Diss. ETH No. 16589

Efficient Design Space Exploration for Embedded Systems

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
Doctor of Sciences

presented by
SIMON KÜNZLI
Dipl. El.-Ing.,
Swiss Federal Institute of Technology Zürich, Switzerland
born August 4, 1975
citizen of Switzerland

accepted on the recommendation of
Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Luca Benini, co-examiner

2006

Examination date: April 20, 2006

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

TIK-SCHRIFTENREIHE NR. 81

Simon Künzli

Efficient Design Space Exploration for Embedded Systems

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A dissertation submitted to the
Swiss Federal Institute of Technology Zurich
for the degree of Doctor of Sciences

Diss. ETH No. 16589

Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Luca Benini, co-examiner

Examination date: April 20, 2006

Abstract

Design space exploration is an important factor in embedded systems design. During several steps in a state-of-the-art design flow, designers have to decide between many design alternatives. The decisions are located at various levels of abstraction. In addition, the choices affect several design goals, the alternatives therefore represent a multi-criteria decision problem.

Further, the space of possible solutions is normally very large, i.e., many design alternatives exist. As a consequence, exhaustive search of the design space is prohibitive, and more sophisticated techniques have to be used to find “good” solutions. To judge the quality of a new design, the performance of a system for a given application is one core criterion. Potential performance metrics are memory demand, response time, or data throughput of an application. As a consequence, one may ask the following questions:

- How can we assess the performance of a new design for a certain application?
- How do we find new design points in the design space?
- How can we automate the design space exploration process?

In this work, we investigate several aspects of design space exploration problems and try to answer these questions. In particular, we identify and discuss the building blocks for a design space exploration framework, namely design evaluation, search strategies, and design representation. Based on these building blocks, the main contributions of this work can be described as follows:

- A new hybrid method for performance evaluation of embedded systems is presented. The new method allows the combination of existing methods for performance analysis. In particular, these methods can be analytic or simulation-based. We provide the required interfaces for this combination.

- We describe a new evolutionary multi-objective optimisation algorithm, that directly incorporates the user's preferences based on performance indicators. It is easy to use and shows superior performance on test benchmarks and on design space exploration problems.

- A novel software framework for design space exploration is presented. Using the framework we can re-use existing software blocks and need to implement only a few components that represent the specific problem.

Kurzfassung

Die Erforschung des Entwurfsraums ist eine wichtige Tätigkeit im Rahmen des Entwurfs von Eingebetteten Systemen. Wenn Systeme nach aktuellen Entwurfsabläufen entwickelt werden, müssen die Entwickler immer wieder zwischen verschiedenen Entwurfsalternativen auswählen. Diese Entscheidungen werden auf verschiedenen Abstraktionsebenen getroffen. Zusätzlich haben diese Entscheide Auswirkungen auf verschiedene Entwurfsziele, die Auswahl der Entwurfsalternativen stellt deshalb ein Mehrziel-Optimierungsproblem dar.

Der Entwurfsraum der möglichen Lösungen ist normalerweise sehr gross, d.h. es existieren viele verschiedene Alternativen für den Entwurf. Als Konsequenz kann nicht jede Lösung im Suchraum untersucht werden, sondern es müssen geeignete Methoden angewendet werden, um in einem Teil des Suchraumes die "guten" Lösungen zu finden. Die Performanz einer solchen Lösung für eine gegebene Anwendung ist ein Kernkriterium, um die Qualität des Systems zu ermitteln. Mögliche Metriken zur Bestimmung der Performanz sind Speicherverbrauch, Antwortzeit oder Datendurchsatz einer Anwendung.

Als Folge dieser Überlegungen kann man sich die folgenden Fragen stellen:

- Wie kann man die Performanz eines neuen Entwurf für eine gegebene Anwendung bestimmen?
- Wie finden wir überhaupt eine neue Lösung im Entwurfsraum?
- Wie kann die Suche nach neuen Entwürfen automatisiert werden?

In dieser Arbeit untersuchen wir verschiedene Aspekte der Entwurfsraumsexploration und versuchen auf die obigen Fragen Antworten zu finden. Die einzelnen Bausteine für eine automatisierte Exploration, nämlich die Evaluation einzelner Lösungen, unterschiedliche Suchstrategien und die Repräsentation der Lösungen werden vorgestellt. Basierend auf diesen Bausteinen können wir nun die Hauptbeiträge dieser Arbeit präsentieren:

- Wir präsentieren einen neuen hybriden Ansatz zur Performanz-Analyse von Eingebetteten Systemen. Die neue Methode kombiniert existierende Verfahren, insbesondere Simulation und analytische Verfahren, und beschreibt auch die notwendigen Schnittstellen zwischen diesen.

- Ein neuer evolutionärer Algorithmus für die Mehrzieloptimierung wird vorgestellt, welcher Benutzerwünsche an die Suche, ausgedrückt durch Performanz-Indikatoren, direkt berücksichtigt. Der Algorithmus ist einfach zu benutzen und liefert für Probleme, bei welchen es um die Erforschung des Entwurfsraums geht, gute Resultate.

- Wir beschreiben ein Programm-Framework, welches für die Entwurfsraumsexploration verwendet werden kann. Mit dem Framework kann man schon bestehende Software-Blöcke wiederverwenden und muss nur wenige Komponenten, die das aktuelle Problem beschreiben, neu implementieren.

I would like to thank

- Prof. Dr. Lothar Thiele for being my advisor, letting me doing my own work very independently, but also always finding the time to have fruitful discussions if needed.
- Prof. Dr. Luca Benini for his agreement to be my co-examiner, for the interesting discussions on performance evaluation, and for the possibility to visit his lab in Bologna.
- Andreas Meier for always being ready to help and his moral support during the last weeks before submitting this thesis.
- Dr. Lennart Meier for the discussions about proper German language, hyphens, and all the other things that are to be discussed in a shared office.
- Prof. Dr. Samarjit Chakraborty for the collaboration and being my office mate in the first two years of my PhD studies. Starting to work with him was a very smooth dive into research, and also a great pleasure.
- all my colleagues at TIK with whom I had lots of interesting discussions, sometimes ending rather late and off-topic.

The work presented in this thesis was partly supported by the Innovation Promotion Agency CTI/KTI through project number 5500.2. This support is being gratefully acknowledged.

Contents

1 Introduction

An embedded system can be defined as information-processing system that is embedded into its environment. Embedded systems are mostly dedicated to a certain application domain where they are customised to fulfil special-purpose tasks. Examples for embedded systems are processors in cars, mobile phones, or even coffee machines.

Embedded systems are ubiquitous, there are surveys that report of more than 50 embedded processors in an average household in the year 2002 (e.g. in [MSB$^+$02]). Marwedel in [Mar03], and also Wolf in [Wol01] report that high-end cars even contain more than 100 processors at the beginning of the 21st century.

Designing an embedded system is a difficult task because the requirements change for every new system. The design flow has therefore to be flexible and generic enough to cope with the requirements and design constraints that a new design possibly imposes. The complexity of embedded system design is aggravated by the lack of generally accepted architectures, unlike in the domain of general purpose processors.

There are many different approaches to capture the embedded system design flow. In the next section, we will discuss a few examples and try to extract the principles common to all the methods.

1.1 Embedded System Design

Many design methods have been proposed and discussed over the last few decades. For software development, there is the waterfall design flow introduced by Royce [Boe88], in which the main design steps are performed one after the other from high abstraction levels to lower ones, starting from requirements, leading over architectural decisions and software integration, to testing and maintenance of the developed software. In this design model, there is only little feedback to the next-higher abstraction level. This top-down approach is ideal, but not realistic, because most design projects involve several revisions.

This behaviour is better captured with the spiral design flow [Boe88], where several versions of a system are built. Early versions are simple in nature, where later systems are more and more complex. On each design level, the design steps requirements, construction, and testing are performed. A possible drawback of the spiral model is that it may take too long, if too many spirals are performed (see [Wol01]). A design method that is based on a first prototype construction which is then improved in several iterations is called successive refinement design flow [Wol01]. The difference between these design flows lies in the point in time when design steps are taken at what level of accuracy, but not in the design steps itself.

Wolf identified 5 major steps embedded system design [Wol01]. These are (1) the definition of requirements that can be functional as well as non-functional, and (2) the specification of the system behaviour. The specification serves as contract between the designers and the customers. In step (3) the architecture design, the overall structure of the system is designed, then it is partitioned into component blocks for hardware and software parts. In this step, system-level design decisions are made, such as allocation and binding of software blocks to hardware resources. The work presented in this thesis helps the designer to make these decisions. Step (4) deals with the design of the individual hardware and software components, and finally the whole system is integrated and tested in step (5).

Ernst proposes three main types of design tasks which can be identified in all design flows for embedded systems in [Ern03]:

- component/subsystem interfacing
- system verification
- system optimisation and design space exploration

These steps can be identified in many design flows implemented by commercial or academic methodologies. Examples are the SpecC design flow, COSYMA, the IMEC tool flow, Ptolemy II, and OCTOPUS (see [Mar03] and references therein).

As an example, we look at the SpecC design flow. Starting from an executable specification in SpecC, the next step performed in the SpecC

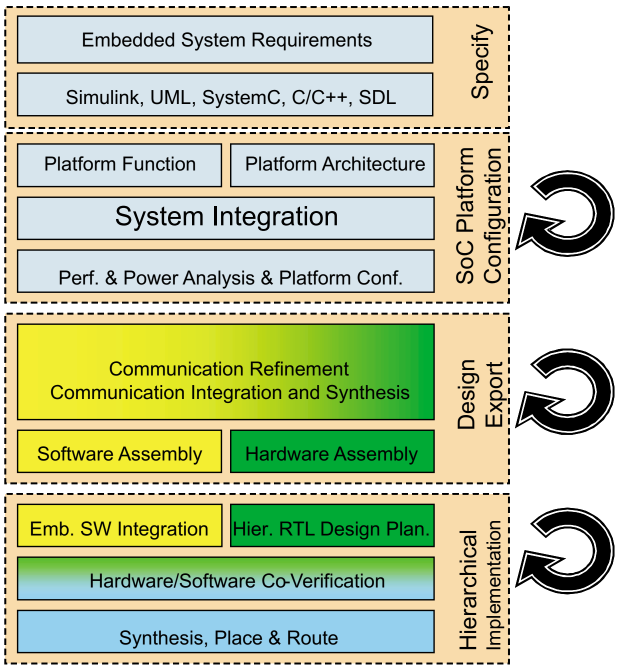

Embedded System Requirements
Simulink, UML, SystemC, C/C++, SDL
Specify
Platform Function
Platform Architecture
System Integration
Perf. & Power Analysis & Platform Conf.
SoC Platform Configuration
Communication Refinement
Communication Integration and Synthesis
Software Assembly
Hardware Assembly
Design Export
Emb. SW Integration
Hier. RTL Design Plan.
Hardware/Software Co-Verification
Synthesis, Place & Route
Hierarchical Implementation

Fig. 1: Graphical representation of the design flow as developed within the SpeAC project.

design flow [GZD+00] for embedded systems is an architecture exploration. After this step, a chosen architecture with allocation, partitioning and scheduling is validated.

The design flow proposed in the European project SpeAC is shown in Figure 1. The design flow tackles different views of a design, namely a module and a system view across various levels of abstraction. At the design specification, for instance UML, Matlab/Simulink, SDL, C/C++ can be used separately or combined within one design. Afterwards SystemC can be used for integrating heterogeneously specified modules into a single simulation model. In Figure 1, also the steps at lower abstraction levels are shown. In the design flow, refinement steps occur at all levels of abstraction. Here, too, there are design space exploration tasks to be solved, e.g. at the platform configuration step, during the communication refinement, but also at low levels of system synthesis and placement/routing.

As a result of this short survey, we can identify the design task of *design space exploration* common to all proposed embedded system design flows. This thesis exactly deals with issues related to this task and proposes new models and methods for design space exploration. They should assist a designer in finding a good design. The next chapter discusses the various aspects of design space exploration and introduces a reader into the topic. This thesis is based on the following journal publications, conference papers, and book chapters: [CKT+03b], [KTZ06], [KTZ05b], [KBTZ04], [CKT03a], [TCGK02a], [TCGK02b], [ZK04], [KPBT06], and [KT06].

1.2 Contributions

In this thesis, we present results that ease the task of design space exploration for embedded systems. The main contributions are presented in the following.

1. We describe a new, compositional performance evaluation method for embedded systems. The new method combines existing approaches for system-level performance analysis, namely a formal method and a simulation-based approach. To enable this combination, we define the interfaces needed between the different performance evaluation methods. As a core of the approach, we propose a method to generate simulation stimuli from analytical models. In addition, we introduce a measure to assess the quality of a generated simulation trace with respect to its analytical description. In order to show the applicability of this new approach for performance evaluation, we implemented an example system for such a combined performance evaluation consisting of a multiprocessor system-on-a-chip. It is based on existing models for simulation and analytical models extended by the needed interfaces for the combination, including an implementation of the simulation trace generation algorithm. This combined model was then used for a case study of an application running on a multiprocessor system.

2. A new evolutionary algorithm IBEA is presented that directly incorporates user preferences. Most popular evolutionary search algorithms incorporate methods to quickly find solutions close to the optimal solution (fast convergence), but also to keep diversity among the solutions. These methods are built-in and cannot be changed by the user. Our new algorithm is more generic in the sense that it allows the user to explicitly advise the algorithm what optimisation goal it should pursue. Furthermore, the new algorithm is easy to use, as it reduces the number of parameters that have to be set by a user compared to other existing black-box optimisation algorithms. This is achieved by an adaptive scaling such that the algorithm works independently of the objective values given to the algorithm. We discuss the performance of the new algorithm IBEA compared with many other state-of-the-art multi-objective optimisers on several test problems.

3. We present a generic software framework to perform design space exploration runs. Using the framework, only a few problem-specific parts have to be implemented to explore a certain design space, other parts can be reused. The tool implements the PISA interface such that many popular multi-objective optimisers can easily be reused without any implementation effort. Like this, the development time for a new design space exploration tool that matches a user's needs is shortened and the developers can concentrate on the problem-specific parts of the exploration tool. It offers the user a convenient graphical user interface and is completely written in Java. We used the framework for two example design space exploration problems, namely a cache optimisation example and for packet processor architectures. The former was used to introduce the framework and to discuss the main components of a design space exploration problem. The packet processor example is more involved and was used as benchmark application to evaluate the performance of different randomised search algorithms for a real-world application.

1.3 Overview

In the following, we give an overview over the contents of this thesis:

- **Chapter 2** gives an overview over the various topics that are involved in the process of design space exploration. We give a classification of existing approaches to design space exploration based on the abstraction level and the exploration method used.

- **Chapter 3** covers performance evaluation methods for single design points in the space of possible designs. We investigate a formal method for system-level performance analysis, show how it can be adapted to be used for design space exploration. We then compare the formal method with a simulation-based approach and finally propose a new performance evaluation method that combines simulation with the formal method.

- **Chapter 4** discusses the different search strategies that exist for multi-objective optimisation problems. Further we propose a new evolutionary algorithm that directly incorporates user preferences. The performance of the new search algorithm is compared to other established algorithms in a case study.

- **Chapter 5** describes tools and applications of the work presented in the previous chapters. It gives an overview over the EXPO tool framework, a generic framework for design space exploration that was used for several exploration problems presented in this thesis. Further, the tool was used as a benchmark problem for evolutionary algorithms and the results of a case study are also presented in this chapter.

- **Chapter 6** concludes the thesis with an outlook for future research and a summary of the contributions.

2

Design Space Exploration of Embedded Systems

Design space exploration is a central step in embedded system design [Ern98]. There may be many different design alternatives that implement a given system specification. These different implementations have to be explored and judged for their quality such that a designer can make a decision which system alternative to implement. Therefore, during the exploration phase, many design alternatives have to be evaluated. Design alternatives may consist of different hardware component allocations, different mappings of software tasks to resources, different scheduling policies implemented on shared resources as well as lower level design parameters such as clock frequency or bus widths.

In actual design flows presented by Marwedel in [Mar03], design space exploration can be found embedded into the design process. For example, in the SpecC design flow [GZD$^+$00], design space exploration is performed after having an executable specification to find an appropriate allocation, partitioning and scheduling. After the exploration, in the SpecC design flow design validation is performed.

The IMEC tool flow also involves design space exploration at several stages as part of the design flow. During the exploration phase, the mapping of tasks to processors is optimised [WMY$^+$01] as well as data transfer and data storage [CdGS98].

Design space exploration is often a problem involving multiple criteria. The design alternatives normally represent a trade-off between different optimisation goals. For instance, if we consider cost and performance as optimisation goals, a processor that can run at higher clock speeds is usually more expensive.

In the following sections, we will introduce a simple example for design space exploration that will guide us through the remainder of this chapter, categorise existing approaches to design space exploration according to their abstraction layer, and cover the different aspects that play a role in the design space exploration process for embedded systems.

2.1 A Simple Example: Design Space Exploration of Cache Architectures

Before we describe the components needed for design space exploration in detail, let us consider a simple example application – the design of a cache subsystem – that will be used throughout the remainder of this chapter for illustration purposes. Note, that it is not the purpose of the example to present any new results in cache optimisation.

Suppose we want to optimise the architecture of a cache for a predefined benchmark application. Restricting ourselves to Level 1 data caches only, the design choices include the cache size, the associativity level, the block size, and the replacement strategy. The goal is to identify a cache architecture that (1) maximises the overall computing performance with respect to the benchmark under consideration and (2) minimises the chip area needed to implement the cache in silicon.

Nr.	Parameter	Range
1	# of cache lines	2^k, with $k = 6 \ldots 14$
2	Block size	2^k Bytes, with $k = 3 \ldots 7$
3	Associativity	2^k, with $k = 0 \ldots 5$
4	Replacement strategy	LRU or FIFO

Tab. 1: Parameters determining a cache architecture.

In Table 1, all parameters and possible values for the cache architecture are given. A design point is therefore determined by three integer values and a Boolean value. The integers denote the number of cache lines, the cache block size and the cache associativity; the Boolean value encodes the replacement strategy: *false* denotes FIFO (first-in-first-out),

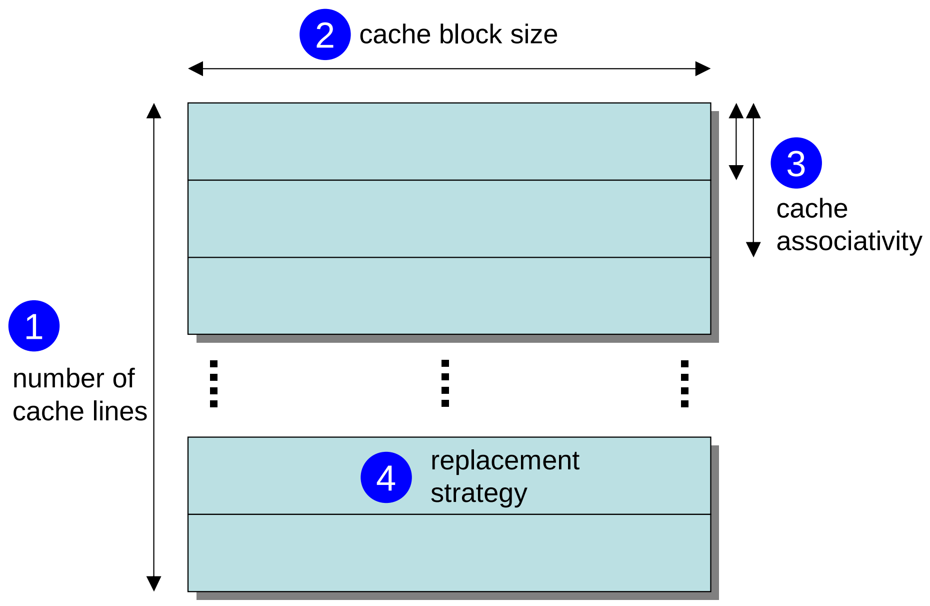


Fig. 2: Illustration of the considered design choices for an L1 data cache architecture.

true denotes LRU (least recently used). Figure 2 graphically depicts the design parameters. The values for the number of cache lines, block size and associativity have to be powers of 2, due to restrictions in the tools used for evaluation of the caches.

The first objective according to which the cache parameters are to be optimised is the CPI (cycles per instruction) achieved for a sample benchmark application, and the second objective is the chip area needed to implement the cache on silicon. To estimate the corresponding objective values, we used two tools, namely `sim-outorder` of SimpleScalar [BA97] and CACTI [SJ01] provided by Compaq. The first tool served to estimate the CPI for the benchmark compress95 running on the plain text version of the GNU public license as application workload. The smaller the CPI for compress95 for a particular solution, the better is this solution for this objective. The second tool calculated an estimate for the silicon area needed to implement the cache. The smaller the area, the better is the cache for the area objective.

The cache subsystem design space exploration example was performed using the EXPO tool framework described in Chapter 5 of this thesis. We used the randomised search algorithm SPEA2 [ZLT02] to perform the optimisation.

The design space with all solutions is shown in Figure 3. These design points have been generated using exhaustive search in order to compare the heuristic search with the Pareto front of optimal solutions. The Pareto

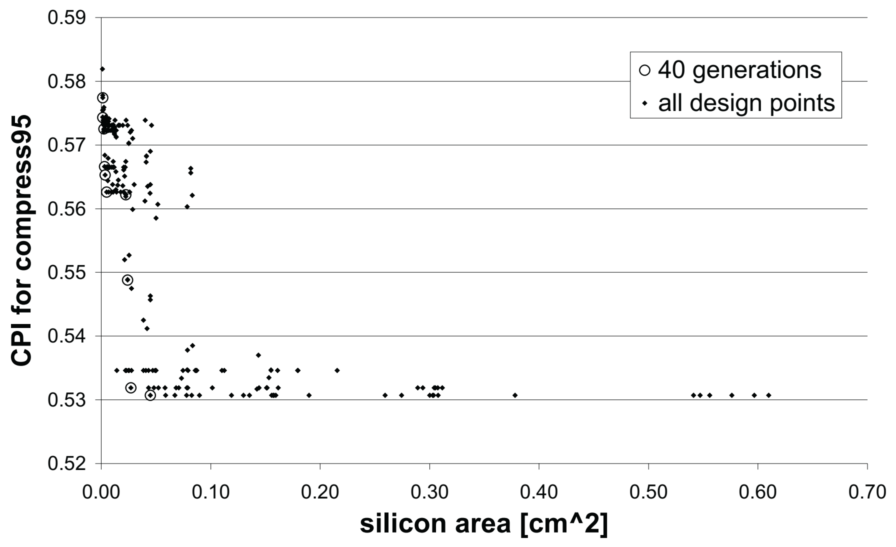


Fig. 3: All 540 possible design points determined using exhaustive search and the design points found by the multi-objective search algorithm SPEA2 after 40 generations.

front denotes the set of solutions that are Pareto-optimal, i.e. the solutions that are not dominated by any other solution in the solution set. A solution *A* dominates another solution *B*, if *A* is at least as good as *B* in all criteria and strictly better in at least one criterion.

The front of non-dominated solutions found for the cache example with SPEA2 after a typical optimisation run with 40 generation for a population size of 6 solutions is marked with circles. The details of the solutions in the population after 40 generations are given in Table 2.

Although the cache design space exploration problem is simple in nature, one can make some observations which also hold for more involved exploration problems. The two objectives, namely the minimisation of the silicon area and the minimisation of the CPI, are conflicting, resulting in an area vs. performance trade-off. This results in the fact that there is not a single optimal solution, but a front of Pareto-optimal solutions. All points on this front represent different promising designs, leaving the final choice for the design of the cache up to the designer's preference.

The reduction of the problem to a single-objective optimisation problem, e.g., using a weighted-sum approach is difficult already for this simple example, because it represents a true multi-objective problem. It is not at all clear how to relate area to performance, which would be needed for the weighted-sum approach.

