TECHNICAL UNIVERSITY OF CATALONIA (UPC)
DEPARTMENT OF SOFTWARE

Ph.D. Dissertation

# Automatic Synthesis and Optimization of Chip Multiprocessors

Nikita Nikitin

Advisor: Prof. Jordi Cortadella

February 2013

# Abstract

The microprocessor technology has experienced an enormous growth during the last decades. Rapid downscale of the CMOS technology has led to higher operating frequencies and performance densities, facing the fundamental issue of power dissipation. Chip Multiprocessors (CMPs) have become the latest paradigm to improve the power-performance efficiency of computing systems by exploiting the parallelism inherent in applications. Industrial and prototype implementations have already demonstrated the benefits achieved by CMPs with hundreds of cores.

CMP architects are challenged to take many complex design decisions. Only a few of them are:

- What should be the ratio between the core and cache areas on a chip?
- Which core architectures to select?
- How many cache levels should the memory subsystem have?
- Which interconnect topologies provide efficient on-chip communication?

These and many other aspects create a complex multidimensional space for *architectural exploration*. Design Automation tools become essential to make the architectural exploration feasible under the hard time-to-market constraints. The exploration methods have to be efficient and scalable to handle future generation on-chip architectures with hundreds or thousands of cores.

Furthermore, once a CMP has been fabricated, the need for efficient deployment of the many-core processor arises. Intelligent techniques for task mapping and scheduling onto CMPs are necessary to guarantee the full usage of the benefits brought by the many-core technology. These techniques have to consider the peculiarities of the modern architectures, such as availability of enhanced power saving techniques and presence of complex memory hierarchies.

This thesis has several objectives. The first objective is to elaborate the methods for efficient analytical modeling and architectural design space exploration of CMPs. The efficiency is achieved by using analytical models instead of simulation, and replacing the exhaustive exploration with an intelligent search strategy. Additionally, these methods incorporate high-level models for physical planning. The related contributions are described in Chapters 3, 4 and 5 of the document.

The second objective of this work is to propose a scalable task mapping algorithm onto general-purpose CMPs with power management techniques, for efficient deployment of many-core systems. This contribution is explained in Chapter 6 of this document.

Finally, the third objective of this thesis is to address the issues of the on-chip interconnect design and exploration, by developing a model for simultaneous topology customization and deadlock-free routing in Networks-on-Chip. The developed methodology can be applied to various classes of the on-chip systems, ranging from general-purpose chip multiprocessors to application-specific solutions. Chapter 7 describes the proposed model.

The presented methods have been thoroughly tested experimentally and the results are described in this dissertation. At the end of the document several possible directions for the future research are proposed.

# Acknowledgments

First of all, I would like to express gratitude to my Ph.D. advisor Prof. Jordi Cortadella for this unique opportunity to work on the thesis in the wonderful city of Barcelona, within a team of talented, open-minded and especially friendly colleagues. These have been great years of experience and unrivaled memories!

A special, sincere and warm Thanks to my family. To my Mom and Dad, grandmas and aunt who were always there to *listen*, share and support my endeavors. To my sister Lialya, eager to exchange the thoughts and feelings, good news and useful lessons learned. To Kate who changed my world *with* a blink of an eye. I love you all.

I would like to acknowledge and thank Prof. Valery Perekatov who noticed my abilities in research and convinced me this would be a worth path to follow. I am also very thankful to Prof. Alexander Marchenko who assisted me in gaining first industrial experience, which essentially contributed to my skills.

Another very special gratitude is to Dima Bufistov, Gladys Utrera and Josep Carmona, who became my friends almost as soon as I landed in Barcelona for the first time, guided me though the very first steps in Spain and kept assisting all these years. And my gratitude to Marc Galceran who has been an excellent guide during my first US trip. Thank you all so much!

A big thanks for sharing the amazing time together and helping with great ideas to all colleagues from UPC and Intel Corporation, especially to Javier de San Pedro, Cesc Guim, Mike Kishinevsky, Umit Ogras, Jordi Petit and Sat Chatterjee.

This time would not had been so exciting without all of you guys, who I met in the office S108 of Edifici Ω. An astonishing team of the office mates with a great sense of humor and excellent problem solving capabilities (I'll never forget our puzzling activities at the whiteboard) is who I was awarded with for the thesis years.

Finally, this page would be incomplete without acknowledging the effort of all people at UPC who make the life of (international) students easier, and in particular, Mercè Juan Badia, Maria Serna and Anna Fàbregas. Thank you!

This work has been supported by a scholarship from Spanish Ministry of Science and Innovation (FPI grant BES-2008-004612), a gift from Intel Corporation and research project FORMALISM (CICYT TIN2007-66523).

# Contents

# Chapter 1

# Introduction

The performance of microprocessors has improved drastically in the last few decades. For many years, the increase in operating frequencies, driven by the rapid downscale of the CMOS technology, has served the primary source for performance improvement of computing systems.

Several important architectural decisions have been introduced, which now represent widely adopted paradigms for microprocessor design [64]. Instruction *pipelining* is a technique used to increase the system throughput and fully exploit the benefits of the high clock rates. *Superscalar* processors improve single-threaded performance by addressing the *instruction-level parallelism* (ILP) inherent in applications. ILP refers to a group of instructions, for which the result of every instruction from the group does not depend on the result of the other instructions from the same group. Hence, superscalar architectures have multiple execution units to process such groups of instructions simultaneously. *Out-of-order* execution extends superscalarity for ILP-driven optimization. It offers on-the-fly reordering of instructions to prioritize execution of those, for which the input data has already been obtained.

The constantly increasing gap between the performance of the processing and memory units, known as the *memory wall problem* [136] imposed significant limitations on the overall performance of Von Neumann architectures. Luckily, the observation of the *spatial* and *temporal* locality of memory accesses made it possible to reduce the severity of this problem by incorporating fast, although low-capacity on-chip caches. As the manufacturing process advanced, the growing transistor budget was further used to alleviate this problem by incorporating larger caches, effectively hiding the memory access latency. However, as the cache latency increased with its size, the growth in the cache size was penalized. This led to the *hierarchical organization of memory subsystem* with several levels of on-chip cache, trading-off the hit-ratio and the access latency, and exploiting the locality of memory references.

The increase in transistor densities faced various issues in the system-level design. As the components of integrated circuits continued to shrink, the inter-component (a.k.a. global) interconnects became the main performance bottlenecks. Global wires, implementing the off-chip communication between components, introduced delays which were significantly greater than the clock period. While pipelining techniques helped to improve the communication throughput, the point-to-point latency of the off-chip communication remained an issue. The decrease in the transistor and wire sizes enabled the integration of several system components (such as processors, controllers and memories) in a single chip, leading to the concept of *System-on-Chip* (SoC). The SoC paradigm eliminated costly global wires by transferring the inter-component communication to the on-chip level.

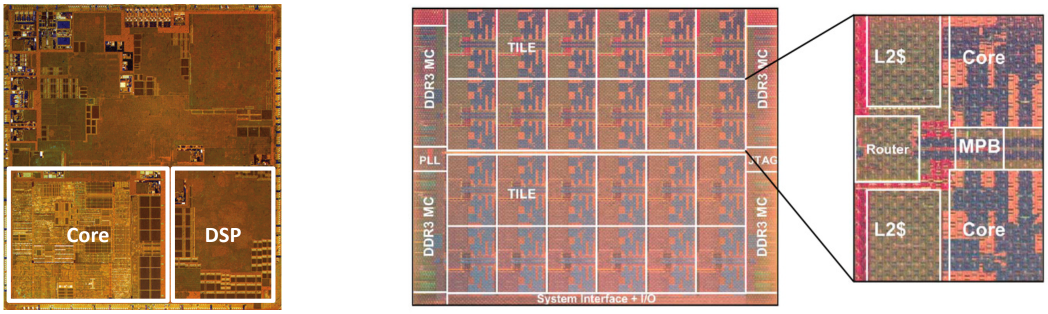
Another fundamental issue caused by the increase in CMOS operating frequencies and transistor densities is *power dissipation* [131]. Both factors contributed to the rapid growth of power density of the integrated circuits, imposing heat dissipation as the main concern for system-level design.

*Chip Multiprocessors* (CMPs), which represent general-purpose SoC implementations, became the latest paradigm to raise the power-performance efficiency of computing systems by exploiting *thread-level parallelism* (TLP) [58]. Instantiation of several simple but power-efficient cores on die enabled simultaneous multithreading, as well as high data exchange rates between cores.

One of the examples of commercial CMP implementation is the Texas Instruments OMAP3530 mobile processor. The layout of this chip is shown in Figure 1.1(a). This multiprocessor is also an example of heterogeneous CMP, incorporating a general-purpose core with graphic accelerator and a digital signal processor (DSP). Heterogeneity contributes to optimize performance and power of the system, as different types of computing tasks can be assigned to different types of cores, which execute these tasks with the highest efficiency (e.g. DSP-related tasks will be typically assigned to DSP cores).

Prototype and industrial CMP implementations have demonstrated computing benefits obtained by CMPs with tens and hundreds of homogeneous cores [73, 118, 119, 74]. The Intel Single-chip Cloud Computer (SCC) processor is one of such examples [73]. The layout of this chip is shown in Figure 1.1(b). This CMP contains 48 processing cores of the Intel Pentium family. The chip is organized into 24 dual-core tiles, each having two L2 cache modules and an on-chip router. A 6x4 mesh-based on-chip interconnect network provides communication between the tiles as well as access to DDR3 memory controllers and I/O interface.

The example of SCC chip emphasizes the importance of the backbone component of CMP: the *on-chip interconnect*. Indeed, for a system to perform efficiently, communication between the cores, memory and input/output subsystems has to be balanced, in order to assure that none of the components is overloaded, while



(a) TI OMAP3530 processor [8]

(b) Intel 48-core SCC chip [73]

Figure 1.1: Examples of heterogeneous and homogeneous CMP implementations.

the others remain underutilized. Actually, the interconnect critically determines the major parameters of the developed system, including area, performance and power consumption. The concept of *Network-on-Chip* (NoC) was proposed about a decade ago and is now established as a leading methodology for the design of high-performance low-power interconnects for many-core systems [41, 92, 58]. In the following sections, the benefits and design issues of the NoC technology will be addressed in more detail.

While CMPs represent one extreme class of SoCs, which are designed to support a variety of applications, another extreme is a broad class of fully customized *Application-Specific Integrated Circuits* (ASICs). In contrast to the general-purpose processors, ASICs are developed for executing a particular application (or a group of applications). This fact allows the design-time customization of an SoC for the requirements of the target application and brings additional degree of speed-up and resource savings. However, custom designs also require significant engineering effort, trading-off the development and fabrication costs.

Figure 1.2 shows an example of an ASIC, found inside the Texas Instruments Digital Radio Mondiale Solution [9]. The SoC diagram (on the left) depicts a block diagram of the chip, with some functionalities being audio decoding, compression and user interface management. The use of a general-purpose core instead of a custom ASIC for this solution would consume area and power resources unnecessarily, most likely requiring an additional cooling module. This SoC represents a typical example about the cost of engineering an ASIC being amortized by an improvement in performance and decrease of the fabrication cost.

The main objective of this thesis is the *design of methods and models for efficient synthesis and optimization of on-chip systems*. While the majority of contributions in this work focus on general-purpose CMPs, they can also be applied (or extended)

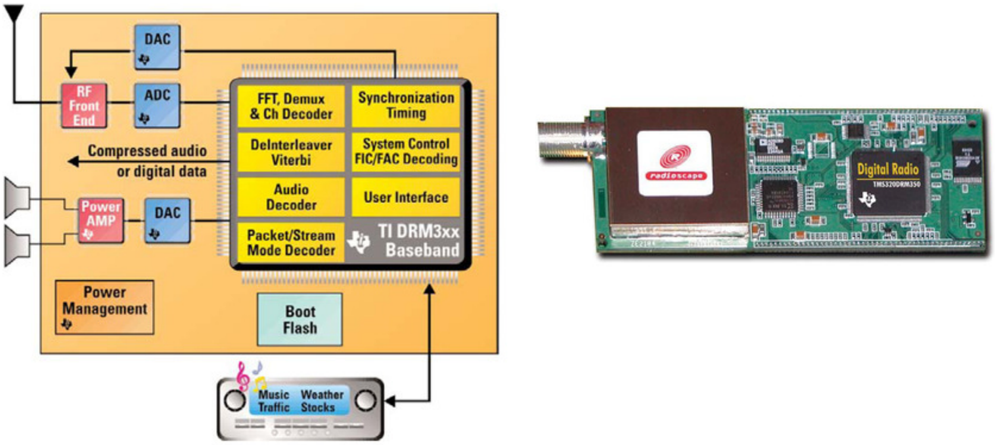


Figure 1.2: An example of ASIC SoC implementation: Texas Instruments TMS320DRM300/350 Digital Radio Mondiale Solution [9]. Block diagram of the system (on the left) and hardware implementation (on the right).

to the ASIC domain. Synthesis of networks-on-chip, as a backbone of CMPs, is of particular interest in this work. Next section provides a deeper look into the CMP organization and presents an overview of the main design challenges for CMPs and NoCs.

## 1.1 Design challenges for CMPs and NoCs

This section starts by explaining a generic organization of many-core CMPs. *Tiled CMPs* are an effective approach to architect general-purpose processors under the intense time-to-market pressure [87, 15]. The replication of tiles provides a rapid way of floorplanning many computing units in one chip and communicating them with scalable interconnect networks. Figure 1.3(a) shows an example of a CMP with 16 tiles, each one including a computing core (C) with private *L1* cache, a larger on-chip cache (L2), and a router (R) that communicates with the on-chip interconnection network (a mesh). Four memory controllers (MC) provide access to the off-chip memory. Intel SCC chip (Fig. 1.1(b)) is one of the tiled CMP implementations.

To exploit the locality of memory references, hierarchical interconnects have been proposed [43, 15]. Several cores can be grouped into one cluster to share the on-chip cache, accessible through a local interconnect (e.g., bus, crossbar, ring, etc). Hierarchy increases the intra-cluster hit-ratio and reduces the traffic in the top-level interconnect. Figure 1.3(b) shows an implementation of a CMP with 4 clusters. Each cluster has two cores with private caches, a shared cache (L3), a local interconnect (IC), a router and a network interface (NI).

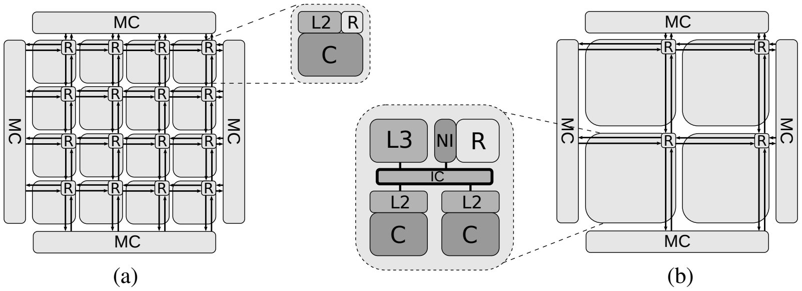


Figure 1.3: CMP layouts: (a) flat, (b) hierarchical.

Tile replication speeds up the engineering process by shifting the focus to the system-level design. Nevertheless, the design cycle remains a complex process, involving many stages, and only realizable by means of divide-and-conquer strategies. Furthermore, different models are employed at every stage, representing the tradeoffs between the speed and the accuracy of modeling. For example, full-system simulation, which is indispensable for validating the chip before fabrication, would not be feasible for design exploration, which requires very fast estimators. As a result, a mismatch in estimating system parameters during the different stages may happen, so that the design stages have to be iterated until the desired constraints are met.

Below several issues are enumerated that are among the most challenging in the system-level design of CMPs:

- *Design space exploration* is the earliest stage in the design process. Given the vast space of design parameters and, therefore, possible configurations, the objective of this stage is to pick out a promising architecture, which will determine the system-level structure of the future chip. The models used at this stage have to provide very fast estimation of the architectural metrics, even at the expense of loss in precision.

- *Physical planning* issues are tightly coupled with those of the exploration problem. Some architectures, which exploration marks out as delivering the highest performance, may reveal floorplanning or routability problems after thorough modeling. Physical planning aims at performing and verifying the chip floorplan and routing.

- *Quality of service (QoS) analysis and validation* aim at the design of models for system performance. Both, analytical models and cycle-accurate simulation tools are indispensable to complete the design process. Fast analytical models are crucial to make the exploration of huge design spaces tractable, while cycle-accurate simulation is required at the late steps of the design process for QoS validation.

- *System power management* focuses on the development of power-saving techniques. Widely applied mechanisms include powering-off unused resources, such as cores and caches, as well as *dynamic voltage and frequency scaling* (DVFS) for power saving, without substantial performance penalties. The need for floorplanning the voltage regulators to support multiple voltage-frequency islands makes this task closely coupled with physical planning.

- *Verification of functional correctness* is another important objective aimed at design validation. Some of the issues include verification of deadlock- and livelock-freedom of the routing algorithms and cache coherency protocols, or proof of memory consistency.

The problems mentioned above focus on the efficient *design* of CMPs. Another challenging topic is related to the efficient *usage* of multiprocessors after fabrication. This topic refers to the problems of *application mapping or scheduling* onto many-core systems. The trends in system design demand novel methods for application mapping, which take into consideration the peculiarities of future CMPs, such as the advanced power management techniques.

### Interconnect design for SoCs

One of the major problems for CMP and ASIC SoC engineering is the design of the on-chip interconnect. The traffic and latency of on-chip communication considerably affect the overall system performance and resource requirements.

Network-on-chip realization of packet-based communication offers the benefits of both point-to-point and bus architectures: multiple routes for packets make parallel communication in a network feasible, while link sharing saves the interconnection resources. This fact promotes NoCs as the outperforming solution in terms of scalability. Other benefits of the on-chip networking are:

- *Quality-of-Service guarantees.* The QoS estimation techniques can be reused or adapted from macro-networking studies, providing the on-chip interconnection with better performance predictability.

- *Fault-tolerance and reliability.* Application of the advanced fault-tolerance and error-recovery algorithms, including the approaches known from generic networking, allows handling reliability problems in complex multiprocessing systems.

- *Tolerance to design variability.* One of the most serious problems that technology scaling brings is the increasing variability of design parameters. *Globally* *asynchronous locally synchronous* (GALS) communication is one of the examples for coping with variability issues, which is facilitated by the modularity of NoCs.

The modern SoCs and NoCs require a thorough design process that involves a variety of problems: topology selection and mapping, physical planning, routing and switching schemes, and other optimization tasks. The large number of options and constraints makes it impossible to fully explore the solution space. On the other hand, dividing the design problem into smaller subproblems and doing a myopic optimization for each one of them may result in largely suboptimal solutions. Therefore, efficient techniques are required to combine the objectives of several design problems and intelligently explore the solution space for on-chip systems.

## 1.2 Motivation and contributions

The techniques developed in this thesis contribute to solve the problems of efficient design and usage of CMPs and NoCs. More precisely, the **objectives of the work** can be summarized as follows:

- *Elaborate methods for efficient analytical modeling and design space exploration of CMPs, incorporating high-level models for physical planning* (Chapters 3, 4 and 5).

- *Propose a scalable application mapping algorithm onto general-purpose CMPs with power management techniques, for efficient deployment of many-core systems* (Chapter 6).

- *Address the issues of the on-chip interconnect design and exploration, by developing a model for simultaneous NoC topology customization and deadlock-free routing* (Chapter 7).

In the following, each of the problems is considered in more detail.

### Analytical modeling and design space exploration

Given the vast space of design parameters, CMP designers are faced with the complex problem of selecting the best architecture subject to a set of constraints. The problem of design space exploration can be considered as the problem of efficient usage of chip resources, such as area and power. One of the possible formulation of the problem is maximization of the system throughput, subject to the constraints on the total chip area and power.

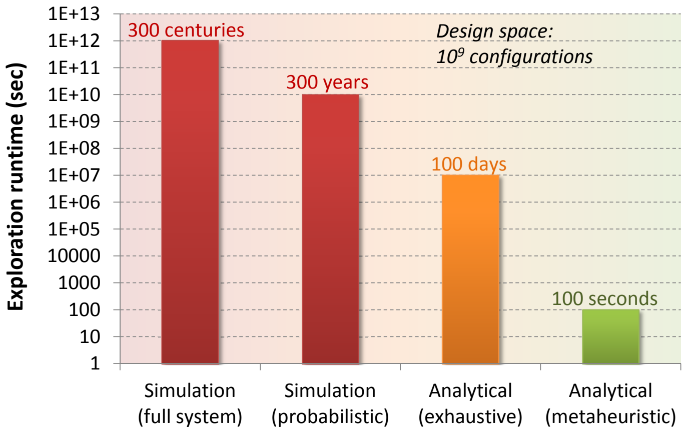


Figure 1.4: Comparison of runtimes for various exploration strategies.

Many design options must be explored, such as the variety of core implementations, interconnect types, topologies, cache hierarchies and memory management policies. Moreover, the amount of configurations increases drastically as the technology advances, allowing more cores and memory to fit into the chip area.

The complexity of the search space makes an exhaustive simulation-driven exploration prohibitively expensive. One of the ways to handle this problem is to decrease the number of points to be considered. Exhaustive exploration is replaced by an intelligent search strategy leveraging the methods of machine learning [77], design of experiments [124] or analytical optimization [31].

Another option to shorten the exploration time is to reduce the cost of evaluating every design point. In this scenario, analytical modeling becomes an effective alternative to simulation for rapidly pruning the design space during early exploration and selecting a small set of promising configurations. Along this line, several analytical models for CMP exploration have been recently proposed (e.g., [107, 31]).

However, the use of analytical models alone is not enough for efficient exploration. Figure 1.4 gives an idea of the ratio of exploration run times, when a simulator or an analytical model is used to estimate the cost of architectures. For a search space comprised of one billion configurations, the exploration task is intractable even when a fast probabilistic simulator is used. Analytical modeling reduces run time to approximately 100 days, which is a feasible period assuming the design cycle of several years, though the timing cost of fixing an error committed at this stage remains very high. Implementation of the intelligent search strategies (e.g. metaheuristics) on top of the analytical model allows to further decrease the search time to the order of minutes, making the exploration tool a very efficient instrument for the designer.

This thesis addresses the techniques for analytical modeling of CMPs and the on-chip interconnects in Chapters 3 and 4. A method for intelligent design space exploration is proposed in Chapter 5.

### Application mapping for tiled CMPs

From a certain point of view, a tiled CMP is similar to a coarse-granularity programmable array (FPGA), with general-purpose logic, specialized units (memories, DSPs) and a distributed communication network for routing. The main difference is that CMPs are software-configurable, whereas FPGAs are hardware-configurable. In a similar way a design has to be transferred to FPGA, an application has to be mapped onto the target CMP.

One of the peculiarities of mapping onto tiled CMPs is the presence of *heterogeneous tiles*, preserving the regularity of the structure, but introducing several classes of processors [86, 14, 38]. Such systems may include some specialized processors (e.g., graphics, DSP) or different implementations of the same architecture (e.g., in-order/out-of-order, multi-threading) with varied power-performance tradeoffs. Figure 1.5 depicts a tiled CMP with three classes of tiles: general-purpose cores (C), cores with graphics units (G) and DSPs (D).

CMPs are designed to operate under a certain power budget that assures the performance and thermal properties of the system. One of the most effective ways to manage power is to floorplan various voltage islands and assign the best voltage and frequency for each core [82].

Unfortunately, voltage islands have a high design cost. Firstly, the floorplanning of the system is constrained by the design of the power delivery network and the location of the level shifters. Secondly, and more important, power management requires different voltage regulators for each power supply. Off-chip regulators need extra area on the PCB that may be unacceptable if the system has a large amount of power domains. On-chip regulators involve a significant area overhead and power consumption due to the large inductances and switching capacitors required to provide a stable supply voltage [79].

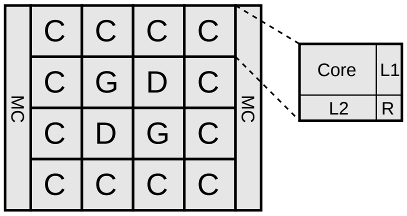


Figure 1.5: Tiled heterogeneous CMP architecture.

It is therefore realistic to consider that future CMPs will have many cores (hundreds) and voltage islands with several cores (e.g., 4 or 8). This fact imposes an additional constraint in the task mapping problem: even though some cores could possibly run at lower voltages and frequencies, sharing the island with other cores may prevent from taking advantage of this flexibility. Hence, it is convenient to allocate tasks in a way that cores within the same voltage islands can share similar voltage/frequency parameters.

The mapping problem for tiled CMPs has to assume that the chip has already been manufactured. Therefore, the voltage islands have been already floorplanned and the maximum bandwidth of the links between cores is also known a priori. Another peculiarity of CMP mapping (as opposed to SoCs), is the presence of traffic to the off-chip memory, as well as the limited bandwidth of the memory controllers (MCs). Finally, the methods proposed for task mapping must be scalable and suitable to handle systems with hundreds of cores.

Chapter 6 describes the contribution of this thesis in the field of application mapping onto tiled heterogeneous CMPs with predefined voltage islands.

### Network-on-Chip topology customization and routing

Consider an example of a system, which is specified by a set of processing elements ($PE$), routers and communication requirements between $PE$s. Assume that the underlying system topology has been selected and the assignment of the $PE$s to the routers has been performed. Some of the design problems one would like to solve are:

- Finding a subset of links satisfying the communication requirements of the system and minimizing the design cost.
- Defining the routing paths for each pair of communicating $PE$s that satisfy the performance requirements.
- Guaranteeing that the selected routes to communicate $PE$s are deadlock-free.

The first problem can be referred to as the *link allocation* problem. The other two problems are related to the efficient *route assignment* with *deadlock avoidance.* These problems have different optimization criteria. By solving one of them optimally and independently from the others, no acceptable solution might be found when solving the subsequent problems. It is therefore necessary to devise non-myopic strategies to explore the solution space in a way that all design constraints are met and the implementation cost is minimized.

Chapter 7 describes the last contribution of this thesis for the simultaneous interconnect topology customization by link allocation and routing.

## 1.3 Document organization

The dissertation is organized as follows:

- Chapter 2 gives an overview of the chip multiprocessing technology and summarizes related work in the field of CMP and NoC synthesis and optimization.
- Chapter 3 presents the first contribution of the thesis, which is an analytical model for a special class of networks-on-chip with constant-length data packets. This specific class of the interconnects finds application in CMP systems.
- Chapter 4 describes the second contribution of the thesis, representing an analytical modeling of the complete CMP. This section emphasizes the importance of modeling contention of the CMP interconnect and captures the cyclic dependency between the latency and traffic of memory requests.
- Chapter 5 proposes a metaheuristic-based intelligent search of the design space to eliminate the need for exhaustive architectural exploration for CMPs. This technique is the third contribution of the thesis.
- Chapter 6 addresses the issues of efficient usage of general-purpose CMPs after fabrication. A task mapping algorithm for tiled CMPs with multiple voltage islands for power management is proposed in this section, which is another contribution of this work.
- Chapter 7 represents the last contribution of the thesis, which is a mathematical model for simultaneous topology customization by link allocation and deadlock-free route assignment for NoCs.
- Chapter 8 concludes the work and outlines possible directions for future research.

# Chapter 2

# The Architecture of Chip Multiprocessors

This chapter provides a brief overview of the chip multiprocessor architecture, with the objective to understand the problem domain and help evaluating the contributions of this thesis. Last section of the chapter summarizes the work in the field, relevant to the problems considered in this dissertation.

When examining the organization of a chip multiprocessor, one may find certain similarities with the structure of a single-core processor. Basic components of the latter would include a processing unit (ALU), memory units (registers and L1-cache), a control unit, a datapath and an input/output subsystem. The components of a chip multiprocessor resemble most of those found in a single-core implementation:

- The *processing cores* are the computing units executing the instruction flow. These may be general-purpose or application-specific processors, offering various cost/performance trade-offs for a variety of workloads.

- The *memory subsystem* is a collection of memory units representing a trade-off between the memory capacity and access latency. On-chip components include the private and shared cache modules.

- The *on-chip interconnect* is the backbone of a chip multiprocessor, providing communication between the cores, memory units and input/output interface. The topic of interconnect design for efficient on-chip communication is of special interest for this work.

- The *input/output interface* is a set of controllers to interact with the off-chip components of the computing system. *Memory controllers* can be considered as a part of I/O interface, providing access to the off-chip memory. Another example is a Quick-Path Interconnect (QPI) [71].

For the problems addressed in this work, the cores, memories and on-chip interconnects represent the CMP components of major interest. In the following sections an overview of the design alternatives for each of these component types is given.

## 2.1 Processing cores

Cores are the main processing units executing the instruction flow. Modern CMP architectures include from several up to hundreds of processing units [73, 118, 119, 74]. The cores of a *homogeneous* CMP are identical, while *heterogeneous* architectures may include various combinations of core implementations for speeding-up particular applications. One of the popular examples of a heterogeneous CMP is the IBM Cell processor [38].

The first computing cores were implemented as sequential finite-state machines for loading, executing and writing the result of the instructions, as given by the original instruction flow. The processing of every instruction could take several cycles limiting the overall performance of the core, measured in *instructions per second*.

It was observed that in the given implementation certain modules of computing logic would *stall* while waiting for the other modules to complete their job: for example, the execution unit would wait for the instruction loading to bring the data in for processing. Instruction *pipelining* was proposed to introduce parallelism and increase the effective load of independent pipeline modules [22]. The flow of instruction processing is divided into several stages and every stage is executed within an individual clock cycle in a pipelined fashion. The stages of a typical pipeline include instruction *fetch*, *decode*, *execute* and *memory write*. Although this technique does not improve the latency of executing a single instruction, it raises the core throughput and enables higher operating frequencies.

It was further noticed that the instruction flow of many applications is inherently parallel, in the sense that there exist groups of instructions whose result does not depend on the other instructions in the group. This property is referred to as *instruction-level parallelism* (*ILP*) of the application. All instructions in every such group can be executed in parallel without affecting the result of the program. This observation led to the decision of supplying cores with multiple execution units, which could handle parallel instructions simultaneously, reducing the total execution time of the program. Cores with replicated execution units are referred to as *superscalar*.

Still, superscalarity enabled the execution of multiple instructions in parallel only when the instructions followed one another in the original instruction flow. In other words, superscalar architectures represented an example of *in-order* execution of the instruction flow. Performance was considerably penalized in cases when a long-latency memory request was issued, causing the whole pipeline to stall for a long number of cycles.

*Out-of-order* execution was proposed to extend the benefits of ILP by locally reordering the instruction flow, and giving priority to those instructions for which the input data had already been computed. An analysis of dependencies in the instruction flow is required to guarantee that the program result is not affected by the manipulations with the order.

While superscalar and out-of-order architectures aim at automating the search of ILP by hardware, *Very Long Instruction Word* (*VLIW*) [54] architectures delegate the responsibility of finding ILP to software, either at the compiler or programmer level. VLIW processors realize a set of "wide" instructions, each implementing several operations (e.g. a sum, an increment and a multiply). The task of defining the instruction patterns is deferred to software, which is shown to be convenient for certain application domains, where such operation patterns are common (e.g. DSP or numerical computations).

Finally, *simultaneous multi-threading* (*SMT*) [126] emerged to exploit the *thread-level parallelism* (*TLP*), as opposed to ILP. There are two primary sources of TLP. First, as in the case of VLIW architectures, TLP can be specified explicitly by the programmer within the software development process. Second, TLP often appears due to the fact that one core is commonly used to run several applications simultaneously. Hence, modern cores implement multi-threading support by replicating the thread-processing units.

An interesting observation regarding the efficiency of advances in single-core implementations is known as the *Pollack's rule* [29] [116]. It states that the core performance, in terms of *instructions per cycle*, $IPC_c$, only grows as a square root of the core area $A_c$:

$$IPC_c = \alpha \cdot \sqrt{A_c},$$

where $\alpha$ denotes the performance of a basic core chosen for comparison. Figure 2.1 plots an illustration of Pollack's rule from [42], constructed for more than a hundred of real processor implementations, using the data from Stanford CPU Database [4]. The performance and area of Intel 80386 processor are used as a baseline.

This rule has served as one of the motivations for chip multiprocessing, favoring instantiation of a multitude of simpler cores rather than several complex designs, for the purpose of throughput optimization [108]. However, it is worth noting that Pollack's rule only predicts a generic trend between performance and cost of single core architectures. Depending on the type of workload, the amount of ILP and TLP in the executed application, a CMP with more complex out-of-order cores may deliver more throughput than the one comprised of many simple core implementations.

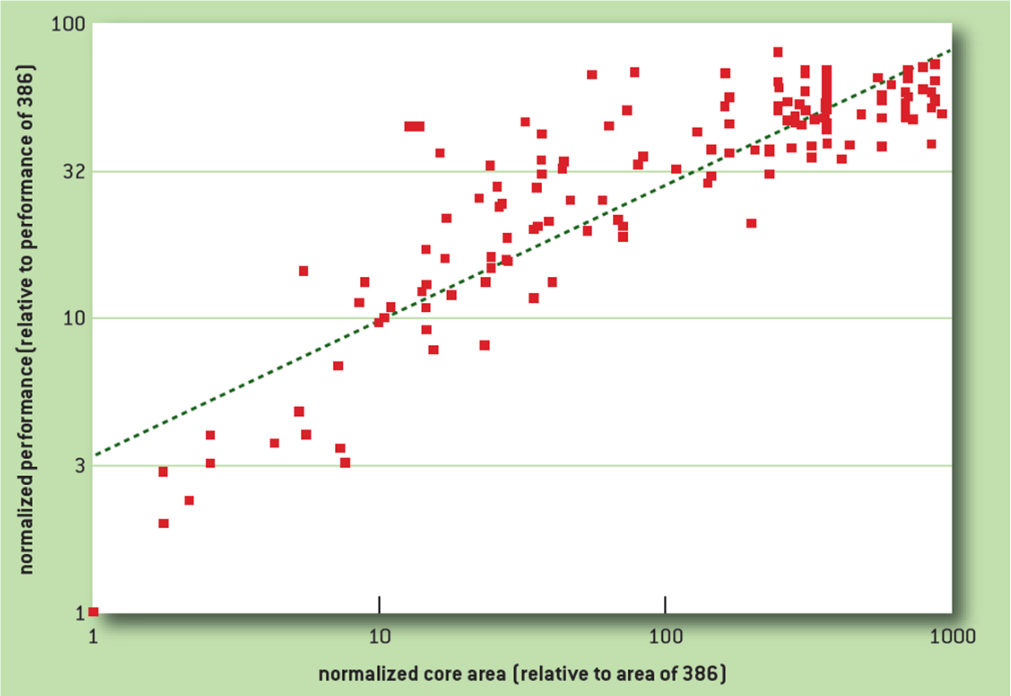


Figure 2.1: Pollack's rule for processing cores [42], [4].

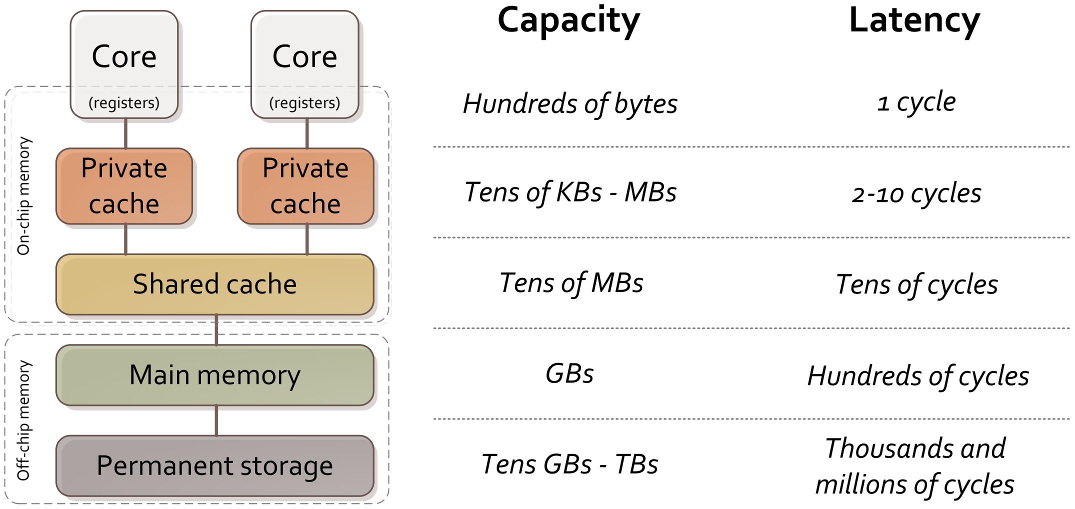
The problem of searching for the throughput-optimal CMP architecture, given a library of components and application workloads, is one of the major problems addressed by this thesis.

## 2.2 Memory subsystem

Processing cores execute *instructions* that operate on *data*. Both instructions and data are stored in *memory*, and hence its organization has a significant impact on the overall performance of the computing system.

Since its conception, dynamic RAM technology continuously evolved, with about 15% increase in performance and 2× increase in density every two years [29]. At the same time, the performance of processing cores advanced exponentially, leading to the constantly growing gap between the performance of computing and memory units, known as *the memory wall* [136].

The principle of *locality* of the memory references helped to alleviate the memory wall problem. It was observed that at least two major types of locality are inherent to the memory access patterns. *Temporal locality* refers to the fact that if a particular memory address is accessed at some moment in time, it is very likely that the same address will be requested *again* in the near future. *Spatial locality* observes also that



| Capacity | Latency |
|---|---|
| *Hundreds of bytes* | *1 cycle* |
| *Tens of KBs - MBs* | *2-10 cycles* |
| *Tens of MBs* | *Tens of cycles* |
| *GBs* | *Hundreds of cycles* |
| *Tens GBs - TBs* | *Thousands and millions of cycles* |

Figure 2.2: Hierarchical organization of memory subsystem.

if a particular memory address is accessed at some moment in time, it is very likely that the *nearby* addresses will be requested in the near future.

These observations made it possible to effectively reduce the average memory latency by organizing fast caches of moderate sizes, even when it is not feasible to supply fast and large memory which would store all data requested by applications. According to the principle of locality, the regions of memory which have a high probability of being accessed in the near future are loaded and kept in the caches for improving the average memory latency.

The introduction of caches led to *hierarchical* organizations of the memory subsystem, which includes several layers of memories of increasing sizes and latencies. Figure 2.2 gives an overview of the hierarchical organization. In addition to the core registers, which have a very fast access time, but a limited capacity of only hundreds of bytes, the on-chip memory subsystem is represented by several levels of cache. The fastest private first-level cache, *L1*, typically has the latency of several cycles and a capacity of tens of kilobytes. A larger private cache, *L2*, can be added, providing capacities of several MBs and latencies around ten cycles. A *shared* cache of a larger size, often called a *last-level cache, LLC* is included to extend the on-chip memory with a flexible storage, which can be redistributed among the cores according to the needs of the executed applications.

The off-chip memory subsystem includes the main dynamic RAM and the permanent storage. The capacities of the former have the order of GBs at the expense of

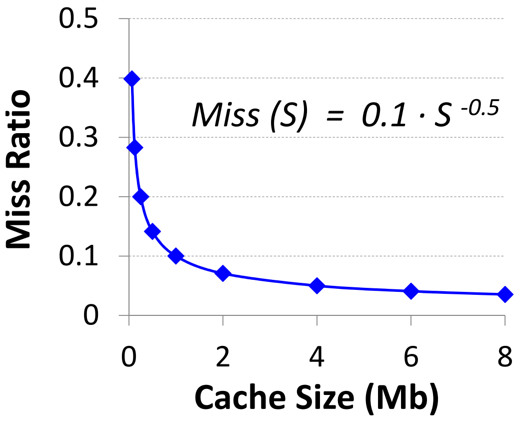


Figure 2.3: Power-law cache miss model.

an increase in latency of 10× or more. Dynamic RAM is backed up by a significantly slower, though non-volatile memory, which provides virtually infinite capacity.

One of the problems in the design space exploration of CMP systems is how to properly architect the memory subsystem, in terms of specifying the number of levels in the hierarchy, distributing cache capacity across the hierarchy levels and organizing cache sharing.

A possible way to characterize an application from the perspective of its memory requirements, is a model that defines the *miss ratio as a function of the cache size*. Miss ratio for a given cache size is defined as the fraction of memory accesses for which the data has not been found in the cache. Figure 2.3 plots an example of a miss-ratio curve, approximated with a power-law:

$$MR = MR_0 \cdot S^{-\alpha},$$

where $MR_0 = 0.1$ and $\alpha = 0.5$. Power-law was found to be a good analytical function approximating the cache miss behavior [63]. The miss-ratio dependency on cache size will be actively used in this work to represent the application model.

## 2.3 On-chip interconnects

On-chip interconnect is a backbone of a CMP, realizing data communication between the processing and memory units of a system.

This section overviews two basic on-chip interconnect architectures, crossbars and buses, summarizing their advantages and drawbacks and motivating the need for on-chip networking, which is introduced in the next section.

## Crossbars

The simplest and most intuitive way of organizing the communication between several components is by using point-to-point connections, such as a *crossbar*. This organization provides every pair of communicating components with a private link, resulting into a high degree of parallelism in data transmission.

Figure 2.4(a) shows one possible implementation of a crossbar connecting four components [1]. Generally, every receiver will have a single queue for incoming data arriving from all senders. Hence, an arbiter is required to resolve the cases of contention, i.e. when several inputs request the same output port. In the depicted implementation, arbitration signals are used as select signals for input multiplexers, assigning the winning inputs to every output.

In some special cases of high-performance crossbars a dedicated queue may exist for every sender-receiver pair. In this case crossbar resembles a collection of buffered point-to-point links performing independently, so arbitration between different inputs is not needed.

The main issue with crossbars is their limited scalability, since the amount of resources (mainly links) increases quadratically with the quantity of attached components. Although academic research has demonstrated that crossbar configurations with up to 128 components are possible [113], industrial implementations reveal a rather conservative view on crossbar scalability, with the maximum system size not exceeding several dozens of components.

