System-level design space exploration (DSE), which is performed early in the design process, is of eminent importance to the design of complex multi-processor embedded system architectures. During system-level DSE, system parameters like, e.g., the number and type of processors, the type and size of memories, or the mapping of application tasks to architectural resources, are considered. Simulation-based DSE, in which different design instances are evaluated using system-level simulations, typically are computationally costly. Even using high-level simulations and efficient exploration algorithms, the simulation time to evaluate design points forms a real bottleneck in such DSE. Therefore, the vast design space that needs to be searched requires effective design space pruning techniques. This thesis presents different methods for iteratively reducing the number of simulations needed during system-level DSE.

Pruning Techniques for Multi-Objective System-Level Design Space Exploration

Roberta Piscitelli

University of Amsterdam

Pruning Techniques for Multi-Objective System-Level Design Space Exploration

Roberta Piscitelli

Pruning Techniques

for Multi-Objective System-Level Design Space Exploration

Pruning Techniques
for Multi-Objective System-Level Design Space Exploration

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. D.C. van den Boom
ten overstaan van een door het college voor promoties ingestelde
commissie, in het openbaar te verdedigen in de Agnietenkapel
op donderdag 18 september 2014 om 12:00 uur
door

Roberta Piscitelli

geboren te Napels, Italië.

Promotiecommissie:

Promotor: Prof. dr. ir. C.T.A.M. de Laat
Copromotor: Dr. A. D. Pimentel

Overige leden:
Dr. ir. G. Beltrame
Prof. dr. J.A. Bergstra
Prof. dr. R.J. Meijer
Prof. dr. ir. L. Raffo
Dr. T. Stefanov

Faculteit: Faculteit der Natuurwetenschappen, Wiskunde en Informatica

This work has been supported by the MADNESS STREP-FP7 European Project, Ministere de l'Education, du Loisir et du Sport (MELS) Quebec, and Zeno Karl Schindler Foundation, Geneve Switzerland.

Cover: Redwood Forest of California, by R. Piscitelli

Author email: roberta.piscitelli83@gmail.com

Acknowledgements

Facilis descensus Averno:
noctes atque dies patet atri ianua Ditis;
sed revocare gradum superasque evadere ad auras,
hoc opus, hic labor est.

Aeneid (VI, 126-129), Virgil

It all started four and half years ago.

A PhD is a psycological trip, and somehow, my research topic was itself reflecting many issues in my life. Life choices have different reasonings: there are different *objectives* to accomplish, and several *constraints*. In other words, any decision we make in our life is the result of an *optimisation problem*. Sometimes we try to *explore* different solutions, to see if we can improve our existence. Doing so, I have been hopping between Italy, Netherlands, Spain, Canada and Switzerland. We always try to get a trade off between our objectives; however, we are never sure whether we have reached the optimal situation. At the same time, the number of possibilities to explore is *huge*, and it is almost impossible to explore *all* of them.

I do not really know at which point of the exploration I am.. The only thing I can say about those years, is that I have had a lot of fun.

First of all I would like to thank my co-promotor, Andy Pimentel. He not only gave me the scientific support and supervision that a graduate student can expect from her professor, but he also gave me freedom and trust to pursue my own direction of research, and most important: patience than was perhaps warranted by my seeming determination to write papers not at last moment. Besides that, he has been my inspiring design space exploration philosopher. I am glad he trusted I was crazy enough to join his group.

Giovanni Beltrame, from École Polytechnique de Montréal, was my primary supervisor during my visit in Canada. His ideas, his research, and especially his unique enthusiasm, gave me a new, awakening energy to keep pursuing this path. Working with him and Brett Meyer was exciting, and extremely fun.

I am also grateful to Francesco Regazzoni, to support me and to show new inspiring research directions during the last months of my PhD.

There are other researchers whom I would specifically like to thank for the support they have given me over the years: Mark Thompsons, Todor Stefanov, Teddy Zhai, Emanuele Cannella, Paolo Meloni, Luigi Raffo, Daniela Dimitrova, Peter van Stralen, and Simon Polstra, for their cooperation within the MADNESS project.

Furthermore, I thank my promotor Cees de Laat for his support of my research and Erik Hitipeuw for his lightning-fast response regarding some last-minute administration issues.

Over the years it has been my good fortune to encounter many people who have given me more of their time, companionship, professional and personal help, and above all: my colleagues from the CSA group (especially Mark, Peter, Toktam, Simon, Ke Na, Merijn, Roy, Michiel and Sebastian), my colleagues Jacopo and Alain from the MistLab, and the all the ALaRI team.

Peter was sharing with me the same love for sunlight, and somehow he could nicely tolerate my mania for vanilla perfumes. Mark was a great company for all the cakes and chocolate we ate during the long afternoons in the office. Ke Na and Merijn have always entertained me with intere-

sting philosophical debates, garnished with tea, biscuits and coffee at The Elephant.

I am also in debt of affection with Karel, who passionately helped me reviewing the dutch text of this thesis.

I am, of course, strongly indebted to my sister, my mother and all my friends, and in particular: Lara, Daniela, Mark, Emiliano, Alessandra, Hoda, Lucia and Xun, for their monumental, unwavering support and encouragement on all fronts. They have truly always been there for me, and without them none of this would have been even remotely possible.

Amsterdam, *Roberta Piscitelli*
August 2014

Contents

Chapter 1

Introduction

1.1 Introduction

The design of modern embedded systems has become increasingly complex. There is a wide range of design parameters that have to be tuned up to find the optimal tradeoff in terms of several design requirements. Those systems should be low cost, small in terms of area, light weight and be power efficient, since they are often battery-based devices. This is in contrast with the requirements of achieving real-time, performance and providing reliable and secure operation. As result, the increasing market for compact embedded computing devices is leading to new multi-processor system-on-a-chip (MPSoC) architectures designed for embedded systems, providing task-level parallelism for streaming applications integrated in a single chip. Those MPSoC systems are composed of different types of processing units, memories, and specialised hardware components. For example, modern smartphones include different processors and hardware blocks to support GPS-based navigation, internet browsing, video capture and processing, and, naturally, speech processing. Such embedded systems can be found also in modern TVs, car navigation systems, and common household devices.

Designers must address new challenges that were not present before:

such MPSoC architectures are heterogeneous in nature and are required to be general enough to be used across several different applications in order to be economically viable, leading to recent attention to parameterized MPSoC platform architectures. On the other hand, they have very different design constraints such as power efficiency, timing requirements or performance budgets. In the remainder of this chapter, we describe the background of the embedded systems field, discuss the motivation of the work presented in this thesis, and address the main research question.

1.2 Problem description

Platform based design of heterogeneous multi-processor system-on-chip (MPSoC) systems is becoming today's predominant design paradigm in the embedded systems domain [81]. In contrast to more traditional design paradigms, platform based design shortens design time by eliminating the effort of the low-level design and implementation of system components. A platform based design environment typically consists of a fixed, parameterizable platform or a set of (parameterizable) components that can be combined in specific ways to compose a platform.

The parameters make it possible to adjust platforms and individual components to the required application domain and platform design requirements. Examples of platform parameters are:

- type of general processing unit used, which can be a general purpose processor like ARM and MIPS cores, or a dedicated hardware component unit like Application Specific Integrated Circuits (ASICs) specialized for Discrete Cosine Transform (DCT) operations or Variable Length Encoding (VLE).

- type of communication infrastructure, which can be shared bus or crossbar based architecture.

- memory subsystems, which can vary for latency and capacity levels.

- HW/SW partitioning: determining which tasks will be implementetd in software and which tasks as fixed ASICs or reconfigurable hardware blocks.

A platform instance is a set of parametrized components that are selected from a library. These parametrized MPSoCs architectures must be tuned (i.e., their configuration parameters must be appropriately chosen) to find the best trade-off in terms of a set of metrics (e.g., energy and delay) for a given class of applications. This tuning process is called *Design Space Exploration (DSE)* [78]. This process allows to explore a wide range of early design choices which heavily influence the success or failure of the final product.

In general, DSE involves the minimisation (or maximization) of multiple objectives. DSE is the first step for a multi-objective optimisation procedure, as shown in Figure 1.1. The solution of multi-objective optimization problems consists of finding the points of the Pareto curve [24], i.e. all the points which are better than all the others for at least one objective. Consequently, in Step 2 higher-level information is used to choose one of the obtained trade-off points.

State-of-the-art solutions for system-level DSE are essentially composed of two elements:

- *searching in the design space*
- *evaluating a single design point in the design space*

The most straightforward but least efficient approach to determine the Pareto-optimal set of configurations of a parameterized SoC architecture with respect to multi-objective design optimization criteria is to do an exhaustive search of the configuration space. However, a Pareto curve for a specific platform is available only when all the points in the design space have been evaluated and characterized in terms of the metrics of merit. This exhaustive search approach is often unfeasible due to large design spaces, and long evaluation times. Therefore, meta-heuristic algorithms

(like genetic algorithms, simulated annealing, and ant colony optimization) are often used.

The evaluation of a single design point in the design space consists of objective values like performance, system resilience and power consumption, and a mechanism for traversing the design space to search for an optimal (set of) design point(s). To evaluate a single design point, roughly three approaches are available: 1) performing measurements on a prototype implementation, 2) simulation-based measurements and 3) estimations based on some kind of analytical model. Each of these methods has different properties with respect to evaluation time and accuracy. Evaluation of prototype implementations provides the highest accuracy, but long development and/or synthesis times prohibit evaluation of many design options. Analytical estimations are considered the fastest, but accuracy is limited since they are typically unable to sufficiently capture particular intricate system behaviour. Simulation-based evaluation fills up the range between these two extremes: both highly accurate (but slower) and fast (but less accurate) simulation techniques are available. This trade-off between accuracy and speed is very important, especially for early system-level DSE in which the design space that needs to be explored is vast and some accuracy can often be traded for efficiency to cope with these large design spaces. Current DSE efforts typically use simulation or analytical models to evaluate single design points together with a heuristic search method [39] to search the design space. These DSE methods search the design space using only a finite number of design-point evaluations, not guaranteeing to find the absolute optimum in the design space, but they reduce the design space to a set of design candidates that meet certain requirements or are close to the optimum with respect to certain objectives.

Our focus is on *system-level mapping DSE*, where mapping involves two aspects: 1) allocation and 2) binding. Allocation deals with selecting the architectural components in the MPSoC platform architecture that will be involved in the execution of the application workload (i.e., not all platform components need to be used), modelling the configuration problem as well. Subsequently, the binding specifies which application task or application

communication is performed by which MPSoC component. State-of-the-art DSE approaches typically use either simulation or an analytical model to evaluate mappings, where simulative approaches prohibit the evaluation of many design options due to the higher evaluation performance costs and analytical approaches may suffer from accuracy issues.

DSE is making design decisions in the early design stages is *crucial* to reduce the number of implementation options and thereby reducing the total design effort. *Design space pruning* is a technique to make the optimisation process of the DSE more efficient, allowing to search larger design spaces or to find optimal design quicker.

Pruning techniques can, therefore,be applied to

- Speed up the design point evaluation;
- Optimize the heuristic search in the design space.

In every design phase, a subset from the non-pruned design options is selected and evaluated.

The feedback from the evaluation determines which of the candidates will be used in the next (lower) level of abstraction in the design process.

In this thesis, Evolutionary multi-objective optimization (EMO) algorithms are applied to support the design space exploration of multi-processor system-on-chip architectures.

We focus on efficient techniques to *prune* the design space while using the evolutionary optimization search algorithms, as shown in Figure 1.1.

Therefore, the research question of this thesis is:

How can we use pruning techniques to speed up the evaluation of a design point and optimise the search in the design space?

1.3 Objectives and organisation of the thesis

The work presented in this thesis has been performed in the context of several system-level simulation frameworks. In particular, we used Sesame

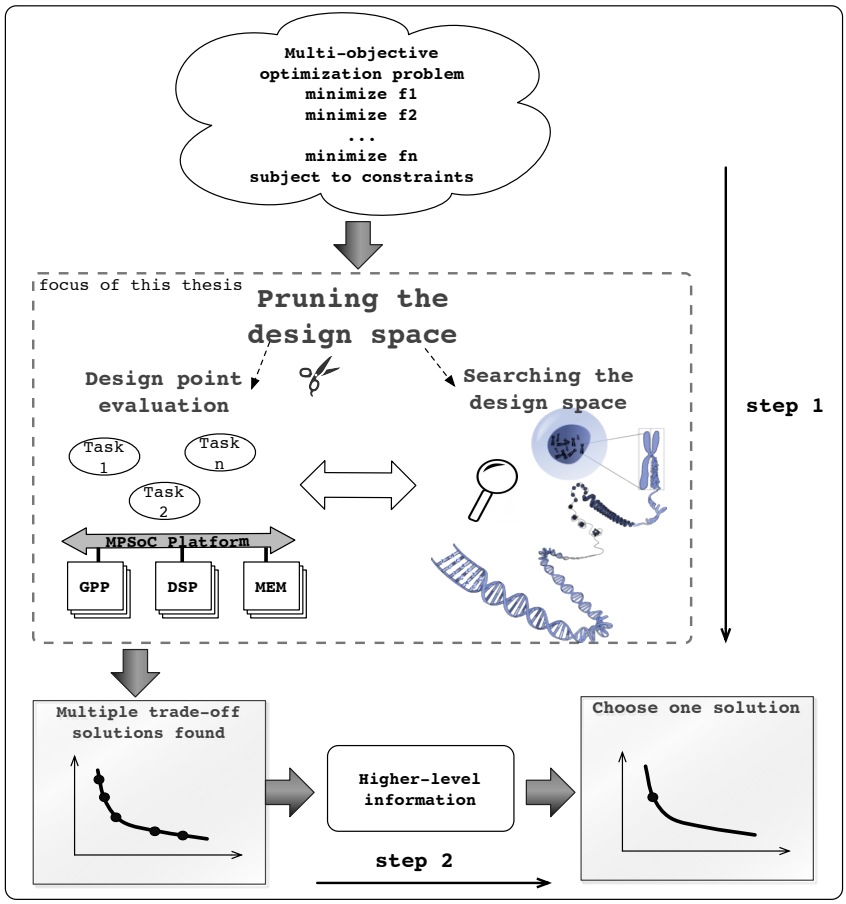


Figure 1.1: *Schematic of a multi-objective optimization procedure.*

[78] for efficiently evaluating non-functional behaviour (like performance, and cost) of an embedded system at a high level of abstraction. Initially Sesame did however not yet support system level power/energy consumption analysis. Therefore, the initial part of this thesis focuses on extending the objective space of our DSE with the introduction of a complete power modelling framework for multi-processors systems on chip (MPSoC) within Sesame. This thesis also studies DSE for lifetime optimisation of systems. In order to estimate system resilience, we employ the CQSA framework [67], which allows slack-based design space exploration for networks on

chip. The Critical Quantity Slack Allocation (CQSA) jointly optimises system resilience and cost by determining (a) how much slack should be allocated in the system, and (b) where in the system it should be allocated, such that the system mean-time-to-failure (MTTF) is increased in the most area-efficient ways possible.

The main contributions of this thesis are:

- Extending the objective space with the introduction and implementation of a complete framework for high-level power estimation for MPSoC. The technique is based on abstract execution profiles, called event signatures, and it operates at a higher level of abstraction than, e.g., commonly-used instruction-set simulator (ISS) based power estimation methods and should thus be capable of achieving good evaluation performance.

  This is essential in the context of the first phase of DSE.

- An iterative design space pruning methodology based on static throughput analysis of different application mappings.

  By interleaving these analytical throughput estimations with simulations, our hybrid approach significantly reduces the number of simulations that are needed during the process of DSE.

- A study on different strategies for interleaving fast but less accurate analytical performance estimations with slower but more accurate simulations during DSE

- *Failure scenario memoization* pruning techniques to reduce the computational cost of system lifetime estimation by storing and reusing estimated lifetime values for systems with one or more failed components. The lifetime of all partially failed systems is derived and saved (the memory storage cost of such values is negligible); when a previously explored partially-failed system is encountered a second time, its expected lifetime is read from a database rather than re-estimated.

- Correlation-based architecture distance metrics for efficiently pruning the slack-allocation based DSE for improving system resilience of NoC based MPSoCs. In modern platform- and network-on-chip based design, components are clustered around switches in the on-chip network. When clusters and the tasks mapped to them are considered to be symmetric, some configurations have the same effect on the overall system lifetime. This can be leveraged to reduce the number of evaluations.

To summarise, this thesis studies pruning techniques to speed up the search in the design space and the evaluation of a design point according to several objectives. The thesis is, therefore, organised into the following parts:

- background (Chapters 1 and 2),
- extending the design space with the objective of power/energy consumption (Chapter 3), and
- pruning techniques for system performance (Chapter 4) and lifetime optimisation (Chapter 5 and Appendix).

Chapter 2 gives an overview of the preliminary information necessary for understanding the rest of the thesis. We first describe the basic knowledge about multi-objective optimisation problems. Then, we explain the multi-objective optimisation problem in the context of design space exploration of embedded systems. We describe evolutionary algorithms as heuristic methods for searching in the design space, with a brief description of the genetic algorithm NSGA-II we use throughout this thesis. Afterwards, we discuss several metrics for evaluating the quality of the solutions obtained while performing design space exploration using heuristic search. The second part of Chapter 2 illustrates Sesame, the main framework used for the evaluation of a single design point. In particular, we provide a quick overview of the application, mapping and architecture models used in Sesame, since they will be the focus for the optimisation and modelling techniques used in Chapter 3 and 4.

Chapter 3 is dedicated to the first step for multi-objective DSE, which is introducing extra objective functions and simulation models. In this chapter, we present a full system-level MPSoC power estimation framework based on the Sesame framework, in which the power consumption of all the system components is modelled using signature-based models. The MPSoC power model has been incorporated into Daedalus, which is a system-level design flow for the design of MPSoC based embedded multimedia systems [90, 73]. This let us validate the high-level power models against real MPSoC implementations on FPGA.

Next two chapters focus essentially on the optimisation of the other two design objectives, system performance and lifetime. Chapter 4 is the first part of pruning techniques for multi-objective DSE focusing on performance evaluation and optimisation. This chapter deals with a new, hybrid form of DSE, combining simulations with analytical estimations to prune the design space in terms of application mappings that need to be evaluated using simulation. To reach our goal, the DSE technique uses an analytical model that estimates the expected throughput of an application (which is a natural performance metric for the multimedia and streaming application domain we target) given a certain architectural configuration and application-to-architecture mapping. In the majority of the search iterations of the DSE process, the throughput estimation avoids the use of simulations to evaluate the design points. However, since the analytical estimations may in some cases be less accurate, the analytical estimations still need to be interleaved with simulation-based evaluations in order to ensure that the DSE process is steered into the right direction.

We studied different techniques for interleaving these analytical and simulation-based evaluations in our hybrid DSE.

Finally, Chapter 5 focuses on pruning techniques for an important metric in modern embedded systems, which is the expected lifetime. Redundant hardware is typically employed to improve system lifetime. For instance, *slack allocation*, which overdesigns the system by provisioning execution and storage resources beyond those required to operate failure-free, has been proposed as a low-cost alternative to replicating resources [22, 67].

When components fail, data and tasks are re-mapped and re-scheduled on resources with slack; as long as performance constraints are satisfied, the system is considered to be operational despite component failure. For any given system, the design space of possible slack allocations is large and complex, consisting of every possible way to replace each component in the initial system with another component from a library.

In Chapter 5 we propose an exploration framework for Network-on-Chip (NoC) based MPSoCs that substantially reduces the computational cost of slack allocation. First, we develop *failure scenario memoization* to reduce the computational cost of lifetime estimation by storing and reusing estimated lifetime values for systems with one or more failed components. Second, we introduce a correlation-based architecture distance metric to identify symmetries for clusters of components called *islands*. In modern platform- and network-on-chip based design, components are clustered around switches in the on-chip network. When clusters and the tasks mapped to them are considered to be symmetric, some configurations have the same effect on the overall system lifetime. This can be leveraged to reduce the number of evaluations.

Chapter 2

Multi-objective Design Space Exploration [1]

2.1 Introduction

The problem of identifying optimal design points can be generally described as *multi-objective optimization problem*. In most design problems, the objectives to be taken into account are many and often conflicting.

The role of multi-objective optimization in the design industry is becoming increasingly relevant. The growing computational power of modern computers, in fact, provides designers with the ability to build complex parametric models which can be used to achieve automatic optimization procedures. The classical approach, which is still widely used to tackle multi-objective optimization problems, consists of transforming the multi-objective problem into a single-objective problem by formalizing a degree of preference among the objectives; the thus obtained single-objective problem, is then solved using one of the classical techniques of optimization, either deterministic or stochastic. In this perspective, the multi-objective problem is seen as a particular case of the mono-objective problem. This approach presents three main disadvantages:

[1] The contents of this chapter have been published as [1]

- The variety of solutions to a multi-objective problem is thus reduced a single solution resulting in a significant loss of information.
- The choice of one solution among the infinite possible (or rather, between the n numerically available) through additional information is made a priori, that is, without a complete information on all the possible solutions.
- There are some cases of (non-convex) problems in which the pure multi-objective approach provides solutions that would be impossible from a mathematical point of view to get through a classical approach, since a classical approach is not capable of making a distinction between local optimal solutions and globally optimal solutions, and will treat the former as actual solutions to the original problem.

An other approach, derived from Pareto's theory, does not require an a priori choice of the degree of preference and reverses the point of view considering the single-objective problem as a special case of the multi-objective problem. The result of the optimization is not just one but a variety, a sampling of the infinite sub-optimal solutions. Several evolutionary and non-evolutionary methods have been specifically developed for multi-objective optimization. In this work, Evolutionary multi-objective optimization (EMO) algorithms are applied to support the design space exploration. In particular, we focus on efficient techniques to *prune* the design space while using the evolutionary optimization search algorithms, as shown in Figure 1.1. Pruning techniques are applied to

- Speed up the design point evaluation;
- Optimize the heuristic search in the design space.

Further details of those two mechanisms will be discussed in Chapters 4 and Chapter 5. The first section of this chapter describes *multi-objective optimization using evolutionary algorithms*, with particular attention to the Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II). NSGA-II is one of the populari EMO algorithms used for finding (sub)-optimal

solutions in a DSE problem. Section 2.3 introduces the different metrics used to compare the quality of the final solutions. Finally, we present the Sesame framework for modelling MPSoC design instances used for fitness evaluation within the GA.

2.2 Multi-objective Optimization using Evolutionary Algorithms

Most of the optimization problems involve more than one objective to be optimized. The objectives are often conflicting, i.e., maximize performance, minimize cost, maximize reliability, etc. In that case, one extreme solution would not satisfy all objective functions and the optimal solution of one objective will not necessary be the best solution for other objective(s). Therefore, different solutions will produce trade-offs between different objectives and a set of solutions is required to represent the optimal solutions for all objectives.

A multi-objective optimization problem can be defined as the minimization or maximization of a real-valued function on a specific set. The importance of this mathematical model is obviously derived from the fact that many real problems are addressed when using such model. In the following, when not differently specified, we will consider a multi-objective optimization problem as a vector function that maps a set of m parameters (namely decision variables) to a set of n objectives. Formally:

$$\begin{array}{ll} min/max & \mathbf{y} = f(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})) \\ \text{subject to} & \mathbf{x} = (x_1, x_2, \ldots, x_m) \in \mathbf{X} \\ & \mathbf{y} = (y_1, y_2, \ldots, y_n) \in \mathbf{Y} \end{array} \tag{2.1}$$

where $\mathbf{x}$ is called the *decision vector*, $\mathbf{X}$ is the *parameter space*, $\mathbf{y}$ is the *objective vector*, and $\mathbf{Y}$ is the *objective space* [96].

The set of solutions of a multi-objective optimization problem consists of all decision vectors for which the corresponding objective vectors cannot be improved in any dimension without stripping of rank in an other; this can be explained by the following definitions:

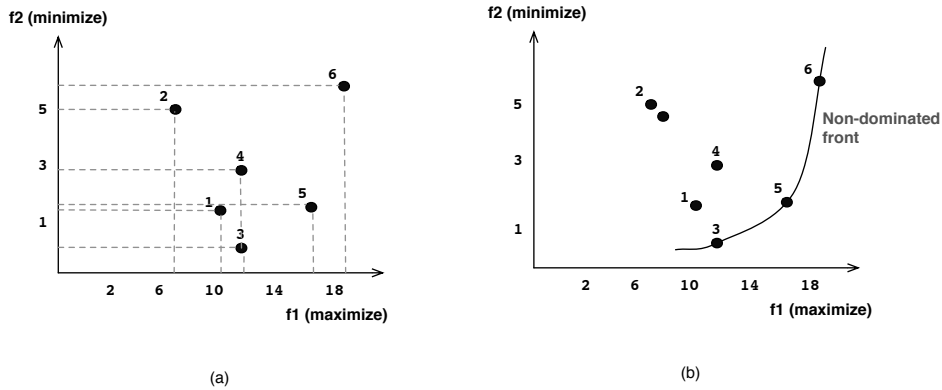


Figure 2.1: *The non-dominated front formed by the non-dominated solutions.*

Definition 2.2.1. *Given a maximization problem and consider two decision vectors* $a, b \in \mathbf{X}$.

Then, solution a is said to dominate *solution b (also written as* $\mathbf{a} \succ \mathbf{b}$*) iff*

$$\begin{aligned} \forall i \in \{1, 2, \ldots, n\} : \quad & f_i(\mathbf{a}) \geq f_i(\mathbf{b}) \quad \wedge \\ \exists j \in \{1, 2, \ldots, n\} : \quad & f_j(\mathbf{a}) > f_j(\mathbf{b}) \end{aligned} \tag{2.2}$$

Definition 2.2.2. *All decision vectors which are not dominated by any other decision vector of a given set are called* non dominated *regarding this set.*

Definition 2.2.3. *The decision vectors that are non dominated within the entire search space are denoted as Pareto optimal and constitute the so-called* Pareto-optimal set *or* Pareto-optimal front.

For a given set of solutions (or corresponding points in the objective space, for example, those shown in Figure 2.1), a pair-wise comparison can be made using the above definition to determine whether one point dominates the other. All points that are not dominated by any other member of the set are called the non-dominated points of class one, or simply the non-dominated points. For the set of six solutions shown in the figure, they are points 3, 5, and 6.

One property of any two such points is that a gain in an objective from one point to the other happens only due to a sacrifice in at least

one other objective. This trade-off property between the non-dominated points makes the practitioners interested in finding a wide variety of them before making a final choice. These points make up a front when viewed them together on the objective space.

Usually, we are only interested in *Pareto-optimal solutions*;

For several optimization problems the design space is too large to be explored; in this case the real Pareto optimal set is unknown.

According to the definition of Pareto optimality, moving from one Pareto-optimal solution to an other necessitates trading off.

2.2.1 Principles of Evolutionary Multi-Objective Optimization Search

Multi-objective Optimization problems can be identified by two aspects: search in the design space and decision making. The first of these two aspects refers to an optimization process in which the set of feasible solutions is represented by the Pareto Optimal solutions. As in single objective optimization problems, the search area typically is too large to be exhaustively explored, implying that the convergence to an optimal solution of the problem in question is not guaranteed. The second aspect (decision making) refers to the problem of choosing the "best" solution within the entire set of Pareto Optimal solutions. The Decision Maker (DM) is in charge of choosing the "best" solution. For what regards the search in the design space, we resort to evolutionary algorithms. Evolutionary algorithms are very effective in solving multi-objective problems because they are able to manage simultaneously a vast set of solutions (the so-called *population*). This feature allows evolutionary algorithms to find a substantial number of Pareto Optimal points within a short time. It's important to note that Evolutionary Algorithms do not necessarily converge to the exact global optimum, but only for a set of solutions that meet the requirements. Moreover, evolutionary algorithms are little affected by the shape and continuity of the Pareto front to search and, therefore, can be used successfully even in presence of discontinuous and / or concave fronts: most classical methods are not capable of making a distinction between local optimal solutions and global optimal solutions in a non-convex space, and they are

designed to work with continuos variables only [28].

The term Evolutionary Algorithms (EA) indicates a class of optimization methods that simulate processes of natural evolution [16]. After a succession of several generations, the populations evolve according to the laws of natural selection and survival of the fittest.

Biological systems are of great importance due to their robustness and their ability in solving a wide range of issues essential to their survival course. They are the result of an evolutionary process that bases its success on mechanisms such as *selective breeding* of the best individuals, *recombination* of their chromosomes and some random *mutations*. Although the exact function of the principles of natural evolution is still under investigation, the basic principles are clear:

- Natural evolution acts on **chromosomes** of individuals, rather than individuals, or on the genetic coding (genotype) of the physical characteristics of the living organism (phenotype), as shown in Figure 2.2.

- The processes of natural selection favour the reproduction of the most efficient individuals (and, therefore, of chromosomes) in terms of adaptivity. Essentially, individuals of a population compete to seek and obtain resources needed for survival. Similarly, individuals compete for obtaining a mate. Mating is useful because it maximises genetic recombination and it improves diversity. Individuals who become more adapted to survival and reproduction will then have a greater number of descendants. Therefore, **selection** is the process in which the phenotype influences in some way the genotype.

- The mechanism of reproduction forms the core of the evolutionary process: combining genetic codes of two individuals and the introduction of random mutations from an adaptive point of view. The combination (**crossover**) of the features of different ancestors may produce a very adapted (super fit) offspring, whose surviving ability is superior to the one of each parent. In this way populations evolve and become increasingly adapted to their environment.

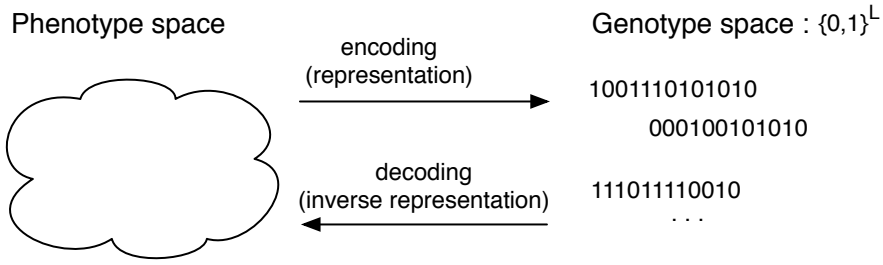


Figure 2.2: *Chromosome representation.*

- Natural evolution works on populations of individuals through a process of generations that has no historical memory, but relies solely on the interaction between each individual and the ecological environment in which it lives.

