1 Acceptance Function: $Accept(\Delta C, T)$

In this work, we add two parameters k and C_0 in the exponential form of the acceptance function presented in [31].

$$Accept(\Delta C, T) = True \Leftrightarrow random() < \frac{1}{\exp\left(\frac{\Delta C}{kC_0T}\right)} \tag{6.1}$$

In this normalized form, the cost difference ΔC is normalized by the parameters $k(0 \leq k \leq 1)$ and the initial cost $C_0 = Cost(S_0)$ (initial mapping

solution s_0). With this acceptance function, the range of the acceptance possibilities is kept in a relevant range even if the cost function changes for different problems. This makes the selecting of the temperature easier than using the initial exponential form. Compared to the acceptance function utilized in Chapter 5, the possibility of accepting an uphill move is in the range of $(0.0, 1.0]$.

6.1.2 Objective Function

The objective of the mapping is to minimize the communication energy consumption in the tile assignment stage (Section 3.3.3). Similar to the SA algorithm (Algorithm 5) in Chapter 5, the weighted communication of an application (WCA) is used as the objective function $Cost(S)$ in the SA algorithm in this chapter.

$$WCA = \sum_{\forall c_{ij}} vol_{ij} \times \gamma_{ij} \tag{6.2}$$

where vol_{ij} and γ_{ij} are the data volume and the length of the routing path for the communication c_{ij}.

6.1.3 Parameters Selection

The method developed in Chapter 5, i.e., the Nelder-Mead simplex method is used to select the set of optimized parameters including the initial temperature T_0, the final temperature T_f, the cooling ratio q and the normalized ratio k. These parameters are employed in the reference SA algorithm with full temperature range from T_0 to T_f (called full-range SA), and also used to derive temperature t_k applied in the proposed t_k-SA algorithm.

6.2 Accelerated Simulated Annealing

The basic idea of the accelerated t_k-SA algorithm is to start the SA algorithm from an optimized solution with a lower cost. Corresponding to the optimized initial solution, an appropriate lower initial temperature t_k is determined. By doing so, the runtime of the SA algorithm is decreased because only a portion of temperatures in the full temperature range of $[T_0, T_f]$ is proceeded. The determination of the temperature t_k is critical to the proposed algorithm in terms of runtime and the quality of the final solution. It is based on the analysis of the typical behavior of the general SA algorithm.

6.2.1 Typical Behavior of SA

SA operates like a pure randomizing optimization at the beginning and a pure greedy optimization at the end. To analyze this typical behavior of the

SA algorithm, we apply the general SA algorithm to map a H.264 decoder (H264) application ([122]). Figure 6.2 depicts the annealing process of the SA algorithm with respect to the temperature t against the mean of WCAs (M_{WCA}) of all accepted solutions at temperature t. SA starts from $T_0 = 7.5$ and terminates at $T_f = 7.8e - 5$ approximately.

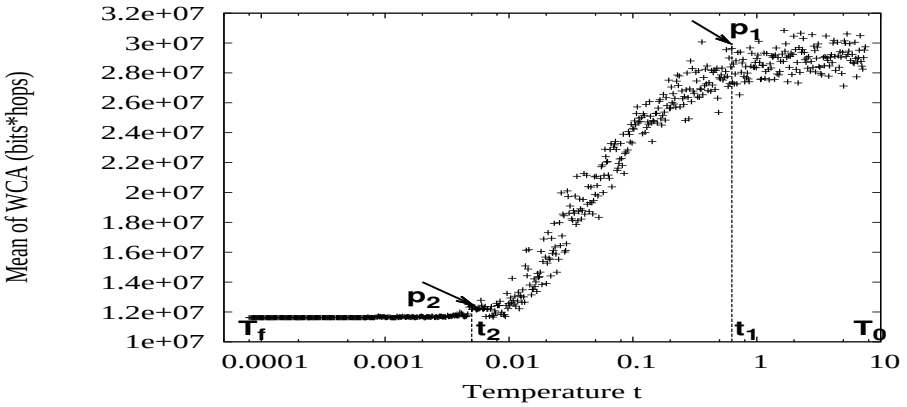


Figure 6.2: Typical Behavior of SA (H264)

As we can see from Figure 6.2, before SA gets to the greedy point (roughly at point P_1), M_{WCA} varies slightly across a wide range of temperatures (from T_0 to nearly 0.65). No obvious optimizations have been achieved in this period since the acceptance possibilities are close to 1 so that almost all moves in this period are accepted. From the point P_1 to point P_2, SA becomes greedy and M_{WCA} is decreased while the temperature falls down. Note that M_{WCA} increases at some points. This illustrates the major feature of SA algorithm that some uphill moves are accepted, which prevents the optimization process from ending with a local minimum. After point P_2, all accepted solutions converge to the optimal one and M_{WCA} remains constant over these temperatures. At the end, SA obtains the best solution with lowest cost and terminates after temperature T_f.

6.2.2 Acceleration Method

We also apply the general SA algorithm to other benchmark applications used in this chapter. The experiments show that the SA algorithm behaves similarly as in Figure 6.2. Based on the analysis of the typical behavior of SA, it is possible to accelerate the annealing process of SA if we can start the SA algorithm from a temperature t_k between t_1 and t_2 (shown in Figure 6.2). Provided that M_{WCA} at temperature t_k in the accelerated SA is comparable to that in the SA algorithm operating in the full temperature range of $[T_0, T_f]$, the process from temperature T_0 to t_k in the full-range

SA can be skipped. This is because the accelerated SA can approximately reproduce the behavior of the full-range SA from temperature t_k to T_f. With the reproduced behavior, the quality of the final solution is maintained by the accelerated SA algorithm.

Based on this consideration, we propose an accelerated SA algorithm, t_k-SA algorithm, for the application mapping problem. In the algorithm, t_k stands for an intermediate temperature between T_0 and T_f. The t_k-SA algorithm starts from temperature t_k with an optimized mapping solution S_{heur}. To make the t_k-SA algorithm faster than the full-range SA algorithm, the runtime of obtaining S_{heur} and t_k must be shorter than that of performing SA from T_0 to t_k. The heuristic algorithm for obtaining S_{heur} and determining t_k are presented in the following sections.

6.2.3 Heuristic for Optimized Initial Solution

The tree-model based mapping algorithm presented in Chapter 4 is applied to get the optimized initial solution S_{heur}. Simply speaking, the mapping algorithm operates in two steps. Firstly, a NoC is abstracted into an extended tree which represents the connections and distances among the tiles on the NoC. Secondly, the tasks of the given application are mapped onto the NoC using the extended tree. Since the extended tree places the network tiles with shorter average distance (to other tiles) onto higher-level tree nodes, it is reasonable to map the tasks in the application with larger communication volumes on nodes at as high level as possible so that the WCA can be minimized.

Because of the usage of the tree-model of the NoC, the mapping algorithm is significantly efficient in terms of algorithm runtime. The experiment shows that the runtime of tree-model based mapping is on average 10% of that of the greedy incremental (GI) algorithm. In addition, the runtime increases slightly while the number of tasks to be mapped increases. The quality of the mapping is comparable to the GI algorithm. These features, especially the efficiency of the runtime, make the tree-model based algorithm suitable for generating an optimized initial mapping solution to speed up the SA algorithm.

6.2.4 Determination of t_k

Given the optimized initial solution, an appropriate temperature t_k must be determined. If the selected t_k is too low, the t_k-SA can prematurely end up with a local minimum. Conversely, if the t_k is too high, the optimization that has been achieved by the initial mapping solution S_{heur} can be degraded because more uphill moves will be accepted at temperature t_k.

The method presented here tries to utilize the typical behavior depicted

in Figure 6.2 to determine the temperature t_k. Given the initial solution S_{heur}, the principle of the proposed method is to find a temperature t_k at which M_{WCA} is approximately equal to the WCA of the solution S_{heur}. The method is reasonable because the accepted solutions are controlled by the temperature in the acceptance function. Given an initial solution S_i, M_{WCA} of all accepted solutions will converge to the WCA of S_i if the amount of WCA increases and decreases of all solutions can offset each other.

In a SA algorithm, the accepted solutions at the temperature t comprise a *Markov chain* which is a sequence of probabilistic moves and each move to the next solution is only dependent on the current solution [123]. To calculate the M_{WCA} at temperature t, we have to generate the *Markov chain* first. The fact is, without the real run of the SA algorithm, it is impossible to get the exact Markov chain at a temperature. However, note that in the SA algorithm applied in this work, with the functions $Move(S,T)$ and $Accept(\Delta C,T)$ presented previously, the acceptance of a new solution in a Markov chain at temperature t is decided by a uniformly random probability. The Markov chain at temperature t is a result of a series of probabilistic movements. Based on the probability theory, we can use the *expected values* (or *mathematical expectations*) of a finite number of trial moves to approximate the M_{WCA} at a temperature t.

We assume that the SA algorithm starts from solution S_0, and $S_1, S_2, \ldots, S_{W-1}$ are the sequence of trial solutions generated by the function $Move(S,T)$. With this set of W trial solutions, the M_{WCA} at temperature t can be calculated as

$$M_{WCA} \approx \frac{1}{W}\left(\sum_{i=0}^{W-1} Cost(S_i)\cdot P_{accept(\Delta C_i,t)}\right) \tag{6.3}$$

where $Cost(S_i)$ and $P_{accept(\Delta C_i,t)}$ are the WCA of the solution S_i and the probability of accepting the solution $P(S_i)$ at temperature t respectively.

Since all downhill moves ($\Delta C \leq 0$) are certainly accepted, and an uphill move ($\Delta C > 0$) is accepted with the probability defined in the acceptance function $Accept(\Delta C,t)$ (Equation 6.1), $P_{accept(\Delta C,t)}$ can be defined as

$$P_{accept(\Delta C_i,t)} = \begin{cases} 1 & \Delta C_i \leq 0 \\ \frac{1}{\exp\left(\frac{\Delta C_i}{kC_0t}\right)} & \Delta C_i > 0 \end{cases} \tag{6.4}$$

Using the above $P_{accept(\Delta C_i,t)}$, Equation 6.3 is rewritten as:

$$M_{WCA} \approx \frac{1}{W}\left(\sum_{i=0}^{W_1} Cost(S_i) + \sum_{i=0}^{W_2} \frac{Cost(S_i)}{\exp\left(\frac{\Delta C_i}{kC_0t}\right)}\right) \tag{6.5}$$

where W_1 and W_2 are the number of downhill and uphill moves in the trial *Markov chain* at temperature t respectively.

With the approximation of M_{WCA}, we try to find the temperature t_k which satisfies the following equation:

$$M_{WCA}^{t_k} \approx Cost(S_{heur}) \tag{6.6}$$

Based on the Equation 6.6, the following method is proposed for calculating the temperature t_k:

1. Starting from solution S_{heur}, generate a finite number of W trial solutions in accordance with the function $Move(S, T)$;
2. Let $t_k = (T_0 + T_f)/2$;
3. Using the set of trial solutions, calculate $M_{WCA}^{t_k}$.
4. If $M_{WCA}^{t_k} < Cost(S_{heur})$, raise t_k according to a binary search and repeat 3 and 4; If $M_{WCA}^{t_k} > Cost(S_{heur})$, lower t_k according to a binary search and repeat 3 and 4; If $|M_{WCA}^{t_k} - Cost(S_{heur})| < \epsilon$, return t_k as the initial temperature for the t_k-SA algorithm.

Since all the downhill moves are accepted, in the given set of trial solutions, the sum of the costs of all downhill solutions ($\Delta C \leq 0$) remains constant regardless of the changes of t_k. Hence, $M_{WCA}^{t_k} < Cost(S_{heur})$ means that less uphill moves have been accepted due to a lower t_k. Therefore, the temperature needs to be raised in the next iteration. Conversely, the current t_k needs to be lowered if $M_{WCA}^{t_k} > Cost(S_{heur})$. ϵ is an user-defined positive real number. In this work, ϵ is set as $0.01 \times Cost(S_{heur})$.

6.2.5 t_k-SA Algorithm

Putting all above analysis and techniques together, the t_k-SA algorithm is presented in Algorithm 6, where the function $sa()$ refers to the Algorithm 1. S_{heur} and t_k replace S_0 and T_0 in the Algorithm 1 respectively. The parameters k, C_0 and q are kept the same as those applied in the full-range SA (Algorithm 5).

Algorithm 6: t_k-SA Algorithm

1 Generate the optimized starting solution S_{heur} using the tree-model based application mapping algorithm.
2 Determine the temperature t_k.
3 Find the best solution by applying S_{heur} and t_k to SA, $S_{best} = sa(S_{heur}, t_k)$.
4 Return S_{best}.

6.3 Experimental Evaluation

To evaluate the efficiency of the proposed t_k-SA algorithm, we experiment t_k-SA with a set of benchmarks and compare the results with the full-range SA algorithm (Algorithm 5).

6.3.1 Experiment Setup

Five benchmark applications are selected for the comparisons, including a video object plane decoder (VOPD) and a MPEG4 from SUNMAP [121], a multimedia systems application (MMS) [3], a H.264 decoder (H264) [122] and an image processing application BASIZ ([124]). The CCGs of these applications are derived from original descriptions. The size of the benchmarks and corresponding NoCs are summarized in Table 6.1.

For each benchmark application, the full-range SA algorithm is first performed for the tile assignment. Parameters T_0, T_f, k, q and C_0 are obtained using the method in Chapter 5. Afterward, the proposed t_k-SA algorithm is employed. The performances of these two algorithms, in terms of the runtime and the number of iterations needed for finding the final solution, are compared. To evaluate the quality of the final mapping solution, we use the NoCmap simulator presented in the NoCmap project [120] to get the communication energy consumption. Both algorithms were run on a Desktop PC with 3.0 GHz Intel Core2 Duo CPU and 8.0 GB of memory.

6.3.2 Result and Analysis

Applied Parameters

Table 6.1 shows the parameters applied in the full-range SA and t_k-SA algorithm respectively. $C_{S_{heur}}$ is the WCA of the mapping solution generated by the tree-model based application mapping algorithm. We can see from Table 6.1, with an optimized initial solution, the initial temperature t_k of the t_k-SA algorithm is hundred times lower than T_0 in the full-range SA algorithm. Starting from such a lower temperature t_k, t_k-SA skips an amount of temperatures in the full-range SA algorithm. The number of skipped temperatures is shown in the column "STs" in Table 6.1.