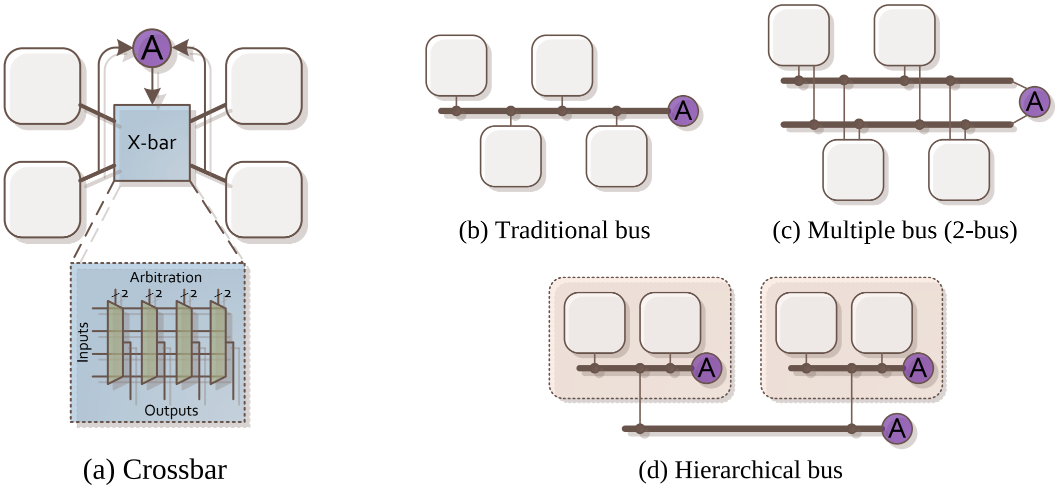
In application-specific systems, where the communication flows are known at design time, interconnect topologies can be customized to produce *partial crossbars*, which save resources by eliminating certain communication paths. Several methodologies for automatic ASIC-driven generation of partial crossbars have been proposed in the literature, e.g. [97], [94].

## Buses

The fact that communication between a pair of components normally does not occur continuously in every cycle results in crossbar underutilization. This observation led to the paradigm of *resource sharing* for on-chip communication, which is traditionally represented by the family of *bus* topologies.

Buses can be seen as the opposite of a crossbar: while the latter implements a dedicated link for commuting every pair of components, bus provides a single shared link for any communication, making this architecture very resource-efficient. Figure 2.4(b) shows a single bus connecting four components, and its arbiter. The

[1] Here and further in this chapter, components are illustrated with rounded-corner squares, while arbiters are represented by circles labeled with 'A' inside.



(a) Crossbar

(b) Traditional bus

(c) Multiple bus (2-bus)

(d) Hierarchical bus

Figure 2.4: Interconnect solutions for four components: (a) crossbar, (b)-(d) buses.

sender issues a request for bus access to the arbiter. Once granted the ownership, the sender establishes communication through the bus with the receiver and data is transmitted. Upon the end of data transmission, the bus ownership is returned to the arbiter.

Note that this communication policy is intrinsically sequential in the sense that the next sender can not start transmitting data before the previous sender has finished the transmission. This limitation imposes a serious bandwidth penalty for the case when two or more low-latency communications need to be realized simultaneously.

Another problem of bus scalability is the growing latency due to the long links required to connect all components. Bus pipelining can alleviate the problem for transferring large amounts of data in a single communication, but point-to-point latency remains an issue.

Several extensions of a single bus have been proposed to tackle the mentioned problems. The *multiple bus* topology is an extension of the traditional single bus architecture, implementing several transmission lines and enabling certain degree of parallelism in bus transactions. Figure 2.4(c) presents a 2-bus organization with two links, which allow up to two simultaneous data transmissions. Assuming that two different senders request communication with two different receivers at the same time, the arbiter may assign one of the available links for every sender-receiver pair. As a result, both communications occur simultaneously without affecting each other.

Another bus extension is the *hierarchical bus* topology, shown in Figure 2.4(d). This architecture is beneficial for the communication patterns where certain degree of locality of the on-chip requests is observed. Components that require high inter-communication bandwidth are organized in clusters and communicate via local buses, supplied with dedicated arbiters. In case an inter-cluster communication is needed, it is realized via the global bus. Note that local communication is performed through shorter (hence, faster) local buses, effectively decreasing the average point-to-point communication latency. Hierarchical architectures can be extended to an arbitrary number of levels straightforwardly.

It is clear that the performance improvement obtained by adding multiple bus links or hierarchy levels comes at the cost of additional resources, such as area and power, and represents one of the most important trade-offs in bus design. Other common trade-offs include synchronous vs. asynchronous bus operation, separation of signal types (address, control, data) over different physical links vs. multiplexing of signals on the same physical link.

A number of bus standards and communication protocols have been adopted and extensively used. Some of the most popular standards include the Advanced Microcontroller Bus Architecture (AMBA) [1], and STBus [7].

## 2.4 Networks-on-Chip

Inspired by the idea of macro-networks, the *Network-on-Chip* (NoC) concept was introduced to tackle the scalability issues of the on-chip communication [92], [41]. This notion changes the interconnection paradigm from routing wires to sending data packets across the network [41]. NoCs represent a trade-off between high-performance point-to-point connections and resource-efficient buses, since they offer both, data transfer parallelism and link sharing between the flows.

The network-on-chip abstraction is depicted in Figure 2.5. A network is composed of a set of *routers* (R), and physical *links* between them. The exact topology may

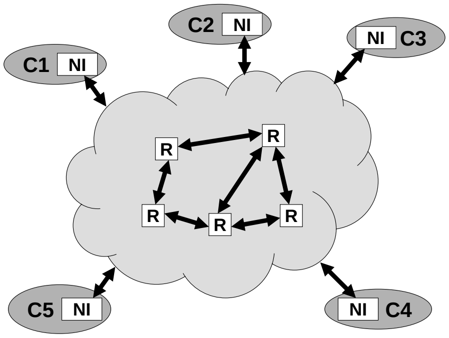


Figure 2.5: Network-on-Chip abstraction.

vary broadly, from regular meshes and rings in general-purpose CMPs to highly irregular solutions in ASICs. Cores (C1-C5) are connected to the network via the *network interfaces* (NI). The role of the NI is to convert the data generated by the cores into the packets that can be sent through the network, and vice versa, decode the packets arriving at the destination nodes. The term *node* is used to denote a generic traffic injector or consumer, connected to a network, which can be a processing core, a cache memory or any other type of on-chip device, such as a memory controller.

Typical objectives in NoC design are the low-latency, high-bandwidth communication and the easiness of network scalability. Although similar to those of the traditional macro-networks, the requirements for on-chip networks include several specific properties:

- *Low area* is an essential requirement to improve the chip yield, manufacturability and reduce leakage power of the interconnect.
- *Higher bandwidth*, compared to the traditional networks, is needed to incorporate additional transactions, such as the traffic of cache-coherency protocols and accesses to the shared memory.
- *Fault-tolerance* and *reconfiguration* abilities are important for guaranteeing the interconnect correctness and performance in the presence of faulty components.
- *Variability handling* becomes a necessary property for designing robust systems, given the increasing uncertainty of the CMOS devices implemented with deep-submicron technologies.

In the rest of this section, the main design issues for NoCs are outlined, such as the selection of network *topology*, *switching* technique, *flow control* mechanism and *routing* algorithm. At the end of the section an overview of the NoC router internals is provided.

## Topology

The topology of the network-on-chip is an essential design parameter which critically affects the performance and cost of the on-chip communication. Topology primarily determines the communication bandwidth and latency by defining the number of on-chip routers and links, their relative position and connectivity.

The classification of NoC topologies can be done according to different criteria, among which the most interesting in our context are the following:

- *Direct* vs. *indirect* topologies. Direct topologies assure that every router has a connection to at least one node, while indirect topologies allow routers which only have connections to other routers.

- *Flat* vs. *hierarchical* topologies. Hierarchy typically resembles tree-like structures with clearly marked clusters of local communication. As in case of hierarchical bus, these topologies are effectively embedded in systems with significant locality of the on-chip communication.

- *Regular* vs. *irregular* topologies. Regularity can be defined as a presence of a certain pattern or symmetry when connecting links to routers, as well as in distribution of the link lengths and hop counts. Regular topologies are better fit for general-purpose designs, while irregular topologies allow to benefit from design-time specialization.

Typical trade-offs in topology design can be analyzed by enumerating the most common topological properties:

- *Bisection bandwidth* is defined as the minimal bandwidth over all possible partitionings of the topology into two equal halves. This property is often used to predict the performance of the network analytically.

- Network *diameter* is the maximum number of hops between any pair of nodes. Diameter minimization is an important objective for improving the network performance.

- *Link count* and *router count* define the total number of links and routers respectively, and are used to approximate the resource cost of the network.

- *Router degree* represents the number of input and output ports of the router and is closely related to the delay propagation through the router and the area and power consumed by the router.

Finding the balanced choice of the aforementioned parameters for the given communication pattern and performance constraints, so as to minimize the resource cost, constitutes the topology selection problem for network-on-chip. The problems of topology exploration and application-specific customization are among the important problems addressed by this thesis. For this reason, a broader glance at the topological choices relevant for this work is next presented.

### Meshes

A 2-dimensional mesh, depicted in Fig. 2.6(a), is the most widely adopted topology for network-on-chip, both in research and industry. Its popularity is determined by several factors, such as regularity, good physical properties and scalability. The layout of regular 2D mesh facilitates the chip floorplanning and is especially valuable for *tiled* architectures, which are popular for rapid prototyping of many-core systems. Indeed, to extend the mesh to an arbitrary number of nodes one simply needs to add rows or columns to the mesh without the need to redesign the basic network components, such as routers and links.

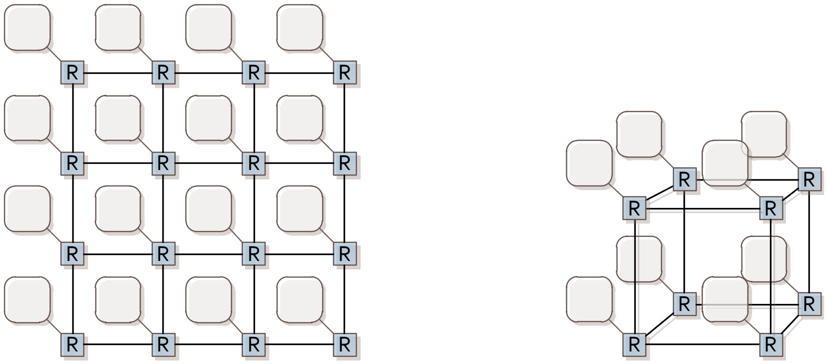
The family of mesh topologies is known as *k-ary n-meshes*, where $k$ defines the number of nodes per dimension and $n$ is the number of dimensions. Meshes with dimensions of three and higher have a limited applicability due to issues of physical implementation. Figure 2.6(b) sketches a 3D mesh (cube), which despite its nice theoretical properties, has to be planarized when implemented in traditional CMOS technology. Planarization leads to an increase of delay in certain links, so the final performance of this topology will be strongly affected by physical implementation. It is worth noting that with the emergence of three-dimensional semiconductor technologies (e.g. stacked memories), 3D topologies become more popular. However, the interest in meshes of higher dimensions is still limited to the research community.

Although meshes can adopt any arbitrary number of components without significant design effort, they are considered to be poorly scalable: the network diameter grows linearly with the mesh dimensions and its bisection bandwidth scales slower than the communication requirements, as the number of components increases.

*Concentrated mesh* or *C-mesh* was proposed to decrease the average latency in traditional meshes (Fig. 2.7(a)). The reduction in latency is obtained through the increase of router degree and concentration of larger number of nodes at one router. Although this organization incurs a penalty on network throughput, it becomes beneficial when a certain degree of traffic locality is observed.

Another family of solutions that improve network performance in presence of local traffic is *hierarchical* mesh-based topologies. An example of two-level mesh-bus architecture is shown in Fig. 2.7(b). Similar to a hierarchical bus, this topology implements individual low-latency bus interconnects for local communication in clusters. Additionally, a high-throughput global mesh is responsible for inter-cluster communication. The benefit of hierarchical approach is the separation of local and global traffic for eliminating their interference, although at the cost of additional resource consumption.

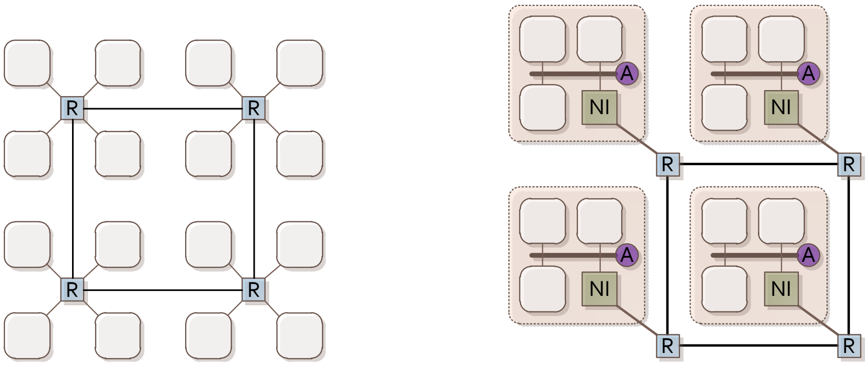
Hierarchical meshes are of high interest for this work, since they are considered as a promising architecture for low-power high-performance interconnects of CMPs with hundreds or even thousands of computing units.



(a) 2D mesh (4-ary 2-mesh)

(b) 3D cube (2-ary 3-mesh)

Figure 2.6: K-ary n-mesh topologies.



(a) Concentrated mesh

(b) Hierarchical mesh-bus NoC

Figure 2.7: Extensions of 2D meshes.

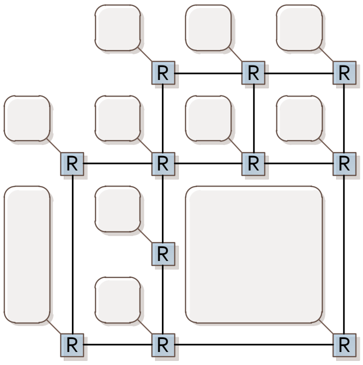


Figure 2.8: Irregular mesh topology.

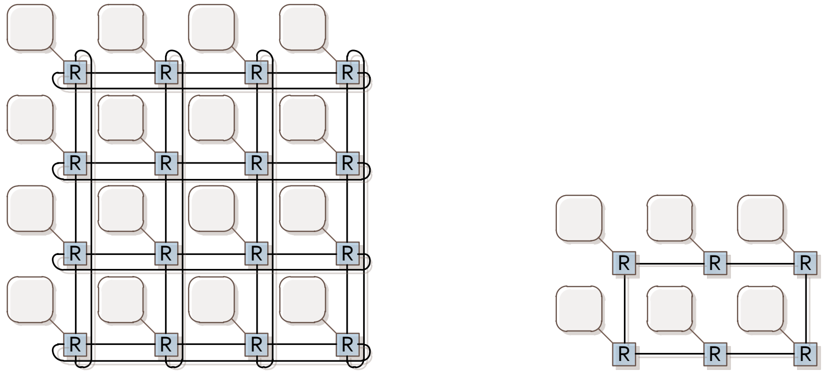
Finally, *irregular meshes* represent a specific class of mesh-based topologies, which are especially popular among the ASIC community, where heterogeneous nodes (i.e. cores, memories) appear frequently. Heterogeneity of node sizes leads to irregular floorplans, which as Figure 2.8 proposes, can be arranged in a grid of a fine granularity and effectively connected by a mesh-based structure. The obtained mesh may miss some routers, as compared to a regular implementation; the links may have different lengths and router degrees may vary - explaining the "irregular mesh" terminology. The fact that the interconnect topology is grid-based speeds-up the design time considerably, making irregular meshes popular for application-specific systems. The aspects related to mesh topology customization are considered later in this thesis.

### Tori and rings

A two-dimensional torus is another widely used solution for NoCs and represents a mesh with added wraparound links, as in Fig. 2.9(a). The torus family is denoted as *k-ary n-cubes*, with the same notation for $k$ and $n$ as in the case of meshes.

Tori of dimension two and higher are exposed to the same planarization problems as high-dimension meshes. Depending on the implementation, wraparound links may exhibit delays similar to traversing the whole chip in one dimension. For certain chip sizes this results into a severe mismatch between the predicted link delay and its silicon realization, causing a noticeable performance penalty.

One-dimensional tori, also known as *rings* (Fig. 2.9(b)) represent another example of broadly applied topologies, due to several convenient properties. First, rings are regular symmetric topologies which are easily floorplanned and implemented in silicon. One can choose among uni-directional and bi-directional ring implementations to explore typical cost/performance trade-off. Another reason for using rings



(a) 2D torus (4-ary 2-cube)

(b) Ring (6-ary 1-cube)

Figure 2.9: K-ary n-cube topologies.

is the low router degree, consisting of only three ports for uni-directional or four ports for bi-directional rings, thus simplifying router design and cost significantly.

The drawback of rings is the low bisection bandwidth, which is however compensated by low area and power requirements, making rings a good choice for hierarchical interconnect architectures.

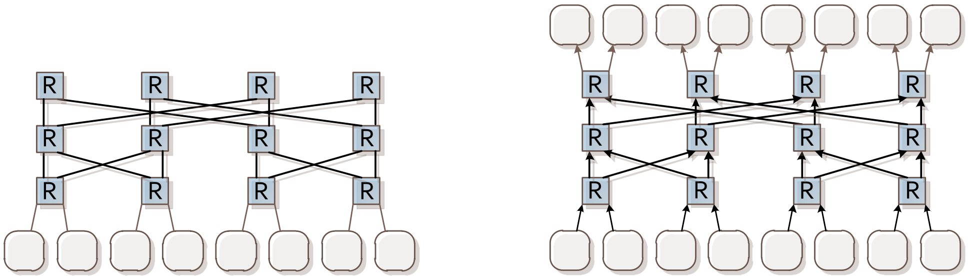
### Fat-trees and butterflies

As opposed to meshes and tori, which are examples of direct topologies, fat-trees represent indirect topologies, implemented as multistage interconnect networks (MINs), as shown in Fig. 2.10(a). The data has to traverse several stages of intermediate routers in order to reach destination.

The main benefit of this topology is the high bisection bandwidth, which scales linearly with the number of network nodes. Nevertheless, the cost of increased bandwidth is paid with the augmenting number of routers. Even more important, the problem of physical implementation of the MIN-based topologies in silicon appears due to the complex wiring between routers. Fat-tree topologies continue raising interest in the domain of optical interconnects. Recent techniques aimed at improving the physical planning of NoCs based on fat-trees [138].

Butterfly networks are also based on MINs and similar in their physical structure to fat-trees. However, these are uni-directional interconnects, with sender and receiver nodes situated along the opposite sides of the network (Fig. 2.10(b)). The routing path between any sender-receiver pair in butterfly network has a predetermined number of stages. The drawback of this topology is the existence of only one path between a communicating pair of nodes. This observation requires additional mechanisms for implementing fault-tolerance techniques in the presence of faulty links or routers.

A similar notation is used for *k-ary n-trees* and *k-ary n-flies*, where $k$ denotes the number of nodes connected to each router and $n$ is the number of MIN stages.



(a) Fat-tree (2-ary 3-tree)

(b) Butterfly (2-ary 3-fly)

Figure 2.10: Trees and butterflies.

## Switching

The switching technique determines the policy for allocating the network resources required for data transmission through the interconnect. Several switching schemes are considered in this section.

The technique of reserving a dedicated channel between sender and receiver is referred to as *circuit switching*. Prior to any data transmission, a path has to be allocated between the two nodes, guaranteeing that all shared resources have been properly allocated and the data can be commuted avoiding interference with other flows. Once the data transmission has been completed, the path is deallocated and its resources are released to favor communication of other flows.

The major benefit of circuit switching is the absence of contention between different flows. This property eliminates the need for buffering resources, effectively reducing the NoC area and power requirements. On the other hand, the need for path allocation and deallocation not only introduces a performance penalty, but also reduces the amount of parallelism in communication. Indeed, the requirement to establish and maintain a path between different parts of the network reserves resources unnecessarily, blocking the data transmission of the other flows with a high probability.

*Packet switching* is an alternative approach which does not require path reservation, since the data packets determine their paths independently as they propagate through the network. Sending data in packets allows communication at a finer level of granularity. The downside of the technique is a chance for contention when several packets compete for the same shared resource, such as a network link. This implies the allocation of buffer resources across the network so that the packets can be temporarily stored and then forwarded, after contention has been resolved.

The basic implementation of packet switching is hence referred to as *store-and-forward*. Every router input is provided with a buffer space, with a size greater or equal to the size of one data packet. In this scenario, a blocked packet is stored in the input buffer and waits for the output channel to be released. Consequently, the requirements for a packet to continue transmission is the availability of the output channel as well as the sufficient amount of buffer space at a receiving router.

The requirement for the complete packet to arrive at the router input before initiating its transmission through the router was found too restrictive. Since packets are typically formed by a number of smaller fragments, called *flits* (flow units), the *header flit* of the packet was allowed to start transmission through the router before the whole packet arrived to the router input. This approach was called *virtual cut-through* and allowed for slight performance improvement over the store-and-forward technique.

Later, the virtual cut-through scheme was extended and applied to the whole sequence of flits. This new scheme was called *wormhole* switching. The wormhole paradigm considers a packet as a sequence of flits, led by the header flit. As the header flit propagates through the network, it establishes the path, so that the subsequent flits follow the same path in order, experiencing no contention from the other packets. The path is reserved until the tail flit of the packet propagates, deallocating the resources. In this sense packet propagation resembles the behavior of a worm, stretching across several network routers, and contributing to the name of the technique. Wormhole switching considerably reduces the buffering requirements of a NoC, though the inter-packet contention is increased, as distributed packets block a higher number of router resources.

The addition of *virtual channels* to wormhole switching allows to alleviate the contention problem at the cost of larger buffer space. The concept of a virtual channel is used to denote a buffered logical channel for transmitting data between sender and receiver. Virtual channel has a dedicated physical buffer, however one physical link is multiplexed between several virtual channels. Assuming $N$ virtual channels, the total buffer space of the router inputs is subdivided into $N$ smaller buffers, while $N$ virtual links are multiplexed over one physical link. This organization enables the support of $N$ message classes, whose packets are stored in individual buffers at every router, and hence avoid blocking packets among different message classes.

The wormhole technique is the widely adopted scheme for switching in NoCs and is the one that will be assumed in the work presented in this thesis.

### Flow Control

The mechanisms of flow control actually govern the process of packet and flit propagation in the network. They define the conditions for the data units to advance and guarantee the correctness of data transmission.

A simple approach to manage data propagation is for the receiver to send an acknowledgment to the sender, once the data has been received. For this purpose, an *ack* signal is emitted by the receiver if the data has been accepted successfully, or a negative acknowledgment *nack* otherwise. This scheme is known as *ack/nack* flow control.

An alternative approach, which eliminates the unnecessary data transmission if the receiver is not ready to accept data, is to supply every sender with a *go/stop* signal governed by the receiver. The receiver maintains the signal in the *go* state while it is ready to accept new data and changes it to the *stop* state otherwise.

A more advanced flow control technique is realized by means of *credit exchange* between sender and receiver. Every sender is assigned an initial number of credits, specifying the amount of data units which the sender is allowed to transmit. The sender can only transmit data when its credit is positive. The transmission of every data unit decrements the number of available credits. This mechanism prevents the buffer overflow at the receiver if the initial number of credits does not exceed the buffer size. The receiver in turn supplies sender with additional credits as soon as the data units depart from the buffer so that the sender is able to resume data transmission.

## Routing

The routing strategy has a significant impact on network performance. Routing algorithms are used to determine the actual paths of packet propagation through the network. Every path is defined by the sequence of links and routers that the packet uses to travel between the sender and the receiver.

Routing algorithms can be classified according to the following criteria [92]:

- *Static* vs. *dynamic* routing. Static (also known as *deterministic*) algorithms define the paths for every sender-receiver pair prior to the beginning of data transmission and these paths remain fixed throughout the communication process. Dynamic (also known as *adaptive*) algorithms generate paths for every packet, considering the current state of the network, typically with the objective of avoiding congestion regions and minimize packet latency.

- *Minimal* vs. *non-minimal* routing. Minimal algorithms generate routes with the minimum hop-count between source and destination nodes, while non-minimal algorithms can favor longer paths for better communication properties (e.g. load distribution or fault tolerance).

- *Source* vs. *distributed* routing. In source routing, the complete path for a packet is defined at the source node and is incorporated in the packet header for taking routing decisions as the packet propagates through the network. In distributed routing the directions for a packet at every router are taken locally, typically by using *routing tables.*

An important and widely-used category of deterministic minimal routing algorithm, suited for regular topologies such as meshes and tori, is *dimension-ordered* routing [92]. With this approach, an order for the dimensions of the topology is defined and the packets are always sent according to the selected order of dimensions. A popular dimension-ordered algorithm for 2D topologies is *$XY$-routing*, when the packets are first routed in the $X$-dimension and then by the $Y$-dimension.

*Deadlock avoidance* is one of the important properties of the routing algorithm, which can be achieved by dimension-ordered routing in certain topologies (i.e. meshes and hypercubes) or alternatively by prohibiting routes from taking certain *turns* of the topology [59, 34]. A generic approach to guaranteeing deadlock freedom for both static and adaptive routing algorithms is based on resolving cyclic dependencies in the *channel dependency graph* [50, 52].

## Router Architecture

A router is a basic building block of an on-chip network, which performs the principle tasks for enabling communication: data buffering, arbitration and switching. The architecture of an on-chip router is highly dependent on its implementation. Figure 2.11 outlines a generic structure of a router with $n$ inputs, $m$ outputs and $k$ virtual channels [58].

The main components include the input buffers, routing and switching logic, and a crossbar switch. The internal operation of a router can be well understood by considering how a single flit traverses the router. This process can be summarized by the following steps:

- *Buffering* is performed when a flit arrives at an input port. For this purpose, every input port implements a collection of buffering slots, which can be separated into $k$ groups to support $k$ virtual channels. In the latter case, the arriving flits are demultiplexed from the physical link and stored in the appropriate buffer. Credits are delivered to the sender as the buffer slots become available, to resume data transmission.

- *Route computation* is the first step in calculating the output port, to which the flit has to be transmitted. The route computation logic can be designed either to extract the routing information from the flit itself, as in source routing, or to consult the routing tables, as in distributed routing.

- *Virtual channel allocation* is the following step, aimed at configuring the index of the virtual channel at the output port. The virtual channel allocation logic also selects the flit to be transmitted in the current cycle, among the flits waiting for transmission at the input virtual channels (as shown in the figure).

- *Switch allocation* aims at configuring the crossbar to connect the selected inputs with the outputs, according to the routing decision for the current cycle.

- *Switch traversal* finalizes the process of router traversal for a flit, after which the flit continues its path through the output link.

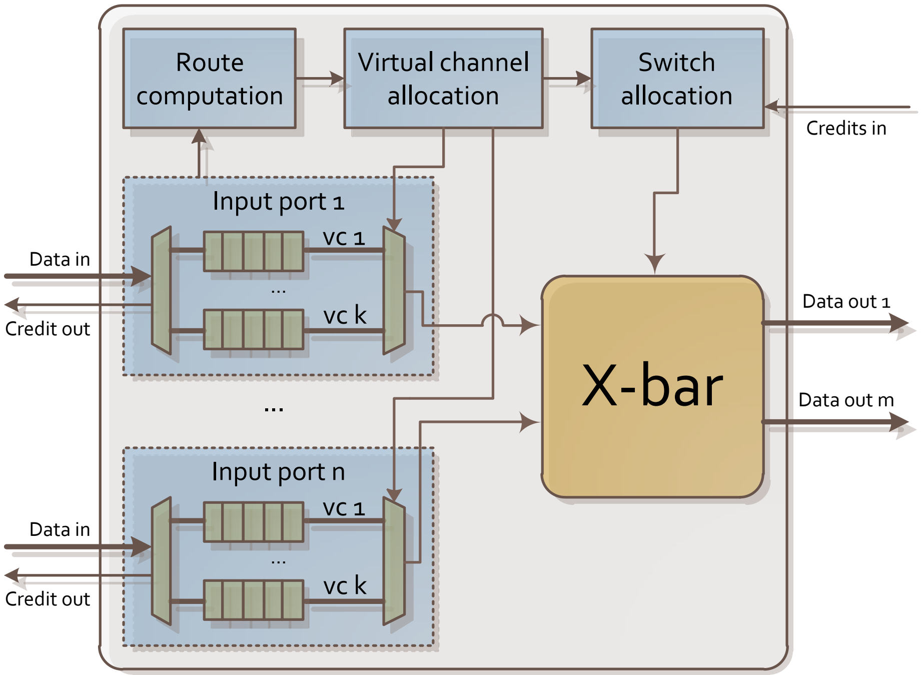


Figure 2.11: On-chip router architecture.

Note that depending on the implementation, some of the previous stages can be pre-computed (e.g. route computation) or performed in parallel (e.g. virtual channel and switch allocation), effectively reducing the router latency.

## 2.5 Related work

This section presents an overview of the state-of-art related to the problems addressed by this thesis.

### Analytical modeling of on-chip interconnects

The problem of analytical modeling of on-chip interconnects aims at estimating the system parameters (e.g. throughput, power, total latency) analytically, without the need for simulation. In this section we consider the problem of modeling the total latency for the data requests to traverse the interconnect from source to destination nodes. This latency consists of two terms. The first term is the *static* (or *hop-count*) latency, which depends on the geometrical distance between the source and destination nodes, and hence can be predicted statically, once the interconnect topology and the routing function have been defined. The second component is the *dynamic*

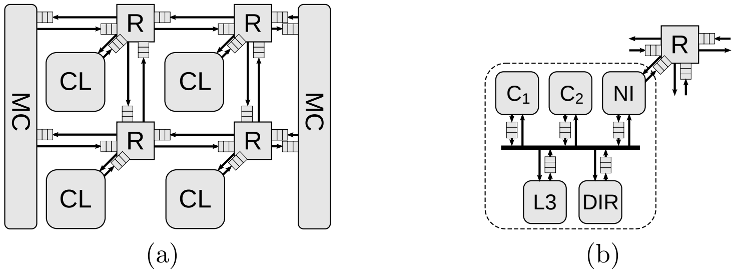


Figure 2.12: Queueing model for (a) mesh NoC and (b) bus-based cluster.

(or *contention*) latency. Contention happens in the interconnect when several data requests compete for the same shared resource, such as a bus or a NoC link. This results in additional delays experienced by the data in the buffers (queues) distributed over the on-chip interconnect.

The first analytical models appeared for basic types of multiprocessor interconnection networks, such as buses, multiple buses and crossbars [23, 121]. The emergence of more complex on-chip interconnects attracted a variety of novel techniques to the traffic modeling problem.

Figure 2.12 depicts a generic structure of a CMP with a hierarchical network-on-chip interconnect. Without loss of generality, a two-level hierarchical interconnect is shown, with a 2D mesh at the top level. Figure 2.12(a) shows the queueing representation of the top-level mesh. The mesh routers ($R$) have up to five input-buffered ports to store the incoming flits. The primary ports of the routers are connected to the clusters ($CL$), which in case of a flat CMP organization may consist of one device (e.g. a core with private caches in Figure 1.3(a)). Figure 2.12(b) presents an example of queueing model for a bus-based cluster, corresponding to one tile of the hierarchical CMP. This cluster consists of five devices, communicating via a shared bus: two cores with private caches, an instance of an *L3* shared cache, a directory and a network interface. Every device has a buffer to store the requests to the bus. To distribute the off-chip memory traffic uniformly over the mesh and avoid high contention of certain routers, memory controllers may have multiple connections to the mesh, as shown in Figure 2.12(a).

One of the approaches to estimate the contention delays in the interconnect is to apply *queuing theory* (QT) [81]. This technique models every router (or bus) as a single server and estimates the waiting time in the input queues of the server. In this work, one of the common flow control techniques is considered: *the wormhole routing* [40]. Multiple analytical models have been proposed for wormhole routers, but most of them are based on the standard M/G/1 and M/M/1 queueing models. A simplified M/G/1-based model is proposed in [49].

The model in [103] offers a convenient definition of queue delays via the injection rates in a closed form. The method calculates the probabilities of the packets coming from different bus (or router) inputs to move towards the same output. It represents an efficient generalization of the M/G/1 queueing model [81]. Another advantage of this model is the capability to deal with a variety of interconnect types, such as buses and router-based topologies (including meshes, uni- and bi-directional rings, and other topologies). The drawback of this model is that it is only capable of estimating the average packet latency in the network, which not sufficient for the QoS guarantees for individual end-to-end delay constraints.

In [56], an approach similar to [103] is proposed, offering an accurate backpressure analysis at the cost of the model efficiency. In [133] an M/M/1 approximation of link delay is used for the capacity and flow allocation task that can be applied to general networks. In [61] the authors extend the approximation of the M/M/1 model by an empirical estimation for capacity allocation under the assumption of finite buffers. Another work in [18] introduces an accurate model for heterogeneous NoCs that can be useful for exploring variable number of virtual channels and link capacities at the different levels of the hierarchical interconnect.

All mentioned QT-based models make a common assumption that the process at every router input has a Poisson distribution. While this is an acceptable assumption for the traffic sources (by definition of the model), it is known that the service times become correlated with the packet length as the packet propagates over the network [20]. Due to this fact, the distribution of the flow for the intermediate routers changes. To relax this effect, the widely applied *Kleinrock independence approximation* allows one to treat the input flows at intermediate routers still having Poisson properties. This approximation is reasonable when the packet lengths have a distribution close to exponential so that the packet service times are nearly exponential as well. However, as simulations show, in the common situation of fixed packet length, this assumption makes the analytical model too pessimistic. Chapter 3 of this thesis presents a QT-based analytical model, which eliminates the need for Kleinrock approximation and improves accuracy of modeling the networks with fixed packet length.

Queuing approaches have certain limitations, such as difficulties in modeling non-stationary traffic of bursty nature, which is especially important for heterogeneous configurations. To overcome these limitations, alternative approaches to modeling the interconnect traffic can be considered. In [25] and [26] a model inspired by statistical physics is proposed to capture non-stationary traffic behavior in network-on-chip. These works argue for self-similarity of the on-chip traffic and describe packet arrival as a non-stationary multi-fractal process.

Network calculus [83] is another popular approach that has been successfully applied for traffic analysis of the on-chip interconnects [123, 65].

## CMP design space exploration

The field of CMP design space exploration has been widely studied in the last few years. Many simulation-based frameworks extensively investigate the parameters of multi-core architectures and memory hierarchies. Exhaustive simulations were previously used for performance-oriented exploration of core and cache organizations and the off-chip memory bandwidth [69]. The work in [85] emphasizes the importance of joint optimization for the variables of core architecture (such as superscalar width and pipeline depth) and system-level variables (core count, cache size, operating voltage and frequency). Additionally, various area and thermal constraints are considered. The impact of the chip floorplan on CMP design space exploration is investigated in [93]. The power/thermal characteristics of various floorplans are analyzed, demonstrating the importance of accounting for thermal effects at the architectural level.

As the manufacturing process advances, more components fit the chip area, increasing the number of possible configurations in the design space exponentially. Therefore, exhaustive exploration of every configuration within the design space becomes intractable. Efficient simulation-driven exploration can be investigated by the means of intelligent search techniques. The work in [77] compares the application of several metaheuristics with machine-learning methods, reporting orders of runtime savings compared to exhaustive simulation. In [72] predictive modeling is used to reduce the search space by simulating sample points and teach the models to describe relationships among design variables.

A similar idea is exploited in [124], however with the application of the Design of Experiments paradigm (DoE) [120]. This statistical method enables a careful selection of a set of design points to be simulated, with the same objective of analyzing relationships between different design parameters. The obtained information is further used to tune the architectural parameters of the system, with the help of heuristics. DoE has been successfully applied to CMP design in conjunction with the Response Surface Modeling (RSM), which is another statistical method exploring the relationship between the independent and response design variables [75, 112]. After a set of architectural configurations has been generated by DoE with the objective to reduce the search space, RSM is used to refine the set and incorporate any system-level constraints to limit the set of feasible configurations. Furthermore, the DoE and RSM stages may be iteratively repeated, as in [112].

Multicube Explorer [139] represents a flexible framework for application-specific design space exloration. The evaluation of the system-level metrics is based on simulation. The tool includes implementation of several evolutionary metaheuristics, as well as the combinations of DoE and RSM approaches, giving a broad spectrum of methodologies for efficient simulation-based exploration.

As another source of speed-up for the exploration process, analytical models emerged to replace costly simulations and provide quick estimations of the CMP performance. The model in [107] studies the trade-off between the number of cores and the on-chip memory size for throughput optimization. The latency model includes a contention penalty with linear dependency on the number of cores. Apart from being inaccurate, this approximation does not allow to compare interconnects with various parameters and topologies. The work in [91] analyzes finite cache penalties in memory hierarchies, but the interconnects are also restricted to buses.

McPAT [84] is another powerful tool with low-level analytical models for area, timing, and power of multi-core architectures. However, the lack of traffic and throughput models makes it unsuitable for the characterization of hierarchical on-chip interconnects.

In [32], the authors introduce an energy-performance analytical model for CMP architectures, however they only consider bus interconnects with a simplified contention model. Their model of a CMP approximates the architectural trade-offs (such as the miss-ratio of applications versus the cache size, or the core area versus the core performance) with convex continuous functions. This enables the application of the analytical optimization methods and guarantees the uniqueness of the solution. On the one hand, this is an elegant and high-efficient approach to the complex exploration problem. On the other hand, since the architectural design space for CMPs is highly discrete, an approximation with convex functions may be rather inaccurate. Hence, the solution generated by such an approach will be far from the optimum.

In general, since analytical methods inevitably introduce certain inaccuracy when estimating the configurations, their objective is to reduce the design space, rather than to select the "best" architecture. Hence, it is highly desirable for the analytical approach to generate a moderately-sized set of nearly-optimal architectures. All these architectures can be further simulated to select the best one. This procedure reduces the chance of choosing suboptimal architectures due to inaccuracy of the analytical model.

## Task mapping for CMPs

The problem of application mapping onto the on-chip systems aims at optimizing the application performance and minimizing the total energy consumption by the system. The work in [33] proposes a framework for accurate compiler-level mapping of applications onto homogeneous mesh CMPs through detailed analysis of the instructions and allocation of data. A methodology for hardware/software partitioning and mapping of applications by parallelism extraction from the software code is presented in [16].

Given the increasing demand of scalability to the mapping algoritms, the mentioned techniques can be effectively leveraged for profiling an application and partitioning it into a set of parallel tasks. These tasks can be subsequently mapped to a CMP with an algorithm of a higher-level abstraction, therefore demonstrating better scalability. For this reason, the application to be mapped is commonly represented as a graph of parallel tasks (Fig. 2.13) with specified average communication requirements between the tasks [89].

An approach to multi-objective mapping onto mesh-based NoCs for power/performance optimization is described in [13]. Evolutionary algorithms are applied for efficient exploration of the mapping design space. Another method proposed in [68] addresses energy-aware mapping for regular topologies and uses bandwidth reservation to satisfy the performance requirements of the application. Branch-and-bound is used to minimize the total energy cost.

A number of methodologies combining mapping with several other design problems have been proposed. The work in [62] describes an approach to couple NoC topology mapping, routing and slot allocation. Real-time communication requirements are considered, guaranteeing the bandwidth and latency constraints of the application. In [111] an application-specific design flow for mapping and customization of regular topologies is proposed, with the objective to improve system performance, while keeping the original area and energy budgets. The methodology is applied to the STNoC technology. Another approach in [96] incorporates the floorplanning information into the NoC design and mapping processes so as to account for the wiring complexity of the final design. The developed tool encompasses application mapping, topology and routing synthesis, simulation and physical design for NoCs. The authors in [95] present another mapping methodology, which is combined with the generation of the optimal network-on-chip topology. Their method applies for systems that may operate in several modes, with highly varying bandwidth and latency requirements. This methodology also aims at smooth switching between different operating modes, which implies no loss in performance for applications.

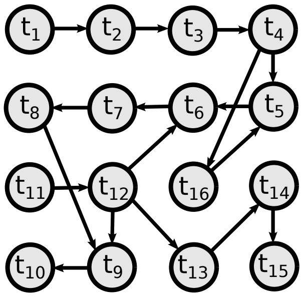


Figure 2.13: Application task graph to be mapped onto CMP

A more recent work in the field of NoC mapping considers the aspects of system robustness and yield improvement [36]. The mapping algorithm is aimed at minimizing the system power and latency, while exploiting the dynamic rerouting techniques to enable correct functionality in the presence of faulty NoC links. Given the growing interest of 3D architectures, the techniques for mapping onto 3D NoCs have drawn recent attention. In [130] the problem of run-time incremental mapping is discussed and solved heuristically, minimizing the total energy, satisfying the thermal constraints of the applications and reducing the possibility of resource fragmentation.

The emergence of advanced techniques for power management, such as voltage islanding [82, 106], triggered new research in mapping algorithms. The approach in [88] considers performance constraints of the application, although it does not account for the communication component of power. A more realistic approach is proposed in [57] in which computation and communication are both optimized taking into account a third component related to voltage shifters. Thermal-aware island partitioning via evolutionary algorithms is proposed in [70]. The distinction of different processor classes is introduced in [127], but assuming that every processor can run at an independent voltage level.

The research on power-aware mapping usually assumes that the voltage islands are defined pre-silicon during task mapping in application-specific SoCs, and often disregarding the cost of implementing the voltage islands. The problem of application mapping onto general-purpose chip multiprocessors has to address the efficient usage of CMPs *after fabrication*. This means that the voltage islands have already been planned, imposing an additional constraint for the mapping problem: all cores in each island have to run at the same voltage level. As it was discussed in Section 1.2 voltage islands have a high design cost, hence it is realistic to assume that for CMPs with hundreds of cores every island will incorporate several cores. Hence, the mapping algorithm has to intelligently select groups of tasks for the same voltage island, in order to minimize the power of the island, while delivering the required performance for all tasks.

The mapping approach proposed in Chapter 6 of this thesis contributes by considering the predefined maps of voltage islands and a variety of processing units, offered by heterogeneous CMPs. It shows high scalability, due to a high-level abstraction of application with a task graph.

## Topology customization and routing for NoCs

Topology of the on-chip interconnect crucially affects the main parameters of the system, such as performance, area, power consumption, reliability and fault tolerance. Furthermore, the selection of the NoC topology is tightly coupled with the process of designing the routing algorithm and guaranteeing its correctness (i.e. deadlock- and livelock-freedom).

The topology selection problem for NoC is strongly affected by the application domain. Due to the unpredictability of the traffic at design time, general-purpose CMPs tend to instantiate topological solutions based on regular structures. Widely-applied regular topologies include rings [90], two-dimensional meshes and tori [60], concentrated meshes [15] and flattened butterflies [78]. Hierarchical topologies, such as hybrid NoC-buses [43] have demonstrated to improve the average system latency.

Using regular topologies for application-specific systems not only helps to shorten the design time, but also favors better electrical properties of the interconnect. However, since the traffic requirements for ASICs can be estimated a priori, the design of custom topologies is typically preferred in order to optimize the system performance and resource consumption. A variety of works addresses the problem of effective topology generation for ASICs [114, 104, 125]. In [115] a formal description and a methodology for the problem of custom NoC synthesis are presented. The wide spectrum of studies offer topology comparisons in terms of the selected criteria. Some examples of these studies can be found in [19] and [128].

However, even the interconnect topologies based on regular structures can be efficiently *customized* for application-specific systems. Customization may be performed either by removing particular links from the full topology, as in [27], or by adding new links between the components [105]. The latter work shows that the “small-world” properties of the resulting solution can deliver significant performance improvements.