Evolutionary Algorithms are based on principles very similar to those of evolution in nature, and in addition they possess a dual purposes: first, they are useful for deeply understanding the processes of development of living systems, and secondly they aim to introduce the same characteristics of robustness and adaptability of the organic processes in artificial intelligence, in order to solve more complex problems (having constraints and discontinuous Pareto-optimal region) with respect to traditional methods. Evolutionary Algorithms make use of random search, although the whole process is driven by a selective reproduction; moreover, they are based on the **encoding of the parameters** to be optimized rather than the parameters themselves. Binary encoding is the most common method for encoding the parameters; each individual is a set of bits, 0 or 1, representing a parameter of the design point, as shown in Figure 2.2.

An alternative to binary encoding is a many-character encoding: instead of having only 0 or 1, a larger alphabet is used. This alphabet may contain strings, integers or real values. The large freedom in choosing an alphabet makes this encoding applicable to several problems. A practical example of this many-character encoding is illustrated in Section 2.2.3.

Evolutionary algorithms operate in parallel on a population of solutions distributed on the search surface. In addition, they are equipped with a

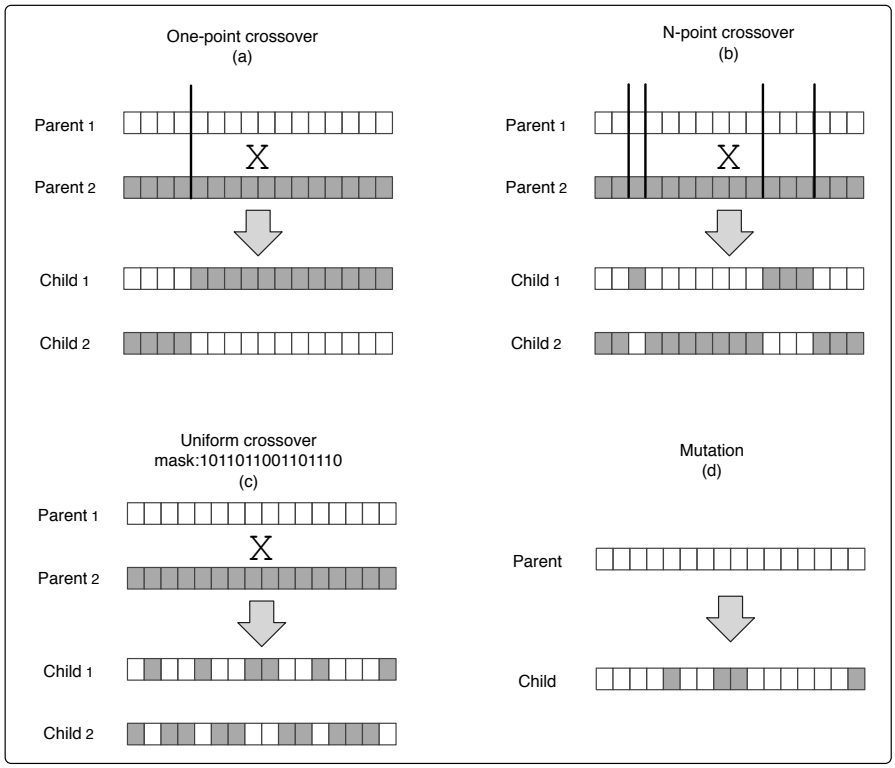


Figure 2.3: *Genetic operators.*

fitness function, which is used to guide to the entire evolution process. Essentially, a fitness function classifies the design points according to the objectives in the optimisation problem.

The basic principles of genetic algorithms have been introduced for the first time by Holland in 1975 [68]. During the execution, the algorithm repeatedly intervenes to modify a population which consists of a number of solutions (individuals) : at each iteration, it operates on a random selection of individuals of the current population, using them to generate new elements of the population, which will replace an equal number of individuals already present, and thereby forming a new population for the next iteration (or generation) through crossover and mutation. This succession of generations evolves towards an optimal solution of the assigned problem.

More in detail, the crossover operator takes two individuals as parents and creates two different offspring individuals by recombining the parents. During crossover, substrings from two parents are swapped between these parents with a fixed probability. There are many ways to implement crossover. In the one-point crossover shown in Figure 2.3(a), two parent individuals are cut at a random point and the segments after the cut point are swapped to create the offspring. In the n-points crossover (Figure 2.3(b)), n crossover points are chosen. This type of crossover is essentially a generalisation of the one-point crossover. The main drawback of those two methods is that they cannot generate any schema.

An other type of crossover that is capable of generating any schema, is *uniform crossover*. In this method each bit in the offspring is randomly selected, either from the first parent or from the second one. A crossover mask with the same length as the parents is randomly created and the parity of the bits in the mask indicates which parent will supply the offspring with which bits. An example is given in Figure 2.3(c).

Finally, the mutation operator randomly alters each bit of an individual according to a certain probability. This operator presents two main features: first, it prevents the algorithm to be trapped in a local optimum; second, it helps to maintain genetic diversity in the population. A practical example is also shown in Figure 2.3(d).

2.2.2 Elitist Non-dominated Sorting GA or NSGA-II

The Elitist Non-dominated Sorting GA (NSGA-II) is based on different levels of classification of individuals. Let P_0 be the initial population of size N. An offspring population Q_t of size N is created from current population P_t. Before the selection is performed, the combined population $R_t = Q_t \cup P_t$ is classified according to the *non-domination*: all the design points are ranked through a non-dominated sorting based on their dominance depth.

The process continues until all members of the population are classified into fronts $F_1, F_2, \ldots$.

The next population P_{t+1} is composed by individuals from the fronts $F_1, F_2, \ldots$, until the population size exceeds N; since the individuals in the

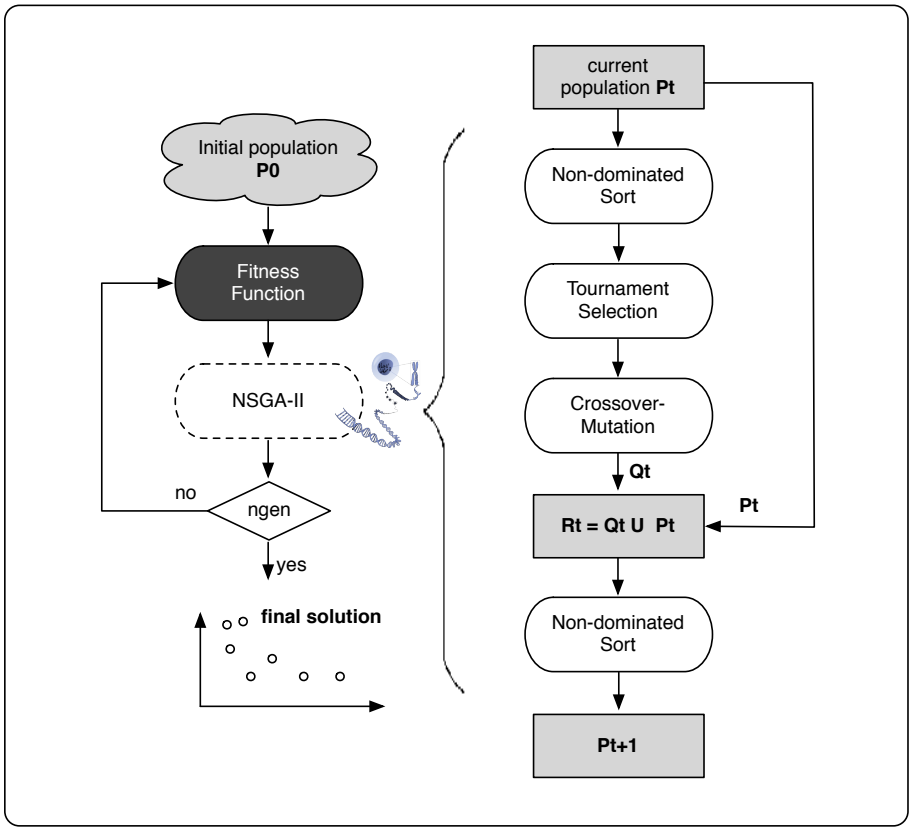


Figure 2.4: *Schematic of the NSGA-II procedure.*

first front have the best fitness value, they will be reproduced more than the rest of the population.

NSGA-II uses niching techniques (segmentation) providing each an individual parameter called *crowding distance*. This parameter measures the average side-length of the hypercube enclosing a solution without including any other solution in the population, as shown in Figure 2.5. Solutions of the last accepted front are ranked according to the crowded comparison distance. Crowding distance is used by the algorithm to ensure adequate distribution of individuals, in order to lead the population to adequately explore the entire space of objectives.

A detailed scheme of the procedure is illustrated in Figure 2.4. Initially,

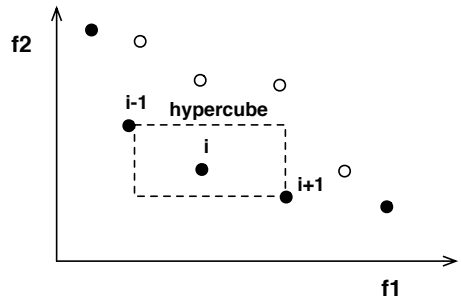


Figure 2.5: *the crowding distance.*

a population P_0 of size N is created; this population is sorted based on the non-domination through the fitness function (1 is the best level, 2 the next level and so on).

At this point, the offspring population is generated through the three operators for tournament selection, crossover and mutation.

Consequently, each of elements of the new population P_{t+1} is ranked and it is sorted in ascending order according to the Pareto dominance concept. The new parent population is composed by adding the solutions from the first front and the following fronts until exceeds the population size. Crowding distance is calculated during the population reduction phase and in the tournament selection for deciding the winner. The algorithm continues till the number of iterations *ngen* is reached.

2.2.3 Applications of NSGA-II: the Application Mapping Problem

This section will describe how a GA, and more specifically NSGA-II, can be deployed to perform mapping DSE for MPSoCs.

As explained before, NSGA-II uses an encoding as a string-like representation for each possible solution (the chromosome). In this case the problem is finding an optimal design candidate in a large space of possible design candidates that can be evaluated within Sesame as fitness function.

If there is a choice between M different types of processors and a maximum of N processors, then the meta-platform consists of all the possible platform components permutations and the mapping determines the final

configuration. A mapping between an application and a possible configuration of the parameterized SoC architecture corresponds to a chromosome of the NSGA-II. In particular, we use a gene for each parameter of the parameterized SoC architecture and allow that gene to assume only the values admissible by the parameter it represents; we assume that there are no functional restrictions on the processors: all processors can execute all of the tasks. Moreover, we assume to use a crossbar-based architecture, therefore each pair of processors can communicate so that there are no topological restrictions. The crossbar in the proposed platform fully connects all processors, so processes can communicate regardless on which processor they are mapped. The result is that any task can be mapped onto any processor so that we do not have to make special provisions for infeasible mappings. Given an application with N tasks and M processing elements, the mapping is a function that maps N tasks onto a M-processor system:

$$\begin{array}{ccc} Task_1 & \Rightarrow & Processor_1 \\ Task_2 & \Rightarrow & Processor_3 \\ Task_3 & \Rightarrow & Processor_1 \\ \cdots & & \cdots \\ Task_N & \Rightarrow & Processor_M \end{array}$$

The resulting chromosome C can be schematised as a vector of N processor identifiers:

$$C = [p_1, p_2, \cdots, p_{N-1}]$$

where the i-th index denotes the mapping target of task i.

All possible combinations of integers will result in valid mappings, as long as as those numbers are within the range of processor identifiers. This actually implies that all the crossover and mutation operators will result in a feasible mapping.

In case these conditions are not met, the so-called *repair mechanisms* can be used [33]: the repair mechanism presented in [33] repairs by randomly mapping the tasks to a feasible processor. There are three possible repair strategies (no-repair, moderate-repair, and extensive-repair

strategies): the first (no-repair) strategy repairs the invalid individual at the end of the optimization process, and all non- dominated solutions are output. The second one (moderate repair), repairs the individuals at the end of each variation step, thus allowing infeasible individuals to enter the mutation step. The last technique (full repair) repairs all invalid individuals immediately after every variation step, helping to explore new feasible areas over unfeasible solutions.

2.3 Design Metrics for analyzing Performance of DSE

There are two goals of an EMO procedure: (i) a good convergence to the Pareto-optimal front and (ii) a good diversity in obtained solutions. Since both are conflicting in nature, comparing two sets of trade-off solutions also require different performance measures. Three different sets of performance measures were used:

1. Metrics evaluating convergence to the known Pareto-optimal front (such as error ratio, distance from reference set, etc.),

2. Metrics evaluating the spread of solutions on the known Pareto-optimal front (such as spread, spacing), since the non-dominated solutions are required to cover a wide range for each objective function value, and

3. Metrics evaluating certain combinations of convergence and spread of solutions (such as hypervolume, coverage, R-metrics, etc.).

In the following subsection, we provide an overview of the deployed metrics in this work.

2.3.1 The Hypervolume

The hypervolume (HV) [96] indicates the closeness of the solution set to the reference Pareto front. The hypervolume represents the size of the region dominated by the solutions in the Pareto optimal set. The reference point

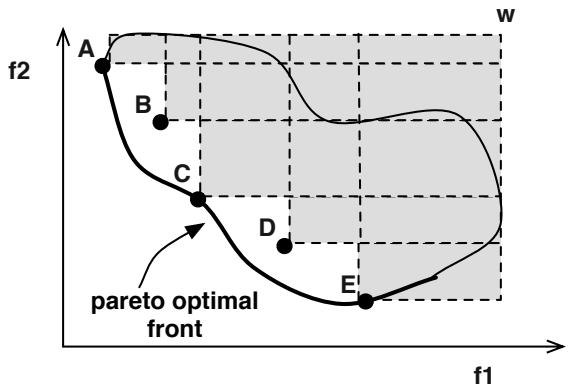


Figure 2.6: *The hypervolume enclosed by the non- dominated solutions.*

can simply be found by constructing a vector of worst objective values. The hypervolume metric is interesting because it is sensitive to the closeness of solutions to the true Pareto optimal set as well as the distribution of solutions across the objective space. The hypervolume value is calculated by summing the volume of hyper-rectangles constructing the hypervolume. A Pareto optimal set with a large value for the hypervolume is desirable [89]. The hypervolume represents the size of the region dominated by the solutions in the Pareto optimal set. In Figure 2.6, the gray region represents this metric for two objectives (f1 and f2) where these objectives are to be minimized. The reference point (W) can simply be found by constructing a vector of worst objective values.

2.3.2 Average Distance from Reference Set (ADRS)

This criterion corresponds to how much the heuristic solutions approximate the exact Pareto set after a fixed amount of simulations. In particular, we use the *Average Distance from Reference Set* (ADRS) [26], which measures the distance from the solution set $p(A)$ set and the Pareto-optimal set

$R = p(\Omega)$:

$$ADRS(p(A), R) = \frac{1}{|R|} \sum_{x_p \in R} \min_{\vec{a} \in p(A)} d\{\vec{x_p}, \vec{a}\}$$

where

$$d\{\vec{x_p}, \vec{a}\} = \max_{j=1,\cdots,M} \left\{0, \frac{f_j(\vec{a}) - f_j(\vec{x_p})}{f_j(\vec{x_p})}\right\}$$

and M is the number of objective functions.

A smaller ADRS value indicates that the distribution of the solutions is closer to the reference Pareto front, and therefore better.

2.3.3 The normalized ∇ metric

The normalized ∇ metric [33] measures the spread of solutions. It refers to the area of a hyper-rectangle formed by the two extreme solutions in the objective space, thus a bigger value spans a larger portion and therefore is better. The *nabla*-metric calculates the volume of a hyperbox formed by the extreme objective values observed in the Pareto optimal set:

$$\nabla = \prod_{m=1}^{M} (f_m^{max} - f_m^{min}) \tag{2.3}$$

Where M is the number of objectives, (f_m^{max} and f_m^{min}) the maximum and respectively minimum values of the m^{th} objective in the Pareto optimal set. A bigger value spans a larger portion and therefore is better. This metric does not reveal the exact distribution of intermediate solutions, so we have to use another metric for evaluating the distribution.

2.3.4 σ_{mst}-metric for measuring distribution

For measuring the distribution of solutions in a Pareto optimal set, we use the σ_{MST} metric [89]. The σ_{mst} is the standard deviation of the edges weights in the Minimum Spanning Tree (MST) generated by Pareto op-

timal solutions:

$$\sigma_{mst} = \sqrt{\frac{1}{|E|-1}\sum_{i=1}^{|E|}(\overline{w} - w_i)^2} \tag{2.4}$$

Where $|E|$ is the number of edges in the MST, w_i is the weight of the ith edge and $\overline{w}$ is the average weight of the edges in the MST. The σ_{mst} metric measures the standard deviation of the edges weights in the MST. The edges weights denote the minimum distances between connecting solutions. Therefore, a smaller value indicates that the distribution of the solutions is closer to the uniform distribution and thus is better.

2.4 The Sesame environment

The traditional practice for embedded systems evaluation often combines two types of simulators, one for simulating the programmable components running the software and one for the dedicated hardware parts. However, using such a hardware/software co-simulation environment during the early design stages has major drawbacks: (i) it requires too much effort to build, (ii) it is often too slow for exhaustive explorations, and (iii) it is inflexible in quickly evaluating different hardware/software partitionings. To overcome these shortcomings, a number of high-level modelling and simulation environments have been proposed in recent years. An example is our Sesame system-level modelling and simulation environment [78], which aims at efficient design space exploration of embedded multimedia system architectures.

In this thesis, we deploy this framework as fitness function for the GA-based DSE.

The Sesame framework [78], provides methods and tools for the efficient modelling and simulation of heterogeneous embedded multimedia systems. Using Sesame, a designer can model embedded applications and SoC architectures at the system-level, and map the former onto the latter to perform application-architecture co-simulations for rapid performance evaluations. Based on these evaluations, the designer can further

refine (parts of) the design, experiment with different hardware/software partitionings, perform co-simulations at multiple levels of abstraction, or mixed level co-simulations where architecture model components operate at different levels of abstraction. To achieve this flexibility, Sesame recognizes separate application and architecture models within a single system simulation. The application model defines the functional behavior of an application, including both computation and communication behaviors. The architecture model defines architecture resources and captures their performance constraints. An explicit mapping step maps an application model onto an architecture model for co-simulation.

2.4.1 Application layer

For application modeling, Sesame uses the Kahn Process Network (KPN) model of computation [52]. In a KPN, in which parallel and autonomous processes are implemented in a high-level language and they communicate with each other via unbounded FIFO channels. The communication and the synchronisation in a KPN is arranged by FIFO channels using blocking FIFO read and non-blocking write primitives. Applications specified as process networks allow a more natural mapping of processes to processing elements of the MPSoC architecture than a sequential program specification. Moreover, this model is *deterministic* and it fits with the targeted media-processing application domain. Determinism implies that the same application input always results in the same application output, irrespective of the scheduling of the KPN processes. This provides us with a lot of scheduling freedom when mapping KPN processes onto architecture models for quantitative performance analysis.

The code of each Kahn process is instrumented with annotations, which describe the application's computational actions, thus capturing the workload of an application. The reading from and writing to FIFO channels represents the communication behaviour of a process within the application model. In particular, when the Kahn model is executed, each process records its computational and communication actions, and generates a trace of *application events*. These application events are an abstract

representation of the application behaviour and are necessary for driving an architecture model. There are three types of application events: the *communication events* read and write and the *computational event* execute. Each event has a set of arguments to describe what is performed. For instance, the Execute(DCT) event describes that a Discrete Cosine Transform is performed. Read and Write events contain the information relative to the Kahn channel used for the communication and the amount of data transmitted, which, according to the application, may deploy different units as a pixel or a complete frame.

In Chapter 4 we employ also a subclass of the KPN model, which is called Polyhedral Process Network (PPN). In PPNs blocking read and write primitives are used.

Moreover, the functional behaviour of each process is expressed in terms of polyhedral descriptions. This implies that everything concerning the execution is known at compile-time, allowing the calculation of buffer sizes and schedules for merging processes.

2.4.2 Architecture Layer

The architecture model describes the hardware components in the system. The main function of this layer is simulating the performance (or power, as it will be discussed later on) consequences of the computation and communication events generated by the application model. Since the functional behaviour is already captured by the application model, which drives the architecture simulation, the architecture layer purely accounts for architectural (performance) constraints.

An architecture model is constructed from generic building blocks provided by a library, which contains template performance models for processing cores, communication media (like buses), and various types of memory.

The architecture models, implemented in Pearl [71], are highly parameterized black box models, which can simulate the timing characteristics of a programmable processor, a reconfigurable component, or a dedicated hardware core by simply changing the latencies associated to the incoming application events. The timing consequences of application events are

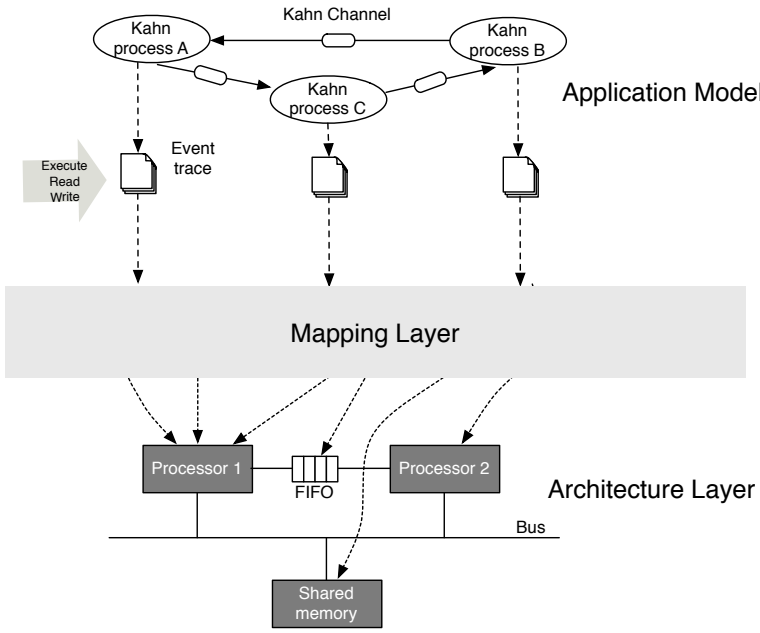


Figure 2.7: *A close-up of the layers in Sesame: application model layer, architecture model layer, and the mapping layer which is an interface between application and architecture models*

simulated by parameterizing each architecture model component with an event table containing operation latencies. The table entries can include, for example, the latency of an *execute* event, or the latency of a memory access (*read/write* event) in the case of a memory component. With respect to communication, issues such as synchronization and contention on shared resources are also captured in the architecture model.

Figure 2.7 illustrates a detailed view of layers in Sesame. In this example, the application model consists of four Kahn processes and five FIFO channels. The architecture model contains two processors and one shared memory. To decide on an optimum mapping, there exist multiple criteria to consider: maximum processing time in the system, power consumption and the total cost. This section aims at defining a mapping function, shown in Figure 2.7, to supply the designer with a set of best alternative mappings under the mentioned system criteria.

2.4.3 Mapping Layer

To realize trace-driven co-simulation of application and architecture models, Sesame has an intermediate mapping layer with two main functions. First, it controls the mapping of Kahn processes onto architecture model components by dispatching application events to the correct architecture model component. Second, it makes sure that no communication deadlocks occur when multiple Kahn processes are mapped onto a single architecture model component. In this case, the dispatch mechanism also provides various strategies for application event scheduling.

The mapping layer comprises of virtual processors and FIFO buffers for communication between the virtual processors. As illustrated in Figure 2.7, there is a one-to- one relationship between the Kahn processes in the application model and the virtual processors in the mapping layer. The same is true for the Kahn channels and the FIFO buffers in the mapping layer. However, the unbounded Kahn FIFO channels are mapped onto bounded FIFO buffers in the mapping layer. The size of the FIFO buffers in the mapping layer is parameterized and dependent on the architecture.

Mapping an application model onto an architecture model is illustrated in Figure 2.7. FIFO channels between the Kahn processes are also mapped (shown by the dashed arrows) in order to specify which communication medium is utilized for that data-exchange. If the source and sink processes of a FIFO channel are mapped onto the same processing component, the FIFO channel is also mapped onto the very component meaning that it is an internal communication. The latter type of communication is inexpensive as it is solely handled by the processing component and does not require access to other components in the architecture.

2.5 Conclusions

This chapter focused on evolutionary multi-objective algorithms and embedded systems design. We analyzed how to EMO algorithms can be used to solve DSE problems applied to embedded systems design. In order to

evaluate the fitness of the design points, we presented the Sesame simulation framework. Sesame is a high-level trace-based simulator which allows to explore different mapping configurations of streaming application onto MPSoC.

Chapter 3

Extending the objective space[1]

3.1 Introduction

An important element of system-level design is the high-level modelling for architectural power estimation. This allows to verify that power budgets are approximately met by the different parts of the design and the entire design, and evaluate the effect of various high-level optimizations, which have been shown to have much more significant impact on power than low-level optimizations [53].

Previously, the Sesame framework was mainly focused on the system-level performance analysis of multimedia MPSoC architectures. So, it did not include system-level power modelling and estimation capabilities. In [88], we introduced the concept of *computational event signatures*, allowing for high-level power modelling of microprocessors (and their local memory hierarchy). This signature-based power modelling operates at a higher level of abstraction than commonly-used instruction-set simulator (ISS) based power models and is capable of achieving good evaluation performance. This is important since ISS-based power estimation generally is not suited for early DSE as it is too slow for evaluating a large design space: the evaluation of a single design point via ISS-based simulation with a

[1] The contents of this chapter have been published in [2, 6, 4]

realistic benchmark program may take in the order of seconds to hundreds of seconds. Moreover, unlike many other high-level power estimation techniques, the signature-based power modelling technique still incorporates an explicit micro-architecture model of a processor, and thus is able to perform micro-architectural DSE as well.

In this chapter, we present a full system-level MPSoC power estimation framework based on the Sesame framework, in which the power consumption of all the system components is modelled using signature-based models. The MPSoC power model has been incorporated into Daedalus, which is a system-level design flow for the design of MPSoC based embedded multimedia systems [90, 73]. Daedalus offers a fully integrated tool-flow in which system-level synthesis and FPGA-based system prototyping of MPSoCs are highly automated. This allows us to quickly validate our high-level power models against real MPSoC implementations on FPGA.

Extending the Sesame framework to also support power modelling of MPSoCs could be done fairly easily by adding power consumption numbers to the event tables. So, this means that a component in the architecture model not only accounts for the timing consequences of an incoming application event, but also accounts for the power that is consumed by the execution of this application event (which is specified in the event tables now). The power numbers that need to be stored in the event tables can, of course, be retrieved from lower-level power simulators or from (prototype) implementations of components. However, simply adding fixed power numbers to the event tables would be a rigid solution in terms of DSE: these numbers would only be valid for the specific implementation used for measuring the power numbers. Therefore, we propose a high-level power estimation method based on so-called event signatures that allows for more flexible power estimation in the scope of system-level DSE. As will be explained in the next sections, signature-based power estimation provides an abstraction of processor activity and communication in comparison to traditional ISS-based power models, while still incorporating an explicit micro-architecture model and thus being able to perform micro-architectural DSE.

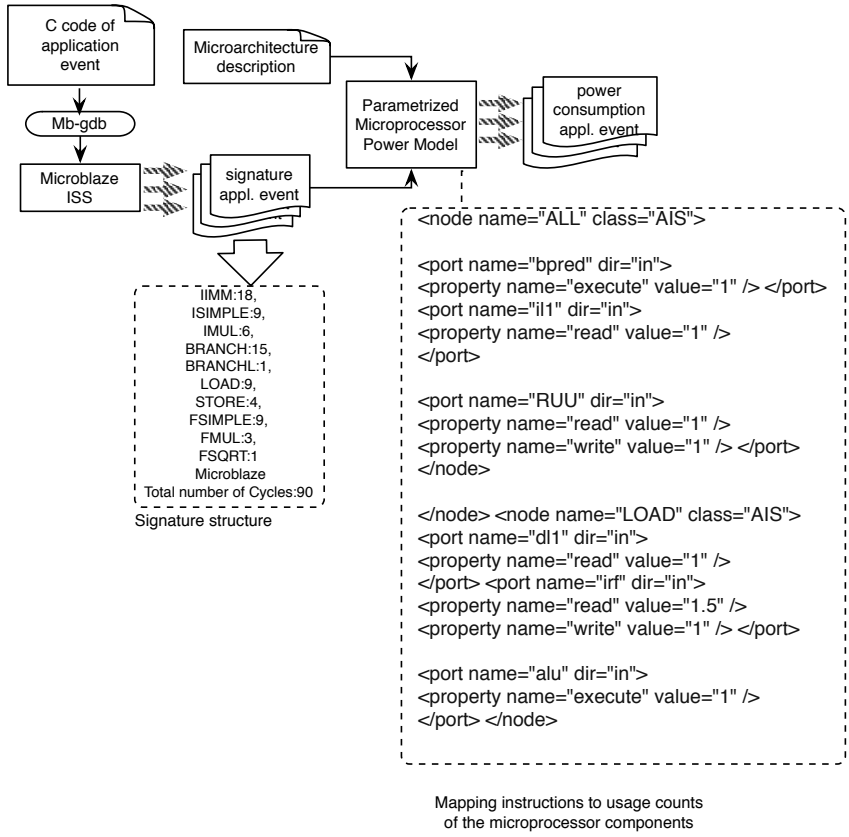


Figure 3.1: *Computational event signature generation for Microblaze*

3.2 Event signatures

An event signature is an abstract execution profile of an application event that describes the computational complexity of an application event (in the case of computational events) or provides information about the data that is communicated (in the case of communication events). Hence, it can be considered as meta-data about an application event.