Figure 4 shows how the design space exploration is embedded into a

No.	CPI	Area	Design Parameters			
1	0.5774	0.001311	LRU	2^7 cache lines	block size 8	d. m.
2	0.5743	0.001362	LRU	2^7 cache lines	block size 8	2 sets
3	0.5622	0.022509	FIFO	2^8 cache lines	block size 64	8 sets
4	0.5725	0.002344	LRU	2^7 cache lines	block size 16	2 sets
5	0.5488	0.024018	LRU	2^{10} cache lines	block size 32	8 sets
6	0.5319	0.027122	LRU	2^{10} cache lines	block size 32	16 sets
7	0.5666	0.002898	LRU	2^6 cache lines	block size 32	2 sets
8	0.5653	0.003629	FIFO	2^6 cache lines	block size 64	d. m.
9	0.5307	0.044902	FIFO	2^{10} cache lines	block size 64	8 sets
10	0.5626	0.004907	LRU	2^6 cache lines	block size 64	2 sets

Tab. 2: Details of 10 non-dominated solutions for the simple example of a cache exploration found after a typical design space exploration run. These solutions are marked with circles in Figure 3.

hierarchical design trajectory for embedded systems. If we consider our example to be on the current abstraction level, the evaluation of a solution leads to requirements for sub-components on the lower level, e.g. the area on silicon that is available for the implementation in our example. In addition, the evaluation provides properties of the component to the higher level of abstraction. The various abstraction layers for design space exploration are discussed in the next section. In the later sections, we use the cache example to introduce the building blocks for design space exploration.

2.2 Abstraction Layers in Design Space Exploration

There are many existing approaches that make use of an automated or semi-automated design space exploration in embedded systems design. Exploration of implementation alternatives happens at various levels of abstraction in the design. These various layers are described next and existing design space exploration approaches are classified accordingly:

- *Logic Design and High Level Synthesis*: Here, one is concerned with the synthesis of digital logic starting from either a register-transfer specification or a more general imperative program. The manual design of dedicated computing units is also included. Typical design choices concern speed vs. implementation area vs. energy consumption, see e.g. [BBB01, CSH00].

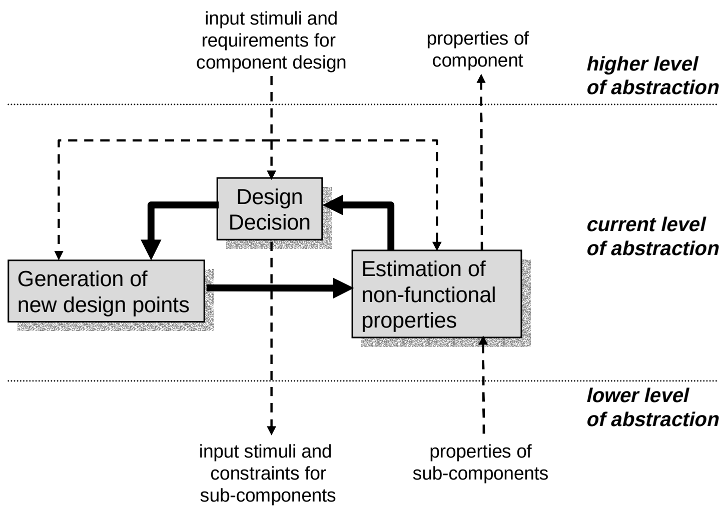

input stimuli and requirements for component design
properties of component
higher level of abstraction
Design Decision
Generation of new design points
Estimation of non-functional properties
current level of abstraction
lower level of abstraction
input stimuli and constraints for sub-components
properties of sub-components

Fig. 4: Embedding of exploration in a hierarchical design trajectory for embedded systems.

- *Programmable Architecture*: The programmable architecture layer contains all aspects below the instruction set. For example, it contains the instruction set definition, the microprocessor architecture in terms of instruction level parallelism, the cache and memory structures. There are numerous examples of exploration on this level of abstraction; they concern different aspects such as caches and memories [GG03, SC99, SCK04], or the whole processor architecture, especially the functional unit selection [HHBS99, PSZ03, RCR04].

- *Software Compilation*: This layer comprises all ingredients of the software development process for a single task, such as code synthesis from a model-based design or a high-level program specification. Within the corresponding compiler, possible exploration tasks are code size vs. execution speed vs. energy consumption. There are attempts to perform a cross-layer exploration with the underlying processor architecture, see e.g. [ZTB00b, APS04].

- *Task Level*: On the task level, the whole application is partitioned into tasks and threads. Therefore, the task level refers to operating system issues like scheduling, memory management and arbitration of shared resources. Typical trade-offs in choosing the scheduling and arbitration methods are energy consumption vs. average case vs. worst case timing behaviour, see e.g. [BBTZ01].

- *Distributed Operation*: Finally, there exist applications that run on distributed resources. The corresponding layer contains the hardware aspects of distributed operation (such as the design of communication networks) as well as methods of distributed scheduling and arbitration. On this level of abstraction, which is sometimes called system level, one is interested in the composition of the whole system that consists of various computing and communication resources. System-level design not only refers to the structure of the system, but also involves the mapping of application to the architecture and the necessary (distributed) scheduling and arbitration methods. This highest level of abstraction seems to be especially suited for exploration methods, see e.g. results on the communication infrastructure [LRD04, ETZ00], on distributed systems [ABD$^+$04] or multiprocessor systems and systems-on-chip, e.g. [ARS98, GVNG98, GVH02, BTT98, TCGK02a].

The above approaches combine several important aspects such as the integration of the exploration into the whole design process, the specific estimation method used to evaluate the properties of design points and finally the method that is used to perform the actual exploration.

2.3 Design Evaluation

There are a lot of different criteria for which one could try to optimise an embedded system. Besides the “classic” non-functional properties, such as power consumption, silicon area or resource utilisation, the implementation cost or manufacturability are also examples for design criteria.

Common to all the criteria is the need for a tool to evaluate a design point in the design space. Based on the representation of the solution such a tool captures the quality of a design (cf. Figure 5). The quality of a property can normally be expressed using a number. In that case, two design points can be compared for this property and a designer can immediately see, which one of the two design points is better with respect to the design criterion in question, based on this number.

Dependent on the level of abstraction, the evaluation tools are based on formal methods, simulation or even measurements. One requirement for the design evaluation of a single design point in design space exploration is evaluation time. If the evaluation for a single design point takes too much time, the method is prohibitive for exploring a huge design space. Nevertheless, it may be possible to use the evaluation method in a later phase for a pruned design space.

Performance evaluation of embedded systems is covered in more detail in Chapter 3. A formal performance evaluation method is introduced and compared with an existing established performance evaluation

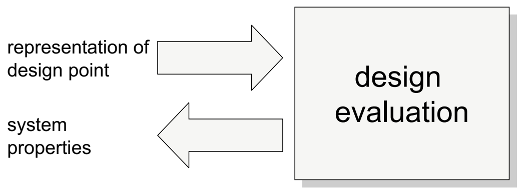


Fig. 5: A design evaluation method takes the representation of a design point as input and provides the system properties as output.

method, namely simulation. We show that in order to speed up the evaluation time, we can use approximations for the formal method. Further, we propose a new evaluation method that combines simulation-based approaches with analytical methods.

For the cache example introduced in Section 2.1 we use two different tools to assess the performance number for a given solution. On one hand side, we use SimpleScalar [BA97], an instruction set simulator to retrieve the cycles per instruction (CPI) for a benchmark application. The evaluation of the cache architecture for CPI is therefore based on simulation. On the other hand, to obtain the area in silicon needed to implement the cache we use CACTI [SJ01], a tool that is based on formal analysis.

2.4 Exploration Method

The existing approaches for design space exploration can also be classified in a way that is orthogonal to the abstraction layers, namely the methods that are applied to perform the exploration itself. This way it becomes apparent that the exploration process is largely independent of the abstraction level.

If only a single objective needs to be taken into account in optimisation, the design points are totally ordered by their objective value. Therefore, there is a single optimal design (if all have different objective values). The situation is different if multiple objectives are involved. In this case, design points are only partially ordered, i.e. there is a set of incomparable, optimal solutions. They reflect the trade-offs in the design. Optimality in this case is usually defined using the concept of Pareto-dominance: A design point dominates another one if it is equal or better in all criteria and strictly better in at least one. In a set of design points, those are called Pareto-optimal which are not dominated by any other. Using this notion, available approaches to the exploration of design spaces can be characterised as follows:

1. *Exploration by hand*: The selection of design points is done by the designer himself. The major focus is on efficient estimation of the selected designs, e.g. [GVNG98].

2. *Exhaustive Search*: All design points in a specified region of the design parameters are evaluated. Very often, this approach is combined with local optimisation in one or several design parameters in order to reduce the size of the design space, see e.g. [SC99, ZSXS03].

3. *Reduction to a Single Objective*: For design space exploration with multiple conflicting criteria, there are several approaches available that reduce the problem to a set of single criterion problems. To this end, manual or exhaustive sampling is done in one (or several) directions of the search space and a constraint optimisation, e.g. iterative improvement or analytic methods is done in the other, see e.g. [CSH00, GG03, LRD04, RCR04].

4. *Black-box Randomised Search*: The design space is sampled and searched via a black-box optimisation approach, i.e. new design points are generated based on the information gathered so far and by defining an appropriate neighbourhood function (variation operator). The properties of these new design points are estimated which increases the available information about the design space. Examples of sampling and search strategies used are Pareto Simulated Annealing [CJ98] and Pareto Tabu Search, e.g. [APS04, PSZ03], evolutionary multi-objective optimisation [ABD$^+$04, BTT98, ETZ00, TCGK02b], or Monte Carlo methods improved by statistical estimation of bounds, e.g. [BBB01]. These black box optimisations are often combined with local search methods that optimise certain design parameters or structures, e.g. [BBTZ01].

5. *Problem-dependent Approaches*: In addition to the above classification, one can find also a close integration of the exploration with a problem-dependent characterisation of the design space. Several possibilities have been investigated so far:

   - Use the parameter independence in order to prune the design space, e.g. [GVH02, PG02].
   - Restrict the search to promising regions of design space, e.g. [HHBS99].
   - Investigate the structure of the Pareto-optimal set of design points, for example using hierarchical composition of sub-component exploration and filtering [ARS98, SCK04].

- Explicitly model the design space, use an appropriate abstraction, derive a formal characterisation by symbolic techniques and use pruning techniques, e.g. [NSK02].

Finally, usually an exhaustive search or a black-box randomised search is carried out for those parts of the optimisation that are inaccessible for tailored techniques.

From the above classification, one can state that most of the above approaches use randomised search techniques one way or the other, at least for the solution of subproblems. This observation does not hold for the exploration by hand or the exhaustive search, but these methods are only feasible for small design spaces with a few choices of the design parameters. Even in case of a reduction to a single objective or in the case of problem-dependent approaches, sub-optimisation tasks need to be solved, either single objective or multi-objective and randomised (black-box) search techniques are applied.

2.5 Summary

In this chapter, we introduced design space exploration as a central step in embedded system design. The main building blocks for a design space exploration are the problem specification, the design representation, the evaluation of a single design point and the exploration method. We revised existing approaches and classified them according to the abstraction layer and to the exploration method used.

Based on the concepts introduced in this chapter, the remainder of this thesis further investigates issues related to design space exploration. In the following, Chapter 3 is dedicated to performance evaluation of single design points. In Chapter 4, we will introduce evolutionary algorithms as exploration method. The performance of different algorithms is investigated and a new evolutionary algorithm is proposed. In Chapter 5, we present a tool framework for design space exploration and applications thereof.

3

Design Evaluation

The evaluation of design points is one of the building blocks for design space exploration of embedded systems. There are many different properties of an embedded system a designer can be interested in. In [VG01], the authors present a long list of such potential design metrics, including power consumption, size, unit cost and performance for given applications.

In this chapter, we look at performance analysis techniques for embedded systems. First, we discuss existing approaches in Section 3.2. We then introduce Real-Time Calculus as formal method to assess performance numbers of a system in Section 3.3. Next, we briefly introduce simulation-based approaches for performance evaluation. These simulation-based approaches are then used for a comparative study with Real-Time Calculus.

After revising these existing approaches and discussing their use for design space exploration, we present the main results achieved in this work that are related to performance evaluation. Namely, we introduce a new approach to performance analysis, which combines Real-Time Calculus with a simulation-based approach in Section 3.6. Further, the necessary interfaces for this combination are described and we present a case study based on a multiprocessor platform. In the last section of this chapter, we look more carefully at the interface needed to couple Real-Time Calculus with simulation, and propose a generator for simulation traces that comply with a formal load specification.

Throughout this chapter, we use two example embedded systems. The example system 1 is a hypothetical network processor architecture which is built around a PowerPC processor. The example system 2 consists of an architecture with multiple ARM cores connected through a parameterisable interconnection network. Both example systems were modelled with analytical techniques as well as for simulation and are introduced in the next section.

3.1 Example Systems

3.1.1 Network Processor

Figure 6 shows a hypothetical network processor architecture built out of blocks from an existing core library [IBMa, IBMb]. In the figure, PPC refers to the PowerPC 440 core, and PLB and OPB refer to two buses called the Processor Local Bus and the On-chip Peripheral Bus provided by the CoreConnect [IBMb] architecture for interconnecting cores and custom logic. For our example, we assume a simple network packet forwarding application running on this architecture. The numbers on the arrows in this figure either indicate actions that are to be performed by the different blocks as a packet flows through the architecture, and they are ordered according to the numbering. The system is described in more detail in [Wor01].

From Figure 6 it is possible to construct a task graph considering the appropriate packet transfers from one resource to another. This task graph together with a model for the hardware resources can then be used for formal analysis as we will see later in this chapter. With this analysis we can compute the load on the different buses (such as the OPB and the PLB), the on-chip memory requirement of this architecture to store the buffered packets in front of each resource, and the end-to-end packet delays.

3.1.2 Multiprocessor Platform MPARM

MPARM is a multi-processor virtual platform [LAB+04]. Its purpose is the system-level analysis of design tradeoffs in the usage of different processors, interconnects, memory hierarchies and other devices.

It consists of a parameterisable number of ARM7 processor cores and an AMBA interconnection network [AMB]. The platform includes several memory devices, which can be used as private or shared memories (cf. Figure 7).

For the examples presented in this chapter, we investigated 2 different applications running on this system architecture. First, we looked at a pipelined matrix multiplication application with 8 pipeline stages

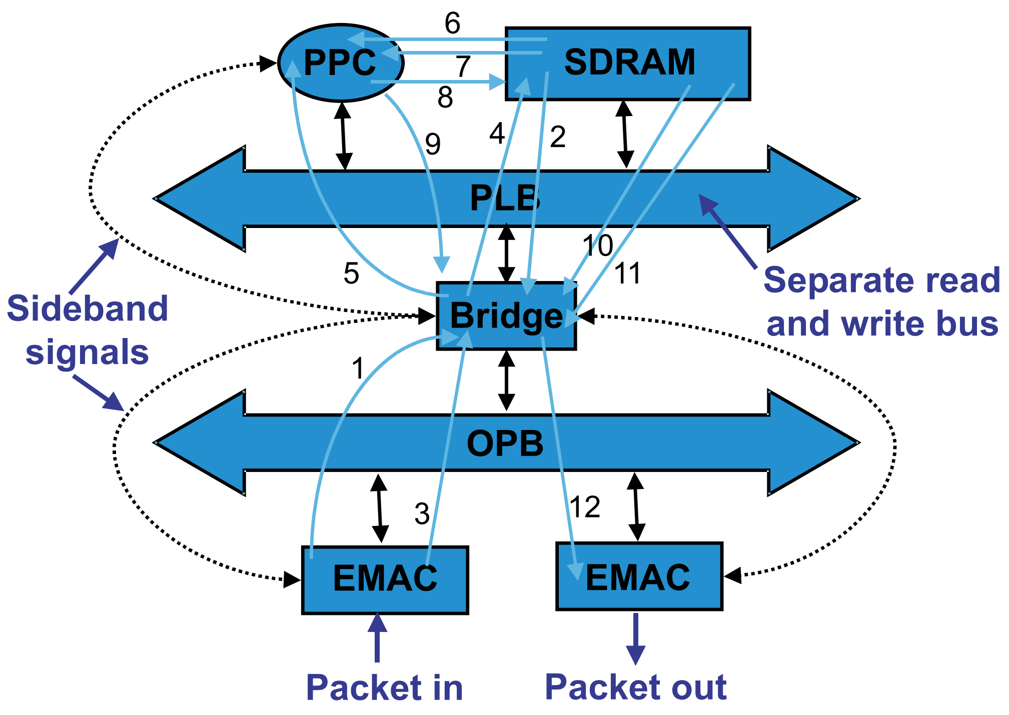


Fig. 6: A system-level model of a network processor. The figure shows the path that a packet follows through the architecture. The numbers on the arrows indicate the different actions involved (which are explained in Table 3) while the packet travels through the architecture, and specify the order in which these actions are executed.

mapped on 4 processor cores. We investigated 2 different scenarios with a different mapping of the tasks to the computing resources as shown in Figure 8. In one mapping, we tried to balance the load among the processors, for the other mapping, the goal was to minimise the communication needed between the processors.

Second, we analysed a GSM encoder application mapped to 2 processors. The application was partitioned into 2 tasks and the tasks communicate through a FIFO queue located in the shared memory. This second example will be used for the case study of the new hybrid approach for performance analysis in Section 3.6.

3.2 Performance Analysis

Recently, many processing devices for embedded systems are designed as systems-on-chip (SoC) [Wol01]. Using this design paradigm, a complete

Step	Action
1	Sideband signal from EMAC to bridge (indicating that a new packet has arrived)
2	Bridge gets a ”buffer descriptor” (BD) from the SDRAM
3	Packet is sent from EMAC to bridge over the OPB
4	Packet is sent from bridge to SDRAM over the PLB write bus
5	Sideband signal from Bridge to PPC (indicating that the new packet has been stored)
6	CPU get buffer descriptor over the PLB read bus
7	CPU gets packet header over the PLB read bus
8	CPU processes header, and stores it back to SDRAM over the PLB write bus
9	Sideband signal from bridge to CPU (indicating that the packet can be sent out)
10	Bridge gets buffer descriptor over the PLB read bus
11	Bridge gets packet over the PLB read bus
12	Packet sent out to specified a EMAC over the OPB

Tab. 3: Sequence of actions for every processed packet in the architecture model shown in Figure 6.

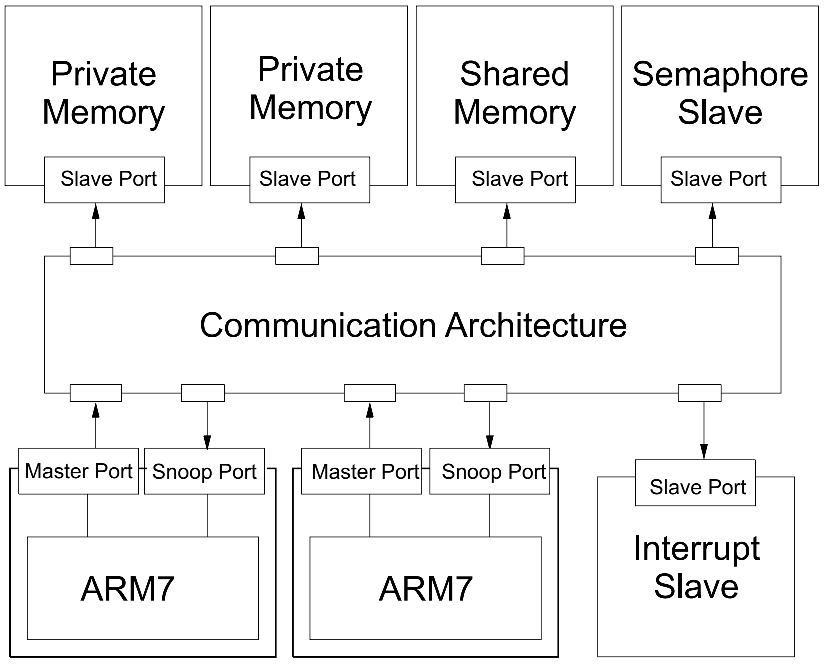


Fig. 7: MPARM platform architecture.

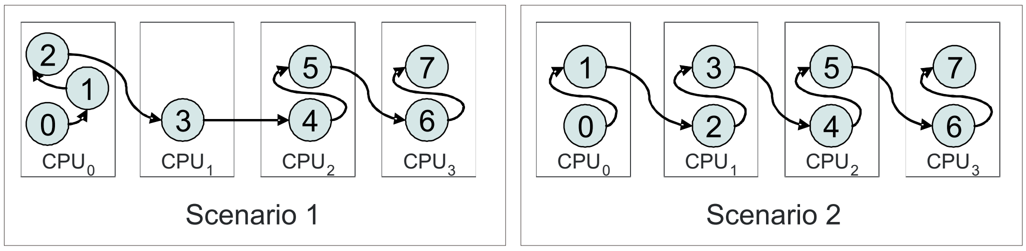


Fig. 8: Mapping of the matrix multiplication tasks T0–T7 to the 4 processors for the two scenarios discussed in the example.

system consisting of computing, storage and communication resources is integrated on the same chip. Such a system may consist of several IP cores and dedicated hardware, as the Cell processor announced recently by Sony, IBM and Toshiba [PAB+05].

The complexity of these SoC designs, coupled with issues like short time-to-market and low cost, have led to new design paradigms such as platform-based design [KMN+00]. These are based on the concept of *reuse* at several levels of abstraction, where designers rely on the use of intellectual property blocks or cores from some library (such as the IBM Blue Logic Core Library [IBMa]), or on cores provided by a third-party vendor. Since such cores are already pre-designed and verified, a designer can now concentrate on the overall system rather than the individual components, and also reduce the number of steps required to translate a system-level design into a final product.

Analysing such system platforms to verify timing and other system properties pose a major challenge because they depend on the interfaces and properties (such as arbitration schemes on buses) of the different cores, and also on the RTOS and other components of the software platform. The problem gets aggravated in the context of embedded systems because of their generally heterogeneous architecture, where different scheduling and resource sharing strategies are used on the different buses and processors.

Performance evaluation of embedded systems can be broadly divided in two main areas: simulation-based approaches and formal methods. Most of the existing approaches rely on simulation (for example VCC [VCC] and Seamless [Sea]), and hence suffer from the problems of high running time, incomplete coverage and failure to identify corner cases. The last two problems are aggravated by the fact that in many cases system integrators do not have a full understanding of the functionality and the interfaces of the different cores, but only understand their high-level input/output behaviour. Therefore, if guarantees on system properties are required then some form of static formal analysis is inevitable.

The trend for simulation-based performance analysis goes to full system simulation. Tools as Mentor Graphics' Seamless [Sea] support the co-simulation of complete hardware-software systems. In order to cope with the high simulation times caused by the increased complexity of the designed systems, simulation is used also on higher levels of abstraction in early design phases.

In [CB02] a modelling framework is presented which is composed of independent application, system and traffic models. The application is modelled using the Click modular router from MIT [KMC+00]. Click consists of a collection of software modules for describing various router functionality. Such modules in Click are called *elements*, and by putting together different elements in the form of a graph (which is called a *configuration*) it is possible to construct IP routers, firewalls, QoS routers, etc.

The framework in [CB02] is based on compiling Click modules for the Alpha ISA [Alp92]. The architecture to be evaluated is simulated using SimpleScalar [BA97] and it implements the Alpha instruction set. The compiled Click modules are then executed on this architecture. By simulating this execution using different traffic traces, the profiled code yields various information such as instruction count, details regarding cache behaviour, etc. These are then used to compute various performance metrics for the architecture being evaluated, related to packet latency, bandwidth and resource utilisation. For elements which do not have any software implementation (such as dedicated hardware units for header parsing) and can not be simulated, the profile and external dependencies need to be provided manually by the user.