Finally, topology selection is one of the most tightly related problems to VLSI layout aspects. Thus, the works in [137] and [11] incorporate physical planning tools into the topology selection process in order to optimize physical properties of the design.

The *routing algorithm* is another important property of the on-chip interconnect, significantly affected by the interconnect topology. Static routing can either be implemented as a dimension-ordered strategy [39] or by means of the routing tables [68]. In adaptive routing the paths are decided at runtime [39]. This approach can incorporate fault-tolerance techniques to increase the network reliability, since packets may choose alternative paths, when certain components fail to operate as expected. A combination of static and dynamic strategies is also possible and it was proposed in [67].

Routing techniques are mostly topology dependent. For instance, dimension-ordered routing is not feasible for irregular meshes. To adapt this popular routing strategy, several works propose its extension for irregular topologies [28, 122, 66].

An indispensable property of routing algorithm is its correctness, such as the freedom from deadlocks, livelocks and starvation [52]. The odd-even routing [34] and turn prohibition [59] schemes were developed to guarantee deadlock-freedom in meshes. A more general approach to deadlock-free routing is based on the construction and analysis of the *channel-dependency graph*, with the objective to remove cyclic dependencies [52, 39]. Insertion of virtual channels to resolve cycles in the channel-dependency graph for deterministic routing was first proposed in [40]. In [51] a necessary and sufficient condition for deadlock-free adaptive routing functions were formulated and [50] proposed two methodologies for deadlock-free adaptive routing by insertion of either virtual or physical channels for improved fault-tolerance.

# Chapter 3

# Modeling Networks with Constant Service-Time Routers

This chapter describes the first contribution of the thesis, which is the queuing model for estimating the average end-to-end latency in NoC. The model addresses a special class of networks with *fixed packet length*. This assumption holds for certain on-chip systems, such as CMPs, where the data packets carry cache lines of a fixed size. This research was published in [100].

The objective of the analytical model for on-chip interconnect is to provide a rapid and accurate estimation of the network contention delay. *Queueing theory* (QT) [81] is often used to model the network contention. QT represents a technique to estimate the queueing delays of users waiting for some service of the system, given the customer distribution and the service process characteristics. Therefore it can be applied to model the delays of packets waiting for the router service. The QT provides an M/D/1 model describing a system with Poisson input and constant service time [20].

This work shows that due to the correlation of packet size and service time, the Poisson property of the flow degrades as packets propagate through the network. This dependency affects substantially the delay values so that the M/D/1 modeling with the Poisson input assumption is no longer accurate. Hence [100] proposes *a number of intuitive equations to significantly improve the delay estimation accuracy for the networks with constant service-time routers (CSTR).* The new model is referred to as the *constant-time model* (CTM).

## 3.1 Model overview

Modeling a network as a system of routers requires an approach for calculating packet delays depending on the network topology. An example of a 3x3 mesh network is presented in Figure 3.1. Each router (R) is connected to a processing element (PE) and a set of neighboring routers by two unidirectional links. In our model we focus on the delay estimation (waiting time) at the input buffers. Hence, we assume each router to have an input buffer from each neighbor and one from the PE.

A net is an end-to-end route in the network, from one PE to another, and can be represented as a sequence of routers that a packet must propagate through. Each router may have an arbitrary number of input channels.

Consider the simple example in Fig. 3.2: a chain of two servers with constant service time $T$ and a Poisson arrival process with rate $\lambda_1$. The classical QT model for a constant-time server with Poisson input is the M/D/1 model. The waiting time in the first input queue $W_{S1}$ can be estimated using Pollaczek-Khinchin (P-K) formula [81]:

$$W = \frac{\lambda T^2}{2(1 - \lambda T)}. \tag{3.1}$$

However, unlike the systems with exponentially distributed service times, the output from an M/D/1 system is no longer a Poisson process: the time between two consecutive outputs is guaranteed to be greater than or equal to the service time value $T$. In other words, a constant-time server produces a *de-randomization* of the Poisson process, thus reducing the degree of randomness of the inter-arrival times, which is the main cause of contention delays at the input queue.

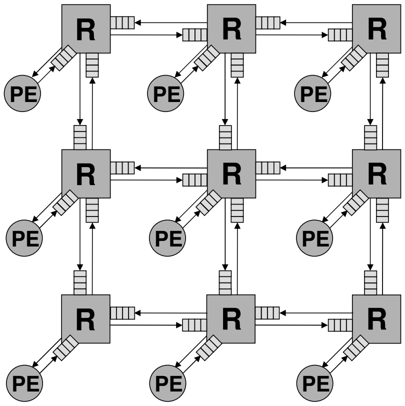


Figure 3.1: 3x3 mesh Network-on-Chip example.

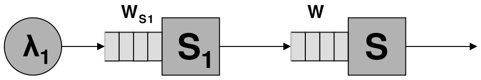


Figure 3.2: A simple server chain.

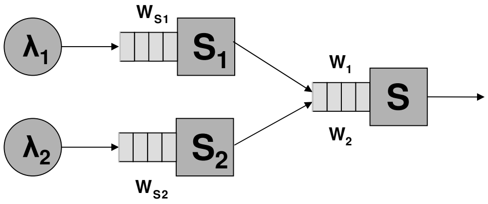


Figure 3.3: Merging two M/D/1 output flows.

While $W_{S1}$ can be accurately predicted with (3.1), using the same equation to estimate the waiting time at the input queue of S may result in significant errors. More precisely, the waiting time $W$ will be equal to zero as the successive arrivals of packets at S occur not earlier than $T$, which is exactly the time to process a packet by server S. That means that the incoming packet will never wait for his service. This fact illustrates the inaccuracy of applying (3.1) for the estimation of the waiting time at server S.

In the general case, a router may have an arbitrary number of inputs, including constant-time server output flows. The goal of this work is to derive the equations for an accurate queueing delay estimation, as an alternative to the P-K equation.

The qualitative importance of an accurate queueing delay estimation is shown in the following example. Consider now two inputs to server S, each one being an output from the constant service time systems S1 and S2 (Fig. 3.3). Table 3.1 reports the estimation of the waiting times $W_1$ and $W_2$ by the M/D/1 model and the constant service time model (CTM) of this work depending on different input rates ($\lambda_1$ and $\lambda_2$). The total delay at the server for the packets of the first flow is $D_1 = T + W_1$, where $T$ is the packet service time (assumed to be 1 cycle for the example).

Table 3.1: Modeling results for the example in Fig. 3.3.

| | | CTM | | | M/D/1 | | SIM | Error | |
|---|---|---|---|---|---|---|---|---|---|
| $\lambda_1$ | $\lambda_2$ | $W_1$ | $W_2$ | $D_1$ | $W_1, W_2$ | $D_1$ | $D_1$ | CTM | M/D/1 |
| 0.1 | 0.1 | 0.07 | 0.07 | 1.07 | 0.13 | 1.13 | 1.07 | 0.1% | 6% |
| 0.3 | 0.3 | 0.53 | 0.53 | 1.53 | 0.75 | 1.75 | 1.54 | 0.7% | 14% |
| 0.5 | 0.1 | 0.29 | 0.47 | 1.29 | 0.75 | 1.75 | 1.29 | 0.3% | 36% |

The traffic rates of the flows (flits/cycle) are specified in the first two columns of the table. The following three columns show the waiting times and the delay of the first flow (in cycles) obtained by the CTM model. Next, the estimations for $W_1, W_2$ and $D_1$ obtained with the M/D/1 model are reported. The column `SIM` displays the delay of the first flow obtained by simulation. Finally, the error of $D_1$ with regard to the simulation is reported in the last two columns.

One can easily observe the reduction of waiting times, which is the result of the de-randomization ($W_1$, $W_2$ of CTM vs those by M/D/1). Another important aspect is that the Poisson-input models assume equal waiting times for all input streams regardless their rate. This is not true in the case of constant service time systems. This results into larger differences in delay estimation (see the difference between $W_1$ and $W_2$ in the last row of the CTM model). We also note that the difference between our model and simulation is less than 1% in this example, while M/D/1 reveals up to 36% of overestimation. This simple example shows that proposed CTM approach can provide more accurate estimations. This is essential for QoS optimization.

## 3.2 Queueing model

This section presents the main contribution of this chapter: the QT model for constant-time routers.

### Definitions and assumptions

We use the following definitions:

- A *router* is a basic entity that routes traffic in a network. The router is also referred to as *server* in the nomenclature of queueing theory.
- A *packet* is a data transmission entity. A packet consists of one or more *flits* that are the minimum transmission units.
- An *input flow*, also referred to as *input process*, is an arrival process at one of the router input buffers.
- A *traffic source (sink)* of the net is a processing element that injects (consumes) packets to (from) the network.
- The *traffic rate* $\lambda_k$ of the net $k$ is the average rate of packet generation at the net source.

- The *waiting time* at the input buffer of a router is the average steady-state time the packets spend in the buffer before being processed by the router. In the QT nomenclature it is referred to as a *queueing delay* at the input queue (buffer).

The following assumptions are considered:

- Traffic sources generate packets according to a Poisson distribution.
- Traffic sinks consume packets immediately.
- The input buffers of the routers have infinite capacity.
- The packets have fixed size and the routers take constant time to process them.

## A simple model for a 2-input router

We have already stated an important difference between a system with exponentially distributed service times and one with constant-time service, assuming a Poisson input in both. The output process from the former is also a Poisson process of the same rate and exponentially distributed inter-arrival times. Thus we may use (1) for M/M/1 systems to estimate the waiting time for any of the routers. The output flow from a constant service time system with Poisson input has a complex distribution discussed in [109] and [110]. The important property due to its deterministic service time is that the time between two successive outputs is not less than the service time $T$. Because of this fact, the waiting time for all the routers in a chain will be equal to zero except for the first one (Fig. 3.2). The first router delay can be successfully estimated with (1), since the packet generators are said to generate packets according to a Poisson distribution.

Consider the two-input server example presented in Fig. 3.3. All three servers S1, S2 and S, have a constant service time $T$. Both sources generate packets assuming a Poisson distribution with average traffic rates $\lambda_1$ and $\lambda_2$. Our goal is to model the waiting times $W_1$ and $W_2$ at the input queue of the server S.

For example, note that if the flow $\lambda_2$ were not sending packets, i.e. $\lambda_2 = 0$, then $W_1$ would be equal to zero as in the single-input server case. This fact supports the idea that the waiting time $W_1$ is generated by the packets of the complementary flow $\lambda_2$ and its value depends on the traffic rate of both flows.

To simplify the analysis, we use the concept of *mean residual service time* $R(\lambda)$ for an input flow [20]. If an incoming packet $P_i$ arrives at the server queue at time $t_i$ while some other packet $P_j$ is being processed by the server, then the residual time $R_i$ for the packet $P_i$ is the time left for $P_j$ to finish its service. The mean residual service time is an average value of residual times for each packet defined by the service time and the flow traffic rate. The following equation represents its steady-state value [20]:

$$R(\lambda) = \frac{1}{2}\lambda T^2. \tag{3.2}$$

Using the definition of residual time, the Pollaczek-Khinchin equation can be rewritten as

$$W(\lambda) = \frac{\lambda T^2}{2(1-\lambda T)} = \frac{R(\lambda)}{1-\lambda T}. \tag{3.3}$$

We generalize the above expression by distinguishing traffic rates in the P-K formula. Let us consider the traffic flow of rate $\lambda_{tr}$ that is merged with some complimentary flow of rate $\lambda_{res}$ at the router input. As our experiments show, the waiting time for the packets of flow $\lambda_{tr}$ will depend on both rates. Then we can rewrite (3.3) as

$$W(\lambda_{tr}, \lambda_{res}) = \frac{\lambda_{res} T^2}{2(1-\lambda_{tr} T)} = \frac{R(\lambda_{res})}{1-\lambda_{tr} T}. \tag{3.4}$$

This generalized waiting time can be treated as that of the traffic flow $\lambda_{tr}$ experiencing a delay produced by the complementary flow with rate $\lambda_{res}$, inducing a residual time $R(\lambda_{res})$. Using (3.4) in combination with the standard M/D/1 equation (3.3) we propose the following empirical expression that was found to provide an accurate estimation for $W_1$ and $W_2$:

$$\begin{aligned} W_k &= W(\lambda_1 + \lambda_2) - W(\lambda_1) - W(\lambda_2) + W(\lambda_k, \lambda_j) = \\ &\phantom{=} W(\sum_{i=1,2} \lambda_i) - \sum_{i=1,2} W(\lambda_i) + W(\lambda_k, \lambda_j), \end{aligned} \tag{3.5}$$

where $k \in \{1, 2\}$ and $j$ is the complementary flow for $k$.

The form of (3.5) was suggested intuitively resulting from the numerous experimental observations and was further empirically verified for a large range of input parameters $\lambda_i$ and $T$. The intuition behind this equation can be explained with the following considerations. Expression (3.5) has three terms: the first one is the M/D/1 waiting time (3.3) that input packets would observe if both inputs were Poisson processes. The second term is the sum of the M/D/1 waiting times for each separate flow. It can be considered as the measure of "de-randomization" introduced by the source constant-time router. In fact this is the waiting time packets of each

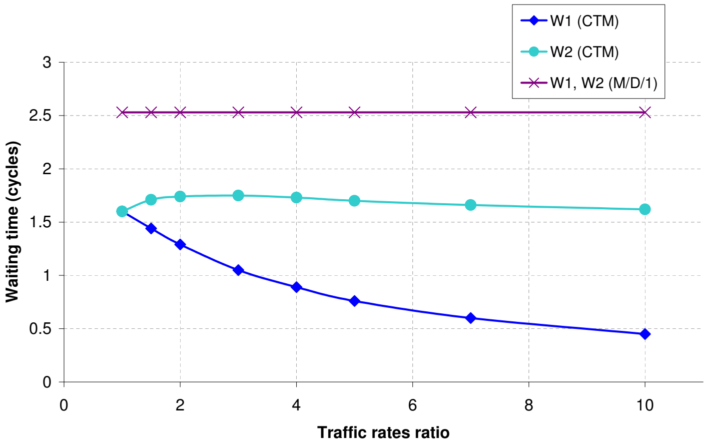


Figure 3.4: Estimated waiting time for different traffic rates ratio ($\lambda_1/\lambda_2$).

input process spend at the source router. The last term estimates the impact of the complementary inputs on the $k$-th input, similarly as it was discussed in (3.4).

An important fact that is not evident and not observed in M/D/1 systems modeling is that the waiting time for the input flows differs when the traffic rates are not equal. This phenomenon is also proved by simulation. When two flows interact at the router input, the packets of the flow with greater traffic rate have less average input delay. Indeed, if a packet belongs to the flow with the smallest rate it has greater probability to be blocked by a packet of the complementary flow. As a result, in average its waiting time will be greater than that of the packet of complementary flow. An illustration of this fact is presented in Fig. 3.4. The traffic rate ratio $\lambda_1/\lambda_2$ ranges from 1 to 10 while their sum is kept constant ($\lambda_1+\lambda_2 = 0.4$ flits/cycle). The waiting time $W_1$, $W_2$ estimations by both models are depicted.

One can observe the increasing difference between $W_1$ and $W_2$ estimated by the CTM model as the rate ratio increases. In contrast, the M/D/1 model provides a pessimistic constant waiting time for both flows that does not depend on the ratio of traffic rates.

## Generalization for an N-input router

Equation (3.5) can be generalized for an arbitrary number of inputs $N > 2$. The waiting time $W_k^N$ at input $k$ can be calculated as

$$W_k^N = W(\sum_{i=1..N} \lambda_i) - \sum_{i=1..N} W(\lambda_i) + W(\lambda_k, \sum_{i=1..N,\ i\neq k} \lambda_i) \tag{3.6}$$

It is also valuable to notice that this equation holds for the case of one input flow ($N = 1$). In this case we obtain $W_1^1 = 0$, which is consistent with zero delay at all routers in a chain, except for the first one.

## Hybrid process at the router input

Another important case to consider when modeling a network is a hybrid input process consisting of Poisson flows and constant-time router outputs. It is necessary for modeling the traffic coming from two different sources: the Poisson processes from the PE's and the traffic coming from constant-time routers. We start again considering a simple two-input server example where one net is an M/D/1 output and another is a Poisson source (Fig. 3.5).

Combining (3.2), (3.3) and (3.4) we obtain the following expressions:

$$W_1 = W(\lambda_1 + \lambda_2) - W(\lambda_1) - R(\lambda_2) + W(\lambda_1, \lambda_2), \tag{3.7}$$

$$W_2 = W(\lambda_1 + \lambda_2) - W(\lambda_1) + R(\lambda_1). \tag{3.8}$$

This result can be extended to the general case presented in Fig. 3.6. Consider a complex arrival process at the input queue of server S: an arbitrary number $d$ of M/D/1 output flows ($\lambda_1, .. , \lambda_d$) and $p$ Poisson flows ($\lambda_{d+1}, .. , \lambda_{d+p}$). Let us denote the set of all M/D/1 outputs as $D$ and set of all Poisson sources as $P$. Let also $N = d + p$ be the total number of inputs.

As the sum of Poisson processes with rates $\lambda_k,\ k = d+1,\ ..\ , d+p$ is also a Poisson process of the total rate $\lambda_p = \sum_{k=d+1}^{d+p} \lambda_i$, we can treat the source flows as a single input with the total rate $\lambda_p$. This results into packets of all the flows in $P$ experiencing equal waiting time.

Finally we present the generalized equations for the waiting time $W_k^N$ estimation. For any input flow $k \in D$

$$\begin{aligned}\forall k \in D: \quad W_k^N \quad = \quad & W(\sum_{i=1..N} \lambda_i) - \sum_{i \in D} W(\lambda_i) - \\ & \sum_{i \in P} R(\lambda_i) + W(\lambda_k, \sum_{i=1..N,\ i \neq k} \lambda_i),\end{aligned} \tag{3.9}$$

and for any Poisson input flow $k \in P$

$$\forall k \in P: \quad W_k^N = W(\sum_{i=1..N} \lambda_i) - \sum_{i \in D} W(\lambda_i) + \sum_{i \in D} R(\lambda_i). \tag{3.10}$$

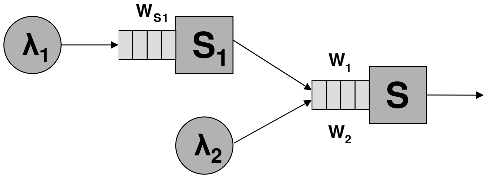


Figure 3.5: Merging an M/D/1 output and a Poisson flow.

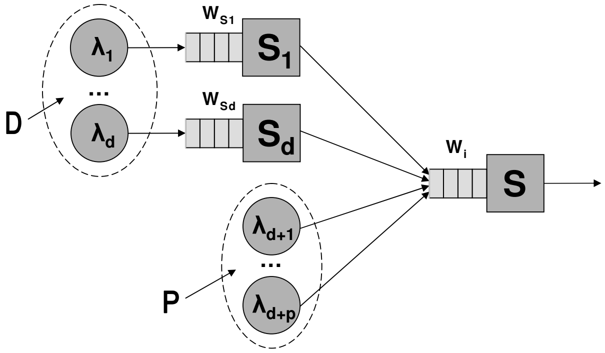


Figure 3.6: Hybrid input process at a constant-time router.

We note that in case $P = \emptyset$, equation (3.9) reduces to (3.6), thus demonstrating that (3.9) is the generalized case of the server not having traffic source inputs. Also note that in case of pure Poisson input flows, i.e. $D = \emptyset$, equation (3.10) is reduced to

$$\forall k \in P : \quad W_k^N = W(\sum_{i=1..N} \lambda_i), \tag{3.11}$$

that is exactly the waiting time expression for the M/D/1 system. Thus, *our equations are consistent with the M/D/1 model and provide a simple and accurate extension for the hybrid case of input processes.* Another interesting fact is that we can apply (3.10) for waiting time estimation at the sources of the nets and at any point in which the input traffic can be modeled by a pure Poisson process estimated by the Pollaczek-Khinchin formula (3.1).

We are using the pair (3.9)-(3.10) to predict contention delays at the input buffers of the network routers and build our delay model around them. As we show in the experimental section these equations allow accurate estimations for a wide range of input model parameters.

## 3.3 Network model

This section introduces the model of a network used in this work.

### Router model abstraction

We represent a network as a system of connected routers. Given the single router model presented in the previous section, we now use it to model the behavior of a network. We use the router model under the following assumptions:

- The router is regarded as a black-box with constant service time for incoming packets and infinite capacity buffers.
- The router can only transmit one packet at a time. Simultaneous packet transmission is not considered.
- The strategy of processing inputs in order may vary. We discuss it in the experimental section.

### Extension of the model for a network

As we have already discussed, (3.9) and (3.10) provide the waiting time estimation at the input of an arbitrary router depending on the network topology and traffic rates at the inputs. The traffic process in a network is a complex process that is the result of merging and splitting the traffic flows of individual nets. In this work we address the waiting time estimation at the input buffers in case of merging $N$ flows at the router input but we do not discuss the split process. However, as will be shown in the experimental section, by only considering merging processes we already obtain a significant accuracy in the estimation.

An important feature of a Poisson process served by a constant-time router is that it accumulates "de-randomization" introduced by the latter. Formally, let us consider $N$ processes that have constant-time outputs, each one having some traffic rate $\lambda_i$. Once these processes have been merged at the constant-time router they become a single constant-time output process with the rate $\lambda = \sum_{i=1}^{N} \lambda_i$ that satisfies (3.9)-(3.10). We represent this fact in Fig. 3.7. Here two flows $\lambda_1$ and $\lambda_2$ that are M/D/1 outputs merge at router S to form a single flow at its output. We use (3.9) to estimate the waiting times at router S. Now the new flow travels to the router S', where it merges with the flow $\lambda_3$. In order to estimate the waiting times at router S', we apply (3.9) again assuming two input flows with rates $\lambda = \lambda_1 + \lambda_2$ and $\lambda_3$.

This fact basically says that the waiting time of a constant service time system output process does not depend on the way it propagates in the network but only

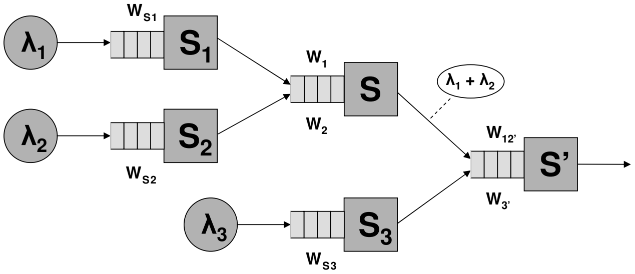


Figure 3.7: Successive flow merge.

on the traffic rate at a particular router. As a result, we do not require information about the flow propagation and are able to calculate delays at each router independently. Below we show how independent router delays are joined to form the end-to-end delay of each net.

## Net delay estimation for wormhole routing

The estimation of the full delay for a particular net is performed based on the wormhole routing strategy. In wormhole routing, a packet is transmitted by processing the request of its first flit, also referred to as a *header flit*. The header flit notifies the router to be served as soon as it arrives at the router input queue. Once the router has granted the connection to the header flit, the rest of the packet flits follow in a pipeline manner. To introduce the equation for the delay we use following nomenclature:

- The *routing path* $P_i$ of a net $i$ is the sequence of routers traversed by the packets of this net, from source to destination routers (including both).
- The *packet size* $S$ represents the number of flits in a packet. The packet size includes the header flit and is a constant value for all packets in our model.
- The *header service time* $HS$ denotes the time necessary for the router to grant a connection, i.e. find the appropriate output channel and establish the connection. $HS$ does not include the waiting time at the queue.
- The *flit transmission time* $FT$ is the time necessary to transmit one flit that is not a header in a pipeline manner. In our model we assume $FT = 1$ cycle.
- The *end-to-end average packet delay* $D_i$ of a net $i$ is the average time the packets of the net spend in the network, starting from the injection at source router and exiting at the output of the destination router.

- The *average waiting time* $W_{ij}$ at router $j \in P_i$ of the packets of net $i$ is the waiting time the packet header spends in the router queue before being processed.

The end-to-end net delay model for wormhole routing that we use incorporates two terms. The first term depends on the topology and geometrical distance between the net endpoints, thus it can be predicted statically. It is usually referred to as a hop-count delay $D_i^{hc}$ of net $i$. As follows from the wormhole routing strategy, the hop-count delay consists of the time to propagate header ($HS$) and the time necessary for the rest of packet flits to reach the destination router:

$$D_i^{hc} = \sum_{j \in P_i} HS + FT(S-1) \tag{3.12}$$

The second term of the net delay is the contention delay $D_i^c$, that is the sum of the input buffer delays $W_{ij}$ over all routers in the path $P_i$:

$$D_i^c = \sum_{j \in P_i} W_{ij} \tag{3.13}$$

The contention delay occurs when several input packets compete for the same router outputs. It is estimated with the model we present in Sect. 3.2. Hence, the full end-to-end delay of net $i$ is defined by the following expression:

$$D_i = D_i^{hc} + D_i^c = \sum_{j \in P_i} (HS + W_{ij}) + FT(S-1) \tag{3.14}$$

Finally we use (3.14) to calculate the end-to-end delay of every net assuming that (3.9)-(3.10) provide the waiting times $W_{ij}$. The constants $HS$, $FT$ and $S$ are the input parameters. According to the wormhole strategy, the packet router service time $T$ used in the calculation of $W_{ij}$ is the sum of the header service time and the propagation time for the remaining flits. Formally, $T$ is defined as:

$$T = HS + FT(S-1). \tag{3.15}$$

## 3.4 Experimental results

Traffic flows in a network are complex flows that can be described as a system of merging and splitting processes at every router. The delays experienced by every input also depend on the priority scheme of the merging inputs. As it was discussed, the equations (3.9)-(3.10) provide an estimation of the waiting time in the input buffers of constant-time router assuming a first-come, first-served (FCFS) scheme of merging the arrival processes at the router inputs.

In this section we first show the accuracy of our equations by analyzing networks characterized by the merge of processes. We present an experimental proof of the fact that our equations are capable of estimating the delay within a wide range of input parameters such as load and packet size. Then we also compare our model with M/D/1 for different priority schemes at the router inputs, namely *round-robin* (RR) and *longest queue* (LQ). LQ stands for prioritizing service of the input with the largest number of flits in the buffer. Finally we apply (3.9)-(3.10) to arbitrary networks and show that our equations provide a noticeable improvement in the estimation of end-to-end delays, even without a detailed analysis of flow splitting.

An important fact about the experiments is that we investigate *worst-case* delay errors over the network, i.e. the largest error estimating the net (buffer) delays from all nets (buffers) in the network. Many of the models suggested so far only estimate average net delays. This is not suitable for QoS evaluation with end-to-end delay constraints.

We compare our CTM model and the classical M/D/1 model with the simulations performed by an accurate flit-level simulator written in C++. Even though we use mesh topologies in the experiments, our methodology is applicable to any type of network since the delay estimation is independent from the network topology (discussed in Sect. 3.3). The simulations on meshes are performed to simplify the benchmark suite and the architecture of the simulator.

## Single-output router networks

This is a special type of networks we use to emphasize the accuracy of our model for constant service time networks. As the pair (3.9)-(3.10) provides delay estimation for a merging process, we first focus our analysis on this type of networks to avoid splitting at the output of the router, i.e. every router sends packets only to one direction. We start with a 3x3 mesh network with uniformly distributed traffic and FCFS priority scheme. In this experiment we measure the deviations between the net and buffer delay values obtained by simulation and estimation by CTM and M/D/1 models. We show the modeling error dependency on the maximum router utilization (flits/cycle) determined by the network load.

### Packet size variations

Varying the packet size from 1 to 100 flits and the router utilization from 0.10 to 0.90 flits/cycle, we note that the CTM worst-case error does not exceed 0.25% for net delays and 2% for buffer delays, while the errors produced by the M/D/1 model are 9% for net delays and almost 100% for buffer delays. In the other experiments we fix packet size to 5 flits per packet.

### Changing priority schemes

Although our experiments assume a FCFS priority scheme for the router inputs, we show that it can also be a good approximation for other schemes such as RR and LQ. Table 3.2 presents the relative errors of the worst-case estimation of net and buffer delays versus simulation at different traffic loads. We select three values for router utilizations to demonstrate model behavior at low, medium and high traffic loads.

The first two columns of the table represent the utilization and the priority scheme. The values in the other columns report the errors between the estimated ($D_{est}$) and simulated ($D_{sim}$) delays, calculated as

$$Err = \frac{|D_{sim} - D_{est}|}{D_{sim}} \cdot 100\%. \tag{3.16}$$

The third and fourth columns represent the worst-case errors of the net and buffers delays estimated by the CTM model with respect to simulation. The last two columns are the errors of the M/D/1 estimation. One can see a significant improvement in the accuracy of CTM model against the classical M/D/1 model assuming low and medium traffic loads (utilizations). At high loads, the FCFS scheme still provides a high accuracy. For the other schemes, the gap between CTM and M/D/1 tends to reduce.

Table 3.2: Relative delay error between simulation and analytical models for a 3x3 single-output network.

| Utilization | Priority | CTM error (%) | | M/D/1 error (%) | |
|---|---|---|---|---|---|
| (flits/cycle) | scheme | net | buffer | net | buffer |
| 0.10 | FCFS | 0.06 | 1.67 | 1.39 | 95.46 |
| | RR | 0.05 | 2.55 | 1.39 | 94.22 |
| | LQ | 0.05 | 2.58 | 1.41 | 95.29 |
| 0.50 | FCFS | 0.07 | 0.89 | 5.43 | 83.46 |
| | RR | 0.32 | 6.73 | 5.73 | 94.79 |
| | LQ | 0.72 | 5.76 | 5.98 | 93.09 |
| 0.90 | FCFS | 0.24 | 0.65 | 5.38 | 61.49 |
| | RR | 16.44 | 34.17 | 20.62 | 77.81 |
| | LQ | 13.65 | 22.34 | 14.29 | 73.22 |

**Non-uniform traffic**

In this experiment (Table 3.3), we change the traffic distribution to be highly non-uniform. We observe that the M/D/1 model provides an overly pessimistic estimation of the delays for a medium traffic load. On the other hand, the CTM model still provides an accurate estimation for the FCFS scheme.

The difference between the CTM and M/D/1 models increases for all priority schemes. Hence, the CTM model is more accurate for the delay estimation assuming a non-uniform traffic distribution.

## General networks

In this section, we present results for two general networks without the single-output router assumption. Although our model does not consider the splitting of the traffic flows, its application improves the delay estimation in comparison with the M/D/1 model, even for general networks with arbitrary configuration.

Table 3.4 represents the results for 3x4 mesh with highly communicating central nodes. The conclusions for this experiment are similar to those for the previous experiments. The relative error is sometimes reduced by several tens of percent.

*A large example:* We also present comparative results for an 8x8 mesh NoC under a variety of traffic loads. We have generated an arbitrary 8x8 network with 64 nets that are randomly distributed over the network, each net having the same traffic rate. Fig. 3.8 depicts the traffic distribution between the network links for this particular example, so that the darker arrows correspond to the links with higher traffic rates. The experiments were carried out for the utilizations in the range of 0.1 to 0.7 flits/cycle. The relative errors for the net delays are shown in Fig. 3.9. There is a tangible reduction of the error of CTM model at low and medium loads when compared to M/D/1 model. Another important fact is that the simulation of every configuration took several minutes to ensure a level of confidence smaller than 1%. The application of the CTM model took about 0.3 msecs.

Table 3.3: Relative delay error between simulation and analytical models for a 3x3 network with non-uniform traffic.

| Utilization | Priority | CTM error (%) | | M/D/1 error (%) | |
|---|---|---|---|---|---|
| (flits/cycle) | scheme | net | buffer | net | buffer |
| 0.50 | FCFS | 0.51 | 1.41 | 23.80 | 122.50 |
| | RR | 5.95 | 21.15 | 23.98 | 123.66 |
| | LQ | 16.02 | 42.87 | 27.24 | 161.17 |

Table 3.4: Relative delay error between simulation and modeling for a 3x4 network with highly communicating central nodes.

| Utilization | Priority | CTM error (%) | | M/D/1 error (%) | |
|---|---|---|---|---|---|
| (flits/cycle) | scheme | net | buffer | net | buffer |
| 0.10 | FCFS | 0.34 | 19.02 | 1.97 | 99.92 |
| | RR | 0.58 | 27.96 | 2.23 | 114.94 |
| | LQ | 0.68 | 25.86 | 2.25 | 111.41 |
| 0.50 | FCFS | 2.01 | 14.15 | 7.05 | 71.41 |
| | RR | 5.75 | 34.89 | 9.39 | 85.60 |
| | LQ | 3.93 | 31.21 | 8.83 | 84.74 |
| 0.90 | FCFS | 5.54 | 23.93 | 11.80 | 76.65 |
| | RR | 28.05 | 78.76 | 33.14 | 102.35 |
| | LQ | 13.31 | 54.12 | 18.87 | 100.28 |

To sum up, the CTM model with different priority schemes provides a notable improvement in accuracy in comparison with the M/D/1 model within wide range of traffic loads. The buffer delay estimation improvement reaches up to several times in absolute value, while the net delay mainly determined by the hop count at low and medium loads improves up to several tens of percent. This is a significant result for designs that have QoS constraints for end-to-end latencies.

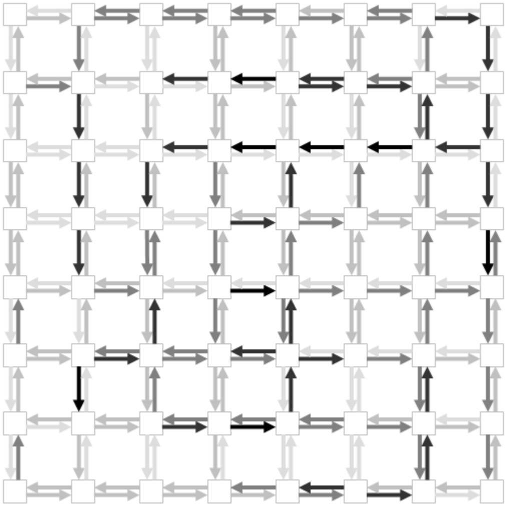


Figure 3.8: Traffic distribution for an 8x8 mesh NoC example.

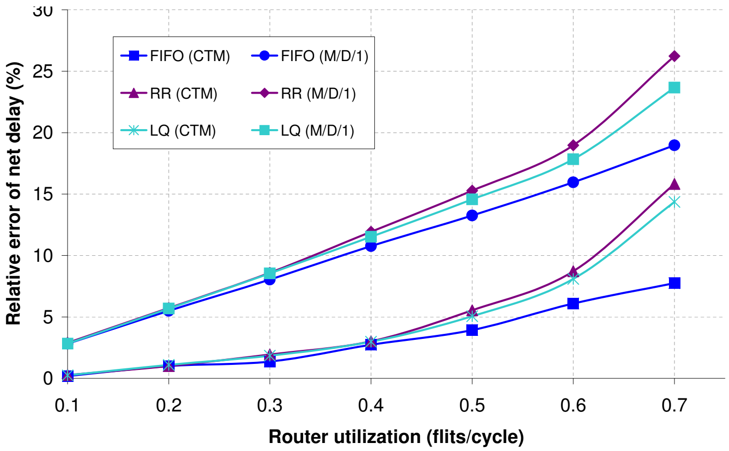


Figure 3.9: Relative errors between simulation and modeling for 8x8 NoC.

## 3.5 Conclusions

This work addresses the problem of modeling constant service time systems via queueing theory. The major contributions can be summarized as follows:

- It has been demonstrated that the classical Markovian models provide pessimistic approximation of waiting times for CSTR systems, due to the effect of “de-randomization” at intermediate routers.
- A constant-time model has been proposed to eliminate the assumptions of the Markovian models and improve the modeling precision.
- The results have been validated for networks with variable sizes, different packet lengths, traffic loads and priority schemes.

The traffic process in a network is a complex process that is the result of merging and splitting the traffic flows of individual nets. This work addresses the estimation of the waiting time at the router inputs for the case of merging an arbitrary number of flows. Future work on this model should cover the splitting process to improve the modeling precision.

# Chapter 4

# Analytical Modeling of CMP Architectures

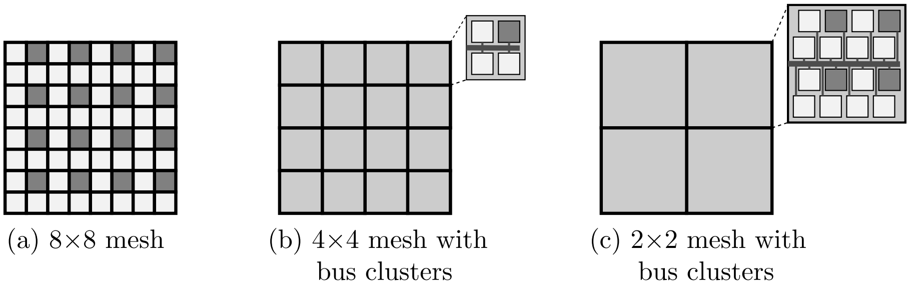
This chapter addresses the problem of analytical modeling of CMP architectures. The contribution in this field is the analytical method for estimating performance of large-scale hierarchical CMPs with hundreds and thousands of cores, published in [102]. Specifically, it was shown that contention of the interconnect network has a major significance when analyzing CMP performance. The proposed approach not only captures the behavior of different core architectures and cache hierarchies, but also estimates the traffic and contention of various interconnect topologies.

To model the contention in the interconnect network, the cyclic dependency between latency and memory traffic is formulated as a system of non-linear equations. Two numerical methods are proposed to resolve it efficiently: fixed-point iteration and bisection. These methods can use any black-box analytical model for latency, making the approach flexible to incorporate new interconnect models. As the performance of a CMP is highly determined by the executed applications, the proposed method can be parametrized with various workload models.

The application of this model to the exploration of large-scale CMPs is discussed in the next chapter.

## 4.1 The importance of contention: an example

Consider a CMP with 48 cores and 16 shared on-chip cache modules. Figure 4.1 presents three (of the many) possible architectures with such parameters. One of the architectures has an 8×8 structure of regular tiles connected with a mesh (Figure 4.1(a)). The cores and caches are shown as light and dark squares, respectively. Solid lines represent the mesh links.


(a) 8×8 mesh

(b) 4×4 mesh with bus clusters

(c) 2×2 mesh with bus clusters

Figure 4.1: Possible architectures for a 48-core CMP.

Table 4.1: Performance of architectures in Figure 4.1.

| **Architecture** | $L_{est}$ | $\theta_{est}$ | $L_{sim}$ | $\theta_{sim}$ |
|---|---|---|---|---|
| (a) | 11.17 | 8.23 | 11.26 | 8.16 |
| (b) | 10.12 | 9.04 | 10.40 | 8.81 |
| (c) | 9.95 | 9.19 | 16.69 | 5.58 |

To take advantage of the locality of memory accesses, several cores and caches can be grouped in a cluster and communicate via the local interconnect. For instance, clusters with bus interconnects were shown to notably improve the average communication latency [43]. This fact encourages the exploration of hierarchical interconnects. Figure 4.1(b) describes the CMP organization with 16 clusters, each one having three cores, one cache and a shared bus. The clusters communicate via the top-level 4×4 mesh. Another option is to increase the cluster size up to 16 components (12 cores and 4 caches) and decrease the dimensions of the top-level mesh (Figure 4.1(c)).

One of the problems of architectural exploration is to select the configuration with the best performance. We first estimated the throughput of each configuration using only the static (hop-count) latency of the network, i.e., assuming no contention. In this experiment we assumed the ideal throughput of cores (under the assumption of zero-latency memory) to be 2.0 `IPC` (instructions per cycle), and the number of memory references per instruction to be 0.5. The values of static latency (in cycles) and the estimated throughput (in `IPC`) are displayed in the columns $L_{est}$ and $\theta_{est}$ of Table 4.1. Hierarchical architectures show a higher performance due to the exploited locality: configuration (c) has the largest size of local cache (per cluster), hence the increased local hit ratio. Therefore, (c) shows the highest estimated throughput.

However, this conclusion is incorrect when network contention is taken into account. Simulation reveals rather distinct performance numbers, reported in the $L_{sim}$ and $\theta_{sim}$ columns of Table 4.1. For configurations (a) and (b), the estimated throughput with no contention is close to the one reported by simulation. However, the performance of configuration (c) drops by about 40%. In fact, simulation concludes that (c) is the worst in terms of performance.

The reason of this significant discrepancy is the network contention. It occurs because of the competition between memory requests for the shared resources of the interconnect. This results in longer latencies, decreasing the overall performance of the system. In this example, configuration (b), which incorporates hierarchy at some extent, is the one with the highest throughput and represents the best architectural trade-off between cache locality and communication parallelism.

## 4.2 Analytical performance model

This section introduces the models for the evaluation of CMP performance and power. First, we explain the assumptions and input parameters of the models. Next, the equation for modeling static latency is presented. This equation is then extended to consider the contention component of communication. Finally, the throughput model is next discussed and the formula for the memory traffic rate is derived, emphasizing the cyclic dependency between traffic and latency.

### Assumptions and input parameters of the models

This work focuses on systems with two-level hierarchical interconnect fabrics. However, the approach can be applied for an arbitrary number of hierarchical levels, including the particular case of flat interconnects. Several components are grouped into a cluster: cores, components of the memory subsystem and the local interconnect. The top-level interconnect provides communication between the clusters and access to the off-chip memory (Figure 1.3(b)).

The system has $N$ cores in total with parametrizeable architectures: in-order, out-of-order, single- or multi-threaded. The workload model assumes that every core is executing an application, characterized by two parameters. $\mathtt{IPC}_0$ is the ideal throughput of the core for the application, i.e., the amount of instructions per cycle executed by the core, assuming zero-latency memory. $\mathtt{MPI}$ is the average number of memory references generated per instruction.

Without loss of generality, we assume that the memory subsystem has four hierarchy levels. Every core has a private *L1* cache and possibly, a private *L2* cache of larger size but higher latency. The clusters incorporate modules of a distributed *L3* cache, shared by all cores. The off-chip memory is accessible via a set of memory controllers. The latencies of the caches and off-chip memory are parameters of the model.

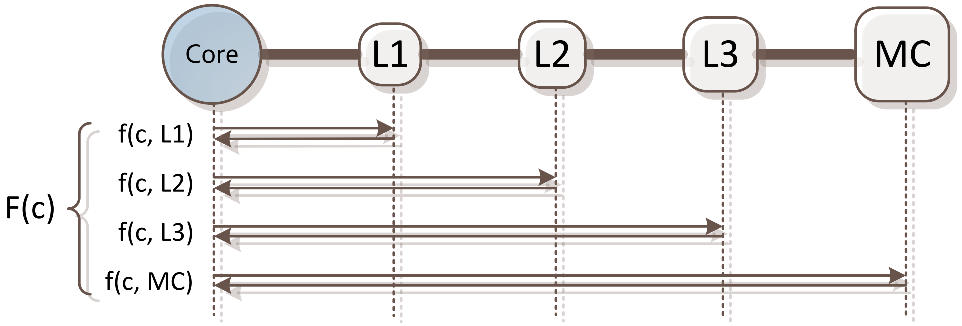


Figure 4.2: Memory flows $F(c)$ of a core.