3.2.1 Computational events signatures

A computational signature describes the complexity of computational events in a (micro-)architecture independent fashion using an Abstract Instruction Set (AIS) [88]. Currently, our AIS is based on a load-store architecture and consists of *instruction classes*, such as Simple Integer Arithmetic, Simple Integer Arithmetic Immediate, Integer Multiply, Branch, Load, and Store. The high level of abstraction of the AIS should allow for capturing the computational behaviour of a wide range of RISC processors with different instruction-set architectures. To construct the signatures, the real machine instructions of the application code represented by an application event (derived from an instruction set simulator as will be explained below) are first mapped onto the various AIS instruction classes, after which a compact execution profile is made. This means that the resulting signature is a vector containing the instruction counts of the different AIS instruction classes. Here, each index in this vector specifies the number of executed instructions of a certain AIS class in the application event. We note that the generation of signatures for each application event is a one-time effort, unless e.g. an algorithmic change is made to an application event's implementation.

To generate computational signatures, each Kahn application process is simulated using a particular Instruction Set Simulator (ISS), depending on the class of target processor the application will be mapped on. For example, we currently use ISSs from the SimpleScalar simulator suite [14] for the more complex multiple-issue processors, while we deploy the Microblaze cycle-accurate instruction-set simulator provided by Xilinx for the more simple soft cores. Taking the signature generation for the MicroBlaze processor as an example in Figure 3.1, application files are loaded into mb-gdb, which is the GNU C debugger for MicroBlaze. Mb-gdb is used to send instructions of the loaded executable files to the MicroBlaze instruction set simulator, which performs cycle-accurate simulation of the execution of the software programs, as in [76].

Using these ISSs, the event signatures are constructed – by mapping

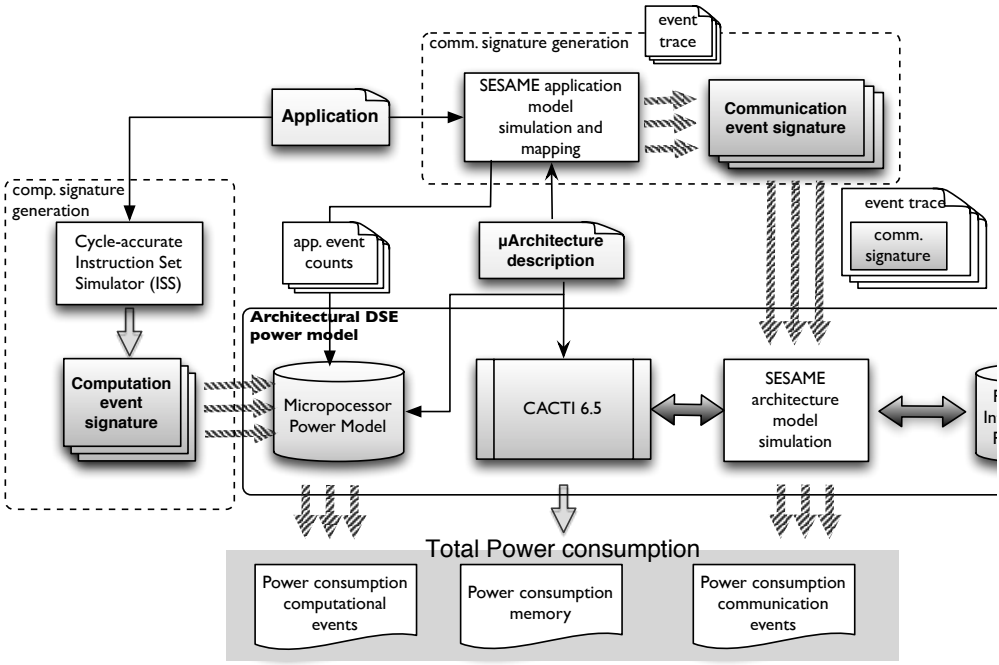


Figure 3.2: *System-level Power estimation framework*

the executed machine instructions onto the AIS as explained above – for every computational application event that can be generated by the Kahn process in question. The event signatures act as input to our parameterized microprocessor power model, which will be described in more detail in the next section. For each signature, the ISS may also provide the power model with some additional micro-architectural information, such as cache miss-rates, branch misprediction rates, etc. In our case, only instruction and data cache miss-rates are used. As will be explained later on, the microprocessor power model subsequently uses a micro-architecture description file in which the mapping of AIS instructions to usage counts of micro-processor components is described.

The microprocessor power model also uses a micro-architecture description file in which the mapping of AIS instructions to usage counts of microprocessor components is described. An example fragment of this

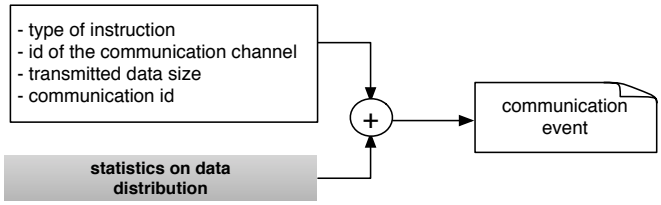


Figure 3.3: *Structure of communication events.*

mapping description is shown in Figure 3.1. It specifies that for every AIS instruction (indicated by the ALL tag), the instruction cache (il1) is read, the register update unit (RUU) is read and written, and branch prediction is performed. Furthermore, it specifies that for the AIS instruction LOAD, the ALU is used (to calculate the address), the level-1 data cache (dl1) is accessed, and that the integer register file (irf) is read and written. With respect to the latter, it takes register and immediate addressing modes into account by assuming 1.5 read operations to the irf on average. In addition, the micro-architecture description file also contains the parameters for our power model, such as e.g. the dimensions and organization of memory structures (caches, register file, etc.) in the microprocessor, clock frequency, and so on. Clearly, this micro-architecture description allows for easily extending the AIS and facilitates the modeling of different micro-architecture implementations.

3.2.2 Communication event signatures

In Sesame, the Kahn processes generate *read* and *write* communication events as a side effect of reading data from or writing data to ports. Hence, communication events are automatically generated. For the sake of power estimation, the communication events are also extended with a signature, as shown in Figure 3.3.

A communication signature describes the complexity of transmitting data through a communication channel (e.g., FIFO, Memory Bus, PLB Bus) based on the dimension of the transmitted data and the statistical distribution of the contents of the data itself.

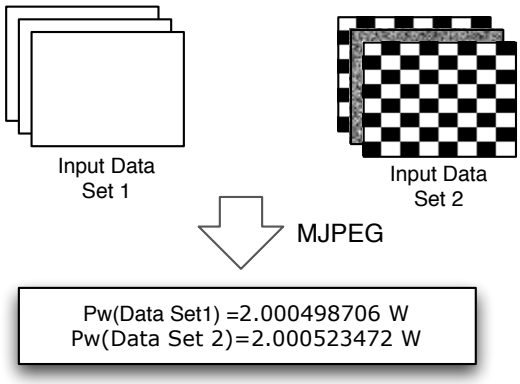


Figure 3.4: *Measured power consumption of MJPEG application mapped on one Microblaze using two different input sets.*

More specifically, we calculate the average Hamming distance of the data words within the data chunk communicated by a *read* or *write* event (which could be, e.g., a pixel block, or even an entire image frame), after which the result is again averaged with Hamming distance of the previous data transaction on the same communication channel. This way, we can get information about the usage of the channel and the switching factor, which is related to the data distribution. In our transaction-level architecture models, we use the assumption that the communications performed by the KPN application model are not interleaved at the architecture level. E.g., if a pixel block is transferred between two KPN processes, then the architecture model simulates the (bus/network) transactions of the consecutive data words in the pixel block, without interleaving these transactions with other ones. In Figure 3.4 we show the impact on power for a MJPEG application using input sets with different data distribution. In the first input data set picture, the correlation between pixel blocks is very high, and consequently the average Hamming distance of the data will be zero. This results in lower power values with respect of the second Input Data Set picture, which presents a higher Hamming Distance distribution.

3.2.3 Signature-based, system-level power estimation

In Figure 3.2, the entire signature-based power modeling framework is illustrated. First the event traces are generated, together with the communication signatures.

The Kahn application model is used to generate the event traces, which represent the workload that is imposed on the underlying MPSoC architecture model. During this stage, the *average Hamming distance*, as explained in the previous subsection, is computed. This information is then integrated in the trace events, forming the communication signature. The communication signature generation is mapping dependent: communication patterns change with different mappings. For instance, mapping two communicating tasks into the same processing unit reduces the data exchanged on the channels, thus decreasing the dynamic power due to communication. Conversely, mapping two tasks that exchange a lot of data into different processing units, increases the amount of exchanged data and thus the signature.

In addition, the computational signatures are generated (Figure 3.2, left side). In particular, the Kahn application processes for which a power estimation needs to be performed, are simulated using the ISS, constructing the event signatures (as explained in the previous section) for every computational application event that can be generated by the Kahn process in question. After the computational event signatures have been generated, the power consequences of trace events generated by the application model, are computed. As explained in the following section, we do this using a micro-architecture description file in the microprocessor power model, which describes the mapping of AIS instructions to usage counts of microprocessor components.

The Sesame architecture model simulates the performance and power consequences of the computation and communication events generated by the application model. To fullfill this task, each architecture model component is parameterized with an event table containing the latencies of the application events it can execute. Moreover, each architecture model com-

Table 3.1: *Different possibilities of reusing signatures in DSE*

comp. signatures	**comm. signatures**
μ-architectural exploration	μ-architectural exploration
mapping exploration (limited)	architectural exploration

ponent now also has an underlying signature-based power model. These models are activity-based. The activity counts are derived from the different application events in the event traces as well as the signature information of the separate events. The total power consumption is then obtained by simply adding the average power contributions of microprocessor(s), memories and interconnect(s).

The structure of the entire system-level power model is composed by separate and independent modules, which allow for the reuse of the different underlying component models as well as the generated signatures (as shown in Table 3.1). For example, once computational signatures are generated for application events, it is possible to explore different micro-architectures executing the same application with the same mapping. Moreover, given the computational event signatures, it is also possible to do mapping exploration, limited to the case of homogeneous systems, since using an heterogeneous system would require the regeneration of the computational events for each type of processing unit.

Communication signatures can be reused for both micro-architectural and architectural exploration.

3.3 MPSoC Power Model

We construct a high-level power estimation method for MPSoC based on the previously discussed event signatures that allows for flexible power estimation in the scope of system-level DSE. As will be explained in the subsequent subsections, signature-based power estimation provides an abstraction of processor (and communication) activity in comparison to e.g. traditional ISS-based power models, while still incorporating an explicit micro-

architecture model and thus being able to perform micro-architectural DSE. The power models are based on FPGA technology, since we have incorporated these models in our system-level MPSoC synthesis framework Daedalus [73], which targets FPGA-based (prototype) implementations. The MPSoC power model is formed by three main building blocks, modelling the microprocessors, the memory hierarchy and the interconnections respectively. The model is based on the activity counts that can be derived from the application events and their signatures as described before, and on the power characteristics of the components themselves, measured in terms of FPGA Look-Up Tables (LUTs) used. More specifically, we estimate through synthesis on FPGA the maximum number of LUTs used for each component. The resulting model is, therefore, a compositional power model, consisting of the various components (for which the models are described below) used in the MPSoC under study.

The currently modelled building blocks – network components as well as processor and memory components – are all part of the IP library of our Daedalus synthesis framework [73], which allows the construction of a large variety of MPSoC systems. Consequently, all our modeled MPSoCs can actually be rapidly synthesized to and prototyped on FPGA, allowing us to easily validate our power models.

In the remainder of this chapter, we will focus on homogeneous systems, but the used techniques do allow the modeling and simulation of heterogeneous systems as well.

3.3.1 Interconnection Power model

In this section, we derive architectural-level parameterized, activity based power models for major network building blocks within our targeted MPSoCs. These include FIFO buffers, crossbar switches, buses and arbiters.

Our network power models are composed of models for the aforementioned network building blocks, for which each of them we have derived parameterized power equations. These equations are all based on the com-

mon power equation for CMOS circuits:

$$P_{interconnect} = V_{dd}^2 \, fC\alpha \tag{3.1}$$

where f is the clock frequency, V_{dd} the operating voltage, C the capacitance of the component and α is the average switching activity of the component respectively. The capacitance values for our component models are obtained through an estimation of the number of LUTs used for the component in question as well as the capacitance of a LUT itself. Here, we estimate the number of LUTs needed for every component through synthesis, after which the capacitance is obtained using the X-Power tool from Xilinx [94]. The activity rate α is primarily based on the *read* and *write* events from the application event traces that involve the component in question. For example, for an arbiter component of a bus, the total time of read and write transactions to the arbiter (i.e., the number of *read* and *write* events that involve the arbiter) as a fraction of the total execution time is taken as the access rate (i.e., activity rate). Consequently, the power consumption of an arbiter is modelled as follows:

$$P_{arbiter} = \beta \times V_{dd}^2 \times f \times C_{LUT} \times n_{LUTs} \times access\_rate \tag{3.2}$$

where C_{LUT}, n_{LUTs}, f, V_{dd} are respectively the estimated capacitance of a LUT, the estimated number of LUTs needed to build the arbiter, the clock frequency and the operating voltage. β is a scaling factor obtained through an initial calibration of the model against real measurements, and

$$access\_rate = \frac{T_{reads} + T_{writes}}{T_{total\_exec}}$$

Here, T_{reads} and T_{writes} are the total times spend on the execution of read and write transactions, respectively, and T_{total_exec} is the total execution time.

For communication channels like busses, not only the number of *read* and *write* events play a role to determine the activity factor, but also the data that is actually communicated. For this purpose, we consider

the *Hamming Distance distribution* between the data transactions, as explained in the previous section on communication signatures. Thus, every communication trace event is carrying the statistical activity-based information of the channel from/to which the data is read/written. Consequently, for any activity (read/write of data) in the channel, the dynamic power of the interconnection is calculated according to technology parameters and the statistical distribution of the data transmitted. Hence, for every packet transmitted over the channel, the estimated power is computed in the following way:

$$P_{chan} = \beta \times V_{dd}^2 \times f \times C_{chan} \times n_{LUTs} \times Hamm\_dist(e) \tag{3.3}$$

where β, C_{chan}, f, V_{dd}, n_{LUTs} are again the scaling factor, estimated capacitance of the communication channel, clock frequency, the operating voltage, and number of LUTs needed to build the interconnection channel. The $Hamm\_dist(e)$ parameter is the average Hamming distance of the data transmitted in the *read/write* events. Leakage power in FPGAs is dependent on design-specific parameters; in particular, it is proportional to the amount of LUTs used in the architecture design. In our models, leakage power is calculated according to the estimated look-up tables needed to build a particular interconnection. Through XPower we initially compute the amount of LUTs deployed for each architectural component in our library and proportionally compute the estimated leakage power consumption by considering the final amount of LUTs of the platform design.

3.3.2 Memory Power model

For on-chip memory (level 1 and 2 caches, register file, etc.) and main memory, we use the analytical energy model developed in CACTI 6.5 [72] to determine the power consumption of read and write accesses to these structures. These power estimates include leakage power. The access rates for the processor-related memories, such as caches and register file, are derived from the computational signatures, as will be explained in the next subsection. Moreover, we use the cache miss-rate information provided by

the ISS used to generate the computational signatures to derive the access counts for structures like the level-2 cache and the processor's load/store queue.

For the main memory and communication buffers, we calculate the access rate in the same fashion as for a network arbiter component as explained above: the communication application events are used to track the number of accesses to the memory. That is, the total time taken by read and write accesses (represented by the communication application events) to a memory as a fraction of the total execution time is taken as the access rate. Subsequently, the signal rate represents the switching probability of the signals. For every read/write event to the memory, the average Hamming distance contained in the communication event signature is extracted and the signal rate is calculated as follows:

$$signal\_rate = \gamma \times Hamm\_dist(e)$$

where the γ is again a scaling factor obtained through pre-calibration of the model.

3.3.3 Microprocessor Power model

The microprocessor model that underlies our power model is based on [88]. It assumes a dynamic pipelined machine, consisting of one arithmetic logical unit, one floating point unit, a multiplier and two levels of caches. However, this model can easily be extended to other processor models, by simply introducing new units. For the power model of the clock component, three sub-components are recognized: the clock distribution wiring, the clock buffering and the clocked node capacitance. We assume a H-tree based clock network using a distributed driver scheme (i.e. applying clock buffers) [88].

The capacitance of the buffers is modeled to be a fraction of the capacitance of the wiring network. This fraction is dependent on the number of buffers, which is calculated by first taking the ratio of the capacitance of the wiring network and the capacitance of a single buffer. Over this

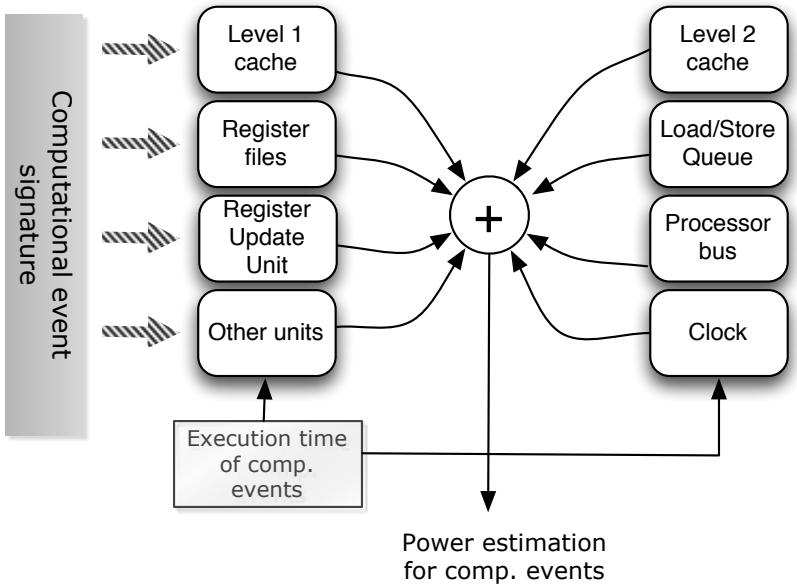


Figure 3.5: *Different components in the microprocessor power model*

the fourth root is taken, where the value four is actually a parameter, the optimal stage ratio, but this value is fixed within the model.

$$\text{buffers} = \sqrt[4]{\frac{C_{wiring}}{C_{single\_\text{buffer}}}} \tag{3.4}$$

$$C_{\text{buffers}} = C_{wiring} \times \frac{1}{1 - \frac{1}{\text{buffers}}} \tag{3.5}$$

For the clocked node capacitance $C_{clocked}$, only memory components are considered. Here, in [88] the authors use the number of read and write ports and the blocksize to calculate the capacitance:

$$C_{clocked} = ports \times blocksize \times C_{trans} \tag{3.6}$$

The capacity for switching a port is acquired from CACTI, and is equal to the capacitance of a transistor. The clocked node capacitance of each

memory structure is summed to the total clocked node capacitance.

The power consumption of a computational application event is calculated by accumulating the power consumption of each of the components that constitute the micro-processor power model, as shown in Figure 3.5. More specifically, the first step to calculate an application event's power consumption is to map its signature to usage counts of the various processor components. So, here it is determined how often e.g. the ALU (see Other Units in Figure 3.5), the register file and the level-1 instruction and data caches are accessed during the execution of an application event.

The microprocessor power model uses an XML-based micro-architecture description file in which the mapping of AIS instructions to usage counts of microprocessor components is described. This micro-architecture description file also contains the parameters for our microprocessor power model, such as e.g. the dimensions and organization of memory structures (caches, register file, etc.) in the microprocessor, clock frequency, and so on. Clearly, this micro-architecture description allows for easily extending the AIS and facilitates the modeling of different micro-architecture implementations.

The above ingredients (the event signatures, additional micro-architectural information per signature such as cache statistics, and the micro-architecture description of the processor) subsequently allow the power model to produce power consumption estimates for each computational application event by accumulating the power consumption of the processor components used by the application event.

3.4 The Daedalus Exploration Framework

Daedalus offers a fully integrated tool-flow in which design space exploration (DSE), system-level synthesis, application mapping, and system prototyping of MPSoCs are highly automated, which allows a direct validation and calibration of our power model. In Figure 1, the conceptual design flow of the Daedalus framework is depicted.

A key assumption in Daedalus is that the MPSoCs are constructed from

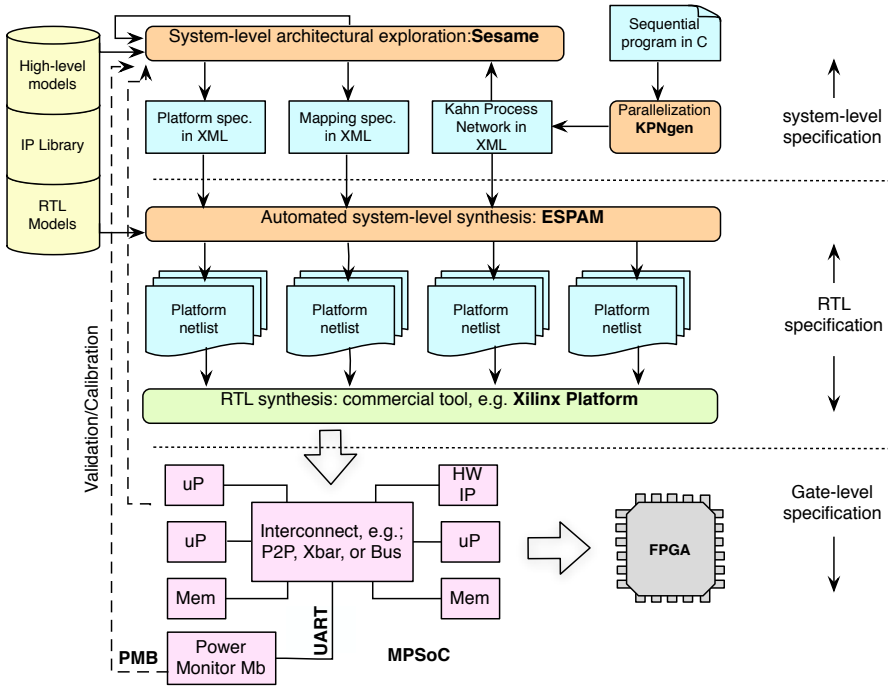


Figure 3.6: *The Daedalus design and validation tool-flow.*

a library of pre-defined and pre-verified IP components. These components include a variety of programmable and dedicated processors, memories and interconnects, thereby allowing the implementation of a wide range of MPSoC platforms. So, this means that Daedalus aims at composable MPSoC design, in which MPSoCs are strictly composed of IP library components. Daedalus consists of three core tools.

Starting from a sequential multimedia application specification in C, the KPNgen tool [52] allows for automatically converting the sequential application into a parallel Kahn Process Network (KPN) specification. Here, the sequential input specifications are restricted to so-called static affine nested loop programs, which is an important class of programs in, e.g., the scientific and multimedia application domains.

The generated or handcrafted KPNs (the latter in the case that, e.g.,

the input specification did not entirely meet the requirements of the KPN-gen tool) are subsequently used by the Sesame modeling and simulation environment [78],[64] to perform system-level architectural design space exploration. For this reason, Sesame uses (high-level) architecture model components from the IP component library (see the left part of Figure 3.6). As discussed before, Sesame allows for quickly evaluating the performance of different application to architecture mappings, HW/SW partitionings, and target platform architectures. Such exploration should result in a number of promising candidate system designs, of which their specifications (system-level platform description, application-architecture mapping description, and application description) act as input to the ESPAM tool [90],[73]. This tool uses these system-level input specifications, together with RTL versions of the components from the IP library, to automatically generate synthesizable VHDL that implements the candidate MPSoC platform architecture. In addition, it also generates the C code for those application processes that are mapped onto programmable cores. Using commercial synthesis tools and compilers, this implementation can be readily mapped onto an FPGA for prototyping. Such prototyping also allows for calibrating and validating Sesameś system-level models.

Ultimately, Daedalus aims at traversing an entire design flow going from a sequential application to a working MP-SoC prototype in FPGA technology with the application mapped onto it in a matter of hours. Evidently, this would offer great potentials for quickly experimenting with different MP-SoC architectures and exploring design options during the early stages of design.

3.5 Validation

As mentioned before, we have integrated our power model into the Daedalus system-level design flow for the design of MPSoC based embedded multimedia systems [90, 73]. This allows for direct validation and calibration of our power model.

Daedalus offers a fully integrated tool-flow in which design space ex-

ploration (DSE), system-level synthesis, application mapping, and system prototyping of MPSoCs are highly automated, which allows a direct validation and calibration of our power model.

3.5.1 Experimental results

By deploying Daedalus, we have designed several different candidate MPSoC configurations and compared our power estimates for these architectures with the real measurements. The studied MPSoCs contain different numbers of Microblaze processors that are interconnected using a crossbar network or a point-to point network. The microcontroller softcores on the FPGA device used in the framework do not use caches at this moment. This is considered to be future work. The validation environment is formed by the architecture itself and an extra Microblaze. This extra Microblaze polls the power values in the internal measurement registers in our target Virtex-6 FPGA, and interfaces an I2C controller in the FPGA design with the I2C interface of the PMBus controller chip [10]. In order to do this, it runs a software driver which implements the PMBus protocol [10]. The extra Microblaze prints out the values read through the UART to the pc (a 2.66Hz Intel dual core purely used to collect the output data), as shown in Figure 3.6. In this way, we have a fully automated system to register the power values of an architecture running a particular application with a given mapping. As we introduced an extra Microblaze in the design, the resulting power consumption of the system is scaled by a fixed factor, which is dependent on the measurement infrastructure. This is, however, not a problem since our primary aim is to provide high-fidelity rankings in terms of power behavior (which is key to early design space exploration) rather than obtaining near-perfect absolute power estimations [46]. Evidently, the additional power consumed by the extra Microblaze does not affect the fidelity of the rankings (i.e., the extra Microblaze exists in every MPSoC configuration), while the power measurements obtained are much more accurate compared to e.g. using a simulator [18].

The results of the validation experiments are shown in Figures 3.7,3.8,3.9 and 3.10. In the experiments we compare the total power consumption,

which is both leakage and dynamic power. In these experiments, we mapped three different parallel multimedia applications onto the target MPSoCs: a Motion-JPEG encoder (Mjpeg), a Periodogram, which is an estimate of the spectral density of a signal, and a Sobel filter for edge detection in images. In addition, for each of the applications, we also investigated two different task mappings onto the target architectures. Here, we selected one "good" mapping, in terms of task communication, as well as a "poor" one for each application. That is, in the "good" mapping we minimize task communications, while in the "poor" one we maximize task communications. The experiments in Figures 3.7,3.8,3.9,3.10 apply the following notation: $app_{name}\_n_{proc}\_mapping_{type}$, where app_{name} is the application considered, n_{proc} indicates the number of processors used in the architecture (e.g., "3mb" indicates an MPSoC with 3 MicroBlaze processors), and $mapping_{type}$ refers to the type of mapping used. With respect to the latter, the tag *mp1* indicates the good mapping, while *mp2* refers to the poor mapping. For the Motion-JPEG application, we also considered two different data input sets: the first input set (*ds1*) is characterized by a high data-correlation, while the second input set (*ds2*) has a very low data correlation, in terms of measured average *Hamming distance distribution* of the input data.

That is, we tested the power model on two different communication architecture configurations: the first one is crossbar-based, while the second one is a point-to-point network based on FIFOs. The power values in Figures 3.7,3.8,3.9,3.10 are scaled by a factor of 2W for the sake of improved visibility. Most charts show a very little difference between the good and bad configurations (mp1 vs mp2) for a number of processors greater than 2; this is explained by the fact that a design with a larger number of processors implies a higher use of the communication channels. Given an application with m tasks and n processors, if $m >> n$, then this implies that a good mapping can be beneficial for reducing tasks communication. However, in the case of $m = n$, the tasks mapping cannot avoid substantial communication.

The results in Figures 3.7,3.8,3.9,3.10 show that our power model per-

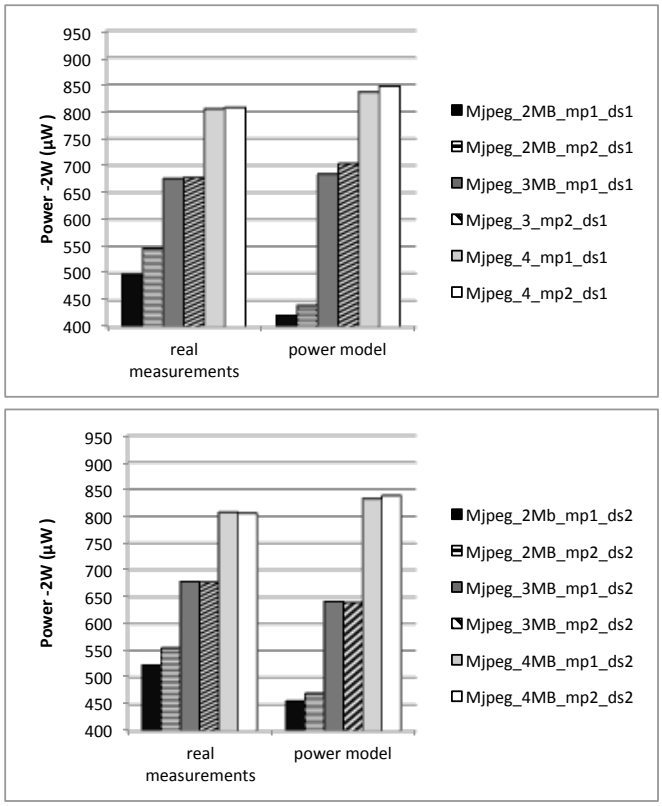


Figure 3.7: *Mjpeg application with input set ds1 (up) and input set ds2 (down) on a crossbar-based architecture*

forms quite decently in terms of absolute accuracy. We observed an average error of our power estimations of around 7%, with a standard deviation of 5% for the crossbar networks, and an average error of our power estimations of around 10%, with a standard deviation of 6% for the point-to-point networks. More important in the context of early design space exploration, however, is the fact that our power model appears to be very capable of estimating the right power consumption trends for the various MPSoC configurations, applications and mappings. We explicitly checked the fidelity of our estimations in terms of quality ranking of candidate architectures

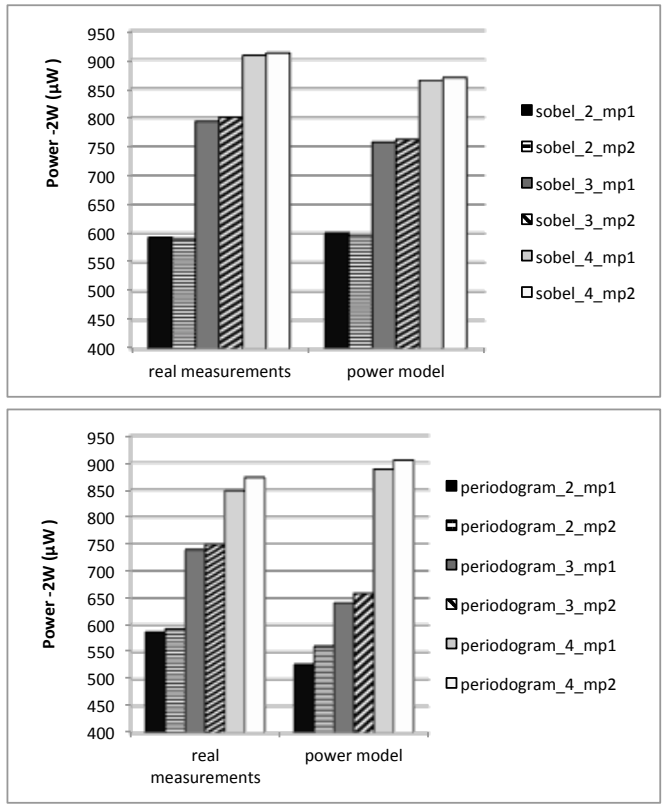


Figure 3.8: *Sobel filter (left) and Periodogram application (right) on a crossbar-based architecture*

by ranking all design instances according to their consumed power for a specific application. Our estimates result in a ranking of the power values that is correct for every application we considered, therefore, showing a high fidelity. This high-fidelity quality-ranking of candidate architectures thus allows for a correct candidate architecture generation and selection during the process of design space exploration.