In contrast to this approach, the work done in [Wor01] models an architecture in SystemC [GLMS02]. This work mostly focuses on the communication subsystem and the memory organisation of an architecture. The models are then simulated on packet traces and performance metrics such as bus utilisation, memory fill levels, and packet delays are evaluated. This work forms the basis of the simulation results that we use in this chapter and further details on it are given in Section 3.4.

The MPSoC simulation platform described in [MAS+05] combines a cycle-accurate simulation of a parameterisable communication infrastructure with instruction set simulators for the processing elements and therefore represents a framework that allows us to simulate complete embedded systems consisting of software and hardware. In contrast to the approaches presented in [Wor01] and [CB02] this framework can be used to evaluate both communication and computation parts of an embedded system.

Besides the simulation-based performance evaluation methods, formal methods are emerging that enable the analysis of whole systems using holistic [PEP05] and compositional approaches. In particular, the system can be analysed using models of the individual components that can be later composed to system models that capture the complete system [RJE03, HHJ+05].

Especially for the domain of network processors, which is the application domain for the first example used in this chapter, an analytical performance model is considered in [FW02]. Here the different components that make up the architecture, and the interconnection among these components (the so called *architecture template*) is fixed. The design decisions that are to be made in deriving a concrete architecture from such a template, consist of choosing the values of the various parameters such as the bus width, cache sizes, etc. The architecture considered consist of a number of multi-threaded processors organised in clusters. Each cluster consists of a number of processors, each having its own cache, and the cluster communicates with an off-chip memory using its own memory interface. The parameters that can be changed are the number of threads running in each processor, the cache sizes, the number of processors in each cluster, the number of clusters in the network processor, the width of the memory channels, etc. For evaluating a set of parameters, an analytical model for multi-threaded processors proposed by Agarwal [Aga92] is used. Most of this work can be viewed as a model for the cache/memory subsystem of a network processor architecture. The analytical model is then evaluated on a benchmark workload [WF00] consisting of a mix of header-processing and payload-processing applications. For each application, properties such as load and store instruction frequencies and instruction and data cache miss rates are measured using processor and cache simulators. These values are then used to evaluate an architecture in terms of its processing power per unit chip area.

A general approach to timing analysis for heterogeneous systems was presented in [RE02] and [RZJE02]. It is based on identifying architectural components for which analysis methods are already known in the literature, and then combining these to obtain a compositional description of the complex system-level timing behaviour. The main contribution of this work is a method to adapt outgoing event streams from one component to match the input event model of another component which is required to process this outgoing stream. This gives a means for formally composing different architectural components and reasoning about the behaviour of the entire system.

The main drawback of this approach is that it can only accommodate standard event models like purely periodic, periodic with jitter, periodic with bursts, and sporadic. In practice, the event streams involved in a system usually do not conform to any of these standard models. But while analysing such systems, these streams are approximated by some standard model which minimises the error. This introduces several modelling complexities, and when worst case bounds for a system are required, such approximations using standard event models give overly conservative bounds. These analytical methods are embedded into the SymTA/S tool, described in [HHJ+05].

The analytical model [TCGK02a, TCGK02b] that is discussed in more detail in Section 3.3 and used for the comparative study in Section 3.5 uses a model for both the architecture and the traffic traces on which the architecture is evaluated. In contrast to the work in [FW02] the architecture layout or the topology in this case is not fixed. Therefore, different combinations of processors, buses and their interconnection can be modelled and evaluated.

Actually, there is no sharp division into simulation-based approaches and formal methods for system-level analysis. There exist approaches that abstract communication internals in system simulation and use transaction-level modelling in SystemC [CG03]. Lahiri *et al.* present a combined approach to communication analysis which uses simulation for parameter extraction and then a formal method for fast performance analysis [LRD04]. Bobrek *et al.* also combine simulation with an analytical method in [BPN+04], with focus on the analysis of shared resource contention. They simulate parallel execution of threads and record accesses to shared resources, while a formal analysis model is then used to determine the adjustment of the timing caused by the shared resource contention.

The formal methods used for design space exploration throughout this thesis is introduced in Section 3.3. In Section 3.4, we will revise existing simulation-based approaches that were used for the case studies presented in this thesis. Results obtained by these simulation-based approaches are compared with the results obtained by the system-level formal performance analysis model in Section 3.5. For a realistic network processor architecture, we consider performance metrics as the line speed or the end-to-end throughput that can be supported by the architecture, or the on-chip cache/memory requirement of the architecture. Finally, we present a new approach to performance analysis in Section 3.6 that is inspired by the idea of core-based design and represents a hybrid approach consisting of simulation-based and formal analysis components. The hybrid approach is applied in a case study for multiprocessor SoC performance evaluation. The automated way the interfaces between the different domains are provided makes this new approach also suitable for design space exploration.

3.3 Formal Performance Analysis

In this section, we introduce Real-Time Calculus as a framework for formal performance analysis of embedded systems.

3.3.1 Event Models and Resource Capabilities

We describe the event model which forms the basis of the formal performance analysis framework presented in this section and also a means of modelling the processing capability of resources which process event streams. It may be noted that in contrast to previous work on the performance evaluation of distributed embedded systems which relies on statistical bounds (see for example [KM98]) we are interested in analysing architectures and deriving worst-case bounds on system properties like response times, on-chip memory requirements and loads on various components. We also show how standard event models such as periodic or periodic with jitter can be represented by our event model.

Event models. For a given event stream, let $R(t)$ denote the number of events that arrive in the time interval $[0, t]$. Further, assume that the number of events arriving within any interval of time is bounded above by a right-continuous sub-additive function called the *upper arrival curve*, denoted by α^u. Similarly, a lower bound on the number of events arriving is given by a *lower arrival curve* α^l. R, α^u and α^l are related by the following inequality:

$$\alpha^l(t-s) \le R(t) - R(s) \le \alpha^u(t-s), \ \forall 0 \le s \le t \tag{3.1}$$

Therefore, $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ can be interpreted as the minimum and maximum number of events arriving within *any* time interval of length Δ. Any standard event model can be represented in our model by an appropriate choice of α^l and α^u. For example, a periodic event stream with period p can be represented by an α^l and α^u, both of which are staircase functions of step width p and height 1, with $\alpha^l(t) = 0$ for all $t < p$ and $\alpha^u(t) = 1$ for all $t < p$. This is because within any time interval of length less than p, the minimum number of events that can be seen is zero, and within any time interval of length p^+, the minimum number of events that can be seen is equal to one. Similarly, the maximum number of events that can be seen within any time interval of length p and p^+ is one and two respectively. See Figure 9(a) for a graphical representation of the arrival curves for a periodic event stream.

Following the same reasoning, the class of event streams with period p and jitter j can be represented by an upper and a lower arrival curve of the form shown in Figure 9(b). Given any particular instance of such a periodic with jitter event stream, the corresponding upper and lower arrival curves would lie within the arrival curves shown in Figure 9(b), and therefore these curves represent the upper and lower bounds on the maximum and minimum number of events that can arrive within any time interval for any event stream with period p and jitter j.

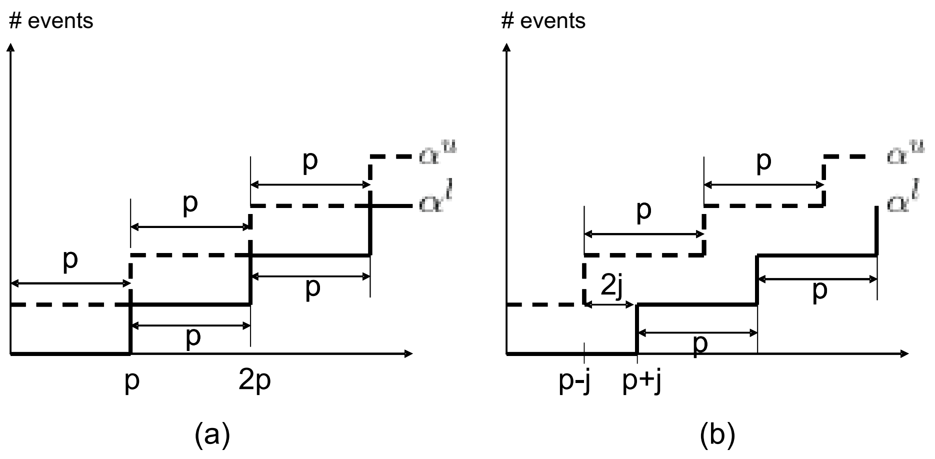


Fig. 9: **(a)** Upper and lower arrival curves describing a purely periodic event stream with period p. **(b)** Upper and lower arrival curves of the class of event streams with period p and jitter j.

At the same time, given any finite length arbitrary event trace and a real number Δ, it is possible to determine the values of $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ corresponding to the event trace, by sliding a window of length Δ over the trace and recording the minimum and maximum number of events lying within the window respectively. The upper and the lower arrival curves corresponding to the trace can be determined by following this procedure for different values of Δ. The sliding window procedure is described in more detail in Section 3.6 and is used for the combination of the different approaches.

Processing capability. Similar to the upper and lower arrival curves, we use β^u and β^l to denote upper and lower *service curves* of a resource with the following interpretation. If $C(t)$ denotes the number of processing units (might be in terms of processor cycles, time units, etc.) available from the resource over the time interval $[0, t]$, then the following inequality holds.

$$\beta^l(t-s) \leq C(t) - C(s) \leq \beta^u(t-s), \ \forall 0 \leq s \leq t. \tag{3.2}$$

Hence, $\beta^u(\Delta)$ and $\beta^l(\Delta)$ give an upper and lower bound on the resource capability over any time interval of length Δ.

There exist several possibilities to obtain the service curves describing a resource:

- They can be extracted from benchmark simulation runs, where simple workloads are processed by the resource.
- They can be obtained from data sheets describing the resource.
- They can be obtained from measurements of a real system.

3.3.2 Analysing a Single Task

Assume the scenario from Figure 10. In this figure, a processing element is shown (circle on the right). It is associated with a buffer to store incoming events waiting to be processed. On the processing element a task that that has to be executed for each of the events is implemented. By this, the computing capabilities of the resource are being modified. Further, an event stream is given with the corresponding arrival curves denoting the worst-case bounds for event arrivals. These events are stored in the buffer if needed and processed by the processing element.

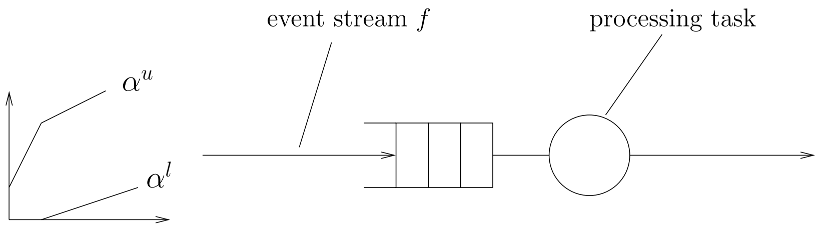


Fig. 10: Processing element executing a single task to process event stream f.

If we know the system from Figure 10, i.e. the upper and lower arrival curves α^u and α^l, and in addition know the processing element's computing capabilities described with service curves β^u and β^l, we can compute the description of the event stream after being processed. We can also compute the remaining service that can be offered by the resource after the event stream has been processed. We denote the event stream leaving the resource after being processed with arrival curves $\alpha^{u\prime}$ and $\alpha^{l\prime}$. We also call these arrival curves "outgoing". Similarly, we denote the computing capabilities remaining in the resource with service curves $\beta^{u\prime}$ and $\beta^{l\prime}$. Figure 11 gives a graphical representation for the curve processing. The curves can be related by the following expressions.

$$\begin{aligned}\alpha^{l\prime}(\Delta) &= \min\{\inf_{0\le\mu\le\Delta}\{\sup_{\lambda>0}\{\alpha^l(\mu+\lambda)-\beta^u(\lambda)\} \\ &\qquad +\beta^l(\Delta-\mu)\},\beta^l(\Delta)\} \\ \alpha^{u\prime}(\Delta) &= \min\{\sup_{\lambda>0}\{\inf_{0\le\mu<\lambda+\Delta}\{\alpha^u(\mu)+\beta^u(\lambda+\Delta-\mu)\}\end{aligned} \tag{3.3}$$

$$-\beta^l(\lambda)\}, \beta^u(\Delta)\} \tag{3.4}$$

$$\beta^{l\prime}(\Delta) = \sup_{0\le\lambda\le\Delta} \{\beta^l(\lambda) - \alpha^u(\lambda)\} \tag{3.5}$$

$$\beta^{u\prime}(\Delta) = \max\{\inf_{\lambda>\Delta}\{\beta^u(\lambda) - \alpha^l(\lambda)\}, 0\} \tag{3.6}$$

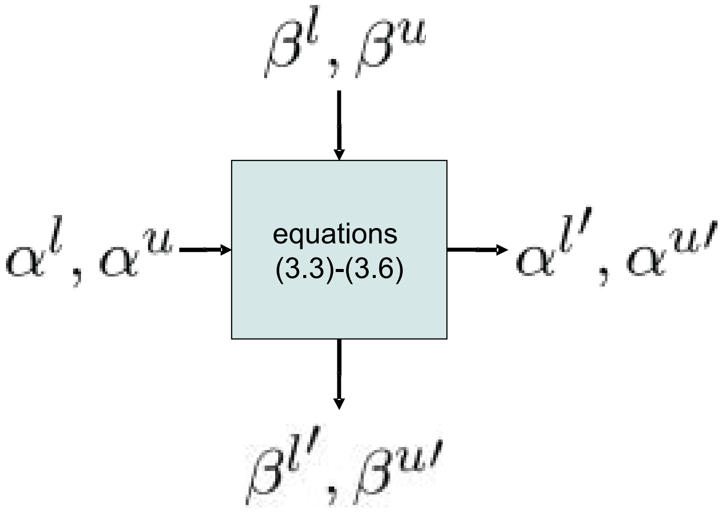

β^l, β^u
α^l, α^u
equations (3.3)-(3.6)
$\alpha^{l\prime}, \alpha^{u\prime}$
$\beta^{l\prime}, \beta^{u\prime}$

Fig. 11: Simple processing node for arrival curves.

These results are based on generalising ideas from network calculus as applied to the domain of communication networks (see [BT01] for details), and hold specifically for infinite event streams that span over time $t = -\infty$ to $t = +\infty$. Therefore, these are suited for modelling event streams such as periodic, sporadic, etc., which do not have any specific starting time (see also [CKT03a]). For modelling finite length event traces, the relations used in [TCGK02a, TCGK02b] may be used.

Moreover, it is not only possible to compute the outgoing event stream and the remaining resources for a given task, but also bounds on the delay experienced and backlog of unprocessed events. Assume α^l and α^u be the lower and upper arrival curves of an event stream entering a node whose input service curves are given by β^l and β^u. Then the maximum delay experienced by an event and the maximum number of backlogged events from the stream waiting to be processed can be given by the following inequalities.

$$delay \le \sup_{t\ge0} \left\{\inf\{\tau \ge 0 : \alpha^u(t) \le \beta^l(t+\tau)\}\right\} \tag{3.7}$$

$$backlog \le \sup_{t\ge0}\{\alpha^u(t) - \beta^l(t)\} \tag{3.8}$$

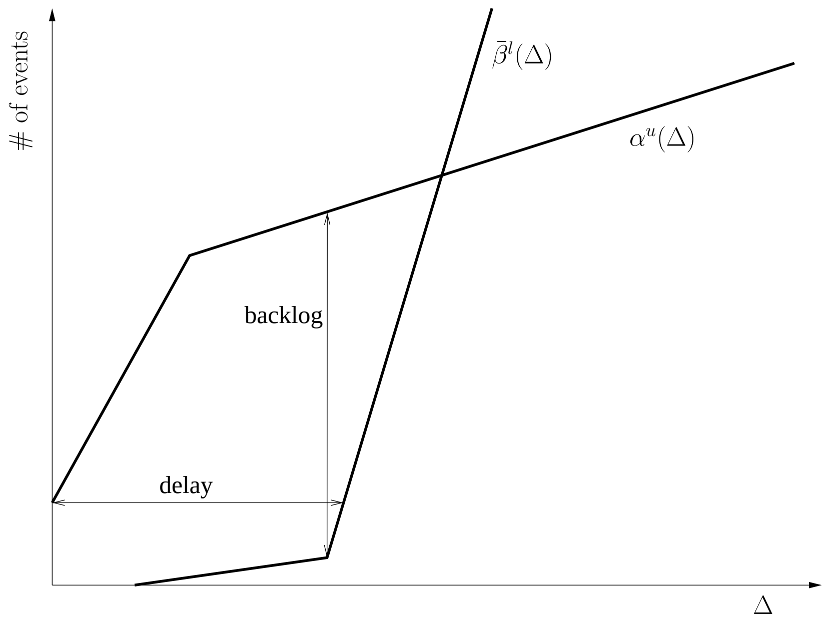


Fig. 12: Graphical representation of equations (3.7) and (3.8)

In Figure 12, a graphical interpretation of the Equations (3.7) and (3.8) is given. The maximum delay corresponds to the maximum horizontal distance between the upper arrival curve (α^u) and the lower service curve (β^l), whereas the maximum backlog corresponds to the maximum vertical distance between the two curves. For a physical interpretation of these inequalities, we refer the reader to [BT01]. Lastly, if β^u and $\beta^{l'}$ are the initial upper service curve and the lower remaining service curves of a resource, then its long-term utilisation in the worst case can be given by:

$$utilisation = \lim_{\Delta\to\infty} \frac{\beta^u(\Delta) - \beta^{l'}(\Delta)}{\beta^u(\Delta)} \tag{3.9}$$

In order to apply the equations (3.3)–(3.6) we have to make sure that both, the arrival curves and the service curves use the same units. Often the arrival curves describe the minimum and maximum number of events in an interval Δ, but the service curve give bounds on the number of processor cycles per time interval Δ. In order to be able to correctly use the equations for remaining service and outgoing event stream, we have to scale the input curves accordingly.

For event event stream or packet flow f, let w_f be its worst-case per event processing requirement on the resource, b_f the best-case processing requirement. From now on, we will assume w_f and b_f to be defined in time units, i.e. the resource takes between b_f and w_f time units to process each event of the stream f. To take these different processing requirements into account, we scale $\bar{\alpha}_i^l$ and $\bar{\alpha}_i^u$ appropriately before using Equations (3.3)–(3.6). $\bar{\alpha}^l(\Delta)$ and $\bar{\alpha}^u(\Delta)$ denote lower and upper event arrival curves, in other words, they give bounds on the number of events that can arrive in any interval Δ. Hence we have,

$$\alpha^u = w_f \bar{\alpha}^u, \qquad \alpha^l = b_f \bar{\alpha}^l \tag{3.10}$$

We scale the lower arrival curve with b_f — and not w_f or the average execution requirement — to ensure that the resulting, scaled arrival curve still represents a correct lower bound. Similarly, we scale the upper arrival curve with w_f. This procedure may introduce an error due to uncertainties caused by the variable execution requirement, but it is conservative and the resulting bounds are valid and safe. Using these scaled arrival curves, we can now compute the remaining service curves as shown in Figure 13.

The outgoing arrival curves representing the processed event stream can be computed using scaled service curves (as depicted in Figure 13):

$$\overline{\beta}^u = \lceil \beta^{u\prime} / b_f \rceil, \qquad \overline{\beta}^l = \lfloor \beta^{l\prime} / w_f \rfloor \tag{3.11}$$

In [MKT04] and [Max05], Maxiaguine *et al.* present workload curves, a further refinement of the scaling to reduce the error introduced and provide more accurate results. Workload curves capture variable execution times of tasks more accurately than just using best-case and worst-case execution times.

Now, we know how to calculate the outgoing event stream in form of arrival curves and the remaining resources in form of service curves, if we look at a single event stream for which a task has to be processed on a single resource. Next, we will look at the case, where multiple event streams have to be processed on the resource.

3.3.3 Multiple Streams on a Resource

If multiple event streams have to be processed by a resource, some sort of scheduling is necessarily involved. Scheduling means that some mechanism has to decide which event stream should be served at a certain point in time. In the next paragraphs we will have a closer look at a priority-based scheduling scheme.

Let us assume that there are n event streams entering a resource whose processing capability is bounded by the service curves β^l and β^u. Each event stream i is constrained by the arrival curves $\bar{\alpha}_i^l$ and $\bar{\alpha}_i^u$ and let the streams be ordered according to their priorities, i.e. stream 1 has the highest priority and stream n the lowest.

In the case of static priority scheduling, the resource processes the event streams in the order of decreasing priority, and the resulting arrival

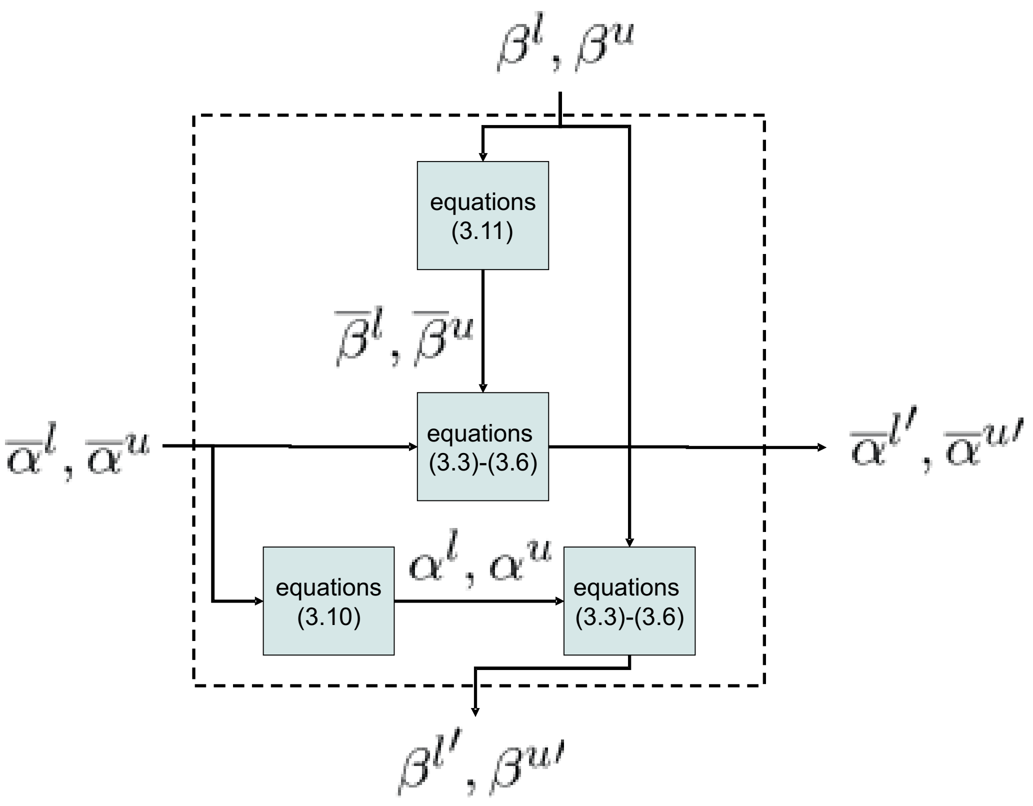

β^l, β^u
equations (3.11)
$\overline{\beta}^l, \overline{\beta}^u$
$\overline{\alpha}^l, \overline{\alpha}^u$
equations (3.3)-(3.6)
$\overline{\alpha}^{l\prime}, \overline{\alpha}^{u\prime}$
equations (3.10)
α^l, α^u
equations (3.3)-(3.6)
$\beta^{l\prime}, \beta^{u\prime}$

Fig. 13: Curve processing including scaling of arrival and service curves.

and service curves are computed using Equations (3.3)–(3.6). For the event stream 1, the service curves of the unloaded resource serve as an input. For the ith event stream, the input service curve is equal to the remaining service curve after processing the $(i-1)$th stream, for $i \geq 2$. This can be formally stated as follows: $\beta_1^u = \beta^u$, $\beta_1^l = \beta^l$, $\beta_i^u = \beta_{i-1}^{u\prime}$, $\beta_i^l = \beta_{i-1}^{l\prime}$, $i = 2, \ldots, n$, where $\beta_{i-1}^{u\prime}$ and $\beta_{i-1}^{l\prime}$ for $i = 2, \ldots, n$ are determined from β_{i-1}^u, β_{i-1}^l, α_{i-1}^u and α_{i-1}^l by applying Equations (3.5) and (3.6). Lastly, the remaining service curve after processing all the event streams is given as follows: $\beta^{u\prime} = \beta_n^{u\prime}$, $\beta^{l\prime} = \beta_n^{l\prime}$. This can be used to process other event streams, possibly using a different scheduling discipline, in a hierarchical manner.