Iterations and Runtime

As anticipated from the skipped temperatures presented in Table 6.1, the iterations (consecutive moves in SA), as well as the runtime of the algorithm are reduced in the t_k-SA algorithm. As shown in Table 6.2 , for the five benchmarks, the number of iterations in the t_k-SA algorithm is on average 58% of that in the full-range SA algorithm. As a result, the runtime of the

Table 6.1: Parameters Applied in SA and t_k-SA

App	Cores	NoC	SA			t_k-SA		STs
			T_0	T_f	C_0	t_k	$C_{S_{heur}}$	
VOPD	16	4x4	14.15	5.48e-5	8.27e7	0.048	5.27e7	40
H264	16	4x4	7.53	8.00e-5	2.86e7	0.044	1.99e7	255
MPEG4	12	4x3	8.40	3.19e-5	1.47e8	0.041	8.00e7	56
MMS	25	5x5	21.28	1.97e-6	1.72e9	0.083	9.76e8	275
BASIZ	26	6x5	4.78	1.31e-6	1.82e7	0.021	1.09e7	133

t_k-SA algorithm is decreased and an average speedup of 1.55 is achieved over the full-range SA algorithm. For application H264, the largest speedup of 1.84 is achieved due to the lowest percentage of iterations (48%). Note that the runtime of the t_k-SA algorithm includes the time of generating the initial solution S_{heur} and that of determining the temperature t_k.

Table 6.2: Iterations and Runtimes for Benchmarks

Benchmark	Iterations			Runtimes		
	SA	t_k-SA	pct	SA	t_k-SA	speedup
VOPD	2.26e4	1.23e4	54%	0.079	0.050	1.58
H264	1.96e5	9.34e4	48%	0.582	0.317	1.84
MPEG4	1.76e4	1.00e4	57%	0.038	0.024	1.58
MMS	5.43e5	3.57e5	66%	4.542	3.720	1.22
BASIZ	2.55e5	1.64e5	64%	2.253	1.462	1.54
Avg.	-	-	58%	-	-	1.55

Accuracy of Temperature t_k

The temperature t_k is the most important parameter which affects the performance of the t_k-SA algorithm and the quality of the final mapping solution. The closer the acquired t_k approaches to that in the full-range SA algorithm, the more accurately the t_k-SA can reproduce the behavior of the full-range SA algorithm in the period after temperature t_k. As shown in Table 6.3, two values are used for the measurement of the accuracy of the temperature t_k. The first one, $M_{WCA}^{t_k-SA}$, stands for the mean of WCAs at temperature t_k in the t_k-SA algorithm. The second one, M_{WCA}^{SA}, is the mean of WCA at temperature t_k (or nearly t_k) in the full-range SA algorithm. The variance between $M_{WCA}^{t_k-SA}$ and $C_{S_{heur}}$ can measure the accuracy of t_k determined using Equation 6.6. Furthermore, the variance between $M_{WCA}^{t_k-SA}$

and M_{WCA}^{SA} can verify how efficiently the t_k-SA algorithm reproduces the behavior of the full-range SA algorithm in the period after temperature t_k.

As we can observe from Table 6.3, the $M_{WCA}^{t_k-SA}$ is very close to the $C_{S_{heur}}$ in all applications except for MMS. This demonstrates that the method proposed in this work is efficient for determining the temperature t_k. For all applications, $M_{WCA}^{t_k-SA}$ and M_{WCA}^{SA} are comparable. The largest variance is less than 7% in the application VOPD. This means that the t_k-SA and full-range SA algorithms behave similarly at the temperature t_k. Since both of them use the same set of parameters and functions, they are more likely to behave similarly after temperature t_k.

Table 6.3: Mean of WCA at t_k for Benchmarks

App	t_k	$C_{S_{heur}}$	$M_{WCA}^{t_k-SA}$	M_{WCA}^{SA}
VOPD	0.048	5.27e7	5.37e7	5.02e7
H264	0.044	1.99e7	1.93e7	1.87e7
MPEG4	0.041	8.00e7	7.90e7	8.47e7
MMS	0.083	9.76e8	1.31e9	1.29e9
BASIZ	0.021	1.09e7	1.05e7	1.06e7

Figures 6.3 and 6.4 show the behavior of the t_k-SA and full-range SA algorithms for mapping application $H264$ respectively. M_{WCA} stands for $M_{WCA}^{t_k-SA}$ and M_{WCA}^{SA} respectively. With same scale in both figures, we can see that the behavior of the t_k-SA algorithm approximate that in the full-range SA algorithm (after temperature t_k) very well. This convinces the efficiency of using the t_k-SA algorithm to speed up the general SA algorithm without the loss of the mapping quality.

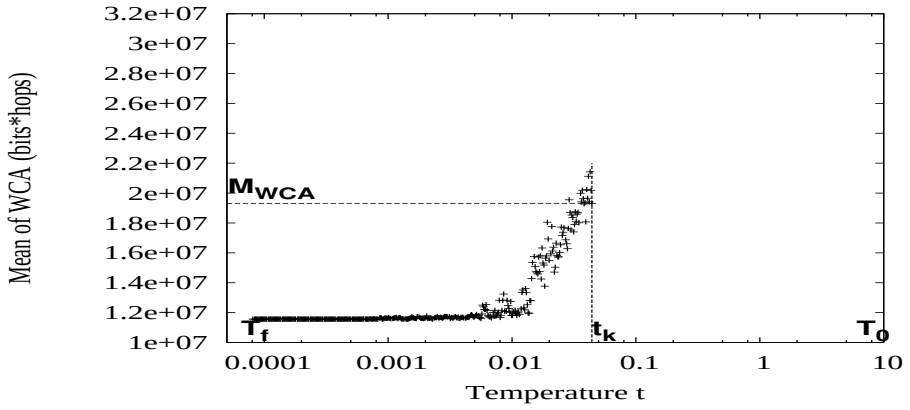


Figure 6.3: Behavior of t_k-SA Algorithm (H264)

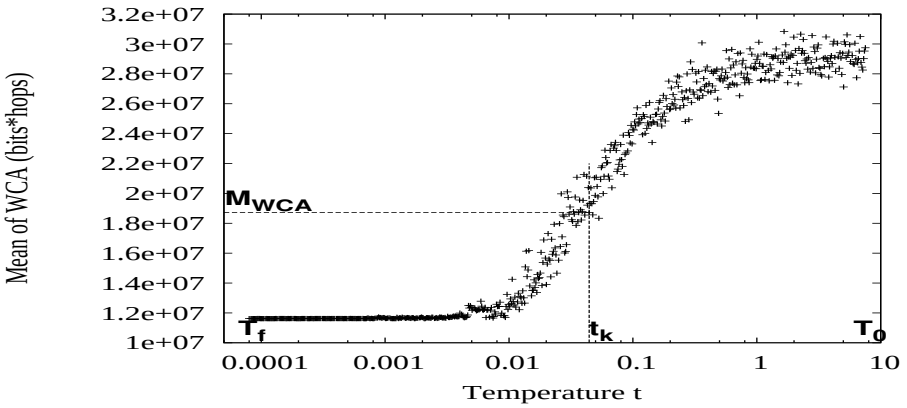


Figure 6.4: Behavior of full-range SA Algorithm (H264)

Quality of Mapping Solution

The qualities of mapping solutions obtained by the t_k-SA and full-range SA algorithms are first evaluated mathematically by the WCAs of the mapping solutions. Figure 6.5 shows the WCAs of mapping solutions for each application found by the t_k-SA and full-range SA algorithm respectively. For applications H264 and MMS, the WCAs of both algorithms are almost the same. For other applications, the WCA using the t_k-SA algorithm is slightly smaller than that using the full-range SA algorithm. The largest reduction of WCA is about 5% in the case of application BASIZ.

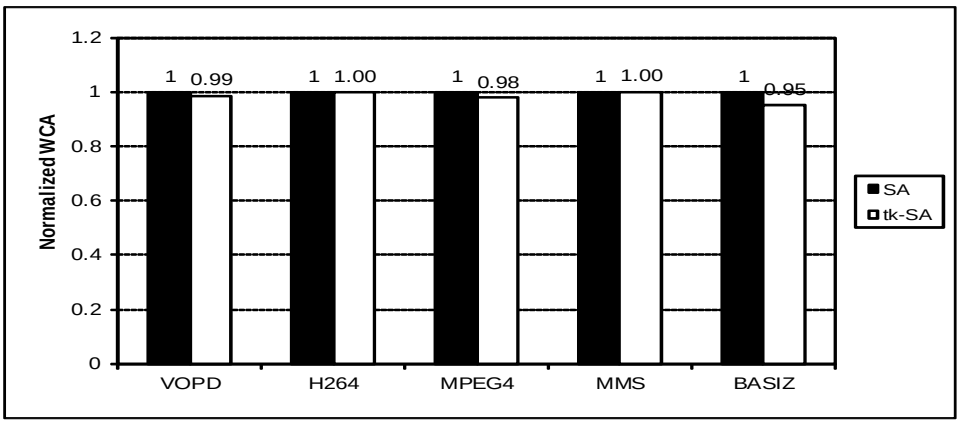


Figure 6.5: Comparison of WCA

The quality of mapping solutions of the t_k-SA and full-range SA algorithms is further evaluated by the communication energy consumption obtained from the NoCmap simulator. As anticipated by the WCA, the communication energy consumptions are comparably the same between so-

lutions of the two algorithms. Figure 6.6 shows that the maximal difference exists again in the case of application BASIZ, which is less than 5%.

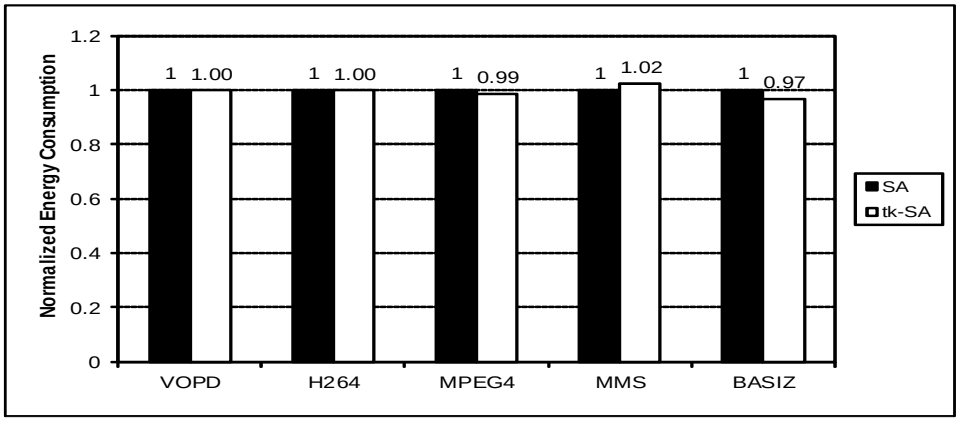


Figure 6.6: Evaluation of Energy Consumption

The comparisons of the WCA and the communication energy consumption verify the efficiency of the proposed t_k-SA algorithm. Although smaller number of iterations and shorter time are used, the t_k-SA algorithm achieves the same good quality of the mapping solutions as that of the full-range SA algorithm.

6.4 Chapter Summary

An accelerated SA algorithm, t_k-SA algorithm, was proposed in this chapter. Instead of using a random mapping solution, the t_k-SA algorithm employs an optimized mapping solution as the initial solution. Based on the analysis of the typical behavior of the general SA algorithm, an efficient method for calculating the initial temperature t_k which corresponds to the optimized initial solution was proposed. The accuracy of the temperature t_k determined by the proposed method was quantitatively verified.

Starting from the lower initial temperature t_k, the t_k-SA algorithm saves the iterations and runtime required for converging to the optimal mapping solution. The experimental results showed that, for the given set of benchmark applications, on average, the t_k-SA algorithm uses 58% iterations of that in the full-range SA algorithm. As a result, an average speedup of 1.55 is achieved by the t_k-SA algorithm. In addition, using the temperature t_k, the t_k-SA algorithm reproduces the behavior of the full-range SA algorithm. This enables the accelerated t_k-SA algorithm to obtain the mapping solution which has the same quality as that found by the full-range SA algorithm.

Chapter 7

Resource-Aware Multi-Application Mapping

The massive parallel computing performed on many-core platforms is the future of computing [125]. With increasing processing capability, communication channels and flexible scalability, many-core NoC provide great potential for solving today's increasingly complex problems. With abundant processing cores available on many-core NoCs, it is no longer reasonable to only focus on the implementation of one single application. Instead, the design focus should shift from the single-application to the multi-application scenarios, more precisely, to deploy multiple applications simultaneously on a many-core NoC. For instance, [66] presents a concrete example of a 167-core computing platform, where 9 cores are dedicated to realize a JPEG encoder, and 15 cores are utilized to implement a H.264 encoder. Such physical separation of multiple applications onto different regions of the NoC may reduce the design complexity (e.g., imposed by context switching). And the communication interference between applications will be significantly alleviated. A system with less communication interference will benefit from lower latency, higher throughput and shorter execution time [126].

While the majority of previous works focus on single application mapping, few works have been undertaken for mapping multiple applications on a NoC platform. [127] presented a methodology to map multiple use-cases onto the NoC architecture, with each use-case executing on specific time slots. This method incurs considerable design and verification complexity dealing with context switching, as well as the timing overhead. In [128], multiple applications are mapped on different regions of the NoC so that each application has no or smallest interference with other applications. Search heuristics including simulated annealing, greedy incremental and Tabu search were applied for mapping multiple applications. Due to the larger search space imposed by the multi-application mapping prob-

lems, the method is time-consuming. Multi-application mapping is still an open problem and more novel methods need to be developed.

This chapter presents a novel methodology for mapping multiple applications, which is aware of the available IP cores on the many-core platforms. The multi-application mapping extends the single-application mapping which we have addressed in previous chapters and consists of two steps: application mapping and task mapping. The two-step mapping method first finds an area on the NoC for each application and then maps all tasks of the application into the region. With general applicability to the unbounded mapping (UM) and bounded mapping (BM) problems, the method is adaptive to the availability of on-chip cores. Several strategies are proposed for application mapping and task mapping. The proposed two-step mapping methodology aims to minimizing the communication energy consumption of multiple applications and the timing overhead due to the core sharing by multiple tasks in the BM case.