Since every design point evaluation takes only 0.16 seconds on average, the presented power model offers remarkable potentials for quickly experimenting with different MPSoC architectures and exploring system-level design options during the very early stages of design.

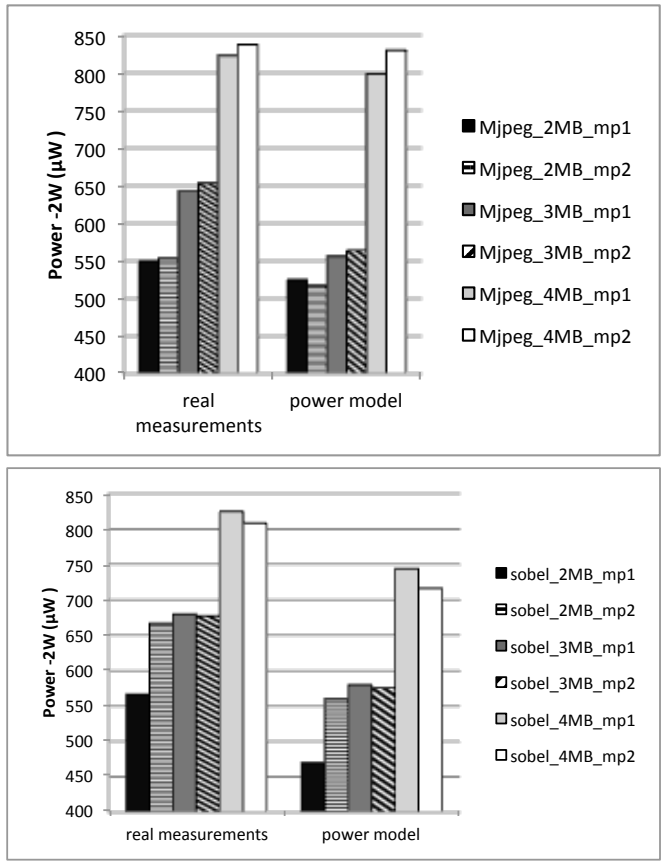


Figure 3.9: *Mjpeg (up) and Sobel (down) applications in a point-to-point FIFO architecture*

3.6 Related Work

There exists a fairly large body of related work on system-level power modeling of MPSoCs. For example, in [35] developed a SoC power estimation method based on a SystemC TLM modeling strategy. It adopts multi-accuracy models, supporting the switch between different models at run-time according to the desired accuracy level. The authors validate their model using the STBus NoC, and an analytical power model of this NoC. An MPEG4 application was tested, achieving up to 82% speed-up

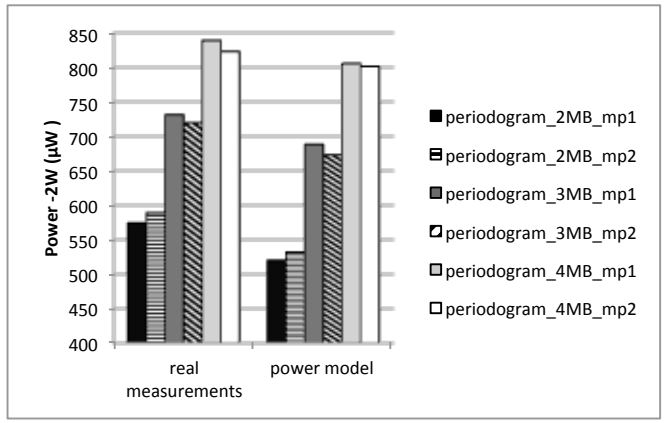


Figure 3.10: *Periodogram application in a point-to-point FIFO architecture*

compared to TLM BCA (Bus-Cycle Accurate) simulation.

Atitallah et. al. [15] uses a stack of abstract models. The higher abstraction model, named Timed Programmer View (PVT) omits details related to the computation and communication resources. Such an abstract model enables designers to select a set of solutions to be explored at lower abstraction levels. The second model, CABA (Cycle-Accurate Bit-Accurate), is used for power estimation and platform configuration.

In [70] a system-level cycle-based framework to model and design heterogeneous MPSoC (called GRAPES), is presented. C++/SystemC based IP system modules can be wrapped to act as plug-ins, which are managed by the simulation kernel in a TLM fashion. Those modules are managed by the GRAPES kernel, which is the core of the simulation framework. To estimate power during a simulation, they add a dedicated port to each component, which communicates with the corresponding power model. This feature permits to characterise each component with a set of Activity Monitors (inside the Component Module) necessary for the power estimation.

[92] presents a simulation-based methodology for extending system performance modelling frameworks to also include power modelling. They demonstrate the use of this methodology with a case study of a real, complex embedded system, comprising the Intel XScale embedded micro-

processor, its WMMX SIMD co processor, L1 caches, SDRAM, and the on-board address and data buses.

In [61], a power estimation framework for SoCs is presented, using power profiles to produce cycle accurate results. The SoC is divided in its building blocks (e.g. processors, memories, communication and peripherals) and the power estimation is based on the RTL analysis of each component. The authors validate the framework using an ARM926EJ-S CPU and the AMBA AXI 3.0 as NoC. Speed-up compared to a gate level simulation is on average 100 times faster. Previous work does not address high level power modelling of MPSoCs on FPGA.

In [80], an efficient Hybrid System Level (HSL) power estimation methodology for FPGA-based MPSoC is proposed. Within this methodology, the Functional Level Power Analysis (FLPA) is extended to set up generic power models for the different parts of the system.Then, a simulation framework is developed at the transactional level to evaluate accurately the activities used in the related power models. With respect to this work, our processor model can easily model different kinds of RISC processors by simply introducing new units.

Moreover, there also exist a considerable number of research efforts that only focus on the power modelling of the on-chip network of MPSoCs. Examples are [75, 45, 57, 63]. Many of the above approaches calibrate the high-level models with parameters extracted from RTL implementations, using low-level simulators for the architectural components. In [75], a rate-based power estimation method is presented. In the first phase it considers data-volume, estimating the average power in function of the total transmitted data: in the second phase, it calibrates the model through definition of the consumed power for each transition rate. In particular, the calibration uses a RTL model of the NoC, while the latter uses an actor-oriented model. After the calibration, a power dissipation table is generated for each injection rate and router element. Using linear approximation, they determine the power dissipation for each injection rate. In [45] an energy estimation model based on the traffic flow in the NoC's building blocks (routers and interconnection wires) is presented. The au-

thors represents the amount of energy consumed in the transmission of a data bit throughout the NoC (in its routers and interconnection wires). In [57] a NoC power and performance analysis with different traffic models, using analytical models, is presented. The authors target a NoC with a mesh topology. The employed traffic models are: uniform, local, hot-spot and matrix transpose. Results were compared to Synopsys Power Compiler and Modelsim, showing an error of 2% for power estimation and 3% for throughput. In [63] a methodology for accurate analysis of power consumption of message-passing primitives in a MPSoC is proposed, and, in particular, an energy model which allows to model the traffic-dependent nature of energy consumption through the use of a single, abstract parameter, namely, the size of the message exchanged. An ISS performs cycle accurate simulation of the cores, while the rest of the system is described in SystemC at signal level. In [31], the authors employ a framework that takes as input message flows, and derives a power profile of the network fabric. The authors map the CPU data-path as a graph, and the application as a set of messages that flow in this graph. Those mapped CPUs are connected into the network fabric, mapping the entire MPSoC as a network. The authors make use of a network power estimation tool, called LUNA, to evaluate the power dissipation of the entire MPSoC.

To the best of our knowledge, none of the previous existing efforts have incorporated the power models in a (highly automated) system-level MPSoC synthesis framework, allowing for accurate and flexible validation of the models. Instead, most existing works either use simulation-based validation (e.g. [35, 45, 57, 31, 75]), or validation by means of measurements on fixed target platforms (e.g. [92, 61]). Consequently, in general, related system-level MPSoC modeling efforts do also not target FPGA technology in their system-level power models. A recent contribution similar to our approach is presented by [84]. In [84], the authors propose a very accurate energy model for streaming applications modelled as Polyhedral Process Networks (PPN) and mapped onto tile-based MPSoC platforms with distributed memory. The energy model is based on the well- defined properties of the PPN application model. To guarantee the accuracy of the

energy model, values of important model parameters are obtained by real measurements. The main difference with our approach is that this model requires the analysis of communication contention for each mapping using System-C simulations. This results in more accurate results but slower power estimation time.

3.7 Conclusion

We presented a framework for high-level power estimation of multiprocessor systems-on-chip (MPSoC) architectures on FPGA. The technique is based on abstract execution profiles called "event signatures", and it operates at a higher level of abstraction than, e.g., commonly-used instruction-set simulator (ISS) based power estimation methods and should thus be capable of achieving good evaluation performance. The model is based on the activity counts from the signatures, and from the power characteristics of the components themselves, measured in terms of LUTs used. The signature-based power modeling technique has been integrated in our Daedalus system-level MPSoC synthesis framework, which allows a direct validation and calibration of the power model. We compared the results from our signature-based power modeling to those from real measurements on a Virtex 6 FPGA board. These validation results indicate that our high-level power model achieves good power estimates in terms of DSE.

Since every design point evaluation takes only 0.16 seconds on average, the presented power model offers remarkable potentials for quickly experimenting with different MPSoC architectures and exploring system-level design options during the very early stages of design.

Chapter 4

Pruning techniques for performance estimation[1]

4.1 Introduction

Methods for evaluating a single design point in the design space roughly fall into one of three categories: 1) measurements on a (prototype) implementation, 2) simulation based measurements and 3) estimations based on some kind of analytical model. Each of these methods has different properties with regard to evaluation time and accuracy. Evaluation of prototype implementations provides the highest accuracy, but long development times prohibit evaluation of many design options. Analytical estimations are considered the fastest, but accuracy is limited since they are typically unable to sufficiently capture particular intricate system behavior. Simulation-based evaluation fills up the range in between these two extremes: both highly accurate (but slower) and fast (but less accurate) simulation techniques are available. This trade-off between accuracy and speed is very important, since successful design space exploration (DSE) depends both on the ability to evaluate a single design point as well as being able to efficiently search the entire design space. Current DSE efforts typically use

[1]The contents of this chapter have been published as [79, 5]

simulation or analytical models to evaluate single design points together with a heuristic search method [39] or statistical techniques [51, 82, 95] to search the design space. These DSE methods search the design space using only a finite number of design-point evaluations, not guaranteeing to find the absolute optimum in the design space, but they reduce the design space to a set of design candidates that meet certain requirements or are close to the optimum with respect to certain objectives.

Our focus is on *system-level mapping DSE*, where mapping involves two aspects: 1) allocation and 2) binding. Allocation deals with selecting the architectural components in the MPSoC platform architecture that will be involved in the execution of the application workload (i.e., not all platform components need to be used). Subsequently, the binding specifies which application task or application communication is performed by which MPSoC component. As mentioned above, state-of-the-art DSE approaches typically use either simulation or an analytical model to evaluate mappings, where simulative approaches prohibit the evaluation of many design options due to the higher evaluation performance costs and analytical approaches suffer from accuracy issues. This chapter deals with a new, hybrid form of DSE, combining simulations with analytical estimations to prune the design space in terms of application mappings that need to be evaluated using simulation. To this end, the DSE technique uses an analytical model that estimates the expected throughput of an application (which is a natural performance metric for the multimedia and streaming application domain we target) given a certain architectural configuration and application-to-architecture mapping. In the majority of the search iterations of the DSE process, the throughput estimation avoids the use of simulations to evaluate the design points. However, since the analytical estimations may in some cases be less accurate, the analytical estimations still need to be interleaved with simulative evaluations in order to ensure that the DSE process is steered into the right direction.

We studied different techniques for interleaving these analytical and simulative evaluations in our hybrid DSE. We will demonstrate that by properly interleaving the analytical and simulative estimations, signific-

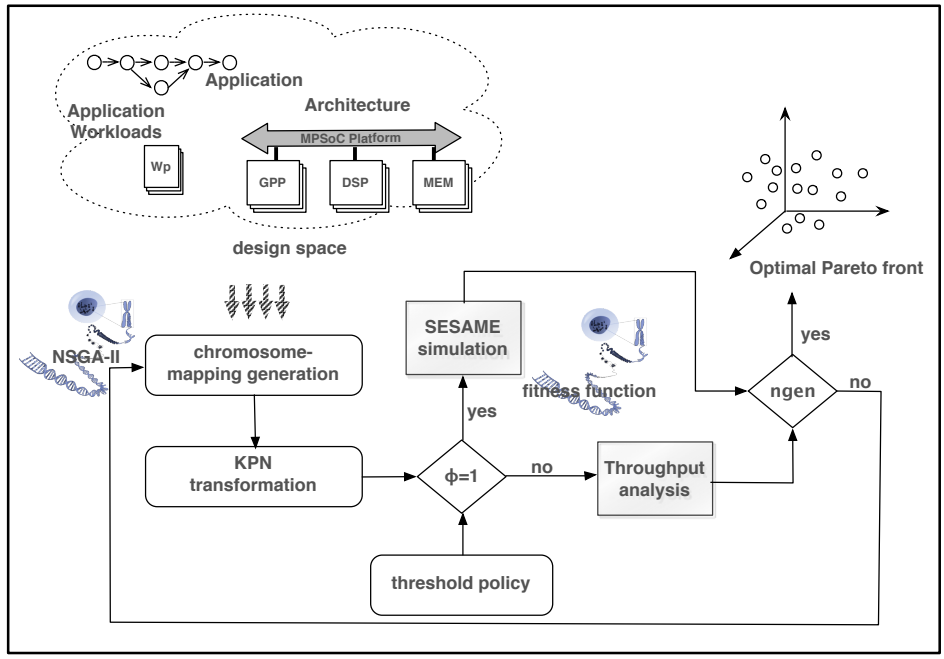


Figure 4.1: *Driving experiments with the expected throughput.*

ant efficiency improvements can be obtained while still producing similar solutions in terms of quality as compared to pure simulation-based DSE.

4.2 Combining throughput analysis and simulation

To evaluate design points during system-level DSE by means of simulation, we deploy the Sesame simulation framework [78]. As Sesame allows for rapid performance evaluation of different MPSoC architecture designs, application to architecture mappings, and hardware/software partitionings with a typical accuracy of 5% compared to the real implementation [78, 73].

In Figure 4.1, the entire DSE framework is shown. We adopt a hybrid approach where Sesame simulations are interleaved with analytical throughput analysis. The throughput analysis is based on the application graph, the individual task workloads and the mapping. It is used to

quickly predict the performance consequences of different design points as represented by the application mapping on the underlying architecture. As these fast analytical evaluations are interleaved with the slower simulative evaluations in a way such that most evaluations are performed analytically, this approach significantly improves the efficiency of the DSE process. Consequently, this would allow for searching a much larger design space.

The application is represented as a Kahn Process Network (KPN) [52]. As will described in the next section, before performing the throughput analysis, we need to perform some transformations to the application graph of the KPN in order to take into account mapping decisions. The subsequent throughput analysis – performed on the transformed KPN – should be fast and capable of correctly capturing the throughput trend for different mappings. The analysis requires the process workloads W_{P_i} as a parameter for the throughput modelling. The workload W_{P_i} of an application process P_i denotes the number of time units that are required to execute a single invocation of the process on a particular processor, i.e., the pure computational workload, excluding the communication. It should be provided by the designer who can obtain it, for example, by executing the process once on the target platform, or by using an instruction set simulator.

As will be shown later on, the analytical throughput model may encounter accuracy problems when the application graph is cyclic. To correct such errors during DSE, we interleave the throughput estimation with real simulations, according to the value of a binary function ϕ, which can be set according to a predefined policy: $\phi = 1$ implies that a real simulation is used and $\phi = 0$ means that an analytical estimation is used.

In our DSE framework, we use the widely-used NSGAII genetic algorithm [29] to actually search through the mapping design space. This results in a hybrid DSE method with the following steps, as shown in Figure 4.1:

1. Perform an initial model calibration and generate the application workloads W_{P_i}, as explained in [73]. This is a one-time effort, and the same for both the simulation model and analytical throughput

model.

2. Generate an initial population of unique mappings.

3. Transform the application KPN according to the mappings in the population and build the corresponding merged KPNs (as will be explained in the next section).

4. Perform the static throughput analysis for the merged KPN graphs and identify the best mappings based on the highest estimated throughput.

5. In case of $\phi = 1$, we interleave the throughput analysis with real simulation, in order to correct the ranking obtained in the previous steps of the NSGAII evolutionary algorithm.

6. Verify the stopping criterion. If the mapping population within the NSGAII algorithm remains unchanged or a maximum number of iterations has been performed, the algorithm stops. Otherwise, change the mapping population using NSGAII's genetic operators, and restart from the third step.

4.3 Modeling application mappings as merged Kahn Process Networks

Applications in our DSE framework are modeled using KPNs [52], in which parallel processes communicate with each other via unbounded FIFO channels.

Starting from a KPN, to perform throughput analysis one needs to take into account the mapping since the performance is mapping dependent. As we want to perform the throughput analysis only at the KPN level, we have to represent the mapping inside the KPN itself. For this purpose, we use *merging transformations* on the KPN to reflect the mapping of the different processes. Consequently, if two processes are mapped onto the same architectural component, they are merged into a single process in the

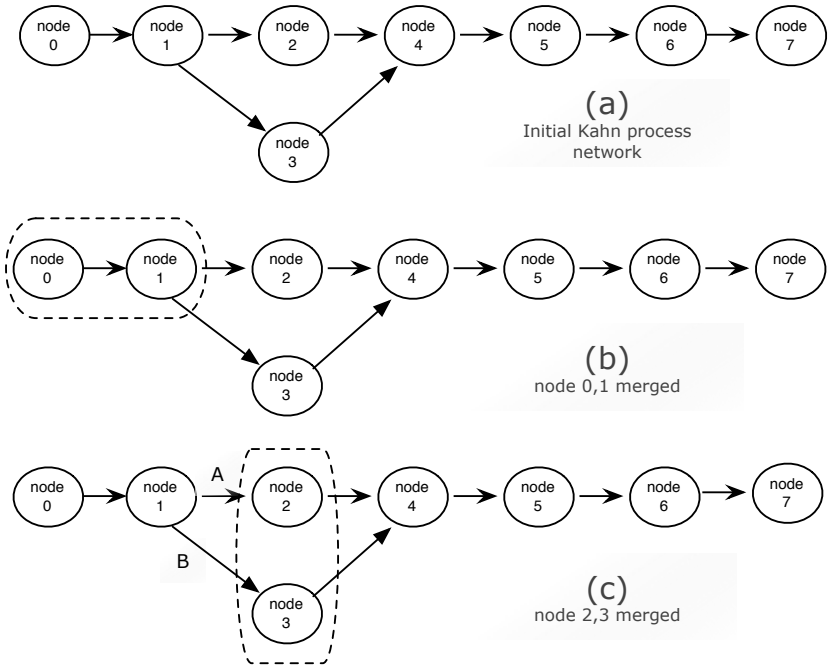


Figure 4.2: *Process merging in an example Kahn Process network.*

KPN, as is illustrated in Figure 4.2. Figure 4.2(a) shows the initial example KPN consisting of eight processes. Performing throughput analysis on this KPN assumes that each process is mapped onto a different processor and each KPN channel is mapped onto a unique communication memory in the MPSoC (i.e., all the connections are point-to-point connections). The KPNs in Figures 4.2(b) and 4.2(c) subsequently reflect the decisions that, respectively, KPN processes 0,1 and 2,3 are mapped onto a single processor. Mapping multiple KPN tasks onto one processor allows for MPSoC implementations with less processing and communication components, i.e. with reduced implementation cost, but at the cost of potentially additional execution overhead. For example, in case of a homogeneous MPSoC and a KPN model in which processes exchange data tokens of uniform size, the performance of such mapping *in terms of throughput* can only be the same or lower (so never higher) than the performance of a mapping in which

each task is mapped onto a different processor [66].

Subsequently, to assess the performance of a mapping decision, we perform throughput analysis on the transformed KPN.

4.3.1 Process Throughput and Throughput Propagation

Our throughput analysis is based on and extends the work presented in [66], in which the solution approach for the overall KPN throughput modeling relies on calculating the throughput τ_{P_i} of a process (i.e., node) P_i for all KPN processes and propagation of the lowest process throughput to the sink process. Here, we use a depth first search to determine the order of the processes for propagating throughputs. For a process P_i, the propagation consists of selecting either the aggregated incoming FIFO throughput $\tau_{F_{aggr,P_i}}$ or the isolated process throughput $\tau_{P_i}^{iso}$.

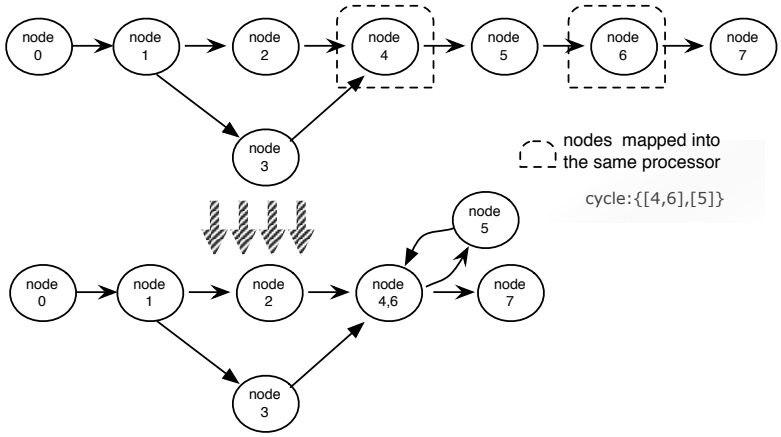
The isolated throughput $\tau_{P_i}^{iso}$ is the throughput of a process P_i when it is considered to be completely isolated from its environment. This means that the isolated process throughput is determined only by the workload W_{P_i} of a process and the number of FIFO reads/writes per process execution provided that no blocking occurs:

$$\tau_{P_i}^{iso} = \frac{1}{W_{P_i} + x \cdot C^{Rd} + y \cdot C^{Wr}} \tag{4.1}$$

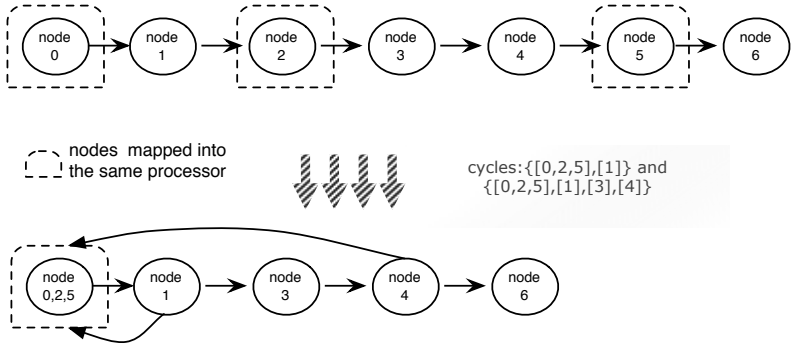
where x and y denote how many FIFOs are read and written per process execution and C^{Rd} and C^{Wr} the performance costs for reading/writing a token from/to a FIFO channel. The throughput of a FIFO-channel f is determined by the throughput of the processes accessing it:

$$\tau_f = min(\tau_f^{Wr}, \tau_f^{Rd}) \tag{4.2}$$

Subsequently, the throughput τ_{P_i} of a process P_i is determined by either the throughput of the FIFOs from which process P_i receives its data or by the computational workload of the process itself, i.e., $\tau_{P_i}^{iso}$. For merged KPN processes, the incoming FIFO throughput is the aggregated throughput of



(a) An example of mapping that generates one cycle



(b) An example of mapping that generates two cycles

Figure 4.3: *Transformation into a cyclic KPN.*

the merged channels and the isolated throughput is calculated using the aggregated computational workloads. Consequently, the throughput associated to each process in an acyclic KPN graph is computed as:

$$\tau_{P_i} = min(\tau_{F_{aggr,P_i}}, \tau_{P_i}^{iso}) \tag{4.3}$$

For example, for the merged processes 2,3 in Figure 4.2(c),

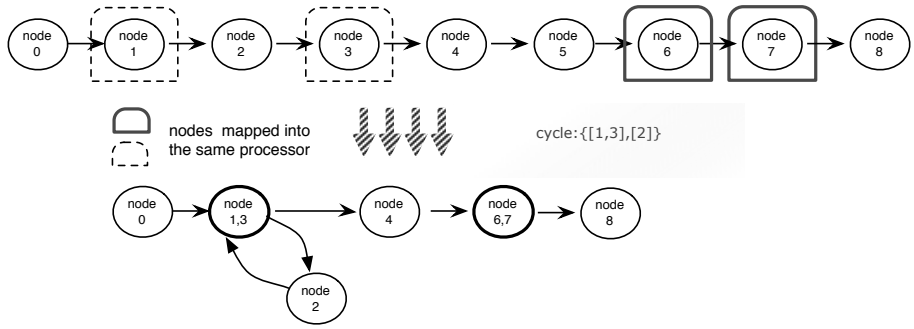


Figure 4.4: *Another example of process merging in a Process network.*

$$\tau_{F_{aggr,P_{2,3}}} = \tau_{f_a} + \tau_{f_b}$$

and

$$\tau^{iso}_{P_{2,3}} = \frac{1}{W_{P_2} + W_{P_3} + 2 \cdot C^{Rd} + 2 \cdot C^{Wr}}$$

.

4.3.2 Handling cycles

It is possible that the aforementioned merging transformation to account for mapping decisions might introduce new cycles in the transformed KPN. As shown in Figure 4.3(a), if processes 4,6 are mapped onto the same processor, this results in a cycle containing process 5 and the merged process $4, 6$. In Figure 4.3(b), processes 0, 2 and 5 are mapped to the same processor, resulting in a KPN with two cycles.

Cycles in a KPN are responsible for sequential execution of some of the processes involved in the cycle. The sequential execution can vary from a single initial delay to a delay at each execution of some of the processes. For throughput modeling, these cycles must be taken into account, but to do so in an accurate fashion is not trivial.

In Figure 4.4, processes 1,3 are mapped onto the same processor, this results in a cycle containing process 2 and the merged process $1, 3$. For nodes 6, 7 the resulting merged node does not generate any cycle. Previous

research on throughput analysis of KPNs has not addressed the handling of cycles. In [66], the authors only consider acyclic KPN graphs. A preliminary process throughput analysis in case of dataflow loops is suggested in [42] in terms of *mapping rules*, but the proposed rules have never been elaborated nor verified. Based on this approach, we conservatively approximate the isolated throughput of a process P_i that is member of a cycle by:

$$\tau^{iso}_{cyc_{P_i}} = \frac{1}{\sum_{P_j \in Cycle} \frac{1}{\tau^{iso}_{P_j}}} \tag{4.4}$$

From equation 4.4, it is clear that the isolated throughput of a cycle is lower than the regular isolated throughput ($\tau^{iso}_{P_i}$) of any of the processes involved in the cycle. It also implies that the isolated throughput of a cycle can be lower than the isolated throughput of the bottleneck process. This is an important observation because, in such a case, the throughput of the cycle will determine the overall KPN performance. To conclude, the throughput associated to each process P_i will be computed as:

$$\tau_{P_i} = min(\tau^{iso}_{cyc_{P_i}}, \tau_{F_{aggr,P_i}}, \tau^{iso}_{P_i}) \tag{4.5}$$

For example, in Figure 4.3(b) two cycles are generated due to the KPN transformation. In this case, we assume that the resulting $\tau^{iso}_{cyc_{P_i}}$ for a process P_i would be

$$\tau^{iso}_{cyc_{P_i}} = min(\tau^{iso}_{cyc_{P_i}}(1), \ldots, \tau^{iso}_{cyc_{P_i}}(n)) \tag{4.6}$$

where $\tau^{iso}_{cyc_{P_i}}(1), \ldots, \tau^{iso}_{cyc_{P_i}}(n)$ are all the throughputs of the cycles involving process P_i.