If other scheduling policies (such as EDF, GPS) or arbitration schemes in case of communication resources (such as TDMA) should be used, this results in a changed arrangement of the single processing nodes compared to the one shown for priority-based scheduling in Figure 14. Note that also the available service curves for the individual tasks may be changed. For further information about other scheduling policies to be used with Real-Time Calculus the reader is referred to e.g. [TW05],[TCGK02a].

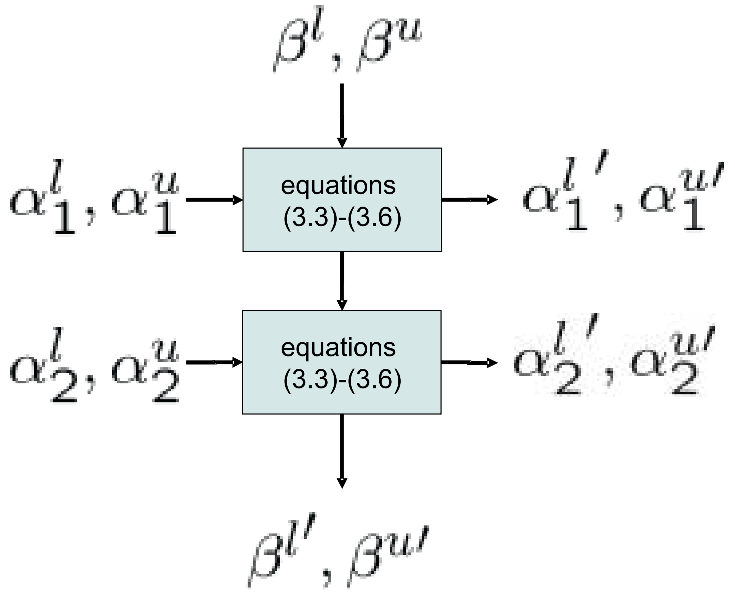

β^l, β^u
α_1^l, α_1^u
equations (3.3)-(3.6)
$\alpha_1^{l\prime}, \alpha_1^{u\prime}$
α_2^l, α_2^u
equations (3.3)-(3.6)
$\alpha_2^{l\prime}, \alpha_2^{u\prime}$
$\beta^{l\prime}, \beta^{u\prime}$

Fig. 14: Curve processing for 2 event streams using a priority-based scheduling.

3.3.4 Complex Systems with Multiple Resources

In this section, we extend the analysis method to platform architectures containing more than one resource. An example for such an architecture is given in Figure 6. It describes an hypothetical example network processor architecture. The numbered arrows denote the sequence of steps that is performed by the architecture to forward a packet from one network interface to the other. The architecture is introduced in more detail in Section 3.3.6.

Event streams flow through a network of resources based on the order in which they need to be processed. From the arrival curves describing the input load to a resource, we can compute the outgoing arrival curves with Equations (3.3) and (3.4). The arrival curves of an outgoing event stream from a resource serve as input arrival curves to another resource. Similarly, the processing capability of a resource is reduced by the processing of the incoming event stream. The remaining processing capability, as captured in the outgoing upper and lower service curves is used to process other event streams.

This model of an architecture can be represented as a *scheduling network*. The nodes of this network represent event processing functions that are implemented on the various resources. The inputs to each such node are the arrival curves of an event stream that is to be processed, and the service curve of the resource, representing the processing capability available to the function that is being implemented on the resource.

The outputs describe the resulting arrival curves of the processed event streams and the remaining service curves of the (partially) used resource. These arrival and service curves then serve as inputs to other nodes of the scheduling network. Note that “resources” in our framework refer to both communication (such as buses) and computation (such as processors) resources. The exact construction of the scheduling network for an architecture depends on the scheduling policies on the different architectural components, an example of which is shown in the next section.

Given a scheduling network corresponding to an architecture, it is possible to determine the timing properties (such as jitter and burst lengths) of the processed event streams from their outgoing arrival curves. Further, it is also possible to determine properties of the architecture such as the on-chip memory requirement and the loads on the different components such as processors and buses.

3.3.4.1 Analysis using a Scheduling Network

In Section 3.3.2 we described how to compute the delay and backlog experienced by an event stream passing through a single resource node processing the stream. For this we characterised the event stream using its arrival curves and the resource node using its service curve and also derived formulas for the maximum utilisation of this resource and the outgoing arrival and resource curves. Now we extend these results to consider the case where the event stream passes through multiple resource nodes as shown in Figure 6.

The outgoing arrival curves capture the characteristics of the processed event stream (for example its burstiness and long term rate), which might be different from the characteristics the stream has before entering the resource. Similarly the outgoing service curve indicates the remaining process capability of the resource after processing the event stream. The idea now is to use this outgoing arrival curve as an input to another resource node (more precisely, the resource node where the next event processing task as given by task graph described above is implemented). In the same way, the outgoing service curve of the first resource is used to process events from a possibly second stream. This procedure can be illustrated using a *scheduling network*. For example, Figure 15 shows the scheduling network corresponding to the event traversal through the architecture shown in Figure 6.

In general, multiple event streams enter an embedded system and are processed by the different resources in the order specified by the task graph described above. As events from several streams arrive at a resource, they are served in an order determined by the scheduling policy implemented at the resource. For example, many buses use a fixed

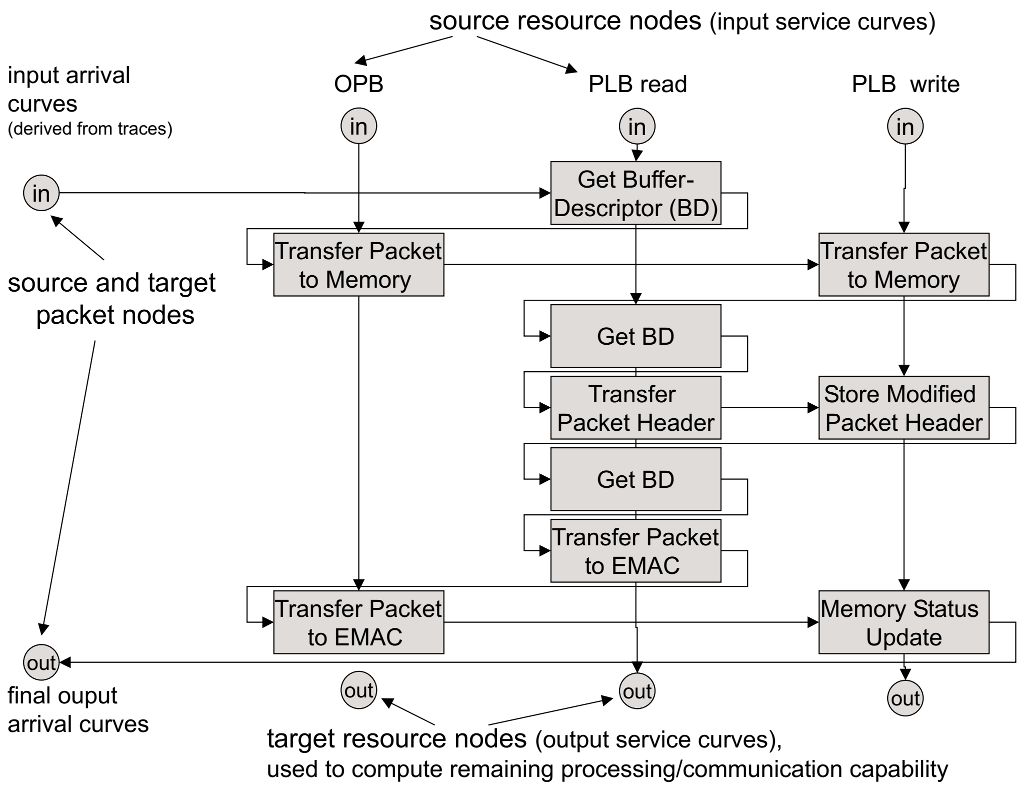

source resource nodes (input service curves)
input arrival curves
(derived from traces)
OPB
PLB read
PLB write
in
Get Buffer-Descriptor (BD)
Transfer Packet to Memory
Transfer Packet to Memory
source and target packet nodes
Get BD
Transfer Packet Header
Store Modified Packet Header
Get BD
Transfer Packet to EMAC
Transfer Packet to EMAC
Memory Status Update
out
final ouput arrival curves
target resource nodes (output service curves),
used to compute remaining processing/communication capability

Fig. 15: The scheduling network for the architecture given in Figure 6.

priority bus arbitration scheme. Other scheduling policies might be FCFS and round robin. We illustrate the analytical model here assuming that all the resources use fixed priority. However, the model can be extended to incorporate other scheduling policies as well (see [TW05]).

Let us assume that there are n event streams $f_1, \ldots, f_n$ arriving at a resource r, which serves these streams in the order of decreasing priorities, i.e. f_1 has the highest priority and f_n the lowest. For each event of the stream f_i, some task t_i implemented on r processes the event and this requires at most $w(t_i, r)$ processing units from r. $b(t_i, r)$ denotes the best case execution requirement. For example, $w(t_i, r)$ might be the number of processor instructions, or bus cycles in case r is a communication resource. We henceforth denote $w(t_i, r)$ by w_i, and $b(t_i, r)$ by b_i, when it is clear which resource is being referred to. Each event stream f_i arriving at r is associated with its upper and lower arrival curves $\bar{\alpha}_i^u$ and $\bar{\alpha}_i^l$ respectively and receives a *service* from r which can be bounded by the upper and lower service curves β_i^u and β_i^l respectively. The service available from r in the unloaded state (i.e. before any of the event streams $f_1, \ldots, f_n$ are served) is bounded by the upper and lower service curves β^u and β^l respectively.

In the fixed priority scheme r services the flows in the order of decreasing priorities and the remaining service curve after processing an event stream is used to serve the lower priority streams. The resulting arrival curves and the remaining service curves can be computed using Equations (3.3)–(3.6) given in Section 3.3.2.

Since events from different streams might have different maximum processing requirements given by $w_1, \ldots, w_n$, and also different minimum processing requirements given by $b_1, \ldots, b_n$ the arrival curves first need to be scaled as described in Section 3.3.2 to calculate the remaining service curves using Equations (3.5)–(3.6), as shown in Figure 13.

Similarly, to calculate the outgoing arrival curves, we first need to scale back the service curves as follows. If β^u and β^l are the service curves for a resource node, then $\overline{\beta}^u = \lceil \beta^u / b_i \rceil$ and $\overline{\beta}^l = \lfloor \beta^l / w_i \rfloor$. The floor/ceiling functions are used since a subsequent resource node can start processing an event only after the task implemented on r finishes processing it. Using these adapted service curves, we can then compute the outgoing arrival curves describing the processed event stream using Equations (3.3)–(3.4).

3.3.4.2 Scheduling Network Construction

Using the results in the last section we now describe the procedure for constructing a scheduling network. This can then be used to determine properties of the architecture such as the on-chip memory requirement, the end-to-end delay experienced by packets and the utilisation of the different on-chip resources such as processors and buses.

The inputs necessary for constructing such a network are the task graph which denotes for each event stream the sequence of processing tasks that are to be executed on any event of the flow and the target architecture on which these tasks are mapped. Furthermore, the scheduling policy on the resources also influences the scheduling network construction.

The scheduling network contains one *source resource node* and one *target resource node* for each resource used in the architecture. Similarly, for each event stream there is a *source event node* and a *target event node*. For each processing task of an event stream there is a node in the network marked with the task and the resource on which this task is implemented, e.g. if task u is implemented on r_u, we denote this by (u, r_u). For two consecutive tasks u and v of an event stream, if u is implemented on a resource r_u and v on a resource r_v then there is an edge (drawn horizontally in Figure 15) in the scheduling network from the node (u, r_u) to (v, r_v). For a given event stream, if u and v are two tasks implemented on the same resource r and u precedes v in the task graph, then there is an edge (drawn vertically in Figure 15) from the node (u, r) to the node (v, r).

The arrival curves of the event streams and the service curves of the resources pass from one node to the next in the scheduling network and get modified in the process, following Equations (3.3)–(3.6).

Now, we look at how the overall processing delay and the total backlog can be bounded. For this we have to compute the *accumulated lower service curve*. For a given event stream f, let α_f^u be its upper arrival curve before entering the network processor. Suppose this stream passes through nodes of the scheduling network which have their input lower service curves equal to $\beta_1^l, \ldots, \beta_m^l$. Then the *accumulated lower service curve* β^l used to serve this event stream can be computed as follows.

$$\begin{aligned}
\hat{\beta}_1^l &= \beta_1^l \\
\hat{\beta}_{i+1}^l &= \inf_{0 \le t \le \Delta} \left\{ \hat{\beta}_i^l(t) + \beta_{i+1}^l(\Delta - t) \right\},\ i = 2, \ldots, m-1 \\
\beta^l &= \hat{\beta}_m^l
\end{aligned}$$

We can give the end-to-end delay and the total backlog experienced by events from stream f by:

$$\mathit{delay} \le \sup_{t \ge 0} \left\{ \inf\{\tau \ge 0 : \alpha_f^u(t) \le \beta^l(t+\tau)\} \right\} \tag{3.12}$$

$$\mathit{backlog} \le \sup_{t \ge 0} \{ \alpha_f^u(t) - \beta^l(t) \} \tag{3.13}$$

Compared to independently deriving the delay and backlog at single resources using inequalities (3.7) and (3.8) from Section 3.3.2 and adding them, the inequalities (3.12) and (3.13) give tighter bounds.

3.3.5 Adaptation for Design Space Exploration

Equations (3.3)–(3.6) are clearly expensive to compute for general arrival and service curves. Moreover, these equations need to be computed for all the nodes of a scheduling network. Additionally, if these curves are to be meaningfully derived out of packet traces (as shown later in this chapter), then the resulting curves can be described by a few parameters such as the maximum packet size, the short-term burst rate, and the long-term packet arrival rate. In view of this, we propose a piecewise linear approximation of all arrival and service curves. Using these approximations, the Equations (3.3)–(3.6) can be efficiently computed, thereby avoiding the computational bottleneck involved in dealing with general curves.

Each curve in this case is approximated using a combination of three straight line segments. This allows us to exactly model an arrival curve in the form of a T-SPEC [SW97], which is widely used in the area of communication networks. Figure 16 shows the resulting form of the upper and lower curves (both arrival and service). Here q_1^u represents the maximum possible load on a resource for processing one packet. The slope r^u of the middle segment of the upper curve can be interpreted as the burst rate, and the slope s^u as the (load on a resource due to the) long term packet arrival rate. In the case of communication resources, q_1^u represents the maximum packet size. The values of p^u and p_1^l, p_2^l can be computed from these parameters.

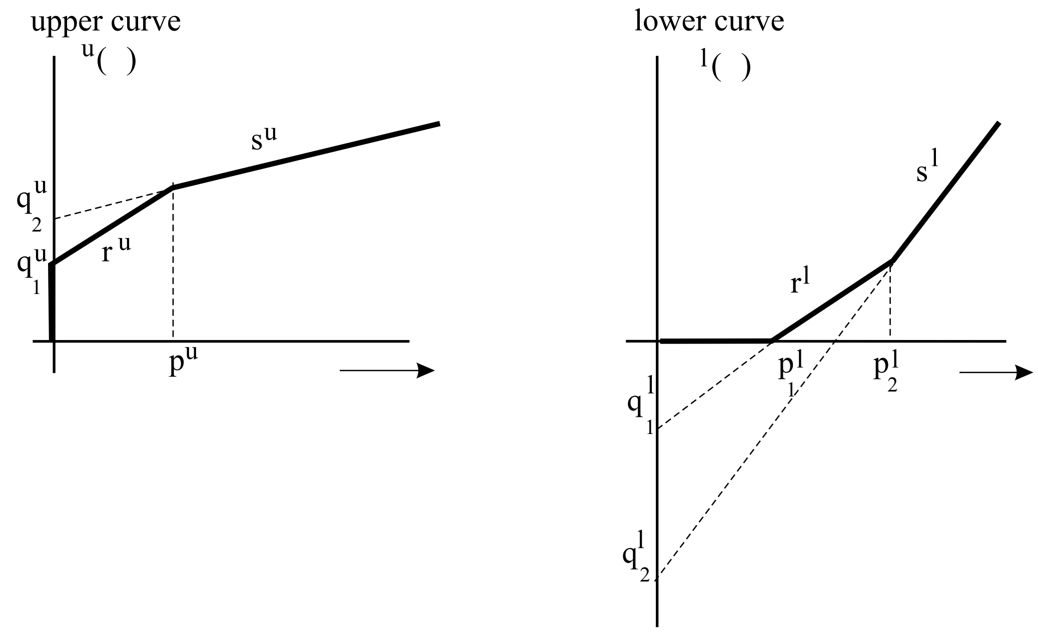


Fig. 16: Piecewise linear approximations of the upper and lower (arrival and service) curves.

Any upper (say γ^u) and lower (say γ^l) curves can now be written as the following.

$$\begin{aligned}\gamma^u(\Delta) &= \min\{q_1^u + r^u\Delta, q_2^u + s^u\Delta\}\\ \gamma^l(\Delta) &= \max\{q_2^l + s^l\Delta, q_1^l + r^l\Delta, 0\}\end{aligned}$$

where,

$$\begin{aligned}&q_2^u \geq q_1^u \geq 0, \quad r^u \geq s^u \geq 0, \quad r^u = s^u \Leftrightarrow q_1^u = q_2^u\\ &q_2^l \leq q_1^l \leq 0, \quad 0 \leq r^l \leq s^l, \quad r^l = s^l \Leftrightarrow q_1^l = q_2^l\end{aligned}$$

Using these approximated curves, the Equations (3.3)–(3.6) as well as maximum delay and backlog can be evaluated efficiently. In the following, we will give the results for Equation (3.6) as an example for the approximations implemented in the tool for design space exploration presented in Chapter 5. We give the calculations for all the equations and their derivation in Appendix B.

Prop. 1: (Remaining Upper Service Curve) *Given the lower arrival and upper service curves $\alpha^l = L(q_{1\alpha}, q_{2\alpha}, r_\alpha, s_\alpha)$ and $\beta^u = U(q_{1\beta}, q_{2\beta}, r_\beta, s_\beta)$ respectively, then the remaining upper service curve $\beta^{u\prime} = U(q_1, q_2, r, s)$ is determined as follows:*

$$
\begin{aligned}
r &= \begin{cases} 0, & if\, s_\beta < s_\alpha \\ r_\beta, & if s_\beta \geq s_\alpha \wedge p_\beta < p_{1\alpha} \\ r_\beta - r_\alpha, & if\, s_\beta \geq s_\alpha \wedge p_\beta \geq p_{1\alpha} \end{cases} \\
s &= \begin{cases} 0, & if\, s_\beta < s_\alpha \\ s_\beta - s_\alpha, & if\, s_\beta \geq s_\alpha \end{cases} \\
q_1 &= \begin{cases} 0, & if\, s_\beta < s_\alpha \\ q_{1\beta}, & if\, s_\beta \geq s_\alpha \wedge p_\beta < p_{1\alpha} \\ q_{1\beta} - q_{1\alpha}, & if\, s_\beta \geq s_\alpha \wedge p_\beta \geq p_{1\alpha} \end{cases} \\
q_2 &= \begin{cases} 0, & if\, s_\beta < s_\alpha \\ q_{2\beta} - q_{2\alpha}, & if\, s_\beta \geq s_\alpha \end{cases}
\end{aligned}
$$

3.3.6 Example System 1: Network Processor

In this section, we show how the results presented so far can be used to formulate an analytical performance evaluation model for an example network processor. The results here were originally derived in [TCGK02b] and [TCGK02a]. In the following sections, we will use the same network processor example modelled as a simulation system (see Section 3.4.1) and compare the results using the two different evaluation methods for this example system (see Section 3.5).

We view a network processor as a collection of different processing elements (such as CPU cores, micro-engines, and dedicated units like hardware classifier, cipher, etc.) and memory modules connected to each other by communication buses. On each of these processing elements one or more packet processing tasks are implemented. Depending on the sequence in which these tasks process a packet, and the mapping of these tasks on to the different processing elements of the network processor, any packet entering the network processor follows a specific sequence through the different processing elements. The flow to which this packet belongs is associated with its arrival curves. Similarly all the resources have their associated service curves. As the packets of the flow move from one processing element to the next, and also cross communication resources such as buses, both, the arrival curves of the flow and the service curves of the resources get modified following the Equations (3.3)–(3.6) given in Section 3.3.2. Given this, the maximum end-to-end delay experienced by any packet, the on-chip memory requirement of the network processor, and the utilisation of the different resources (both computation and communication) can now be computed using Equations (3.7), (3.8) and (3.9).

To formally state the above procedure, consider that for the set of flows F entering the network processor, there is a task graph $G = (V,E)$. Any vertex $v \in V$ denotes a packet processing (or communication) task. For any flow $f \in F$, let $V(f) \subseteq V$ denote the set of packet processing tasks that have to be implemented on any packet from F. Additionally, a subset of directed edges from E defines the order in which the tasks in $V(f)$ should be implemented on any packet from f. Therefore, if $u, v \in V(f)$ represent two tasks such that for any packet belonging to f, the task v should be implemented immediately after task u, then the directed edge (u, v) belongs to the set E. Hence, for each flow f there is a unique path through the graph G starting from one of its source vertices and ending at one of its sink vertices. The vertices on this path represent the packet processing tasks that are to be implemented on packets from f.

3.4 Simulation-based Performance Analysis

Simulation-based approaches are widely used for performance evaluation of embedded system. There exist many different approaches on various abstraction layers. It's beyond the scope of this thesis to give a complete overview over all the existing approaches. In general, simulation is well established and accepted by designers for performance evaluation. In comparison to the formal approach presented in Section 3.3, simulation deals with real workload inputs, i.e. non-functional and functional properties of the system can be determined. In contrast to the worst-case analysis that is performed with formal methods, simulation can be used to determine the average-case behaviour. In the context of design space exploration, the use of simulation-based approaches to performance evaluation is limited due to the often high running times.

SystemC [GLMS02],[SYS] is a system description language that can be used to model the behaviour level of systems. It consists of a set of library routines and macros implemented in C++, which makes it possible to simulate hardware building blocks and concurrent processes written in standard C++. SystemC supports the communication of C++ objects in a simulated real-time environment. SystemC is both a description language and a simulation kernel. The code written will compile together with the library's simulation kernel to give an executable that behaves like the described model when it is run.

SystemC supports the simulation of a system at various levels of abstraction — from cycle-accurate up to the behavioural level. This fact, together with the possibility of simulation of hardware and software described with the same language are the main sources of success for this language.

In the remainder of this section, we briefly describe two simulation frameworks for performance evaluation written in SystemC: (1) the approach presented in [Wor01, CKT$^+$03b] used for network processors architectures and (2) the platform described in more detail in [MAS$^+$05] for multi-processor system on a chip architectures.

3.4.1 Example System 1: Network Processor

In this section, we describe the methodology for performance evaluation of the example network processor architecture introduced in Section 3.1.1 based on simulation, which forms the basis for comparing the results obtained from the analytical framework presented in Section 3.3.6.

The work described in this section was done by Frédéric Worm. We could reuse the performance models that he wrote in SystemC for his Master's thesis [Wor01] for our purposes. This section therefore gives only an overview over the work presented in [Wor01] and more detailed explanations concerning the models can be found there. Here, the primary goal is to illustrate the abstraction level at which the different components of the architecture are modelled for the simulator.