7.1 Multi-Application Mapping Problem

In contrast to the single-application mapping problem, a multi-application mapping problem consists of multiple independent applications, each of them can be represented by a TG (task graph) or a CCG (core communication graph) presented in Section 3.2.2. The objective functions and constraints of the set of applications may vary. The many-core NoC is represented by the NoC model presented in Section 3.2.1.

Since in the BM case, the sequencing of tasks on individual cores will significantly impact the execution time due to data dependencies, the time consumed for each application needs to be formulated. While the computation time is given in the TG of one application, it is difficult to obtain the communication time at the design time because it is dependent on the particular mapping solution, as well as the switching and routing techniques applied in the many-core NoCs. Hence, in this work, we approximate the communication time by the data volume given in the TG of an application. The approximation method is explained in Section 7.3.2.

Given a TG in which the communication time is approximated by the data volume, the Earliest Completion Time (ECT) of a task t_i, denoted by $ECT(t_i)$, is defined as follows:

$$ECT(t_i) = \max_{1 \leq k \leq p} (ECT(t_k) + CT_{ki} + ET_i) \tag{7.1}$$

where task t_i has p parent tasks and t_k is its kth parent task. CT_{ki} and ET_i are the communication time of the communication Com_{ki} and the execution time of task t_i respectively. $ECT(t_e) = ET_e$, if t_e is an entry task.

Similarly, the Latest Completion Time (LCT) of a task t_i, denoted by $LCT(t_i)$, is defined as follows:

$$LCT(t_i) = \min_{1 \leq u \leq w} (LCT(t_u) - CT_{iu} - ET_u) \tag{7.2}$$

where task t_i has w child tasks and t_u is its uth child task. $LCT(t_x) = ECT(t_x)$, if t_x is an exit task.

Those tasks with same ECT and LCT compose the *critical path (CP)* of an application. A CP is the longest path from an entry task to an exit task. The sum of the computation and the communication times of tasks on the CP, i.e., the length of the CP, determines the shortest execution time of an application.

7.1.1 Problem Formulation

In the single application mapping case, given a many-core NoC with a particular topology and an application implemented by a set of tasks, the role of mapping is to decide how to topologically place each task onto a core on the NoC so that the design objectives are achieved subject to specific constraints. For the multi-application mapping problem, we consider how to simultaneously map multiple applications on a NoC. In contrast to single-application mapping, the multi-application mapping contains not only the placement of tasks but also that of applications on the NoC.

In this work, the objective of the multi-application mapping problem is to minimize the energy consumptions both in UM and BM cases. In addition, since the mapping and scheduling of tasks in BM case will change the execution time of an application, BM method also aims to minimize the length of the critical path for each application.

Using the aforementioned application and NoC models, we define the multi-application mapping problem as follows:

Given a set of TGs and a NoC;

Find a mapping area for each application on the NoC, and a tile within the mapping area for each task of the application with either unbounded or bounded number of cores;

Such that for the given set of TGs, the communication energy consumption on the NoC is minimized. And, in the bounded mapping, the execution time of each application is minimized.

7.1.2 Work Flow of Multi-Application Mapping

Figure 7.1 illustrates the work flow of the proposed methodology for solving the above multi-application mapping problem. The given set of applications are mapped based on the order they arrive (first-come, first-served), or their priorities (if more than one arrive at the same time). The system first decides

whether the unbounded or bounded mapping should be applied. If there are enough cores for all applications, the unbound mapping is applied, otherwise bounded mapping is used. Then, a two-step mapping is performed for each application, including application mapping and task mapping phases. In both unbounded and bounded cases, the WNAD-Based algorithm (Section 7.2.2) is proposed for application mapping. For task mapping, the t_k-SA mapping algorithm presented in Chapter 6 is applied to the unbounded mapping, and a task-sequence-based (TSB) algorithm (Section 7.3.3) to the bounded mapping.

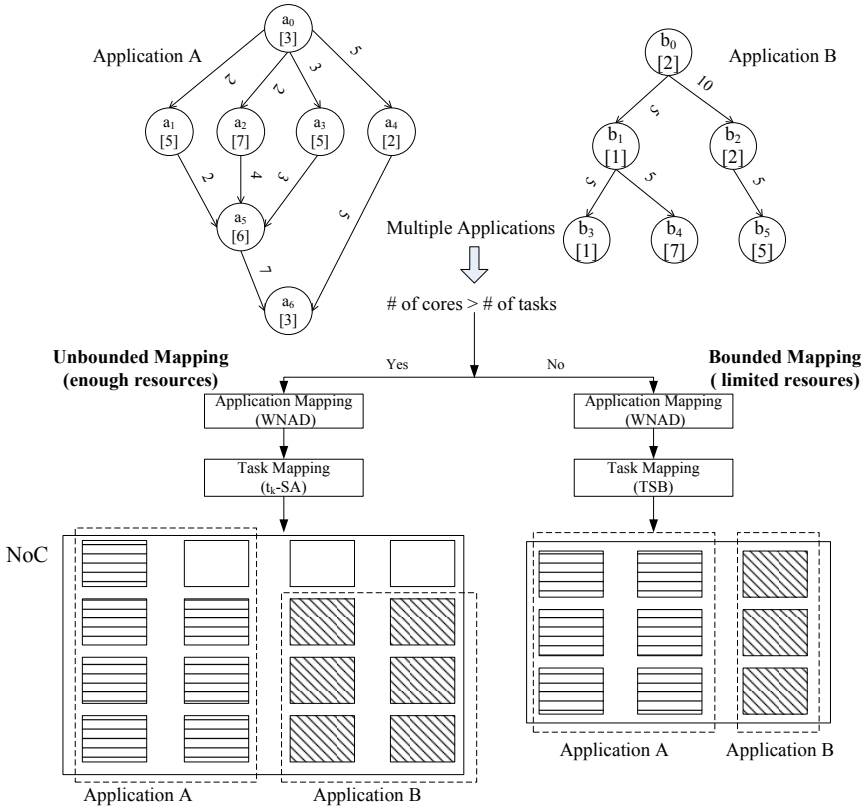


Figure 7.1: Work Flow of Multi-Application Mapping

7.2 Unbounded Mapping

With enough on-chip cores, each task can be exclusively mapped on a single core. The objective of the UM is to minimize the communication energy consumption by shortening the communication distances among IP cores, especially for those with large data transmissions.

7.2.1 Objective Formulation

To evaluate the communication energy consumption, we use the volume-based energy consumption model presented in Section 3.4. According to this model, the energy consumption of one communication com_{ij} is linearly proportional to the products of the data volume $vol(com_{ij})$ and the length of the routing path γ_{ij}. In order to minimize the communication energy consumption, we need to find the mapping solution in which the communicating cores are mapped as close as possible. One way to achieve this is to map an application onto a continuous and compact area rather than multiple separated areas on the NoC. For this purpose, we use the nodes average distance (NAD) mentioned in [27] to evaluate the quality of the candidate mapping areas. NAD is defined as the average distance between two randomly selected nodes on a NoC. For a $X \times Y$ mesh NoC, the NAD is calculated by Equation (7.3)

$$NAD = \frac{X+Y}{3} \times \left(1 - \frac{1}{X \times Y}\right) \tag{7.3}$$

The Equation (7.3) implies that for a given application, the average communication distance among tasks varies when different areas are chosen for mapping the application. For example, for an area with size 6×4, the NAD is approximately 3.2, but that for an area with size 3×3 is approximately 1.8. This means, on average, a transmission in the former case needs to go a longer way and consumes more energy than that in the latter one. Therefore, an area with smaller NAD is preferable for mapping an application.

Given a selected mapping area, the next step to minimize the communication energy consumption is to find an optimized mapping solution. As presented in previous chapters, for a single application, a lower energy consumption can be achieved by minimizing the WCA (weighted communication of an application). The WCA is defined as the sum of the product of the communication $vol(com_{ij})$ and the length of the routing path γ_{ij} for all communications of an application. We extend the concept of WCA to the weighted communication of multiple applications (WCMA). For a set of M applications, the WCMA is the sum of WCAs of all M applications as follows:

$$WCMA = \sum_{k=1}^{M} \sum_{\forall com_{ij}} (vol(com_{ij}) \times \gamma_{ij}) \tag{7.4}$$

Based on these formulations, the objective of the multi-application mapping problem in UM case is transformed into: (1) to search for a mapping area with smallest NAD for each application and, (2) to find the optimized

task mapping solution yielding the minimized WCA for each application and consequently the minimized WCMA for multiple applications.

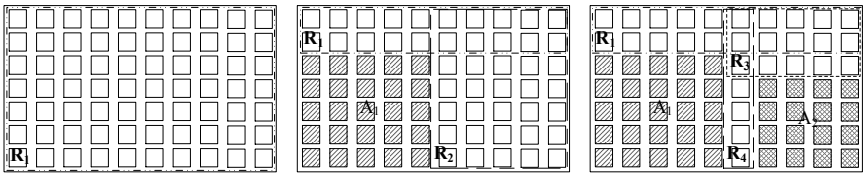
7.2.2 Two-Step Multi-Application Mapping

To reach the above objectives, we propose a multi-application mapping method which consists of two sequential steps: *Application Mapping* and *Task Mapping*. The role of application mapping is to find an optimal mapping area with minimal NAD on the NoC for each application. The task mapping works after application mapping and targets at mapping all tasks of an application onto the mapping area with the purpose of minimizing the WCA.

NAD-Based Application Mapping

Implied by Equation 7.3, for a given application, if we can find a square area to map it, a minimal NAD can be achieved. On a 2-D mesh NoC, when a square or a rectangle area is allocated to an application, the remaining area on the NoC is divided into several rectangles. Therefore, the main job of application mapping is how to efficiently allocate the unused area on the NoC to each application. Based on this notion, we adopt the concept of maximal empty rectangle (MER), which was originally used to solve the placement problem in FPGA design [129].

Maximal Empty Rectangle An MER is an empty rectangle that is not contained by any other empty rectangles. In our case, an MER represents a cluster of spare nodes on the NoC that could be used for mapping an application. Figure 7.2 shows an example of application mapping using the MER approach. At first, the whole NoC is represented by one MER R_0 (Figure 7.2a). After the mapping of application A_1, the R_0 is split into R_1 and R_2 (Figure 7.2b). In Figure 7.2c, the R_2 is further divided into R_3 and R_4 after the application A_2 has been mapped. The MERs R_1, R_3 and R_4 can be used for mapping the following applications.



(a) (b) (c)

Figure 7.2: Application Mapping Using MER

Objective MER Selection For a given application with m tasks, application mapping tries to find an optimal or nearsub-optimal objective MER to map the application. Based on the state of MERs on the NoC, the cases that application mapping may encounter include: (1) there is at least one candidate MER that can accommodate the given application; (2) the total amount of nodes in all MERs is sufficient to accommodate the given application, but none of them can accommodate the application alone.

For the first case, there is one sub-area in each candidate MER, on which if the given application is mapped, the minimal NAD can be achieved. We denote the minimal NAD in the ith candidate MER by NAD_i, and the sub-area by $Area_i$. Assuming there are k candidate MERs, application mapping will select the jth MER which has the smallest NAD_j among all NAD_i as the objective MER, and the area $Area_j$ as the mapping area $Area_{map}$ for the given application. The selection function is described as follows:

$$Area_{map} = Area_j, \exists j, NAD_j = \min_{i=1}^{k} NAD_i \tag{7.5}$$

To deal with the second case, the *Largest Size + Combining (LS+C)* strategy is applied. In this case, the application has to be mapped on several noncontiguous MERs. To avoid increasing communication cost between more noncontiguous MERs with small size , LS+C chooses the free MER with largest number of spare nodes as the primary area and then combines the nearest free MERs to get adequate nodes for mapping the application.

MER Merging When the execution of an application finishes, the area occupied by the application can be released and merged with neighboring free MERs to form larger MERs for the following mapping.

Combining these strategies together, the NAD-Based application mapping algorithm is described as Algorithm 7.

WNAD-Based Application Mapping Algorithm

In Algorithm 7, the MERs which cannot accommodate the given application would not be selected as an objective MER as long as there are candidate ones, even if some of them have smaller NADs than the selected objective MER and can accommodate most number of tasks of the application. Figure 7.3a is an example of application mapping using NAD-Based Algorithm. After the application A_1 and A_2 have been mapped, the candidate MER R_1 is selected, although the non-candidate MER R_2 with the smaller NAD and close size (15) for the application A_3. This is because the combination of several noncontiguous MERs is likely to induce higher NAD and WCA than a continuous MER. However, if tasks which most affect the WCA are mapped in an area with smaller NAD, the rest of the tasks can have

Algorithm 7: NAD-Based Application Mapping Algorithm

Input : A: a set of applications (DAGs), CCRG: a 2-D mesh NoC with size $W \times H$

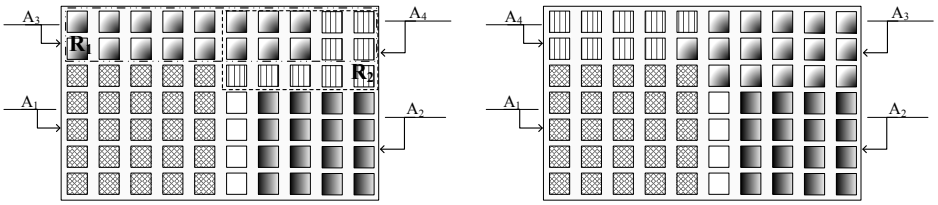
Output: The mapping area $Area_{map}$ for each application

```
1  Initiate the original MERs list R_0 with size W × H.
2  for each application A_i do
3      if more than one MER can accommodate A_i then
4          Set NAD_min = NAD_R0, Area_map = R_0;
5          for each MER R do
6              Compute the minimal NAD NAD_R and the corresponding
               mapping area Area_R in R for application A_i;
7              if NAD_R ≤ NAD_min then
8                  NAD_min = NAD_R;
9                  Area_map = Area_R;
10     else
11         Use the LS+C strategy to find a mapping area Area_map;
12     Fragment the objective MER and update the MERs list;
13 if application A_j is completed then
14     Merge the area occupied by A_j with neighboring free MERs;
15     Update the MERs list;
```

limited impact on the overall WCA even if they are mapped on separate MERs. Based on this consideration, we extend the NAD-based application mapping algorithm to the weighted NAD (WNAD) based algorithm. The WNAD of an MER is defined as follows:

$$WNAD = \frac{N_{tasks}}{N_{nodes}} \times NAD \tag{7.6}$$

where the first factor is the *weighted ratio* which indicates the percentage of tasks of an application that can be mapped on an MER. N_{tasks} is the number of tasks of an application and N_{nodes}, the number of nodes available in the MER. For a candidate MER presented above, the weighted ratio equals to 1 and the WNAD strategy is equivalent to the NAD strategy. An MER with a lower WNAD can accommodate most tasks of the application and has a smaller NAD. Using the WNAD strategy, both the candidate and non-candidate MERs presented in the Section 7.2.2 can be evaluated together for finding the objective MER. If a non-candidate MER is selected as an objective MER, the LS+C strategy is used to combine the non-candidate MER with its nearest neighboring MERs to form the mapping area $Area_{map}$.