The term *memory flow* is used to denote a feasible communication between a core and a component of the memory subsystem. For example, each core may access its own *L1* or *L2* caches, or any of the *L3* modules or *MC*s. In this work we only consider the request and reply flows between the cores and the memories. However, a specific coherence protocol can be modeled by adding the synchronization traffic.

Figure 4.2 gives an example of the set of all possible memory flows for a core $c$, which is denoted as $F(c)$. In this example, a pair of the request and reply flows between $c$ and a particular level of memory hierarchy, $Lx$, is denoted as $f(c, Lx)$, e.g. the flows between $c$ and *L1* are denoted as $f(c, L1)$.

Every flow $f \in F(c)$ is realizable with probability $p_f$, that defines the probability for $c$ to request data from a certain memory component. In our work we calculate these probabilities using a *model of cache miss behavior* for the workload in consideration, which represents the dependency between miss ratio and cache size. A *power law* model was proven to be a good approximation [63]:

$$Miss(S) = \kappa S^{-\alpha}, \tag{4.1}$$

where $S$ is the cache size, and $\kappa$, $\alpha$ are the model parameters (Fig. 4.3). Alternatively, the miss model can be precharacterized using simulation and specified as a set of cache-size / miss-ratio points and interpolation between them.

Since *L3* is a distributed cache, its access latency depends on the cluster where the requested data is stored. In this work the probability to find the data in a particular cluster is assumed to be inversely proportional to the distance between the requesting core and the cluster. However, the method can be parameterized with any other model for distributed cache.

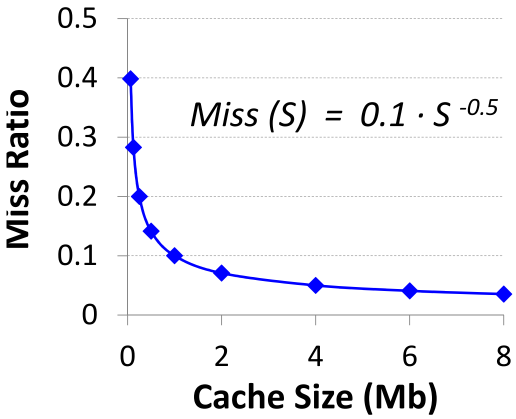


Figure 4.3: Power-law cache miss model.

## Static latency

In this section we describe how to calculate the average static latency of memory accesses for a core $c$ in the presence of memory hierarchy. Given the probability $p_f$ for each particular flow $f \in F(c)$ and its static latency $L_f$, the *average static latency* $L_c^{st}$ is:

$$L_c^{st} = \sum_{f \in F(c)} p_f L_f. \tag{4.2}$$

Since requests to *L3* and *MC* are sent via the communication network, its delay must also be considered. This delay is defined using the routing function $\mathcal{R} : f \rightarrow \pi(f)$, that for any flow $f$ returns its routing path $\pi(f)$. In this work we consider the *XY-routing* function [39], however any deterministic or even adaptive routing can be used, specifying the probabilities for certain paths. The total latency to access an *L3* instance is the sum of the network traversal latency along the path $\pi(f)$ and the *L3* latency. The total latency of the off-chip memory accesses is calculated likewise.

The flow probabilities $p_f$ are obtained using the miss ratio model, *Miss(S)*, given by (4.1). Assuming the sizes $S_{L1}$, $S_{L2}$ of the two low-level caches, the probabilities to access them are:

$$\begin{aligned} p_{L1} &= 1 - \mathit{Miss}(S_{L1}), \\ p_{L2} &= (1 - p_{L1})(1 - \mathit{Miss}(S_{L2})). \end{aligned}$$

As *L3* is shared, the miss ratio is defined by the effective *L3* size, $S_{L3}^{eff}$, seen by each core [12]. To estimate $S_{L3}^{eff}$ we use the concept of the average number of cores

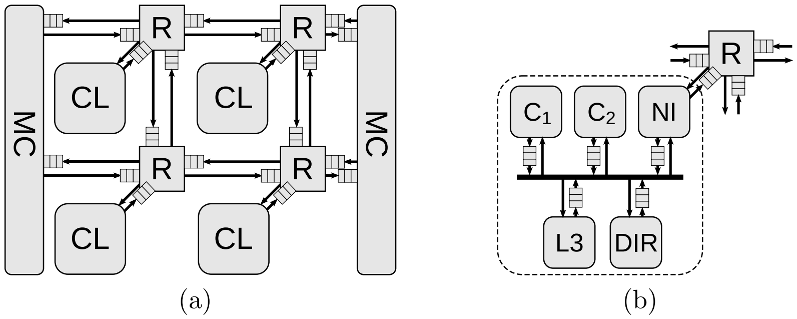


Figure 4.4: Queueing model for (a) mesh and (b) cluster.

sharing each line, as proposed by [12]. The probability to access *L3* is then:

$$p_{\mathit{L3}} = (1 - p_{\mathit{L1}})(1 - p_{\mathit{L2}})(1 - \mathit{Miss}(S^{\mathit{eff}}_{\mathit{L3}})).$$

Finally, $p_{\mathit{L3}}$ should be multiplied by the probability to find the data in a particular *L3* instance (cluster). A similar strategy is used to calculate the probabilities of flows to every memory controller.

## Queueing model for the on-chip interconnect

Equation (4.2) describes the static latency of memory accesses. Another important part of the communication delay is the dynamic or contention latency [39]. Contention happens in the interconnect fabrics when several packets compete for the same shared resource, such as a bus or an NoC link. This results in additional delays experienced by packets in the buffers distributed over the on-chip interconnect. One of the approaches to estimate the contention delays is to model the CMP as a system of queues and apply queuing theory to calculate the buffer delays.

In this work two-level hierarchical interconnects are assumed, with a 2D mesh at the top level. At the lowest (cluster) level, buses, uni- and bi-directional rings are used for the interconnection. Figure 4.4(a) shows the queueing representation of the top-level mesh. The mesh routers (*R*) have up to five input-buffered ports to store the incoming flits. The primary ports of the routers are connected to the clusters (*CL*), which in case of a flat CMP organization may consist of one device (e.g. a core with private caches in Figure 1.3(a)). Figure 4.4(b) presents an example of queueing model for a bus-based cluster, corresponding to one tile of the hierarchical CMP. This cluster consists of five devices, communicating via a shared bus: two cores with private caches, an instance of an *L3* shared cache, a directory and a network interface. Every device has a buffer to store the requests to the bus. To distribute the off-chip memory traffic uniformly over the mesh and avoid high contention of certain routers, we assume that memory controllers have multiple connections to the mesh, as shown in Figure 4.4(a).

## Total memory latency

The average total latency for core $c$, $L_c$, is calculated by adding the static latency ($L_c^{st}$) and the queue delays along the communication paths ($w_q$). Hence, given the paths $\pi(f)$ for every flow $f$, we extend equation (4.2) accordingly:

$$L_c = L_c^{st} + \sum_{f \in F(c)} \left( p_f \sum_{q \in \pi(f)} w_q \right). \tag{4.3}$$

To find the values for $w_q$, an analytical model for the on-chip interconnects can be used. In this work we apply the model from [103], which offers a convenient definition of queue delays via the injection rates in closed form. The method calculates the probabilities of the packets coming from different bus (or router) inputs to move towards the same output. It represents an efficient generalization of the M/G/1 queueing model [81]. The efficiency of this model is essential for our iterative procedure, described in Sect. 4.3. Another advantage of this model is the capability to deal with a variety of interconnect types, such as buses and router-based topologies.

Given the vector of injection rates, $\overline{\lambda} \in \mathbb{R}^N$, the model in [103] proposes to express queue delays in the form of a system of equations with a matrix $W$:

$$\overline{w}_q = W(\overline{\lambda}), \tag{4.4}$$

where $\overline{w}_q$ is the vector of delays for all queues of the interconnect. The exact form of the matrix $W$ is given by the expressions (5) and (18) in [103]. What only remains is to compute the rates $\overline{\lambda}$, which is covered in the next section.

## Throughput model

The throughput of a CMP and the traffic in the interconnect are closely related. To derive the exact dependencies, we start with the performance model for a single core, given in [64]. For a core with the average rate of accesses to remote memory (*RemRate*), and the cost of an access (*RemCost*), the average number of cycles for executing an instruction, `CPI`, is:

$$\texttt{CPI} = \texttt{CPI}_0 + \mathit{RemRate} \cdot \mathit{RemCost}, \tag{4.5}$$

where $\mathtt{CPI}_0 = 1/\mathtt{IPC}_0$ is the ideal CPI under the assumption of zero-latency memory. For a *single-threaded in-order* core, the cost of a remote access is the average latency, given by (4.3), and the remote rate is given by the MPI value. As throughput is typically measured in IPC, the reciprocal of CPI, from (4.5) we obtain:

$$\theta_c = \frac{1}{\mathtt{CPI}} = \frac{1}{\frac{1}{\mathtt{IPC}_0} + \mathtt{MPI} \cdot L_c}. \tag{4.6}$$

The throughput of the entire CMP, $\Theta$, is then calculated as the total performance of individual cores: $\Theta = \sum_c \theta_c$.

The rate of memory accesses, $\lambda_c$, is the probability for a core to issue a remote memory request per cycle. $\lambda_c$ is proportional to the core throughput and the MPI:

$$\lambda_c = \theta_c \cdot \mathtt{MPI} = \frac{\mathtt{MPI}}{\frac{1}{\mathtt{IPC}_0} + \mathtt{MPI} \cdot L_c}. \tag{4.7}$$

## The cyclic dependency between memory latency and traffic

One can observe an intuitive result: the latency of the memory requests traversing the interconnect depends on the injection rate of requests, due to the network contention. On the other hand, the request rate is determined by the latency, as no new memory requests are issued if the execution of cores stalls due to the absence of data. These facts emphasize the *cyclic dependency between the latency and rate of memory requests.*

More formally, in order to calculate the buffer delays, equation (4.4) requires the injection rates, $\overline{\lambda}$, at every input (source) of the interconnect, while equation (4.7) gives the rates of request generation per core. Note that the injection rates in a *flat* interconnect are directly defined by the core rates: for a CMP with $N$ cores, $\overline{\lambda} = \{\lambda_1, .., \lambda_N\}$. In case of a *hierarchical* interconnect fabric, the core rates will correspond to the injection rates at the sources of the cluster-level interconnects, such as the bus in Figure 2.12(b). The injection rates to the top-level mesh can be calculated, given the fraction of inter-cluster traffic. The latter is defined by the probabilities of access to the *L3* and the off-chip memory, discussed in section 4.2. Below we directly consider the dependency of memory latency on the core rates.

We observe the following system of dependencies:

$$\forall c = 1, .., N : \begin{cases} L_c = L(\overline{\lambda}) \\ \lambda_c = \lambda(L_c), \end{cases} \tag{4.8}$$

where $L_c$ is the average memory latency for core $c$, defined by (4.3) that depends on the injection rates of all cores, $\overline{\lambda}$, due to definition of queue delays $\overline{w}_q$ in (4.4).

$\lambda_c$ is the injection rate for core $c$, thus depends only on the latency $L_c$ (4.7). *The dependencies* (4.3), (4.4) *and* (4.7) *create a system of equations* with respect to the vectors of variables $\overline{L}$, $\overline{\lambda}$, and $\overline{w}_q$. In Section 4.3 we describe the methods to resolve this system.

## 4.3 Analytical methods for latency estimation

There are several methods to solve non-linear systems of equations [30]. For example, the solution of a non-linear system (4.8) can be found using one of the Newton-based methods, typically available in a generic solver, such as MATLAB [5]. The downside of this approach is the need for the calculation of function derivatives, which is a computationally expensive process. Apart from that, this approach only works for analytical models that can be represented with closed-form equations.

In this section we propose two numerical methods to efficiently resolve the cyclic dependency (4.8). The first method, *fixed-point iteration*, delivers the solution with a desired precision in case of convergence and can be applied to arbitrary configurations. The second approach, based on *bisection*, always converges for our problem but finds an approximate solution. However, it resulted to be a good approximation for tiled homogeneous CMPs (see Section 4.3).

The subgradient method [21] was also considered as an alternative to bisection. Although it is more accurate, it is also significantly slower. In this work we focus on the first two methods, since their performance and quality represent the best compromise for the evaluation of homogeneous architectures.

### Fixed-point iteration

The algorithm proposed in this section is a popular numerical method for solving systems of nonlinear equations [30]. While the theoretical speed of convergence of this method is relatively slow, it performs well in practice due to its low cost for a single iteration. Given a system of equations in the form:

$$\overline{x} = F(\overline{x}), \tag{4.9}$$

where $\overline{x}$ is the vector of variables and $F$ is the system matrix, and an initial guess $\overline{x}_0$, the following iterative procedure can be used to find a solution $\overline{x}^*$ (fixed point) of the system:

$$\overline{x}_{n+1} = F(\overline{x}_n),\ n \geq 0.$$

In our setting, $\overline{x}$ is composed of the variables $\{\overline{L}, \overline{\lambda}, \overline{w}_q\}$ and matrix $F$ is defined by the right-hand terms of the equations (4.3), (4.4) and (4.7). For the initial guess,

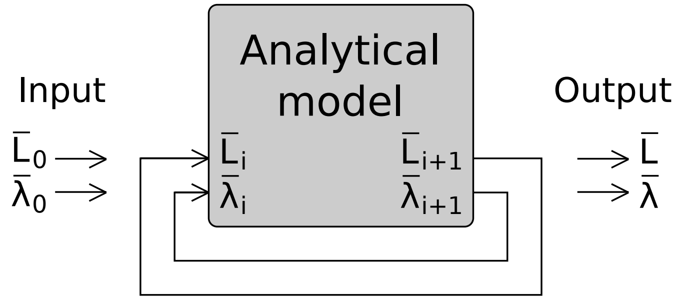


Figure 4.5: Iterative calculation of latency and traffic, using analytical model.

$\overline{x}_0$, we use static latencies (4.2) and compute other values using the same equations: $\overline{L}_0 = \overline{L}^{st}, \overline{\lambda}_0 = \lambda(\overline{L}_0), \overline{w}_{q,0} = W(\overline{\lambda}_0)$.

The benefit of the proposed method is that it does not require closed-form analytical expressions for latencies. Furthermore, any black-box model for the dependency of interconnect latency on injection rate can be used. The method hence maintains the modular structure of hierarchical interconnects and permits plugging independent models for different topologies, such as bus (cluster-level) and mesh (top-level). This makes the approach a valid tool for future interconnect modeling.

Figure 4.5 illustrates the iterative procedure. The initial values for vectors of variables $\overline{L}_0$ and $\overline{\lambda}_0$ are obtained with an approximation by static latency. The analytical model (equations (4.7) and (4.3)) is used to improve the estimations of variables $\overline{L}_i$ and $\overline{\lambda}_i$ at every iteration. The iterative process stops and returns final values once the required precision for latency and traffic is reached.

As a numerical method, fixed-point iteration is subject to convergence issues. For a system in the form (4.9), the sufficient condition for convergence is [30]:

$$\sum_i \left| \frac{\partial F}{\partial x_i} \right| < 1.$$

In our case, this requires the latency to grow slowly with the injection rate, and vice versa. This condition holds for the communication networks that perform far from their saturation throughput (for instance, see chapter 23 in [39]). Although this is a sufficient condition, it is not necessary for convergence. In practice we observe that for the majority of configurations the iterative procedure converges.

A second issue of the fixed-point iteration is due to the analytical models based on queueing theory: the queueing models work under the assumption of the system being in the steady-state [81]. This means that for any router with service time $T$ and the sum of arrival rates to its inputs $\lambda$, the following condition must hold: $\lambda T < 1$. In other words, there should be no unbounded packet accumulation in the input queues of the router. Unfortunately, this requirement may be not satisfied by the initial solution. From (4.7) we know that the latency $L_c$ and the memory access rate $\lambda_c$ are inversely proportional. Since static latency is taken as the initial value of $L_c$, the total latency may be highly underestimated for the configurations with high contention. As a result, the initial value of $\lambda_c$ will be overestimated and may violate the steady-state condition.

To handle this situation as well as the configurations for which the fixed-point iteration diverges, we propose a method based on the bisection search of $\lambda_c$, to find a reasonable and fast approximation to the solution.

## Bisection search for traffic rate

The advantage of the bisection method is that it always converges for our model (due to the intermediate value theorem [30]). Since every core generates traffic at certain rate, $\lambda_c$, *multidimensional bisection* [135] can be applied to find the exact rates. However, a good approximation to the exact rates can be obtained by using the less complex *unidimensional bisection*. By simulation we observed that the traffic rates of the cores of tiled CMPs with homogeneous clusters change proportionally to their estimates, obtained by the static latency. Hence, we initialize the vector of injection rates $\overline{\lambda}$ with the values estimated by static latency, and on every bisection step adjust all rates in the same proportion.

To introduce the bisection more formally, let us rewrite equation (4.7) by isolating $L_c$, and using the star symbol to distinguish it from the latency in (4.3):

$$L_c^*(\lambda_c) = \frac{1}{\lambda_c} - \frac{1}{\texttt{MPI} \cdot \texttt{IPC}_0}. \tag{4.10}$$

From (4.3) and (4.10) we define the average latencies $L(\overline{\lambda})$ and $L^*(\overline{\lambda})$ as the functions of the vector $\overline{\lambda}$:

$$\begin{aligned} L(\overline{\lambda}) &= \frac{1}{N} \sum_{c=1}^{N} L_c(\overline{\lambda}), \\ L^*(\overline{\lambda}) &= \frac{1}{N} \sum_{c=1}^{N} L_c^*(\lambda_c). \end{aligned}$$

Finally we introduce the *latency difference function*, $\mathrm{F}(\overline{\lambda})$:

$$F(\overline{\lambda}) = L(\overline{\lambda}) - L^*(\overline{\lambda}).$$

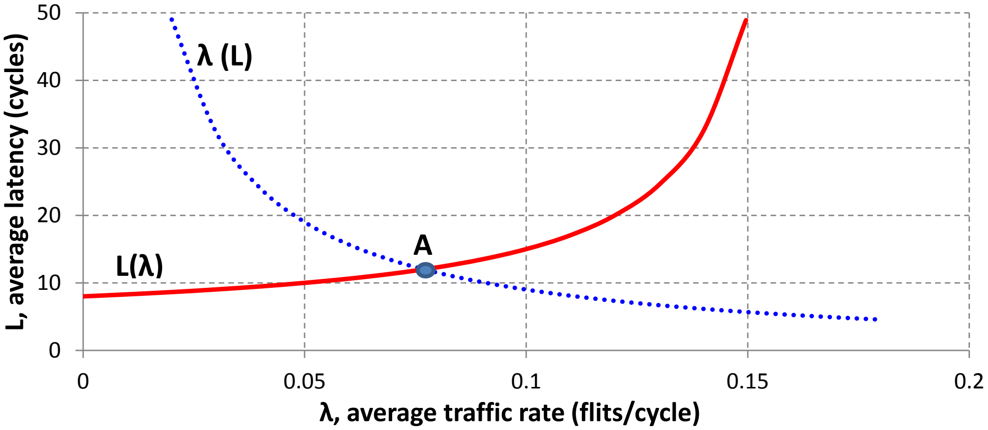


Figure 4.6: Behavior of the latency functions $L(\overline{\lambda})$ and $L^*(\overline{\lambda})$.

Figure 4.6 shows the typical behavior of these functions, emphasizing the cyclic dependency (4.8). To depict a 2D view of this behavior, we plot $L(\overline{\lambda})$ and $L^*(\overline{\lambda})$ as a function of the average rate $\Lambda = \frac{1}{N}\sum_{c=1}^{N}\lambda_c$. The curve $L^*(\overline{\lambda})$ shows that the average rate of memory requests increases as the latency decreases. On the contrary, $L(\overline{\lambda})$ shows that the average latency increases with the injection rate. The real values for latency and traffic are defined by the intersection point A of these curves, that can be found as a root of $F(\overline{\lambda})$. Hence, we use the bisection as a root-finding method, that does not require the exact knowledge of the function $F(\overline{\lambda})$ and can be used with any black-box analytical model for latency.

Bisection searches for $\overline{\lambda}$ that satisfies the condition $|F(\overline{\lambda})| < \epsilon$, where $\epsilon$ is the solution tolerance. The initial range for $\overline{\lambda}$ is limited by the traffic, obtained with static latency: $\overline{\lambda}_{min} = \overline{0}$, $\overline{\lambda}_{max} = \overline{\lambda}(L_c^{st})$. Assuming the proportionality in variation of the individual components of $\overline{\lambda}$, all components are updated simultaneously. For any pair of consecutive iterations $i$ and $i+1$, either $\overline{\lambda}_{min}^{i+1} = \overline{\lambda}^{i}$ when $F(\overline{\lambda}^{i}) < 0$, or $\overline{\lambda}_{max}^{i+1} = \overline{\lambda}^{i}$ when $F(\overline{\lambda}^{i}) > 0$. The iteration is continued until the required tolerance for $F(\overline{\lambda})$ or $\overline{\lambda}$ is met [30].

## 4.4 Extensions of the model

This section describes two extensions of the performance model from Section 4.2. First, it is shown how to adapt the model to capture the behavior of multithreaded and out-of-order core architectures. Afterwards, an extension to model the power consumption is presented.

## Multithreaded and out-of-order cores

Equation (4.7) can be extended for the case of multithreaded and out-of-order cores. A *multithreaded* core can be modeled as a group of single-threaded cores. The latency for each thread remains $L_c$, but the total memory rate becomes $\lambda_c^{mt} = M\lambda_c$, where $M$ is the number of threads.

The difference in modeling an *out-of-order* core is that the remote memory access does not force the core to stall, hence the effective remote latency $L_c$ decreases [64]. Following the techniques in [53], we make two adjustments to capture this behavior in our model. First, the authors in [53] consider the short latencies of *L1* and *L2* to be hidden by instruction reordering. In our model, it can be modeled by excluding the flows between the core and the first two levels of local cache from the memory-flow set: $F_{OoO}(c) = F(c) \setminus \{f(c, \mathit{L1}), f(c, \mathit{L2})\}$ (see Fig. 4.2). Equation (4.3) is used with the new $F_{OoO}(c)$ to calculate the average total latency.

The second adjustment to the model is done by considering *memory-level parallelism* of the workload (`MLP`), which is the average number of memory requests issued in parallel [35]. When a core issues several requests in parallel, the latency penalty is amortized due to overlapping of the requests. We capture this fact by adjusting the `MPI` value to reduce the fraction of memory requests per instruction (and hence the penalty):

$$\mathtt{MPI}_{OoO} = \frac{\mathtt{MPI}}{\mathtt{MLP}}.$$

This new value is used with (4.6) and (4.7) to obtain the throughput and traffic of the out-of-order cores.

## Power model

The analytical model can also be extended to estimate power consumption. In this work we model power using a first-order approximation of leakage and dynamic power for individual components, such as cores, caches and interconnects. Leakage power of component $c$ is proportional to the unit leakage $p_{leak,c}$ and the area $A_c$:

$$P_{leak,c} = p_{leak,c} \cdot A_c.$$

Dynamic power is primarily defined by the utilization of components. For cores, it is proportional to the throughput of the core $\theta_c$, the energy of executing single instruction $E_{inst,c}$, and the core frequency $\mathit{Freq}_c$:

$$P_{dyn,c} = E_{inst,c} \cdot \theta_c \cdot \mathit{Freq}_c.$$

Similarly, the cache power depends on the number of accesses to the cache per cycle (e.g. traffic), $\Lambda_c$, and the energy per access $E_{acc,c}$:

$$P_{dyn,c} = E_{acc,c} \cdot \Lambda_c \cdot \mathit{Freq}_c.$$

Dynamic power for the components of on-chip interconnect, such as routers and links, is proportional to the traffic through the component, $\Lambda_c$, and the energy per flit transmission $E_{\mathit{flit},c}$:

$$P_{dyn,c} = E_{\mathit{flit},c} \cdot \Lambda_c \cdot \mathit{Freq}_c.$$

Here the areas $A_c$ and the performance metrics $\theta_c$ and $\Lambda_c$ of the components are calculated by the performance model. Frequency is a parameter of exploration. To find the power and energy coefficients we employ the data obtained from [117] for the cores, CACTI 5.3 model [2] for caches and Orion 2.0 model [76] for on-chip routers and links.

## 4.5 Experimental results

In this section we describe several experiments used to validate the proposed analytical method for efficiency and quality. Validation is performed with respect to simulation. Next subsections describe our simulation environment and the experiments.

### Simulation environment

A cycle-accurate simulator for hierarchical CMP interconnect networks has been designed [45] using BookSim 2.0 [39] as underlying infrastructure. The simulator can model the contention of the interconnect network at flit level.

Three enhancements were made to BookSim. First, the probabilistic traffic injection patterns were replaced by *state-machine models for cores, caches and memory controllers.* The cores inject memory requests according to parameters characterizing the average workload of the system. Cores are stalled when they are waiting for responses from memory. Both in-order and out-of-order cores can be modeled. Memories accept requests from cores and send replies after a predefined latency.

*Support for hierarchical topologies* was added, thus enabling the simulation of multi-level interconnect networks with an arbitrary number of levels. Finally, models for *bus and multibus* topologies were also created.

Each simulation was run long enough to obtain a 2% relative error (the same value used for the analytical model) with a 95% confidence degree. The 95% confidence interval is guaranteed using the batch means method [55].

Table 4.2: Performance comparison of analytical methods.

| **Test** | **Mesh** | **Content. latency** | **Num. of var./eqn.** | **Runtime (sec)** | | |
|---|---|---|---|---|---|---|
| | | | | MATLAB | Fixed-point | Bisection |
| T1 | $2 \times 2$ | 5% | 236 | 0.023 | 0.001 | 0.001 |
| T2 | $4 \times 4$ | 13% | 1224 | 1.412 | 0.001 | 0.002 |
| T3 | $6 \times 6$ | 8% | 3108 | 30.831 | 0.002 | 0.003 |
| T4 | $8 \times 8$ | 12% | 6128 | 408.539 | 0.006 | 0.010 |
| T5 | $10 \times 10$ | 23% | 10620 | $>$ 1h | 0.010 | 0.012 |
| T6 | $10 \times 10$ | 46% | 10620 | $>$ 1h | 0.022 | 0.015 |
| T7 | $10 \times 10$ | 55% | 10620 | $>$ 1h | NA | 0.016 |

## Efficiency of the model

In this section we compare the efficiency of the three analytical methods for resolving the cyclic dependency presented in Section 4.3: solver (MATLAB), fixed-point iteration (FP) and bisection (BS). We generated a set of CMPs having flat mesh topologies with various dimensions and contention degrees. The reason to select flat interconnects is to demonstrate that even for rather simple architectures the obtained system of equations is hard to be tackled in a straightforward way.

The test cases and the results are summarized in Table 4.2. The first three columns show the test name, mesh dimension and the ratio of contention latency with respect to the average total latency. The fourth column represents the number of variables and equations in the obtained system. The fifth column shows the time required to find a solution using the general nonlinear solver provided by MATLAB. The last two columns show the time consumed by the FP and BS methods. For each test case the three methods converged to the same solution, within the given tolerance region of 2%.

Test cases T1 to T5 accentuate how the MATLAB time grows with the mesh size. The solution of T5-T7 could not been found within an hour. Clearly, this straightforward method is not acceptable for efficient exploration of CMPs.

The purpose of test cases T6 and T7 is to compare the FP and BS methods. T6 has higher contention than T5. As a result, FP takes more time to converge than BS. T7 has even more contention, resulting in a violation of the steady-state assumption for the queueing model (see Section 4.3). Hence, FP can not be used in this case, so BS is the only option.

We observed that FP typically outperforms BS when the contention component of latency is moderate, i.e. does not exceed about 30-40% of the total latency. Hence we choose to run FP first and use BS only when the former method fails.

Table 4.3: Parameters of the exploration space $\mathbb{S}_1$.

| **Parameter** | $\mathbb{S}_1$ **values** |
|---|---|
| Chip area | 350 mm$^2$ |
| Core area | 1.25 mm$^2$ |
| Core ideal `IPC` | 2.0 |
| L1 size | 64, 128 Kb |
| L2 size | 128 Kb to 1 Mb |
| Mesh dimensions | 2×2 to 10×10 |
| Off-chip memory latency | 100 cycles |
| Type of cluster interconnect | Bus, uni-ring, bi-ring |
| Interconnect link width | 256 bits |
| Workload `MPI` | 0.25 |
| Workload `MLP` | 1.25 |

Table 4.4: Cache area-performance model.

| **Cache size** | 64K | 128K | 256K | 512K | 1M | 2M | 4M | 8M |
|---|---|---|---|---|---|---|---|---|
| **Area (*mm*$^2$)** | 0.063 | 0.125 | 0.25 | 0.5 | 1.0 | 2.0 | 4.0 | 8.0 |
| **Latency (*cycles*)** | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

## Quality of the model

To validate the quality of the model in estimating performance, we carry out a CMP design space exploration experiment. For every configuration of the design space we obtain the throughput using both analytical modeling and simulation. The search space in this experiment is made intentionally small in order to allow an exhaustive simulation of all configurations. With this experiment we demonstrate that the analytical model selects a set of best-throughput architectures very similar to the simulator, but in much shorter time.

The parameters of the exploration space $\mathbb{S}_1$ for this experiment are listed in Table 4.3. The value chosen for chip area is typical for CMP exploration studies [107, 32]. The estimates for the area of each component are derived from the parameters of the Intel Core 2 Duo E6400 processor [6]. The ideal `IPC` of core is obtained from [117]. We also assume that the core is out-of-order (OoO).

We scale the core area and memory density down to $16nm$ to allow hundreds of cores within the chip area. Table 4.4 shows the area-performance model for cache memory.

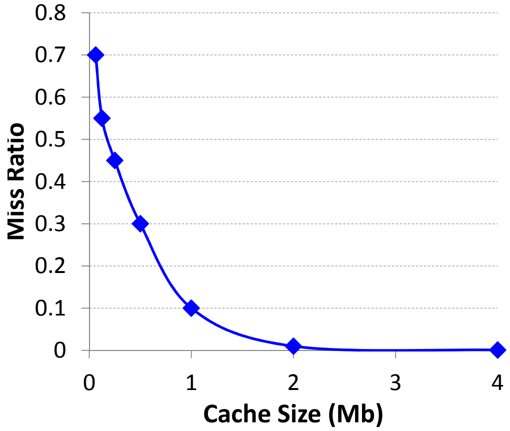


Figure 4.7: Cache miss model of *soplex* application.

Without loss of generality, we assume all cores to be running the *soplex* application from SPEC CPU2006. Figure 4.7 depicts the miss ratio produced by the application as a function of the cache size, obtained through simulation.

The number of cores and cache sizes are varied to explore the trade-off between computing units and on-chip memory. Three types of local interconnects are considered inside the clusters: buses, uni- and bi-directional rings. The exploration of the mesh dimensions compromises the number of clusters and processors per cluster. The maximum power of all configurations is limited to $130W$.

Given these parameters, our framework generates 2095 feasible configurations. The simulation of all configurations takes about 18 hours, while the analytical model takes 48 seconds, delivering more than a 1300x speedup. The best configuration obtained by simulation has a throughput of 72.07 IPC. This architecture has a 2×2 mesh (4 clusters with bi-ring interconnect, 21 cores per cluster), a total of 84 cores with 128Kb *L1*, 1Mb *L2* private caches and 150Mb shared *L3* cache.

In Fig. 4.8 we sort the configurations by throughput horizontally, as estimated by simulation. One can see that the analytical model tracks the simulation curve with reasonable accuracy. The analytical model underestimates contention and, for this reason, the discrepancy with simulation increases with higher values of contention. Configurations with similar throughput may have different contention values, hence the noisy behavior of the modeling curve. The average absolute error in throughput is 7.7%, which corresponds to the error reported by the latency model [103].

The worst-case error for all nearly-optimal configurations does not exceed 10% (low-error zone), although it grows as we move away from the optimum. This is explained by the fact that architectures with balanced interconnects tend to have higher throughput and less contention. The precision of the analytical model drops

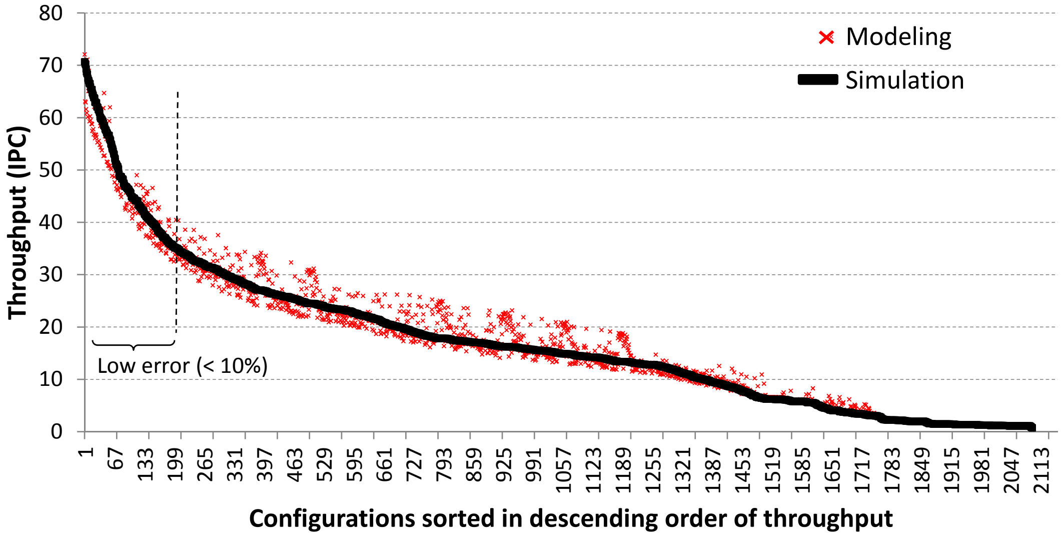


Figure 4.8: Throughput comparison for analytical model and simulation.

when the contention increases, hence the error grows for configurations which are far from the optimum. Since the design exploration problem is aimed at selecting configurations with the highest throughput, this loss of precision is not critical. If required, the quality of the model can be further improved by considering alternative analytical models in conjunction with the iterative techniques proposed in this work.

What really matters for exploration are the relative, rather than the absolute values of throughput. When exploring a huge design space we would like to discard suboptimal architectures and keep a moderate subset of promising solutions. These configurations can be further simulated to select the best one. Hence, we are interested in comparing the *order of configurations by the highest throughput*, as delivered by the analytical model and the simulation. Here is where our technique demonstrates very accurate results: Figure 4.9 shows the comparison for the best-throughput order. To make the picture illustrative, we only consider the 50 best configurations, although this tendency is maintained for the whole set of configurations. The horizontal axis specifies the number $N$ of best configurations chosen *by simulation*. The vertical axis indicates the minimum number of best configurations chosen *by the analytical model* that include the $N$ best ones by simulation. For example, the point with coordinates (1;1) means that the best configuration by simulation is also the best one by modeling. The rightmost point on the plot (50;60) means that the 60 best configurations by modeling include the 50 best by simulation. This is actually a very accurate result for the analytical model, considering that two thousand configurations are compared.

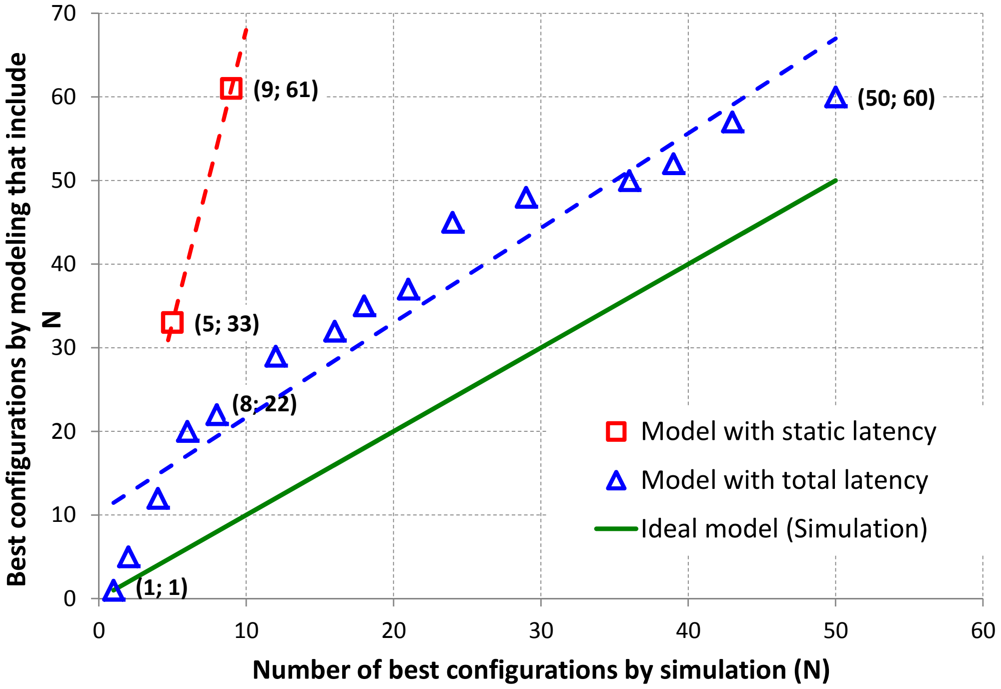


Figure 4.9: Order comparison for analytical model and simulation.

We also demonstrate that estimations based on static latency deliver a deficient order. It biases the exploration towards configurations with large bus-based clusters, given the fact that the long contention latency in the buses is not considered. The point (5;33) in Figure 4.9 means that the fifth best configuration is on the 33rd position when not considering contention. Note that modeling with only static latency fails to discover the four best configurations due to the power constraint. In the absence of contention, the latency (4.3) is underestimated, while the throughput (4.6) and traffic (4.7) are overestimated, resulting in overestimation of power (see Sect. 4.4). The problem could be avoided by setting the power constraint to a higher value. However, a higher power limit would bring new configurations into consideration, aggravating the deviations in the order with respect to simulation.

## Scalability of the model

To investigate the scalability of the analytical model, we generated several CMPs with mesh-of-buses topology and similar structure. Each cluster contains four components (three cores and one cache module) and a bus interconnect. The top-level mesh dimensions are varied from 2×2 to 16×16, producing CMPs with 16 to 1024 components. For each test case we executed both fixed-point and bisection and com-

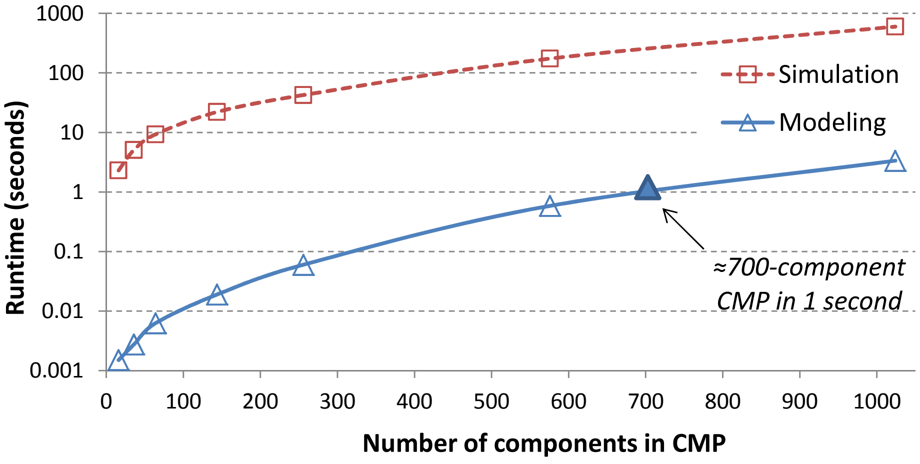


Figure 4.10: Performance comparison of the analytical model and simulation.

pared the average runtime value of these two methods with simulation. Figure 4.10 shows the comparative results.

Our probabilistic simulator demonstrates very good performance. Simulation of the 16-component CMP takes just 2.5 seconds, and for the 1024-component CMP about 600 seconds. However, the analytical model yet brings about three orders of magnitude improvement in efficiency. For the 16- and 1024-component CMPs modeling took only 0.002 seconds and 3.3 seconds respectively. In one second our method handles a CMP with nearly 700 components. This result justifies high scalability of the proposed method and its ability to efficiently explore architectures with many hundreds of cores.

## 4.6 Conclusions

Analytical models for CMP performance are crucial to make the architectural exploration possible. This work shows that such models need to incorporate the contention factor in order to adequately estimate performance. We have presented an analytical method to model contention of hierarchical interconnects, by resolving the cyclic dependency between the memory latency and traffic.

The important conclusions drawn from the experiments are:

- The proposed numerical methods for resolving the cyclic dependency (4.8) demonstrated several orders of magnitude improvement in performance compared to a generic solver. The approach was shown to be suitable for efficient analytical modeling of CMP architectures.

- The scalability of the model was illustrated by estimating the performance of a CMP with nearly 700 components (cores, memories, routers) in one second.
- The estimation of CMP throughput using the approximation by static latency (i.e. without considering the interconnect contention) resulted in a poor ranking, which did not match well with simulation.
- The exploration of the design space, using the proposed model as an architecture estimator selected a very similar set of best-throughput architectures as simulation. Furthermore, modeling delivered more that 1300x speedup, as compared to simulation.

The techniques described in the following chapter show how this model can be combined with an intelligent search strategy to avoid exhaustive exploration.

# Chapter 5

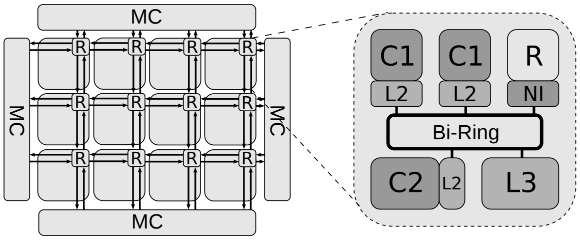
# Metaheuristics for Architectural Exploration

The goal of this chapter is to propose a methodology for efficient architectural exploration of large-scale hierarchical CMPs within vast design spaces. Efficiency is achieved by using analytical modeling instead of simulation, and by replacing the exhaustive exploration by an intelligent search strategy. The proposed approach is built on top of the analytical power-performance model presented in Chapter 4 and performs an efficient metaheuristic-based search that does not require simulation. To the best knowledge of the author, this work becomes the first one to integrate an analytical model for CMP with metaheuristic optimization.