3.4.1.1 Modelling Environment and Software Organisation

The overall approach is based on using a library of reusable component models written in SystemC. These include models for communication resources, different memories, network interfaces, and processor cores. Each of these models can be seen as an abstract description of an existing hardware component which can be found from a core library [IBMa], or can also be a model of a new component which does not exist yet. In an architecture composition step the selected component models are combined into a system model which is then evaluated through simulation. To drive the simulation, either synthetic workloads can be used, or real traffic traces can serve as workload. Synthetic traces can for instance be traces with periodic packet arrival. During simulation, the model execution performance data is collected and stored to files, such that the data can be evaluated later.

It is not necessary that every component model is implemented on the same level of abstraction. The models are implemented as black boxes having abstract interfaces. The components only interact through these interfaces. This allows us to refine the individual components if a model with more detail is needed for some component. It also supports easy exchangeability among different models of the same component. Every component model is separated into two layers—a so called abstract functional layer and a data collection layer. The functional layer describes the functional behaviour of a component and defines the component interfaces, while the data collection layer exclusively deals with gathering statistics which are used to evaluate different performance metrics. This separation enables independent and selective refinements on the functional layer and also flexible customisation of the data collection mechanisms without interference among these layers.

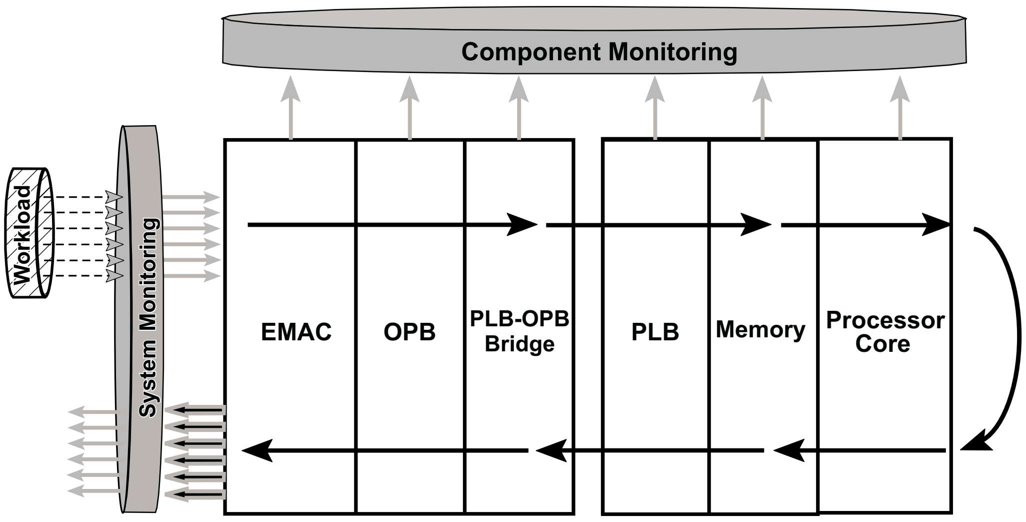


Fig. 17: System evaluation and component evaluation of a network processor architecture.

With the simulation framework, we can do a performance evaluation either as a component-level evaluation or a complete system evaluation using the data collected during simulation. These two ways are illustrated in Figure 17. The component evaluation is based on the data collected through the data collection layer of each component model. Examples of such evaluation metrics can be memory fill levels, bus re-arbitration counts, and the load on the different components. System specific aspects of an architecture like the overall system throughput, or end-to-end packet delays are evaluated using the system evaluation mechanism. In contrast to the component evaluation approach, the data in this case is not gathered within a specific component but is collected using the workload travelling through the entire system (in the figure, this procedure is denoted with system monitoring).

Figure 18 shows an overview of the entire simulation framework. The abstract models are created based on specification documents. The models are then enhanced with the component-specific and system-specific data collection layer. Such models build the “Model Library” which is also shown in the figure. The models for the individual components can then be composed together and the system can be simulated.

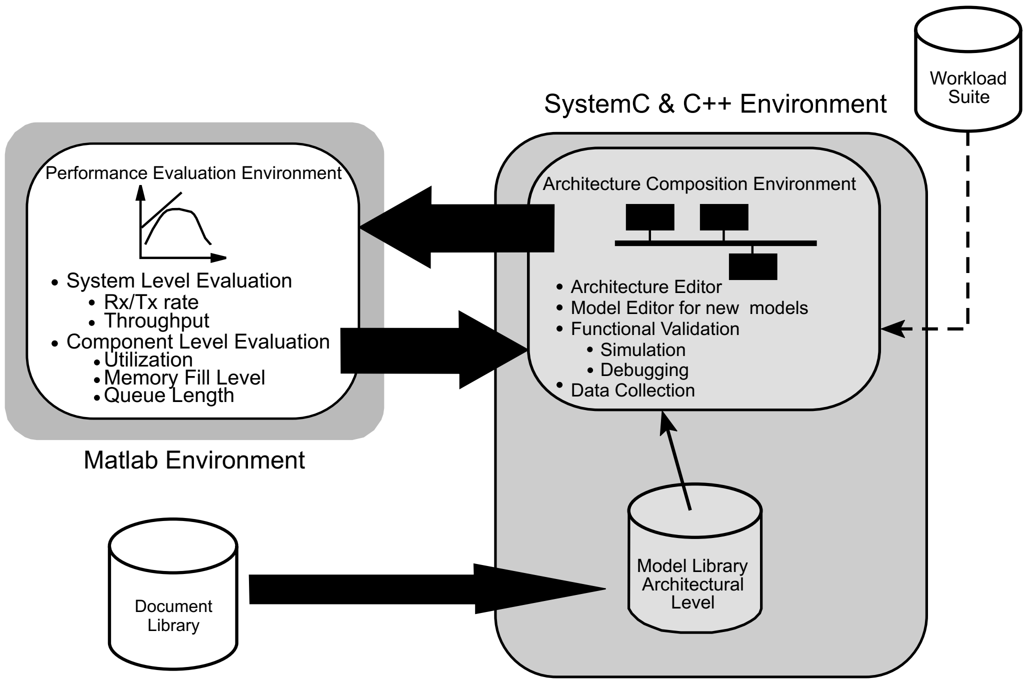

Workload Suite
SystemC & C++ Environment
Performance Evaluation Environment
• System Level Evaluation
• Rx/Tx rate
• Throughput
• Component Level Evaluation
• Utilization
• Memory Fill Level
• Queue Length
Matlab Environment
Architecture Composition Environment
• Architecture Editor
• Model Editor for new models
• Functional Validation
• Simulation
• Debugging
• Data Collection
Model Library Architectural Level
Document Library

Fig. 18: Overview of the simulation framework that is used for comparing the results obtained by the analytical method.

3.4.1.2 Component Modelling

In this section we briefly describe how each component of the reference network processor architecture is modelled in the simulation. This will enable a meaningful comparison between the results obtained by this simulation framework and those obtained from the analytical model of the same architecture described in Section 3.3.6. The models for actual designs are usually available from core libraries as hard or soft cores. The SystemC models used for the simulation were created for the work done in [Wor01] based on their specification. The following discussion below refers to these SystemC models.

Bus Models. The bus models used in the reference architecture are based on the CoreConnect bus architecture [IBMb], designed to facilitate core-based designs. CoreConnect involves three buses: The Processor Local Bus (PLB) is for interconnecting high performance cores with high bandwidth and low latency demands, such as CPU cores, memory controllers and PCI bridges. The On-Chip Peripheral Bus (OPB) hosts lower data rate peripherals such as serial and parallel ports and UARTs. The PLB is fully synchronous, has decoupled address, read and write data lines and transfers on each data line are pipelined. Different masters may be active on the PLB address, read and write data lines and access to the PLB is granted through a central arbitration mechanism that allow masters to compete for bus ownership.

The models used for the PLB and the OPB are both cycle accurate. Both these models do not transfer any data nor consider any address, but model the signal interchanging according to the bus protocol.

There is a third bus in CoreConnect, which we do not use in our study. This is called the Device Control Register (DCR) bus, and is used for reading and writing low performance status and configuration registers.

The features of the PLB that are modelled for our study include the arbitration mechanism, the decoupled address and read and write lines, a fixed length burst protocol, and a slave enforced arbitration mechanism. The OPB is much less sophisticated compared to the PLB and most of its features are modelled. For both the buses, the arbitration algorithm uses a round robin strategy among the requesting masters on a given priority level, and typically there are four priority levels.

Ethernet Core (EMAC) Model. The Ethernet core or EMAC is a generic implementation of the Ethernet media access control (MAC) protocol, supporting both half-duplex (CSMA/CD) and full duplex operations for Ethernet, Fast Ethernet and Gigabit-Ethernet. The EMAC has two large FIFOs to buffer packets, and two OPB interfaces—one for providing access to its configurations and status registers, and the other is a bidirectional interface to transfer data to and from the PLB-OPB bridge.

The model of the EMAC only contains receiving and transmitting channels, where a receive channel can be considered as an input port and a transmit channel as an output port.

The set of receiving channels constitutes the traffic input to the network processor. Each one reads packet information from a real traffic trace at a parameterisable rate. Within a receive channel there are two threads of activity. The first one reads the input packet traces, and writes each resulting packet into a FIFO. The second thread implements the communication protocol with the PLB-OPB bridge and transfers packet to memory as long as the FIFO is not empty. The transmit path consists only of a *transmit packet* thread, which is active as long as packets are waiting to be transferred for the appropriate port.

PLB-OPB Bridge Model. The PLB-OPB Bridge is a specialised module which combines pure bridge functionality with DMA capability, and it can effectively handle packet transfer overheads. An EMAC communicates with a PLB through the PLB-OPB bridge. Since as an OPB slave, the EMAC can not inform the bridge of its willingness to transfer a packet, the EMAC to PLB-OPB bridge interface has *sideband* signals to meet this purpose. These do not form a part of the OPB bus, and nearly all signals are driven by the EMAC and sampled by the bridge.

The PLB-OPB bridge is modelled as two independent paths, receive and transmit, and both offer the bridge functionality and arbitrate among active channels. Each path implements the PLB-OPB bridge to EMAC communication protocol by driving the required sideband signals and accessing buses. Bus accesses are concurrent and therefore both paths can contend for their access, especially on the OPB.

Memory Model. The memory is accessed via the PLB and can either be on-chip or off-chip. It is modelled in a high level way, where only parameters like average or worst case transfer latency are considered.

Software Application and Timing. A simple high-level model of software application is used. It primarily consists of the following. For each packet the software can cause a pure delay without generating any PLB load, representing processing time in the CPU. Second, there can be a PLB load generated by the software (for example, this might consist of reading and writing packets by the CPU to and from the memory). Lastly, the timing model is based on using the PLB clock as the system time.

In Section 3.5, we will compare the results obtained with this simulation framework written in SystemC with the results obtained using the analytic model of the network processor.

3.4.2 Example System 2: Multiprocessor Platform

We present here a second simulation platform. In contrast to the last section, this framework supports the cycle-true simulation of communication and computation subsystems. This platform was also used for a comparative study (cf. Section 3.5) and for the new hybrid approach for performance analysis presented in Section 3.6.

The MPARM system simulation environment [LAB+04], performs functional, timing-accurate simulation of ARM-based multi-processor systems and is also written in SystemC. MPARM provides a complete analysis toolkit allowing to monitor performance and energy dissipation (based on industry-provided power models) of platform components for the execution of software routines as well as of an entire benchmark. The simulation is cycle accurate and bus-signal accurate. Our virtual platform leverages technology-homogeneous (0.13 μm) power models of all system components (processor cores, system interconnect, memory devices) provided by STMicroelectronics [LPB04, BZZ04]. The processor core models take into account the cache power dissipation, which accounts for a large fraction of overall power.

3.5 Comparison between Simulation and Formal Method

We present here the results of two comparative studies: (1) we analysed the communication system for a packet processor architecture, and (2) we analysed the performance of a multiprocessor platform for a distributed application.

3.5.1 Example System 1: Network Processor

This section presents a comparison of the performance evaluation results obtained by the analytical framework presented in Section 3.3 with the results obtained by detailed simulations based on the models discussed in Section 3.4. The results are obtained for the example network processor architecture introduced in Section 3.1.1. The discussion of the results is based on the assumption that there is a high confidence in the simulation results.

The network processor architecture which we use as a basis for the comparative study can be matched by many existing network processors (such as the family of processors described in [Pow]). The architectural components modelled in this study are from an existing core library [IBMa].

We evaluate the reference architecture using three different performance metrics:

1. The line speed or the end-to-end throughput that can be supported by the architecture. This is measured using the utilisation of the different components of the architecture and hence also identifies the component which acts as the bottleneck. During a design space exploration, identifying the utilisation of the different components goes beyond measuring the overall throughput of the system because a designer in generally interested in identifying whether all the components in the architecture have a moderately high utili-

sation at the maximum supported line speed, or whether it is one single component that acts as a bottleneck.

2. The maximum end-to-end delay that is experienced by packets from the different flows being processed by the architecture.

3. The total on-chip buffer/memory requirements, or in other words, the on-chip memory fill level.

The results of the analytical framework should be judged on the basis of how closely the data related to these performance metrics for the reference architecture match those obtained using simulation. Rather than absolute values, it is more interesting to analyse the behaviour of the architecture (for example with increasing line speeds, or increasing packet sizes for the same line speed), and see if the same conclusions can be drawn from both the evaluation techniques. In addition to this trend analysis comparison, we also take into account the time it takes to compute the evaluation data by the analytical framework and compare it to the simulation time required to generate this data.

3.5.1.1 Reference Architecture and Parameters

The system-level model of a network processor that is used for this study is shown in Figure 6. The different actions that are executed while each packet travels through this architecture, and the order in which they are executed is given in Table 3. The model effectively deals with the communication subsystem of the architecture, and the software application running on the CPU core (indicated by PPC - PowerPC) is modelled as simply performing two reads from the SDRAM and one write to the SDRAM for every processed packet. The amount of data read or written (and hence the traffic generated on the PLB), however, depends on the packet size and this is appropriately modelled.

As seen in Figure 6, the architecture is composed of two Ethernet media access controllers (EMACs), a slow on-chip peripheral bus (the OPB), a fast processor local bus (the PLB) consisting of separate read and write lines, a PLB-OPB bridge, a SDRAM and a processor core (PPC). Each EMAC consists of one receive and one transmit channel, and is capable of reading packets at parameterisable input rates.

In the simulation, the entire path of a packet through the modelled architecture can be described as follows. First a receive channel of an EMAC reads a packet from a file containing the packet traces (only packet lengths are used, and all packets arrive back-to-back with a fixed inter-frame gap; this is described in further details later), and allocates a packet record which contains the packet length and the source EMAC identification.

This packet record models the packet inside the processor architecture and generates a load equivalent to the size of the packet. The channel then requests service to the PLB-OPB bridge via a sideband signal, which is served following a bridge internal arbitration procedure. The PLB-OPB bridge fetches a *buffer descriptor* for the packet (which is a data structure containing a memory address in the SDRAM, where the received packet is to be stored). This fetching operation involves the SDRAM and generates traffic on the PLB read bus, equal to the size of the buffer descriptor. Following this, the received packet is stored in the SDRAM at the location specified by the buffer descriptor. This involves the packet traversing through the OPB to the PLB-OPB bridge, and then through the PLB write bus to the SDRAM. This generates a load equal to the size of the packet, on both the buses. Since data on the PLB is sent in bursts, the PLB-OPB bridge schedules a PLB transfer only when sufficient data is gathered. As the EMAC channel is served over and over again, the packet is written part by part into the SDRAM. After the packet transfer is complete, the bridge informs the EMAC receive channel via a sideband signal, and also notifies the application software running on the processor core (again by a sideband signal) that the packet is now available in the memory. It is then processed by the software as soon as the processor becomes available. This processing involves a buffer descriptor transfer from the SDRAM to the processor core via the PLB read bus, followed by a packet header transfer, again from the SDRAM to the processor core via the PLB read bus. The packet header is then processed in the processor core (for example implementing some lookup operation) and written back into the SDRAM over the PLB write bus.

After the completion of this processing, the software notifies the bridge (via a sideband signal) that the packet is now processed and is ready to be sent out through the chosen transmit channel of the EMAC. As in the receive path of the packet, the bridge gets the buffer descriptor of the packet from the SDRAM via the PLB read bus, and then the packet traverses over the PLB read bus and the OPB to an EMAC transmit channel. After the completion of the packet transfer the EMAC notifies the bridge via a sideband signal, which then reads certain status information and releases the buffer descriptors.

This entire process happens concurrently for two packet flows entering through the two EMACs of the architecture. All the components involved also work concurrently and the two buses (the PLB and the OPB) use first-come-first-serve as a bus arbitration policy. The main complexity in the analysis of this system is due to concurrent operation of the different components. Hence it is non-trivial to evaluate how the system behaves with increasing line speeds, variations in packet sizes, and what is the maximum line speed that it can support without packet dropping.

Parameters. As already mentioned, the EMAC can read packets at different input line speeds. The line speeds used for the evaluation range from 100 Mbps to 400 Mbps, the former representing a nominal load situation and the later a loaded situation.

The OPB modelled has a width of 32 bits and a frequency of 66.5 MHz. The read and the write data paths of the PLB are of 128 bits and operate at 133 MHz. The size of a PLB burst is limited to a maximum of 64 Bytes. Therefore, the PLB-OPB bridge gathers up to 64 Bytes (which is only one OPB burst transfer) before scheduling the PLB transfer. There are two different kinds of buffer descriptors, small and large ones. The small buffer descriptors refer to memory locations/buffers with 64 Bytes of size, while the large ones refer to buffers with a size of 1472 Bytes. As a consequence, 64 Bytes sized packets require only one small buffer descriptor and packets larger than 64 Bytes require an additional large buffer descriptor. Both *small* and *large* buffer descriptors are of size 64 Bytes each. Therefore, the traffic generated by a packet on any of the buses depends not only on its own size, but also on the buffer descriptors associated with it. All packets and the buffer descriptors reside in the SDRAM described above.

3.5.1.2 Evaluation Method and Comparisons

The reference architecture described above is evaluated using simulation and the analytical framework using two different workload types—synthetic traces with same-sized packets, and real traces from NLANR [LNA]. For the synthetic traces, packet sizes of 64, 128, 512, 1024, 1280 and 1500 Bytes are used. The real traces are used only to exploit the impact of real world packet size distributions on the system performance. They are time compressed and adjusted and only the packet sizes are retained. Therefore, in both the cases packets arrive back-to-back (to exert the maximum stress on the architecture) with an inter-frame gap equal to 20 Bytes.

The overall scheme for comparing the results obtained using the analytical framework with those obtained from simulation is shown in Figure 19. The different components of the architecture are modelled in either SystemC in the simulation-based evaluation, or analytically using the model presented in Section 3.3. To compute the required parameters of an analytical component model (such as the transfer time of a single packet over an unloaded bus), either simulation results using simple workloads are used (as presented in step 1 of the figure) or data sheets of the component are used. The component models are then composed together (using the methods described in Section 3.3.6 in the case of the analytical framework, and using standard SystemC composition techniques in the case of

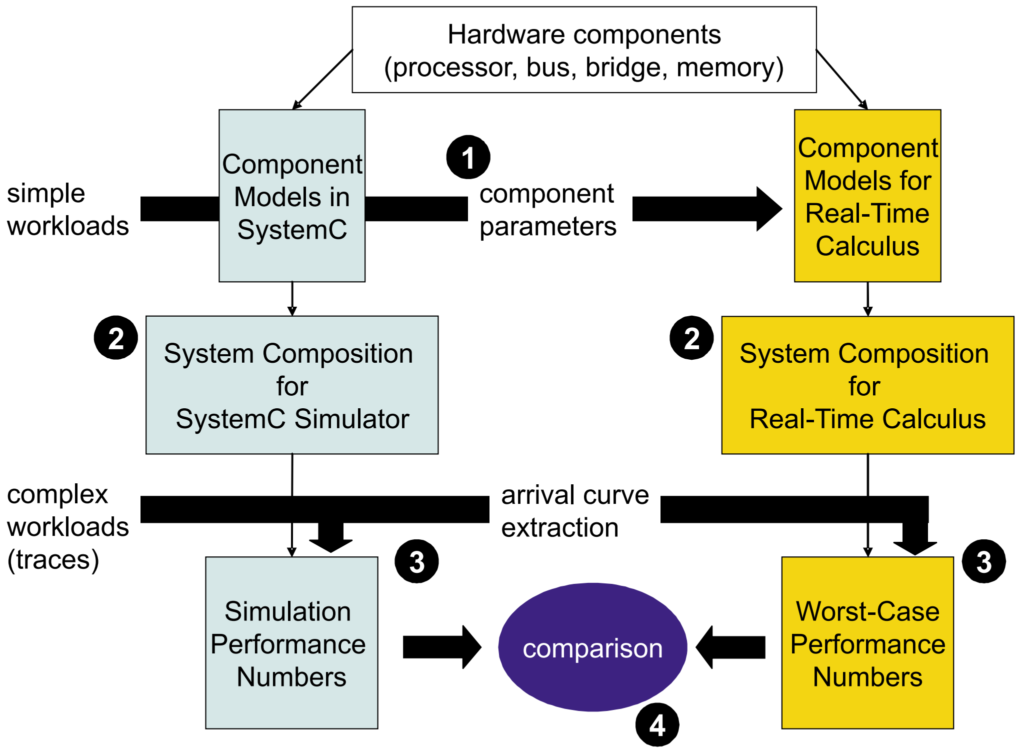

Hardware components
(processor, bus, bridge, memory)
simple workloads
Component Models in SystemC
1
component parameters
Component Models for Real-Time Calculus
2
System Composition for SystemC Simulator
2
System Composition for Real-Time Calculus
complex workloads (traces)
arrival curve extraction
3
3
Simulation Performance Numbers
comparison
Worst-Case Performance Numbers
4

Fig. 19: The overall scheme for comparing the results from the analytical framework with those obtained by simulation.

simulation) to obtain a system model of the architecture (step 2). These system models can now be used for performance analysis. To drive the system analysis, either real packet traces are used, or the corresponding arrival curves (step 3). Finally, the performance numbers obtained either with the analytical model or the SystemC simulation are compared (step 4 in the figure).

The analytical model considered here does not use real packet traces, but uses arrival curves modelling the traces in terms of their maximum packet size, burstiness, and long term arrival rate. These parameters were extracted from the traces as shown in Figure 20 and fed into the model for evaluation. For the upper arrival curve, the maximum number of Bytes that can arrive (at the network processor) at any time instant is given by the largest sized packet, the short-term burst rate is given by the maximum number of largest sized packets that can be seen occurring back-to-back in the trace, and the long-term arrival rate is given by the total length (in Bytes) of a trace divided by the time interval over which all the packets in this trace arrive.