(a) NAD-Base Mapping (b) WNAD-Base Mapping

Figure 7.3: Comparison of Mappings for Application A_3

Using WNAD strategy, the Algorithm 7 is extended to Algorithm 8. Figure 7.3b is an example of using WNAD algorithm to map the same set of applications mentioned in Figure 7.3a. The main difference between the NAD and the WNAD algorithms is that, the application A_3 is mapped on the area combined by MER R_2 and one node in MER R_1, instead of the area completely inside the MER R_1. As a result, the NAD of the mapping area is decreased from 3.13 to 2.83.

Algorithm 8: WNAD-Based Application Mapping Algorithm

Input : A: a set of applications, NoC: a 2-D mesh with size $W \times H$
Output: The mapping area $Area_{map}$ for each application

```
1  Initiate the original MERs list R_0 with size W × H.
2  for each incoming application A_i do
3      Set WNAD_min = WNAD_{R_0}, Area_map = R_0;
4      for each MER R do
5          Compute the minimal WNAD WNAD_R and the
           corresponding mapping area Area_R in R for application A_i;
6          if WNAD_R ≤ WNAD_min then
7              WNAD_min = WNAD_R;
8              Area_map = Area_R;
9      if the objective MER is a non-candidate MER then
10         Use the LS+C strategy to form a mapping area Area_map.;
11     Fragment the objective MER and update the MERs list;
12 if application A_j is completed then
13     Merge the area occupied by A_j with neighboring free MERs;
14     Update the MERs list;
15 Repeat 2-14 until mapping area Area_map for each application is
   found.
```

t_k-SA Task Mapping

When the mapping area for a given application has been obtained in the application mapping stage, the role of task mapping is to map the tasks of the application into the mapping area with the purpose of minimizing the WCA. The task mapping is a single-application mapping problem. Therefore, the t_k-SA algorithm (Algorithm 6) presented in Chapter 6 is applied.

7.3 Bounded Mapping

7.3.1 Application Mapping

In the BM case, a set of m applications are to be mapped on a bounded number of IP cores. The m applications consist of p tasks in total and each of them has p_0, p_1, ..., p_m number of tasks respectively. The number of cores provided by the NoC is q and $q < p$. We define the *scale-down ratio* by q/p which means that the number of cores allocated to an application has to be scaled down by q/p.

Given the number of cores allocated to each application, either NAD or WNAD-Based algorithm presented previously can be applied to the application mapping step.

7.3.2 Task Mapping Analysis

The task mapping algorithm in the BM case utilizes the tree-model based algorithm presented in Chapter 4. The first phase of the task mapping is the same as that in the original tree-model based algorithm, i.e., the abstraction of a mapping area into an extended tree. The major difference lies in the second phase, i.e., mapping tasks onto the extended tree.

Since a task in the bounded case has to share a core with other tasks, the ECT (Equation 7.1) of a task is not only determined by the completion time of its parent tasks, but also by the completion time of the task preceding it in the task sequence in the same core. On other words, mapping a task on a core may modify the length of the critical path and in turn change the execution time of an application. For example, in Figure 7.4a, if we assume that tasks t_0, t_2 and t_3 are sequentially assigned to the same core (surrounded by the dotted circle), the execution of task t_3 cannot start until the task t_2 finishes, even though its predecessor task t_0 has already completed. In this case, the ECT of the task t_3 is changed from 11 to 15 time units.

Therefore, in the bounded mapping, to minimize the execution time of an application, the task sequence in which all tasks are executed consecutively on a core should be optimized whenever a new task is mapped on the core. To evaluate the impact imposed by various task sequences on the execution

time of an application, the dependency imposed to a task by sharing the same core with tasks that have no data dependency with the task, needs to be denoted in the TG (task graph). We refer to this dependency as the *virtual dependency (VD)* in comparison with the data dependency between two tasks in the original TG.

Modified Task Graph

A VD is added into a TG if one task is following another task in the task sequence of a core while in reality there is no dependency between them in the original TG. As shown in Figure 7.4b, the VD between tasks t_2 and t_3 is depicted by a directed dotted line from t_2 to t_3. The weight of a VD is zero because in fact there is no communication between them.

The presences of VDs modify a TG by adding new dependency constraints into the original TG and consequently modify the ECT and LCT (Equation 7.2) of a task and other tasks that depend on it. These modifications will finally change the CP of the TG. As shown in Figure 7.4, after t_3 is mapped to the core where tasks t_0 and t_2 have already been mapped, the CP of the TG is changed from $t_0 -> t_2 -> t_5 -> t_6$ (Figure 7.4a) to $t_0 -> t_2 -> t_3 -> t_5 -> t_6$ (Figure 7.4b). The length of the CP is increased by 3 time units.

Clearly, mapping a task to different cores will result in different modified TGs and corresponding CPs. The variants of task sequence in one core will also generate different TGs and CPs. In Figure 7.4, if the execution order of tasks t_2 and t_3 reverses, a VD of task t_2 on task t_3 is added and a CP with longer length ($t_0 -> t_3 -> t_2 -> t_5 -> t_6$) is resulted in. To achieve a minimized CP of an application, in each step of mapping a task, we need to find the right core and the right position in the task sequence of the core for the given task. By mapping the given task on this core and putting it in the proper position in the task sequence, the minimal CP can be achieved. This is the main idea of the task mapping in the bounded mapping.

Communication Time

To calculate and compare different CPs derived from different modified TGs, both the execution times of tasks and communication times of inter-task communications in a TG should be specified. In fact, both of them are dependent on the computation and communication resources on which an application is to be mapped. Especially for the communication time between two tasks, it would not be known until both tasks are mapped on the cores of a NoC. Some previous works surveyed in [22] addressing the scheduling problem based on the assumption that, either there is no communication cost, or multiprocessors are fully connected. Hence, the communication cost

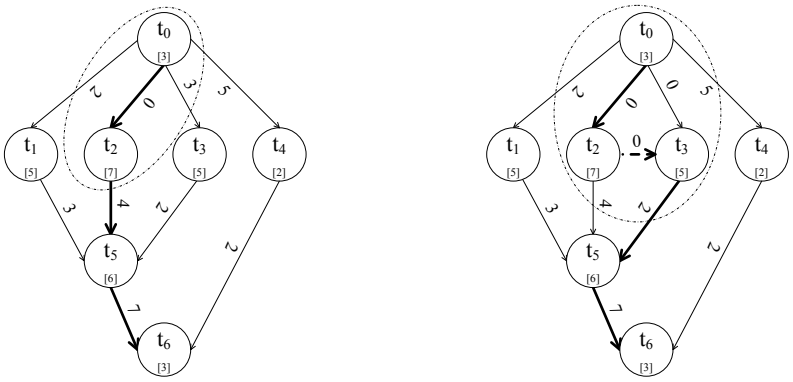


Figure 7.4: Modified Task Graph with Virtual Dependency

could be specified in a TG beforehand. Clearly, this is unrealistic for most cases, such as on a many-core NoC based system presented in this work. In practice, it is possible to get data volume by application analysis. Based on a 2D mesh NoC where the wormhole packet switching is applied, we can approximate the communication time with data volume between any two IP cores. This is because, on such a NoC, the network latency of wormhole routing is $(L_f/B)D + L/B$, where L_f is the length of a flit, B is the channel bandwidth, D is the length of the routing path, L is the length of the data. If $L_f << L$, the communication time is dominated by the part of L/B [130]. Therefore, given the channel bandwidth B, it is reasonable to use L/B to approximate communication time when we compute the length of the CP of a TG. In addition to the approximation, the communication time between two tasks assigned to the same core is not considered since the data is not communicated on the NoC.

7.3.3 Task-Sequence-Based Task Mapping

Using the techniques of virtual dependency and modified task graph, we develop the Task-Sequence-Based (TSB) mapping algorithm for task mapping in BM. The algorithm first selects a task and then compare all CPs resulted from modified TGs when the task is mapped to each possible position in the task sequence of each candidate core. The core and the position in the task sequence which achieve the minimal length of CP will be chosen for mapping the selected task.

Task Selection

As in the tree-model based mapping algorithm, at first, the task with the largest communication volume with other tasks will be selected and mapped on the root node of the extended tree. Then the task with largest APT (see Section 4.2.2) will be selected as the next task to be mapped on the extended tree.

Core Selection

Unlike UM, in BM both free cores and the used cores on which some tasks have already been mapped, need to be evaluated for finding the optimal task mapping. In case that there are spare cores on the extended tree, the way by which a core is selected in the tree-model based mapping algorithm is used to find a spare core for the selected task. The mapping on this core has to be compared against the mappings on the used cores. In case that there is no spare core available on the NoC, the mapping solutions on all used cores are compared with each other to find the optimal task mapping. The one achieving the shortest length of CP will be chosen as the objective core for the given task. If the mappings on several cores get the same CP, the WCA is used to determine the best one.

Comparison of Task Sequence

For a core and a selected task, the different position in the task sequence where a task is inserted will result in different modified TGs and CPs. To compare with the mapping solutions on other cores, the comparison among all possible positions in the task sequence of a core has to be done first. When the selected task is inserted in one position in the task sequence, the TG is modified correspondingly. The length of the CP of the modified TG is calculated and compared with the shortest one obtained so far. Finally, the algorithm gets the shortest CP and the position in the task sequence which achieves the shortest CP. A task cannot be put in a position before any preceding predecessor task in the original TG.

Using the methods presented above, the TSB mapping algorithm is described in Algorithm 9.

7.4 Quantitative Evaluation

To evaluate the effectiveness of the proposed multi-application mapping methods, we experimented the algorithms with a set of benchmarks on an in-house many-core NoC simulator [131].

Algorithm 9: Task-Sequence-Based Task Mapping Algorithm

Input : TG, extended tree ET of the mapping area
Output: Mapping solution M

```
1  Calculate CV for all tasks t_i ∈ T;
2  Select the task t_b with largest CV, map this node onto the root node
   n_r of ET, create T' = {t_b} and M = {< n_r, t_b >}, remove t_b from T;
3  while T is not empty do
4      Calculate APT_{t_j} for all t_j ∈ T;
5      Select the task t_b with the largest APT;
6      Set minimal CP CP_min the CP of original DAG;
7      if there are free nodes on ET then
8          select one free node n_i using tree-model based algorithm and
           map t_b on it;
9          calculate the CP_i of the DAG;
10         if CP_i < CP_min then
11             CP_min < CP_i
12     for each used node n_j on ET do
13         Set minimal CP CP_j on node n_j the CP of original DAG;
14         for each possible position s in the scheduling sequence of n_j do
15             insert the task t_b on the position s of the scheduling
               sequence ;
16             add a virtual dependency in the TG and modify the TG;
17             update the ECT and LCT of each task;
18             calculate the CP_{j_s} of the modified DAG;
19             if CP_{j_s} < CP_j then
20                 CP_j < CP_{j_s}
21         if CP_j < CP_min then
22             CP_min < CP_j
23     Map t_b on the node n_k which gets the minimal CP CP_j;
24     Put t_b on the optimal position in the scheduling sequence in n_k;
25     Append pair < n_map, t_b > into M;
26     Append t_b into T', remove T_b from T;
```

7.4.1 Experimental Setup

A system-level many-core NoC simulator is built to analyze the execution time and communication energy of the experimental benchmarks, under different settings. The in-house simulator is cycle-accurate in terms of network transmission, where each stage of the data switching and routing process in

the routers is modeled explicitly.

The NoC platform is either a 9×9 mesh for unbounded mapping, or a 7×7 mesh for bounded mapping. The simulator adopts X-Y deterministic routing and wormhole switching. Each router has 5 IN/OUT ports with 2 flit-depth buffers in each input port. To estimate the energy consumption, each link between routers is assumed as 1 mm long and 32 bits wide. Orion 2.0 [132] is used to model the energy consumed on routers and links.

Four benchmarks are simulated, including an image processing application (BASIZ, [124]), a MPEG encoding application [133], and kernels for Gaussian Elimination (GE) and Laplace Equation (LE) algorithms [134]. BASIZ is a parallel application with 26 tasks detecting image zones with more brightness and color intensity. MPEG has 21 tasks. The 18-task GE and 16-task LE algorithms are widely used for digital signal processing.

Mapping Step	Unbounded Mapping	Bounded Mapping
Application Mapping	WNAD NAD	WNAD
Task Mapping	t_k-SA GI	TSB NoP

Table 7.1: Mapping Algorithms for Experimental Comparison

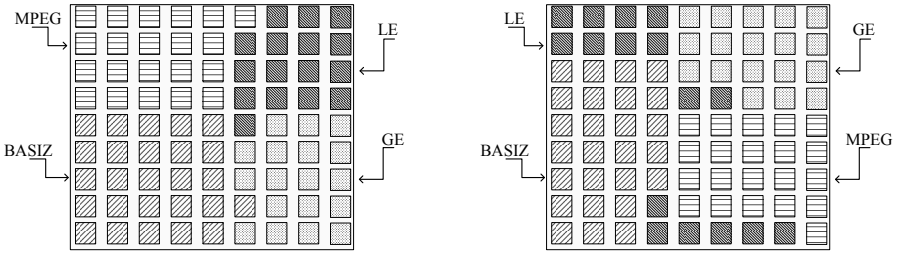
The proposed multi-application mapping methods are compared with existing techniques (Table 7.1). In UM, for application mapping we compare the WNAD-Based (Algorithm 8) algorithm for application mapping with NAD-Based algorithm (Algorithm 7). We will demonstrate the benefits of WNAD-Based algorithm by the mapping analysis of the four benchmarks in Section 7.4.2. For task mapping, we compare t_k-SA algorithm (Chapter 6) with Greedy Incremental (GI) algorithm [135]. The GI algorithm is evaluated with superior mapping results compared to previous works. The combination of application and task mapping algorithms leads to four mapping methods: WNAD+t_k-SA, WNAD+GI, NAD+t_k-SA and NAD+GI.