Architectural exploration for CMPs must be performed on a highly discrete design space. For this reason, hill-climbing strategies for combinatorial optimization are a promising approach, as it was shown in [77]. Two probabilistic metaheuristics based on extensions of hill climbing are considered: *Simulated Annealing* [80] and *Extremal Optimization* [24]. SA has been successfully applied in many design automation problems and EO has recently emerged as a very competitive alternative. For our problem, both heuristics exhibit similar performance and results, although for some design spaces one may behave more efficiently than the other. This chapter explains the customization of both methods for the exploration problem.

## 5.1 The Exploration Problem

The exploration problem addressed in this chapter assumes tiled hierarchical CMP architecture, the benefits of which for shortening the CMP design cycle have already been discussed in the previous chapters. Tiled architectures are organized as arrays of identical clusters, each one incorporating one or several computing cores, caches and an on-chip interconnect. Figure 5.1 shows an example of a tiled hierarchical



$X$ = 4
$Y$ = 3
$\mathcal{I}$ = “Bi-ring”
$N_1$ = 2
$N_2$ = 1
$L1_1$ = 64K
$L2_1$ = 512K
$L1_2$ = 128K
$L2_2$ = 1M

Figure 5.1: A configuration example and its variables.

Table 5.1: Configuration parameters for 2 types of cores.

| **Name** | **Description** |
|---|---|
| $X, Y$ | Dimensions of the top-level mesh |
| $\mathcal{I}$ | Type of interconnect in a cluster (e.g. bus, ring) |
| $N_1, N_2$ | Number of the first/second-type cores in a cluster |
| $L1_1, L1_2$ | *L1* cache size for the first/second type of cores |
| $L2_1, L2_2$ | *L2* cache size for the first/second type of cores |

configuration, comprised of a 2D array (mesh) of clusters, where every cluster has two *C1* and one *C2* cores with private *L2* caches, a shared *L3* cache, a router of the inter-cluster interconnect network and a bi-directional ring for intra-cluster communication.

Consider an exploration design space $\mathbb{S}$ that contains all possible architectural configurations. Each configuration is uniquely determined by the values of the variables shown in Table 5.1. These variables include dimensions of the top-level mesh ($X$, $Y$), type of interconnect within a cluster ($\mathcal{I}$), and for every type of core $i$, the number of cores per cluster ($N_i$) and the sizes of the *L1* and *L2* caches ($L1_i$, $L2_i$). Table 5.1 shows the complete list of variables when the design space contains two types of cores. The process of adding new variables to the design space is straightforward.

Figure 5.1 also enumerates the design variables and their values for the considered configuration. Note that while the listed variables are the independent (free) variables of the exploration problem, the size of *L3* cache is chosen to be a dependent variable. In other words, *L3* cache fills all the area which is left after placing the cores, private caches and interconnect. For this reason, the size of the *L3* cache is computed using the area constraint and is not a free variable. An alternative formulation where *L3* is a free variable is also possible. The number of memory controllers is kept constant (we assume one *MC* at every side of the mesh).

The problem of architectural exploration consists in finding a configuration (or several configurations) to optimize the selected design metrics, subject to a set of constraints. For simplicity, in this work we focus on the following formulation: *Maximize the overall chip performance (`IPC`), subject to the resource budget, i.e. area and power constraints.* The methodology described here applies to other formulations of the problem and also allows the incorporation of additional metrics, such as thermal and other physical parameters.

To enforce the satisfaction of constraints we use penalty functions. For every constrained metric the objective is penalized once the metric value exceeds the constraint value.

Let us consider an area constraint indicating that the total area cannot exceed the value *MaxArea*. We define the *relative excess* of *Area* as

$$\mathit{Ex(Area)} = \frac{\max(0, \mathit{Area} - \mathit{MaxArea})}{\mathit{MaxArea}}. \tag{5.1}$$

The excess is zero when the area constraint is satisfied. Otherwise, it gives a relative degree of area violation. Next, we define the area penalty in the objective function:

$$\mathit{Pen}(\mathit{Area}) = \frac{1}{1 + \lambda \cdot \mathit{Ex(Area)}}, \tag{5.2}$$

where $\lambda$ is a penalty factor that grows as exploration advances. This decreases the probability of accepting infeasible configurations as the exploration evolves towards the final solution. Now consider the objective function in the form

$$\mathit{Obj} = \mathtt{IPC} \cdot \mathit{Pen}(\mathit{Area}).$$

If the area constraint holds ($\mathit{Area} \leq \mathit{MaxArea}$), then $\mathit{Pen}(\mathit{Area}) = 1$ and the objective is equal to the `IPC` of configuration. If $\mathit{Area} > \mathit{MaxArea}$, then $\mathit{Pen}(\mathit{Area}) < 1$ and the `IPC` value in the objective function is degraded.

Penalty functions in the form of equation (5.2) allow adjusting the objective of infeasible configurations with respect to the violation degree. This mechanism makes the design space smoother and allows leaving the feasibility region temporarily, rather than stopping at its boundaries. When the exploration is out of the feasibility region, the penalty grows with the distance to the region, thus forcing the exploration to progressively return to the feasibility region.

The complete objective function used in this work contains the penalty terms for area, power and aspect ratio of the top-level mesh, *AR*. These terms are defined similar to (5.2), so the final equation is:

$$\mathit{Obj} = \mathtt{IPC} \cdot \mathit{Pen}(\mathit{Area}) \cdot \mathit{Pen}(\mathit{Power}) \cdot \mathit{Pen}(\mathit{AR}). \tag{5.3}$$

One of the main properties of the exploration approach is its scalability, i.e. the ability to efficiently discover promising configurations within vast design spaces. Exhaustive exploration is therefore not feasible, and in this work metaheuristic-based search is considered as a strategy to achieve the efficiency in exploration. In the following sections customization of metaheuristics for the exploration problem is proposed.

## 5.2 Transformations

The concept of configuration *neighborhood* has a key importance for metaheuristics. The neighborhood of some configuration $\mathcal{C}$ is defined using a set of rules to obtain neighbors from $\mathcal{C}$. We refer to these rules as *transformations*. More precisely, every transformation identifies one or several variables of $\mathcal{C}$ that are modified to create a neighbor.

As an example, consider the `IncX` transformation, which increments the $X$-dimension of the top-level mesh by one. If $\mathcal{C}$ is a configuration with a 4×3 mesh (as in Figure 5.1), then `IncX` defines a neighbor of $\mathcal{C}$ which is a 5×3 mesh with the same values for the other variables.

Selecting the neighborhood size is another important aspect of metaheuristics. Small neighborhoods may cause metaheuristics to get stuck in a local optimum and lead to suboptimal solutions. Large neighborhoods may become the reason for slow advance through the design space and a significant increase of the algorithm execution time. The neighborhood size for $\mathcal{C}$ is primarily determined by the cardinality of the transformation set. Hence, an important question to solve is to find a moderately sized set of transformations to balance performance and quality. Another property that must be guaranteed is the reachability of any configuration $\mathcal{C} \in \mathbb{S}$ from any other $\mathcal{C}_0 \in \mathbb{S}$, via a sequence of transformations.

Three types of transformations are proposed for efficient exploration, as described below. To provide a visible example of the transformation set, Table 5.2 enumerates all transformations for two types of cores. Section 5.5 presents an example of how different types of transformations have an impact on the efficiency of the search.

### First-Order Transformations

First we define a group of basic transformations, each affecting only one variable of the configuration. They are referred to as *first-order* transformations. With every variable $V$ (from Table 5.1) we associate a pair of transformations, `IncV` and `DecV`, that increase and decrease the value of $V$ to the next available value in the domain, respectively. For example, if $X$ is allowed to take any integer value between 2 and 10, then the `IncX` transformation applied to a 4×4 mesh (Fig. 5.2(a)) will produce

Table 5.2: List of transformations for 2 types of cores.

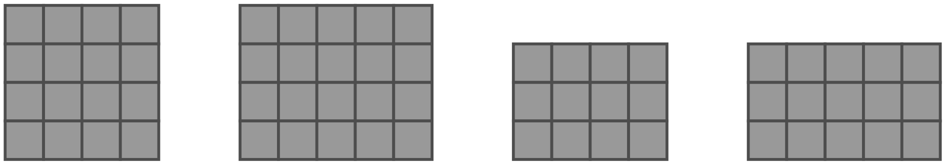
| **First-order** | | **Second-order** | |
|---|---|---|---|
| IncX | DecX | IncX-DecY | IncY-DecX |
| IncY | DecY | IncX-DecN$_1$ | IncN$_1$-DecX |
| Inc$\mathcal{I}$ | Dec$\mathcal{I}$ | IncX-DecN$_2$ | IncN$_2$-DecX |
| IncN$_1$ | DecN$_1$ | IncY-DecN$_1$ | IncN$_1$-DecY |
| IncN$_2$ | DecN$_2$ | IncY-DecN$_2$ | IncN$_2$-DecY |
| Inc$\mathit{L1}_1$ | Dec$\mathit{L1}_1$ | IncN$_1$-DecN$_2$ | IncN$_2$-DecN$_1$ |
| Inc$\mathit{L1}_2$ | Dec$\mathit{L1}_2$ | **Reclustering** | |
| Inc$\mathit{L2}_1$ | Dec$\mathit{L2}_1$ | RecIncX | RecDecX |
| Inc$\mathit{L2}_2$ | Dec$\mathit{L2}_2$ | RecIncY | RecDecY |

a 5×4 mesh (Fig. 5.2(b)). `DecY` applied to a 4×4 mesh will produce a 4×3 mesh (Fig. 5.2(c)). The other variables remain unchanged.

For the other variables, the next available values may be different. For example, the next available value for a memory of 128KB can be 256KB. The first-order transformations are summarized in the left column of Table 5.2. Every type of core implies independent transformations. For example, if there are two different types of cores, then `IncN`$_1$ and `IncN`$_2$ are different transformations, increasing the number of cores of either first or second type.

An order for interconnect types can be assigned according to some criteria, such as an increase in bisection bandwidth. Hence, the next value for the current interconnect type will be another type with the increased bisection bandwidth. As an example, a bus will be followed by a uni-directional ring, which will be followed by a bi-directional ring.

Transformations may also produce illegal configurations. For example, if $X$ has to be in between 2 and 10, then `DecX` applied to a 2×2 mesh will produce a configuration which is outside the design space. In this case `DecX` is an infeasible transformation and will not be considered when exploring the neighborhood of a 2×2 mesh.



(a) Original: 4×4 (b) `IncX`: 5×4 (c) `DecY`: 4×3 (d) `IncX`, `DecY`: 5×3

Figure 5.2: Transformations applied to a 4x4 mesh.

It is obvious that first-order transformations guarantee the reachability between any pair of configurations. Indeed, if configurations $\mathcal{C}_0$ and $\mathcal{C}$ differ in some variable $V$, then a sequence of either `IncV` or `DecV` will equalize the values.

## Second-Order Transformations

First-order transformations still define small neighborhoods that may impede the search to escape from local optima with high probability and increase the metaheuristic running time. This fact is particularly important for highly constrained design spaces, when the penalty for violating a constraint is too high[1].

Consider again the configuration with 4×4 mesh shown in Figure 5.2(a). `IncX` produces a 5×4 mesh with 25% more area and a low probability to be accepted if the area penalty is high (Fig. 5.2(b)). On the other hand, `DecY` yields a 4×3 mesh which has 25% less area (and hence, less computing cores and cache, Fig. 5.2(c)). It is not likely to be accepted either, since the performance of this configuration is significantly lower, compared to the original solution. However, if we apply both transformations *simultaneously*, we obtain a 5×3 mesh, as shown by Fig. 5.2(d). This solution has comparable area and performance, and hence higher probability to be accepted.

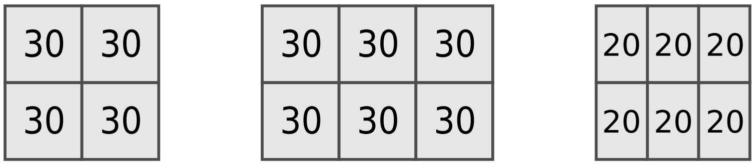
We refer to the transformations that perform a simultaneous update of two variables in the current configuration as *second-order* transformations. Rather than proposing transformations for all pairs of variables, we select a small group of variables, which we observe to enhance the neighborhood effectively. In particular, we create second-order transformations for the mesh dimensions ($X$ and $Y$) and the number of cores of each type ($N_i$). For any pair of these variables, one is increased, while the other is decreased. An example of the `IncX-DecY` transformation was shown in Fig. 5.2(d). The right column of Table 5.2 enumerates the second-order group.

## Reclustering

Finally we introduce the third type of transformation, *reclustering*, which decreases the search time notably, as it extends the neighborhood with promising configurations that were previously achievable only after a sequence of steps.

The idea of reclustering is illustrated in Fig. 5.3. Assume the original configuration is a 2×2 mesh with 30 cores per cluster, i.e. 120 cores in total (Fig. 5.3(a)). The `IncX` transformation will deliver a 3×2 mesh, as in Fig. 5.3(b). However, as the number of cores per cluster has not changed, the new configuration will have 180 cores with a 50% area increase. This is likely to cause unacceptable area and

[1] The mechanism of penalty functions used for constraint modeling is explained in section 5.1.



(a) 2×2 mesh with 120 cores

(b) 3×2 mesh with 180 cores

(c) 3×2 mesh with 120 cores

Figure 5.3: Reclustering of the 2×2 mesh.

power penalties and prevent the exploration from ever escaping the 2×2 size neighborhood. To compensate, we propose adjusting the number of cores accordingly, so that the total number of cores remains constant. In the example, the new number of cores per cluster becomes $\lfloor \frac{30 \cdot 2}{3} \rfloor = 20$. Figure 5.3(c) shows the new configuration. We refer to this procedure as reclustering, as it permits changing the number of clusters without causing strong penalties for any of the metrics.

Reclustering affects $k + 1$ variables, where $k$ is the number of core types in the design space. Indeed, either variable $X$ or $Y$ is updated to resize the mesh while all $N_i$ variables ($i = 1, .., k$) are updated to adjust the number of cores. There are four reclustering transformations, which are defined by either increasing, or decreasing any of the mesh dimensions. They are listed in the right column of Table 5.2.

## 5.3 Exploration with Simulated Annealing

The *Simulated Annealing* (SA) [80] algorithm is outlined in procedure 1. It starts with some initial configuration, that can be chosen randomly. We typically assign the lowest feasible value for every variable of configuration to prevent the constraints from being violated.

The algorithm implements a conventional annealing schedule. Given an initial temperature $T_{init}$ and cooling factor $\alpha < 1$, a new configuration ($NewC$) is generated (lines 5-6). It may be accepted probabilistically, depending on the current temperature $T_{cur}$ (line 7). The value of $T_{cur}$ decreases as the system evolves in time (line 10). Penalization weight $\lambda$ grows in time to decrease the probability of accepting infeasible solutions (line 11). For every temperature, a number of moves that depends on the size of the problem ($k = P$, the total number of available transformations) is generated.

$NewC$ is obtained by selecting some transformation $t_i$ and applying it to $CurC$. The probability distribution for the selection of $t_i$ can be specified by the user. In this work we used uniform probabilities for all $t_i$ and selected the transformation to be applied at every iteration randomly.

**Procedure 1** SimulatedAnnealing

```
$CurC \leftarrow BestC \leftarrow$ "Some initial solution"
$T_{cur} = T_{init}$
while improvement in last $k$ iterations do
  for P iterations do
    select $t_i$ randomly with uniform probability
    generate $NewC$ by applying $t_i$ to $CurC$
    if Accept($CurC$, $NewC$) then $CurC \leftarrow NewC$
    if Obj($CurC$) > Obj($BestC$) and Feasible($CurC$) then $BestC \leftarrow CurC$
  end for
  $T_{cur} = \alpha \cdot T_{cur}$
  $\lambda = \lambda / \alpha$
end while
return $BestC$
```

**Procedure 2** Accept($CurC$, $NewC$)

```
1: if Obj($NewC$) > Obj($CurC$) then return true
2: else return true with probability $P_a$, defined by (5.4)
```

The new configuration is accepted probabilistically according to Procedure 2. Configurations with better objective value (calculated using (5.3)) are always accepted (line 1), other configurations are accepted with probability $P_a$:

$$P_a = e^{-\gamma}, \quad \gamma = \frac{\text{Obj}(CurC)}{\text{Obj}(NewC)} \cdot \frac{1}{T_{cur}}. \tag{5.4}$$

This probability depends on two exponent factors. The former avoids the acceptance of solutions with high degradation of the objective function. The latter increases the probability of hill climbing as the temperature cools down.

We use a commonly applied strategy for defining the initial temperature, $T_{init}$ [134, 17]. Running the algorithm for $P$ iterations and always accepting the new configurations, we calculate the average cost variance as:

$$\Delta = \frac{1}{P} \sum_{i=1}^{P} \frac{\text{Obj}(CurC)_i}{\text{Obj}(NewC)_i}.$$

This value is further used in (5.4) in place of the objective ratio to evaluate $T_{cur}$ which gives a high initial acceptance probability, e.g. around 0.95.

Note that the best solution is updated only when the *CurC* is feasible as given by the FEASIBLE(*CurC*) procedure in line 8. This procedure returns true if and only if *CurC* satisfies all constraints (for area, power and aspect ratio in our case).

## 5.4 Exploration with Extremal Optimization

*Extremal Optimization* (EO) [24] is inspired by the principle of evolution in ecosystems, which were observed to evolve by selecting against its worst components (or features). We draw here an analogy with the exploration problem, by considering every configuration to be determined by the conjunction of its variables, $\mathcal{P}$, similarly to the components (features) of the ecosystem.

Given a configuration, EO evaluates the *fitness* of its variables by comparing the current objective value with that of the neighbors, defined by the transformation set. A variable is well-fit if the configuration objective can not improve significantly by applying any of the transformations changing this variable. EO focuses on improving the status of variables with low fitness.

Since there are transformations that update several variables simultaneously, it is more convenient to work with the *transformation fitness.* As opposed to variables, well-fit transformations are those that cause higher improvement of the objective, when applied. More precisely, given the current configuration *CurC*, the fitness of transformation $t_i$ is:

$$\Phi = Obj(NewC), \tag{5.5}$$

where *NewC* is the configuration obtained by applying $t_i$ to *CurC.* To maximize the objective function, at every iteration EO algorithm selects a transformation with high fitness and applies it to the current configuration.

Local optima are avoided by randomizing the selection process. The transformations are ranked according to their fitness in descending order (the best transformations have lower indices in the rank). The transformations are randomly selected by some probability distribution biased towards the ones with highest fitness values. The power-law distribution is a typical one for EO. For example, if the system has $N$ transformations ranked from 1 to $N$ in descending order of their fitness, the index $i$ of the selected transformation can be calculated as follows:

$$i = \lceil N \cdot p^{\tau} \rceil \tag{5.6}$$

where $p$ is a random number obtained from a uniform distribution in the interval $[0, 1]$ and $\tau$ is the power law exponent.

**Procedure 3** ExtremalOptimization

```
 1: $CurC \leftarrow BestC \leftarrow$ ”Some initial solution”
 2: while some improvement in the last $k$ iterations do
 3:     local search: evaluate $P$ randomly selected transformations and accept only
        those that improve the Obj($CurC$);
 4:     sort all transformations in descending order of $\Phi$;
 5:     select $t_i$ according to equation (5.6);
 6:     apply $t_i$ to $CurC$;
 7:     if Obj($CurC$) > Obj($BestC$) and Feasible($CurC$) then $BestC \leftarrow CurC$;
 8:     $\lambda = \lambda \cdot \beta^{\tau}$;
 9: end while
10: return $BestC$
```

The EO algorithm is described in Procedure 3. After the definition of an initial solution, the execution is continued until no further improvement is observed during a certain number of iterations.

In this work we use a variation of EO called *Continuous Extremal Optimization* [140]. This variant combines EO with a local search at the beginning of each iteration, contributing to improve the objective value of the final solution and the speed of the algorithm. Local search implements hill climbing by sequentially trying $P$ random transformations and accepting only those that improve the objective.

To select a transformation $t_i$, all transformations are sorted according to their fitness value (5.5). The power law described by equation (5.6) is used to randomize the selection. Finally $t_i$ is accepted unconditionally and *BestC* is updated if the objective is better than any other configuration visited so far. Penalization weight $\lambda$ increases as the system evolves, as in the SA algorithm. Since EO does not have the notion of temperature, we select a geometric law to update $\lambda$ (line 8), where $\beta$ is a constant, slightly greater than one (such as 1.01). The selection of $\beta$ has to assure that $\lambda$ grows slow enough for an efficient exploration, which implies a periodical acceptance of infeasible configurations.

## 5.5 Experimental results

In this section we present the experiments used to validate the quality and efficiency of metaheuristics. For this purpose we consider a substantially large search space $\mathbb{S}_2$, containing about $1.5 \cdot 10^9$ configurations. The parameters of $\mathbb{S}_2$ and their comparison with the parameters of space $\mathbb{S}_1$ from previous chapter are given in Table 5.3.

There are three types of cores available in $\mathbb{S}_2$, which is the major reason for the dramatic explosion in the number of configurations. Figure 5.4 plots the ideal

Table 5.3: Parameters of the exploration spaces $\mathbb{S}_1$ and $\mathbb{S}_2$.

| **Parameter** | $\mathbb{S}_1$ **values** | $\mathbb{S}_2$ **values** |
|---|---|---|
| Core types | *C2* | *C1*, *C2*, *C3* |
| L1 size | 64, 128 Kb | 64, 96, 128 Kb |
| L2 size | 128 Kb to 1 Mb | 64 Kb 1 Mb |
| Mesh dimensions | 2×2 to 10×10 | 2×2 to 16×16 |
| Chip area | 350 mm$^2$ | |
| Memory density | 1 mm$^2$ / Mb | |
| Off-chip memory latency | 100 cycles | |
| Type of cluster interconnect | Bus, uni-ring, bi-ring | |
| Interconnect link width | 256 bits | |
| Workload `MPI` | 0.25 | |
| Workload `MLP` | 1.25 | |

performance of every type of core (*C1*, *C2* and *C3*) as a function of the core area. The parameters for *C2* are obtained from [117], whereas the parameters for *C1* and *C3* are generated by applying Pollack's rule [116] to the parameters of *C2*. We also assume that the smallest core, *C1*, implements in-order execution (IO), while the other two implement out-of-order execution (OoO). The models for in-order and out-of-order architectures were explained in Section 4.4.

Additional values for cache sizes and mesh dimensions also contribute to the expansion of the design space.

Without loss of generality, we assume all cores to be running the same application. For this study we select the *soplex* application from SPEC CPU2006. Figure 5.5 depicts the miss ratio produced by the application as a function of the cache size.

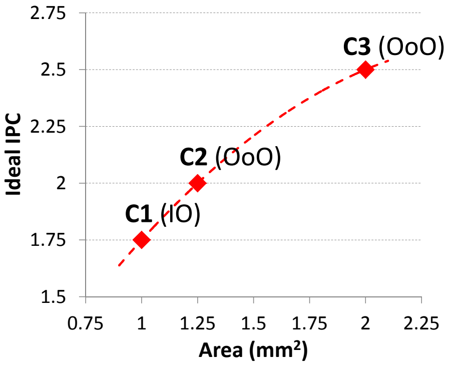


Figure 5.4: Core types for $\mathbb{S}_1$ and $\mathbb{S}_2$ exploration.

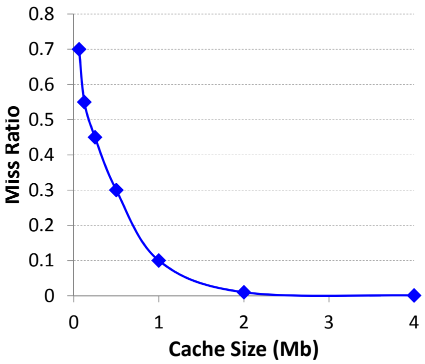


Figure 5.5: Cache miss model of *soplex* application.

There are several ways how exploration can be performed for multiple applications. If all cores can execute only one type of application simultaneously, which can change in time though, one can evaluate a configuration with a weighted objective function. For example, if for two applications $\mathcal{A}_1$ and $\mathcal{A}_2$ some configuration $\mathcal{C}$ delivers performance $\texttt{IPC}(\mathcal{A}_1)$ and $\texttt{IPC}(\mathcal{A}_2)$ respectively, then the aggregate performance for $\mathcal{C}$ can be defined as $\texttt{IPC}(\mathcal{C}) = \alpha \cdot \texttt{IPC}(\mathcal{A}_1) + \beta \cdot \texttt{IPC}(\mathcal{A}_2)$, where $\alpha$ and $\beta$ are the user-defined weights.

In the other scenario, several applications can be executed in parallel. For this case, the applications have to be distributed among the cores assuming some mapping algorithm, and the exploration procedure can be applied without any change.

## Evolution of search in time

Figure 5.6 shows the evolution of the best throughput discovered by the heuristics as the exploration evolves. To emphasize the need for intelligent search, we compare SA and EO with a naive *Random Best* (RB) strategy. RB simply generates random solutions, without tracking any history, keeping only the best known result.

The exploration is performed with a maximum power $P_{\max} = 180W$, which was found to be a reasonable value to explore the trade-off between throughput and power. The metaheuristic parameters are $\alpha = 0.995$ and $\tau = 1.6$.

One can observe that SA discovers the optimum[2] (103.26 IPC) in about 160 seconds. EO reaches the optimum in just 80 seconds. However, for RB it takes almost 400 seconds to find a configuration with 85.71 IPC, 17% worse than the optimum. The exhaustive exploration of all configurations in $\mathbb{S}_2$ would have taken more than 100 days using a single-core machine. These facts justify the importance of the metaheuristics in the exploration.

## Importance of selecting the set of transformations

Next, the importance of several transformation types is demonstrated. Figure 5.7 shows the evolution of the best discovered throughput in time (using Simulated Annealing), when the power constraint is set to 180W. The dashed line shows an exploration trace for the case when only the first-type transformations are considered. It takes approximately 600 seconds to reach the optimum solution of 103.26 IPC. The solid line depicts a trace for the case when all three transformation types are used. Time to reach the optimum solution drops to 120 seconds, delivering a 5x speed-up.

[2] We have computed the optimum by exhaustively running the model for all configurations in $\mathbb{S}_2$. A high-performance computing cluster was used, which effectively reduced the exploration time to five days.

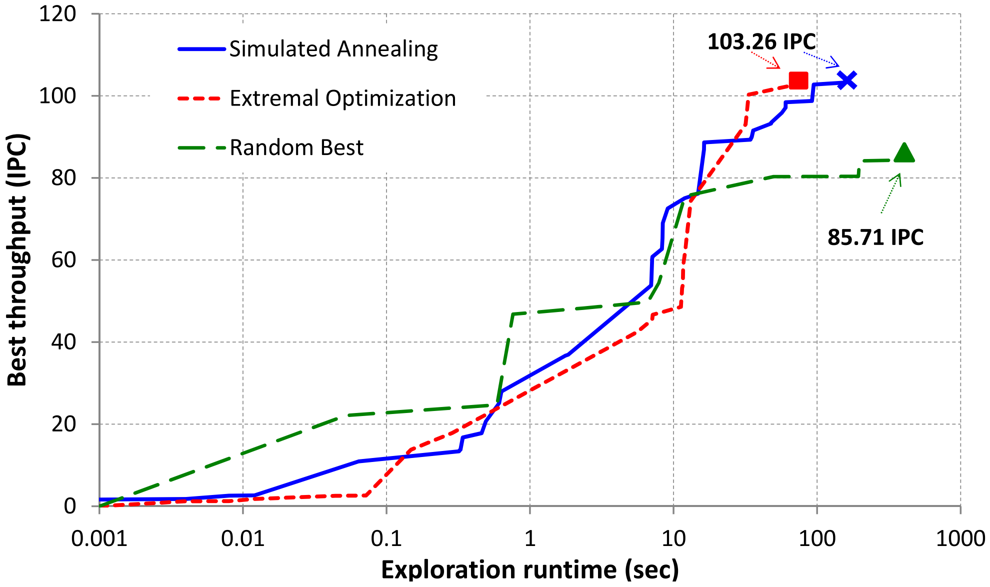


Figure 5.6: Evolution of best discovered throughput in time.

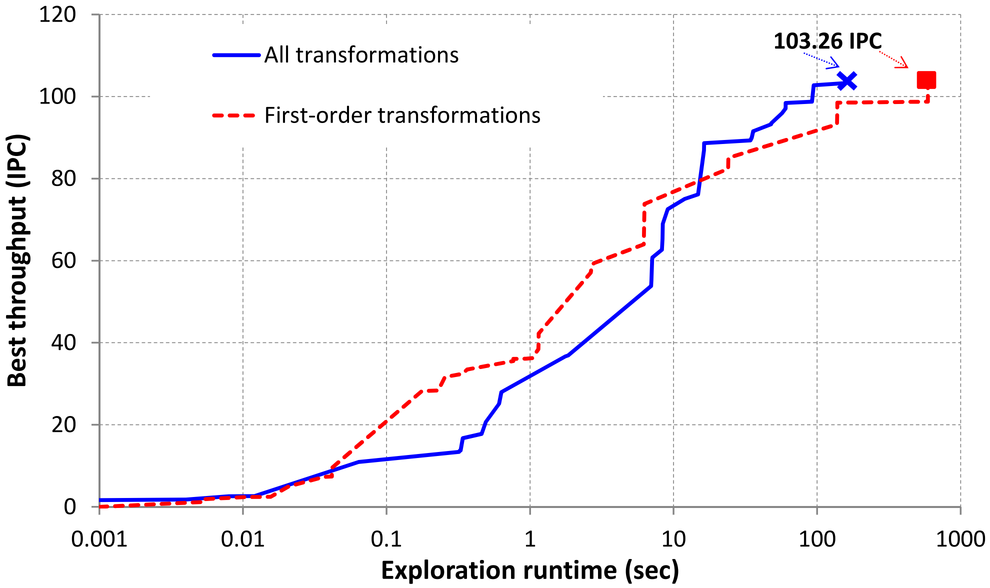


Figure 5.7: Evolution of the exploration objective in time for different sets of transformations.

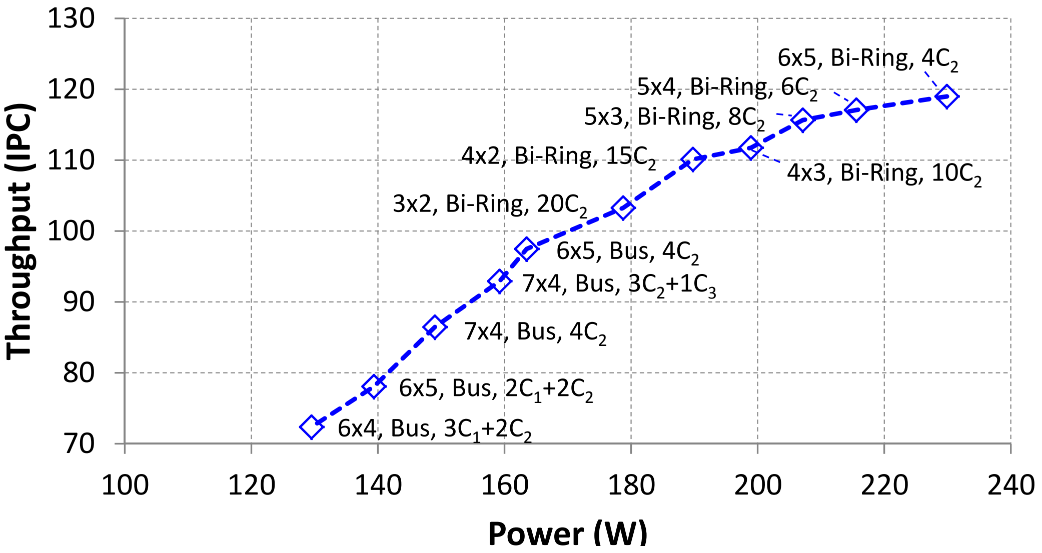


Figure 5.8: Power-performance trade-off in $\mathbb{S}_2$.

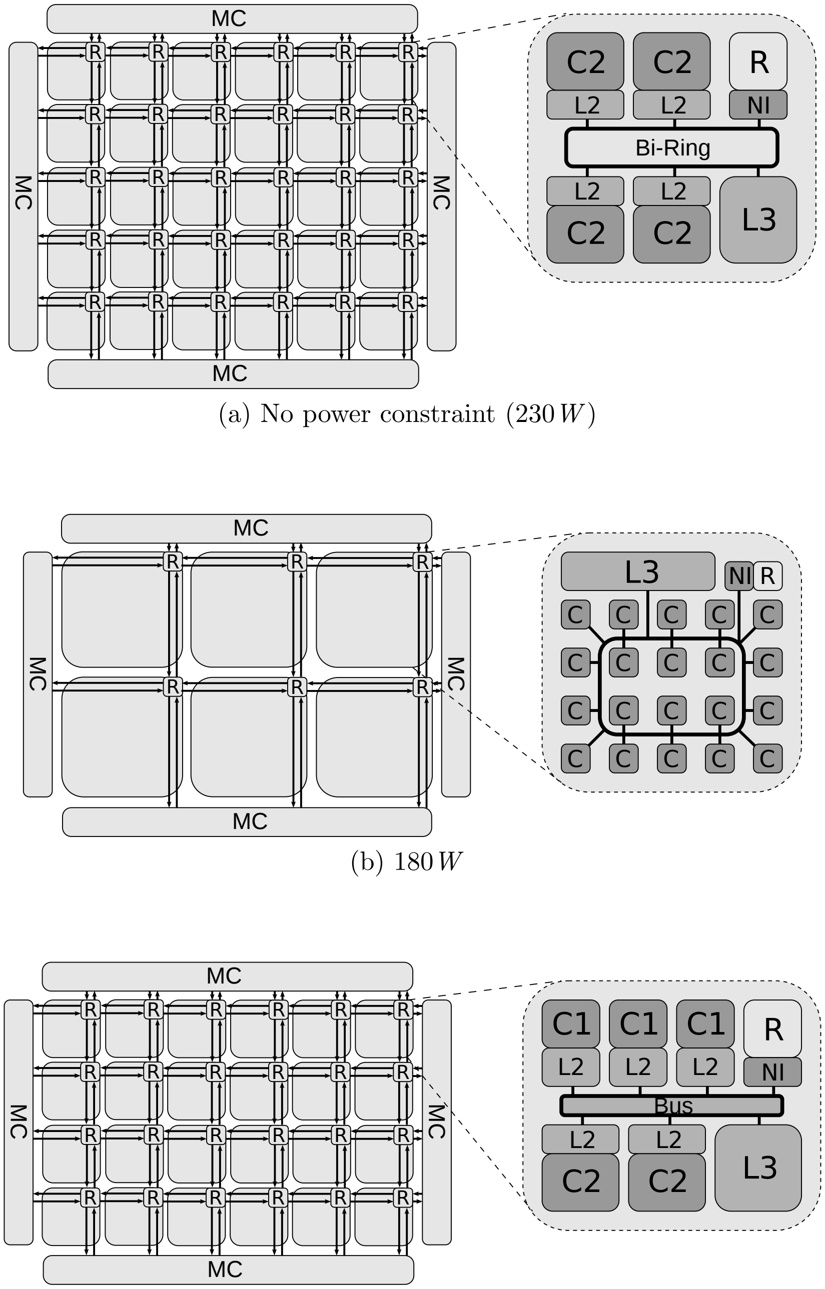
This example emphasizes how the second-type and reclustering transformations improve the performance of metaheuristic-based search.

## Power-performance exploration

This study shows how our model can be used to explore power-performance tradeoffs. Figure 5.8 depicts configurations with best throughput (Y-axis) discovered by metaheuristics with different power budgets (X-axis). We start with the power limit that delivers highest performance, and reduce the budget in amounts of $10W$ until we reach $130W$. This plot illustrates the expected decrease in performance for configurations with lower power budgets.

The configuration with highest performance (118.99 IPC) has a power of $230W$. The block diagram of this configuration is shown in Fig. 5.9(a) and represents a 6×5 mesh with four *C2* cores and bi-ring interconnect in clusters. As we reduce the power budget, the mesh dimensions are reduced, decreasing the number of high-performance but power-hungry routers. The clusters incorporate more cores, resulting into higher latencies of memory access. Therefore, performance of configurations decreases. Note that the bi-ring interconnects are preferred for budgets greater than $180W$. The layout of the last configuration with a bi-ring, at $180W$, is shown in Fig. 5.9(b). It has a 3×2 mesh and 20 *C2* cores per cluster.

Bi-rings deliver the highest performance, however they are less power-efficient when compared to buses. Hence, buses replace rings as soon as the power budget



(a) No power constraint ($230\,W$)

(b) $180\,W$

(c) $130\,W$

Figure 5.9: Best configurations, discovered by exploration for some power budgets.

drops significantly (below $180W$). With further decrease in the budget, power is saved by either reducing the number of cores, or by selecting more power-efficient cores (e.g. *C1* vs. *C2*). Thus, the best configuration for $130W$ contains 24 bus-based clusters with 3 *C1* and 2 *C2* cores (Fig. 5.9(c)).

Also note that *C2* cores are typically preferred when the power budget does not impose strong limits. This is explained by the fact that *C2* are the most efficient in terms of performance per area unit. *C2* is out-of-order, hence it hides memory latency more efficiently than the in-order *C1* core. At the same time, even though *C3* is out-of-order too, *C2* delivers more performance per unit area, which makes it the most competitive among the three types of cores.

## Comparison with simulation

The main question this section tries to answer is: how can we check that the accuracy of the results given by metaheuristics is acceptable for a huge search space?

To answer this question one would need to simulate all configurations exhaustively. This task is intractable due to enormous computational cost. What we propose is to run the exploration and store the $n$ best configurations discovered by the search ($n$ is a parameter). Afterwards, we simulate those $n$ configurations and check whether the best configuration retains its rank after simulation.

In this experiment we run the EO algorithm with $\tau = 1.6$ and a power constraint of $180W$. As the search evolves, a set of $n = 100$ best configurations is maintained. Upon the algorithm termination these configurations are simulated. The dotted line in Fig. 5.10 shows the throughput values in descending order, as calculated by the analytical model, with the maximum being 103.26 `IPC` (#1) and the minimum

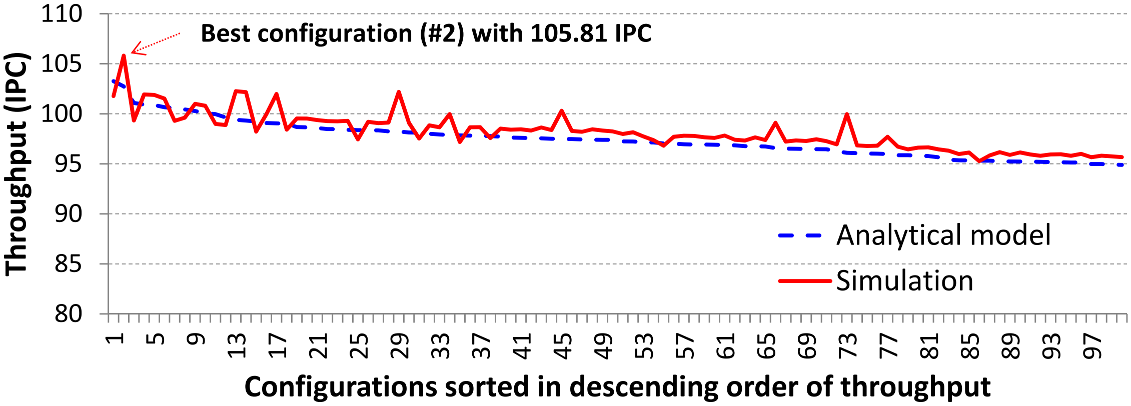


Figure 5.10: Throughput of 100 best configurations.

94.77 IPC (#100). Then, all 100 configurations are simulated, and for each one the throughput obtained by simulation is plotted (solid line in Fig. 5.10).

The best configuration by simulation is #2, with a throughput of 105.81 IPC. Although this configuration is assigned the second place in the order created by model, the difference between the simulated throughput of #1 and #2 is less than 4%. Certain deviations in this experiment are inevitable due to the simplifying assumptions of the analytical model. However, for the majority of configurations the tendency for the throughput is to decrease as the rank id increases, indicating a good correlation between the simulator and the model.

## 5.6 Conclusions

This chapter shows how to effectively reduce the number of configurations considered during the exploration by using metaheuristics for the combinatorial optimization. This contribution is indispensable to make the architectural design space exploration of future hundred- and thousand-core CMPs tractable.

The main conclusions from experimental results are summarized below:

- The metaheuristic-based search can discover the optimal configuration of a large design space, such as $\mathbb{S}$, within tens of seconds. Naive search strategies show worse results, even with longer running times (17% lower throughput with 5x increase in exploration time).
- The proposed methodology can be efficiently applied to power-performance exploration of CMPs. An example of the power-constrained exploration was given above.
- By simulating 100 configurations with the top throughput, discovered by the search, it was demonstrated that the throughput rankings between the model and simulation match closely. This fact is an additional confirmation of the quality of the analytical model introduced in Chapter 4, when applied to large search spaces.

These results confirm that the metaheuristic-based search strategy, supplied by the carefully selected transformations, can replace exhaustive search and substantially reduce the exploration time.

There remain open questions related to the selection of transformation set in an optimal way. Chapter 8 discusses some of these questions, which require future investigation.

# Chapter 6

# Static Task Mapping for Tiled Chip Multiprocessors

The previous contributions of this thesis focus on different aspects of the *design* of CMPs. The contribution described in this chapter aims at the efficient *usage* of CMPs after manufacturing, by addressing the problem of application task mapping.

To assure the performance and thermal properties of the system, CMPs are designed to operate under a certain power budget. An effective way to manage CMP power is to floorplan several voltage islands and assign the best voltage and frequency for each island. The number of voltage islands is constrained by the design of the power delivery network and costly implementation of voltage regulators. It is therefore realistic to consider that CMPs with hundreds of cores will have voltage islands with several cores (e.g., 4 or 8). This fact imposes an additional constraint in the task mapping problem: even though some cores could possibly run at lower voltages and frequencies, sharing the island with other cores may prevent from having this flexibility.

The examined problem consists of *statically mapping* a set of parallel tasks onto a many-core CMP and selecting the voltages of islands so that the total system power is minimized. Every task is considered as an infinite process (encountered in control, automotive and robotics systems) and has an associated throughput constraint that guarantees the required QoS for that task. A variety of processor classes is supported, each one characterized by a set of voltage/performance/power parameters used to find the best performance/power trade-off for each task.

The main contributions of this work can be summarized as:

- Mathematical formulation of a mixed-integer linear programming problem (MILP) delivering optimal mapping solutions for the examples of small size.

- Heuristical mapping optimization by Simulated Annealing (SA).
- Scalable approach based on *Extremal Optimization* (EO) [24], shown to outperform the optimization by SA, both in quality and computational cost.

This work has been published in [101] and [99].