Similarly, for the lower arrival curve, the maximum time interval over

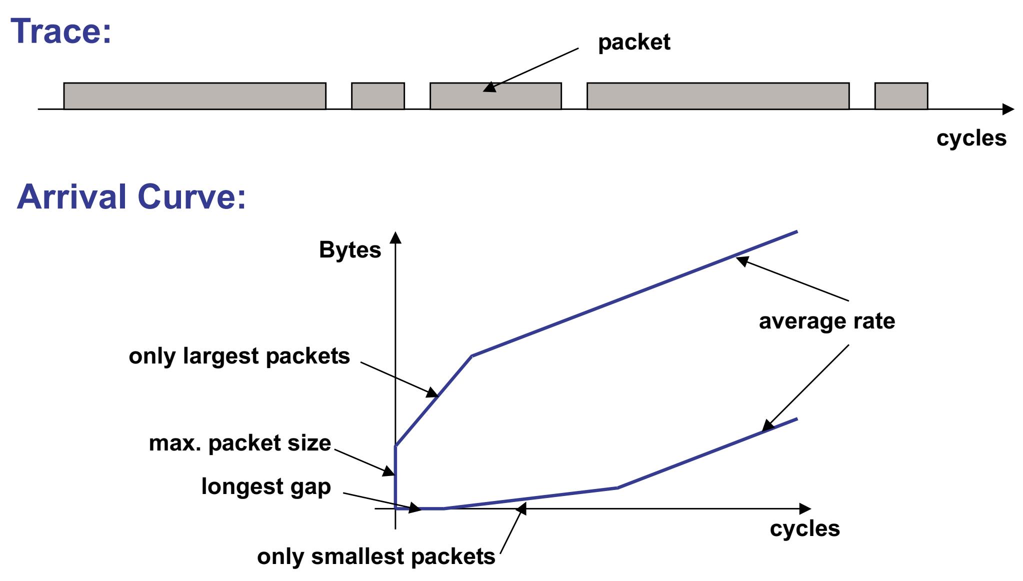


Fig. 20: Obtaining arrival curves from packet traces in the analytical framework.

which no traffic can arrive is equal to the inter-frame gap in the trace (equal to 20 Bytes), the bound on the minimum number of Bytes that can arrive over a time interval is given by the maximum number of minimum sized packets occurring back-to-back in the trace, and the long-term arrival rate is equal to that in the upper arrival curve.

Given any packet trace, arrival curves such as those shown in Figure 20 can be derived from the trace and they capture the traffic arrival pattern given by the trace. Note that here we restrict each arrival to be made up of a combination of three line segments in order to simplify the computations involving these curves. However, in general they can be arbitrarily complex to capture the exact details of a trace. But, this increased generality comes at the cost of increasing the computational complexity. As mentioned in Section 3.3.6 the analytical model composes the different component models, resulting in a *scheduling network*. For the architecture we study here (Figure 6), the corresponding scheduling network is given in Figure 15.

3.5.1.3 Evaluation Results

Tables 4 and 5 give the utilisation values of the three buses (the OPB, and the PLB read and write bus) when the model is fed with synthetic traces consisting of fixed sized packets. Here, six different packet sizes have been used, from 64 Bytes to 1500 Bytes. For each packet trace and bus combination, the table compares the results obtained from the analytical method with those resulting out of simulation for different line speeds. To give an impression of how the utilisation of the different buses increase with the line speed for the same packet size, in Figure 21 we plot the utilisation values for the trace containing 512 Bytes sized packets. As can be seen from Table 4 and 5, the results for the other traces are very similar, and hence we do not plot them.

There are two things to be noted from these values. First, with increasing line speeds (and therefore more packets per time unit to be processed), the utilisation of the different buses also increase, and as expected, this increase is proportional to the increase in the line speed. Second, the results from the analytical method and the simulation match very well for the utilisation. In Figure 21, for each line speed there are three bars, each corresponding to the OPB, the PLB read and the PLB write bus. From the figure can be seen that the maximum line speed that this architecture can sustain is in the range of 400 Mbps and the OPB acts as a bottleneck.

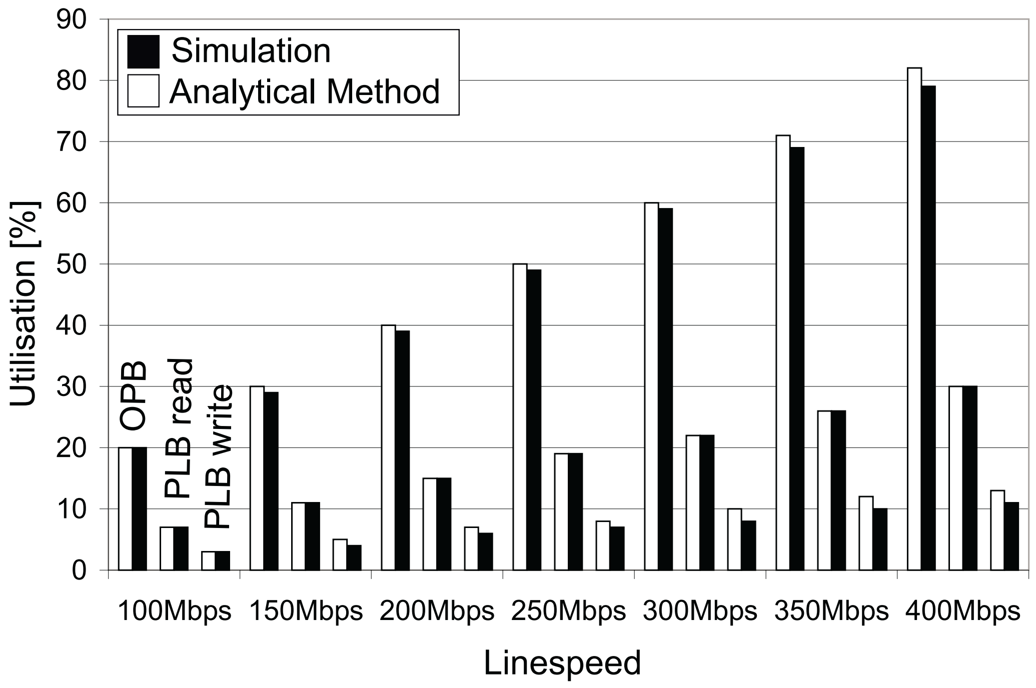


Fig. 21: Utilisation values for different line speeds for the trace containing 512 Bytes sized packets. In this bar graph, for each line speed, the first bar indicates the utilisation of the OPB, the second bar shows the utilisation of the PLB read bus and the third bar corresponds to the PLB write bus. For each bus, the white bar gives the result computed by the analytical method, and the black bar gives the result obtained from simulation.

In Figure 22 we show the utilisation values of the PLB read bus for

Packet Size 64 Bytes

	OPB		PLB read		PLB write	
	AnM	Sim	AnM	Sim	AnM	Sim
100 Mbps	18	18	14	13	6	5
150 Mbps	27	28	20	19	9	8
200 Mbps	36	37	27	25	12	10
250 Mbps	45	46	34	31	15	13
300 Mbps	54	55	41	37	17	15
350 Mbps	63	65	48	40	20	17
400 Mbps	72	76	55	47	23	20

Packet Size 128 Bytes

	OPB		PLB read		PLB write	
	AnM	Sim	AnM	Sim	AnM	Sim
100 Mbps	19	19	15	14	5	5
150 Mbps	29	28	22	21	7	7
200 Mbps	38	38	30	28	10	10
250 Mbps	48	47	37	35	12	12
300 Mbps	57	56	45	42	15	15
350 Mbps	69	66	52	48	17	16
400 Mbps	79	75	59	53	20	18

Packet Size 512 Bytes

	OPB		PLB read		PLB write	
	AnM	Sim	AnM	Sim	AnM	Sim
100 Mbps	20	20	7	7	3	3
150 Mbps	30	29	11	11	5	4
200 Mbps	40	39	15	15	7	6
250 Mbps	50	49	19	19	8	7
300 Mbps	60	59	22	22	10	8
350 Mbps	71	69	26	26	12	10
400 Mbps	82	79	30	30	13	11

Tab. 4: The utilisation values of the OPB, and the PLB read and write buses, when the model is fed with three different synthetic traces consisting of fixed sized packets (ranging from 64 to 512 Bytes). For each trace and for each bus, the first column (marked as AnM - analytical method) gives the results obtained using the analytical model, and the second column (marked as Sim) gives the results obtained by simulation.

Packet Size 1024 Bytes

	OPB		PLB read		PLB write	
	AnM	Sim	AnM	Sim	AnM	Sim
100 Mbps	20	20	6	6	3	2
150 Mbps	30	30	9	9	4	4
200 Mbps	40	40	12	12	6	5
250 Mbps	50	50	15	15	7	6
300 Mbps	60	59	18	18	9	7
350 Mbps	71	69	21	21	10	8
400 Mbps	83	79	25	24	12	9

Packet Size 1280 Bytes

	OPB		PLB read		PLB write	
	AnM	Sim	AnM	Sim	AnM	Sim
100 Mbps	20	20	6	6	3	2
150 Mbps	30	30	9	9	4	4
200 Mbps	40	40	12	12	6	5
250 Mbps	50	50	15	15	7	6
300 Mbps	60	60	18	18	9	7
350 Mbps	71	69	20	21	10	8
400 Mbps	83	79	23	24	12	9

Packet Size 1500 Bytes

	OPB		PLB read		PLB write	
	AnM	Sim	AnM	Sim	AnM	Sim
100 Mbps	20	20	6	6	3	2
150 Mbps	30	30	9	9	4	4
200 Mbps	40	40	12	12	6	5
250 Mbps	50	50	14	15	7	6
300 Mbps	61	60	17	18	9	7
350 Mbps	72	70	20	21	10	8
400 Mbps	83	80	23	24	12	9

Tab. 5: The utilisation values of the OPB, and the PLB read and write buses, when the model is fed with three different synthetic traces consisting of fixed sized packets (ranging from 1024 to 1500 Bytes). For each trace and for each bus, the first column (marked as AnM - analytical method) gives the results obtained using the analytical model, and the second column (marked as Sim) gives the results obtained by simulation.

fixed line speeds. For a fixed line speed, as the packet size is increased, the fraction of the utilisation that comes from the packet traversal increases, since there is less total inter-frame gap in the whole trace (assuming that the trace size in Bytes remains the same). This is because the number of packets in the trace decrease.

However, the number of buffer descriptor and packet header traversals also decreases and therefore the fraction of the bus utilisation that is caused by this also decreases. These effects can be seen in Figure 22. As the packet size is doubled from 64 to 128 Bytes, the first component mentioned above plays a dominating role and hence the utilisation slightly increases. Thereafter, the effect of the second component dominates and the utilisation falls, and then remains almost constant since there is no significant change in the number of packets as the packet size is increased from 1024 to 1280 Bytes and then from 1280 to 1500 Bytes. The same results for the OPB is shown in Figure 23. However, in this case the utilisation increases with increasing packet size because these are no packet header or buffer descriptor transfers over this bus.

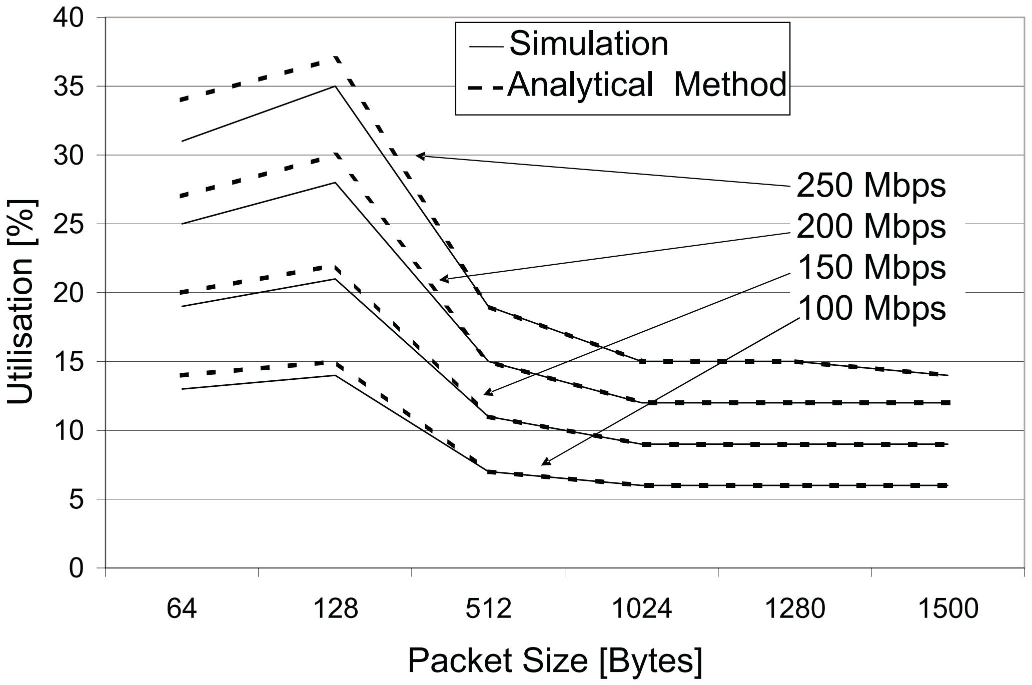


Fig. 22: The variation of the PLB read bus utilisation with increasing packet size for four different line speeds.

It is to be noted that the match between the analytical results and the simulation is always close enough to deduce the above conclusions from the analytical results itself (with significant savings in evaluation time).

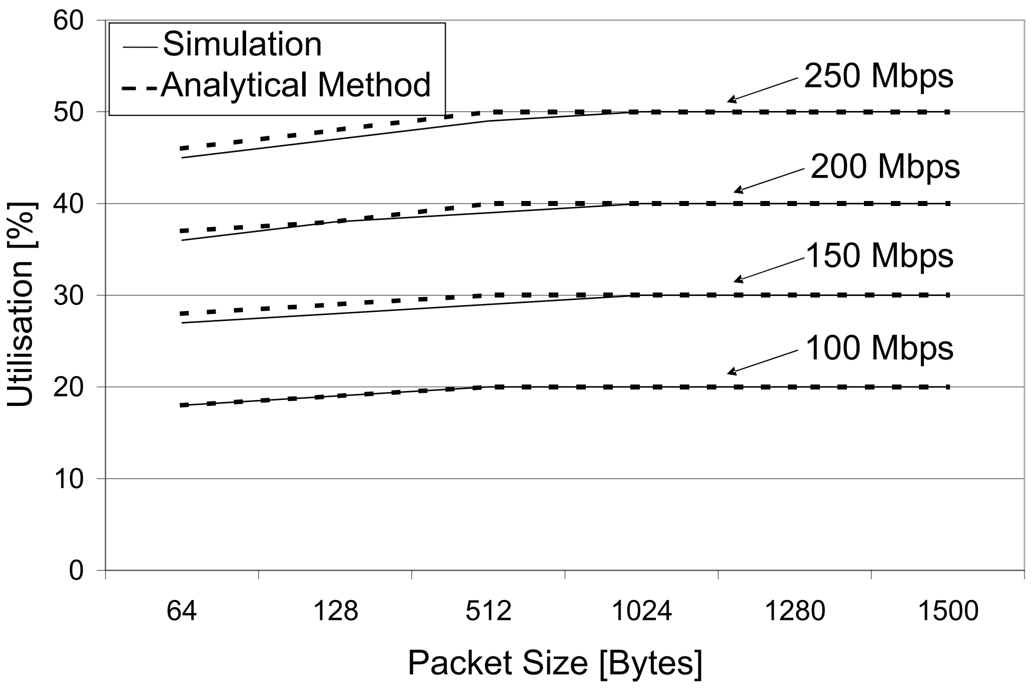


Fig. 23: The variation of the OPB utilisation with increasing packet size for four different line speeds.

For the fixed size packet traces, we do not consider the end-to-end packet latencies and the memory fill levels, since for all low load situations they remain constant and do not depend on the input line speed.

Next we consider the results generated by real traces obtained from the National Laboratory for Applied Network Research (NLANR) [LNA]. Use three different traces—FL (traces from a number of Florida universities), SDC (traces collected from the San Diego Supercomputer Center) and TAU (traces from the Tel Aviv University). Each trace is made up of traffic patterns for two different lines and these are fed into the two EMACs in our architecture). The main motivation behind using these traces is to see the effect of real-life packet size distributions on the architecture. The line speeds used for all the traces vary from 100 Mbps to 400 Mbps as before.

Figure 24 shows the variation in the utilisation of the three different buses for the different line speeds for the FL input traces. We do not give the results for the other traces, because they lead to similar results as the FL traces. In other words, the architecture behaves almost identically for the different traces. It may be noted that, as before, there is a close match between the results from the simulation and the analytical framework.

Recall that the bus arbitration mechanism used in our reference archi-

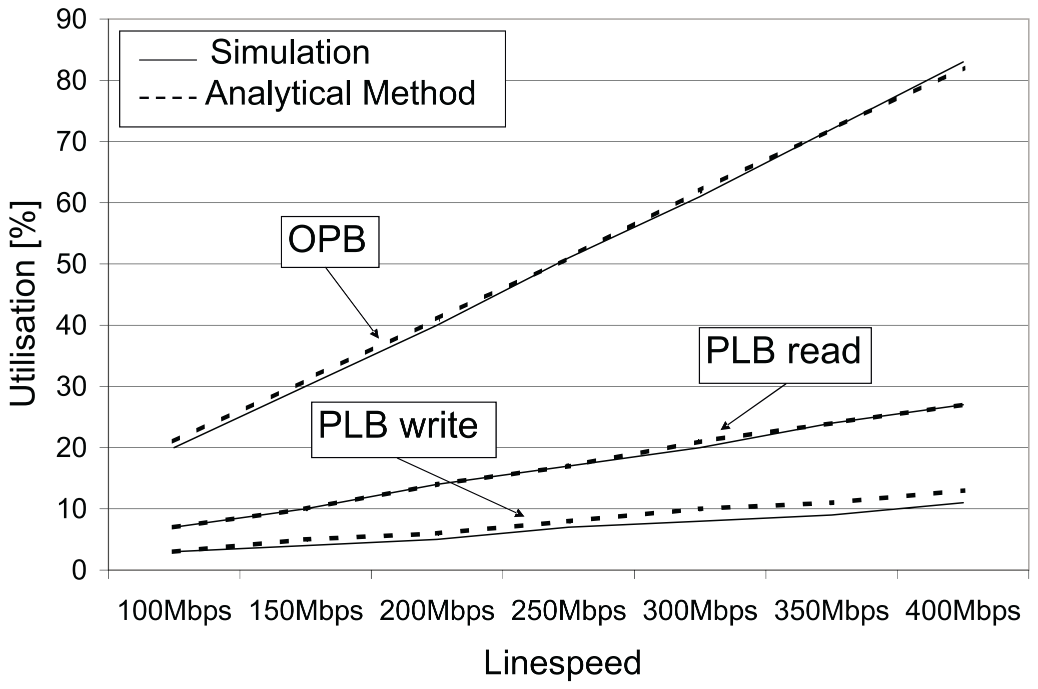


Fig. 24: The utilisation of the OPB and the PLB read and write buses under different line speeds for the FL packet trace.

tecture is always first-come-first-serve (FCFS). Unfortunately, for FCFS there does not yet exist tight bounds for delay and backlog in the analytical framework that we consider here (there exist tight bounds for static priority, round-robin, time division multiplexing, etc.). To get around this problem, we use fixed priority based arbitration mechanisms in the analytical model and compare them with FCFS used in the simulation. Towards this, one of the packet flows (recall that each trace is made up of two flows) in a trace is assigned a higher priority over the other in all the buses. For computing the end-to-end packet latency, the maximum delay experienced by the lower priority flow now gives an upper bound on the maximum delay experienced by any packet when FCFS is used. Similarly, we use the maximum delay experienced by the higher priority flow as a lower bound on the maximum delay experienced by any packet in the case of FCFS. These results are shown in Figure 25 for the FL trace. Note that again, for all the three traces, the results are similar, and we therefore only give the figure for the FL trace. The delay values obtained through simulation lie in between the delay values (obtained from the analytical method) for the high and the low priority flows. These results indicate that the architecture is sufficiently provisioned for one flow, even for high line speeds, since the maximum delays experienced by packets from the high priority remain constant with increasing line speeds for all the three buses. For the low priority flow, as expected, the delay values increase with increasing line speeds. When FCFS arbitration policy is used, the delays suffered are more than those suffered by packets from the high priority flow, but less than those suffered by the low priority flow.

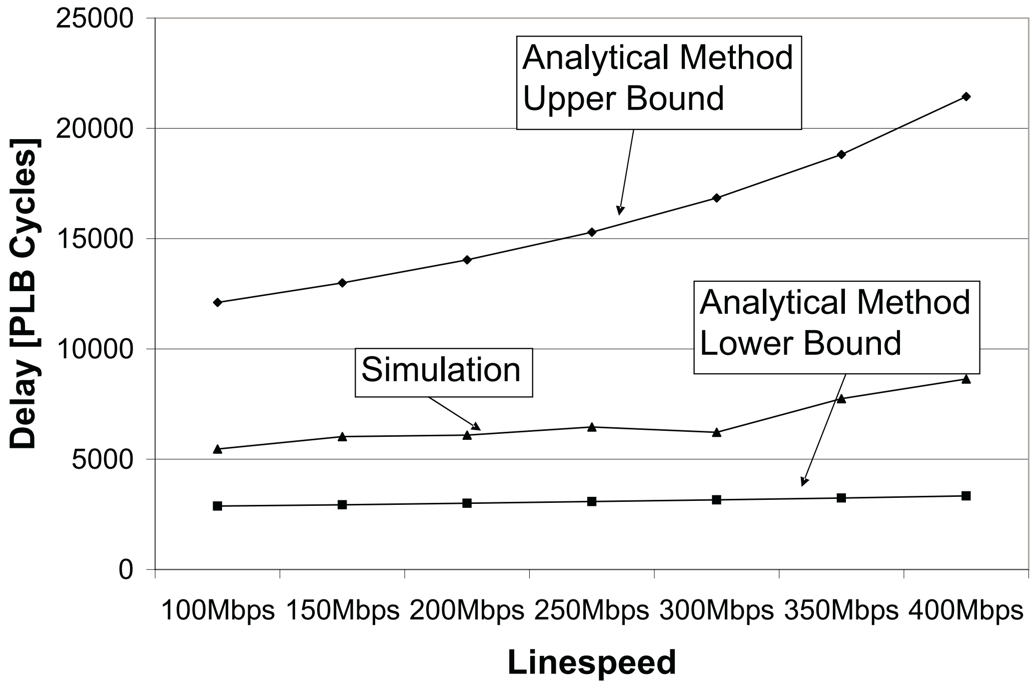


Fig. 25: The maximum end-to-end delays experienced by packets of the FL trace under different line speeds. For the two flows that make up this trace, the analytical results show the delay experienced by the higher and the lower priority flows when using priority based arbitration at the buses. The simulation results are based on FCFS implemented at all the buses.

In Figure 26, for each trace we first assign a high and a low priority to the two flows and measure the maximum delay experienced under this priority assignment using the analytical method. Then we reverse this priority assignment and again estimate the maximum delay, and finally average the two maximum delays for each flow. Figure 26 shows this averaged maximum delay (we choose the flow for which this averaged maximum delay has a higher value) for the analytical method and the simulation results, that are based on FCFS at all the buses, as before.

Lastly, Figure 27 shows the on-chip memory requirements (measured in terms of the backlog) obtained by the analytical method and by simulation. In the analytical case, as before, we use priority based arbitration at the different buses and the simulation results are based on FCFS.