In bounded mapping, we compare our TSB algorithm with non-optimized (NoP) task mapping, since there is no existing work addressing the same problem. The NoP mapping algorithm maps the tasks based on their sequential order defined in the task graph and randomly selects the core on which a task is mapped. It considers the dependency of tasks defined in the task graph but without further optimization of scheduling.

7.4.2 Results of Unbounded Mapping

The four benchmarks are mapped on a 9×9 NoC, with the results illustrated in Figure 7.5. From Figure 7.5a and 7.5b, we can observe the effectiveness

of WNAD-Based application mapping algorithm. Using the WNAD-Based algorithm, the application BASIZ is mapped onto an area of a 5×5 square plus a neighboring node. MPEG and GE are also mapped on contiguous areas. When mapping the application LE, there is still a contiguous area left for it. In contrast, using the NAD-Based algorithm, the application LE is mapped onto three separate areas. Since NAD-Based algorithm always tries to allocate a candidate MER which can completely accommodate an application in the application mapping, it usually assigns more cores to an application than required. As a result, the first three applications, BASIZ, MPEG and GE, are mapped to the areas which are similar to those in the WNAD-Based algorithm. When it comes to application LE, the mapping area has to be formed by the fragment areas left on the NoC, which results in a larger NAD than that using the WNAD-Based algorithm.



(a) Application Mapping with WNAD-Based Algorithm

(b) Application Mapping with NAD-Based Algorithm

Figure 7.5: Mapping Result of Unbounded Mapping on a 9×9 NoC

The normalized communication energy consumption for different mapping algorithms is illustrated in Figure 7.6. We can observe that the application mapping has great impact on the energy consumption. As anticipated from the increased NAD in Figure 7.5b, the largest difference of energy consumption is shown in the application LE. The energy consumptions of NAD-based algorithm are 136% and 98% higher than those of WNAD-based algorithm, corresponding to the t_k-SA and GI task mapping algorithms respectively. For other three applications, the energy consumptions of WAND-based application mapping with t_k-SA and GI task mapping are also lower than those of NAD-based algorithm.

Another observation is that the t_k-SA algorithm outperforms the GI algorithm in task mapping for all applications. The energy consumptions of WNAD+GI mapping for application BASIZ, MPEG, GE and LE are respectively 42%, 15%, 29% and 54% higher than those of WNAD+t_k-SA mapping. Similarly, the energy consumptions of NAD+GI mapping for the applications are respectively 37%, 24%, 6% and 28% higher than those of NAD-t_k-SA mapping.

In all combinations of application mapping and task mapping algorithms, WNAD+t_k-SA achieves the lowest energy consumption. This result verifies the efficiency of the proposed application mapping and task mapping algorithms in minimizing the communication energy consumption of multiple applications.

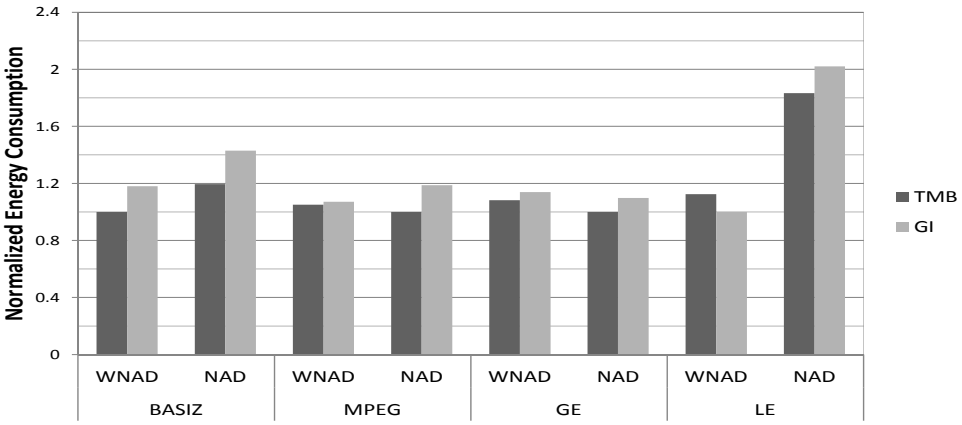


Figure 7.6: Normalized Communication Energy Consumption of Four Benchmarks with Different Mapping Methods for UM

7.4.3 Results of Bounded Mapping

Figure 7.7 illustrates the mapping results of four benchmarks on a 7×7 NoC in BM with WNAD-based application mapping algorithm. Since Section 7.4.2 already shows the effectiveness of WNAD-based application mapping algorithm compared to NAD-Based algorithm, here we focus on analyzing the effectiveness of TSB task mapping algorithm compared to NoP mapping. Table 7.2 illustrates the mapping and task sequence of benchmark GE as an example, using TSB and NoP mapping algorithms. The core at the bottom-left corner of a NoC is indexed as 0 and that at the top-right is indexed as $n-1$, where n is the size of the NoC. The second and third columns show the mapping and task sequence by WNAD+TSB and WNAD+NoP mapping respectively.

The communication energy consumption for different mapping algorithms is illustrated in Figure 7.8. We can observe that the energy consumption of TSB mapping is less than that of NoP mapping for all four benchmarks. The maximal reduction of energy consumption, about 57%, is in the case of application MPEG. The energy consumption is decreased by 49%, 33% and 28% for application BASIZ, GE and LE respectively.

The execution times of four benchmarks with different mapping methods in BM are illustrated in Figure 7.9. The TSB mapping achieves consider-

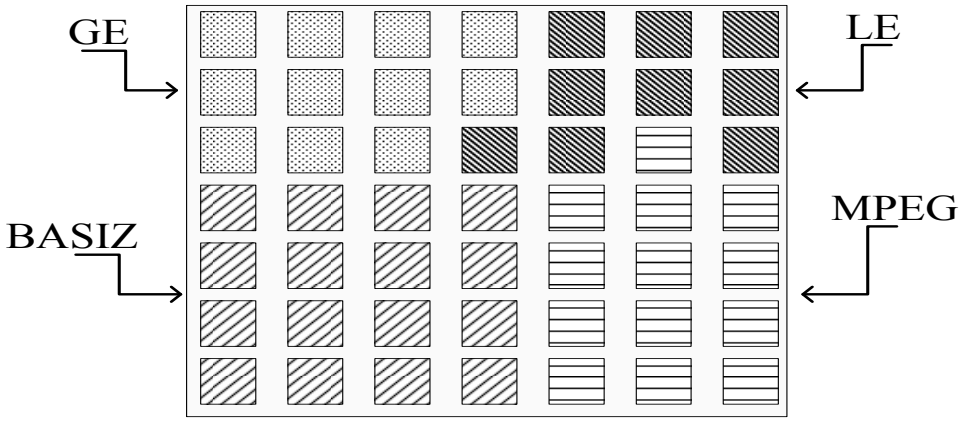


Figure 7.7: Result of Bounded Mapping with WNAD Application Mapping Algorithm on a 7×7 NoC

Core Index	WNAD+TSM	WNAD+NoP
28	4	15
29	3	4
30	16	2 16 17
35	0 2	7 11
36	6 8 11 10 14	0
37	1	6
38	13 15 17	5
42	12	8
43	7	1 12
44	9	10 13 14
45	5	3

Table 7.2: Mappings and Task Sequences of GE with Different Task Mapping Algorithms

able execution time reductions for applications BASIZ and GE with 52% and 40% respectively. The critical paths of these two applications are minimally influenced by different task mappings, thus the TSB mapping works effectively on reducing the length of critical paths. For applications MPEG and LE, they either have multiple critical paths or the critical path is easily changed during task mapping. TSB has limited effectiveness for these types of applications.

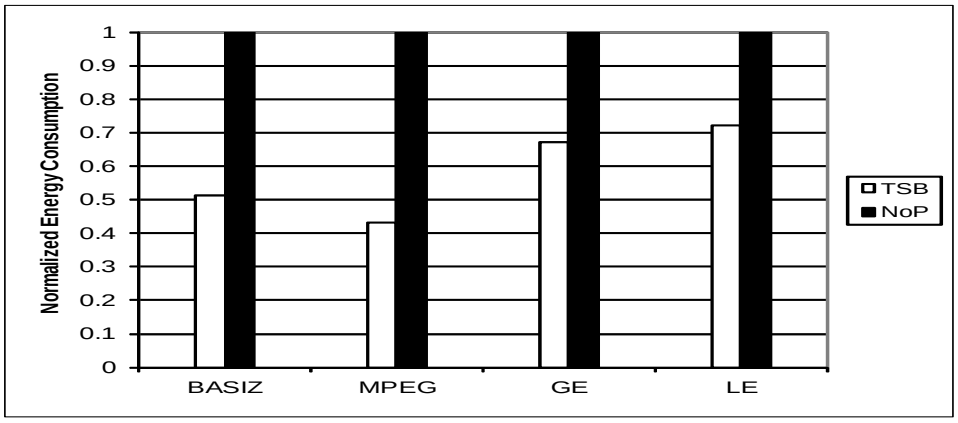


Figure 7.8: Communication Energy Consumption of Four Benchmarks with Different Mapping Methods in BM

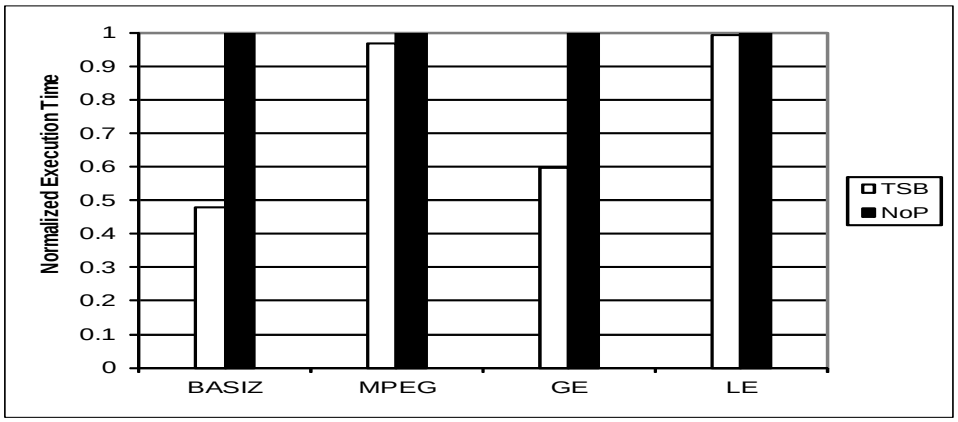


Figure 7.9: Execution Times of Four Benchmarks with Different Mapping Methods in Bounded Mapping

7.4.4 Experiment Summary

The quantitative evaluation demonstrates the effectiveness of the proposed two-step multi-application mapping methods in terms of energy consumption and execution time. For application mapping, the WNAD algorithm considerably reduces the communication energy with the four applications mapped on the 9×9 NoC. For task mapping, the t_k-SA algorithm outperforms the GI algorithm in terms of energy saving in the UM case. The proposed TSB algorithm is able to reduce the energy consumption and execution time for the BM. With the two steps combined, our adaptive mapping methods achieve significant energy and execution time saving for multi-application mapping with various number of cores. It should be noted

that the effectiveness of our algorithm is influenced by the characteristics of applications.

7.5 Chapter Summary

Many-core architectures enable simultaneous execution of multiple applications on a chip. A novel methodology for mapping multiple applications on many-core NoC platforms was proposed in this chapter. Composed of application mapping and task mapping, the two-step mapping methodology aims to minimize the communication energy consumption and the execution time of each application. The adaptive mapping methods for bounded or unbounded number of cores were presented respectively.

With an unbounded number of cores where each task can be allocated with a dedicated core, WNAD-Based application mapping algorithm and the t_k-SA task mapping algorithm minimize the communication energy consumption of multiple applications. For the bounded mapping where several tasks need to share one core, TSB task mapping algorithm not only minimizes the communication energy consumption, but also reduces the execution time overhead caused by core sharing. As demonstrated by the quantitative evaluation of practical benchmarks, the proposed two-step mapping methodology achieves smaller energy consumption and execution time compared to non-optimized mapping and existing works.

The proposed mapping methodology provides a general framework for mapping multiple applications, where the algorithms employed in both application and task mapping steps are not limited to those that have been proposed in this work. Following the two-step methodology, we can apply other optimization algorithms to application and task mapping with respect to specific design metrics.

Chapter 8

Conclusion

Many-core platforms have become the norm of massively parallel computing. One important issue in implementing parallel computing on a many-core platform is how to allocate the computation resource (IP cores) and the communication resource (communication channels) for a set of tasks which represent the target applications, i.e., the application mapping problem. Proper application mapping methods can minimize the energy consumption and improve the system performance. A general application mapping process consists of multiple stages, each of them focusing on a particular resource allocation problem. The collective effort of these stages leads to the overall optimal mappings. This thesis defines the general framework of application mapping, which integrates three optimization stages-IP selection, tile assignment and communication mapping. Methods of accelerating the general SA algorithm have been proposed to reduce the complexity of SA while keeping the good quality of mapping solutions. Additionally, a novel multi-application mapping methodology is proposed to perform multi-application mapping with bounded or unbounded number of IP cores.

8.1 Three-Stage Application Mapping

Given a set of applications and a many-core platform, application mapping determines to which resources on the platform a task will be allocated, and on which routing path in the communication network an inter-task communication will be routed. More concretely, on a heterogeneous many-core platform, application mapping can be carried out through three stages-IP selection, tile assignment and communication mapping. Each of them deals with a specific resource allocation problem. Minimizing energy consumption is the major objective of application mapping addressed in this thesis. IP selection selects an appropriate IP core for each task with the consideration of minimizing the energy consumed for the computation. Tile assignment

and communication mapping target the minimization of the energy consumed for the inter-core communication. Tile assignment determines which tile on the platform will be assigned to an IP core selected in the previous stage. This stage performs the physical placement of the selected IP cores. Tile assignment optimizes the core-to-node mapping in order to map cores with large amount of communication onto adjacent nodes, so that less energy would be consumed for communication. The communication mapping chooses one routing path for each inter-node communication by taking the traffic situation on channels (e.g., links on a NoC) into account, in order to balance the traffic and alleviate the network contention.