## 6.1 An example of the mapping problem

This section discusses the task mapping problem using a small example. Let us assume a task graph with four tasks (Fig. 6.1(a)). There are three flows between the tasks, with the bandwidths specified in the arcs of the graph (in Gbps). Figure 6.1(b) depicts a CMP with four processors. There are two classes of processors: $C_1$ (light) and $C_2$ (dark). The task graph must be mapped into the CMP.

### Communication-optimal mapping

Figure 6.1(c) shows a task mapping that optimizes the communication metric, that is the product of bandwidth and hop-count. Assuming the distance between the neighboring processors is one hop, the communication cost of this mapping is

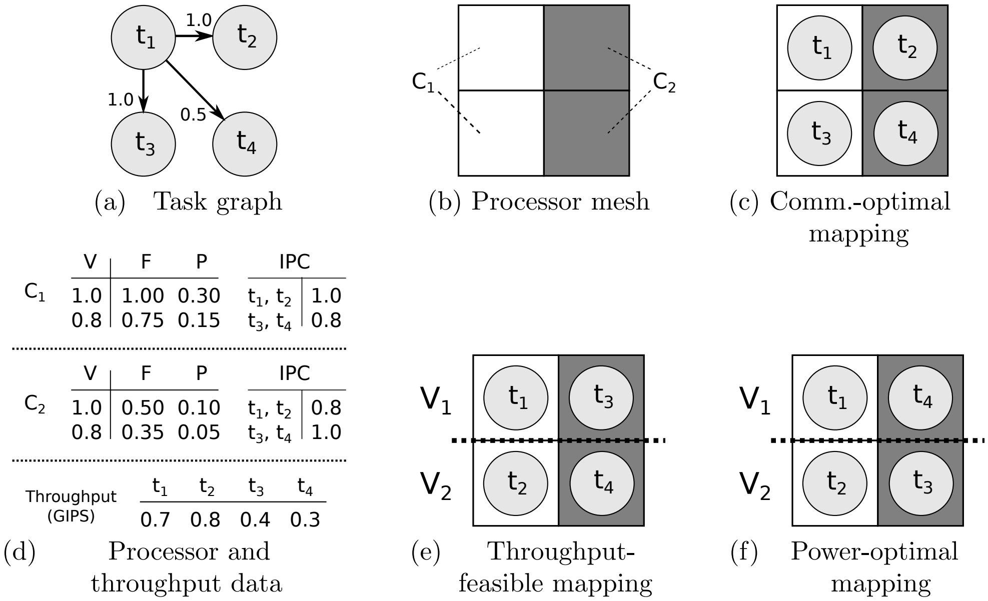
$$CCost_1 = 1.0 \cdot 1 + 1.0 \cdot 1 + 0.5 \cdot 2 = 3.0 \ (Gbps).$$

### Throughput-feasible mapping

Now let us take into account the processor parameters and consider the throughput requirements of the tasks. Figure 6.1(d) describes the processor parameters. They can operate at two voltages, 1.0 and $0.8V$. The corresponding frequency ($F$, in *GHz*) and power ($P$, in $W$) for each voltage is shown in the tables. Due to the nature of the tasks and the implementation of each processor, each task may be executed with a different performance (Instructions Per Cycle (IPC)) in each class of processors. Finally, each task may require a specific throughput (given in giga-*IPS* in Fig. 6.1(d)).

The mapping in Fig. 6.1(c) is infeasible for the specified throughput constraints. Consider task $t_2$ assigned to a $C_2$-processor. The maximum performance that $C_2$ can provide for $t_2$ is $\texttt{IPC}(t_2) \cdot F(1.0V) = 0.8 \cdot 0.5 = 0.4$ *GIPS*, while the throughput requirement for $t_2$ is 0.8 *GIPS*.

To satisfy the requirements, tasks $t_2$ and $t_3$ are swapped (see Fig. 6.1(e)). This mapping satisfies the throughput constraints and still keeps the optimal value for the communication metrics.



(a) Task graph (b) Processor mesh (c) Comm.-optimal mapping

(d) Processor and throughput data (e) Throughput-feasible mapping (f) Power-optimal mapping

Figure 6.1: Task mapping example.

### Power-optimal mapping

As a final step, let us consider the partitioning of the CMP into voltage islands. Let us assume the CMP has two islands, separated by the bold dotted line, as shown in Fig. 6.1(e). Processors in the same island must operate at the same voltage level, that is the minimal voltage required to satisfy all the throughput constraints for the tasks mapped to this island.

For the mapping in Fig. 6.1(e), the upper island has to operate at $1.0V$ dictated by the throughput constraint of $t_3$. The lower island also has to run at $1.0V$, because of $t_2$. Thus, the computation power, calculated using the data from Fig. 6.1(d), is $P_{comp} = 0.30 + 0.10 + 0.30 + 0.10 = 0.80\ W$. Let the energy to transfer one bit for one hop be $E_{bit} = 0.1nJ/bit$. Then the communication power is

$$P_{comm} = CCost_2 \cdot E_{bit} = 3.0Gbps \cdot 0.1nJ/bit = 0.3\ W,$$

and the total power $P = P_{comp} + P_{comm} = 1.10\ W$.

Notice that if we swap tasks $t_3$ and $t_4$ (Fig. 6.1(f)), the upper island can lower the voltage to $0.8V$ without violating the throughput constraints. The new computation power is $P_{comp} = 0.15 + 0.05 + 0.30 + 0.10 = 0.60\ W$. The communication cost is increased: $CCost_3 = 1.0 \cdot 1 + 1.0 \cdot 2 + 0.5 \cdot 1 = 3.5\ (Gbps \cdot hop)$, so the communication power becomes $P_{comm} = CCost_3 \cdot E_{bit} = 3.5 \cdot 0.1 = 0.35\ W$. However, the total power $P = 0.95\ W$ decreases, making the assignment in Fig. 6.1(f) the best one in terms of total power.

The previous example demonstrates the importance of the task mapping problem when trying to minimize power consumption in a CMP with multiple classes of processors and voltage islands. The next section shows how optimal solutions for small instances of the problem can be found based on an MILP formulation.

## 6.2 A mathematical model

This section gives a formal definition of the problem via a Mixed-Integer Linear Programming model. This model will be later used as the basis of a heuristic method for large-scale systems based on Simulated Annealing and Extremal Optimization.

### Parameters of the problem

The parameters of the problem are summarized in Table 6.1. The variables of the MILP formulation are outlined in Table 6.2.

A *task graph* $TG(\mathcal{T}, \mathcal{F})$ is a directed graph with vertices representing the tasks $t_i \in \mathcal{T}$. Each arc represents a flow $f_{sd} \in \mathcal{F}$ that defines the communication from task $t_s$ to $t_d$. Every flow has a minimum required bandwidth $B_{sd}$. Every task $t_i$ has a throughput constraint $\texttt{IPS}(t_i)$, that is the minimum number of instructions per second required to provide the service delivered by the task. $\Lambda(t_i)$ defines the total traffic rate between $t_i$ and the memory controller. The ratio between the traffic to and from the controller is specified by the parameter $\rho$. Note that $\Lambda(t_i)$ value can be approximated, given the amount of data, operated by the task (i.e. the working set), and the size and miss-ratio of the tile cache.

A CMP is represented by a *mesh of processors* $PM(\mathbb{P}, \mathcal{L})$ with dimensions $W \times H$, where $\mathbb{P}$ is the set of processors and $\mathcal{L}$ is the set of communication links. Links are organized in an on-chip network with a regular mesh topology [92]. The communication capacity between the neighboring cells is determined by the global parameter $\texttt{Cap}$ (all links are assumed to have the same capacity). Every cell represents a processor $p_j$, belonging to one of the processor classes in $\mathbb{C} = \{c_1, .., c_C\}$. Different classes of processors have distinct performance executing each task. The performance of $p_j$ to execute task $t_i$, measured in instructions per cycle, is specified by the function $\texttt{IPC}(t_i, p_j)$.

The processors may operate at different voltages. We assume a set of voltages $\mathcal{V} = \{v_1, .., v_V\}$ available for all processors. The frequency and power of $p_j$ operating at voltage $v_k$ are defined by the functions $F(p_j, v_k)$ and $P(p_j, v_k)$, respectively. Every $p_j$ belongs to some voltage island $\iota_n$, as defined by the island map $\mathcal{I}$. The voltage of an island can be adjusted independently of the other islands, however, all processors in an island must operate at the same voltage.

| Task parameters | |
|---|---|
| $TG$ | Task graph with tasks $t_i$ and flows $f_{sd}$ |
| $B_{sd}$ | Bandwidth requirement for flow $f_{sd}$ |
| $\texttt{IPS}(t_i)$ | Throughput requirement for task $t_i$ (instr./sec.) |
| $\Lambda(t_i)$ | Traffic of task $t_i$ to the memory controller |
| $\rho$ | Ratio of traffic rates to and from controller |
| Processor grid parameters | |
| $PM(\mathbb{P}, \mathcal{L})$ | Mesh of processors ($\mathbb{P}$) with communication links ($\mathcal{L}$) |
| $\texttt{Cap}$ | Maximum capacity of the communication links |
| $\texttt{IPC}(t_i, p_j)$ | Performance of $p_j$ executing $t_i$ (instr./cycle) |
| $\mathcal{V}$ | Set of available operating voltages $\upsilon_k$ |
| $F(p_j, \upsilon_k)$ | Frequency of processor $p_j$ at voltage $\upsilon_k$ |
| $P(p_j, \upsilon_k)$ | Power of processor $p_j$ at voltage $\upsilon_k$ |
| $MC(p_j)$ | Memory controller associated with $p_j$ |
| $\texttt{McDist}(p_j)$ | Distance from $p_j$ to associated controller |
| $\texttt{McBw}$ | Maximum bandwidth of memory controllers |
| Voltage island parameters | |
| $\{\iota_n\}$ | Set of voltage islands |
| $\mathcal{I}$ | Map from processors to voltage islands |

Table 6.1: Input parameters of the problem.

| Variable | Type | Description |
|---|---|---|
| $a_{ijk}$ | Binary | task $t_i$ is assigned to processor $p_j$ with voltage $\upsilon_k$ |
| $\upsilon_k^n$ | | voltage island $\iota_n$ operates at voltage $\upsilon_k$ |
| $r_{sd}^l$ | | link $l$ belongs to the route of flow $f_{sd}$ |
| $m_{sd}^l$ | Real | mapping indicator for the terminals of $f_{sd}$ |
| $h_{sd}^x, h_{sd}^y$ | | hop-count ($x$ and $y$) of route $f_{sd}$ |

Table 6.2: Variables of the MILP formulation.

A CMP has a set of controllers to access the off-chip memory. Guided by the existing implementations [118, 119], in this work we assume controller placement at the periphery of the mesh. However, this does not limit the proposed approach from having the controllers placed inside the mesh, that was demonstrated beneficial by recent research [10]. Another assumption we make is that every processor $p_j$ is associated with *one* controller, as defined by the function $MC(p_j)$. This assumption can be eliminated by specifying the probabilities of accessing different controllers for $p_j$. Function $\texttt{McDist}(p_j)$ returns the hop-count distance from $p_j$ to the related controller. The `McBw` parameter sets the maximum controller bandwidth to guarantee performance of memory access.

## Cost function

The goal of the model is to minimize power consumption under a set of design and performance constraints.

The binary variables $a_{ijk}$ define whether task $t_i$ is mapped onto processor $p_j$ operating at voltage $v_k$. The total power consumption for computation can be defined as follows:

$$P_{comp} = \sum_{t_i \in \mathcal{T}} \sum_{p_j \in \mathbb{P}} \sum_{v_k \in \mathcal{V}} a_{ijk} \cdot P(p_j, v_k).$$

The power consumption for communication has two terms: the on-chip communication, defined by the flows between the tasks and the off-chip communication, defined by the traffic to the memory controllers. To model the first term, we introduce the variables $h^x_{sd}$ and $h^y_{sd}$ that represent the hop-count of flow $f_{sd}$ in the x- and y-directions, respectively. Assuming minimal-path routing, the power consumption for inter-task communication can be defined as

$$P^t_{comm} = \sum_{f_{sd} \in \mathcal{F}} B_{sd} \cdot (h^x_{sd} + h^y_{sd}) \cdot E_{bit},$$

and the term related to communication with memory controllers

$$P^{mc}_{comm} = \sum_{t_i \in \mathcal{T}} \sum_{p_j \in \mathbb{P}} \sum_{v_k \in \mathcal{V}} a_{ijk} \cdot \Lambda(t_i) \cdot \texttt{McDist}(p_j) \cdot E_{bit},$$

where $E_{bit}$ is the estimated energy for transmitting one bit over a link. The objective of the problem is to minimize the total power:

$$\min : \quad P = P_{comp} + P_{comm} = P_{comp} + P^t_{comm} + P^{mc}_{comm}. \tag{6.1}$$

## Constraints

The first two constraints are the classical requirements for an assignment problem. Every task $t_i$ has to be assigned to some processor $p_j$ and every processor can hold one task at most:

$$\forall t_i \in \mathcal{T}: \quad \sum_{p_j \in \mathbb{P}} \sum_{v_k \in \mathcal{V}} a_{ijk} = 1. \tag{6.2}$$

$$\forall p_j \in \mathbb{P}: \quad \sum_{t_i \in \mathcal{T}} \sum_{v_k \in \mathcal{V}} a_{ijk} \leq 1. \tag{6.3}$$

The next step is to model the communication component of the power. A set of constraints is introduced to calculate the hop-count of each flow assuming an XY-routing. Each processor $p_j$ is located in a tile at column $x_j$ and row $y_j$ of the mesh (Fig. 6.2a). The coordinates are uniquely defined by the index $j$: $x_j = j \bmod W$ and $y_j = \lfloor j/W \rfloor$, where $W$ is the width of the mesh. For any task $t_i$, we define $(x_i, y_i)$ as the location of the processor assigned to the task. Then, the location is specified by the expressions over the task assignment variables:

$$\begin{aligned} x_i &= \sum_{p_j \in \mathbb{P}} (j \bmod W) \sum_{v_k \in \mathcal{V}} a_{ijk} \\ y_i &= \sum_{p_j \in \mathbb{P}} (\lfloor j/W \rfloor) \sum_{v_k \in \mathcal{V}} a_{ijk}. \end{aligned} \tag{6.4}$$

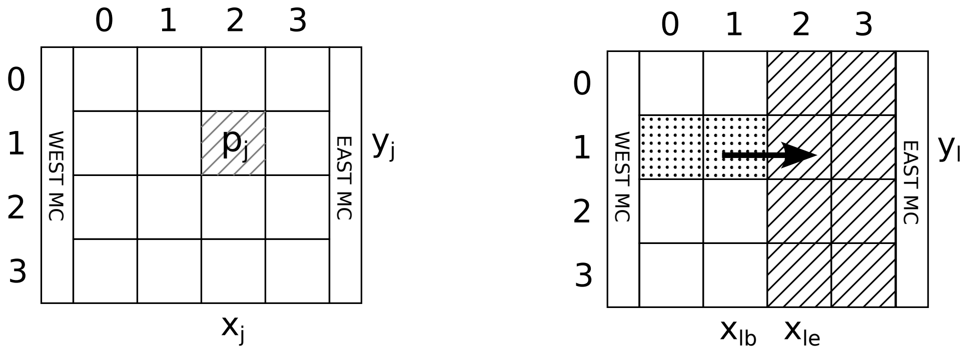
For every flow $f_{sd}$, the source and destination tasks, $t_s$ and $t_d$, are mapped onto processors $p_s$ and $p_d$, with coordinates $(x_s, y_s)$ and $(x_d, y_d)$, respectively, defined by (6.4). The horizontal hop-count, $h^x_{sd} = |x_s - x_d|$, and the vertical hop-count, $h^y_{sd} = |y_s - y_d|$ are defined by the following constraints[1]:

$$\begin{aligned} x_s - x_d \leq h^x_{sd}, \qquad x_d - x_s \leq h^x_{sd} \\ y_s - y_d \leq h^y_{sd}, \qquad y_d - y_s \leq h^y_{sd}. \end{aligned} \tag{6.5}$$

The next group of constraints defines the relations between voltage islands and throughput. Let the binary variable $v^n_k$ represent the fact that the voltage island $\iota_n$ operates at voltage $v_k$. First, for every island $\iota_n$ only one voltage has to be selected:

$$\forall \iota_n \in \mathcal{I}: \quad \sum_{v_k \in \mathcal{V}} v^n_k = 1. \tag{6.6}$$

[1] the pair of inequalities and the fact that the $h$ variables are implicitly minimized with the cost function (since this implies minimization of power), guarantee the equality with the absolute value.



(a) Processor $p_j$ located in the cell $(x_j, y_j)$ of the mesh.

(b) East link from cell $(x_{lb}, y_l)$ to cell $(x_{le}, y_l)$.

Figure 6.2: Definition of the processor and link location in mesh.

To enforce that all processors in the same voltage island work with the same voltage, the following constraint is added:

$$\forall \iota_n \in \mathcal{I},\ \forall v_k \in \mathcal{V}:\ \sum_{t_i \in \mathcal{T}} \sum_{p_j \in \iota_n} a_{ijk} \leq \mathtt{Num}(a_{ijk}) \cdot v_k^n, \tag{6.7}$$

where $\mathtt{Num}(a_{ijk})$ is the number of $a_{ijk}$ variables in the LHS of the inequality. Expression (6.7) in combination with (6.6) guarantees that only the assignment variables for the selected voltage may take non-zero values.

The throughput constraint should guarantee that for each task $t_i$ executed on processor $p_j$ the product of $\mathtt{IPC}(t_i, p_j)$ and the processor frequency $F(p_j, v_k)$ defined by the current voltage, is not less than the required throughput $\mathtt{IPS}(t_i)$. Hence, the following relation is specified for each $t_i \in \mathcal{T}$:

$$\sum_{p_j \in \mathbb{P}} \sum_{v_k \in \mathcal{V}} a_{ijk} \cdot \mathtt{IPC}(t_i, p_j) \cdot F(p_j, v_k) \geq \mathtt{IPS}(t_i). \tag{6.8}$$

The last group of constraints aims at satisfying the requirements for link capacity and memory controller bandwidth, under the assumption of XY-routing. We start by considering the link capacity. The total link bandwidth can be expressed as the sum of the bandwidths of all flows that pass through the link. There are two terms that contribute to link bandwidth, related to the inter-task and memory controller traffic, hence the constraint can be written as

$$\forall l \in \mathcal{L}:\quad \mathtt{TaskTerm}(l) + \mathtt{McTerm}(l) \leq \mathtt{Cap}. \tag{6.9}$$

Let us consider the task term first. In XY-routing, the data is always sent in the X-direction first and the Y-direction afterward. Hence, the route of a flow will pass through a link, only in case the source and destination tasks are mapped to a specific subset of processor locations. Thus, for every link $l$ and flow $f_{sd}$ we define the binary properties, $\texttt{MapSrc}(l, f_{sd})$ and $\texttt{MapDst}(l, f_{sd})$, that indicate whether the source and destination tasks are mapped onto the locations that imply link $l$ to be on the flow route.

To guarantee that $l$ is on the route of $f_{sd}$, both properties should be asserted, i.e., $\texttt{MapSrc}(l, f_{sd}) \cdot \texttt{MapDst}(l, f_{sd}) = 1$. This is a non-linear constraint that we linearize by introducing the real variables $m^l_{sd}$:

$$\texttt{MapSrc}(l, f_{sd}) + \texttt{MapDst}(l, f_{sd}) = m^l_{sd}. \tag{6.10}$$

Since the mapping properties can only take binary values, the $m^l_{sd}$ variable can only take three values: 0, 1, or 2. We use another scaling of $m^l_{sd}$ to the binary variables $r^l_{sd}$, that take non-zero values only when $m^l_{sd} = 2$, i.e. both mapping properties are true:

$$r^l_{sd} \geq m^l_{sd} - 1, \qquad 2 \cdot r^l_{sd} \leq m^l_{sd}. \tag{6.11}$$

The variables $r^l_{sd}$ are equal to 1 if the route of flow $f_{sd}$ goes through link $l$. Now the task term for link $l$ is written as:

$$\texttt{TaskTerm}(l) = \sum_{f_{sd} \in \mathcal{F}} B_{sd} \cdot r^l_{sd}. \tag{6.12}$$

Next we explain how to represent the mapping properties $\texttt{MapSrc}(l, f_{sd})$ and $\texttt{MapDst}(l, f_{sd})$. Consider the horizontal east link of a cell with coordinates $(x_{lb}, y_l)$ to a cell $(x_{le}, y_l)$ (Fig. 6.2b). The XY-route of flow $f_{sd}$ can only pass through the link in case the source task $t_s$ is mapped onto one of the two dotted processor cells. Indeed, the processor $p_s$ should be located on the same row and in a column that is lower than or equal to the origin of the link: $(x_s \leq x_{lb}) \wedge (y_s = y_l)$. Hence, the source mapping property for an east link is:

$$\texttt{MapSrc}(l, f_{sd}) = \sum_{\substack{p_j:\ x_j \leq x_{lb}, \\ y_j = y_l}} \ \sum_{v_k \in \mathcal{V}} a_{sjk}. \tag{6.13}$$

For the destination task $t_d$ the requirement is to be located in a column that is greater than or equal to the link endpoint $x_{le}$ (striped cells). The destination mapping property becomes:

$$\texttt{MapDst}(l, f_{sd}) = \sum_{p_j:\ x_j \geq x_{le}} \ \sum_{v_k \in \mathcal{V}} a_{djk}. \tag{6.14}$$

The mapping properties for south, west and north links are derived in a similar manner.

Now consider the term related to the memory controller traffic. For link $l$ let us denote $\texttt{Req}(l)$ the set of processors that send *requests* to their controllers through link $l$. Similarly, $\texttt{Rep}(l)$ is the set of processors that receive *replies* from controller through $l$. Hence, traffic to and from controllers through $l$ is defined by the rate of tasks mapped to the sets $\texttt{Req}(l)$ and $\texttt{Rep}(l)$:

$$\begin{aligned}\texttt{McTerm}(l) = &\sum_{t_i \in \mathcal{T}} \sum_{p_j \in Req(l)} \sum_{v_k \in \mathcal{V}} a_{ijk} \cdot \rho \cdot \Lambda(t_i) + \\ &\sum_{t_i \in \mathcal{T}} \sum_{p_j \in Rep(l)} \sum_{v_k \in \mathcal{V}} a_{ijk} \cdot (1-\rho) \cdot \Lambda(t_i). \end{aligned} \qquad (6.15)$$

The sets $\texttt{Req}(l)$ and $\texttt{Rep}(l)$ can be expressed similarly to the $\texttt{MapSrc}$ and $\texttt{MapDst}$ properties for links. As an example, let us consider the same east link (Fig. 6.2b). Assuming XY-routing, $p_j$ sends requests in the direction of the associated controller. Hence, the set of processors sending requests through the link is limited by those, associated with the *EAST* controller and located on the same row in the column that is lower or equal to the link origin:

$$\texttt{Req}(l) = \{p_j : (MC(p_j) = EAST) \wedge (x_j \leq x_{lb}) \wedge (y_j = y_l)\}.$$

The sets for other links are derived similarly. The inequalities (6.9) together with the scaling relations (6.10), (6.11) and definitions (6.12)-(6.15) guarantee that the link capacity constraints are met.

The last step is to specify the bandwidth constraints of the memory controllers. The bandwidth of controller $mc_\kappa \in MC$ is defined by the rates of tasks mapped onto processors, associated with $mc_\kappa$:

$$\forall mc_\kappa \in MC\colon \sum_{t_i \in \mathcal{T}} \sum_{p_j : MC(p_j) = \kappa} \sum_{v_k \in \mathcal{V}} a_{ijk} \cdot \Lambda(t_i) \leq \texttt{McBw}. \qquad (6.16)$$

## Problem formulation

The problem can now be formulated as follows.

***Minimize*:** power consumption (6.1)

***subject to*:**

*assignment* constraints and hop-count definition (6.2)-(6.5),

*voltage selection* constraints (6.6), (6.7),
*throughput* constraints (6.8),
*link capacity* constraints (6.9)-(6.15)
and *memory bandwidth* constraints (6.16).

## 6.3 Mapping by metaheuristics

This section discusses two metaheuristics commonly used to solve complex combinatorial problems: *Simulated Annealing* (SA) [80] and *Extremal Optimization* (EO) [24]. Both methods are inspired by equilibrium statistical physics. SA has been successfully applied in many EDA problems, mostly related to layout synthesis. However, EO has emerged as a very competitive alternative that can give superior results in quality and computational cost. This section shows how EO can be customized to effectively solve the task mapping problem. The results prove the superiority with regard to SA.

The SA and EO metaheuristics have already been considered in Chapter 5. In this section they are reviewed again for self-containment of the present chapter.

Both metaheuristics start from an initial mapping obtained by greedily placing the tasks with highest throughput to the fastest processors. It is assumed that the system is not highly throughput-constrained and that a feasible initial assignment can be achieved by a greedy heuristic. The bandwidth constraints may be violated in the initial mapping and will be handled during the optimization process.

### Simulated annealing

The general algorithm for SA is described in procedure 4. To evaluate every configuration, two functions are used. $Cost()$ returns the cost of a configuration, calculated as the total system power according to equation (6.1). $CapP()$ calculates the penalty for link capacity and memory bandwidth violations:

$$CapP = \sum_{l \in \mathcal{L}} \max\left(\frac{B_l - \mathtt{Cap}}{\mathtt{Cap}}, 0\right) + \sum_{mc \in MC} \max\left(\frac{B_{mc} - \mathtt{McBw}}{\mathtt{McBw}}, 0\right),$$

where $B_l$ is the total bandwidth of flows routed through link $l$ and $B_{mc}$ is the bandwidth of controller $mc$. If all constraints are satisfied, then $CapP = 0$.

The SA algorithm implements a conventional annealing schedule. Given the initial temperature $T_{init}$ and the cooling factor $\alpha$, a new solution ($NewSol$) is generated (line 4) and may be accepted probabilistically, depending on the current temperature $T_{cur}$ (line 5). The value of $T_{cur}$ decreases as the system evolves in time (line 8).

**Procedure 4** SimulatedAnnealing

```
$T_{cur} = T_{init}$
while improvement in last $k$ iterations do
  for P iterations do
    generate $NewSol$
    if Accept($CurSol$, $NewSol$) then $CurSol \leftarrow NewSol$
    if Cost($NewSol$) < Cost($BestSol$) then $BestSol \leftarrow NewSol$
  end for
  $T_{cur} = \alpha \cdot T_{cur}$
end while
return $BestSol$
```

**Procedure 5** Accept($CurSol$, $NewSol$)

```
1: $CurCost \leftarrow$ Cost($CurSol$) + $\lambda$ CapP($CurSol$)
2: $NewCost \leftarrow$ Cost($NewSol$) + $\lambda$ CapP($NewSol$)
3: if $NewCost < CurCost$ then return true
4: else return true with probability $P_a$
```

For every temperature, a number of moves that depends on the size of the system ($P$, that is the number of cells) is generated.

$NewSol$ is obtained by swapping a pair of random tasks and is accepted probabilistically according to Procedure 5. To penalize capacity violations, the cost is calculated as shown in lines 1-2, where $\lambda = T_{init}/T_{cur}$ is the weight for penalization, that grows in time. This decreases the probability to accept infeasible solutions as the simulation advances.

Solutions with better cost are always accepted (line 3), whereas worse-cost solutions are accepted with probability $P_a$:

$$P_a = e^{-\gamma}, \quad \gamma = \frac{NewCost}{CurCost} \cdot \frac{1}{T_{cur}}, \tag{6.17}$$

This probability depends on two factors. The former avoids the acceptance of solutions with high cost degradation. The latter increases the probability of hill climbing as the temperature cools down.

As in Chapter 5, the initial temperature, $T_{init}$, is calculated so as to obtain a high initial acceptance probability, such as $P_a = 0.95$. This is done by first evaluating an average ratio of the $NewCost$ and $CurCost$ during $P$ iterations ($P$ is the number of processors), and then using the obtained value in (6.17) to calculate the temperature for the desired value of $P_a$.

## Extremal optimization

This section proposes *Extremal Optimization* (EO) [24] for solving the task mapping problem. The idea of EO was introduced in Section 5.4. As a reminder, EO evolves by selecting against the worst components of the system. For the task mapping problem, EO evaluates the *fitness* of each task in the mapping configuration. A high fitness value indicates that the task has a comfortable low-cost status in the configuration. EO focuses on improving the status of tasks with low fitness.

At each iteration EO selects a pair of tasks to be swapped: an unfavorable task ($t_u$) and a replacement task ($t_r$). Unlike SA, EO uses information about the system cost when selecting the swapped tasks. This results into a faster progress towards the final solution. In addition, EO accepts new solutions unconditionally without depending on any temperature cooling schedule, thus making the algorithm easier to tune.

The mapping problem can be considered as a multiobjective optimization problem, since the $P_{comp}$, $P^{t}_{comm}$ and $P^{mc}_{comm}$ terms of the cost function (6.1) depend on weakly related voltage level and hop-count values. It was observed in [44] (and proved by our experiments) that the multiobjective EO operates better by interleaving the optimization of individual objectives in time, rather than trying to optimize all of them simultaneously. This suggests to introduce different fitness functions for the optimization of three power components and alternate them at different iterations of the algorithm.

The EO algorithm is outlined in procedure 6. After the definition of an initial solution (greedily), the execution is continued until no further improvement is observed during a certain number of iterations.

At the beginning of each iteration, local search is performed by sequentially swapping $P$ random pairs of tasks and accepting only those that improve the cost. This variant contributed to improve the cost of the final solution and the speed of the algorithm [140].

The core of the algorithm focuses on selecting the pair of tasks that must be swapped. The fitness functions alternate depending on the iteration number. In one case, fitness is oriented to improve the power consumption generated by inter-task communication, considering the hop-counts and bandwidth parameters. In the second case, the power of communication with memory controllers is optimized. The last case addresses the power generated by computations.

The first task, $t_u$, is selected by using the $\Phi_u$ fitness function and sorting the tasks according to the fitness value. The second task, $t_r$, is selected by ranking the task according to the improvement in cost that the swap would produce ($\Phi_r$ functions). The power law described by equation (5.6) is used to select the tasks randomly.

**Procedure 6** EXTREMALOPTIMIZATION

```
$CurSol \leftarrow BestSol \leftarrow$ "Some initial solution"
while some improvement in the last $k$ iterations do
    Local search: swap $P$ randomly selected pairs sequentially and
                  accept only those that improve the cost of $CurSol$
    if (iter $mod$ 3) = 0 then /* improve task comm. cost */
        sort all tasks in ascending order of $\Phi_{u,t}^{comm}$
        select $t_u$ according to equation (5.6)
        sort all tasks $t_i \neq t_u$ in ascending order of $\Phi_r^{comm}$
        select $t_r$ according to equation (5.6)
    else if (iter $mod$ 3) = 1 then /* improve mc comm. cost */
        sort all tasks in ascending order of $\Phi_{u,mc}^{comm}$
        select $t_u$ according to equation (5.6)
        sort all tasks $t_i \neq t_u$ in ascending order of $\Phi_r^{comm}$
        select $t_r$ according to equation (5.6)
    else /* improve comp. cost (iter $mod$ 3 = 2) */
        sort all tasks in ascending order of $\Phi_u^{comp}$
        select $t_u$ according to equation (5.6)
        sort all tasks $t_i \neq t_u$ in ascending order of $\Phi_r^{comp}$
        select $t_r$ according to equation (5.6)
    swap tasks $t_u$ and $t_r$ in $CurSol$
    if Cost($CurSol$) < Cost($BestSol$) then
        $BestSol \leftarrow CurSol$
end while
return $BestSol$
```

Finally the locations of tasks of $t_u$ and $t_r$ are swapped unconditionally and *BestSol* is updated if the cost is better than any other solution visited so far.

### Fitness functions

To model the fitness for the power consumption generated by the inter-task traffic on the mesh, $\Phi_{u,t}^{comm}$ ranks the tasks according to the product of total traffic and the square of hop-count of the involved flows:

$$\Phi_{u,t}^{comm}(t_i) = -\sum_{f_{sd}:(t_s=t_i)\vee(t_d=t_i)} B_{sd} \cdot (h_{sd}^x + h_{sd}^y)^2.$$

The square of hop-count tries to balance the length of the flows. It penalizes tasks with longer flows, rather than those with high bandwidth, since $B_{sd}$ is a constant parameter that cannot be changed. The selection of the ranked tasks tends to pick tasks with high communication cost. The negative sign allows to rank the tasks in ascending order of fitness.

Similar fitness is used to select unfavorable task $t_i$ mapped to processor $p_j$ for the controller-related term of power:

$$\Phi^{comm}_{u,mc}(t_i) = -\Lambda(t_i) \cdot \mathtt{McDist}(p_j)^2.$$

Although the fitness functions selected for both terms of communication power look similar, we consider them as individual candidates for multiproduct optimization. The intrinsic difference between the two types of communication is that an inter-task flow depends on mapping of both, source *and* destination tasks, while the memory controller flow depends on one, either source *or* destination task.

The fitness function for the replacement task $t_r$ is the same for both types of communication. It aims at selecting a task that, when swapped with $t_u$, would mostly decrease the communication cost and contribute to reduce the violations of maximum bandwidth:

$$\Phi^{comm}_{r}(t_i) = \mathit{Cost(NewSol)} \cdot (1 + \mathit{CapP(NewSol)}),$$

where *NewSol* is the solution obtained by swapping $t_i$ and $t_u$.

The computation-oriented fitness functions aim at finding power-efficient solutions by smoothing the *voltage spillover* in the voltage islands. Let us call $V_i^{\min}$ the minimum voltage required to guarantee the throughput of task $t_i$ assigned to a processor in some voltage island $\iota_n$. Since task is living in the same island with other tasks, it may not be possible to assign $V_i^{\min}$ to it, as other tasks may require a higher voltage.

We define the *voltage spillover* of $t_i$ as $\mathit{Spillover}_i = V_i^{\min} - \overline{V}$, where $\overline{V}$ is the average minimal voltage of all tasks allocated in the same voltage island. The dispersion of island $\iota_n$ is defined as

$$\mathit{Dispersion}_n = \sum_{t_i \in \iota_n} (\mathit{Spillover}_i)^2.$$

and measures the voltage imbalance for the island. High dispersions imply less power-efficient solutions, as more processors operate at voltages higher than required. Computational fitnesses aim at decreasing the voltage dispersion of the system. The unfavorable component is selected from the tasks with the high spillover value:

$$\Phi^{comp}_{u}(t_i) = -\mathit{Spillover}_i.$$

The replacement task is selected to maximize the product of the cost improvement with the dispersion, penalizing solutions with large capacity violations:

$$\Phi_r^{comp}(t_i) = \frac{1 + \mathit{CapP(NewSol)}}{\Delta \mathit{Cost} \cdot \Delta \mathit{Dispersion}}.$$

## 6.4 Experimental results

The results presented in this section have three primary objectives. Firstly, optimal solutions are obtained for small examples by solving the MILP model. It is shown that metaheuristics can also find the optimum for these examples, and in much shorter time. Secondly, the quality and speed of SA and EO are compared. The latter is demonstrated to outperform in both metrics for a vast space of solutions. Thirdly, the impact of the link capacity and memory bandwidth constraints is discussed.

### Examples and experimental setup

Every test case for the mapping problem is characterized by an application task graph and a target CMP. The parameters of the test cases are presented in Table 6.3. The number of tasks and flows are reported in the second and third columns. The fourth column shows the dimensions of the mesh for the target CMP. The last column displays the number of memory controllers in each test case.

The first group of examples is inspired by realistic applications, widely used in the SoC research domain (e.g. [115, 57]): *Multi-Window Displayer (MWD)*, *MPEG4 decoder (MPEG4)* and *Object Plane Decoder (OPD)*. To explore the scalability of the proposed technique, we generate a group of large examples for mapping onto 8 × 8, 12 × 12, 16 × 16 and 20 × 20-tile CMPs (test cases *64T* to *400T*). The task graphs for these configurations are obtained by combining instances of *MWD*, *MPEG4* and *OPD*. For instance, the task graph for *400T* consists of 30 small applications, 10 instances of each type. To avoid having totally disconnected clusters of tasks, few random flows were added between the components. The third column of Table 6.3 displays the resulting number of flows in graphs.

For the experiments, we have considered three processor classes (C1, C2 and C3) with different frequency and power parameters operating at three different voltages: 1.2V, 1.0V and 0.8V. The parameters are reported in Table 6.4. The distribution of tiles in the CMP is as follows: 20% of the tiles have C1-processors, 30% have C2-processors and 50% have C3-processors. The classes are distributed uniformly in such a way that all voltage islands have a similar mixture of classes. Without loss

| Name | # of tasks | # of flows | Grid size | # of MC |
|---|---|---|---|---|
| *MWD* | 12 | 11 | $4 \times 3$ | 2 |
| *MPEG4* | 12 | 13 | $4 \times 3$ | 2 |
| *OPD* | 16 | 17 | $4 \times 4$ | 2 |
| *64T* | 64 | 90 | $8 \times 8$ | 4 |
| *144T* | 144 | 200 | $12 \times 12$ | 4 |
| *256T* | 256 | 380 | $16 \times 16$ | 8 |
| *400T* | 400 | 595 | $20 \times 20$ | 8 |

Table 6.3: Testcase configurations.

| **Class** | **1.2V** | **1.0V** | **0.8V** |
|---|---|---|---|
| **C1** | 1000MHz, 260mW | 800MHz, 150mW | 600MHz, 70mW |
| **C2** | 450MHz, 200mW | 350MHz, 120mW | 250MHz, 60mW |
| **C3** | 160MHz, 55mW | 130MHz, 30mW | 100MHz, 15mW |

Table 6.4: Parameters of the processor classes.

of generality, we assume that all islands have the same size $S_{vi}$ (number of tiles). Different values have been used in the experiments.

Every task has a different throughput requirement (IPS) and a different performance when executed at each class of processor (IPC). All these values are defined randomly, with IPC values in the interval $[0.5, 2.0]$ and guaranteeing that a feasible mapping exists for the assigned performance and throughput requirements. This randomization contributes to explore a larger set of configurations and to have an unbiased tuning of the metaheuristics.

The traffic $\Lambda(t_i)$ between the task $t_i$ and memory controller was estimated as 20% of total traffic between $t_i$ and all other tasks. The ratio between the request and reply traffic was set to $\rho = 0.2$.

## Comparison with the optimal solution

The MILP formulation allows obtaining optimal solutions for the mapping problem. However, the search of the optimum is computationally affordable only for small examples. We used CPLEX [3] to solve the MILP problem for the test cases of the first group: *MWD*, *MPEG4* and *OPD*. The size of voltage islands $S_{vi}$ was set to four. The time required to find the optimum is displayed in the "MILP" column of Table 6.5. One can observe the two-order increase in runtime for a 16-tile example (*OPD*) in comparison with the 12-tile examples (*MWD*, *MPEG4*).

| Name | MILP | SA | EO |
|---|---|---|---|
| *MWD* | 85.25 | 0.01 | 0.01 |
| *MPEG4* | 120.17 | 0.02 | 0.01 |
| *OPD* | 4594.40 | 1.17 | 0.08 |

Table 6.5: Time to reach the optimal solution (sec).

The metaheuristics are able to achieve the optimal solution for the same examples in much shorter time (columns “SA” and “EO” of Table 6.5). In this experiment the SA and EO algorithms were run for a variety of parameters ($\alpha$ and $\tau$), and the best runtime values were selected. This comparison affirms the fact, that both metaheuristics perform very well for the small examples with known optimum.

## Simulated Annealing and Extremal Optimization: comparison

This section tries to give an apple-to-apple comparison of both metaheuristics for the task mapping problem. The comparison is illustrated using the *256T* example with $S_{vi} = 16$ and represents a typical behavior of the two algorithms for the explored test cases.

The timeout for execution was set to 200 seconds, since no significant improvements were observed after that time for both methods. Figure 6.3 depicts the evolution of the cost function value obtained by SA with various $\alpha$ and by EO with $\tau = 4.0$. The traces corresponding to higher values of $\alpha$ drop slower, but achieve better solutions in the long run.

Let us now consider the EO trace. At every moment in time the current solution found by EO is better than any of the SA solutions, obtained with different $\alpha$ values. The resulting cost discovered by EO upon timeout outperforms any of the SA solutions by 12%. Another important fact is that EO solution cost drops rapidly (0.1-2.5 seconds, depending on the test case), to the 10% of accuracy, with respect to the value obtained in the long run. This makes EO useful to apply when fast estimation of the cost is required, e.g. in exploration loops.

SA requires a careful tuning to eliminate the dependency of the $\alpha$ parameter on the problem size. Otherwise, small changes in $\alpha$ may lead to an important degradation in quality. In this work we do not aim at tuning the SA method. Rather, we perform multiple runs with different $\alpha$ values and select the best results. The aim is to show that EO is a better alternative even with a good tuning of SA.

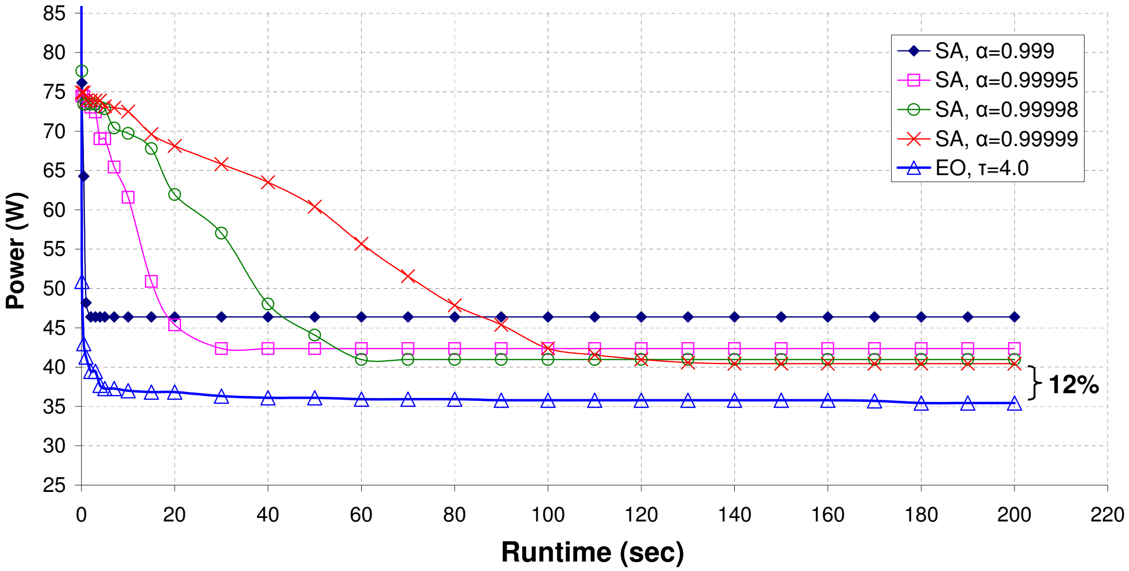


Figure 6.3: Evolution of SA and EO solutions in time.