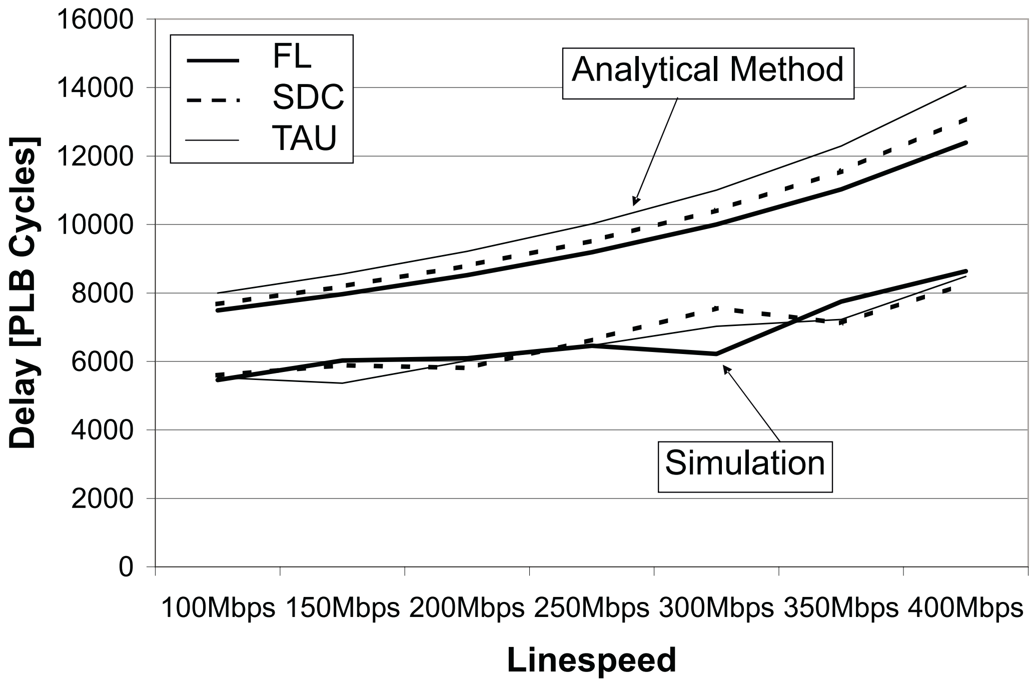


Fig. 26: The maximum end-to-end packet delays experienced by packets of all the traces under different line speeds. The plots corresponding to the analytical method shows the average of the maximum delays experienced by packets from the low and the high priority flows, when using priority based arbitration at the different buses. The simulation results are based on using FCFS at all the three buses.

It may be noted that although the analytical and the simulation results for the end-to-end packet delays and the memory fill levels do not match as closely as the results for the utilisation values, the "trends" indicated by both the methods do match. We believe that such trends, to a large extent, would suffice to make the architectural decisions that are involved in any system-level design space exploration. One of the reasons why there is a mismatch between the values obtained by the analytical method and those obtained by simulation is that the former computes worst-case delays and backlogs. In any particular simulation run, packet sequences, representing a worst-case scenario may not occur. Additionally, the results obtained using the analytical framework to a very large extent depend on how tight are the different bounds for calculating the delay, backlog and resource utilisation for the different scheduling policies.

For all the results reported here, the simulations run in time in the order of a few minutes to several hours. In contrast to these, the analytical procedure completes execution in time less than a second for all the traces and is the only feasible option for performance evaluation in any

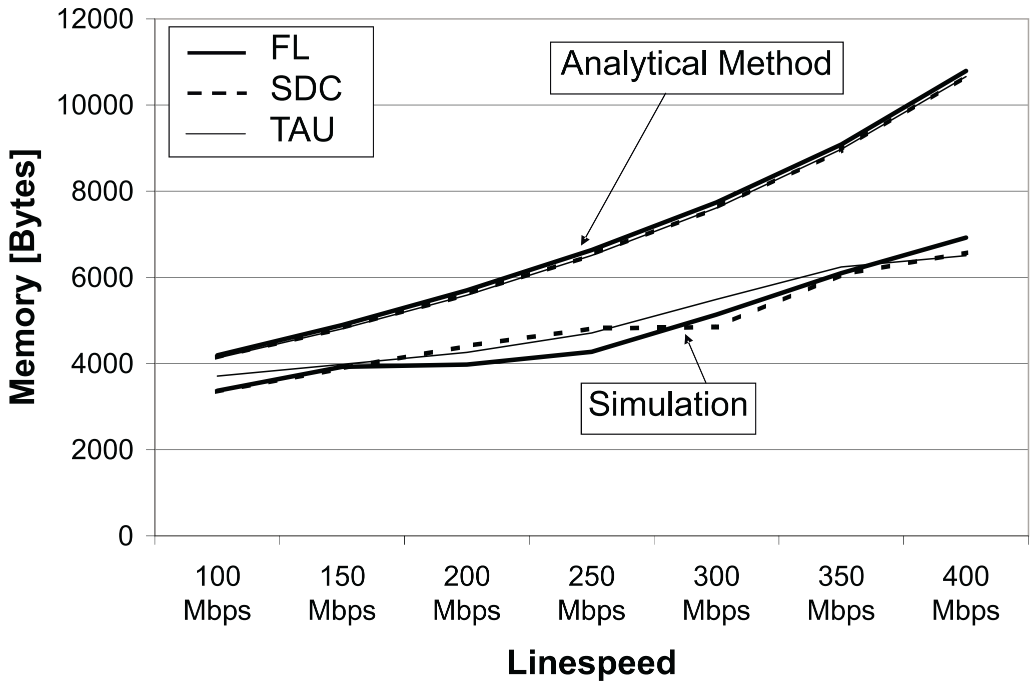


Fig. 27: The buffer memory requirements/memory fill levels for the different traces under different line speeds.

automated design space exploration process.

3.5.2 Example System 2: Multiprocessor Platform

In this section, we present the results for a second comparative study. We compare the results obtained by Real-Time Calculus on the one hand with the results obtained with simulation on the MPARM platform (cf. Section 3.4.2) on the other hand, for the same application. This comparison shows that the obtained results match well, if there exists a good characterisation of the system for both the performance evaluation methods.

In Figure 29, we give the results obtained for the multistage matrix multiplication application that was briefly introduced in Section 3.1.2. It is organised as a pipeline, with 8 consecutive tasks of different worst-case execution times to be executed for the application to complete. The execution platform used for the comparison consists of 4 ARM7 processors connected with a AMBA bus. The data transfer between the tasks is done using a shared memory attached to the bus. We modelled two scenarios with a different mapping of the tasks to the resources (see Figure 8).

Since the execution costs of the tasks are not equal we have exploited two different optimisation approaches. For the first, we concentrate some critical tasks on the same core in order to try to reduce the concurrent bus accesses performed by them. While in the second one we balanced the workload on the processors as much as possible. These two methods can, ideally, both give good throughput at the output, so it's important to understand which one of the two to adopt in order to reach the desired performance.

The approach taken to compare the results is depicted in Figure 28. We first defined an input sequence for the simulation (1). From this sequence with which the first stage of the matrix multiplication pipeline is triggered, we computed the upper and lower arrival curves which are used for the formal analysis method (2). We then performed the simulation of the system and the analysis (3), and calculated the arrival curve from the simulation output (4) in order to be able to compare the two approaches (5).

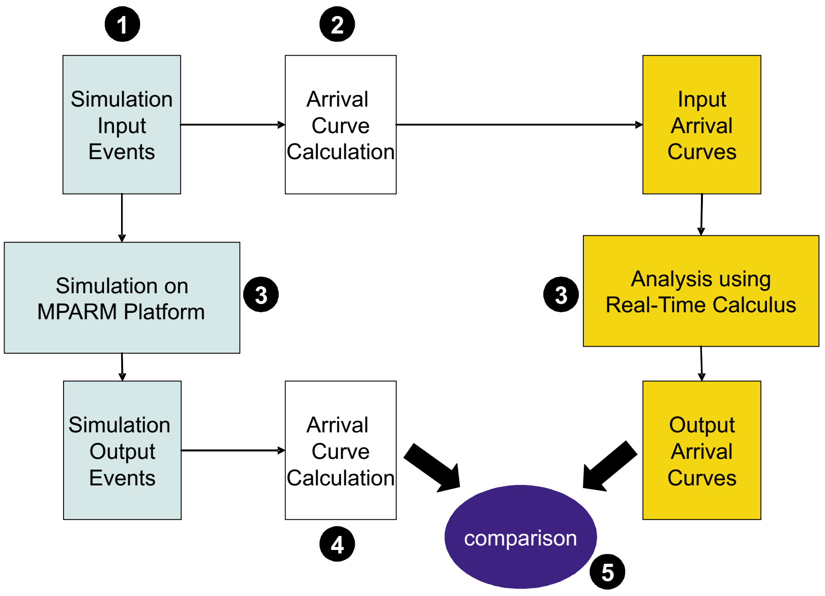

1
2
Simulation Input Events
Arrival Curve Calculation
Input Arrival Curves
Simulation on MPARM Platform
3
3
Analysis using Real-Time Calculus
Simulation Output Events
Arrival Curve Calculation
Output Arrival Curves
4
comparison
5

Fig. 28: Procedure for the comparison of the analytical approach and the simulation approach.

The diagrams in Figure 29(top) depict the output arrival curves for the two scenarios. The curves show on the one hand the arrival curves describing the worst-case output traffic as predicted by Real-Time Calculus and on the other hand they give the arrival curves describing the output traffic generated by the simulation. As the modular performance analysis method is based on worst-case analysis, the curves resulting from the analytical method lie outside of the curves based on simulation, as expected. This can be interpreted as follows: In case of simulation for scenario 1, the

Sc.	Method	Run-Time [s]
1	RTC	0.965
1	Simulation	43–321
2	RTC	0.984
2	Simulation	42–384

Tab. 6: Run-times of evaluation methods for the 2 different scenarios.

smallest time interval where there are 4 events is 6 ms, the largest interval is 9.9 ms for the used trace. For Real-Time Calculus, the smallest interval is 5 ms, the largest interval is 10 ms.

Figure 29(bottom) shows that the trends for the output traffic pattern for the two scenarios that are shown by both Real-Time Calculus (left) and the simulation (right) match well. This result allows us to claim the ability of Real-Time Calculus to predict correctly the behaviour of a component.

The run-times of the two approaches for the example application are shown in Table 6. Real-Time Calculus uses less than a second to complete whereas the corresponding simulation framework takes between 40 seconds an around 6 minutes to complete dependent on the waiting time between consecutive activations of the matrix multiplication. This gain in evaluation time comes at the cost of a pessimistic prediction of task execution, and therefore resource consumption.

The match between the results obtained by the analytical method and the simulation based approach is not always as good as for the matrix multiplication application in the given example. In some cases, the systems can only be characterised less accurate for the formal method and hence the results of the two approaches differ a lot. If caches and therefore less predictable execution times of the tasks are involved, the analytical method provides overly pessimistic results. In some cases, there even exists no formal model for a system's component at all that could be analysed using Real-Time Calculus. To overcome some of the inaccuracies, extensions of Real-Time Calculus exist, e.g. a method to cope with caches is described [WMT04].

3.5.3 Concluding Remarks

We presented (1) a detailed comparison study of an analytical performance evaluation framework for network processor architectures with a simulation based technique, and (2) a comparison between the analytical method and simulation for a multiprocessor platform. The underlying assumption in both studies has been that there is a high confidence in the simulation results. But obtaining these results is time intensive because of the high simulation times involved, and the usually long development

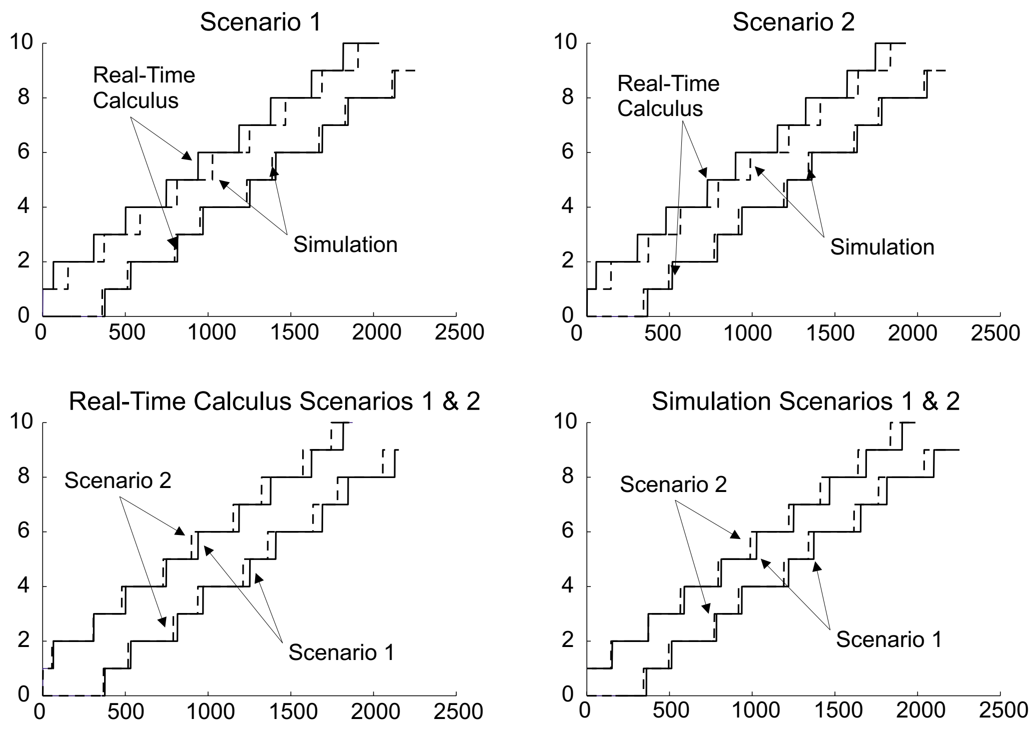


Fig. 29: **top:** Comparison between arrival curves computed from simulation (dashed) and arrival curves computed with Real-Time Calculus (solid) for the two scenarios.

bottom: Comparison between the two scenarios for the analytical method (left), and the simulation (right). The units on the x-axis are cycles, for this experiment 1 cycle $\hat{=}$ $10\mu s$.

times for simulation frameworks. Thereby simulation-based models are inappropriate for early stages of automated design space exploration. The main contribution of this section is a validation of the analytical model against simulation results. We believe that such cross-checking is required to establish the usefulness of analytical models, and also helps in identifying the appropriate design phase where such models can be used and where it is necessary to use detailed simulation to obtain meaningful results.

We also believe that the two models considered here lie at two different extremes of a spectrum of possibilities. For evaluating different aspects of an architecture, different models might be more suitable. We envisage a suitable combination of analytical and simulation based frameworks not only across different abstraction levels of a design flow, but also within the same abstraction level, for evaluating embedded system architectures. For example, it might be more suitable to use simulation to evaluate the cache/memory subsystem of an architecture, while it might be sufficient to use an analytical model to evaluate the on-chip communication architecture. The essential ingredients for such a hybrid framework in the context of network processors are already available. In the next section, we will discuss the steps that have to be performed to meaningfully combining the approaches.

3.6 Combination of Simulation with Formal Method

In this section, we present a new method for performance analysis of embedded systems. Normally, formal or analytical methods as the one presented in Section 3.3 are used for early-phase design evaluation. But if there exist no formal component models with the required precision, simulation-based approaches are used for system-level performance analysis. The often high run-times of simulation runs lead to the new approach described in this section: Analytical methods are combined with simulation-based approaches to speed up simulation. Simulation as well as a formal analysis method are used to evaluate a part of the complete system. We describe in this section how the simulation models can be coupled with the formal analysis framework, specify the interfaces needed for such a combination and show the applicability of the approach using a case study.

Our new method reduces the run-time of an evaluation and reflects the idea of components. In analogy to component-based design where existing IP blocks are combined to form a System-on-Chip, our method allows the designer to reuse existing analysis models for components – be it a simulation model or a formal model – for the individual components and compose them to form the complete system model. The existing component models may result from previous designs or may be delivered by IP vendors. Performance evaluation is then conducted using these trusted models of the components. The contribution of this section includes the definition of a new hybrid approach for performance evaluation. In particular, the definition of the needed interfaces is provided in the next section. These interfaces were implemented, and the new method is used in a case study presented in Section 3.6.3.

3.6.1 Interfaces between Simulation and Formal Method

In this section we will introduce the interfaces needed to combine system simulators with formal analysis models for a hybrid analysis.

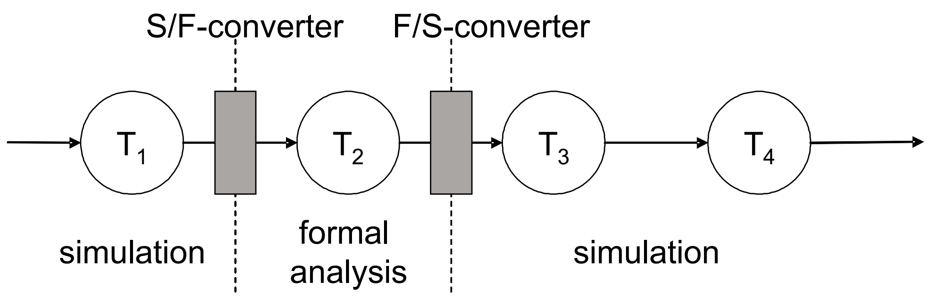


Fig. 30: The task chain for hybrid approach with resource-independent components.

Figure 30 gives an example system model for the hybrid approach. We can identify two problems, (1) the interface from simulation to formal models and (2) the interface from formal analysis models to simulation. In order to introduce these interfaces more formally, we define what we understand by an *event trace* and an *event class*.

Def. 1: ***Event trace.*** *An event trace is a sequence of events* (e_i, t_i)*, where* e_i *denotes the event data and* t_i *the time stamp at which the event data* e_i *is available to the application. The set of all events is denoted as E, i.e.* $(e_i, t_i) \in E$.

Def. 2: ***Event class.*** *An event class is formed by events that have to be processed in the same way by an application. Events belonging to the same event class take the same path through an application task graph. If* $(e_j, t_j) \in E_i$*, then event* (e_j, t_j) *belongs to an event class* E_i.

3.6.1.1 S/F-Converter

The conversion of event streams from the simulation subsystem into an event model for the formal analysis method appears to be much simpler than the reverse direction. Once the simulation of a component has finished, we can analyse the output event traces of the form $(e_1, t_1), (e_2, t_2), \ldots, (e_i, t_i), (e_{i+1}, t_{i+1}), \ldots$. First, we have to classify the events in the event trace and annotate them with the event class they belong to. In the next step we can derive the upper and lower event arrival curve that represent each event class E_k with:

$$R(t) = |\{(e_i, t_i) \in E_k : t_i \leq t\}|$$

From $R(t)$, we can then compute α^l, α^u with the following equations:

$$\alpha^l(\Delta) = \min_{\lambda \geq 0}\{R(\Delta + \lambda) - R(\lambda)\}$$

$$\alpha^u(\Delta) = \max_{\lambda \geq 0}\{R(\Delta + \lambda) - R(\lambda)\}$$

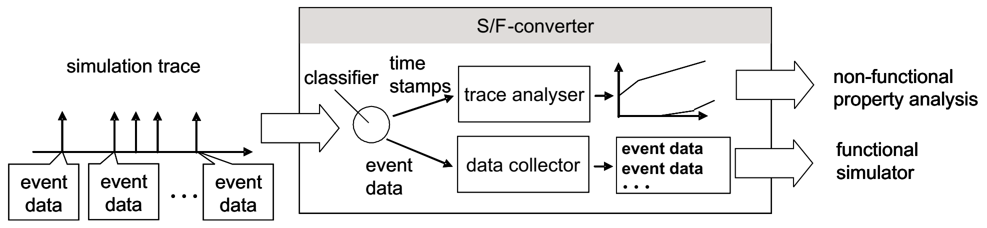


Fig. 31: Interface between formal analysis method and simulator: the S/F-converter.

The event classification and the arrival curve calculation are performed by the *classifier* and the *trace analyser* shown in Figure 31. The event data items e_i are sorted according to the event classes by the *data collector*. The output from the trace analyser, the arrival curves describing the timing of the event stream are then passed to the formal analysis method, whereas the event data collection is passed to the functional simulator for further processing.

3.6.1.2 F/S-Converter

An F/S-converter transforms event models that result from the formal analysis method into event traces used for the simulation. This conversion tool from the formal method to simulation is more involved than the S/F-converter introduced in the previous section. In our setup, with Real-Time Calculus as formal method and the MPARM simulation framework written in SystemC [GLMS02], the problem of designing an F/S-converter (as shown in Figure 32) can be seen as designing a SystemC module, which generates events according to the arrival curve that was obtained using Real-Time Calculus.

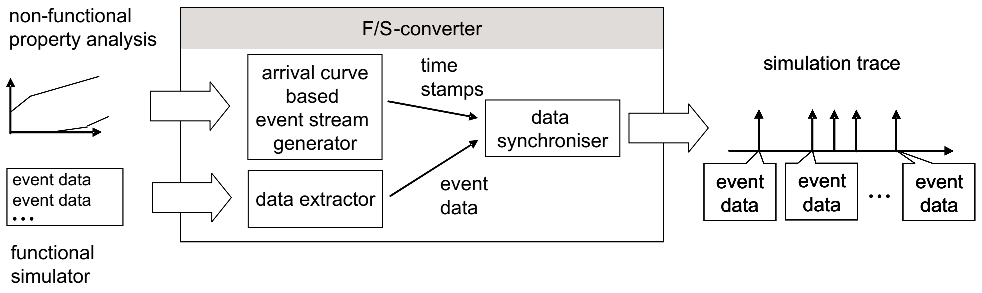


Fig. 32: Interface between simulator and a formal analysis method: the F/S-converter.

The method to determine the time stamps for the event traces, a discussion of the requirements and an indicator to assess the quality of the generated trace is given in Section 3.7. Additional information needed for the simulator besides the time stamp, such as the event type, which may not be included in the analytical description of the event stream, has to be passed to the *data extractor*. In the *data synchroniser*, the event time stamps and the corresponding event data are synchronised and used to trigger the simulator. In the case study presented in the next section, we use a traffic generator written as a module in SystemC to feed the data into the simulator, at times determined by an event trace generator based on the arrival curves obtained by the formal analysis method. The traffic generator module is described in more detail in [MAS+05].

3.6.2 Benefits of hybrid approach

The hybrid approach presented in this section can be used to analyse applications that consist of task chains. It is possible to cut this chain into segments of the chain that are executed on independent hardware resources. These segments can then be either analysed using the formal method, and if not applicable analysed using a simulator (cf. Figure 30).

Using the hybrid approach, we can lower the overall execution time compared to simulation because of two reasons: (1) The run-time of a single evaluation for the hybrid approach is significantly smaller than with simulation, because we replace individual simulator components by formal models. (2) The F/S-converter constructs short, representative traces for simulation from the formal model. As a consequence, less simulation runs are needed for a good coverage.

Assume that for the task chain given in Figure 30, we have to perform n pure simulations to well cover all possible load scenarios. For the hybrid approach, we still have to perform n simulation runs for the components before the S/F-converter, the converter then aggregates the n simulation traces to a single pair of arrival curves, representing the n traces. The analysis has to be performed only once and the output of the formal analysis, a simulation trace generated by the F/S-converter has also to be simulated only once, as the trace is generated based on the aggregation of all input traces.

The analysis method as it is presented in this section can handle feed-forward data flow graphs, without feedback loops across the borders between the different analysis methods. Further, data splits or joins are restricted to occur in simulation components due to limitations of the formal analysis method used. These usage restrictions shall be tackled in future work.

3.6.3 Example System 2: Multiprocessor Platform

For the case study, we analysed a GSM audio encoder application. The task graph of the application consists of a chain of 21 consecutive tasks.

It receives sequences of frames as input that have to be encoded before being transmitted. These input frames are guaranteed to respect certain best/worst case bounds, while the encoded sequences have also to respect bounds in order to guarantee a good communication. We first specified the load traffic that should be supported by the GSM encoder application by means of an arrival curve describing upper and lower bounds on the arrival of packets to be processed by the encoder. After a static profiling of the application we partitioned the task chain to be executed on two processors to obtain a balanced system. The two processors are connected with a AMBA-bus and communicate over a FIFO-queue located in a shared memory attached to the bus.

We performed 3 experiments for the analysis of the system: (1) We used only Real-Time Calculus, (2) we used the hybrid approach presented in this section, and (3) we performed a full system simulation. For all the 3 experiments we used the same input load, either event traces (for the simulation) or the corresponding input arrival curve (for the hybrid approach and the formal analysis). The input load specifies the arrival of frames to be encoded.