With each individual stage focusing on a particular resource allocation problem, the collective effort of the three stages results in an overall optimized mapping. A framework of application mapping which integrates the above three stages is proposed in this thesis. In addition to investigating and formulating individual operations of the three stages, the framework emphasizes on the inter-stage interaction to reach the overall optimal mappings.

8.2 Accelerating High-Performance Mapping Algorithms

Regarding the mapping algorithm, the performance and the complexity are the major metrics in selecting the appropriate algorithms employed in each mapping stage. Compared to other heuristic algorithms like GI and GA, as well as ILP method, SA algorithm has a better trade-off in complexity and mapping quality. However, there are still potentials to speed up the SA algorithm. Based on the study of the generic SA algorithm, it is observed that:

1. The set of parameters and functions applied in the SA algorithm jointly affects the annealing process and the quality of final mappings. The parameters and functions optimized for a specific mapping problem will speed up the annealing process without loss of the mapping quality. Additionally, the set of parameters has to be selected systematically, instead of being set independently.

2. When the annealing process starts from a randomly generated initial solution and a high initial temperature, the generic SA algorithm behaves randomly at the beginning and lots of iterations are wasted without optimization. It is possible to skip the random behavior and to speed up the SA algorithm by starting the annealing from an optimized initial mapping solution and a corresponding initial temperature.

According to these observations, two methods have been proposed for accelerating the SA algorithm in the application mapping. One way is to utilize an optimization method, called Nelder-Mead simplex method, to produce the set of optimized parameters applied in the SA algorithm. With the set of optimized parameters, fewer evaluation iterations are needed compared to the reference SA algorithm in which the set of parameters is set by the random values. The proposed method also demonstrates the necessity of utilizing the systematic approach to produce problem-specific parameters for the SA algorithm, instead of using randomly selected parameters for different problems. Another accelerating method is to start the annealing from an already optimized initial solution, with the appropriate initial temperature. The proposed t_k-SA algorithm can skip the random behavior at the beginning stages of the general SA algorithm. By doing this, the SA algorithm is accelerated and the quality of final solution is kept. The quantitative comparison shows that the SA algorithm utilizing the optimized parameters and functions only uses on average less than 1% iterations of that used in the SA algorithm using empirical parameters, to converge to the final optimized solution. With the optimized parameters, the proposed SA algorithm is on average 237 times faster than the SA algorithm using empirical parameters, while the optimal mapping is still found. Starting from the lower initial temperature t_k, the t_k-SA algorithm saves the iterations and runtime for finding the optimal mapping solution. The t_k-SA algorithm on average uses 58% iterations of that in the full-range SA algorithm. As a result, an average speedup of 1.55 is achieved by the t_k-SA algorithm. In addition, using the temperature t_k, the t_k-SA algorithm reproduces the behavior of the full-range SA algorithm. This enables the accelerated t_k-SA algorithm to obtain the mapping solution which has the same quality with that found by the full-range SA algorithm.

8.3 Towards Multi-Application Mapping

Mapping multiple applications simultaneously is an efficient way to utilize the abundant computation and communication resources on a many-core platform. The design focus should shift from single-application mapping to multi-application mapping. Compared to single-application mapping, there are more challenges in mapping multiple applications due to the resource sharing and competition between applications. It is prohibitively expensive to combine multiple task graphs as a whole and to solve the mapping problem with a large number of tasks with the SA, GA or ILP algorithms. In this thesis, we proposed a two-step multi-application mapping which deals with the problem in a divide-and-conquer way. Consisting of two steps, application mapping and task mapping, the proposed mapping method finds

an area on a many-core NoC for each application and then maps all tasks of the application into the area. The multi-application mapping adapts to the availability of IP cores on the NoC. The weighted NAD (WNAD) based algorithm is proposed for application mapping. The t_k-SA algorithm is applied to the task mapping in the case of unbounded number of cores. The task-sequence based (TSB) algorithm is proposed for the task mapping in the case of bounded number of cores. The quantitative comparison shows that the WNAD+t_k-SA mapping algorithms achieves the lowest communication energy consumption in all combinations of application mapping and task mapping algorithms evaluated. The proposed WNAD+TSB mapping algorithm not only minimizes the communication energy consumption, but also reduces the overhead caused by core sharing between multiple tasks.

8.4 Future Work

This thesis has explored the potential of achieving the optimal mapping by the three-stage framework of application mapping and the developed mapping algorithms. Several directions, however, should still be further investigated.

Firstly, the utilization of GA in application mapping needs to be promoted. Although GA, as a class of of evolutionary algorithm (EA), can theoretically produce better mapping solutions than SA algorithm, it is not yet widely used for application mapping due to its unaffordable computational complexity. Therefore, similar to the techniques which have been proposed for SA algorithm in this thesis, appropriate accelerating techniques need to be developed for GA. For instance, optimized crossover and mutation operators can be proposed so that the convergence of GA can be accelerated and the computational complexity of GA can be decreased.

Secondly, multi-objective mapping problems have to be investigated. The optimization objectives in application mapping are often multiple rather than single, and are sometimes conflicting with each other. For instance, performance and energy consumption are usually a trade-off in application mapping. For such multi-objective mapping problems, there is no single optimal solution, but a set of solutions featuring different trade-offs between the objectives (or Pareto front). To find the Pareto front, algorithms like SA, GA and ILP have to be applied. As discussed previously, the complexities of these algorithms have to be decreased to improve their feasibility.

Last but not least, more novel methods need to be developed for multi-application mapping. Compared to single-application mapping, there are more challenges for mapping multiple applications onto a many-core platform. One of them is how to efficiently allocate the shared resources (e.g., IP cores and communication network) to multiple applications and reduce

the inter-application interference due to the resource sharing. Another one is how to provide a design which can meet individual requirement and constraint, but not over-designed from the system's point of view. Although one methodology is innovated in this thesis, multi-application mapping is still an open problem and more effort needs to be put into this direction.

Bibliography

[1] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach, 4th Edition*. Morgan Kaufmann, 4 edition, September 2006.

[2] G. Blake, R.G. Dreslinski, and T. Mudge. A survey of multicore processors. *Signal Processing Magazine, IEEE*, 26(6):26 –37, november 2009.

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

[4] Gordon E Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):33–35, 1965.

[5] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital Integrated Circuits*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2008.

[6] M. Anis and M.H. Aburahma. Leakage current variability in nanometer technologies. In *System-on-Chip for Real-Time Applications, 2005. Proceedings. Fifth International Workshop on*, pages 60 – 63, july 2005.

[7] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78, Jan 2002.

[8] Lionel Torrens, Pascal Benoit, Gilles Sassatelli, and Michel Robert. *An Introduction to Multi-Core System on Chip-Trends and Challenges*, chapter 1, pages 1–21. Springer Science+Business Media, 2011.

[9] Kunle Olukotun, Lance Hammond, and James Laudon. Chip multiprocessor architecture: Techniques to improve throughput and latency. *Synthesis Lectures on Computer Architecture*, 2(1):1–145, 2007.

[10] Andrs Vajda. *Programming Many-Core Chips*, chapter Multi-core and Many-core Processor Architectures, pages 9–43. Springer US, 2011.

[11] Rainer Buchty, Vincent Heuveline, Wolfgang Karl, and Jan-Philipp Weiss. A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators. *Concurr. Comput. : Pract. Exper.*, 24(7):663–675, May 2012.

[12] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *Solid-State Circuits, IEEE Journal of*, 43(1):29–41, 2008.

[13] Tilera. Tile gx family, 2008.

[14] NVIDIA. Graphics processing units (gpus).

[15] AMD. Amd fusion apus.

[16] Intel. Sand bridge apus.

[17] R. Kumar, D.M. Tullsen, N.P. Jouppi, and P. Ranganathan. Heterogeneous chip multiprocessors. *Computer*, 38(11):32 – 38, nov. 2005.

[18] M.A. Qayum, N.A. Siddique, M.A. Haque, and A.S.M. Tayeen. Future of multiprocessors: Heterogeneous chip multiprocessors. In *Informatics, Electronics Vision (ICIEV), 2012 International Conference on*, pages 372 –376, may 2012.

[19] Fayez Gebali. *Algorithms and Parallel Computing.* John Wiley & Sons, 2011.

[20] Harry F. Jordan and Gita Alaghband. Fundamentals of Parallel Processing. 2003.

[21] TOBIAS BJERREGAARD and SHANKAR MAHADEVAN. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38, 2006.

[22] Yu K. Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, December 1999.

[23] Oliver Sinnen and Leonel Sousa. A platform independent parallelising tool based on graph theoretic models. In *Selected Papers and Invited Talks from the 4th International Conference on Vector and Parallel Processing*, VECPAR '00, pages 154–167, London, UK, UK, 2001. Springer-Verlag.

[24] Concepcio Roig, Ana Ripoll, and Fernando Guirado. A new task graph model for mapping message passing applications. *IEEE Trans. Parallel Distrib. Syst.*, 18(12):1740–1753, December 2007.

[25] Jingcao Hu and R. Marculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 688 – 693, 2003.

[26] Jingcao Hu and R. Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. In *Proceedings of the ASP-DAC 2003. Asia and South Pacific*, pages 233 – 239, jan. 2003.

[27] Tang Lei and Shashi Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *DSD '03: Proceedings of the Euromicro Symposium on Digital Systems Design*, page 180, Washington, DC, USA, 2003. IEEE Computer Society.

[28] Srinivasan Murali and Giovanni De Micheli. Bandwidth-constrained mapping of cores onto noc architectures. In *Proceedings of the conference on Design, automation and test in Europe - Volume 2*, DATE '04, pages 20896–. IEEE Computer Society, 2004.

[29] Chen-Ling Chou and R. Marculescu. Contention-aware application mapping for network-on-chip communication architectures. In *Proc. IEEE International Conference on Computer Design ICCD 2008*, pages 164–169, October 12–15, 2008.

[30] C.A.M. Marcon, E.I. Moreno, N.L.V. Calazans, and F.G. Moraes. Comparison of network-on-chip mapping algorithms targeting low energy consumption. *IET Computers & Digital Techniques*, 2(6):471–482, 2008.

[31] Heikki Orsila, Erno Salminen, and Timo D. Hämäläinen. Best practices for simulated annealing in multiprocessor task distribution problems. *Simulated Annealing, I-Tech Education and Publishing KG.*, pages 321–342, 2008.

[32] Marcus Vinícius Carvalho da Silva, Nadia Nedjah, and Luiza de Macedo Mourelle. Power-aware multi-objective evolutionary optimization for application mapping on noc platforms. In *Proceedings of the 23rd international conference on Industrial engineering and other applications of applied intelligent systems - Volume Part II*, IEA/AIE'10, pages 143–152, Berlin, Heidelberg, 2010. Springer-Verlag.

[33] Bo Yang, Thomas Canhao Xu, Tero Säntti, and Juha Plosila. Tree-model based mapping for energy-efficient and low-latency network-on-chip. In *Proceeding of Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 189 –192, 14-16 2010.

[34] Bo Yang, Liang Guang, Tero Säntti, and Juha Plosila. Parameter-optimized simulated annealing for application mapping on networks-on-chip. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 307–322. Springer Berlin Heidelberg, 2012.

[35] Bo Yang, Liang Guang, Tero Säntti, and Juha Plosila. t(k)-sa: accelerated simulated annealing algorithm for application mapping on networks-on-chip. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 1191–1198, New York, NY, USA, 2012. ACM.

[36] Bo Yang, Liang Guang, T.C. Xu, A.W. Yin, T. Säntti, and J. Plosila. Multi-application multi-step mapping method for many-core network-on-chips. In *Proceeding of NORCHIP, 2010*, pages 1 –6, nov. 2010.

[37] Bo Yang, Liang Guang, T.C. Xu, T. Säntti, and J. Plosila. Multi-application mapping algorithm for network-on-chip platforms. In *Proceeding of Electrical and Electronics Engineers in Israel (IEEEI), 2010 IEEE 26th Convention of*, pages 000540 –000544, nov. 2010.

[38] Bo Yang, Liang Guang, Tero Sntti, and Juha Plosila. Mapping multiple applications with unbounded and bounded number of cores on many-core networks-on-chip. *Microprocessors and Microsystems*, 37(4):460 – 471, 2013.

[39] M. Horowitz and W. Dally. How scaling will change processor architecture. In *Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International*, pages 132 – 133 Vol.1, feb. 2004.

[40] Dong Hyuk Woo and H.-H.S. Lee. Extending amdahl's law for energy-efficient computing in the many-core era. *Computer*, 41(12):24–31, 2008.

[41] Zhiyi Yu. *CMOS Processors and Memories*, chapter Towards High-Performance and Energy-Efficient Multi-core Processors, pages 29–51. Springer Netherlands, 2010.

[42] International Technology Roadmap for Semiconductors. Itrs 2011 edition, 2011.

[43] C.H. (Kees) van Berkel. Multi-core for mobile phones. In *Design Automation and Test Europe Conference (DATE)*, 2009.

[44] David Culler, Jaswinder Pal Singh, and Anoop Gupta. *Parallel Computer Architecture: A Hardware/Software Approach.* Morgan Kaufmann Publishers, 1998.

[45] Thomas Rauber and Gudula Rünger, editors. *Parallel Programming for Multicore and Cluster Systems.* Springer Berlin Heidelberg, 2010.

[46] James R. Goodman. Using cache memory to reduce processor-memory traffic. In *Proceedings of the 10th annual international symposium on Computer architecture*, ISCA '83, pages 124–131, New York, NY, USA, 1983. ACM.

[47] Michael E. Thomadakis. The architecture of the nehalem processor and nehalem-ep smp platforms. Technical report, Supercomputing Facility, Texas A&M University, 2011.

[48] AMD. Amd opteron 6000 series processors, 2013.

[49] Oracle. Oracle's sparc t3-1, sparc t3-2, sparc t3-4 and sparc t3-1b server architecture. Technical report, Oracle, 2011.