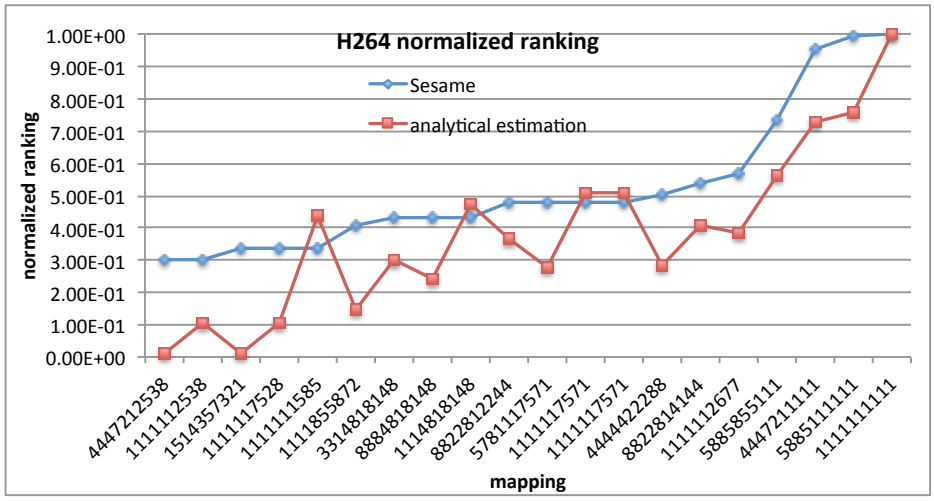


Figure 4.5: *Ranking a GA population using analytical estimation and Sesame simulation.*

4.3.3 A hybrid DSE approach

We use a fast but conservative approximation to estimate the performance in case of cycles. As a consequence, the analytical throughput analysis may present inaccuracies in case cycles are introduced in the transformed KPN, especially when there are many and/or complex cycles [79]. There are more detailed analytical approaches (like SDF^3 [36]) that allow for accurately computing cyclic performance behaviour but these types of analysis are generally very computationally intensive and thus slower then our Sesame simulations.

To demonstrate these inaccuracies, please consider Figure 4.5. For a DSE experiment with a H264 decoder application, this graph shows a snapshot of a single GA search iteration. More specifically, it shows the performance ranking of the design points (i.e. mappings) in the population of the search iteration when evaluating them either using Sesame or analytical estimation. The y-axis shows the normalized performance and the x-axis shows the different design points in the GA population, where the integer strings refer to the processor identifiers the application processes

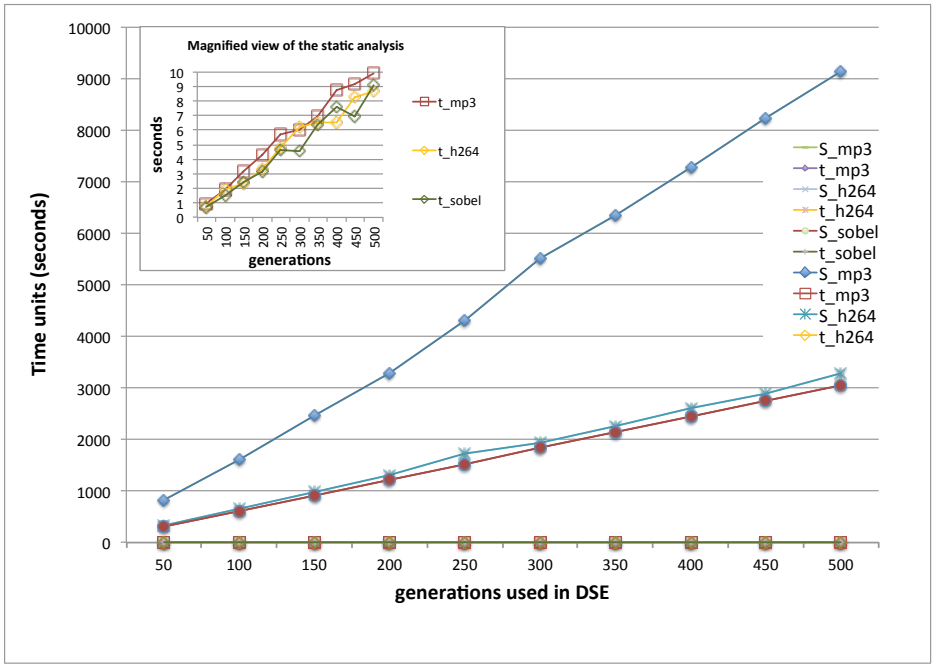


Figure 4.6: *DSE times using different methods.*

are mapped on. E.g., a string "1111112677" means that tasks 1 to 6 are mapped on processor 1, task 7 is mapped on processor 2, etc. The design points on the x-axis have been ordered according to the performance estimation as obtained by Sesame. This implies that the Sesame-based ranking of the population shows a monotonically increasing curve. However, as this is not true for the curve of the analytical estimations, this ranking clearly shows prediction errors.

4.4 Interleaving methods

Using analytical throughput estimation as fitness function during DSE can yield significant efficiency improvements. To demonstrate this, Figure 4.6 shows the wall-clock times for a DSE experiment, using a NSGAII GA, for a heterogeneous 8-processor MPSoC and three multimedia applications: an Mp3 decoder, a H264 decoder, and a Sobel filter for edge detection in

images. The curves labeled with an "S_" prefix show the DSE times when only using Sesame simulations, as a function of the number of generations used in NSGAII. The curves with a "t_" prefix show the results of exclusively using static throughput estimation during DSE. Clearly, the DSE based on analytical throughput analysis can be three orders of magnitude faster than simulation-based DSE. Avoiding simulation-based evaluations by replacing them with analytical evaluations, therefore, appears to be a promising technique for optimizing the DSE process.

The question that remains open is how to exactly perform the interleaving between analytical and simulative evaluations. Here, the ratio between the number of analytical and simulative evaluations plays an important role as this provides a valuable accuracy-performance trade-off. In addition, the decision of *when* (in time) to perform a simulative evaluation instead of an analytical estimation is an important factor in the interleaving strategy. In the remainder of this section, we will propose different strategies for interleaving analytical and simulative evaluations, which subsequently will be assessed in the next section.

4.4.1 Fixed-frequency interleaving

This is the simplest form of interleaving in which a fixed frequency K is chosen such that every K-th search iteration is performed using simulation-based evaluation instead of analytical estimation. For example, in case of 100 search iterations and $K = 10$, every 10th search iteration is performed using Sesame simulation, thereby reducing the number of simulations by 90% (9 out of 10 search iterations are performed using analytical estimation).

4.4.2 Switching method based on the bisection of the generation space

In this method, we divide the iteration space of the DSE according to the number of generations of the genetic algorithm used (NSGAII in our case). We switch from one method to the other (from simulation to analytical estimation, or vice versa) according to the number of generations executed

by the genetic algorithm, as illustrated in Figure 4.7. More specifically, $\phi = 1$ if $c_{gen} \geq K$ (or $\leq K$), where c_{gen} is the current search iteration and n_{gen} is the total number of DSE generations.

Figure 4.7: *NSGAII generation space using the bisection method.*

4.4.3 Temperature-based interleaving

This method is inspired by the simulated annealing technique. That is, the probability of performing a Sesame simulation increases with the number of generations examined in the genetic algorithm. More specifically,

$$\phi = 1 \text{ if } U([0,1]) \geq T$$

$$\text{where } T = \frac{c_{gen}}{n_{gen}}$$

where $U([0,1])$ is a uniform random distribution, and the temperature T is given by the ratio between the current generation c_{gen} and the total number of generations n_{gen} used for the design space exploration.

4.4.4 Population-property based interleaving

The last method we propose is based on the properties of the population in each generation of the GA. It bases the decision of whether to use simulation or analytical estimation on the percentage of design points in the GA's population that contains a cycle in the generated mapping. More specifically, the decision is based on the following algorithm:

for $d_i \in population$ **do**
verify if d_i generates a cycle
if d_i contains a cycle **then**
n_{cycles}++;
end if
end for
if $\frac{n_{cycles}}{n_{pop}} \times 100 \geq$ K **then**
$\phi = 1$;
end if

where n_{pop} is the number of mappings in the population and the threshold value K is a chosen proportion of the population.

Figure 4.8: *Average hypervolume and ∇ values for the different DSE methods applied to the Mp3 decoder, H264 decoder, and Sobel filter applications.*

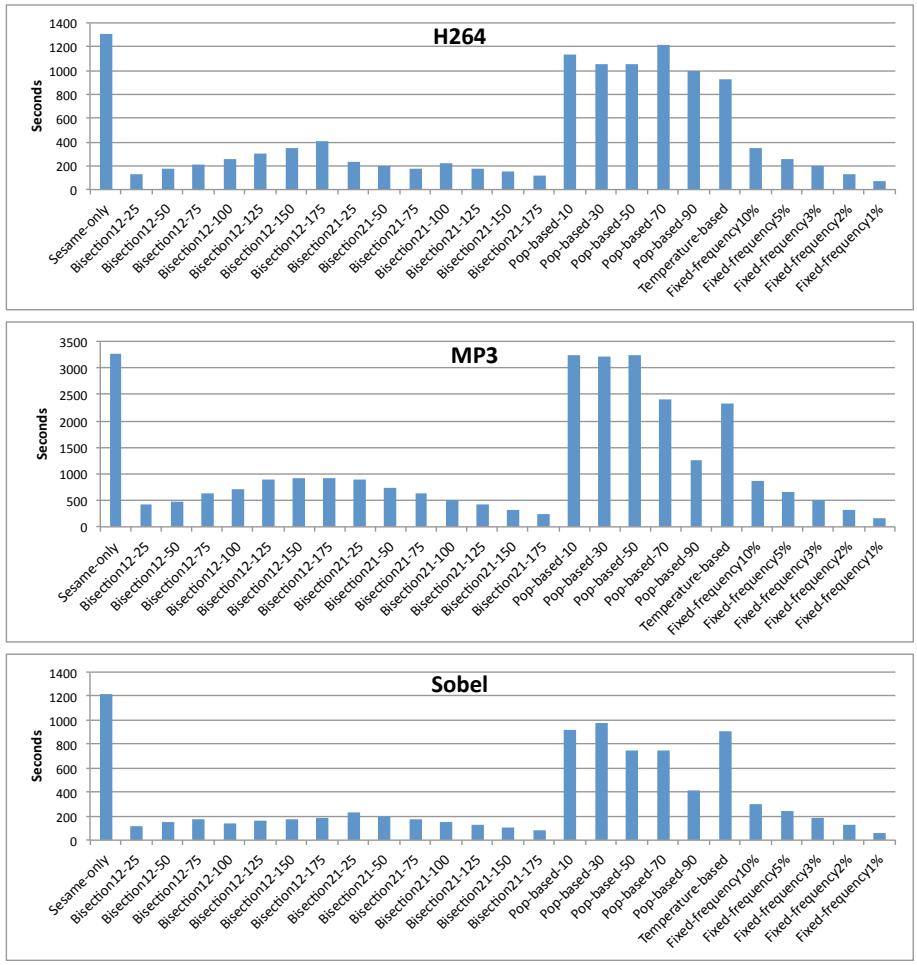


Figure 4.9: *Execution times for the different DSE methods.*

4.5 Experimental results

To evaluate the different interleaving methods, we have experimented with a DSE case study for a heterogeneous MPSoC platform consisting of up to 8 processors (interconnected by a crossbar) of possibly different types: MIPS, ARM or StrongARM. The DSE experiment is performed for three multimedia applications: an Mp3 decoder, a H264 decoder, and a Sobel filter for edge detection in images. The exploration considers two optimization objectives, namely performance (execution time) and system cost, and has been implemented using a NSGAII genetic algorithm performing 200 search iterations. Since genetic algorithms are stochastic processes, all results are averages from 10 execution runs.

To quantify the quality of the obtained Pareto fronts for the different interleaving methods, we consider two aspects: how close the found solutions are to a reference Pareto front and the spread of the solutions along the Pareto front. For this reason, we use the hypervolume (HV) and ∇ metrics. The HV metric [96] measures the hypervolume of the objective space covered by members of a Pareto optimal set and a reference point. It represents the size of the region dominated by the solutions in the Pareto optimal set. The reference point can simply be found by constructing a vector of worst objective values. The hypervolume metric is interesting because it is sensitive to the closeness of solutions to the true Pareto optimal set as well as the distribution of solutions across the objective space. The hypervolume value is calculated by summing the volume of hyperrectangles constructing the hypervolume. A Pareto optimal set with a large value for the hypervolume is desirable [89].

The normalized ∇ metric [33] measures the spread of solutions. It refers to the area of a rectangle formed by the two extreme solutions in the objective space, thus a bigger value spans a larger portion and, therefore, is better. The ∇ metric calculates the volume of a hyperbox formed by the extreme objective values observed in the Pareto optimal set:

$$\nabla = \prod_{m=1}^{M} (f_m^{max} - f_m^{min}) \tag{4.7}$$

Where M is the number of objectives, (f_m^{max} and f_m^{min}) the maximum and respectively minimum values of the m^{th} objective in the Pareto optimal set. A bigger value spans a larger portion and, therefore, is better.

For the HV and ∇ metrics, we use *relative values*. That is, we relate the HV and ∇ values for our hybrid DSE experiments against those from a reference Pareto front. The reference Pareto fronts – one for each application – were obtained by combining the Pareto optimal solutions from 10 runs of Sesame-based DSE. This implies that, e.g., a HV (∇) value of 1.0 means that the experiment in question yields the same HV (∇) value as the reference Pareto front.

In Figure 4.8, the average hypervolume (HV) and relative spread (∇) values (averaged over 10 runs) are shown for the different DSE methods applied to the Mp3 decoder, H264 decoder, and Sobel filter applications. The *Sesame-only* results (left-most bars) form the baseline for our hybrid DSE experiments. These results are averages for a single run (averaged over 10 separate runs) of Sesame-based DSE. So, no hybrid DSE and interleaving are performed in this case, and it solely compares a single simulation-only DSE run to the reference Pareto front. The remaining bars show the results for the various hybrid DSE approaches. The label *Bisection12-K* refers to the bisection-based interleaving method in which the DSE starts with simulation and switches to analytical estimation after K generations. Here, we have experimented with K values that equal to 25, 50, $\cdots$, 175 and $n_{gen} = 200$. Similarly, the label *Bisection21-K* refers to the same bisection-based interleaving but then starting with analytical estimations followed by simulations. The label *Pop-based-K* subsequently refers to the population based interleaving method with a certain K value. In our case, we have varied K from 10% up to 90% with steps of 20%. The fixed-frequency based interleaving method has been applied with K values of 1%, 2%, 3%, 5%, and 10%, as indicated by the labels in Figure 4.8.

A number of observations can be made from Figure 4.8. Looking at the hypervolume, hybrid DSE clearly shows promising results. Many of the hybrid DSE methods are capable of obtaining Pareto fronts with similar, or sometimes even better, HV values as Sesame-only DSE,. But, as will be shown later on, some of these hybrid methods do so at a fraction of the execution time. The spread of solutions (∇) in the obtained Pareto fronts is, however, highly dependent on the interleaving method as well as on the application under study. For example, the fixed-frequency approach with very low simulation frequencies clearly exhibit poorer ∇ values for all three applications. For two out of the three applications, this is also true for the population-based method and the bisection-based method where the DSE starts with simulations and ends with analytical estimations.

There is no clear winner among the hybrid DSE methods. Looking only at the HV values, the population-based approach yields the best results. But looking at the HV/∇ combination, the fixed-frequency interleaving with $K = 10\%$ seems to perform slightly better than the other methods. Overall, the fixed-frequency interleaving with $K \geq 3\%$ and the bisection-based approach where the DSE starts with analytical estimations and ends with simulations (i.e., *Bisection21*) appear to outperform the other hybrid DSE methods (based on the HV/∇ combination).

In Figure 4.9, the execution times (wallclock times) for the different DSE experiments are shown. As can be seen, a Sesame-only DSE experiment of 200 search iterations can take up to several thousands of seconds (like for Mp3). However, by interleaving simulations with analytical estimations several hybrid DSE techniques can significantly reduce the execution time of the DSE experiments. Only population-based and temperature-based interleaving fail to substantially improve the DSE execution times as these methods still use a high number of simulations. The fixed-frequency interleaving with $K = 10\%$ reduces the execution time of the DSE by a factor 4, while a Bisection21 interleaving with $K = 100$ yields a 6 to 8 times performance improvement. Both of these methods produce search results of similar quality as simulation-based DSE. We note that the time savings of hybrid DSE could have also been used for performing more

search iterations, thereby possibly improving the search results. We have not done so in this chapter (we considered the number of search iterations to be fixed), but this is considered as future work.

The timing results of the population-based interleaving method demonstrate that the proportion of cyclic mappings in the GA's population is high since many search iterations use simulation-based evaluation, even with $K = 70$. But since the population-based results are not significantly better in terms of quality (HV and ∇ values), we can further conclude that it is not necessary to avoid the use of analytical estimations every time there are (many) cyclic mappings in the population.

4.6 Related Work

Current state-of-the-art in system-level DSE often deploys population-based Monte Carlo-like optimization algorithms like hill climbing, simulated annealing, ant colony optimization, or genetic algorithms. By adjusting the parameters, or by modifying the algorithm to include domain-specific knowledge, these algorithms can be customized for different DSE problems to increase the effectivity of the search [74, 23]. Another promising approach is based on meta-model assisted optimizations, which combines simple and approximate models with more expensive simulation techniques [65, 77, 32, 13, 55]. In [32], the authors use meta-models as a pre-selection criterion to exclude the less promising configurations from the exploration. In [55], meta-models are used to identify the best set of experiments to be performed to improve the accuracy of the model itself. In [65], an iterative DSE methodology is proposed exploiting the statistical properties of the design space to infer, by means of a correlation-based analytic model, the design points to be analyzed with low-level simulations. The knowledge of a few design points is used to predict the expected improvement of unknown configurations. However, these meta-models usually have design space parameters relative to the micro-architecture of design instances, while they do not address the problem of e.g. topological mapping of an application on the underlying MPSoC architecture. While

micro-architecture parameters like cache size typically affect the system performance in a predictable, often linear, fashion, the resource binding of the application graph to the architectural platform presents a much less predictable performance.

A second class of design space pruning is based on hierarchical DSE (e.g., [49, 69, 54, 33]). In this approach, DSE is first performed using analytical or symbolic models to quickly find the interesting parts in the design space, after which simulation-based DSE is performed to more accurately search for the optimal design points. The main drawback of this method is that if the *first step* is not accurate enough, it may not produce the best set of design points to simulate. In our approach, the pruning and simulation phases are integrated to avoid this problem.

4.7 Towards more accurate pruning: a Future Outlook

In the previous sections, we showed that by properly interleaving analytical and simulative estimations, it is possible to reduce the computational time while still achieving solutions qualitatively comparable to the ones obtained with pure simulation-based DSE. In detail, we introduced an analytical model that estimates the throughput of the target multi-media application given a certain architectural configuration and application-to-architecture mapping. Our results showed that interleaving simulations with the throughput analysis is still necessary. In fact, the analytical estimations may in some cases be not accurate enough, because of estimation inaccuracies due to topological cycles in the dataflow graphs that are generated and used for throughput estimation during the analytical mapping exploration. As a consequence, to ensure that the DSE process is steered into the right direction, the analytical estimations still need to be interleaved with simulative evaluations.

In this section we introduce the basic idea of a possible future research direction towards the improvement of the throughput estimation. More in details, in this section, we introduce an approach for efficiently performing an analytical performance estimation based on Maximum Cycle Mean

(MCM) analysis. MCM analysis aims to correct the estimation errors due to the topological cycles generated during analytical mapping design space exploration. In early design space exploration, a key factor is keeping the throughput calculation fast and sufficiently accurate at the same time. To achieve this, we propose an *approximated MCM analysis* which improves the estimation proposed in the previous sections, achieving performance faster than regular MCM analysis [36].

4.7.1 Background

In this subsection, we provide a brief introduction to dataflow graphs (DFGs), Cyclo-Static Dataflow (CSDF) models, and PPN models. Moreover, we shortly introduce MCM analysis in the context of DFGs. This overview is essential for understanding the ideas presented in next Sections.

Dataflow graphs

Dataflow graphs (DFGs) are an extension of directed graphs. DFGs are widely used (especially) in the realm of multimedia, imaging, and signal processing to describe the flow of data between actors/nodes that transform the data from input streams to output streams. For example, in Homogeneous Synchronous Dataflow (HSDF) graphs, every node consumes/-produces a single unit of data (data token) from/to an edge. Therefore, HSDFs are also referred to single-rate graphs. In Synchronous Dataflow (SDF) [19], on the other hand, each node can consume/produce multiple tokens per edge. Formally, a dataflow graph is represented by a directed weighted graph $G(V, E, d, t)$ where V is the set of computation nodes, E is a set which defines directed edges (or precedence relation) from nodes in V to nodes in V, and $d(e)$ is the delay count (number of initial tokens) for edge $e \in E$. Each node $v \in V$ is associated with a positive integer $t(v)$ which represents the computation time for node v. In a HSDF, a single execution of all computation nodes $v \in V$ is called an *iteration*. The edge delays in a DFG are given in terms of iterations, i.e., an edge e from u to v with delay count $d(e)$ means that the computation of node v at itera-

tion i depends on the computation of node u at iteration $i - d(e)$. In this way, the delay count $d(e)$ on edge $u \rightarrow v$ represents the sequenced relation between computation nodes u and v. For a meaningful dataflow graph (e.g., deadlock-free HSDF), the total delay count of any cycle is non zero.

The order of visiting/executing the nodes in a DFG is called a *schedule*. A schedule length is the time to complete one iteration. A self-timed schedule represents an order in which a computation node is executed as soon as it has all input data available, i.e., as soon as possible. The (average) time needed to execute an iteration is called an iteration period. The main goal in executing DFGs is finding schedules with minimum iteration periods. It is known that the minimum iteration period can be obtained by a *self-timed schedule* [87]. A lower bound on the iteration period, called iteration bound [59, 44, 62], can be found by using the *computation delay ratio (r)* in the following way. The *computation delay ratio* of a cycle $C \in G$ is the ratio of the sum of the computation times of all the nodes in C to the total number of edge delays in C:

$$r(C) = \frac{\sum t(v)}{\sum d(e)} \tag{4.8}$$

where C is a cycle in G and $v, e \in C$. A *critical cycle* is the cycle which has the maximum $r(C)$ in a DFG. It represents the execution of the nodes that takes the largest amount of time. The computation delay ratio of the critical cycle determines the iteration bound of a DFG. That is, the iteration bound B(G) of DFG G is defined as:

$$B(G) = max\{r(C)\}, C \in G. \tag{4.9}$$

The term $r(C)$ is also called cycle mean of cycle C. Similarly, Equation 4.9 is called Maximum Cycle Mean (MCM). A very important property of a DFG is that the iteration period of any schedule cannot be smaller than the iteration bound (i.e., MCM) of the DFG.

Cyclo-Static Dataflow graphs

The Cyclo-Static Dataflow (CSDF) model of computation [20] is an extension of the SDF model that allows a compact representation of applications with cyclically changing, but predefined behavior. In a CSDF, every node j has a *function repertoire*, which is a sequence of functions $f_j(0), f_j(1), \cdots, f_j(S_{j-1})$ of length S_j. The nodes execute a function from their repertoire in the following way: the n^{th} time a node v_j executes, it selects function $f_j(n \pmod{S})_j$. Therefore, a node v_j has S_j execution phases. Consequently, every node $v \in V$ is associated with a sequence $T(j)$ of positive integers $[t_j(0), t_j(1), \cdots, t_j(S_{j-1})]$ which represents the computation time of every function from the function repertoire of node v_j , i.e., the computation time of the node in every execution phase. Like SDF graphs [60], the structure of a CSDF graph can be compactly represented by a topology matrix Γ. The entries of Γ for a node j of a CSDF graph represent production and consumption rates for a complete execution sequence of length S_j. The columns of Γ represent the nodes and the rows of Γ represent the edges. A positive entry $\Gamma(i,j)$ means that node j produces $\Gamma(i,j)$ tokens on edge i accumulated by all phases. A negative entry $\Gamma(i,j)$ means that node j consumes $-\Gamma(i,j)$ tokens from edge i. Given a connected CSDF graph G, a valid static schedule for G is a schedule that can be repeated infinitely on the incoming sample stream and where the amount of data in the buffers remains bounded. A vector $\vec{q} = [q_1, q_2, \cdots, q_N]^T$, where $q_j > 0$, is a *repetition vector* of G if each q_j represents the number of invocations of an actor v_j in a valid static schedule for G. The *repetition vector* of a CSDF graph, is given by

$$\vec{q} = S \cdot \vec{r}, \text{ where } S(i,j) = \begin{cases} S_j & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{4.10}$$

where $\vec{r} = [r_1, r_2, \cdots, r_N]^T$ is a positive integer solution of the balance equation $\Gamma \cdot \vec{r} = \vec{0}$. In Figure 4.11, an example CSDF (Figure 4.11a) and matrices Γ and S (Figure 4.11b) are shown.

Polyhedral Process Networks

A *Polyhedral Process Network* (PPN) [93] is a directed graph $G = (P, E)$ where P is a set of vertices representing processes and E is a set of edges representing communication channels. In terms of behaviour, a PPN is a special case of the KPN model [52]. That is, a process in a PPN first reads data from FIFO channels, then executes a function, and writes results to FIFO channels. Here, processes are synchronised based on the KPN semantics, which implies that any process is blocked when attempting to read an empty FIFO. However, in contrast to KPNs, the PPN model assumes finite FIFO buffers. Therefore, processes also block when attempting to write to a full FIFO. The execution of a process is specified by its *(iteration) domain* which is described by for-loops. This set of iterations is represented using the polytope model [34] and is called process domain, denoted by DM_P. Accessing input/output ports of the PPN process is represented as a subset of the process domain, called input/output port domain. Compared to PPN processes, accessing input/output ports of CSDF actors is described using repetitive production/consumption rates sequences. Another key difference is that synchronisation in PPN is implemented using blocking reads/writes, while in CSDF it is implemented explicitly using a schedule. It has been shown [30, 17] that a PPN can be translated into a CSDF graph in which the production/consumption rates sequences consist only of 0s and 1s. A 0 in the sequence indicates that a token is not produced/consumed, while a 1 indicates that a token

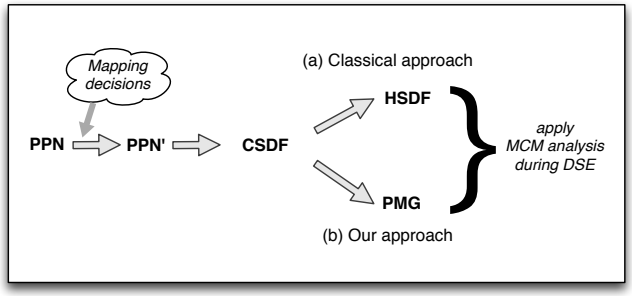


Figure 4.10: *MCM analysis on a PPN.*

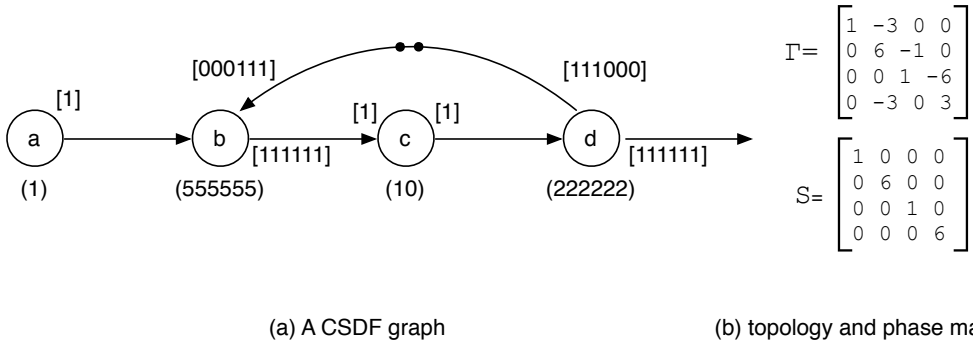


Figure 4.11: *The CSDF graph and its matrices* Γ *and* S*.*

is produced/consumed.

4.7.2 Analytical performance evaluation of a CSDF

The data flow applications considered in the Daedalus framework can be naturally modelled as PPN, therefore, we can restrict the performance analysis to PPNs rather then KPNs. To reach the goal of performing MCM analysis on PPNs, one could first translate the PPN into a dataflow network (specifically into a CSDF [30, 17]). Subsequently, the MCM analysis of a CSDF graph could then be obtained in the traditional fashion by means of conversion to an HSDF graph. In particular, in CSDF graphs, the MCM analysis is applied on the corresponding equivalent HSDF graphs. This approach is illustrated in Figure 4.10(a). The approach has a major drawback: working on HSDF graphs exponentially increases the complexity of computing the MCM value. This significantly limits the practical applicability of the MCM analysis. Recently researchers focused on devising alternative techniques to reduce the complexity of computing MCM and converting SDF to HSDF graphs [38, 37, 27]. The required computational power and the limited scalability of these approaches, however, make them unsuitable for our purpose.