In the experiments, it was also observed that EO is much less sensitive to $\tau$ and to the size of the problem. This simplifies the tuning of the algorithm. Note that some variation of $\tau$ may provide slightly better results for certain examples. However, we do not aim at demonstrating the highest improvement for all test cases. We prefer to emphasize that even having $\tau$ *fixed*, EO is able to outperform SA with *any* $\alpha$. Guided by this reasoning, in the following experiments we always define $\tau = 4.0$. This value was found to deliver good results for all test cases.

## Power optimization with EO

In this section we analyze the final solutions obtained with a timeout of 200 seconds. The goal is to study the reduction in power that EO delivers in comparison with SA for broad set of configurations. It is important to indicate that the results obtained by SA were selected by taking the best solution from all the $\alpha$ values, thus making the analysis independent of the cooling factor.

Three parameters are explored to obtain a comprehensive collection of test cases. Firstly, examples of different size are considered. These include the *64T*, *144T*, *256T* and *400T* configurations from Table 6.3. Secondly, for every test case the size of the voltage islands is varied among 4, 8 and 16 processors. Thirdly, different ratios between the computation power $P_{comp}$ and communication power $P_{comm}$ are considered. This is an important parameter, as it reflects the ability of the approach

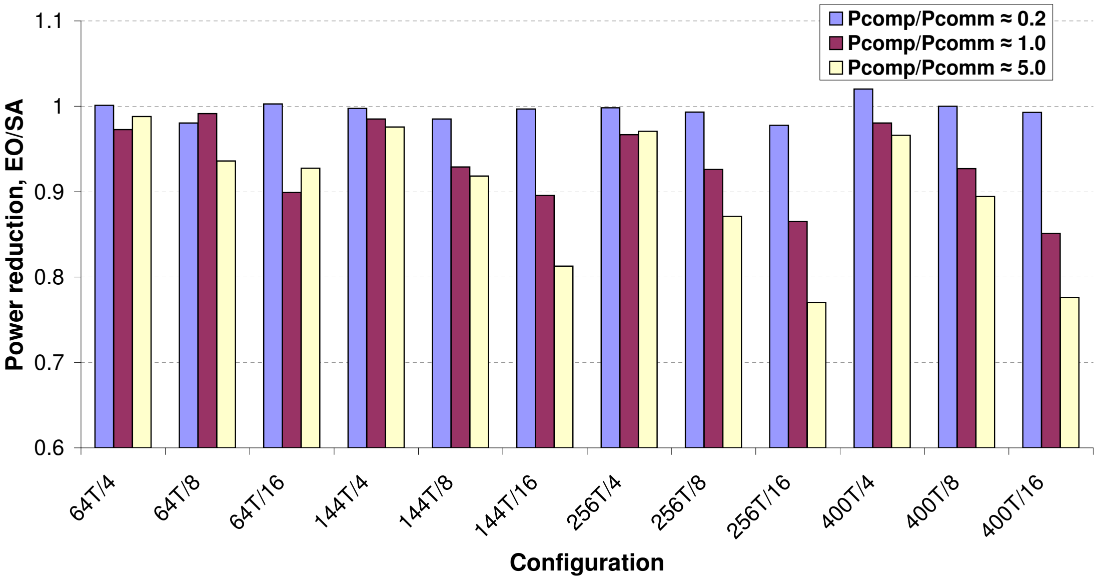


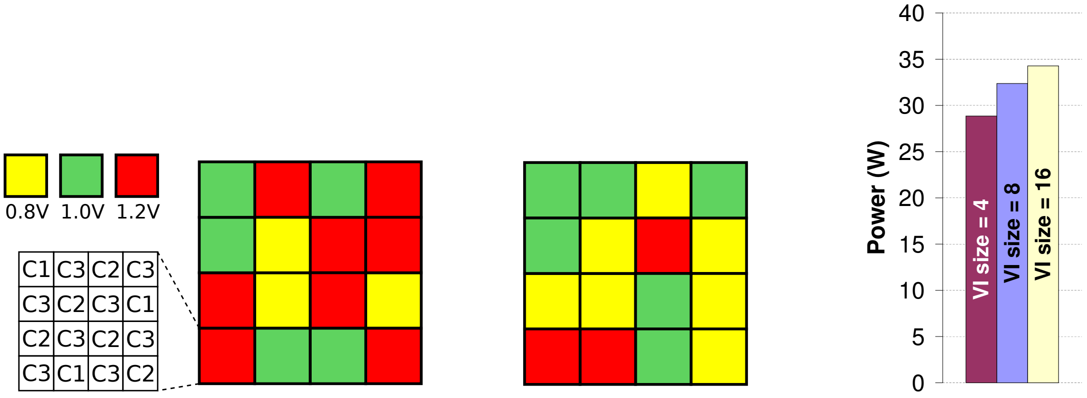
Figure 6.4: Power reduction by EO with respect to SA.

to give priority to one power component or improve both simultaneously. Three values for $P_{comp}/P_{comm}$ are explored: 0.2, 1.0 and 5.0. They and inspired by the results presented in [127].

Figure 6.4 reports the power (equation (6.1)) of the EO solution with respect to the best value obtained by SA with various $\alpha$. For each configuration, denoted as *testcase/*$S_{vi}$ along the X-axis, three values for different $P_{comp}/P_{comm}$ are shown. For the majority of configurations EO outperformed the results of SA, with a maximum gain in power of 22.5% (configuration *256T/16*, $P_{comp}/P_{comm} = 5.0$). Only for 3 of 36 explored configurations (*64T/4*, *64T/16* and *400T/4* with $P_{comp}/P_{comm} = 0.2$) EO was slightly worse than SA. The difference in this case did not exceed 2.0%.

EO tends to perform better at higher $P_{comp}$ as well as for larger values of $S_{vi}$. In other words, EO better minimizes the voltage of islands, due to the consideration of *voltage spillover*. As the island size grows, the amount of tasks, required to be swapped in order to improve the voltage, also increases. This is one of the important features of EO, since it can model the fitness of each component in the system. Modeling the voltage spillover in SA is difficult, since only a global cost is considered in the acceptance of moves and random swaps do not concentrate on the components with worst fitness.

As an example, Fig. 6.5 shows the final voltage distributions for the *256T* example with $S_{vi} = 16$. The system has 16 voltage islands and each island contains 16 processors with a mixture of C1, C2 and C3 classes, as shown in Fig. 6.5(a). The



(a) SA solution (b) EO solution (c) EO power

Figure 6.5: Voltage distribution and power for *256T* example.

final voltage assignment for each island is represented by the three colors in the figure. The solution obtained by SA has 8 islands at 1.2V, 5 at 1.0V and 3 at 0.8V. The one obtained by EO has 3 islands at 1.2V, 6 at 1.0V and 7 at 0.8V. The estimated power consumption of the EO solution is 12% smaller than the SA solution.

Another intuitive result is that the total power grows with the size of voltage islands (Fig. 6.5(c)). The island size sets the number of tiles that must run at the same voltage. Hence, larger islands imply less mapping flexibility for individual tiles to reduce voltage.

## Impact of link capacity and memory bandwidth

The link capacity and memory controller bandwidth constraints have a relevant influence on power consumption. In this example we fix the memory bandwidth and analyze how the solution changes as the link capacity constraint becomes more stringent. The dependency of power on the memory bandwidth has a similar trend. We use again the test case $256T$ with $S_{vi} = 16$ and set the bandwidth of each memory controller to 3 *Gbps*.

The results for SA and EO are plotted in Fig. 6.6. A sequence of solutions for different values of capacity constraints was obtained. The minimum capacity required to obtain feasible solutions was $Cap_{\min} = 0.91$ *Gbps*, and was reached by both methods. The trendlines included in the plot help to analyze the evolution of the solutions as the link capacity changes.

The tendency for power is to increase as capacity constraint tightens up. This happens principally due to the growth in communication cost as the tasks need to be spread to avoid congestion in the links.

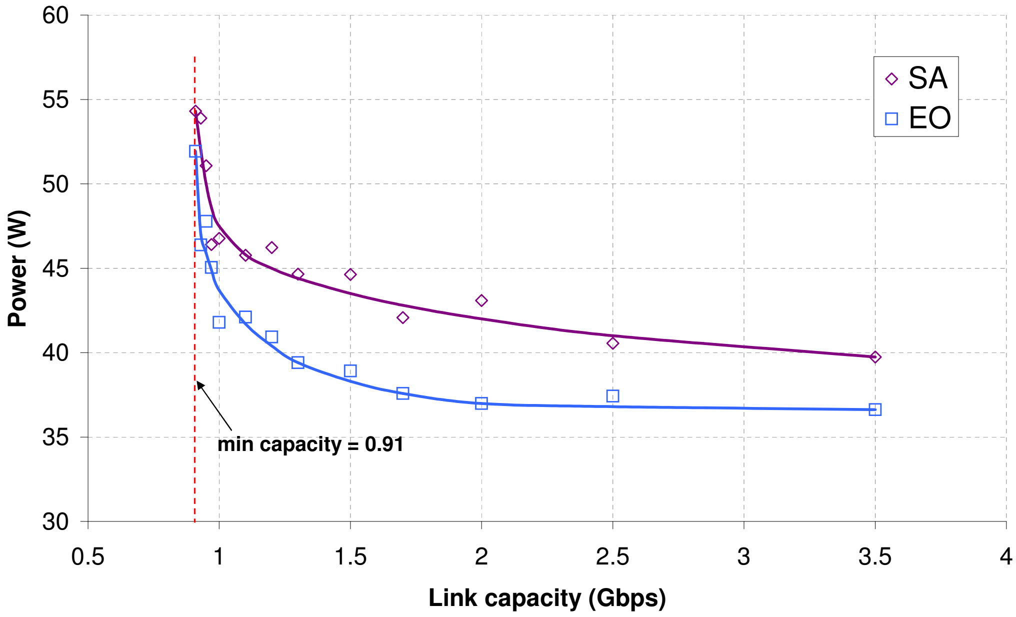


Figure 6.6: Impact of link capacity on system power.

Although the gap between the EO and SA costs decreases as the capacity approaches to $Cap_{min}$, EO wins in power for all considered values. This example represents the typical behavior, observed for both metaheuristics, when optimizing a configuration subject to capacity constraints.

## 6.5 Conclusions

This work has addressed the problem of static task mapping for large-scale tiled CMPs with multiple voltage islands, as one of the approaches to reduce design cost and time-to-market. The problem formulation considers task throughput requirements, on-chip and off-chip memory traffic, and bandwidth constraints.

Experimental results prove that:

- Metaheuristic-driven search efficiently solves the task mapping problem for CMPs with predefined voltage islands. The quality of the search is verified by the ability of metaheuristics to discover optimal solutions for moderate-size examples, for which the optimum is known by solving the MILP.

- EO outperforms SA in both, quality of the final solution and execution time. This fact makes EO very competitive in application to the mapping problem, comparing to the widely adopted SA metaheuristic.
- The proposed method is scalable, as demonstrated by performing task mapping of application graphs with hundreds of tasks onto large-scale CMPs.

The proposed approach has certain limitations. One of them is the requirement for each core to hold at most one task. To eliminate this constraint, the problem of *dynamic task mapping* or *scheduling* has to be addressed. The future work in this direction is summarized in Chapter 8.

# Chapter 7

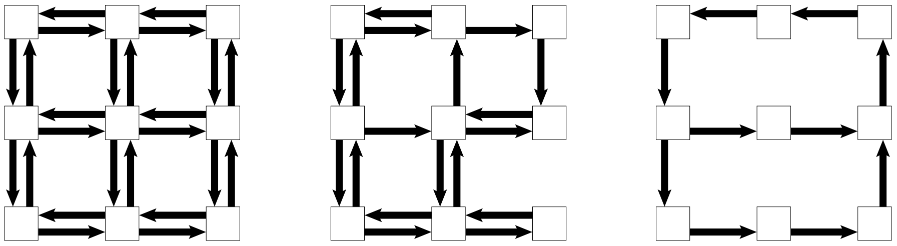
# Link Allocation for NoC Topologies

Networks-on-chip lie at the heart of modern on-chip systems, realizing communication between the components of many-core CMPs and ASICs. In order to make NoCs efficient in terms of performance and cost, a complex design process is required. This process demands for the completion of multiple tasks, including topology selection and mapping, physical planning, design of routing algorithms and switching schemes, and other optimization problems.

Individual tasks are traditionally addressed within a sequential flow, one after another, often leading to suboptimality of the final solution, due to the difference in optimization criteria. Hence, it is very desirable to develop methods which combine several tasks in one problem and explore the complete solution space in a way that all design constraints are met and the implementation cost is minimized.

This chapter proposes a model for simultaneous topology customization and route allocation for networks-on-chip. The presented approach can be applied to both, application-specific SoCs, characterized by the communication graph, and general-purpose CMPs, modeled with a complete communication graph and uniform traffic requirements. It focuses on the strategy for removing particular links from a complete regular topology [27], which is referred to as the *link allocation* problem for regular topologies. More precisely, link allocation can be defined as the problem of searching for a subset of links that satisfy the communication requirements of the system and minimizing the design cost.

The contribution of this research is a mathematical programming model for simultaneous link allocation and deadlock-free route assignment for on-chip interconnects, published in [98]. This model is capable of finding minimal deadlock-free routes for irregular topologies, which is an important problem for the interconnect power minimization [89].



(a) Full mesh (b) Partial mesh (c) Minimal strongly-connected mesh

Figure 7.1: Different link allocation solutions for a 3x3 mesh.

## 7.1 Model overview

This section gives an overview of the contributions of the work by using a simple example. In this work, we consider communication topologies that can be mapped onto a two-dimensional grid. Every link can connect two adjacent routers and the transit time through each link is constant and known a priori. We also assume that the system has already been floorplanned and each *PE* is connected to a router.

Figure 7.1 shows three different topologies based on an underlying 3x3 mesh. Figure 7.1a depicts a fully-connected mesh in which all possible links have been laid out. This topology can provide a very high performance. Every router can reach any other router in no more than four hops. However, this topology requires a costly implementation in wiring and router area. Every router implements a crossbar that has a quadratic cost on the number of links of the router.

On the other hand, Fig. 7.1c shows a mesh with the minimum number of links to preserve strong connectedness, which results in a very area-efficient implementation. However, the diameter of the network doubles since some routes may require eight hops to communicate one *PE* with another. Additionally, some links may become over-congested if they have to be shared among different routes that carry dense traffic, thus incurring in a significant throughput penalty due to the contention in the network.

The designer will probably want to find an intermediate topology that satisfies certain throughput constraints with a reduced implementation cost. Figure 7.1b depicts one of these solutions. The cost has been reduced by removing some of the links of the full mesh. However the diameter has already been increased (some routes require five hops). This may also involve extra congestion in some specific links.

The solution space of this type of problems is huge. It becomes even larger when we consider the route assignment problem. Figure 7.2a depicts the communication

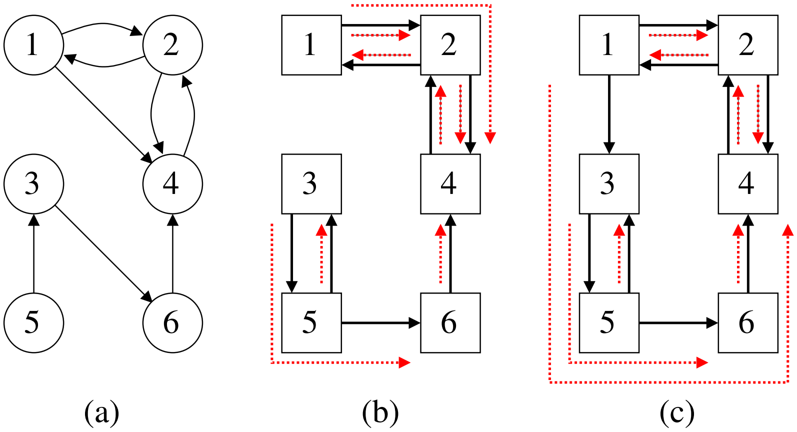


Figure 7.2: (a) Communication graph; (b) and (c) two different link allocation and route assignment solutions.

graph for six $PE$s that need to exchange information. In this particular example, every $PE$ is assumed to be attached to a router. Figure 7.2b shows a solution in which the links have been allocated to provide a minimum hop-count for every communication edge. The routes are represented by the dotted lines. The longest routes are for the pairs $(1, 4)$, with the route $1 \to 2 \to 4$, and $(3, 6)$, with the route $3 \to 5 \to 6$.

Let us assume that the traffic through the links $1 \to 2$ and $2 \to 4$ is very congested due to the intensive communication requirements of the $PE$s attached to those routers. Let us also assume that edge $1 \to 4$ is not critical and has a low priority. Hence, the designer might consider to deviate the traffic $1 \to 4$ through another route, as shown in Fig. 7.2c. This solution implies an extra link in the network, but may contribute to meet the throughput requirements of the system. Note that the route $1 \to 3 \to 5 \to 6 \to 4$ is not using the shortest path. The model presented in this work allows the exploration of non-optimal paths to alleviate the traffic in congested links.

Even though this work assumes a 2D grid as the underlying structure, the topology of the network is not limited to regular meshes. Irregular meshes with $PE$s exceeding the size of one tile can also be considered, thus providing an extra flexibility in the exploration of solutions. Furthermore, the links are not constrained to have equal length. This makes the design flow even more flexible in terms of floorplanning and placement.

### Sketch of the model

To explore the space of solutions, the mathematical model presented in this work introduces a set of constraints for link allocation, route assignment and deadlock avoidance. The set of constraints can be extended to cover other design criteria.

One of the most practically important constraints is the limitation of the number of ports in every router. As it will be discussed further, port limitation highly reduces the complexity of the router, thus saving area and power resources of the system. The traditional link capacity constraint is also supported. These two types of constraints are associated to *physical* requirements of the design.

To guarantee a sufficient *performance* in the system, a set of delay constraints are defined. These constraints are modeled as a maximum hop-count (structural latency) for each net[1]. The *communication* demands are defined by the bandwidth requirement between each pair of *PE*s.

An essential property of route assignment is *deadlock and livelock freedom.* The incorporation of turn prohibition constraints guarantees the absence of any deadlock and livelock in the explored solutions.

The mathematical model includes four optimization objectives in the cost function: the minimization of the number of links in the grid, the maximum hop-count over all nets, the total net delay (sum of all net delays) and the uniform traffic distribution. The first objective is related to the area optimization, whereas the other three objectives guide the search toward increasing the performance of the system.

The constraints of the model can be represented as linear inequalities with integer and real variables. In this way, the problem can be specified with a mixed-integer linear programming (MILP) or mixed-integer quadratic programming (MIQP) model, depending on the cost function. The term *mixed-integer programming* (MIP) will be used to refer to both problems interchangeably. Even though these are NP-complete problems, the experimental results show that optimal solutions can often be found with moderate computational cost. Section 7.3 will present results obtained from different benchmarks.

## 7.2 The integer programming model

This section presents the MIP model for link allocation and route assignment problem. Table 7.1 summarizes the input parameters for a quick reference. We introduce several types of variables and set notations to formulate the problem. The summary of the MIP notations can be found in Table 7.2.

[1] We refer to a net as a logical connection between two *PE*s, represented as an edge in the communication graph.

Table 7.1: Input parameters of the problem.

| Input | Description |
|---|---|
| $(x, y)$ | Grid size |
| $n$ | Number of nets |
| $B_k$ | Required bandwidth for net $N_k$ |
| $D_k$ | Maximum hop-count for net $N_k$ |
| $C_j$ | Capacity of link $L_j$ |
| $P_{i,in}$ | Maximum number of input ports for router $R_i$ |
| $P_{i,out}$ | Maximum number of output ports for router $R_i$ |

Table 7.2: Notation for the MIP problem.

| Notation | Type | Description |
|---|---|---|
| $L_j$ | Binary variable | Link presence in the solution |
| $L_j^k$ | | Link usage by net $N_k$ |
| $T_p$ | | Prohibition of turn $T_p$ |
| $D_{max}$ | Real variable | Maximum net delay |
| $\mathcal{I}(R_i)$ | Set | Incoming links of router $R_i$ |
| $\mathcal{O}(R_i)$ | | Outgoing links of router $R_i$ |

## Parameters and variables of the problem

The problem consists of defining a set of routes in a 2-dimensional grid structure that satisfies a set of physical and performance constraints. The routes must support the communication among the $PE$s of the system.

The grid structure with size $(x, y)$ is represented as a directed graph $G(\mathcal{R}, \mathcal{L})$. The vertices of the grid define a set of routers $\mathcal{R} = \{R_0, .., R_{r-1}\}$, where the total number of routers is $r = x \cdot y$. The edges of the grid define a set of uni-directional links $\mathcal{L} = \{L_0, .., L_{l-1}\}$, where the total number of links for a grid with size $(x, y)$ is calculated as $l = 2\,(x(y-1) + (x-1)y)$.

A global assumption about the grid is that every pair of neighboring routers may have up to two uni-directional links to send data in both directions. Each link $L_j$ has a maximum capacity parameter $C_j$ $(flits/cycle)$. It limits the amount of data that can be transmitted over the link in one cycle.

Another input of the problem is the underlying communication graph $GC(PE, \mathcal{N})$ that represents the logical connectivity of the network. Every vertex represents a processing element and every edge represents a logical connection between a pair of processing elements. Every edge in the set $\mathcal{N} = \{N_0, .., N_{n-1}\}$ is a net of the system. Each net $N_k$ has two associated parameters: the required bandwidth $B_k$ (flits/cycle) and a maximum delay constraint $D_k$ (hops) for the packet transmission from source to destination.

The additional parameters $P_{i,in}$ and $P_{i,out}$ specify constraints on the number of input and output ports for router $R_i$, respectively.

## Path selection constraints

We focus on the deterministic routing path selection, without considering path diversity mechanisms. The latter would allow multiple paths for a pair of communicating processors. On the contrary, we assume there is only one path to send data packets for each communicating pair. Path diversity is an option for the routing path selection task. Our assumption for considering only one path eliminates the need to perform packet ordering at the destination router.

We start the constraint set description with introducing the basic mechanism for path selection and link representation in the model. Any configuration for link allocation contains a subset of links from the full grid. The presence of each link in a configuration is represented by the set of variables $L_j$[2], i.e., $L_j = 1$ iff link $L_j$ is present in the configuration.

The routing paths for every net are represented by another set of binary variables $L_j^k$, specifying the fact net $N_k$ uses link $L_j$ in its routing path. As we show below, this provides high flexibility for the solution space as every net may be routed through any arbitrary subset of links.

To reduce the potentially large number of $L_j^k$ variables $(n \cdot l)$, it is possible to introduce rules that bound a routing region for a net. For example, we may not be interested in having long routing paths for the short nets with source and destination routers located at the neighboring grid vertices. In this case we may limit search region to be within few hops. An example is presented in Fig. 7.3. Net $N_k$ is connecting two neighboring nodes, located in the corner of the grid. By limiting the maximum path length to 5 hops, only 10 links are eligible for selection in the route (marked with dashed lines). The max-hop constraints contribute to significantly reduce the number of link variables.

The sets of variables $L_j$ and $L_j^k$ are both related to the selection of link $L_j$. The $L_j^k$ variable defines the relationship between a link and *a particular* net $N_k$, while $L_j$ defines whether there is *at least one* net $N_{k'} \in \mathcal{N}$ such that $L_j^{k'} = 1$. In other words, $\forall k : L_j^k \Rightarrow L_j$. Assuming both sets $L_j$ and $L_j^k$ consist of binary variables and

[2] For the sake of notation simplicity we use $L_j$ to denote both the variable and the link.

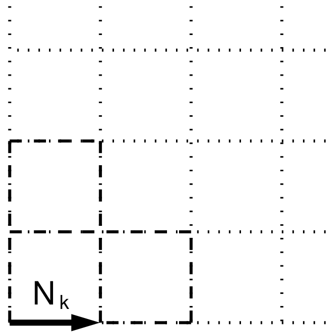


Figure 7.3: Path region limitation for $N_k$ with 5 hops.

$n$ is the total number of nets, we can write the following relations for each $L_j$:

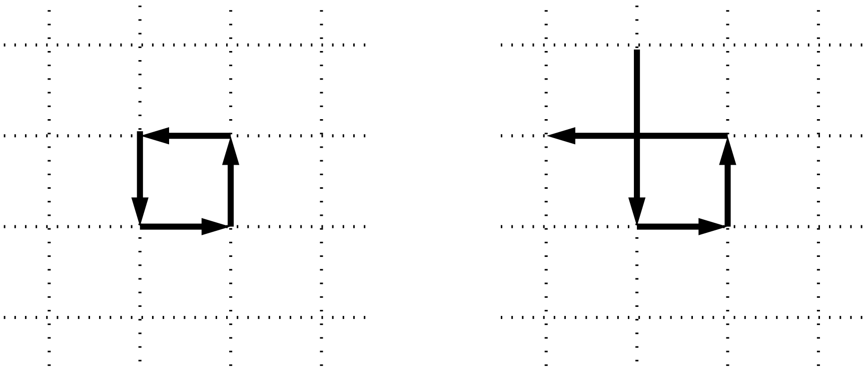
$$\begin{aligned} L_j &\leq \sum_{\mathcal{N}} L_j^k, \\ \sum_{\mathcal{N}} L_j^k &\leq n \cdot L_j. \end{aligned} \tag{7.1}$$

These two constraints guarantee the consistency of the variables from both sets in the MIP model.

We now formulate the set of routing constraints that allow one and only one path selection for each net. Let us denote the set of incoming links to the router $R_i$ as $\mathcal{I}(R_i)$ and the set of outgoing links as $\mathcal{O}(R_i)$. For each net $N_k$ with source at router $R_s$, destination at router $R_d$ and any intermediate router $R_i \in \mathcal{R} \backslash \{R_s, R_d\}$, the following constraints are defined:

$$\begin{aligned} \text{for } R_s &: \sum_{\mathcal{I}(R_s)} L_j^k = 0, \sum_{\mathcal{O}(R_s)} L_j^k = 1 \\ \text{for } R_d &: \sum_{\mathcal{I}(R_d)} L_j^k = 1, \sum_{\mathcal{O}(R_d)} L_j^k = 0 \\ \text{for } R_i &: \sum_{\mathcal{I}(R_i)} L_j^k = \sum_{\mathcal{O}(R_i)} L_j^k. \end{aligned} \tag{7.2}$$

The first two equations in (7.2) represent the boundary conditions on the path for the source and destination routers, while the last one can be treated as a path maintenance constraint for the intermediate routers. Indeed, the source router $R_s$ is the one that injects the $N_k$ packets into the network, thus there should be no input links to this router. The number of output links, carrying the $N_k$ packets from $R_s$ should be equal to one, as we allow only one path for each net. The inverse situation



(a) Closed path cycle.

(b) Open path cycle.

Figure 7.4: Path cycles introducing redundant links.

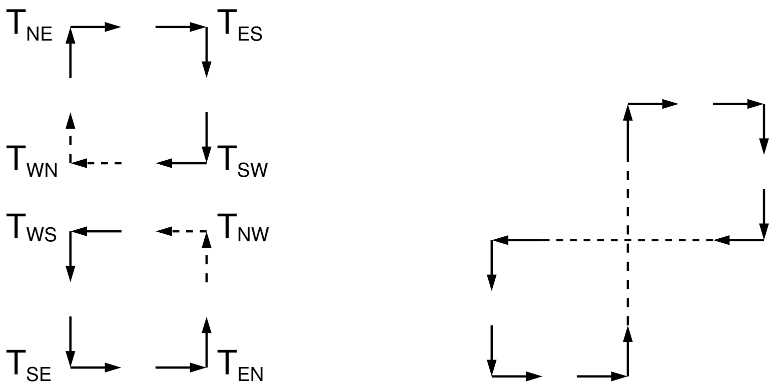
is observed at the destination router $R_d$, that consumes $N_k$ packets: there is one input link that delivers the $N_k$ packets to $R_d$, while the number of output links is zero. The last equation in (7.2) guarantees that if an intermediate router $R_i$ has an input link for $N_k$, then it will have an output link for this net. This condition assures that the path will be constructed correctly from source to destination node [132].

Note that the path constraints in (7.2) do not prevent the configuration from having cycles like the ones depicted in Fig. 7.4. Generally, a closed path cycle (Fig. 7.4a) may occur without breaking any constraint in (7.2), but allocating extra links. An open path cycle (Fig. 7.4b) may also occur, replacing one of the path turns with the sequence of three complementary turns and occupying redundant links. In the MIP model, these cycles can never appear due to the turn prohibition mechanism to avoid deadlocks (which will be described later in this section) and the cost function of the problem, that tends to minimize the number of links in the network.

For efficiency reasons, we have found interesting to add explicit constraints on the number of input (output) path links for each router:

$$\text{for } R_i \ : \sum_{\mathcal{I}(R_i)} L_j^k \leq 1. \tag{7.3}$$

Even though the overall number of constraints in the model increases, our experiments show that the problem is solved faster due to the limitation of the solution space. The constraints (7.3) avoid the exploration of solutions with redundant links that will never be optimal.



(a) 8 possible turns in 2D grid. Two turns, $T_{WN}$ and $T_{NW}$, depicted in dashed lines, are prohibited.

(b) The cycle that still occurs after prohibiting turns $T_{WN}$ and $T_{NW}$.

Figure 7.5: Turns in a 2D grid.

## Deadlock avoidance

Deadlocks and livelocks may occur in the wormhole routing networks due to the limited capacity of the router input buffers [39]. An important property of the routing algorithm is deadlock freedom. In deterministic routing, the propagation paths for every net are defined statically. Thus, deadlock freedom can be guaranteed by incorporating certain restrictions into the path selection procedure.

One of the approaches for deadlock and livelock avoidance is *turn prohibition* [59]. There are eight possible turns a packet may follow in a 2D grid (Fig. 7.5a). We refer to a turn according to the directions of the input and output links of the turn, namely: west-north (WN), north-east (NE), east-south (ES), south-west (SW) in the clockwise direction and west-south (WS), south-east (SE), east-north (EN), north-west (NW) in the counter-clockwise direction. In order to guarantee that deadlocks never occur, certain turns should be prohibited in both cycles (clockwise and counter-clockwise). Specifically, prohibition of one turn from each cycle is enough to assure that the cycles will not occur. However prohibition of some turn pairs will still allow deadlocks resulting from the complex cycles depicted in Fig. 7.5b. Luckily, these are just four pairs and they are easy to identify [59]. Thus, when prohibiting two turns, one from each cycle, we should check that they do not belong to the same pair.

Finally, we want to apply the turn prohibition mechanism to guarantee deadlock freedom in the MIP model. We introduce a set of binary turn variables to represent each one of the 8 possible turns: $\{T_{WN}, T_{NE}, T_{ES}, T_{SW}, T_{WS}, T_{SE}, T_{EN}, T_{NW}\}$. For example, the WN turn will be prohibited in the final solution if and only if $T_{WN} = 1$. We formulate three sets of the turn constraints, based on the considerations above.

First, we have to guarantee that one turn is removed from each of the two potential cycles (Fig. 7.5a), that is

$$\begin{aligned} T_{WN} + T_{NE} + T_{ES} + T_{SW} = 1, \\ T_{WS} + T_{SE} + T_{EN} + T_{NW} = 1. \end{aligned} \tag{7.4}$$

Second, the excluded turns should not belong to the same pair that still allows complex cycles (Fig. 7.5b):

$$\begin{array}{rcl} T_{WN} + T_{NW} & \leq & 1, \\ T_{NE} + T_{EN} & \leq & 1, \\ T_{ES} + T_{SE} & \leq & 1, \\ T_{SW} + T_{WS} & \leq & 1. \end{array} \tag{7.5}$$

To ensure that none of the selected paths incorporates a prohibited turn, we should guarantee that from each pair of links, that contribute to the turn, at most one link can be selected for the net path. We need to formulate these constraints with the net-related variables $L_j^k$.

Two neighboring links may exist in the solution independently, but the turn will occur only when there is a net that traverses these links in sequence. This idea is illustrated with the examples in Fig. 7.6. On the left example, two intersecting nets are depicted: $N_1$ propagating from south to north and $N_2$ from west to east. All four links exist in the routing solution:

$$L_{north} = L_{east} = L_{south} = L_{west} = 1.$$

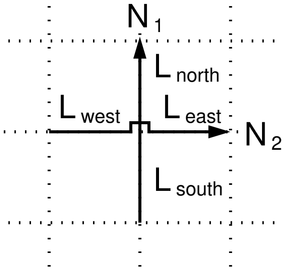
However the solution does not contain any turn. This fact is reflected by the net-related variables that take the following values:

$$\begin{array}{l} L^1_{north} = 1,\ L^1_{east} = 0,\ L^1_{south} = 1,\ L^1_{west} = 0, \\ L^2_{north} = 0,\ L^2_{east} = 1,\ L^2_{south} = 0,\ L^2_{west} = 1. \end{array}$$

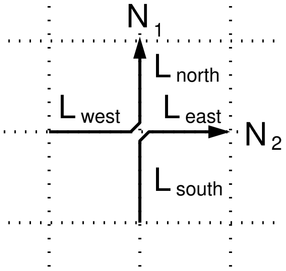
None of the pairs $\{L^k_{west},\ L^k_{north}\}$ or $\{L^k_{south},\ L^k_{east}\}$ has both variables set to 1 (that would describe a turn condition). On the right example, two touching nets are shown: $N_1$ propagating from west to north and $N_2$ from south to east. All the four links are still present, but the net-related variables have different values now:

$$\begin{array}{l} L^1_{north} = 1,\ L^1_{east} = 0,\ L^1_{south} = 0,\ L^1_{west} = 1, \\ L^2_{north} = 0,\ L^2_{east} = 1,\ L^2_{south} = 1,\ L^2_{west} = 0. \end{array}$$

In this case one turn is introduced by each net: an east-north turn $T_{EN}$ by $N_1$ and a north-east turn $T_{NE}$ by $N_2$. In the MIP model, this fact is observed by obtaining two pairs of non-zero variables: $\{L^1_{west} = 1,\ L^1_{north} = 1\}$ and $\{L^2_{south} = 1,\ L^2_{east} = 1\}$.



(a) No turn in case of intersection.



(b) Turns exist in case of touching.

Figure 7.6: Turn existence in dependence of nets positioning.

Therefore, in order to exclude a turn from the solution, we must prevent all contributing link pairs from having both variables set to 1. More formally, if turn $T_p$ is prohibited, then for any net $N_k$ and any pair of links $L_j$ and $L'_j$ that contribute to the turn, the following implication is required: $T_p \Rightarrow \neg(L_j^k \wedge L_{j'}^k)$, that is equivalent to $\neg(T_p \wedge L_j^k \wedge L_{j'}^k)$. This represents the third set of the turn prohibition constraints for the problem, that we can formulate in the following manner. For every net $N_k$, turn $T_p$ and all pairs of links $L_j$ and $L'_j$, that form $T_p$:

$$T_p + L_j^k + L_{j'}^k \leq 2. \tag{7.6}$$

The constraints (7.4), (7.5) and (7.6) are sufficient for the MIP model to ensure a deadlock and livelock-free solution.

## Port limitation constraints

A new design option introduced in this work is the constraint on *port limitation.* A typical router for a 2D grid network has input (I) and output (O) ports in 5 directions, i.e. 10 ports in total (Fig. 7.7). However the router complexity highly depends on the number of ports. For instance, the size of the internal crossbar grows quadratically with the number of ports, contributing to the overall area and power consumption of the network. Also by constraining the number of ports, the physical design of the router and the routing of the wide links become easier. Finally, few-ported routers can often be implemented with single cycle latencies, low area and short cycle time. Many-ported routers often require a trade-off between latency, cycle time and area. By considering the use of few-ported routers, it is possible to have a global view of the optimization problem since we are not a priori restricted to the larger areas and latencies inherent in many-ported routers. Thus, it is useful for the designer to have the capability of limiting the number of ports for each particular router.

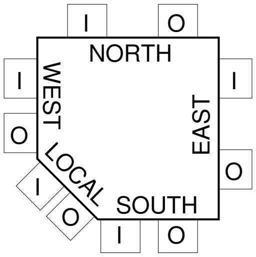


Figure 7.7: NoC router with I/O ports in 5 directions.

The MIP model can be easily extended with port limitation constraints. These limitations can be reduced to limitations on the number of links each router is connected, since each link is connected to a port of the router. The constraints may also distinguish between input and output ports. Let the limits for the number of input and output ports of the router $R_i$ be $P_{i,in}$ and $P_{i,out}$, respectively. Let us also introduce an indicator function $PE(R_i)$, that is equal to one if router $R_i$ has a processing element connected to it, or zero otherwise. If $PE(R_i) = 1$, then a local port connection exists and the port limitation should be decreased by one. We have the following set of constraints for each network router $R_i$:

$$\begin{aligned} \sum_{\mathcal{I}(R_i)} L_j &\leq P_{i,in} - PE(R_i), \\ \sum_{\mathcal{O}(R_i)} L_j &\leq P_{i,out} - PE(R_i). \end{aligned} \tag{7.7}$$

## Link capacity constraints

Another set of constraints in our model refers to the link *capacity*. A link $L_j$ can support a bandwidth up to $C_j$ flits per cycle. The bandwidth of each link is one of the input parameters of the problem.

As one physical link may be used by several nets, the total traffic in the link will be defined by the sum of the bandwidths of all nets that are routed through the link. The net-related variables $L_j^k$ can be used to define whether the path of net $N_k$ uses the link. The following constraint guarantees that the total traffic in the link does not exceed the link capacity:

$$\sum_{\mathcal{N}} L_j^k \cdot B_k \leq C_j. \tag{7.8}$$

## Net delay constraints

The proposed model offers a high flexibility in the selection of the routing path for any net as there are no limitations on the path shape or length. However, designers might be interested in limiting the path hop-count. This may be especially important for time-critical nets or some short nets that we want to prevent from having very long paths. In other words, we want to introduce performance constraints that approximate the *net delay* by the hop-count metric. This simple metric enables us to use the MIP formulation, and yet at the same time accurately captures latency for low traffic loads. The hop-count of a net can be calculated as the sum over all net-related link variables, since only the links with $L_j^k = 1$ contribute to the path. Given a limit $D_k$ for the hop-count of net $N_k$, we obtain the following constraint for the delay of net $N_k$:

$$\sum_{\mathcal{L}} L_j^k \leq D_k. \tag{7.9}$$

## Cost functions

A variety of cost functions are introduced to find solutions with different optimization criteria. These cost functions are further discussed in the experimental section. The first three cost functions are linear, so the obtained problem is classified as a Mixed-Integer Linear Programming (MILP) problem, while the last cost function is quadratic, resulting into a Mixed-Integer Quadratic Programming (MIQP) problem.

The cost functions do not need to be used in isolation. Linear combinations with weighted coefficients can be used for a multiple cost optimization.

### Number of links

The total number of links is obtained by summing variables $L_j$ over the set $\mathcal{L}$:

$$\min \sum_{\mathcal{L}} L_j. \tag{7.10}$$

Solving the MIP problem with the cost function in the form (7.10) tends to find a feasible routing solution with minimized area and power consumption.

### Maximum net delay

We may want to minimize the maximum net delay over all nets in the network in order to find a feasible routing solution with the highest performance (net delay constraints for particular nets may still be incorporated). The introduction of a new variable to represent the maximum delay, $D_{max}$, is required:

$$\sum_{\mathcal{L}} L_j^k \leq D_{max}. \tag{7.11}$$

and the cost function is simply:

$$\min D_{max}. \tag{7.12}$$

### Total net delay

The minimization of the total delay over all nets (that is equivalent to minimizing the average net delay) can be regarded as another performance metric. The cost function in this case is obtained by summing all the net-related variables $L_j^k$ for all nets from $\mathcal{N}$:

$$\min \sum_{\mathcal{N}} \sum_{\mathcal{L}} L_j^k. \tag{7.13}$$

### Uniform traffic distribution

this cost function aims at assigning a uniformly distributed traffic over all the links of the network. The distribution tends to decrease the contention delays in the network and, hence, increase the overall network performance by improving the throughput and the average packet delay. This cost function introduces quadratic terms, so the problem becomes an IQP problem. A more uniform distribution is obtained by minimizing the sum of the squares of the link traffics:

$$\min \sum_{\mathcal{L}} \left( \sum_{\mathcal{N}} B_k \cdot L_j^k \right)^2. \tag{7.14}$$

## Problem formulation

Having discussed the set of the constraints and the cost functions, we are now ready to present the formulation of the MIP problem for link allocation and route assignment:

***Find***

the optimal value for a cost function obtained as a linear combination of (7.10), (7.12), (7.13) and (7.14)

***subject to***

*physical* constraints (7.7),
*application-specific communication* constraints (7.8),
*performance* constraints (7.9),
*deadlock-avoidance* constraints (7.4), (7.5), (7.6)
and additional model constraints (7.1), (7.2), (7.3), (7.11).

Note that the user is free to select a subset of constraints if certain features are not required for the design. The cost function can also be extended with small effort due to the flexibility of the model.

## 7.3 Experimental results

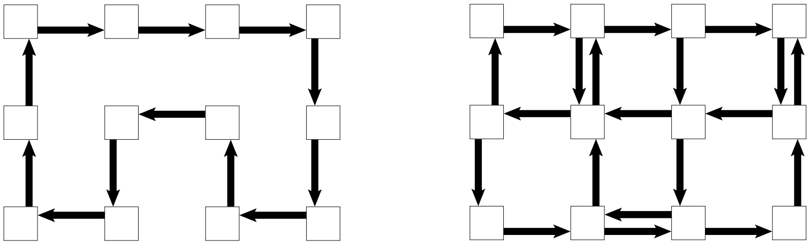
This section presents the experimental results to demonstrate the functionality and the quality of the proposed MIP model. We define the trade-offs and discuss the results for several design problems with various constraint sets and optimization objectives.

We use CPLEX [3] to solve the MIP model and an accurate flit-level C++ simulator with a variety of routing schemes to obtain the network parameters. Different testcases are used throughout the experiments: we start with artificial configurations and we next consider a testcase with typical server workloads from the SPEC2006 benchmarks.

Three types of the experiments are introduced. First, the area-performance trade-off is analyzed. Second, the application of the model for performance optimization by means of the routing paths redistribution is considered. Finally, the use of port limitation constraints for design exploration and tuning is presented.