[50] IBM. Ibm power 7 processor, 2013.

[51] Barbara Chapman, Gabriele Jost, and Ruud Van Der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming.* The MIT Press, 2007.

[52] L. E. Jordan and Gita Alaghband. *Fundamentals of Parallel Processing.* Prentice Hall Professional Technical Reference, 2002.

[53] Hesham El-Rewini and Mostafa Abd-El-Barr. *Advanced Computer Architecture and Parallel Processing.* John Wiley & Sons, Inc., 2005.

[54] Rakesh Kumar, Timothy G. Mattson, Gilles Pokam, and Rob F. Van der Wijngaart. The case for message passing on many-core chips. In Michael Hübner and Jürgen Becker, editors, *Multiprocessor System-on-Chip*, pages 115–123. Springer, 2011.

[55] Xiaowen Chen, Zhonghai Lu, A. Jantsch, and Shuming Chen. Supporting distributed shared memory on multi-core network-on-chips using a dual microcoded controller. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 39–44, 2010.

[56] O.S. Lawlor. Message passing for gpgpu clusters: Cudampi. In *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pages 1–8, 2009.

[57] J. Howard, S. Dighe, S.R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V.K. De, and R. Van Der Wijngaart. A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling. *Solid-State Circuits, IEEE Journal of*, 46(1):173–183, Jan.2011.

[58] Zhiyi Yu, Kaidi You, Ruijin Xiao, Heng Quan, Peng Ou, Yan Ying, Haofan Yang, Ming"e Jing, and Xiaoyang Zeng. An 800mhz 320mw 16-core processor with message-passing and shared-memory inter-core communication mechanisms. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pages 64–66, 2012.

[59] J. Diaz, C. Munoz-Caro, and A. Nino. A survey of parallel programming models and tools in the multi and many-core era. *Parallel and Distributed Systems, IEEE Transactions on*, 23(8):1369–1386, Aug.2012.

[60] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[61] A.W. Yin, T.C. Xu, Bo Yang, P. Liljeberg, and H. Tenhunen. Change function of 2d/3d network-on-chip. In *Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on*, pages 181–188, 31 2011-Sept. 2.

[62] Christopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. *J. ACM*, 41(5):874–902, September 1994.

[63] Wen-Chung Tsai, Kuo-Chih Chu, Yu-Hen Hu, and Sao-Jie Chen. Non-minimal, turn-model based noc routing. *Microprocessors and Microsystems*, (0):–, 2012.

[64] William J. Dally and Brian Towles. Route packets, not wires: on-chip inteconnection networks. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 684–689, New York, NY, USA, 2001. ACM.

[65] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F.

Brown III, and Anant Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5):15–31, 2007.

[66] D. Truong, W. Cheng, T. Mohsenin, Zhiyi Yu, T. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, P. Mejia, Anh Tran, J. Webb, E. Work, Zhibin Xiao, and B. Baas. A 167-processor 65 nm computational platform with per-processor dynamic supply voltage and dynamic clock frequency scaling. In *VLSI Circuits, 2008 IEEE Symposium on*, pages 22 –23, 2008.

[67] G. De Micheli, C. Seiculescu, S. Murali, L. Benini, F. Angiolini, and A. Pullini. Networks on chips: From research to products. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 300–305, 2010.

[68] B.S. Feero and P.P. Pande. Networks-on-chip in a three-dimensional environment: A performance evaluation. *Computers, IEEE Transactions on*, 58(1):32–45, Jan.

[69] Vitor de Paulo and Cristinel Ababei. 3d network-on-chip architectures using homogeneous meshes and heterogeneous floorplans. *Int. J. Reconfig. Comput.*, 2010:1:1–1:12, January 2010.

[70] N.S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, Dec.2003.

[71] Lin Yuan, S. Leventhal, and Gang Qu. Temperature-aware leakage minimization technique for real-time systems. In *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on*, pages 761–764, Nov.2006.

[72] M. Pedram and S. Nazarian. Thermal modeling, analysis, and management in vlsi circuits: Principles and methods. *Proceedings of the IEEE*, 94(8):1487–1501, Aug.2006.

[73] Liang Guang, Ethiopia Nigussie, Juha Plosila, Jouni Isoaho, and Hannu Tenhunen. Survey of self-adaptive nocs with energy-efficiency and dependability. *International Journal of Embedded and Real-Time Communication Systems (IJERTCS)*, 3(2):122, 2012.

[74] G. Dhiman and T.S. Rosing. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 207–212, Aug.2007.

[75] Liang Guang, Ethiopia Nigussie, Lauri Koskinen, and Hannu Tenhunen. Autonomous dvfs on supply islands for energy-constrained noc communication. In *Proceedings of the 22nd International Conference on Architecture of Computing Systems*, ARCS '09, pages 183–194, Berlin, Heidelberg, 2009. Springer-Verlag.

[76] Wooyoung Jang, Duo Ding, and D.Z. Pan. Voltage and frequency island optimizations for many-core/networks-on-chip designs. In *Green Circuits and Systems (ICGCS), 2010 International Conference on*, pages 217–220, June, 2010.

[77] Changbo Long, Jinjun Xiong, and Yongpan Liu. Techniques of power-gating to kill sub-threshold leakage. In *Circuits and Systems, 2006. APCCAS 2006. IEEE Asia Pacific Conference on*, pages 952–955, Dec.2006.

[78] Xuning Chen and Li-Shiuan Peh. Leakage power modeling and optimization in interconnection networks. In *Low Power Electronics and Design, 2003. ISLPED '03. Proceedings of the 2003 International Symposium on*, pages 90–95, Aug.2003.

[79] V. Soteriou and Li-Shiuan Peh. Exploring the design space of self-regulating power-aware on/off interconnection networks. *Parallel and Distributed Systems, IEEE Transactions on*, 18(3):393–408, March,2007.

[80] H. Matsutani, M. Koibuchi, H. Amano, and D. Wang. Run-time power gating of on-chip routers using look-ahead routing. In *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, pages 55–60, March, 2008.

[81] Muhammad Qadri, Hemal Gujarathi, and Klaus McDonald-Maier. Low power processor architectures and contemporary techniques for power optimization a review. *Journal of Computers*, 4(10), 2009.

[82] R. Mullins. Minimising dynamic power consumption in on-chip networks. In *System-on-Chip, 2006. International Symposium on*, pages 1–4, Nov.2006.

[83] R. Marculescu, U.Y. Ogras, Li-Shiuan Peh, N.E. Jerger, and Y. Hoskote. Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(1):3 –21, jan. 2009.

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

[85] Dongkun Shin and Jihong Kim. Power-aware communication optimization for networks-on-chips with voltage scalable links. In *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '04, pages 170–175, New York, NY, USA, 2004. ACM.

[86] K. Srinivasan, K. S. Chatha, and G. Konjevod. An automated technique for topology and route generation of application specific on-chip interconnection networks. In *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, ICCAD '05, pages 231–237, Washington, DC, USA, 2005. IEEE Computer Society.

[87] Ciprian Radu and Lucian Vinţan. Optimized simulated annealing for network-on-chip application mapping. In *Proceedings of the 18th International Conference on Control Systems and Computer Science (CSCS-18), Bucharest, Romania*, volume 1, pages 452–459, Bucharest, Romania, May 24-27 2011. Politehnica Press.

[88] Jia Huang, C. Buckl, A. Raabe, and A. Knoll. Energy-aware task allocation for network-on-chip based heterogeneous multiprocessor systems. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, pages 447 –454, feb. 2011.

[89] Zhonghai Lu, Lei Xia, and Axel Jantsch. Cluster-based simulated annealing for mapping cores onto 2d mesh networks on chip. In *Proceedings of the 2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pages 1–6, 2008.

[90] Andreas Hansson, Kees Goossens, and Andrei Rdulescu. A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic. In *VLSI Design*, volume 2007, page 16. Hindawi Publishing Corporation, 2007.

[91] E. Carvalho and F. Moraes. Congestion-aware task mapping in heterogeneous mpsocs. In *Proc. International Symposium on System-on-Chip SOC 2008*, pages 1–4, November 5–6, 2008.

[92] Liulin Zhong, Jiayi Sheng, Ming'e Jing, Zhiyi Yu, Xiaoyang Zeng, and Dian Zhou. An optimized mapping algorithm based on simulated annealing for regular noc architecture. In *ASIC (ASICON), 2011 IEEE 9th International Conference on*, pages 389–392, 2011.

[93] Ewerson Carvalho, César Marcon, Ney Calazans, and Fernando Moraes. Evaluation of static and dynamic task mapping algorithms in noc-based mpsocs. In *Proceedings of the 11th international conference on System-on-chip*, SOC'09, pages 87–90, Piscataway, NJ, USA, 2009. IEEE Press.

[94] Robert Dick. Embedded systems synthesis benchmark suite (e3s). *http://ziyang.eecs.umich.edu/ dickrp/e3s/*.

[95] Vincenzo Catania Giuseppe Ascia and Maurizio Palesi. Mapping cores on network-on-chip. *International Journal of Computational Intelligence Research*, Vol.1, No.2:109–126, 2005.

[96] Dragos Truscan, Tiberiu Seceleanu, Johan Lilius, and Hannu Tenhunen. A model-based design process for the segbus distributed architecture. In *Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 307–316, Washington, DC, USA, 2008. IEEE Computer Society.

[97] C. Marcon, A. Borin, A. Susin, L. Carro, and F. Wagner. Time and energy efficient mapping of embedded applications onto nocs. In *Proc. Asia and South Pacific Design Automation Conference the ASP-DAC 2005*, volume 1, pages 33–38, January 18–21, 2005.

[98] Suleyman Tosun. New heuristic algorithms for energy aware application mapping and routing on mesh-based nocs. *J. Syst. Archit.*, 57(1):69–78, January 2011.

[99] K. Vivekanandarajah and S.K. Pilakkat. Task mapping in heterogeneous mpsocs for system level design. In *Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on*, pages 56 –65, 31 2008-april 3 2008.

[100] Liyang Zhou, Ming'e Jing, Liulin Zhong, Zhiyi Yu, and Xiaoyang Zeng. Task-binding based branch-and-bound algorithm for noc mapping. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 648 –651, may 2012.

[101] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[102] P. S. Oliveto J. He and X. Yao. Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, pages 100–106, 2007.

[103] Maria Angelova and Tania Pencheva. Tuning genetic algorithm parameters to improve convergence time. *International Journal of Chemical Engineering*, 2011:7, 2011.

[104] Rabindra Ku Jena and Prabat K. Mahanti. Design space exploration of network-on-chip: A system level approach. *International Journal of Computing and ICT Research*, 2:17–25, 2008.

[105] M.V.C. da Silva, N. Nedjah, and L. de Macedo Mourelle. Evolutionary ip assignment for efficient noc-based system design using multi-objective optimization. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 2257 –2264, may 2009.

[106] B. Suman and P. Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57(18):1143–1160, 2006.

[107] S. Tosun, O. Ozturk, and M. Ozen. An ilp formulation for application mapping onto network-on-chips. In *Application of Information and Communication Technologies, 2009. AICT 2009. International Conference on*, pages 1 –5, oct. 2009.

[108] Jiayi Sheng, Liulin Zhong, Ming'e Jing, Zhiyi Yu, and Xiaoyang Zeng. A method of quadratic programming for mapping on noc architecture. In *ASIC (ASICON), 2011 IEEE 9th International Conference on*, pages 200 –203, oct. 2011.

[109] R. Setiono and Huan Liu. A connectionist approach to generating oblique decision trees. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 29(3):440–444, Jun 1999.

[110] D. Shasha, J.T.L. Wang, Huiyuan Shan, and Kaizhong Zhang. Atreegrep: approximate searching in unordered trees. In *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*, pages 89–98, 2002.

[111] E. John, F. Hudson, and L.K. John. Hybrid tree: A scalable optoelectronic interconnection network for parallel computing. *Hawaii International Conference on System Sciences*, 7:466, 1998.

[112] Jaswinder Pal Singh, Anoop Gupta, Moriyoshi Ohara, Evan Torrie, and Steven Cameron Woo. The splash-2 programs: Characterization and methodological considerations. *Computer Architecture, International Symposium on*, 0:24, 1995.

[113] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, November 2005.

[114] University of Catania. Noxim. *http://www.noxim.org/*.

[115] Bradford M. Beckmann and David A. Wood. Managing wire delay in large chip-multiprocessor caches. In *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, pages 319–330, December 2004.

[116] Avadh Patel and Kanad Ghose. Energy-efficient mesi cache coherence with pro-active snoop filtering for multicore microprocessors. In *Proceeding of the thirteenth international symposium on Low power electronics and design*, pages 247–252, August 2008.

[117] H. Orsila, E. Salminen, and T.D. Hamalainen. Parameterizing simulated annealing for distributing kahn process networks on multiprocessor socs. In *System-on-Chip, 2009. SOC 2009. International Symposium on*, pages 019 –026, oct. 2009.

[118] J.A.Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[119] Moon-Won Park and Yeong-Dae Kim. A systematic procedure for setting parameters in simulated annealing algorithms. *Comput. Oper. Res.*, 25:207–217, March 1998.

[120] SLD:: System Level Design Group @ CMU. Nocmap: an energy- and performance-aware mapping tool for networks-on-chip, http://www.ece.cmu.edu/ sld/software/nocmap.php.

[121] S. Murali and G. De Micheli. Sunmap: a tool for automatic topology selection and generation for nocs. In *Design Automation Conference, 2004. Proceedings. 41st*, pages 914 –919, july 2004.

[122] E.B. van der Tol, E.G.T. Jaspers, and R.H. Gelderblom. Mapping of h.264 decoding on a multiprocessor architecture. In *Image and Video Communications and Processing*, pages 707–718, 2003.

[123] P.J.M. Laarhoven and E.H.L. Aarts. *Simulated annealing: theory and applications*. Mathematics and its applications. D. Reidel, 1987.

[124] Concepcio Roig, Ana Ripoll, and Fernando Guirado. A new task graph model for mapping message passing applications. *Transactions on Parallel and Distributed Systems*, 18(12):1740–1753, 2007.

[125] Krste Asanovic, Ras Bodik, Bryan C. Catanzaro, Joseph J. Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William L. Plishker, John Shalf, Samuel W. Williams, and Katherine A. Yelick. The landscape of parallel computing research: a view from berkeley. (UCB/EECS-2006-183), December 2006.