A better approach would require to perform MCM analysis without passing trough an HSDF graph while embedding the information relative

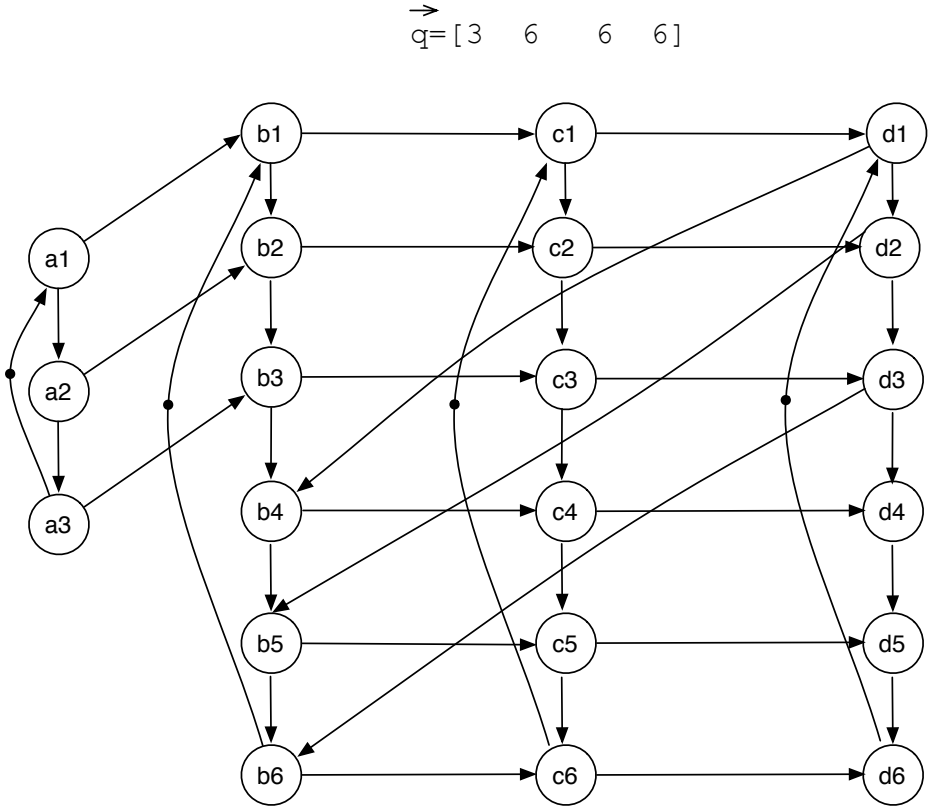


Figure 4.12: *The equivalent HSDF graph.*

to the transformed HSDF graph in a simpler graph. In order to embed the required information, we need a deep understanding of the steps required to transform a CSDF into HSDF graph. In next section, we propose a motivation example that illustrates the increased complexity of the generated HSDF graph, and we construct a Performance Modelling Graph starting from the steps of the HSDF generation algorithm.

4.7.3 Building the equivalent PMG graph

The equivalent HSDF graph of a CSDF graph can be constructed starting from the phase repetition vector q. The algorithm for the conversion [21], has the following procedure:

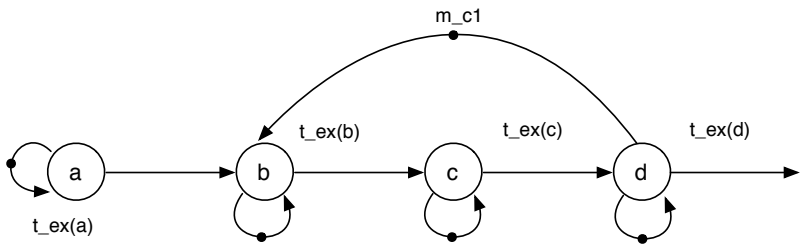


Figure 4.13: *The proposed performance modelling graph.*

- Instantiate an HSDF graph actor for each phase represented in the repetition vector q.
- Instantiate sequence-edges to model the in-order execution.
- Add edges to model communication.

For further details of the conversion algorithm, we refer the reader to work of Bilsen [21]. The equivalent HSDF graph for the example in Figure 4.11 is illustrated in Figure 4.12.

To perform MCM analysis on a simpler graph, we use the idea of representing the execution times of the nodes $t(v)$ and edge delays $d(e)$ of the CSDF graph in a way that they mimic a throughput-approximate HSDF graph. By doing so, we can apply the MCM analysis on this so-called Performance Modeling Graph (PMG), as illustrated in Figure 4.10(b).

By definition, the throughput of a strongly-connected HSDF graph is the inverse of its MCM value. Recall that the MCM of a strongly-connected HSDF graph is equal to the iteration period of the graph, which is also equal to the time to complete one iteration.

One iteration is defined to be the execution of every node once. The difference with the CSDF graphs is that an iteration of an CSDF graph is defined by the execution of its nodes according to the repetition vector q.

That is, in one iteration node v executes q_v times spending time t_{ex} as defined below:

$$t_{ex}(v) = r_v \cdot T(v)$$

where r_v is the corresponding element of vector r ($\Gamma \cdot \vec{r} = \vec{0}$). In one iteration, node v executes r_v phases of size S_j, consequently, the execution time of a node per phase is given by:

$$T(v) = \sum_{k=0}^{S_j - 1} t(k)$$

If we substitute the terms $T(v)$ used in the CSDF graph with the $t_{ex}(v)$ just described, we will obtain a graph that executes its nodes 'once', resembling an HSDF execution.

Importantly, the new execution times are the same as if the graph executes the nodes according to its repetition vector. As result, the iteration periods of the two graphs are the same.

To illustrate the above approach, consider the example CSDF graph shown in Figure 4.11. The resulting PMG graph is shown in Figure 4.13.

Since the used HSDF generation algorithm [21] always introduces back edges connecting the instantiated HSDF actors of each node of the CSDF, to model this communication, we add *back edges* to every node in the PMG. Similarly, since there a back edge forming a cycle in the original CSDF, several back edges are generated in the equivalent HSDF connecting the instantiated actors in the HSDF according to the algorithm rules [21].

For each cycle present in the CSDF graph, we add a *delay* m in the equivalent PMG given by:

$$m = \frac{\max\{t_{n_i}\}_{Cycles_{HSDF}}}{\sum_{v \in Cycle_{CSDF}} t_{ex}(v)}$$

where t_{n_i} are the execution times of the instantiated actors that form a cycle c_i in the HSDF originated by the back-edge that forms a cycle in the original CSDF graph.

For example, considering the example in Figure 4.11 and its corresponding HSDF graph in figure 4.12, we can observe that the cycle formed by the nodes b, c and d generates different cycles in the corresponding data flow graph.

Adding a delay m_{c_1} that includes the information relative to those cycles inside the PMG graph, would allow to analyse only the cycle representing the worst case of all HSDF cycles generated from that back edge. Since the edges that form the above cycles are generated with the algorithm for the conversion to CSDF to HSDF [21], as a future work, we propose to extend the algorithm to directly generate a proper delay m_{c_1} inside the PMG graph.

4.8 Conclusions

In this chapter we presented a technique to reduce the simulation overhead in system-level design space exploration (DSE). To this end, we have presented an iterative design space pruning methodology based on static throughput analysis of different application mappings. By interleaving these analytical throughput estimations with simulations, our hybrid approach can significantly reduce the number of simulations that are needed during the process of DSE. Moreover, we have proposed and examined different strategies for interleaving fast but less accurate analytical performance estimations with slower but more accurate simulations. Experimental results have demonstrated that such hybrid DSE is a promising technique that can yield solutions of similar quality as compared to simulation-based DSE but only at a fraction of the execution time. Finally, we introduced the basic idea of a possible future research direction towards the improvement of the throughput estimation. More in details, we introduced an approach for efficiently performing an analytical performance estimation based on Maximum Cycle Mean (MCM) analysis, in order to to correct the estimation errors due to the topological cycles generated during analytical mapping design space exploration.

Chapter 5

Design Space Pruning for Efficient Slack Allocation and Lifetime Estimation (for NoC-based MPSoCs)[1]

5.1 Introduction

An important metric in modern embedded systems is the expected lifetime: smaller feature sizes, higher operating frequencies, and thermal issues are increasing the failure rate of integrated circuits to the point where device lifetimes are becoming shorter than market expectations. Redundant hardware is typically employed to improve system lifetime. For instance, *slack allocation*, which overdesigns the system by provisioning execution and storage resources beyond those required to operate failure-free, has been proposed as a low-cost alternative to replicating resources [22, 67]. When components fail, data and tasks are re-mapped and re-scheduled on resources with slack; as long as performance constraints are satisfied, the

[1]The contents of this chapter have been based on [7]

system is considered to be operational despite component failure. For any given system, the design space of possible slack allocations is large and complex, consisting of every possible way to replace each component in the initial system with another component from a library.

In this chapter we propose an exploration framework for Network-on-Chip (NoC) based MPSoCs that substantially reduces the computational cost of slack allocation. We make two principal contributions. First, we develop *failure scenario memoization* to reduce the computational cost of lifetime estimation by storing and reusing estimated lifetime values for systems with one or more failed components. The lifetime of all partially failed systems is derived and saved (the memory storage cost of such values is negligible); when a previously explored partially-failed system is encountered a second time, its expected lifetime is read from a database rather than re-estimated. It is worth noting that the larger the design space, the greater the resulting opportunity for reusing lifetime estimation and speeding up the exploration.

Second, we introduce a correlation-based architecture distance metric to identify symmetries for clusters of components called *islands*. In modern platform- and network-on-chip based design, components are clustered around switches in the on-chip network. When clusters and the tasks mapped to them are considered to be symmetric, some configurations have the same effect on the overall system lifetime. This can be leveraged to reduce the number of evaluations.

The rest of the chapter is structured as follows: Section 5.2 presents a summary of recent advances in the DSE and lifetime estimation domains. Section 5.3 presents the key concepts of slack allocation for lifetime improvement. Section 5.4 introduces our proposed methodology to accelerate DSE for lifetime estimation. The application of our methodology is described in Section 5.5. Finally, Section 5.6 draws some concluding remarks.

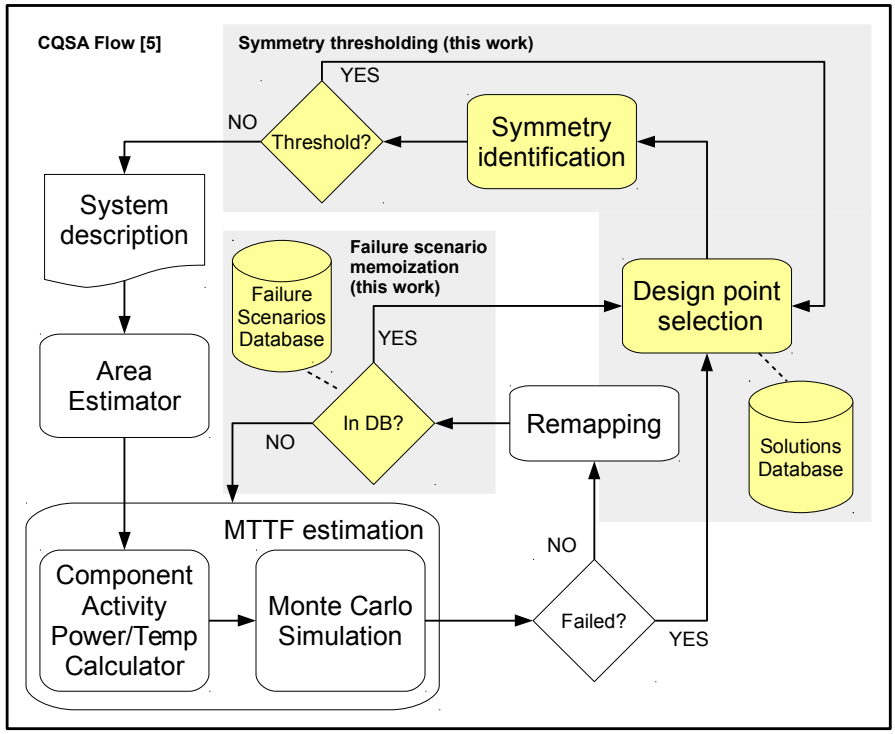


Figure 5.1: *Overall lifetime optimization flow. The grayed areas are the contributions of this work*

5.2 Related Work

5.2.1 System Lifetime Estimation

System lifetime is typically modelled by estimating the system mean time to failure (MTTF), given assumptions about the failure rates of individual components and their relationship to (or dependence on) one another. Historically, the exponential failure distributions have been used because of the ease with which component failure distributions can be combined analytically to determine system failure distributions.

For example, without redundancy, the MTTF of a system consisting of i components is $MTTF_{sys} = (\sum_i \lambda_i)^{-1}$, where λ_i is the failure rate of

the ith component. The exponential distribution, however, because of its assumption of constant failure rate, has been shown to be inaccurate: semiconductor systems tend to have low early lifetime failure rates that grow as the system ages [85]. A number of researchers have alternatively proposed the use of lognormal failure distributions [22, 85]. However, because lognormal distributions cannot be combined analytically to determine system failure distributions, computationally costly Monte Carlo Simulation (MCS) campaigns are often used to estimate $MTTF_{sys}$ [67, 85]; evaluating a single design with MCS may require hundreds if not thousands of trials, which quickly becomes intractable as the design space grows. In this chapter, we will develop a technique for reusing MCS trials across different simulated samples for either the same configuration and different configurations, significantly reducing the computational cost of lifetime estimation.

5.2.2 Design Space Exploration

Current state-of-the-art in system-level DSE often employs population-based, metaheuristic optimization algorithms like hill climbing, simulated annealing, ant colony optimization, or genetic algorithms. By adjusting their parameters, or by including domain-specific knowledge, these algorithms can be customized for different DSE problems, thereby increasing their effectiveness [74, 23, 91].

In [91], the authors exploit knowledge of the platform characteristics to optimise the DSE search process using two extended genetic operators that exploit the system symmetry. This approach only works on fully homogeneous systems (i.e. it is not suitable for heterogeneous systems).

Another promising approach is based on meta-model assisted optimisation, which combines simple and approximate models with more detailed and costly simulations [65, 77, 32, 55]. These methods assume that the meta-models are sufficiently detailed to capture the shape of the design space. Due to the lognormal distributions involved in the estimation of the device lifetimes [85] (i.e., lifetime depends on accumulated wear), this kind of model cannot easily capture the design space.

In [32], the authors prune the design space, eliminating less likely

configurations using meta-models, while in [55], meta-models are used to identify the best set of experiments to be performed to improve the accuracy of the model. As opposed to this work, these methods do not consider and characterise symmetry in the architecture.

5.3 Lifetime optimisation background

When a component fails in an NoC-based MPSoC, the system can remain operational, if sufficient excess resources have been allocated to perform the tasks of the disabled components. These extra resources are called *slack*, i.e., components not needed to satisfy the performance constraints in the original, fully functioning system [22, 67]. The goal of *slack allocation* in the context of lifetime optimisation is to find distributions of slack that extend system lifetime by making a subset of possible component failure sequences survivable.

In this work, we define two forms of slack [67]: a processor's *execution slack* as the total number of unused processor cycles that are available to execute additional tasks in the event that another processor fails or becomes inaccessible; the *storage slack* for memory as the total unused address space that is available to store additional data in the event that a memory component fails or becomes inaccessible.

The design space for slack allocation is exponential in the number of system components and consists of every possible way to replace each component in the initial system with an alternative component from a library. Furthermore, evaluating the lifetime of any individual system is computationally expensive: the use of lognormal failure time distributions requires the use of Monte Carlo Simulation. Therefore, exhaustive search is intractable, and heuristic algorithms are necessary.

5.3.1 Exploring Slack Allocations

Slack allocation is explored starting from a valid system configuration (i.e., component selection and task mapping that fall within the application's performance constraints) and replacing components with over-provisioned

versions, thereby providing additional computation or storage resources. Each new configuration is analysed to evaluate its estimated lifetime and its area, as shown in Figure 5.1. In general, lifetime is expected to increase with slack, as additional slack implies additional opportunities to survive failure. However, total power and power density often increase as well, putting downward pressure in MTTF. For some configurations, allocated slack is not useful, resulting in MTTF degradation. Configurations that are better than all others for either area or lifetime are kept as best solutions.

While heuristic exploration methods are often proposed in the literature, the scope of their applicability is often limited to narrowly defined domains. Metaheuristic search methods like genetic algorithms (GAs), on the other hand, are designed to operate efficiently with relatively little design domain knowledge, but at the price of a greater number of design points to be explored. This can make even metaheuristic exploration intractable for design spaces where the evaluation of a single design point is costly. However, modern designs are often designed around platforms with a limited number of different components that are interconnected in a consistent fashion. Consequently, while two different configurations may allocate slack to different components, the slack allocations may be similar enough to consider them to be equivalent, reducing the need for costly evaluations.

5.3.2 The CQSA framework

The design challenge of slack allocation is efficiently finding the most cost-effective allocations of execution and storage slack. As mentioned before, the slack necessary for a system to survive component failure is quantized. For example, for a system to survive processor failure, enough slack must be allocated so all of its tasks can be re-mapped. Allocating less slack than is required to re-map each of the processors tasks serves no purpose.

Furthermore, system lifetime and yield are likely degraded: lifetime by the increase in system temperature that follows the increase in power consumption of the upgraded processor, and yield by the increase in component area and the resulting increase in vulnerability to defect. We,

therefore, define the critical quantity $[es, ss]$ of a component as the total slack, es MIPS of execution slack and ss KB of storage slack, required to re-schedule and re-map the tasks and data that would be orphaned if that component were to fail.

The Critical Quantity Slack Allocation (CQSA) [67] jointly optimizes system lifetime and cost by determining (a) how much slack should be allocated in the system, and (b) where in the system it should be allocated, such that the system mean-time-to-failure (MTTF) is increased in the most area-efficient way as possible. This framework provides us two functionalities: on one side, it provides an estimation tool to determine system lifetime of a design point; on the other, it allows for DSE of critical quantity slack allocation through a greedy procedure.

System MTTF is a function of:

- the target application and given communication architecture;
- component utilization, which is a function of the tasks mapped to a given component;
- component power, which is derived from component utilization and other parameters; and,
- component temperature, which is derived from system-level temperature modeling.

System cost (area) is determined using system-level floorplanning. The result of CQSA is a set of MTTF-area Pareto-optimal designs with variable trade-offs from which the designer may select the design point(s) most appropriate for the given target application. To achieve this goal, CQSA performs a series of design space explorations, allocating slack and evaluating the resulting cost and lifetime of the system. CQSA allocates slack by replacing low capacity processors and memories with higher capacity slack or memories, creating opportunity for failures to be survivable by enabling tasks to be re-mapped and traffic re-routed in ways that potentially still satisfy performance constraints. Design space exploration focuses on those

quantities of slack expected to result in cost-effective lifetime improvement, those defined by critical quantities of slack, in particular critical quantities for network switches. A detailed analysis of the lifetime improvement that is possible when focusing exploration on switch critical quantities can be found in [67].

Using the m unique critical quantities defined by the n system switches as starting points, CQSA performs a series of exhaustive and greedy execution and storage slack allocations.

By focusing exploration in this way, CQSA efficiently exposes those slack allocations that cost-effectively maximize the number of survivable combinations of processor and switch failures, while pruning away the overwhelming majority of the design space. The CQSA algorithm is composed of three stages that consider different sets of critical quantities.

Stage 0 starts with the baseline architecture and incrementally allocates execution slack to find the best execution slack allocations not covered by a switch critical quantity. If a system has no switch critical quantities (e.g., if no switch failure is survivable under any circumstances), only Stage 0 is performed. Search based on a switch critical quantity proceeds in two steps: first, an exhaustive search is conducted for the allocation of the critical quantity of slack that maximizes system MTTF; second, a greedy search proceeds which incrementally allocates slack. In Stage 1, this greedy search allocates only execution slack. In Stage 2, this greedy search allocates both execution and storage slack.

A practical example is shown in Appendix 7.

5.3.3 System Lifetime Evaluation

System lifetime evaluation represents the inner loop of any lifetime optimisation approach. The process of using Monte Carlo Simulation (MCS) for lifetime estimation is illustrated in Figure 5.1. Given a description of a system (its processors, memories, switches, and interconnection) and a task graph representing the application it executes (its computational tasks, storage tasks, and communication), the initial component utilisation, power consumption, and temperature (based on the derivation of a system-level floorplan) are calculated.

Our approach estimates the MTTF using Monte Carlo Simulation (MCS) [67] to repeatedly generate sample systems which experience randomised sequences of lognormally-distributed component failures and determine at what time, on average, enough components have failed such that the system is no longer able to satisfy its performance constraints.

This process can be essentially divided into two parts: using component utilisation, power and temperature to determine component failure distributions and identify which component fails next, and determining if that failure results in system failure. Component wearout failure distributions are primarily dependent on component temperature. We use a combination of scheduling, component-level power modelling, floorplanning and system-level temperature modelling to derive the component temperatures that are subsequently used to determine component failure distributions.

Using these distributions, we determine which component in a sample system is the next to fail. If, after a failure, the sample system can be re-scheduled, component wear is calculated and component failure distributions are re-calculated to determine the location of the next failure. Otherwise, the sample system has failed, system failure statistics are updated, and our MCS approach proceeds to the next sample system. This process continues until the estimated MTTF converges.

In order to determine the MTTF and Time to First Failure (TTFF) of a system, we must model the changes a system goes through during their lifetime. System lifetime is modelled by simulating sequences of component

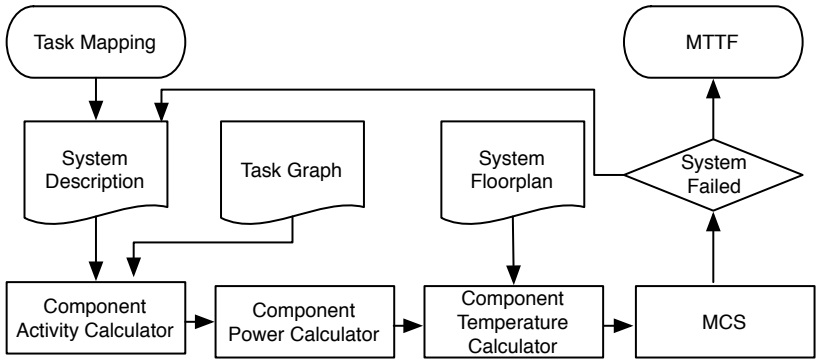


Figure 5.2: *Sytem lifetime evaluation*

failures and determining how long performance constraints are satisfiable.

We must make several assumptions about how components in the system fail and when these failures cause the overall system to cease functioning in order to better define our problem. Individual processors and switches within a system may fail over the course of its lifetime. We assume that memories are not susceptible to permanent failure, as they can be covered by simple row and column redundancy, but they can become inaccessible when the switches to which they are attached fail. That said, our approach could be easily adapted to account for situations in which memories are susceptible to permanent failure by treating them in the same manner as processors during task mapping.

We also assume that systems can automatically detect component failure, at which point the operating system signals our task mapping algorithm to execute. In addition to this method of computing task mappings reactively (i.e., only when necessary), our task mapping algorithm can also be triggered at pre-defined time intervals in an effort to proactively address system lifetime. The task mapping algorithm is responsible for remapping tasks and data from failed resources to those with slack, re-routing the affected traffic, and improving the lifetime of the system through decisions made during this process. As long as a valid task mapping exists, the system can satisfy its performance constraints and continue to function [43]. The task mapping process used is described in [43] and it

is beyond the scope of this chapter.

To accurately estimate the distribution of permanent component failures due to wear-out, we adopted a lognormal failure distribution model for each of three temperature-dependent failure mechanisms [86]: electromigration, time-dependent dielectric breakdown, and thermal cycling.

The probability density function for the lognormal distribution is given by

$$f(x) = \frac{1}{x\sqrt{2\sigma\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}$$

where μ and σ are shape and scale parameters respectively. Generally we use $\sigma = 0.5$ to model the distribution of wear-out failure mechanisms. We can define component MTTF due to a particular failure mechanism (FM) as:

$$MTTF_{FM} = e^{\mu + \frac{\sigma^2}{2}} \tag{5.1}$$

When the MTTF of a component due to a particular failure mechanism is known, Equation 5.1 can be solved for μ, thereby fully specifying the component failure distribution for that particular failure mechanism. In particular, given the three failure mechanisms considered in [67], we have:

Component MTTF due to electromigation

$$MTTF_{EM} = K_{EM} J^{-n} e^{\frac{E}{kT}} \tag{5.2}$$

where K_{EM} is a scaling factor, J is the current density of the component, E is the activation energy for electromigration, k is Boltzmann's constant, T is the temperature of the component in Kelvin, and n is a constant based on the properties of the metal used in the process.

Component MTTF due to time-dependent dielectric breakdown

$$MTTF_{TDB} = K_{TDB} \frac{1}{V_{DD}^{a-bT}} e^{\frac{X + Y/T + ZT}{kT}} \tag{5.3}$$

where K_{TDB} is a scaling factor, V_{DD} is the drain voltage, T is the component's temperature in Kelvin, k is Boltzmanns constant, and a, b, X, Y, Z are

fitting parameters [67].

Component MTTF due to thermal cycling

$$MTTF_{TC} = K_{TC}(\frac{1}{T - T_a})^c \tag{5.4}$$

where K_{TC} is a scaling factor, T is the component's temperature in Kelvin, T_a is the temperature of the surrounding environment, and c is the Coffin-Manson exponent [67].

Each failure mechanism is normalized with scaling factor K so that its MTTF is 30 years for a characterization temperature of 345 K [40]. This normalization equalizes the effect of the three failure mechanisms on a given component.

Figure 5.2 (a more detailed view of the "MTTF estimation" box from Figure 5.1) gives an overview of our lifetime evaluation process for a single Monte Carlo sample. We use an initial task mapping that minimizes power dissipation regardless of the task mapping heuristic that will be used later. We cannot use temperature or wear information to determine the initial task mapping since components have neither a temperature nor an amount of wear associated with them yet. The utilization of each component is first calculated based on that task mapping and given information about the system and the application (Component Activity Calculator block in Figure 5.2). Given component activity, component power dissipation can be derived using data sheet values for processors [9], CACTI [8] for memories, and ORION [53] for switches (Component Power Calculator block in Figure 5.2). Using the floorplan determined using Parquet [12] and per-component power dissipation data, steady-state temperatures for each component are calculated using Hotspot [83] (Component Temperature Calculator block in Figure 5.2). Our temperature modelling assumptions (range, average value, etc.) are designed to match previously published temperature modelling assumptions for the same types of systems [25]. These component temperatures are then used to shape the failure distribution for each failure mechanism.

For each Monte Carlo sample, failure times are randomly selected from the initial failure distributions of each failure mechanism of each component (processors and switches). The failure distribution for each failure mechanism of a component is based on that component's *temperature* which is derived from the current task mapping.

Once the failure distributions are computed, we determine which component in the system has the earliest failure time based on the amount of wear that has been accumulated so far (FM Update/Component Failure block in Figure 5.2). In addition to their place in the simulation, we use the time to failure and accumulated wear values that are calculated in this step as *outputs* from the wear sensors we assume to exist on chip, and these values are used as input to our wear-based task mapping heuristic. We mark this component as failed and proceed to the task mapping process. If we are able to find a valid task mapping, the *system operational* path is taken, and the simulation loop begins another iteration. If a valid task mapping does not exist, the *system failed* path is taken, and we record the current simulation time as the failure time for the sample system.

It is worth noting that the computational cost of MCS can be significant: floorplanning is required to support thermal modelling, and a thermal simulation is required each time a failure is survived. In our experiments we use $10,000$ Monte Carlo samples per design point, which requires up to 2 minutes of simulation for a 26-component system. Consequently, MCS encounters many similar partially failed system configurations that, at most, differ only in the precise sequence in which components have failed and exact times those failures occurred.

5.4 Proposed Design Space Pruning

To reduce the computational cost of design space exploration in the context of lifetime optimisation, we propose two pruning approaches: the first reduces the number of samples evaluated during MTTF estimation by storing and reusing partial results; the second reduces the number of configurations to be explored during DSE by exploiting symmetries in the

design under analysis.

5.4.1 Memoization of Lifetime Estimation

We propose to improve the efficiency of slack allocation by using the *memoization* of system lifetime estimates of intermediate states during failure simulation, thereby accelerating the evaluation of multiple designs in the design space. Memoization saves partial results so that if the same calculation occurs later, the result is available.

The computational cost of MCS-based lifetime estimation is in its repetition: hundreds if not thousands of trials are required for accurate estimation. However, much of this work is redundant: the hottest components tend to fail first, followed by the next hottest, etc., resulting in MCS trials that are qualitatively the same except for small differences in the ordering and timing of component failures. We hypothesise that lifetime estimation will be significantly accelerated if this redundant simulation is replaced by table lookups.

To make memoization most effective, we propose to perform an initial lifetime estimation using exponential, as opposed to lognormal, failure distributions. **This approximation is a key contribution of our methodology.** Since the failure rate of under an exponential distribution is constant, the estimated system lifetime is not dependent on accumulated wear.

Definition 5.4.1. *We define a scenario* s *as a set of working components in an NoC-based MPSoC with a specific task and storage mapping.*

Consider a scenario s reached by the sample systems i and j after two component failures. MCS sample generation implies that the absolute failure times of components in each sample are unlikely to be the same. Despite this, it is entirely possible that each system has experienced the same two failures, if at different times and in different orders, thereby reaching the same scenario. Under a lognormal model, the time-to-failure for system i when reaching scenario s tells us little about the time-to-failure for system j: lognormal time-to-failure is dependent on wear. Under an

exponential model, the probability density distribution function of a failure is:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{5.5}$$

where λ is the rate parameter of the distribution. The component MTTF due to a particular failure mechanism is equal to the mean of the exponential distributed variable

$$MTTF_{FM} = E[f(x)] = \frac{1}{\lambda} \tag{5.6}$$

Under an exponential model, the time-to-failure for scenario s for *any system* is the same, *regardless of the timing and ordering of previous failures*, since an exponential distributed variable X obeys to the *memoryless* property:

$$P(X > s + t | X > s) = P(X > t), \quad \forall s, t \geq 0 \tag{5.7}$$

This relationship implies that if the waiting time X is conditioned on a failure to observe the event over some initial period of time s, the distribution of the remaining waiting time is the same as the original unconditional distribution. For example, if a failure has not occurred after s seconds, the conditional probability that occurrence will take at least t more seconds is equal to the unconditioned probability of observing the event more than t seconds relative to the initial time.

Using this exponential failure model, having observed the time-to-failure for scenario s once, recalculating it would be redundant.

As we just observed, the exponential model has a fixed failure rate and it is much more simple. However, using the exponential distribution leads to inaccurate MTTF estimation. As is shown in Figure 5.3, the MTTF of designs with low lifetime is underestimated, while the MTTF of designs with high lifetime is overestimated. Nevertheless, this inaccuracy is not an issue in the context of DSE for electronic systems: the MTTF **ranking** of any two design points is generally respected, with an acceptable error of a few months over a multi-year lifetime (when projected over the corresponding lognormal distribution).