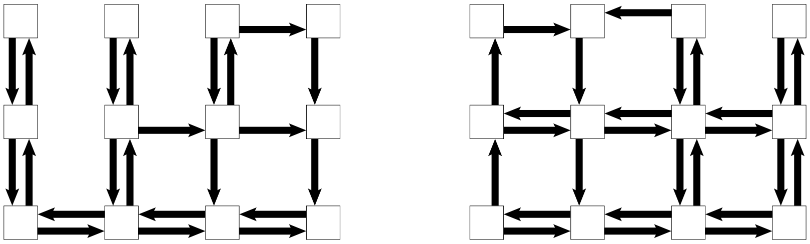
### Area-performance trade-off

One of the optimization tasks in the design of multiprocessor interconnection network is the minimization of the number of links. Given a set of constraints, the goal is to find the minimal number of links that satisfy the constraints, determine the link allocation and assign the traffic routes. This optimization contributes to decrease area and leakage power. However, the average hop-count delay may increase as the number of the links decreases and the packets have to follow longer roundabout paths. For this reason, the dynamic power may also increase. Note, that the variation in power will be defined by the relation between leakage and dynamic power.



(a) Minimal link number (b) Minimal hop-count

Figure 7.8: Link allocation solutions for a 4x3 network.



(a) Minimal link number (b) Minimal hop-count

Figure 7.9: Deadlock-free link allocation for a 4x3 network.

This set of experiments demonstrates the ability of the model to explore the area-performance trade-offs by link reallocation. Given the communication graph, we first search for the minimal number of links to enable the connectedness of all routers and estimate the maximum net delay value. Additionally, we investigate minimal link solutions subject to the limitation on the maximum net delay. In order to find the minimal link allocation, we solve the problem with the cost function in form (7.10). We apply the constraints (7.9) to limit the net delay and (7.4)-(7.6) to guarantee the deadlock freedom.

We show the area-performance trade-off for a 4x3 network with a complete communication graph, i.e. with net between every pair of processing elements. The minimal number of links required to connect all routers is 12 and forms the unidirectional ring topology, that is depicted in Figure 7.8a. For this allocation the packet delivery will take up to 11 hops for some nets. However, the diameter of the network can be reduced to 5 hops. The solution obtained with this delay limitation is presented in Fig. 7.8b. It incorporates 20 links, but the delay for any net is now guaranteed not to exceed 5 hops. Obviously, there is a trade-off between these two cases. We explore it by varying the delay constraint value in the specified range. The set of solution points is displayed in Fig. 7.10 ("Minimal"). Based on this dependency, one can determine the minimal number of links to guarantee a particular network diameter (for example, 14 links are required for the 8-hop network).

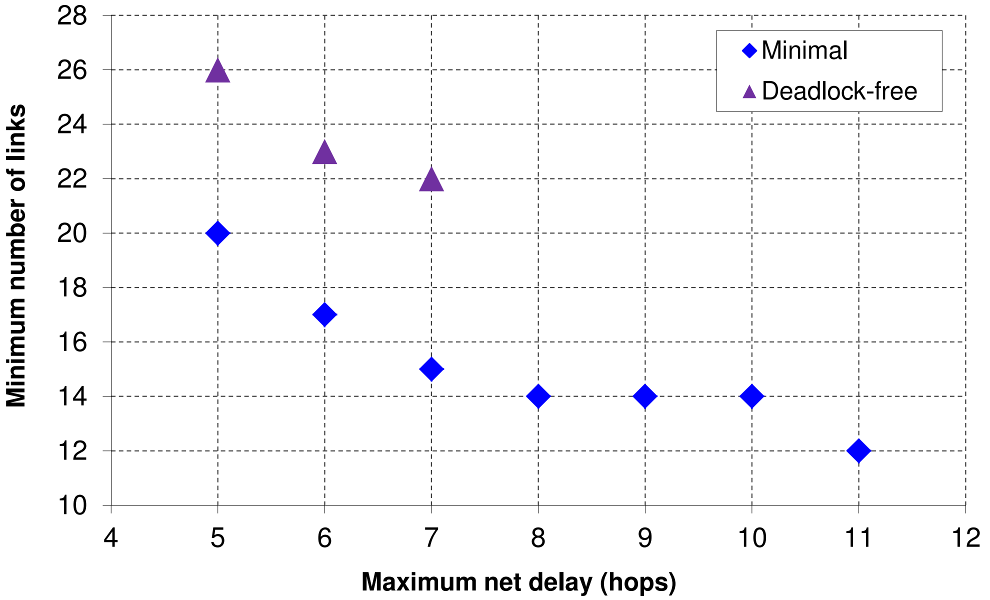


Figure 7.10: Area-performance trade-off points in terms of the link number and net delay for a 4x3 network.

Another important property is deadlock freedom. We provide a similar function after incorporating the turn prohibition constraints into the problem. Due to the extra limitations in routing paths, deadlock-free solutions tend to include more links for the same hop-count limit. Thus, the minimal number of links to provide full connectivity is 22 (Fig. 7.9a) and the solution with minimal delay of 5 hops has 26 links (Fig. 7.9b). This trade-off is also depicted in Fig. 7.10 ("Deadlock-free").

Finally, we note that even the "Minimal" solutions can be designed to be deadlock-free by choosing a suitable architecture. For example, the solution shown in Fig. 7.8a can be realized in practice by using a token-ring architecture [47] or virtual channels and dateline scheme [39]. Our model is also capable of discovering well-known structures, such as the bi-directional ring, that is seen in the variety of cell processors [90]. This proves that the class of the generated solutions is actually used in practice.

The area-performance trade-offs discussed in this section demonstrate the suitability of the model for design exploration and optimization. By incorporating additional application constraints, the user is allowed to perform more accurate, application-specific optimizations.

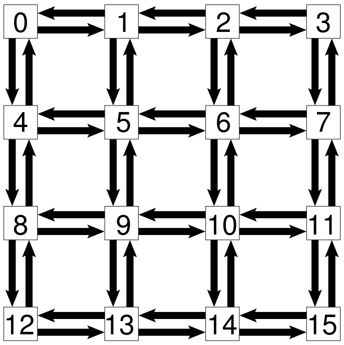


Figure 7.11: Underlying mesh for a typical server workload testcase.

## Performance optimization by route reassignment

Another application of the model is related to the optimization of the network delay by routing path redistribution. In this experiment we assume the communication requirements of the network, including the nets and their bandwidths, are specified. The objective is to minimize the average packet delay of the network. The average optimal hop-count delay can be obtained with the cost function (7.13), but the contention affects the delay value significantly once the network enters the saturation region. However, the contention delays can be alleviated by distributing the traffic uniformly over the network. For this purpose, we are using the cost function (7.14) to select routing paths that distribute the traffic more uniformly. The quality of the solution is estimated by simulation and compared to that of the XY and odd-even routing algorithms. Using the example of a typical server workload, we demonstrate that the obtained solutions improve the network delay as compared to the classical routing algorithms for a wide range of injection rates. Furthermore, the throughput increases as the saturation occurs at higher injection rates.

In this experiment we are considering the typical server workload traffic pattern collected using the SPEC2006 benchmarks. A 16-core application is assumed to be mapped onto a 4x4 full mesh, with all links present (Fig. 7.11). The cores connected to the routers 1 and 2 are the memory controllers that receive high traffic from the other cores. The traffic injected by each core is assumed to have Poisson distribution.

In Fig. 7.12, the comparison for the average delay estimation at different traffic rates is shown. We draw the packet delay as a function of the total injection rate to the network (in packets/cycle), for each of the three mentioned routing algorithms: XY, odd-even and the one using routing tables, based on the MIP solution. The timeout for the MIP solution was set to 1000 seconds. Simulation shows that the average packet delay, obtained with the MIP routing, is better than the delays of XY or OE schemes in the large range of injection rates. The XY-routing is only winning slightly the MIP configuration when the injection rates are small, as contention is low and the XY scheme results into the most uniform solution. However, as soon as

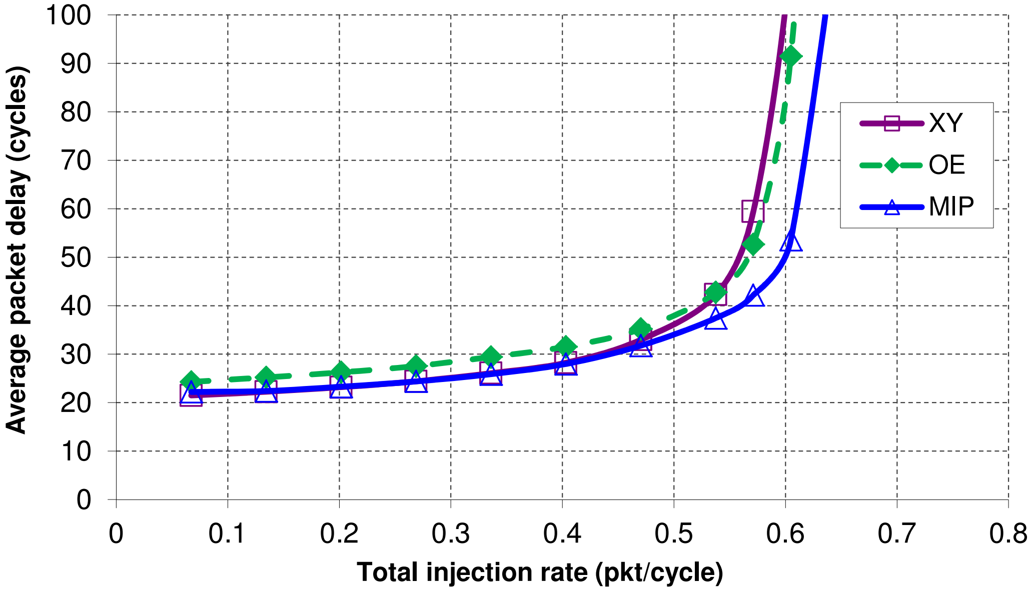


Figure 7.12: Average delay depending on the injection rate.

contention effects start to contribute significantly to the delay value (injection rates ≥ 0.3 pkt/cycle), the MIP routing improves the average packet delay, as compared to both XY and odd-even schemes. It can be also seen from the graph that the saturation occurs at higher rates, hence, the network throughput is increased.

## Design optimization by port limitation

Port limitation is another feature introduced by the model in order to extend the user design flexibility. The ability to limit the number of ports provides the means to account for the router design complexity at the network planning stage. Typical routers with 5-in and 5-out ports (Fig. 7.7) have complex designs and are not capable of running the full bandwidth. Hence, a mismatch between the network floorplanning stage and the router functionality appears, resulting in a potential loss of performance. By limiting the maximum number of ports of the routers, the use of complex routers during the network planning is avoided. This limitation also allows the optimization of area and leakage power.

We use a simple intuitive model to measure the area variation of the components in the network. We assume that the major network components are the links and the routers. The total link area is proportional to the number of links that is obtained from the MIP solution. The area of the router can be approximated by the complexity of the crossbar and buffer area [48]. The crossbar area has a quadratic dependency on the number of ports, while the buffer area dependency is linear.

We assume that the leakage power is proportional to the network area. In order to estimate the variation of the delay and the dynamic power of the solution, we use a simulator with the incorporated Orion power model [129].

The same typical server workload example of the system, mapped to the 4x4 network, will be used to demonstrate the port limitation functionality. We perform a set of experiments, aimed at finding the optimal route assignment and link allocation, subject to additional limitations on the maximum number of input and output ports of the routers. Further we estimate the network parameters and make comparison to the results obtained for the full mesh solution with XY-routing.

In these experiments, the number of ports includes the local connections to *PE*s (see discussion of (7.7)). In the full mesh solution, there are no limitations on the number of router ports. The largest routers with 5 input and 5 output ports appear in locations 5, 6, 9 and 10 (Fig. 7.11). Table 7.3 shows the results of solving the route assignment and link allocation problem with the number of input, output or both types of ports limited to 4. Each row is related to a different experiment with a particular port limitation. The first two columns of the table represent the maximum number of ports that a router may have. The values in the following columns are normalized to those obtained for the full-mesh solution with XY-routing. Thus, the ratio of the total link, crossbar and buffer area is reported in columns from third to fifth. The average packet delay and dynamic power are reported in the last two columns.

An example of the network layout for the experiment with 4-input and 4-output port limitation (4th experiment in Table 7.3) is depicted in Fig. 7.13. This layout contains 42 links instead of the 48 links in the full-mesh solution. The total area of links, crossbars and buffers in the presented solution has been decreased by 12.5%, 13.7% and 6.4%, respectively. One can observe the average packet delay increase by 9.9% as well as the dynamic power increase by 6.4%. The increase of average delay, unless the contention is high, is caused by the removal of links, since the minimal path for the neighboring routers rises to 3 hops. However, the increasing delay

Table 7.3: Port limitation results for server workload testcase.

| Port limit | | Area | | | Average | Dynamic |
|---|---|---|---|---|---|---|
| in | out | link | xbar | buffer | delay | power |
| 5 | 5 | 1.000 | 1.000 | 1.000 | 1.003 | 1.001 |
| 5 | 4 | 0.916 | 0.928 | 0.969 | 1.037 | 1.027 |
| 4 | 5 | 0.916 | 0.928 | 0.969 | 1.040 | 1.037 |
| 4 | 4 | 0.875 | 0.863 | 0.936 | 1.099 | 1.064 |

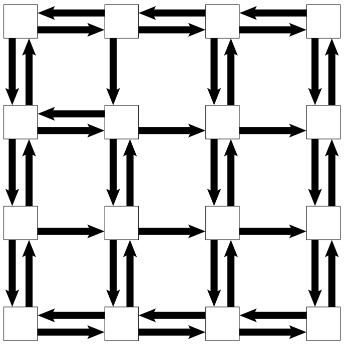


Figure 7.13: Network layout with 4 input and 4 output port limitation. Links connecting routers with the co-located *PE*s are not shown.

may be an acceptable solution if certain nets are not critical (see discussion of the example in Fig. 7.2). Otherwise, a designer is allowed to put a limiting hop-count constraint for the critical nets. The variation in power should be calculated together with the leakage power decrease, that is correlated with the network area. The total power estimation is technology dependent, but due to the growing importance of the leakage power resulting from the technology downscale [76], the variation in power may be negligible as compared to the area savings.

This example demonstrates the potential introduced by the port limitation mechanism. Its applicability can be combined with other design constraints. This provides a designer with a vast spectrum of possibilities for exploration and tuning.

## Computational time

The computational complexity of the IP model depends significantly on the number of binary variables of the model that determines the span of the branch-and-bound search. For a square mesh of $P$ processing elements, the number of variables is about $4P^3$. Still, efficient ILP solvers can handle this model for problems with moderate size.

Table 7.4 shows the CPU time for solving the model with the link minimization cost function (7.10), that is the most time consuming linear cost function. The third column shows the time to find the minimal number of links that guarantee network connectedness. The last two columns report the CPU times for finding deadlock-free solutions, which are larger due to the introduction of the turn prohibition constraints.

When an optimal solution is hard to find, a feasible solution close to the optimal might be also sufficient. The column "Feasible" reports the time required by the solver to find the optimal solution, while the rest of the time was spent to prove the non-existence of a better solution. The solution for the 4x4 network was obtained

Table 7.4: CPU time for link minimization.

| Number of cores | Network size | Minimal link count (sec) | Deadlock-free (sec) | |
|---|---|---|---|---|
| | | | Feasible | Optimal |
| 8 | 4x2 | 0.21 | 0.70 | 1.98 |
| 9 | 3x3 | 0.81 | 5.31 | 31.74 |
| 10 | 5x2 | 0.55 | 4.40 | 23.60 |
| 12 | 4x3 | 7.86 | 49.90 | 4031.25 |
| 16 | 4x4 | 147.96 | 2117.89 | Timeout |

by defining a CPU timeout of three hours. The reported solution is the last one obtained within the timeout, without knowing whether it was optimal or not. Given the behavior for the other cases, we conjecture that this solution is very close to the optimal.

The results show that optimal solutions can be obtained for moderate size networks. The model is also useful to partially explore the search space with CPU time limits, still obtaining high-quality solutions.

## 7.4 Conclusions

The results of this work can be summarized as follows:

- This research states the link allocation problem for regular topologies explicitly and proposes an approach for solving this problem simultaneously with the route assignment task.

- This work is the first one to formulate an integer programming model for deadlock-free routing by turn prohibition.

- A variety of constraints and cost functions is analyzed and applied for design space exploration of the on-chip systems. The model is capable of discovering well-known structures, such as the bi-directional ring, that is seen in commercial implementations [90]. This proves that the class of the generated solutions is actually used in practice.

It has to be mentioned that the proposed model has a high computational complexity, as the majority of integer programming models. To improve the scalability of the approach, relaxation techniques for mixed-integer models can be considered. Future work for this problem is summarized in Chapter 8.

# Chapter 8

# Conclusions and Future Work

The objective of this chapter is to conclude the work and to outline the directions in which the research could be continued.

This work presents a number of contributions in the field of architectural design and deployment of many-core chip multiprocessors. First, the problem of efficient architectural design space exploration is addressed. Efficiency is obtained through the usage of analytical models and intelligent search strategies, instead of simulation-based exhaustive exploration.

An important effect of the closed CMP system is demonstrated, which is the cyclic dependency between the latency and traffic of requests to the memory subsystem. This dependency is crucial to accurately estimate the interconnect contention, when evaluating the CMP performance analytically. Several numerical methods for resolving this dependency are presented and further used to build an analytical model of a complete CMP. The efficiency of the model is demonstrated by its ability to estimate the performance of a CMP with 700 components (including cores, memories and on-chip routers) in one second.

The developed model is applied as a rapid and accurate performance estimator in a framework for architectural exploration. This framework encompasses many architectural parameters, including the core count, cache sizes, number of levels of the memory hierarchy, interconnect topologies, core architectures and others. Metaheuristic-based search is proposed to efficiently navigate through a large design space and discover optimal (or nearly-optimal) configurations, demonstrating orders of magnitude savings in run time, when compared to simulation-based methods. The search method is also shown to outperform simple exploration strategies both in CPU time and quality. An example of applying the framework to power-performance exploration of CMPs is given.

Another contribution of this work is in the field of analytical modeling for on-chip interconnects. It is shown that the classical Markovian models provide pessimistic approximation of waiting times for interconnects with constant-length packets. The constant-time model was proposed to eliminate the assumptions of the markovian models and improve the modeling precision.

This thesis makes a contribution for the problem of task mapping onto prefabricated CMPs with multiple voltage islands. Metaheuristic-driven search is shown to efficiently solve the mapping problem. The quality of the search is verified by the ability of metaheuristics to discover optimal solutions for moderate-size examples, for which the optimum is known by solving the MILP. Scalability of the method is demonstrated by performing task mapping of application graphs with hundreds of tasks.

Finally, this research contributes to the problem of topology customization for on-chip interconnects. An approach for solving the link allocation problem simultaneously with the route assignment task is proposed. This work is the first one to formulate an integer programming model for deadlock-free routing by turn prohibition. The model is capable of discovering well-known interconnect solutions, which are seen in commercial implementations.

## Future work

There are several directions in which the presented research can be continued.

The following considerations can be taken into account to extend the developed methodology for architectural exploration. The approach described in this document performs high-level floorplanning of a chip by selecting dimensions of the top-level interconnect. However, elaboration of the detailed *cluster-level floorplans* is essential to obtain accurate estimation of the chip area. Additionally, *wire-planning* has to be performed to verify routability of the produced floorplans. The initial work in this direction has justified the importance of physical planning for CMP architectural exploration [46], [37].

Organization of the access to off-chip memory brings another dimension of exploration. The *quantity and placement of memory controllers, and location of the injection ports* have a strong impact on the system performance. Various alternatives of the memory controller architecture can also be considered. Additionally, the influence of *cache-coherence* protocols on system performance has to be studied. Traffic models have to include extra messages generated by the protocols and possibly affecting the interconnect contention.

Another interesting direction is to extend the model and architectural exploration methodology for *heterogeneous CMPs*. Possibility of having heterogeneous clusters on-chip results into an enormous increase of the design space. This fact requires a clever selection and fine tuning of the transformation set to obtain reasonable compromise between run time and solution quality. Furthermore, analytical model has to be verified to provide accurate performance estimation for heterogeneous configurations. Alternative numerical methods can be considered for resolving the cyclic dependency between memory traffic and latency.

The task mapping approach proposed in this work has a limitation of assigning only one task per core. To eliminate this constraint, the problem of dynamic task mapping or scheduling has to be addressed. Another important aspect that is not considered in current formulation of the problem is thermal-aware task mapping. Thermal issues influence the primary metrics of the chip, such as performance and power dissipation, and additionally affect the circuit lifetime.

The major issue of the proposed model for link allocation and routing is the high computational complexity, which is typical for the majority of integer programming models. One of the options to improve the scalability of the approach is to explore relaxation techniques for IP models. Another option is to consider alternative solution strategies, such as heuristical search, compromising search time and solution quality. The described model also assumes that the application is pre-mapped to the initial topology. Development of a combined method for mapping and topology customization with route allocation can be a next step in enhancing the proposed methodology.

# Bibliography

[1] ARM AMBA specification: http://www.arm.com/products/system-ip/amba/.

[2] CACTI. http://www.hpl.hp.com/research/cacti/.

[3] CPLEX. http://www.ilog.com/products/cplex.

[4] CPU Database. http://cpudb.stanford.edu.

[5] MATLAB. http://www.mathworks.com.

[6] Parameters of Intel Core 2 Duo E6400 processor at CPU Database. http://cpudb.stanford.edu/processors/1088.

[7] ST Microelectronics website: http://www.st.com/.

[8] Texas Instruments OMAP3530 Application Processor. http://www.ti.com/product/omap3530.

[9] Texas Instruments TMS320DRM300/350 Digital Radio Mondiale Solution. http://www.ti.com/lit/ml/sprt354/sprt354.pdf.

[10] D. Abts, N. D. E. Jerger, J. Kim, D. Gibson, and M. H. Lipasti. Achieving predictable performance through better memory controller placement in many-core CMPs. In *ISCA*, pages 451–461, 2009.

[11] T. Ahonen, D. A. Sigüenza-Tortosa, H. Bin, and J. Nurmi. Topology optimization for application-specific networks-on-chip. In *Proceedings of the 2004 international workshop on System level interconnect prediction*, pages 53–60, 2004.

[12] A. R. Alameldeen. *Using compression to improve chip multiprocessor performance.* PhD thesis, 2006.

[13] G. Ascia, V. Catania, and M. Palesi. Multi-objective mapping for mesh-based noc architectures. In *Intl. Conf. Hardware/Software Codesign and System Synthesis*, pages 182–187, 2004.

[14] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, pages 506–517, 2005.

[15] J. Balfour and W. J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *Proc. Intl. Conf. on Supercomputing*, pages 187–198, 2006.

[16] G. Beltrame, D. Sciuto, C. Silvano, P. Paulin, and E. Bensoudane. An application mapping methodology and case study for multi-processor on-chip architectures. In *Intl. Conf. Very Large Scale Integration*, pages 146=–151, Oct. 2006.

[17] W. Ben-Ameur. Computing the initial temperature of simulated annealing. *Comput. Optim. Appl.*, 29(3):369–385, Dec. 2004.

[18] Y. Ben-Itzhak, I. Cidon, and A. Kolodny. Delay analysis of wormhole based heterogeneous NoC. In *NOCS*, pages 161–168, May 2011.

[19] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Trans. Parallel Distrib. Syst.*, 16(2):113–129, 2005.

[20] D. Bertsekas and R. Gallager. *Data Networks.* Prentice Hall, second edition, 1992.

[21] D. P. Bertsekas. *Nonlinear Programming.* 2nd edition, Sept. 1999.

[22] D. Bhandarkar. RISC architecture trends. In *Proc. CompEuro*, pages 345–352, 1991.

[23] L. N. Bhuyan, Q. Yang, and D. P. Agrawal. Performance of multiprocessor interconnection networks. *Computer*, 22(2):25–37, Feb. 1989.

[24] S. Boettcher and A. G. Percus. Extremal optimization: Methods derived from coevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, Florida, USA*, pages 825–832, 1999.

[25] P. Bogdan, M. Kas, R. Marculescu, and O. Mutlu. Quale: A quantum-leap inspired model for non-stationary analysis of NoC traffic in chip multiprocessors. In *International Symposium on Networks-on-Chip (NOCS)*, pages 241–248, May 2010.

[26] P. Bogdan and R. Marculescu. Non-stationary traffic analysis and its implications on multicore platform design. *Computer-Aided Design of Integrated Circuits and Systems*, 30:508–519, Apr. 2011.

[27] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. QNoC: QoS architecture and design process for network on chip. *Journal of Systems Architecture*, 50(2-3):105–128, 2004.

[28] E. Bolotin, A. Morgenshtein, I. Cidon, R. Ginosar, and A. Kolodny. Automatic hardware-efficient SoC integration by QoS network on chip. In *Electronics, Circuits and Systems*, pages 479–482, Dec. 2004.

[29] S. Borkar and A. A. Chien. The future of microprocessors. *Commun. ACM*, 54(5):67–77, May 2011.

[30] R. Burden and D. Faires. *Numerical Analysis.* Brooks Cole, 2010.

[31] A. Cassidy, K. Yu, H. Zhou, and A. Andreou. A high-level analytical model for application specific CMP design exploration. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 1–6, Mar. 2011.

[32] A. S. Cassidy and A. G. Andreou. Beyond Amdahl's law: An objective function that links multiprocessor performance gains to delay and energy. *IEEE Trans. Comp.*, 61(8):1110–1126, Aug. 2012.

[33] G. Chen, F. Li, S. Son, and M. Kandemir. Application mapping for chip multiprocessors. In *Proc. ACM/IEEE Design Automation Conference*, pages 620–625, June 2008.

[34] G.-M. Chiu. The odd-even turn model for adaptive routing. *IEEE Trans. Parallel Distrib. Syst.*, 11(7):729–738, 2000.

[35] Y. Chou, B. Fahs, and S. Abraham. Microarchitecture optimizations for exploiting memory-level parallelism. In *ISCA*, pages 76–87, 2004.

[36] A. D. Choudhury, G. Palermo, C. Silvano, and V. Zaccaria. Yield enhancement by robust application-specific mapping on network-on-chips. In *Intl. Workshop on Network on Chip Architectures*, pages 37–42, 2009.

[37] J. Cortadella, J. de San Pedro, N. Nikitin, and J. Petit. Physical-aware system-level design for tiled hierarchical chip multiprocessors. In *Intl. Symp. Physical Design*, Mar. 2013.

[38] D. Pham et al. Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor. *Solid-State Circuits*, 41:179–196, 2006.

[39] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks.* Morgan Kaufmann Publishers, Inc., 2003.

[40] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, 36(5):547–553, 1987.

[41] W. J. Dally and B. Towles. Route packets, not wires: on-chip inteconnection networks. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 684–689, New York, NY, USA, 2001. ACM.

[42] A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz. CPU DB: recording microprocessor history. *ACM Queue*, 10(4):10–27, Apr. 2012.

[43] R. Das, S. Eachempati, A. Mishra, V. Narayanan, and C. Das. Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs. In *High Performance Comp. Arch.*, pages 175–186, Feb. 2009.

[44] I. De Falco, A. Della Cioppa, D. Maisto, U. Scafuri, and E. Tarantino. A multiobjective extremal optimization algorithm for efficient mapping in grids. In *Applications of Soft Computing*, volume 58, pages 367–377, 2009.

[45] J. de San Pedro. A simulation framework for hierarchical Network-on-Chip systems. Master's thesis, 2012.

[46] J. de San Pedro, N. Nikitin, J. Cortadella, and J. Petit. Physical planning for the architectural exploration of large-scale chip multiprocessors. In *Proc. ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, Apr. 2013.

[47] F. Deslauriers, M. Langevin, G. Bois, Y. Savaria, and P. Paulin. Roc: A scalable network on chip based on the token ring concept. In *Circuits and Systems, 2006 IEEE North-East Workshop on*, page 157, June 2006.

[48] R. R. Dobkin, R. Ginosar, and A. Kolodny. QNoC asynchronous router. *Integration, the VLSI Journal*, 42(2):103 – 115, 2009.

[49] J. T. Draper and J. Ghosh. A comprehensive analytical model for wormhole routing in multicomputer systems. *J. Parallel Distrib. Comput.*, 23(2):202–214, 1994.

[50] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Trans. Parallel Distrib. Syst.*, 4(12):1320–1331, Dec. 1993.

[51] J. Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. *IEEE Trans. Parallel Distrib. Syst.*, 6(10):1055–1067, Oct. 1995.

[52] J. Duato, S. Yalamanchili, and N. Lionel. *Interconnection Networks: An Engineering Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[53] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith. A mechanistic performance model for superscalar out-of-order processors. *ACM Trans. Comput. Syst.*, 27:1–37, May 2009.

[54] J. A. Fisher. Very Long Instruction Word architectures and the ELI-512. *SIGARCH Comput. Archit. News*, 11(3):140–150, June 1983.

[55] G. S. Fishman. Grouping observations in digital simulation. *Management Science*, 24:510–521, 1978.

[56] S. Foroutan, Y. Thonnart, R. Hersemeule, and A. Jerraya. An analytical method for evaluating Network-on-Chip performance. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 1629–1632, Mar. 2010.

[57] P. Ghosh and A. Sen. Energy efficient mapping and voltage islanding for regular NoC under design constraints. *Int. J. High Perform. Syst. Archit.*, 2:132–144, Aug. 2010.

[58] F. Gilabert, F. Silla, M. E. Gomez, M. Lodde, A. Roca, J. Flich, J. Duato, C. Hernández, and S. Rodrigo. *Designing Network On-Chip Architectures in the Nanoscale Era.* CRC Press, 2010.

[59] C. J. Glass and L. M. Ni. The turn model for adaptive routing. *SIGARCH Comput. Archit. News*, 20(2):278–287, 1992.

[60] P. Gratz, C. Kim, R. Mcdonald, S. W. Keckler, and D. Burger. Implementation and evaluation of on-chip network architectures. In *in International Conference on Computer Design*, pages 477–484, 2006.

[61] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. Efficient link capacity and QoS design for network-on-chip. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 9–14, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[62] A. Hansson, K. Goossens, and A. Rădulescu. A unified approach to constrained mapping and routing on network-on-chip architectures. In *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 75–80, 2005.

[63] A. Hartstein, V. Srinivasan, T. Puzak, and P. Emma. On the nature of cache miss behavior: is it square root of 2. *Journal of Instruction-Level Parallelism*, 10, 2008.

[64] J. L. Hennessy and D. A. Patterson. *Computer Architecture, 4th Edition: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., 2006.

[65] T. Henriksson, P. van der Wolf, A. Jantsch, and A. Bruce. Network calculus applied to verification of memory access performance in SoCs. In *Workshop on Embedded Systems for Real-Time Multimedia*, pages 21–26, Oct. 2007.

[66] R. Holsmark, M. Palesi, and S. Kumar. Deadlock free routing algorithms for irregular mesh topology NoC systems with rectangular regions. *Journal of Systems Architecture*, 54(3-4):427 – 440, 2008.

[67] J. Hu and R. Marculescu. Dyad: smart routing for networks-on-chip. In *DAC '04: Proceedings of the 41st annual Design Automation Conference*, pages 260–263, New York, NY, USA, 2004. ACM.

[68] J. Hu and R. Marculescu. Energy- and performance-aware mapping for regular NoC architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24:551–562, 2005.

[69] J. Huh, D. Burger, and S. W. Keckler. Exploring the design space of future CMPs. In *Proc. Int. Conf. Parallel Architectures and Compilation Techniques*, pages 199–210, 2001.

[70] W.-L. Hung, G. M. Link, Y. Xie, N. Vijaykrishnan, N. Dhanwad, and J. Conner. Temperature-aware voltage islands architecting in system-on-chip design. In *Proc. International Conf. Computer Design (ICCD)*, pages 689–696, 2005.

[71] Intel Whitepaper. 64-bit Intel Xeon Processor MP with up to 8MB L3 cache. 2005.

[72] E. İpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz. Efficiently exploring architectural design spaces via predictive modeling. *SIGARCH Comput. Arch.*, 34(5):195–206, Oct. 2006.

[73] J. Howard et al. A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling. *J. Solid-State Circuits*, 46(1):173–183, 2011.

[74] J. Owens et al. GPU computing. *Proceedings of the IEEE*, 96:879–899, May 2008.

[75] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. Construction and use of linear regression models for processor performance analysis. In *Intl. Symp. High-Performance Computer Architecture*, pages 99–108, 2006.

[76] A. Kahng, B. Li, L.-S. Peh, and K. Samadi. Orion 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 423–428, Apr. 2009.

[77] S. Kang and R. Kumar. Magellan: a search and machine learning-based framework for fast multi-core design space exploration and optimization. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 1432–1437, 2008.

[78] J. Kim and W. J. Dally. Flattened butterfly: A cost-efficient topology for high-radix networks. In *Proc. of the Intl. Symp. on Computer Architecture*, 2007.

[79] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *IEEE 14th International Symposium on High Performanc Computer Architecture (HPCA)*, pages 123 –134, Feb. 2008.

[80] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[81] L. Kleinrock. *Queueing Systems, Volume 1.* Wiley-Interscience, 1975.

[82] D. E. Lackey, P. S. Zuchowski, T. R. Bednar, D. W. Stout, S. W. Gould, and J. M. Cohn. Managing power and performance for system-on-chip designs using voltage islands. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 195–202, 2002.

[83] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet.* Springer-Verlag, Berlin, Heidelberg, 2001.

[84] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proc. Int. Symp. Microarchitecture*, pages 469–480, 2009.

[85] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron. CMP design space exploration subject to physical constraints. In *High-Performance Computer Architecture*, pages 17–28, Feb. 2006.

[86] M. Azimi et al. Integration Challenges and Tradeoffs for Tera-scale Architectures. *Intel Technology Journal*, August 2007.

[87] M. Taylor et al. The Raw microprocessor: a computational fabric for software circuits and general-purpose programs. *Micro, IEEE*, 22(2):25–35, Mar. 2002.

[88] W.-K. Mak and J.-W. Chen. Voltage island generation under performance requirement for SoC designs. In *Proc. of Asia and South Pacific Design Automation Conference*, pages 798–803, 2007.

[89] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote. Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives. *IEEE Transactions on Computer-Aided Design*, 28(1):3–21, Jan. 2009.

[90] M. R. Marty and M. D. Hill. Coherence ordering for ring-based chip multiprocessors. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 309–320, Washington, DC, USA, 2006. IEEE Computer Society.

[91] R. E. Matick, T. J. Heller, and M. Ignatowski. Analytical analysis of finite cache penalty and cycles per instruction of a multiprocessor memory hierarchy using miss rates and queuing theory. *IBM J. Res. Dev.*, 45:819–842, Nov. 2001.

[92] G. D. Micheli and L. Benini. *Networks on Chips: Technology and Tools (Systems on Silicon).* Morgan Kaufmann Publishers, Inc., 2006.

[93] M. Monchiero, R. Canal, and A. Gonzalez. Power/performance/thermal design-space exploration for multicore architectures. *Parallel and Distributed Systems*, 19(5):666–681, May 2008.

[94] S. Murali, L. Benini, and G. De Micheli. An application-specific design methodology for on-chip crossbar generation. *IEEE Transactions on Computer-Aided Design*, 26(7):1283–1296, July 2007.

[95] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli. A methodology for mapping multiple use-cases onto networks on chips. In *Proc. Design, Automation and Test in Europe (DATE)*, DATE '06, pages 118–123, 2006.

[96] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. De Micheli, and L. Raffo. Designing application-specific networks on chips with floorplan information. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 355–362, New York, NY, USA, 2006. ACM.

[97] S. Murali and G. D. Micheli. An application-specific design methodology for stbus crossbar generation. In *Design, Automation and Test in Europe (DATE)*, pages 1176–1181, 2005.

[98] N. Nikitin, S. Chatterjee, J. Cortadella, M. Kishinevsky, and U. Ogras. Physical-aware link allocation and route assignment for chip multiprocessing. In *Proc. ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 125–134, 2010.

[99] N. Nikitin and J. Cortadella. Static task mapping for tiled chip multiprocessors with multiple voltage islands. Technical report, `http://www.lsi.upc.edu/~techreps/files/R11-13.zip`.

[100] N. Nikitin and J. Cortadella. A performance analytical model for network-on-chip with constant service time routers. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 571–578, 2009.

[101] N. Nikitin and J. Cortadella. Static task mapping for tiled chip multiprocessors with multiple voltage islands. In *Proc. Architecture of Computing Systems*, pages 50–62, 2012.

[102] N. Nikitin, J. de San Pedro, J. Carmona, and J. Cortadella. Analytical performance modeling of hierarchical interconnect fabrics. In *Proc. ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 107–114, May 2012.

[103] U. Ogras, P. Bogdan, and R. Marculescu. An analytical approach for network-on-chip performance analysis. *Computer-Aided Design of Integrated Circuits and Systems*, 29:2001–2013, Dec. 2010.

[104] U. Y. Ogras and R. Marculescu. Energy- and performance-driven NoC communication architecture synthesis using a decomposition approach. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 352–357, 2005.

[105] Ü. Y. Ogras and R. Marculescu. ”It’s a small world after all”: NoC performance optimization via long-range link insertion. *IEEE Trans. VLSI Syst.*, 14(7):693–706, 2006.

[106] U. Y. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu. Voltage-frequency island partitioning for GALS-based networks-on-chip. In *Proc. ACM/IEEE Design Automation Conference*, pages 110–115, 2007.

[107] T. Oh, H. Lee, K. Lee, and S. Cho. An analytical model to study optimal area breakdown between cores and caches in a chip multiprocessor. In *ISVLSI*, pages 181–186, May 2009.

[108] K. Olukotun. *Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency.* Morgan and Claypool Publishers, 2007.

[109] C. D. Pack. The effects of multiplexing on a computer-communications system. *Commun. ACM*, 16(3):161–168, 1973.

[110] C. D. Pack. The Output of an M/D/1 Queue. *OPERATIONS RESEARCH*, 23(4):750–760, 1975.

[111] G. Palermo, G. Mariani, C. Silvano, R. Locatelli, and M. Coppola. Mapping and topology customization approaches for application-specific stnoc designs. In *Intl. Conf. Application-Specific Systems, Architectures and Processors*, pages 61–68, July 2007.

[112] G. Palermo, C. Silvano, and V. Zaccaria. ReSPIR: A response surface-based pareto iterative refinement for application-specific design space exploration. *IEEE Transactions on Computer-Aided Design*, 28(12):1816–1829, Dec. 2009.

[113] G. Passas, M. Katevenis, and D. Pnevmatikatos. Crossbar NoCs are scalable beyond 100 nodes. *IEEE Transactions on Computer-Aided Design*, 31(4):573–585, Apr. 2012.

[114] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli. Efficient synthesis of networks on chip. In *Proc. ICCD, 2003*, pages 146–150, 2003.

[115] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli. A methodology for constraint-driven synthesis of on-chip communications. *IEEE Transactions on Computer-Aided Design*, 28(3):364–377, Mar. 2009.

[116] F. J. Pollack. New microarchitecture challenges in the coming generations of cmos process technologies. In *IEEE Micro*, page 2, 1999.

[117] T. K. Prakash and L. Peng. Performance characterization of SPEC CPU2006 on Intel Core 2 Duo processor. In *ISAST*, pages 36–41, 2008.

[118] S. Bell et al. TILE64 - processor: A 64-core SoC with mesh interconnect. In *Solid-State Circuits*, pages 88–98, Feb. 2008.

[119] S. Vangal et al. An 80-tile 1.28TFLOPS network-on-chip in 65nm CMOS. In *Solid-State Circuits*, pages 98–589, Feb. 2007.

[120] T. J. Santner, W. B., and N. W. *The Design and Analysis of Computer Experiments.* Springer-Verlag, 2003.

[121] M. Sayeed and M. Atiquzzaman. Multiple-bus multiprocessor under unbalanced traffic. *Computers and Electrical Engineering*, 1999.

[122] M. K. F. Schafer, T. Hollstein, H. Zimmer, and M. Glesner. Deadlock-free routing and component placement for irregular mesh-based networks-on-chip. In *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pages 238–245, Washington, DC, USA, 2005. IEEE Computer Society.

[123] H. She, Z. Lu, A. Jantsch, L.-R. Zheng, and D. Zhou. Traffic splitting with network calculus for mesh sensor networks. In *Proceedings of the Future Generation Communication and Networking*, pages 368–373, 2007.

[124] D. Sheldon, F. Vahid, and S. Lonardi. Interactive presentation: Soft-core processor customization using the design of experiments paradigm. In *Proc. Design, Automation and Test in Europe*, pages 821–826, 2007.

[125] K. Srinivasan, K. S. Chatha, and G. Konjevod. Linear-programming-based techniques for synthesis of network-on-chip architectures. *IEEE Transactions on VLSI Systems*, 14(4):407–420, 2006.

[126] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: maximizing on-chip parallelism. In *Intl. Symp. on Computer architecture*, pages 533–544, 1998.

[127] G. Varatkar and R. Marculescu. Communication-aware task scheduling and voltage selection for total systems energy minimization. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, 2003.

[128] H. Wang, L.-S. Peh, and S. Malik. A technology-aware and energy-oriented topology exploration for on-chip networks. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 1238–1243, 2005.

[129] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. In *Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*, pages 294–305, 2002.

[130] X. Wang, P. Liu, M. Yang, M. Palesi, Y. Jiang, and M. C. Huang. Energy efficient run-time incremental mapping for 3-d networks-on-chip. *J. Comput. Sci. Technol.*, 28(1):54–71, 2013.

[131] N. H. E. Weste and K. Eshraghian. *Principles of CMOS VLSI design: a system perspective.* Addison-Wesley, Boston, MA, USA, 1985.

[132] E. C. Wille, M. Mellia, E. Leonardi, and M. A. Marsan. A lagrangean relaxation approach for qos networks cfa problems. *International Journal of Electronics and Communications*, 63(9):743–753, 2009.

[133] E. C. G. Wille, M. Mellia, E. Leonardi, and M. A. Marsan. Algorithms for IP network design with end-to-end QoS constraints. *Comput. Netw.*, 50(8):1086–1103, 2006.

[134] D. F. Wong and C. L. Liu. A new algorithm for floorplan design. In *Proc. ACM/IEEE Design Automation Conference*, pages 101–107, 1986.

[135] G. Wood. The bisection method in higher dimensions. *Math. Program.*, 55, June 1992.

[136] W. A. Wulf and S. A. McKee. Hitting the memory wall: implications of the obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, Mar. 1995.

[137] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar. A design methodology for application-specific networks-on-chip. *ACM Trans. Embed. Comput. Syst.*, 5(2):263–280, 2006.

[138] Z. Wang et al. A novel low-waveguide-crossing floorplan for fat tree based optical networks-on-chip. In *Proc. Intl. Conf. Optical Interconnects*, May 2012.

[139] V. Zaccaria, G. Palermo, F. Castro, C. Silvano, and G. Mariani. Multicube Explorer: An open source framework for design space exploration of chip multi-processors. *Proc. Architecture of Computing Systems*, pages 1–7, Feb. 2010.

[140] T. Zhou, W.-J. Bai, L.-J. Cheng, and B.-H. Wang. Continuous extremal optimization for Lennard-Jones clusters. *Phys. Rev. E*, 72(1), July 2005.