[126] Francisco Triviño, José L. Sánchez, Francisco J. Alfaro, and José Flich. Virtualizing network-on-chip resources in chip-multiprocessors. *Microprocess. Microsyst.*, 35(2):230–245, March 2011.

[127] Srinivasan Murali, Martijn Coenen, Andrei Radulescu, Kees Goossens, and Giovanni De Micheli. Mapping and configuration methods for multi-use-case networks on chips. In *ASP-DAC '06: Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, pages 146–151, Piscataway, NJ, USA, 2006. IEEE Press.

[128] L. Ost, G. Guindani, F.G. Moraes, L.S. Indrusiak, and S. Matt. Exploring noc-based mpsoc design space with power estimation models. *Design Test of Computers, IEEE*, 28(2):16 –29, march-april 2011.

[129] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast template placement for reconfigurable computing systems. *Design Test of Computers, IEEE*, 17(1):68 –83, jan-mar 2000.

[130] L.M. Ni and P.K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62 –76, February 1993.

[131] Liang Guang, E. Nigussie, J. Plosila, J. Isoaho, and H. Tenhunen. Hls-donoc: High-level simulator for dynamically organizational nocs. In *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2012 IEEE 15th International Symposium on*, pages 89–94, 2012.

[132] Andrew B. Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. Orion 2.0: a fast and accurate noc power and area model for early-stage design space exploration. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 423–428, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[133] Gul Khan and Usman Ahmed. Cad tool for hardware software co-synthesis ofheterogeneous multiple processor embedded architectures. *Design Automation for Embedded Systems*, 12:313–343, 2008. 10.1007/s10617-008-9031-1.

[134] Min you Wu and Daniel D. Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Trans. on Parallel and Distributed Systems*, 1:330–343, 1990.

[135] C.A.M. Marcon, E.I. Moreno, N.L.V. Calazans, and F.G. Moraes. Evaluation of algorithms for low energy mapping onto nocs. In *Proc. IEEE International Symposium on Circuits and Systems ISCAS 2007*, pages 389–392, 2007.

Turku Centre for Computer Science
TUCS Dissertations

1. **Marjo Lipponen**, On Primitive Solutions of the Post Correspondence Problem
2. **Timo Käkölä**, Dual Information Systems in Hyperknowledge Organizations
3. **Ville Leppänen**, Studies on the Realization of PRAM
4. **Cunsheng Ding**, Cryptographic Counter Generators
5. **Sami Viitanen**, Some New Global Optimization Algorithms
6. **Tapio Salakoski**, Representative Classification of Protein Structures
7. **Thomas Långbacka**, An Interactive Environment Supporting the Development of Formally Correct Programs
8. **Thomas Finne**, A Decision Support System for Improving Information Security
9. **Valeria Mihalache**, Cooperation, Communication, Control. Investigations on Grammar Systems.
10. **Marina Waldén**, Formal Reasoning About Distributed Algorithms
11. **Tero Laihonen**, Estimates on the Covering Radius When the Dual Distance is Known
12. **Lucian Ilie**, Decision Problems on Orders of Words
13. **Jukkapekka Hekanaho**, An Evolutionary Approach to Concept Learning
14. **Jouni Järvinen**, Knowledge Representation and Rough Sets
15. **Tomi Pasanen**, In-Place Algorithms for Sorting Problems
16. **Mika Johnsson**, Operational and Tactical Level Optimization in Printed Circuit Board Assembly
17. **Mats Aspnäs**, Multiprocessor Architecture and Programming: The Hathi-2 System
18. **Anna Mikhajlova**, Ensuring Correctness of Object and Component Systems
19. **Vesa Torvinen**, Construction and Evaluation of the Labour Game Method
20. **Jorma Boberg**, Cluster Analysis. A Mathematical Approach with Applications to Protein Structures
21. **Leonid Mikhajlov**, Software Reuse Mechanisms and Techniques: Safety Versus Flexibility
22. **Timo Kaukoranta**, Iterative and Hierarchical Methods for Codebook Generation in Vector Quantization
23. **Gábor Magyar**, On Solution Approaches for Some Industrially Motivated Combinatorial Optimization Problems
24. **Linas Laibinis**, Mechanised Formal Reasoning About Modular Programs
25. **Shuhua Liu**, Improving Executive Support in Strategic Scanning with Software Agent Systems
26. **Jaakko Järvi**, New Techniques in Generic Programming – C++ is more Intentional than Intended
27. **Jan-Christian Lehtinen**, Reproducing Kernel Splines in the Analysis of Medical Data
28. **Martin Büchi**, Safe Language Mechanisms for Modularization and Concurrency
29. **Elena Troubitsyna**, Stepwise Development of Dependable Systems
30. **Janne Näppi**, Computer-Assisted Diagnosis of Breast Calcifications
31. **Jianming Liang**, Dynamic Chest Images Analysis
32. **Tiberiu Seceleanu**, Systematic Design of Synchronous Digital Circuits
33. **Tero Aittokallio**, Characterization and Modelling of the Cardiorespiratory System in Sleep-Disordered Breathing
34. **Ivan Porres**, Modeling and Analyzing Software Behavior in UML
35. **Mauno Rönkkö**, Stepwise Development of Hybrid Systems
36. **Jouni Smed**, Production Planning in Printed Circuit Board Assembly
37. **Vesa Halava**, The Post Correspondence Problem for Market Morphisms
38. **Ion Petre**, Commutation Problems on Sets of Words and Formal Power Series
39. **Vladimir Kvassov**, Information Technology and the Productivity of Managerial Work
40. **Frank Tétard**, Managers, Fragmentation of Working Time, and Information Systems

41. **Jan Manuch**, Defect Theorems and Infinite Words
42. **Kalle Ranto**, Z_4-Goethals Codes, Decoding and Designs
43. **Arto Lepistö**, On Relations Between Local and Global Periodicity
44. **Mika Hirvensalo**, Studies on Boolean Functions Related to Quantum Computing
45. **Pentti Virtanen**, Measuring and Improving Component-Based Software Development
46. **Adekunle Okunoye**, Knowledge Management and Global Diversity – A Framework to Support Organisations in Developing Countries
47. **Antonina Kloptchenko**, Text Mining Based on the Prototype Matching Method
48. **Juha Kivijärvi**, Optimization Methods for Clustering
49. **Rimvydas Rukšėnas**, Formal Development of Concurrent Components
50. **Dirk Nowotka**, Periodicity and Unbordered Factors of Words
51. **Attila Gyenesei**, Discovering Frequent Fuzzy Patterns in Relations of Quantitative Attributes
52. **Petteri Kaitovaara**, Packaging of IT Services – Conceptual and Empirical Studies
53. **Petri Rosendahl**, Niho Type Cross-Correlation Functions and Related Equations
54. **Péter Majlender**, A Normative Approach to Possibility Theory and Soft Decision Support
55. **Seppo Virtanen**, A Framework for Rapid Design and Evaluation of Protocol Processors
56. **Tomas Eklund**, The Self-Organizing Map in Financial Benchmarking
57. **Mikael Collan**, Giga-Investments: Modelling the Valuation of Very Large Industrial Real Investments
58. **Dag Björklund**, A Kernel Language for Unified Code Synthesis
59. **Shengnan Han**, Understanding User Adoption of Mobile Technology: Focusing on Physicians in Finland
60. **Irina Georgescu**, Rational Choice and Revealed Preference: A Fuzzy Approach
61. **Ping Yan**, Limit Cycles for Generalized Liénard-Type and Lotka-Volterra Systems
62. **Joonas Lehtinen**, Coding of Wavelet-Transformed Images
63. **Tommi Meskanen,** On the NTRU Cryptosystem
64. **Saeed Salehi**, Varieties of Tree Languages
65. **Jukka Arvo**, Efficient Algorithms for Hardware-Accelerated Shadow Computation
66. **Mika Hirvikorpi**, On the Tactical Level Production Planning in Flexible Manufacturing Systems
67. **Adrian Costea**, Computational Intelligence Methods for Quantitative Data Mining
68. **Cristina Seceleanu**, A Methodology for Constructing Correct Reactive Systems
69. **Luigia Petre**, Modeling with Action Systems
70. **Lu Yan**, Systematic Design of Ubiquitous Systems
71. **Mehran Gomari**, On the Generalization Ability of Bayesian Neural Networks
72. **Ville Harkke**, Knowledge Freedom for Medical Professionals – An Evaluation Study of a Mobile Information System for Physicians in Finland
73. **Marius Cosmin Codrea**, Pattern Analysis of Chlorophyll Fluorescence Signals
74. **Aiying Rong**, Cogeneration Planning Under the Deregulated Power Market and Emissions Trading Scheme
75. **Chihab BenMoussa**, Supporting the Sales Force through Mobile Information and Communication Technologies: Focusing on the Pharmaceutical Sales Force
76. **Jussi Salmi**, Improving Data Analysis in Proteomics
77. **Orieta Celiku**, Mechanized Reasoning for Dually-Nondeterministic and Probabilistic Programs
78. **Kaj-Mikael Björk**, Supply Chain Efficiency with Some Forest Industry Improvements
79. **Viorel Preoteasa**, Program Variables – The Core of Mechanical Reasoning about Imperative Programs
80. **Jonne Poikonen**, Absolute Value Extraction and Order Statistic Filtering for a Mixed-Mode Array Image Processor
81. **Luka Milovanov**, Agile Software Development in an Academic Environment
82. **Francisco Augusto Alcaraz Garcia**, Real Options, Default Risk and Soft Applications
83. **Kai K. Kimppa**, Problems with the Justification of Intellectual Property Rights in Relation to Software and Other Digitally Distributable Media
84. **Dragoş Truşcan**, Model Driven Development of Programmable Architectures
85. **Eugen Czeizler**, The Inverse Neighborhood Problem and Applications of Welch Sets in Automata Theory

86. **Sanna Ranto**, Identifying and Locating-Dominating Codes in Binary Hamming Spaces
87. **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
88. **Elena Czeizler**, Intricacies of Word Equations
89. **Marcus Alanen**, A Metamodeling Framework for Software Engineering
90. **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
91. **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
92. **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
93. **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
94. **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
95. **Kim Solin**, Abstract Algebra of Program Refinement
96. **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
97. **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
98. **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
99. **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
100. **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
101. **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
102. **Chang Li**, Parallelism and Complexity in Gene Assembly
103. **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
104. **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
105. **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
106. **Anna Sell**, Mobile Digital Calendars in Knowledge Work
107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Säntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
109. **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
110. **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
111. **Camilla J. Hollanti**, Order-Theoretic Mehtods for Space-Time Coding: Symmetric and Asymmetric Designs
112. **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
113. **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
114. **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
115. **Petri Salmela**, On Commutation and Conjugacy of Rational Languages and the Fixed Point Method
116. **Siamak Taati**, Conservation Laws in Cellular Automata
117. **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
118. **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
119. **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
120. **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
121. **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
122. **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
123. **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
124. **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
125. **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
126. **Tuomo Saarni**, Segmental Durations of Speech
127. **Johannes Eriksson**, Tool-Supported Invariant-Based Programming

128. **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
129. **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
130. **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
131. **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
132. **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
133. **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
134. **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
135. **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
136. **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
137. **Ville Junnila**, On Identifying and Locating-Dominating Codes
138. **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
139. **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
140. **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems
141. **Arho Suominen**, Notes on Emerging Technologies
142. **József Mezei**, A Quantitative View on Fuzzy Numbers
143. **Marta Olszewska**, On the Impact of Rigorous Approaches on the Quality of Development
144. **Antti Airola**, Kernel-Based Ranking: Methods for Learning and Performace Estimation
145. **Aleksi Saarela**, Word Equations and Related Topics: Independence, Decidability and Characterizations
146. **Lasse Bergroth**, Kahden merkkijonon pisimmän yhteisen alijonon ongelma ja sen ratkaiseminen
147. **Thomas Canhao Xu**, Hardware/Software Co-Design for Multicore Architectures
148. **Tuomas Mäkilä**, Software Development Process Modeling – Developers Perspective to Contemporary Modeling Techniques
149. **Shahrokh Nikou**, Opening the Black-Box of IT Artifacts: Looking into Mobile Service Characteristics and Individual Perception
150. **Alessandro Buoni**, Fraud Detection in the Banking Sector: A Multi-Agent Approach
151. **Mats Neovius**, Trustworthy Context Dependency in Ubiquitous Systems
152. **Fredrik Degerlund**, Scheduling of Guarded Command Based Models
153. **Amir-Mohammad Rahmani-Sane**, Exploration and Design of Power-Efficient Networked Many-Core Systems
154. **Ville Rantala**, On Dynamic Monitoring Methods for Networks-on-Chip
155. **Mikko Pelto**, On Identifying and Locating-Dominating Codes in the Infinite King Grid
156. **Anton Tarasyuk**, Formal Development and Quantitative Verification of Dependable Systems
157. **Muhammad Mohsin Saleemi**, Towards Combining Interactive Mobile TV and Smart Spaces: Architectures, Tools and Application Development
158. **Tommi J. M. Lehtinen**, Numbers and Languages
159. **Peter Sarlin**, Mapping Financial Stability
160. **Alexander Wei Yin**, On Energy Efficient Computing Platforms
161. **Mikołaj Olszewski**, Scaling Up Stepwise Feature Introduction to Construction of Large Software Systems
162. **Maryam Kamali**, Reusable Formal Architectures for Networked Systems
163. **Zhiyuan Yao**, Visual Customer Segmentation and Behavior Analysis – A SOM-Based Approach
164. **Timo Jolivet**, Combinatorics of Pisot Substitutions
165. **Rajeev Kumar Kanth**, Analysis and Life Cycle Assessment of Printed Antennas for Sustainable Wireless Systems
166. **Khalid Latif**, Design Space Exploration for MPSoC Architectures

167. **Bo Yang**, Towards Optimal Application Mapping for Energy-Efficient Many-Core Platforms

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www. tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics and Statistics

Turku School of Economics
- Institute of Information Systems Science

Åbo Akademi University
Division for Natural Sciences and Technology
- Department of Information Technologies

ISBN 978-952-12-2982-4
ISSN 1239-1883

Bo Yang

Towards Optimal Application Mapping for Energy-Efficient Many-Core Platforms