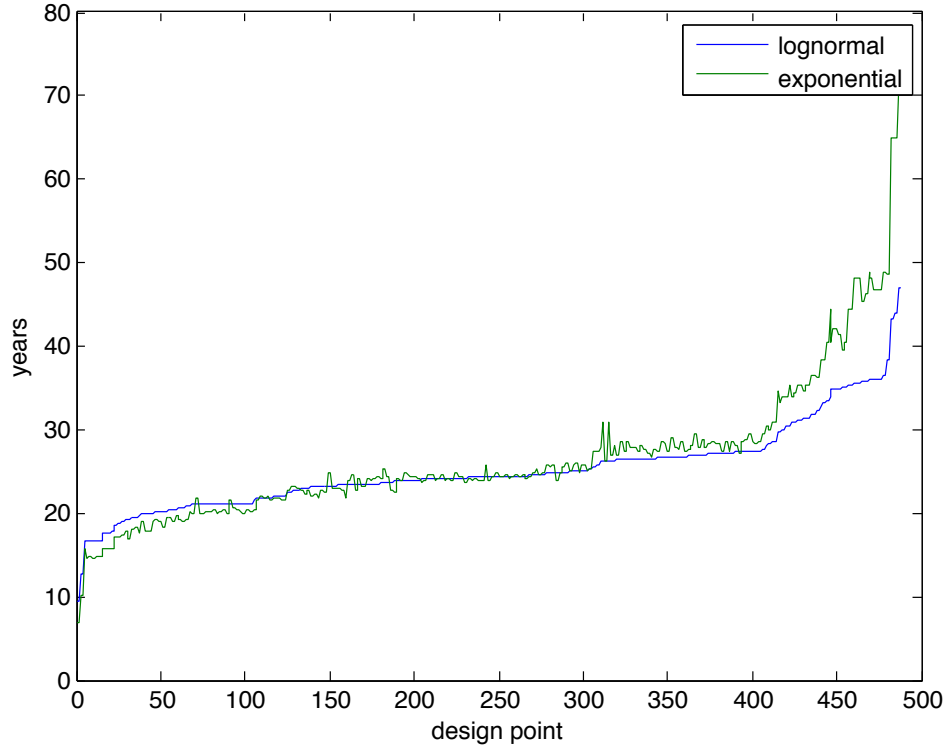


Figure 5.3: *System-lifetime evaluation with lognormal distribution (blue) and exponential distribution (green) for the MWD benchmark on a Mesh-NoC*

In fact, we observe that the maximum error in ranking for the exponential distribution is ± 3 years, which, when projected on a lognormal estimator, is reduced to ± 0.5 years. This implies that using the exponential distribution instead of the lognormal distribution will not affect the search for the Pareto-optimal solutions for the slack allocation problem while performing DSE. Once the Pareto front is estimated, the actual MTTF of each design point on the front can be recalculated using the lognormal distribution, effectively resolving any discrepancy in accuracy.

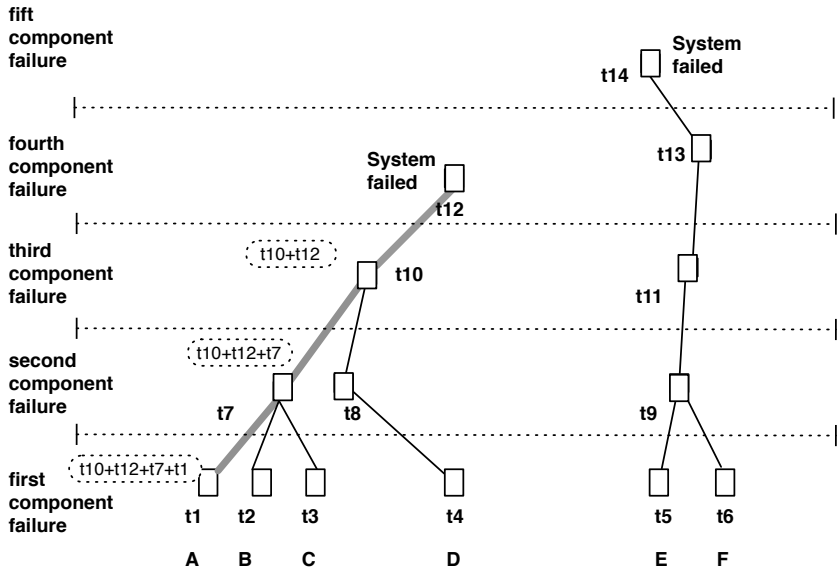


Figure 5.4: *Memoization of failure scenarios during DSE with system-lifetime estimation.*

5.4.2 Lifetime Estimation Approach

Within our lifetime estimation framework, the failure times of partial systems are memoized during Monte Carlo Simulation so that they can be reused during subsequent MCS trials. For each sample system i, when time is advanced to the j^{th} component failure, we record the relative component failure time t_j between the $j-1$th and jth failure. The failure time F_i for system i occurs when the K^{th} failure occurs and the system is unable to meet performance constraints: $F_i = \sum_{j=1}^{K} t_j$. Once the failure time of sample system i is determined, we can work backward to determine the failure time of all preceding functional systems, and store these values for future use. To explore the slack allocation design space, we adopted the popular genetic algorithm NSGA-II [29] (Non-dominated Sorting Genetic Algorithm-II). It is worth noting that the proposed pruning techniques do not depend on a particular search algorithm, and that they can be applied to any metaheuristic.

Figure 5.4 shows a simple example of slack allocation DSE where we

have six architectures to explore, A, B, C, D, E, F. Let us assume that design A is explored first: determining its MTTF requires that four scenarios be evaluated, as shown in Figure 5.4.

Let us assume now that architecture B converges to the same working scenario at the second component failure. We, therefore, have that B can reuse the same scenario prediction as A from the second component failure. Anytime a system configuration has been previously evaluated (i.e. it is known), we can immediately determine its associated failure time, and use it to calculate the overall system lifetime of a sample system, independently of the sequence of failures that lead the system to the known state.

Figure 5.5 shows the number of intermediate scenarios that are explored with and without the proposed memoization technique over the first 200 design points of a DSE run. As the order of the exploration affects memoization performance, we used both random (as done by NSGA-II) and sequential design point selection (as done by exhaustive search). Sequential selection favors memoization because it explores similar configurations first, promoting the generation of a large failure scenario database. Figure 5.5 includes the worst (random) and best (sequential) cases for memoization. Overall, we have a reduction between 15% and 30% in the computational effort, with the perspective that the larger the design space, the greater the benefit.

5.4.3 Exploiting Architectural Symmetry

We propose to further improve the efficiency of slack allocation by taking advantage of the architectural symmetry present in modern platform-based MPSoC designs. In a wide variety of NoC topologies, groups of components have the same "view" of the rest of the system. Consider a ring of homogeneous processors: access to resources in the system is isomorphic for each processor. Even with heterogeneous resources, mesh, torus, tree, and other regular topologies may have components for which access to other components is identical or similar to that of other components in the system. As slack allocated to any one resource with such an isomorphic view may be equivalent to slack allocated to another, when task mappings

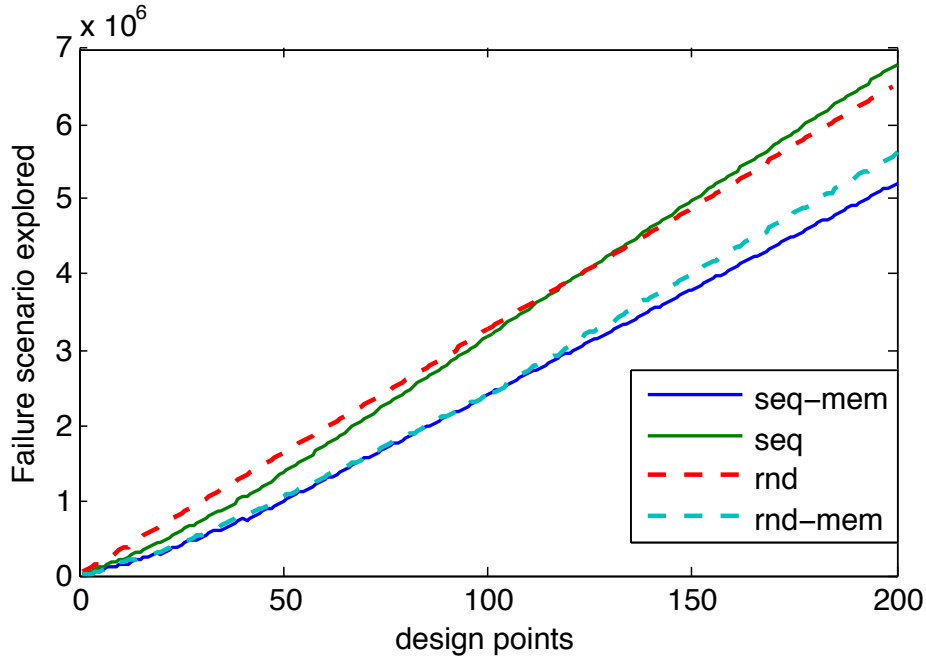


Figure 5.5: *Number of scenarios evaluated during the exploration of the MPEG4 CPL1 benchmark with (-mem) and without the proposed memoization technique using random (rnd) or sequential (seq) selection*

are changed accordingly, such symmetry presents an opportunity for design space pruning: only one such design needs to be evaluated. We speculate that even design points that are almost, but not perfectly, symmetrical will have very similar areas and expected lifetimes. We trade off some accuracy for exploration speed by avoiding the evaluation of configurations with a high degree of symmetry with regard to previously evaluated design points.

To determine the degree of symmetry between design points we introduce a correlation-based architecture distance metric. When the distance between two designs is sufficiently small, the area and MTTF evaluation for one may be used for the other, accelerating design space exploration.

A NoC topology defines the structure of a concrete instance of a network. Typically a topology is defined in terms of computing cores that

are connected to the network by the means of *switches*. The switches in turn are interconnected using links. We assume that each core and each memory has bi-directional links to a switch. Each switch can be connected to multiple cores and memories.

Definition 5.4.2. *An* island *is a set of components (memory/processor) plus one directly connected network switch.*

Definition 5.4.3. *We define the* distance $d(I_A, I_B)$ *between two islands* I_A, I_B *in a network* N *as the minimum number of hops needed for their communication through* N*:*

$$\forall I_A, I_B \in N, \quad d(I_A, I_B) = \min_k \{L_k\}$$

with $L_k = (l_1, l_2, \ldots l_k)$*, where* k *is the* cardinality *of the path* L_k *formed by the hops* $l_1, l_2, \cdots l_k$*.*

Definition 5.4.4. *Given a communication network, an island* I_A *is* equivalent *to another island* I_B*, denoted by* $\equiv$*, if it contains the same number and type of components as* I_B*. Here,* M *is the number of components* C_k^A, C_k^B *in the islands* I_A, I_B*.*

$$I_A \equiv I_B \rightarrow \{C_k^A\} \equiv \{C_k^B\}$$

with $C_k^A \in I_A, C_k^B \in I_B$*, and* $k = 1, \ldots, M$*.*

Definition 5.4.5. *Given a communication network,* $I_A \sim I_B$ *means that an* island I_A *is* **symmetric** *to an island* I_B *if: i)* $I_A \equiv I_B$ *(the islands* I_A*,* I_B *are* equivalent*), or ii)*

$$d(I_A, I_n) = d(I_B, I_m) \quad \forall \{I_n, I_m\} \in N, \quad I_n \equiv I_m$$

That is, the two islands I_A*,* I_B *have the same distance from all the equivalent islands* $\{I_n, I_m\}$ *in the network* N*.*

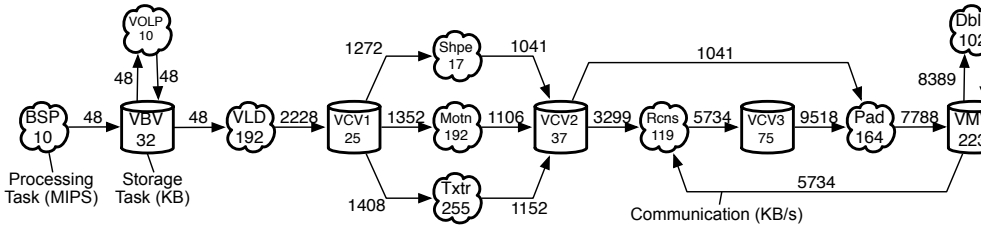


Figure 5.6: *Task graph for the MPEG4-CPL1 application*

Definition 5.4.6. *Given two symmetric islands I_A and I_B, we define the* intra-island distance *as the sum of the minimum distances between the permutations ${}_{a_A}P_M, {}_{a_B}P_M$ of all the possible slack allocations a_A, a_B in the M processor and memory components of the islands I_A and I_B:*

$$d_{intra-I}(I_A, I_B) = \sum d({}_{a_A}P_M, {}_{a_B}P_M), \quad I_A \sim I_B$$

Definition 5.4.7. *Given two architecture instances N_i, N_j of the network N, we define the* architecture distance *as the sum of the* intra-island distances *of all sets of symmetric islands in the network N:*

$$d_{arch}(N_i, N_j) = \sum_{i \in N_i, j \in N_j} d_{intra-I}(I_i, I_j), \quad I_i \sim I_j$$

Let us consider the example in Figure 5.7(b). This communication architecture presents a couple of symmetric islands, namely I_A and I_D. Given any permutation of the same execution and storage slack allocation within those islands, according to our definition the *intra-island distance* $d_{intra-I}(I_A, I_D)$ will be 0. We can assume that design points with $d_{intra-I}(I_A, I_D) = 0$ and the same slack allocations for all the other components would give similar values of system lifetime and area. We confirm this hypothesis by checking the correlation of lifetime results for pairs of design points. Given the architecture in Figure 5.7(b) and the application in Figure 5.6, we create a random set of 450 design points. Then, we compute the architecture distance between the design point with the lowest MTTF and all the others, and sort the design points according to

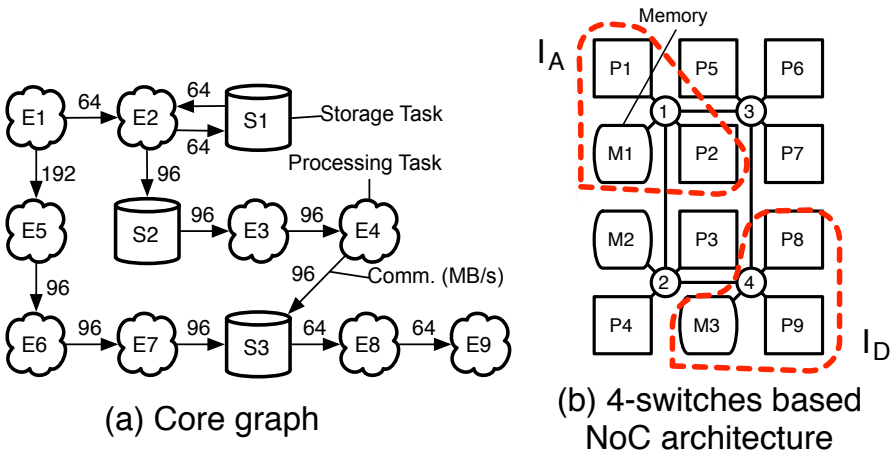


Figure 5.7: *Task graph for the MWD application and its communication architecture*

ascending architecture distance.

Our results are shown in Figure 5.8: the blue line is the ordered MTTF difference between the shortest-lived design point and all other points, and the green line is the corresponding architecture distance. It is easy to verify that the two curves are highly correlated ($\rho = 0.96$), indicating that higher architecture distance leads to higher difference in MTTF between design points. We also observe that design points with the same architectural distance have roughly the same MTTF difference from a reference. For example, Figure 5.8 shows that an architecture distance of 2 corresponds to a maximum MTTF difference 0.45 years.

These results suggest that we can set an architectural distance threshold T_D and reuse previously estimated MTTF values when the architectural distance of a new design point from some other previously evaluated design is below the threshold. We, therefore, only simulate when the distance from the new design to every other previously evaluated design is greater than T_D. It is worth noting that the selection of the threshold depends on the design space and its optimal value can only be found via sampling. However, we found that a conservative threshold $T_D = 2$ did not negatively

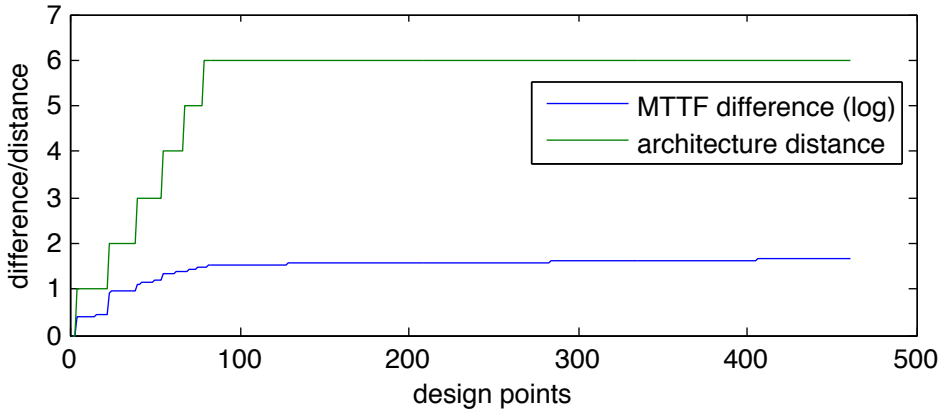


Figure 5.8: *MTTF difference (blue) and Architectural distance (green) between design points for the MPEG4 CPL1 benchmark*

affect the accuracy of our DSE while providing a substantial reduction in the number of explored design points.

5.5 Experimental Results

In our experiments we use two benchmark applications: Multi-Window Display (MWD) [48], and an MPEG-4 Core Profile Level 1 (MPEG4-CPL1) decoder [47, 58]. We show our task graph for MWD and MPEG4-CPL1 in Figure 5.7 and Figure 5.6, respectively. These applications are mapped on 8- and 9-core MPSoCs using different NoC topologies: ring, star, mesh, and tree. We constructed our systems using components from a library consisting of three different ARM processors (M3, ARM9, ARM11), nine SRAMs sized from 64 KB to 2 MB, and network switches with 3x3, 4x4, and 5x5 crossbars.

The ring, star and tree-based NoC designs use five switches to interconnect nine processors and four memories, while the mesh-based design uses eight processors and four memories. Execution slack is allocated by replacing the ARM M3s (125 MIPS, millions of instructions per second) with ARM9s (250 MIPS) or ARM11s (500 MIPS), or by replacing ARM9s with ARM11s. Storage slack is allocated by replacing smaller memories

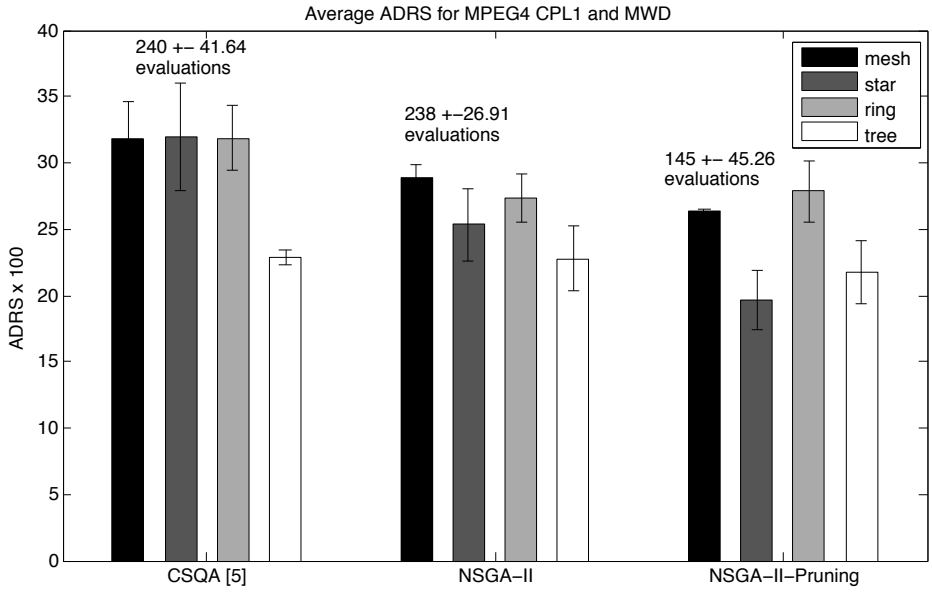


Figure 5.9: *Average ADRS and Evaluation Count for the MPEG4 CPL1 and MWD applications*

with larger ones. We enforce a two-communication-port-per-core limit.

In our experiments, we model failures due to electromigration, time-dependent dielectric breakdown, and thermal cycling [41]. The MTTF of each failure mechanism for each component type is normalized to 30 years for the characterization temperature of 345 K [50]. For the $NSGA - II$ algorithm we use a population of 70 design points and 200 generations.

Our results are summarised in Figure 5.9 and Figure 5.10. We compare the performance of (a) NSGA-II with our proposed pruning approach (NSGA-II-pruning), (b) standard NSGA-II without pruning (NSGA-II), and (c) the Critical Quantity Slack Allocation (CQSA) heuristic approach [67], using two assessment criteria:

- *Quality of the solution set* (ADRS). How well each technique approximates the reference Pareto front (which was obtained using exhaustive exploration) after running to convergence (see Figure 5.9). In par-

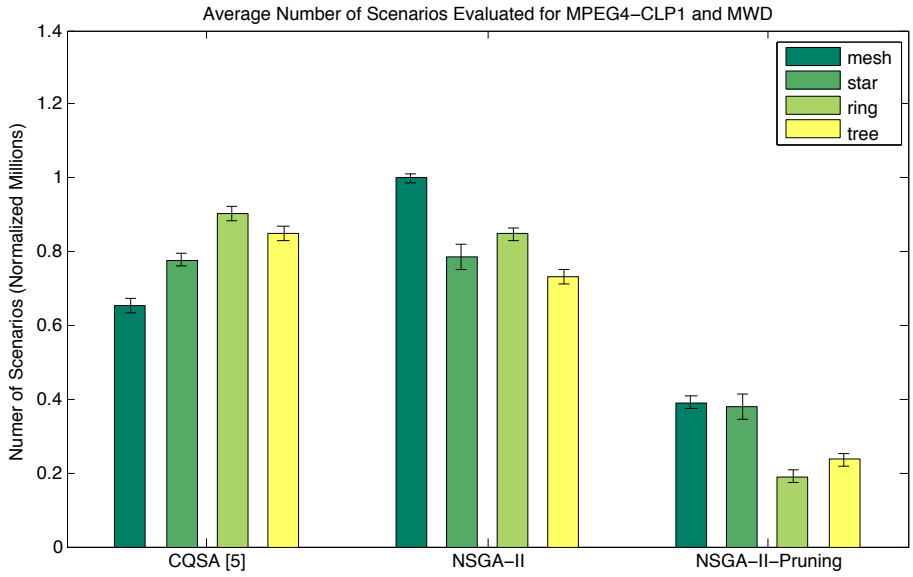


Figure 5.10: *Sample Count for the MPEG4 CLP1 and MWD applications*

ticular, we use the *Average Distance from Reference Set* (ADRS) [26], which measures the distance between the solutions set $p(A)$ and the Pareto-optimal set R, obtained through exhaustive search:

$$ADRS(p(A), R) = \frac{1}{|R|} \sum_{x_p \in R} \min_{\vec{a} \in p(A)} d\{\vec{x_p}, \vec{a}\}$$

where

$$d\{\vec{x_p}, \vec{a}\} = \max_{j=1,\cdots,M} \left\{0, \frac{f_j(\vec{a}) - f_j(\vec{x_p})}{f_j(\vec{x_p})}\right\}$$

and M is the number of objective functions. A smaller ADRS value indicates that the distribution of the solutions is closer to the reference Pareto front and, therefore, better.

- *Number of evaluations and samples.* The total number of design points (shown in Figure 5.9 on top of the bars), and Monte Carlo simulation samples (shown in Figure 5.10, needed by the optimisation

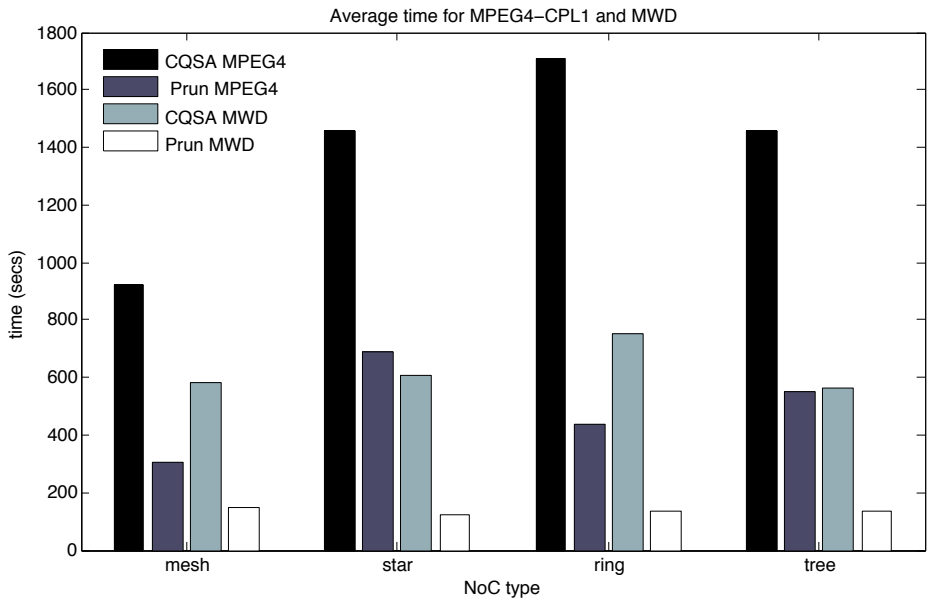


Figure 5.11: *Average DSE time for the MPEG4 CPL1 and MWD applications*

techniques to run to convergence.

5.5.1 Quality of the solution set

We observe that our methodology always produces more accurate results than CQSA, with a global improvement from 0 to 61%. This is due to the fact that CQSA uses a greedy approach which easily leads to sub-optimal results, while our method employs a more effective genetic multi-objective optimization algorithm.

To verify that no additional error is introduced with our pruning methodology, we compare its ADRS with the ADRS obtained using a standard NSGA-II without pruning. The error bars in Figure 5.9 show that the ADRS of our method is not significantly different w.r.t. NSGA-II without pruning, proving that our architecture distance's error is negligible.

This is confirmed by a paired *t-test* with the null hypothesis being that matched samples from NSGA-II and NSGA-II-pruning come from distributions with equal means. The difference between the two distributions is

assumed to come from a normal distribution with unknown variance. The results of the *t-test* for all the four types of NoC-architectures does not reject the null hypothesis with high p-values[2], e.g. $p = 0.6986$ for the star topology.

5.5.2 Number of evaluations and samples

Having assured that our methodology does not negatively affect the accuracy of the exploration, we determine its effect in terms of performance improvement. Using memoization, our methodology reduces the total number of samples explored by $63 \pm 14\%$ on average across the two applications (MPEG4 CPL1 and MWD). The normalized improvement is shown in Figure 5.10 The simulation time is further reduced by the application of our symmetry-based pruning: the number of explored points is reduced by $38 \pm 19\%$, as shown inf Figure 5.9 above the bars.

To understand the actual impact of these results on simulation time, we present the average wall-clock times for a DSE experiment, using both CQSA [67] and the proposed NGSA-II-pruning method in Figure 5.11 (the average wall-clock timing of the NSGA-II without pruning approach is equivalent to CQSA [67] and, therefore, not shown).

These results indicate that our pruning-based approach is a promising technique, yielding to solutions similar in terms of quality but at a fraction of the execution time when compared to the state-of-the-art.

The total improvement for the limited-size examples used in this chapter (to allow for exhaustive search) is above 3 times when compared to CQSA [67] and NSGA-II without pruning.

An interesting characteristic of our method is that it scales with the complexity of the application and architectures used, giving higher rewards for larger design spaces. For instance, using a more complex architecture with 20 processors, 5 memories and 10 switches, corresponds to a reduction in exploration time from ∼30 hours to around 6.5 hours, meaning a factor

[2]The $p - value$ is the probability of observing the given result, or one more extreme, by chance if the null hypothesis is true. Small values of P cast doubt on the validity of the null hypothesis.

of 4.7 on our 8-core Intel i7 @ 2.93Ghz for the MPEG4 CPL1 benchmark with a ring topology. This shows promise of better results for larger design spaces.

5.6 Conclusions

In this chapter we presented two techniques for accelerating design space exploration (DSE) of slack allocation: *failure scenario memoization*, which speeds up MTTF estimation by reusing partial results, and *symmetry thresholding*, a set of metrics to identify similarities among system configurations to reduce the number of MTTF evaluations needed during system-level DSE for reliability. We verified the proposed techniques on four NoC topologies using two different applications.

Our approach globally reduces the number of design points evaluations needed during system-level DSE by a factor from 3 to 5, maintaining the same level of accuracy of state-of-the-art exploration algorithms.

Chapter 6

Conclusion

The design of modern embedded systems has become increasingly complex. There is a wide range of design parameters that have to be tuned up to find the optimal tradeoff in terms of several design requirements. Those systems should be low cost, small in terms of area, light weight and be power efficient, since they are often battery-based devices. This is in contrast with the requirements of achieving real-time, performance and providing reliable and secure operation. As result, the increasing market for compact embedded computing devices is leading to new multi-processor system-on-a-chip (MPSoC) architectures designed for embedded systems, providing task-level parallelism for streaming applications integrated in a single chip.

Platform based design of heterogeneous multi-processor system-on-chip (MPSoC) systems is becoming today's predominant design paradigm in the embedded systems domain [81]. In contrast to more traditional design paradigms, platform based design shortens design time by eliminating the effort of the low-level design and implementation of system components. A platform based design environment typically consists of a fixed, parameterizable platform or a set of (parameterizable) components that can be combined in specific ways to compose a platform.

In this thesis, we have investigated the problem of optimising DSE for

- *searching in the design space*

- *evaluating a single design point in the design space*

6.1 Discussion

Our problem definition (Section 1.2) concluded with the following research question: *"How can we use pruning techniques to speed up the evaluation of a design point and optimise the search in design space?"*. This question was answered in Chapters 4 and 5, where pruning techniques were introduced for optimising system performance and lifetime. We gave an overview of the preliminary information necessary for understanding the rest of the thesis in Chapter 2. We first described the basic knowledge about multi-objective optimisation problems. Then, we explained the multi-objective optimisation problem in the context of design space exploration of embedded systems. We describe evolutionary algorithms as heuristic methods for searching in the design space, with a brief description of the genetic algorithm NSGA-II we used throughout this thesis. Afterwards, we discussed several metrics for evaluating the quality of the solutions obtained while performing design space exploration using heuristic search.

Based on the research presented in this thesis, we draw the following major contributions:

1. We extended the objective space with the introduction and implementation of a complete framework for high-level power estimation for MPSoC. The technique is based on abstract execution profiles, called event signatures, and it operates at a higher level of abstraction than, e.g., commonly-used instruction-set simulator (ISS) based power estimation methods and it is proven to be capable of achieving good evaluation performance. Since every design point evaluation takes only 0.16 seconds on average, the presented power model offers remarkable potentials for quickly experimenting with different MPSoC architectures and exploring system-level design options during the very early stages of design.

2. We introduced a new, hybrid form of DSE, combining simulations

with analytical estimations to prune the design space in terms of application mappings that need to be evaluated using simulation. For this purpose, the DSE technique uses an analytical model that estimates the expected throughput of an application (which is a natural performance metric for the multimedia and streaming application domain we target) given a certain architectural configuration and application-to-architecture mapping. In the majority of the search iterations of the DSE process, the throughput estimation avoids the use of simulations to evaluate the design points. However, since the analytical estimations may in some cases be less accurate, the analytical estimations still need to be interleaved with simulative evaluations in order to ensure that the DSE process is steered into the right direction.

We studied different techniques for interleaving these analytical and simulative evaluations in our hybrid DSE. Experimental results have demonstrated that such hybrid DSE is a promising technique that can yield solutions of similar quality as compared to simulation-based DSE but only at 15% of the execution time.

3. We proposed an exploration framework for Network-on-Chip (NoC) based MPSoCs that substantially reduces the computational cost of slack allocation. First, we develop *failure scenario memoization* to reduce the computational cost of lifetime estimation by storing and reusing estimated lifetime values for systems with one or more failed components.

4. We introduced a correlation-based architecture distance metric to identify symmetries for clusters of components called *islands*. In modern platform- and network-on-chip based design, components are clustered around switches in the on-chip network. When clusters and the tasks mapped to them are considered to be symmetric, some configurations have the same effect on the overall system lifetime. This can be leveraged to reduce the number of evaluations. We verified the proposed techniques on four NoC topologies using two

different applications.

This approach globally reduces the number of design points evaluations needed during system-level DSE by a factor from 3 to 5, maintaining the same level of accuracy of state-of-the-art exploration algorithms.

6.2 Open Issues and Future Directions

There are several interesting further research directions based on the contributions in this thesis.

For istance, in this thesis, we consider the problem of reducing the simulation overhead in system-level DSE. To this end, we have presented an iterative design space pruning methodology based on static throughput analysis of different application mappings. However, the analytical estimations may in some cases be not accurate enough, because of estimation inaccuracies due to topological cycles in the dataflow graphs that are generated and used for throughput estimation during the analytical mapping exploration. There is an opportunity to improve the model by using Maximum Cycle Mean (MCM) analysis. MCM analysis can be used to correct the estimation errors due to the topological cycles generated during analytical mapping design space exploration. Since the throughput calculation needs to be fast and sufficiently accurate at the same time, we propose an *approximated MCM analysis* which improves the estimation proposed in the previous sections, achieving performance faster than regular MCM analysis [36].

In Chapter 2, we extended the design space by presenting a framework for high-level power estimation of multiprocessor systems-on-chip (MPSoC) architectures on FPGA. We have incorporated the power models in a (highly automated) system-level MPSoC synthesis framework, allowing for accurate and flexible validation of the models. Within this context, an other interesting direction would be integrating *security* as possible metric in the early stage of DSE. Security has been intensively studied in the areas of cryptography, computing, and networking. However, security

is not yet well perceived by designers as the hardware or software implementation of specific cryptographic algorithms and security protocols [56]. The first steps for this future work are already taken in [11].

Chapter 7

Appendix

7.1 Using CQSA: example

CQSA assumes as inputs:

- a description of a performance-constrained application, including computation, storage and communication requirements for each software task;
- a fixed communication architecture for a single-chip multiprocessor, including an initial selection of processors, memories, switches and their interconnection; and,
- an initial task-resource mapping, including an assignment of computational tasks to processors, storage tasks to memories, and communication to links and switches.

To facilitate reuse and accommodate internally used tools, these inputs are divided into three files:

- the task graph file, which enumerates the tasks in the system, lists the computational or storage requirements of each, as well as the communication that occurs between pairs of tasks;

- the architecture file, which enumerates the components in the system, each selected from an internal component library, and assigns tasks from the task graph to each; and,
- the netlist file, which specifies how components in the architecture file are initially interconnected.

An example with an MPEG-1 application mapped onto a mesh NoC is given below:

Architecture description and mapping

```
define components

proc1 M3 3 bsp volp vld
proc2 M3 1 dblk
proc3 M3 2 drng1 pad
proc5 M3 1 drng2
proc6 M3 1 rcns
proc7 M3 2 shpe motn
proc8 M3 1 txtr1
proc9 M3 1 txtr2

mem1 MEM64KB 2 vbv vcv1
mem2 MEM64KB 1 vcv2
mem3 MEM128KB 1 vcv3
mem4 MEM256KB 1 vmv

s_1 SW5X5 0
s_2 SW5X5 0
s_3 SW5X5 0
s_4 SW5X5 0

end

define preclusions

s_1  proc1 proc7 mem1
s_4 proc8 proc9 mem2
s_3 proc6 mem3 proc5
s_2 proc2 proc3 mem4
```

```
end
```

The architecture file specifies all of the components (processors, memories and switches) in the system, how tasks (defined in the task graph file) are initially assigned to them, and any failure dependencies that may exist. Components are selected from the internal component library.

Task graph

```
define computation

bsp 30
volp 30
vld 29
shpe 50
motn 50
txtr1 125
txtr2 125
rcns 60
pad 16
dblk 106
drng1 105
drng2 105

vbv 32
vcv1 25
vcv2 37
vcv3 55
vmv 22

end

define communication

bsp vbv 48
vbv volp 48
volp vbv 48
vbv vld 48
vld vcv1 2228
vcv1 shpe 1272
vcv1 motn 1352
vcv1 txtr1 704
vcv1 txtr2 704
```

```
shpe vcv2 1041
motn vcv2 1106
txtr1 vcv2 575
txtr2 vcv2 575
vcv2 rcns 3299
vcv2 pad 1041
rcns vcv3 5734
vcv3 pad 9518
pad vmv 7788
vmv rcns 5734
vmv dblk 8389
vmv drng1 15670
vmv drng2 15670
dblk vmv 5025
drng1 vmv 12080
drng2 vmv 12080

end
```

The task graph file specifies all tasks in the application, their sizes, and how they communicate. The initial assignment of tasks to resources is specified in the architecture file. There are two sections to the task graph file, the computation section, which specifies the name and size of tasks in the application, and the communication section, which specifies the communication between tasks in the application.

Netlist

```
UCLA nets     1.0

NumNets : 18
NumPins : 36

NetDegree : 2   s_1-proc1
s_1 B
 proc1 B
NetDegree : 2   s_1-mem1
s_1 B
mem1 B
NetDegree : 2   s_1-proc7
s_1 B
proc7 B
NetDegree : 2   s_1-s_4
```

```
s_1 B
s_4 B
NetDegree : 2   s_1-s_2
s_1 B
s_2 B
NetDegree : 2   s_4-proc8
s_4 B
proc8 B
NetDegree : 2   s_4-proc9
s_4 B
proc9 B
NetDegree : 2   s_4-mem2
s_4 B
mem2 B
NetDegree : 2   s_4-s_3
s_4 B
s_3 B
NetDegree : 2   s_3-proc6
s_3 B
proc6 B
NetDegree : 2   s_3-proc5
s_3 B
proc5 B
NetDegree : 2   s_3-mem3
s_3 B
mem3 B
NetDegree : 2   s_3-s_2
s_3 B
s_2 B
NetDegree : 2 s_2-mem4
s_2 B
mem4 B

NetDegree : 2 s_2-proc3
s_2 B
proc3 B

NetDegree : 2 s_2-proc2
s_2 B
proc2 B
```

The netlist file specifies all of the links between resources in the system. In general, any resource may be connected to any other resource, though traffic may only be routed through switches.

Chapter 8

Samenvatting

Pruning Techniques for System-Level Design Space Exploration

Het ontwerp van moderne embedded systemen is steeds complexer geworden. Een hoge verscheidenheid aan parameters moeten afgewogen worden om uiteindelijk te kunnen voldoen aan de ontwerpeisen. Het uiteindelijke systeem moet dan klein in oppervlakte zijn, licht van gewicht zijn, en weinig energie verbruiken zodat ze gebruikt kunnen worden in mobiele apparaten. Dit is een tegenstelling ten opzichte van de real-time en hoge mate van snelheid, beschikbaarheid en veiligheid die deze systemen moeten hebben. Om hier toch aan te kunnen voldoen wordt een uitweg gezocht in multi-processor system-on-chip (MPSoC) architecturen. Deze systemen kunnen, op taak granulariteit, parallellisme aanbieden op een enkele chip.

Ontwerp parameter verkenning, Design Space Exploration, (DSE) is het maken van beslissingen gedurende het begin van het project zodat er minder implementaties mogelijk zijn. Hierdoor hoopt men de totale ontwerp last te verlagen. Onwerpruimte verkleining, Design Space Pruning, is het optimaliseren van het DSE proces om meer ontwerpen te kunnen proberen om zo sneller tot een optimaal ontwerp te komen.

Verkleiningstechnieken kunnen worden toegepast om:

- sneller het ontwerp te evalueren.

- de heuristiek te optimaliseren.

In elke ontwerpfase, is een subset van de niet-verkleinde ontwerp opties geselecteerd en geëvalueerd.

De belangrijkste bijdragen van dit proefschrift zijn:

- De uitbreiding van de ontwerpruimte met de introductie en implementatie van een compleet framework voor energie schatting van het MPSoC. De techniek is gebaseerd op abstracte programma uitvoeringsprofielen, genaamd event-handtekeningen. Dit werkt op een hoger abstractieniveau dan, bijvoorbeeld, de veelgebruikte instructieset simulator (ISS). Op basis van de energie schattingsmethoden zou men in staat moeten zijn tot een goede evaluatie van de prestaties.

  Dit is essentieel in het kader van de eerste fase van DSE.

- Een iteratieve ontwerpruimte verkleinings methodologie gebaseerd op statische doorvoer analyse van verschillende implementaties van toepassingen. Door een combinatie van deze analytische doorvoeringsschattingen met simulaties, vermindert onze hybride aanpak het aantal simulaties die nodig zijn tijdens het proces van DSE.

- Een studie naar de verschillende combinaties, snel, maar minder nauwkeurige analytische prestaties, langzaam, maar meer accurate simulaties tijdens DSE

- *Failure scenario memoization* vekleiningstechnieken om de computationele kosten van de levensduurs schatting van systemen te verminderen. Door het opslaan en hergebruiken van geschatte levensduur waarden van systemen met een of meer defecte onderdelen. De levensduur van alle gedeeltelijk mislukte systemen wordt afgeleid en opgeslagen (het geheugen opslag kosten van dergelijke waarden is te verwaarlozen); Wanneer een eerder verkend gedeeltelijk mislukte systeem wederom wordt verkend dan wordt de verwachte levensduur uit een database gelezen in plaats van opnieuw geschat.

- Correlatie-gebaseerde architectuur afstands eenheiden voor het efficïnt snoeien van de op tijdswinst gebaseerde DSE voor het verbeteren van de levensduur in systemen op basis van NOC MPSoCs. In de moderne platform- en netwerk-op-chip gebaseerde ontwerpen, zijn onderdelen geclusterd rond switches. Wanneer clusters en de taken die aan hen toegewezen zijn, gedefinieerd zijn als symmetrisch dan hebben sommige configuraties hetzelfde effect op de totale levensduur van het systeem. Dit kan worden benut om het aantal evaluaties te verminderen.

Om samen te vatten, dit proefschrift bestudeert verkleinings technieken om snel te kunnen zoeken in de ontwerpruimte en de evaluatie van een ontwerppunt volgens verschillende doelstellingen. Het proefschrift is daarom opgesplitst in de volgende onderdelen:

- achtergrond (hoofdstukken 1 en 2),
- uitbreiding van de ontwerpruimte met als doel de snelheid/energie (hoofdstuk 3), en
- verkleiningstechnieken voor de systeemprestaties (hoofdstuk 4) en levensduuroptimalisatie (Hoofdstuk 5 en Bijlage).

Roberta Piscitelli

List of Author's Publications

[1] P. Meloni, S. Pomata, L. Raffo, R. Piscitelli, and A. D. Pimentel. Combining on-hardware prototyping and high-level simulation for dse of multi-asip systems. In *ICSAMOS*, pages 310–317, 2012.

[2] R. Piscitelli and A. D. Pimentel. A high-level power model for mpsoc on fpga. In *IPDPS Workshops*, pages 128–135, 2011.

[3] R. Piscitelli and A. D. Pimentel. Design space pruning through hybrid analysis in system-level design space exploration. In *DATE*, pages 781–786, 2012.

[4] R. Piscitelli and A. D. Pimentel. A high-level power model for mpsoc on fpga. *Computer Architecture Letters*, 11(1):13–16, 2012.

[5] R. Piscitelli and A. D. Pimentel. Interleaving methods for hybrid system-level mpsoc design space exploration. In *ICSAMOS*, pages 7–14, 2012.

[6] R. Piscitelli and A. D. Pimentel. A signature-based power model for mpsoc on fpga. *VLSI Design*, 2012, 2012.

[7] Piscitelli R., Meyer B., Pimentel A. D., and Beltrame G. Design space pruning for efficient slack allocation and lifetime estimation for noc-based mpsocs. In *under submission*, 2014.

Bibliography

[8] Cacti 5.3. http://www.hpl.hp.com/research/cacti.

[9] http: //www.arm.com/products/processors/index.php.

[10] http://pmbus.org/specs.html.

[11] *A Security-enhanced Design Methodology For Embedded Systems*, Reykjavik, Iceland, 07/2013 2013. ICETE.

[12] Saurabh N. Adya and Igor L. Markov. Fixed-outline floorplanning: enabling hierarchical design. *IEEE Trans. VLSI Syst.*, 11(6):1120–1135, 2003.

[13] G. Ascia, V. Catania, A. G. Di Nuovo, M. Palesi, and D. Patti. Efficient design space exploration for application specific systems-on-a-chip. *J. Syst. Archit.*, 53:733–750, October 2007.

[14] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computer system modeling. *Computer*, 35:59–67, 2002.

[15] S. Dekeyser J.-L. B. Atitallah, R. Niar. MPSoC power estimation framework at transaction level modeling. *ICM 2007.*

[16] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996.

[17] Mohamed Bamakhrama, Jiali Teddy Zhai, Hristo Nikolov, and Todor Stefanov. A methodology for automated design of hard-real-time embedded streaming systems. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 941–946, 2012.

[18] J. Becker, M. Huebner, and M. Ullmann. Power estimation and power measurement of xilinx virtex fpgas: Trade-offs and limitations. In *SBCCI '03: Proceedings of the 16th symposium on Integrated circuits and systems design*, page 283, Washington, DC, USA, 2003. IEEE Computer Society.

[19] Greet Bilsen, Marc Engels, Rudy Lauwereins, and J. Peperstraete. Cyclo-Static Dataflow. *IEEE Trans. on Signal Processing*, 44(2):397–408, 1996.

[20] Greet Bilsen, Marc Engels, Rudy Lauwereins, and J. A. Peperstraete. Cycle-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, 1996.

[21] Greet Bilsen, Marc Engels, Rudy Lauwereins, and J. A. Peperstraete. Cycle-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, 1996.

[22] C. Zhu, *et al.* Reliable multiprocessor system-on-chip synthesis. In *CODES+ISSS'07*, October 2007.

[23] V. Catania and M. Palesi. A multi-objective genetic approach to mapping problem on network-on-chip. *JUCS*, 22:2006.

[24] F. Catthoor, K. Danckaert, K.K. Kulkarni, Brockmeyer, E., P.G. Kjeldsberg, van Achteren, and T. T., Omnes. *Data Access and Storage Management for Embedded Programmable Processors*. Springer, 2002.

[25] A.K. Coskun, T.S. Rosing, and K. Whisnant. Temperature aware task scheduling in mpsocs. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1–6, 2007.

[26] Piotr Czyzak and Adrezej Jaszkiewicz. Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47, 1998.

[27] Morteza Damavandpeyma, Sander Stuijk, Marc Geilen, Twan Basten, and Henk Corporaal. Parametric throughput analysis of scenario-aware dataflow graphs. In *ICCD*, pages 219–226. IEEE Computer Society, 2012.

[28] Kalyanmoy Deb. Evolutionary algorithms for multi-criterion optimization in engineering design, 1999.

[29] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2000.

[30] F. Deprettere and Todor Stefanov. Affine nested loop programs and their binary cyclo-static dataflow counterparts. In *In Proc. of the Intl. Conf. on Application Specific Systems, Architectures, and Processors, Steamboat*, pages 186–190, 2006.

[31] N. Eisley, V. Soteriou, and L. Peh. High-level power analysis for multi-core chips. In *CASES '06: Proc. of the 2006 int. conference on Compilers, architecture and synthesis for embedded systems*, pages 389–400, USA, 2006.

[32] M. T. M. Emmerich, K. Giannakoglou, and B. Naujoks. Single- and multiobjective evolutionary optimization assisted by gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation*, 10:421–439, 2006.

[33] C. Erbas, S. Cerav-erbas, and A. D. Pimentel. Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. *IEEE Transactions on Evolutionary Computation,vol.10,no.3*, 10:358–374, 2006.

[34] P. Feautrier. Automatic parallelization in the polytope model. In *The Data Parallel Programming Model: Foundations, HPF Realization, and Scientific Applications*, pages 79–103, London, UK, UK, 1996. Springer-Verlag.

[35] D. Sciuto G. Beltrame and C. Silvano. Multi-accuracy power and performance transaction-level modeling. In *DATE '08: Proc. of the conference on Design, automation and test in Europe*.

[36] A. H. Ghamarian, M. C. W. Geilen, T. Basten, and S. Stuijk. Parametric throughput analysis of synchronous data flow graphs. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '08, pages 116–121, New York, NY, USA, 2008. ACM.

[37] A. H. Ghamarian, M. C. W. Geilen, T. Basten, and S. Stuijk. Parametric throughput analysis of synchronous data flow graphs. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '08, pages 116–121, New York, NY, USA, 2008. ACM.

[38] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, B. D. Theelen, M. R. Mousavi, A. J. M. Moonen, and M. J. G. Bekooij. Throughput analysis of synchronous data flow graphs. In *Proceedings of the Sixth International Conference on Application of Concurrency to System Design*, ACSD '06, pages 25–36, Washington, DC, USA, 2006. IEEE Computer Society.

[39] M. Gries. Methods for evaluating and covering the design space during early design development. *Integr. VLSI J.*, 38:131–183, December 2004.

[40] Zhenyu Gu, Changyun Zhu, Li Shang, and Robert P. Dick.

Application-specific mpsoc reliability optimization. *IEEE Trans. VLSI Syst.*, 16(5):603–608, 2008.

[41] Zhenyu Gu, Changyun Zhu, Li Shang, and Robert P. Dick. Application-specific MPSoC reliability optimization. *IEEE Trans. on VLSI*, May 2008.

[42] Nikolov H. System-level design methodology for streaming multiprocessor embedded systems. In *Ph.D. Thesis*, 2009.

[43] Adam S. Hartman, Donald E. Thomas, and Brett H. Meyer. A case for lifetime-aware task mapping in embedded chip multiprocessors. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES/ISSS '10, pages 145–154, New York, NY, USA, 2010. ACM.

[44] J.W. Herrmann, C.Y. Lee, J.L. Snowdon, University of Florida. Department of Industrial, and Systems Engineering. *A Classification of Static Scheduling Problems.* Department of Industrial and Systems Engineering, University of Florida, 1992.

[45] J. Hu and R. Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. In *ASP-DAC '03: Proc. of the 2003 Asia and South Pacific Design Automation Conference*, pages 233–239, USA, 2003.

[46] P. Huang, M. Hashemi, and S. Ghiasi. System-level performance estimation for application-specific MPSoC interconnect synthesis. In *SASP '08: Proc. of the 2008 Symposium on Application Specific Processors*, pages 95–100, USA, 2008.

[47] J. Kneip *et al.* The MPEG-4 video coding standard - a VLSI point of view. In *the Proceedings of the 1998 Workshop on Signal Processing Systems, SiPS'98*, 1998.

[48] Antoine Jalabert, Srinivasan Murali, Luca Benini, and Giovanni De Micheli. ×pipesCompiler: A tool for instantiating application spe-

cific networks on chip. In *the Proceedings of the 2004 Conference on Design, Automation and Test in Europe, DATE'04*, Feb. 2004.

[49] Z. J. Jia, A.D. Pimentel, M. Thompson, T. Bautista, and A. Núñez. Nasa: A generic infrastructure for system-level mp-soc design space exploration, Proceedings of the IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia '10) 2010.

[50] Joint Electron Device Engineering Council. Failure mechanisms and models for semiconductor devices. *Publication JEP122C*, March 2006.

[51] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. Construction and use of linear regression models for processor performance analysis. In *In Proc. 12th IEEE Symposium on High Performance Computer Architecture*, pages 99–108, 2006.

[52] G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of the IFIP Congress 74*, 1974.

[53] Andrew B. Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *DATE*, pages 423–428, 2009.

[54] J. Kim and M. Orshansky. Towards formal probabilistic power-performance design space exploration. In *Proceedings of the 16th ACM Great Lakes symposium on VLSI*. ACM, 2006.

[55] J. Knowles. Parego: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1), 2006.

[56] Paul Kocher, Ruby Lee, Gary McGraw, and Anand Raghunathan. Security as a new dimension in embedded system design. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, pages 753–760, New York, NY, USA, 2004. ACM. Moderator-Ravi, Srivaths.

[57] S. Koohi, M. Mirza-Aghatabar, S Hessabi, and M. Pedram. High-level modeling approach for analyzing the effects of traffic models on power and throughput in mesh-based nocs. In *VLSID '08: Proc. of the 21st Int. Conference on VLSI Design*.

[58] P. Kuhn and W. Stechele. Complexity analysis of the emerging MPEG-4 standard as a basis for VLSI implementation. In *the Proceedings of SPIE 3309 VCIP Visual Communications and Image Processing*, January 1998.

[59] Chao L. and Sha E. H. Static scheduling for synthesis of dsp algorithms on various models. *Journal of VLSI Signal Processing*, 10:207–223, 1994.

[60] Edward Ashford Lee and David G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.*, 36(1):24–35, January 1987.

[61] I. Lee, H. Kim, P. Yang, S. Yoo, E. Chung, K. Choi, J. Kong, and S. Eo. Powervip: Soc power estimation framework at transaction level. In *Proc. of the 2006 ASP-DAC '06*. IEEE Press, 2006.

[62] C.E. Leiserson and James B. Saxe. Optimizing synchronous systems. In *Foundations of Computer Science, 1981. SFCS '81. 22nd Annual Symposium on*, pages 23–36, Oct.

[63] M. Loghi, L. Benini, and M. Poncino. Power macromodeling of MPSoC message passing primitives. *ACM Trans. Embed. Comput. Syst.*, 6(4):31, 2007.

[64] H. Nikolov A. D. Pimentel C. Erbas S. Polstra M. Thompson, T. Stefanov and E. F. Deprettere. A framework for system-level modeling and simulation of embedded systems architectures. *EURASIP J. Embedded Syst.*, 2007(1):2–2, 2007.

[65] G. Mariani, A. Brankovic, G. Palermo, J. Jovic, V. Zaccaria, and C. Silvano. A correlation-based design space exploration methodology

for multi-processor systems-on-chip. In *Proceedings of the 47th Design Automation Conference (DAC)*. ACM, 2010.

[66] S. Meijer, H. Nikolov, and T. Stefanov. Throughput modeling to evaluate process merging transformations in polyhedral process networks. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 747–752, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[67] Brett H. Meyer, Adam S. Hartman, and Donald E. Thomas. Cost-effective slack allocation for lifetime improvement in noc-based mpsocs. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 1596–1601, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[68] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1996.

[69] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. *SIGPLAN Not.*, 2002.

[70] M. Monchiero, G. Palermo, C. Silvano, and O. Villa. A modular approach to model heterogeneous MPSoC at Cycle Level. In *DSD '08: Proc. of the 2008 11th EUROMICRO Conf. on Digital System Design Architectures, Methods and Tools*, 2008.

[71] H. Muller. Simulating computer architectures. PhD thesis, Department of Computer Science, University of Amsterdam, 1993.

[72] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *MICRO 40: Proc. of the 40th Annual IEEE/ACM Int. Symposium on Microarchitecture*, 2007.

[73] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. Deprettere. Daedalus: Toward com-

posable multimedia MPSoC design. In *Proc. of the 45th ACM/IEEE Int. Design Automation Conference (DAC '08)*, 2008.

[74] H. Orsila, E. Salminen, and T. D. Hämäläinen. Parameterizing simulated annealing for distributing kahn process networks on multiprocessor socs. In *Proceedings of the 11th international conference on System-on-chip*, SOC'09, pages 19–26, Piscataway, NJ, USA, 2009. IEEE Press.

[75] L. Ost, G. Guindani, L. Indrusiak, S. Maatta, and F. Moraes. Using abstract power estimation models for design space exploration in NoC-based MPSoC. *IEEE Design and Test of Computers*, 99(PrePrints), 2010.

[76] J. Ou and V. K. K. Prasanna. Rapid energy estimation for hardware-software codesign using fpgas. *EURASIP J. Embedded Syst.*, 2006(1):17–17, 2006.

[77] G. Palermo, C. Silvano, and V. Zaccaria. Respir: a response surface-based pareto iterative refinement for application-specific design space exploration. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 28, 2009.

[78] Andy D. Pimentel, Cagkan Erbas, and Simon Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Trans. Comput.*, 55(2), 2006.

[79] R. Piscitelli and A. D. Pimentel. Design space pruning through hybrid analysis in system-level design space exploration. In *DATE*, pages 781–786, 2012.

[80] S. NIAR E. SENN S. K. RETHINAGIRI, R. B. ATITALLAH and J. DEKEYSER. Hybrid system level power consumption estimation for fpga-based mpsoc. 29th IEEE International Conference on Computer Design (ICCD), October 2011.

[81] A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and

software design methodology for embedded systems. *IEEE Des. Test*, 18, 2001.

[82] D. Sheldon, F. Vahid, and S. Lonardi. Soft-core processor customization using the design of experiments paradigm. In *In International Conference on Design and Test in*, 2007.

[83] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim.*, 1(1):94–125, March 2004.

[84] Jelena Spasic and Todor Stefanov. An accurate energy model for streaming applications mapped on mpsoc platforms. In *ICSAMOS'13*, pages 205–212, 2013.

[85] Jayanth Srinivasan, Sarita V. Adve, Pradip Bose, and Jude A. Rivers. Exploiting structural duplication for lifetime reliability enhancement. In *ISCA'05*, June 2005.

[86] Jayanth Srinivasan, Sarita V. Adve, Pradip Bose, and Jude A. Rivers. Exploiting structural duplication for lifetime reliability enhancement. *SIGARCH Comput. Archit. News*, 33(2):520–531, 2005.

[87] Sundararajan Sriram and Shuvra S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization.* Marcel Dekker, Inc., New York, NY, USA, 1st edition, 2000.

[88] P. Stralen and A. D. Pimentel. A high-level microprocessor power modeling technique based on event signatures. In *Proc. of the IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia '07)*, 2007.

[89] T. Taghavi and A.D. Pimentel. Design metrics and visualization techniques for analyzing the performance of moeas in dse. In *Proc. of the 11th Int. Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS '11)*, 2011.

[90] Mark Thompson, Hristo Nikolov, Todor Stefanov, Andy D. Pimentel, Cagkan Erbas, Simon Polstra, and Ed F. Deprettere. A framework for rapid system-level exploration, synthesis, and programming of multimedia MPSoCs. In *Proc. of the IEEE/ACM int. conference on Hardware/software codesign and system synthesis*, 2007.

[91] Mark Thompson and Andy D. Pimentel. Exploiting domain knowledge in system-level mpsoc design space exploration. *J. Syst. Archit.*, 59(7):351–360, August 2013.

[92] Ankush Varma, Bruce Jacob, Eric Debes, Igor Kozintsev, and Paul Klein. Accurate and fast system-level power modeling: An xscale-based case study. *ACM Trans. Embed. Comput. Syst.*, 6(4):26, 2007.

[93] Sven Verdoolaege. Polyhedral Process Networks. In Shuvra Bhattacharrya, Ed Deprettere, Rainer Leupers, and Jarmo Takala, editors, *Handbook on Signal Processing Systems*, pages 931–965. Springer, 2010.

[94] Xilinx. `http://www.xilinx.com/products/design_tools/logic_design/verification/xpower.htm`.

[95] J. J. Yi, D. J. Lilja, and D. M. Hawkins. A statistically rigorous approach for improving simulation methodology. In *Proc. of the 9th International Symposium on High-Performance Computer Architecture*, 2003.

[96] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In *Proc. of the 5th International Conference on Parallel Problem Solving from Nature*, 1998.