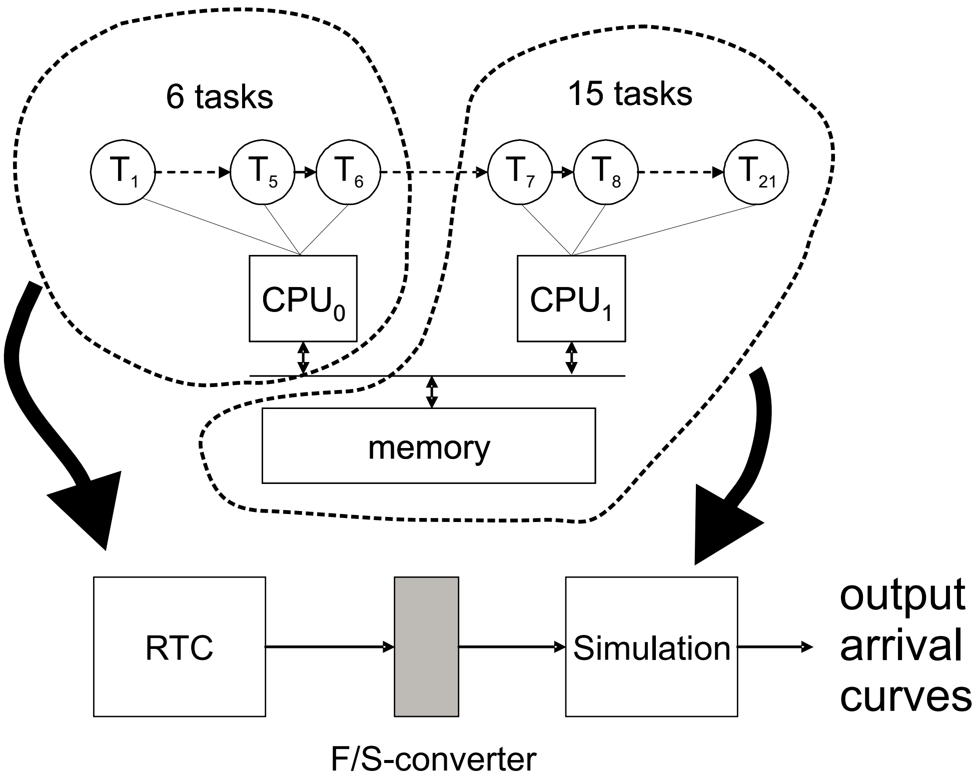


Fig. 33: System model for experiment 2 which is evaluated using the new hybrid approach.

For the hybrid approach, we partitioned the system as shown in Figure 33. As described in Section 3.6.1.2, we use an F/S-converter to interface between Real-Time Calculus and the simulator. We replaced one ARM7 processor by its formal model and integrated a traffic generator into the MPARM simulation framework. The traffic generator puts the intermediate data collected from a functional simulation of the GSM encoder application into the FIFO-queue.

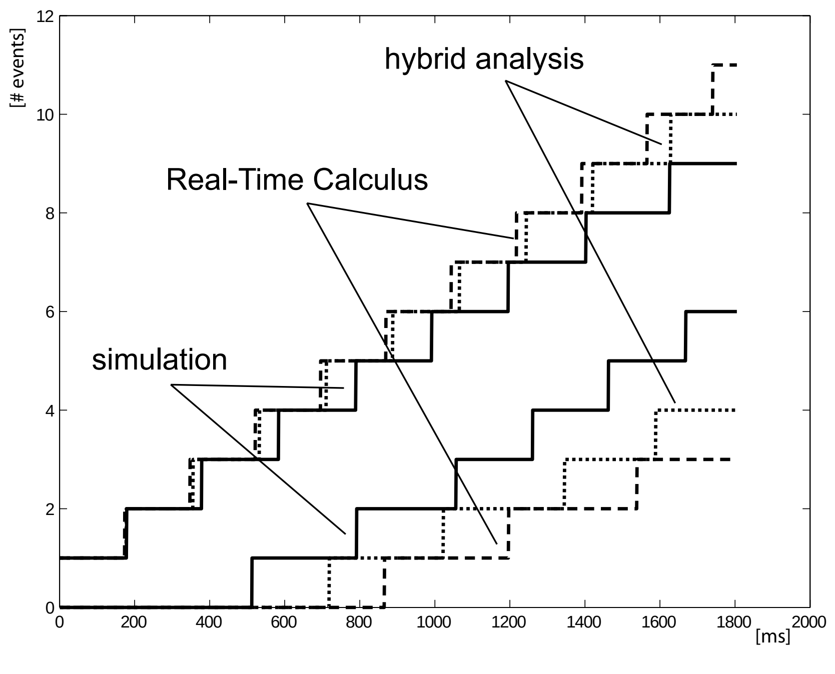


Fig. 34: Output arrival curves for the 3 experiments.

The curves presented in Figure 34 show the output arrival curves for the 3 experiments. The curves give upper and lower bounds on the number of audio frames that are finished processing in any time interval. The upper and lower output curve were calculated for experiment 1 using Real-Time Calculus, and obtained from the output simulation traces in case of the hybrid approach (experiment 2) and the simulation (experiment 3) using the same procedure as described in Section 3.6.1.1.

The outermost curves give the upper and lower arrival curve calculated with Real-Time Calculus. Real-Time Calculus is a method for worst-case analysis, i.e. the other curves *have* to lie within the bounds obtained with this method. The curves for the hybrid approach lie between the curves for Real-Time Calculus and the simulation curves. This behaviour of the hybrid approach is also expected, as the first part of the analysis was performed using a worst-case analysis method, and the synthetic trace used as stimulation for the second part is based on the worst-case arrival curves.

We now look at the fill level of the intermediate buffer between the two processors. In case of Real-Time Calculus we predict that the buffer fill level is at most 5 frames waiting to be processed. For the pure simulation, the maximum buffer fill level varies between 1 and 4. Dependent on the simulation trace used, we derive different design values for the queue size needed (see Figure 35). In contrast, for the hybrid analysis run, we can see that the buffer fill level is at most 4 frames waiting in the queue with only a single simulation.

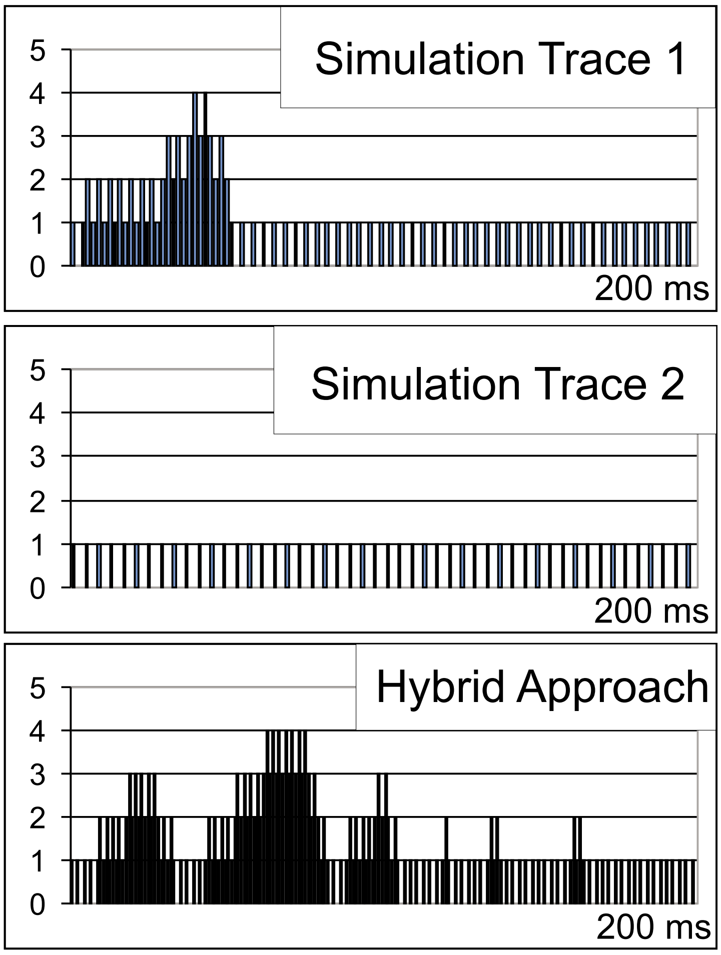


Fig. 35: Buffer fill levels over time for 2 simulation runs and the hybrid approach.

Table 7 gives the run-times for the three different experiments. The times are given for a single evaluation run. The hybrid approach allows to speed up the simulation by a factor of 1.73 for our example. This is a

Exp.	Method	Run-Time [s]
1	RTC	0.273
2	Hybrid	292
3	Simulation	508

Tab. 7: Run-times of evaluation methods for a single run of the GSM encoder.

significant improvement, because we still simulate more than half of the system (cf. Figure 33). Using the new method described in this section, we could (1) speed up the simulation for a single evaluation run and (2) lower the number of simulation runs needed for a complete system evaluation.

3.7 Event Trace Generation

A core element of the hybrid approach presented in the last section is the generation of event traces based on arrival curves as used in the F/S-converter (see Section 3.6.1.2). In this section we describe a method to generate a representative and expressive event trace starting from an upper and a lower arrival curve $[\alpha^l(\Delta), \alpha^u(\Delta)]$. Figure 36 shows as an example the first part of a trace that was generated using the approach presented in the remainder of this section. The upper and lower arrival curves that were used as specification curves for this generation are given in Figure 41 (top,left). In Figure 36 we can identify bursts (marked with (a)), where the generated trace is as bursty as specified by the upper arrival curve. But we can also identify periods in which only a few events are generated (marked with (b) in the figure). In this case, the generated trace represents the lower specification arrival curve.

The generated event stream must respect the specification arrival curves $[\alpha^l(\Delta), \alpha^u(\Delta)]$. But we also need to define the meaning of 'expressiveness' which is done in the next section.

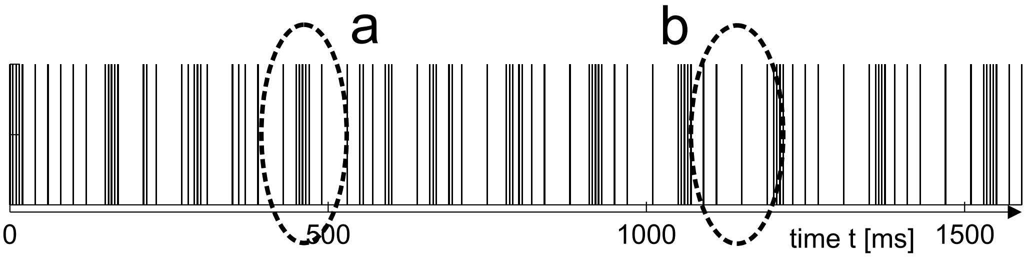


Fig. 36: Generated trace based on the arrival curves for specification 1 in Figure 41.

3.7.1 Requirements and Quality Assessment

It is obvious, that the expressiveness of an event stream very much depends on the further use of it. In simulation and measurements, we are interested in corner case behaviour. Therefore, it appears to be appropriate to require that the event stream follows the short term characteristics (bursts), i.e. $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ for small values of Δ, as well as the long term characteristics (average case) for large values of Δ. In this sense, the generated trace should show "fractal" or self-similar behaviour [LTWW93], i.e. in a short observation interval, the trace should be as bursty as allowed by the upper and the lower specification curve in a short time interval — whereas for a larger observation interval, the trace should represent the whole upper and lower specification curves.

In addition, as the behaviour of different event streams interact on the system under observation, we should require that we find the characteristics of the arrival curves everywhere in the stream. For example, we would expect that bursts or silent intervals occur frequently in the generated event trace. Therefore, we are looking for a new quality indicator I that covers the property that the multi-scale representation of arrival curves can be seen frequently in the generated event stream.

To obtain this indicator value, we first calculate the probability P_τ that the measured arrival curves of an arbitrarily selected trace snippet of length τ match the specified curves $[\alpha^l(\Delta), \alpha^u(\Delta)]$ for all $0 \leq \Delta \leq \frac{\tau}{2}$.

To compute this probability we have to traverse the full trace using a sliding window of size τ, then calculate the upper and lower arrival curves from the trace snippet seen in the sliding window and finally check whether these curves $[\alpha^l_c(\Delta), \alpha^u_c(\Delta)]$ equal the given curves $[\alpha^l(\Delta), \alpha^u(\Delta)]$ for all $0 \leq \Delta \leq \frac{\tau}{2}$.

We calculate the probability P_τ for all values of τ up to some observation window size L which can be chosen arbitrarily. The indicator I is then the minimum over all $\tau \leq L$ of these probabilities P_τ as we want to represent the arrival curves at all scales τ.

Formally, we can compute this indicator I with the following steps:

1. Select all trace snippets T_i of length τ in trace T.

2. Compute the upper and lower curve $[\alpha^l_c, \alpha^u_c]$ from each trace snippet T_i.

3. Set $Z(T_i) = 1$ if $\alpha^u_c(\Delta) = \alpha^u(\Delta)$ and $\alpha^l_c(\Delta) = \alpha^l(\Delta)$ for all $0 \leq \Delta \leq \frac{\tau}{2}$. Otherwise set $Z(T_i) = 0$.

4. Compute the probability $P_\tau = \frac{1}{N} \sum_{T_i \in T} Z(T_i)$ where N denotes the number of considered trace snippets T_i in trace T.

5. Set $I = \min_{\forall \tau \leq L} P_\tau$.

The larger the indicator value I, the better the trace exposes the desired fractal behaviour. For the trace generation algorithm presented next we intend to maximise the indicator value I in order to generate a trace that represents the specification in a any short observation interval as well as in any large observation interval.

3.7.2 General Event Trace Generation

The main idea underlying the proposed event trace generation algorithm is to use a ON/OFF traffic source as shown in Figure 37. The traffic generator has two distinct states: ON and OFF. In existing ON/OFF traffic generation approaches (see e.g. [ALM98],[BC98]) events are generated in the ON state, and in the OFF state no events are generated at all. In contrary to these classical approaches, we generate events that conform with the upper specification arrival curve, if the generator is in the ON state. In this case, if we would compute the upper arrival curve for the generated trace, the upper specification curve would be obtained. In other words the generator creates events as soon as it is allowed by the upper specification curve. Similarly, the generator in the OFF state is as “lazy” as allowed by the lower curve, i.e. it generates an event only if the lower specification arrival curve would be violated otherwise. We also say that the generator “follows” the upper curve while it is in the ON state and “follows” the lower curve in the OFF state.

The basic event trace generation algorithm consists of three main steps:

1. Determine time stamp T at which to switch between ON- and OFF-states.
2. Generate events according to the state while $0 \leq t < T$.
3. If $t = T$ then switch the state, set $t = 0$, and go to step 1.

The time t denotes the time spent in a state. In Figure 37(bottom) an event trace generated with this simple algorithm is given, the grey boxes denote the state of the generator.

The choice of the times at which the traffic source switches between states influences the generated trace and as a consequence also the indicator value I for the generated trace. See Figure 38 for two different traces that are based on the same specification arrival curves. The only difference for these traces are the distributions of the switching times between ON and OFF state of the generator. The second trace is more regular, e.g. we can observe the event pattern between 5000 and 8000 ms, during which no state switches occur at all.

We will next look at a simple example which will lead us to a deterministic algorithm to determine the state switching time. Then, we will randomise the algorithm to make the generated trace less predictable.

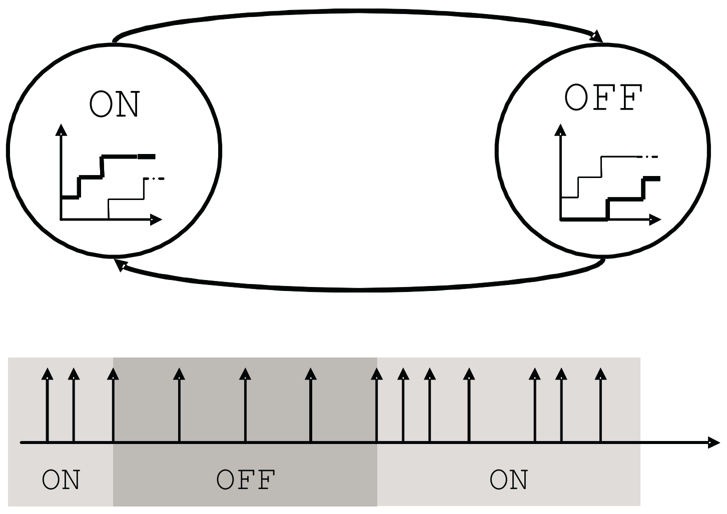


Fig. 37: (top) ON/OFF-automaton used for trace generator. (bottom) Example for a generated trace with boxes indicating whether generator in ON- (light grey) or OFF-state (dark grey).

Assume the upper and lower arrival curve specified in Figure 39(top). For simplicity we only look at three different intervals of length A, B and C with $A = 2B = 4C$. For the example, we use the ON/OFF-automaton from Figure 37, but we restrict ourselves to switches between states after A, B or C time units only. If we are in the ON state for A time units, we denote this by $A \uparrow$, if we are in the OFF state for B time units, we denote this by $B \downarrow$ accordingly.

Furthermore, we assume that after a state change we generate events according to the specification curve, i.e. if we are in the ON state for C time units, we can generate an event sequence that reproduces an upper arrival curve equal to the specification curve up to C time units as shown in Figure 39(top).

In the example, we check the match between the specification curve and the computed arrival curve at discrete times and the step size is set to C. In this setting, the probability to see the upper and lower specification curve in interval $2C$ is the number of times we can see $C \uparrow$ followed by $C \downarrow$ (or vice versa) divided by the total number of checks performed. As we want to achieve a large indicator value I, we have to ensure that for all the different interval sizes (A, B, C) the probability to see the corresponding upper and lower curve is maximised. In other words, we have to generate at least one trace snippet for the upper and the lower specification curve for each of the interval sizes in order to achieve non-zero probabilities for all the intervals. Because we do not know a priori the length of the generated trace, we want to generate all possible trace snippets as soon as possible.

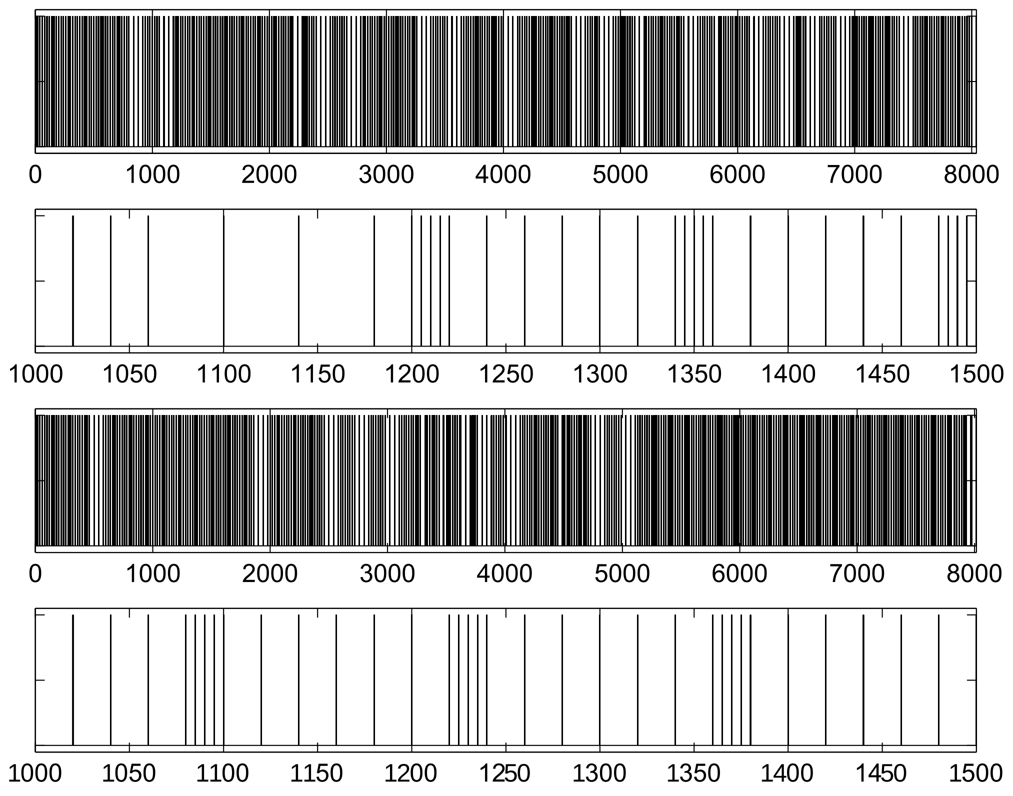


Fig. 38: Two generated traces (plots 1 and 3 from top) based on the same specification. They were generated using a different switching time estimation technique. Plots 2 and 4 from top are magnified snippets of the traces (the traces are shown between 1000 and 1500 ms).

The specification curves shown in Figure 39(top) reveal that the upper curve of length C is contained in the upper curve of length B. This fact helps us to further improve the value of the indicator I: We start the trace generation in the ON state, set the dwell time to the smallest possible interval and generate a trace snippet according to the upper specification curve. In the example, we therefore first generate a trace snippet $C \uparrow$. We then switch state and remain in the OFF state for again C time units, generating a trace snippet $C \downarrow$. Next, we increase the dwell time by the step size, thus in the example we generate a trace snippet $B \uparrow$. We again switch state to OFF, generate the corresponding trace snippet, switch to ON state, increase the dwell time, switch back to OFF state, and so on. Like this, we can not only see the upper and lower specification curve in the the sequence $C \uparrow C \downarrow$, but also in the sequence $C \downarrow B \uparrow$, as the upper curve of length C is contained in the upper curve of length B.

Having the above considerations in mind, we can propose a deterministic algorithm that generates an event trace that leads to a large indicator

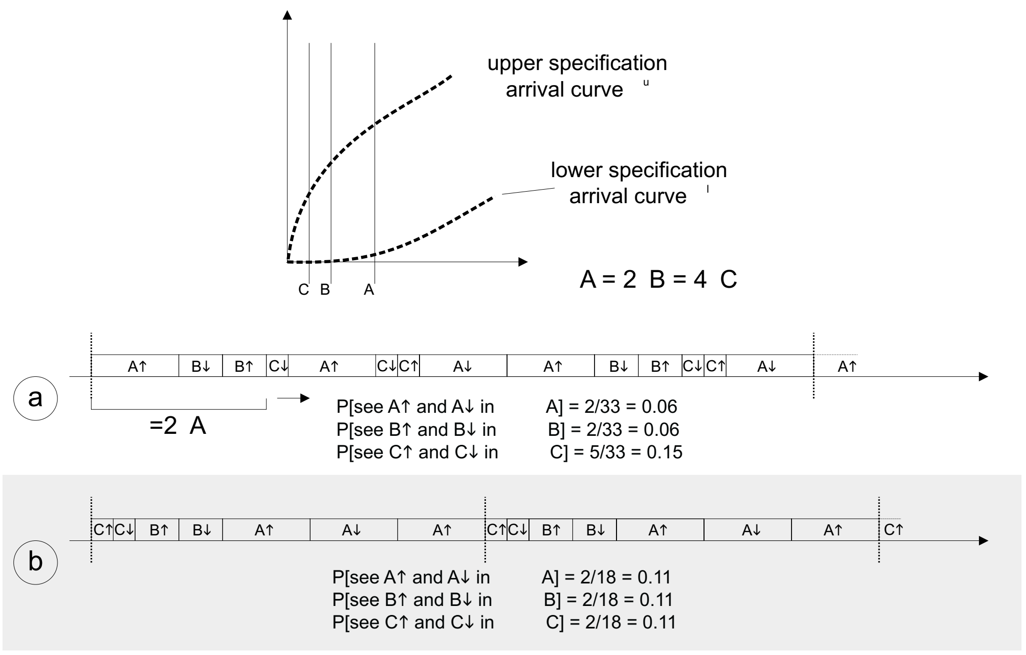

upper specification
arrival curve u
lower specification
arrival curve l
C B A
A = 2 B = 4 C
a
A↑ B↓ B↑ C↓ A↑ C↓ C↑ A↓ A↑ B↓ B↑ C↓ C↑ A↓ A↑
=2 A
P[see A↑ and A↓ in A] = 2/33 = 0.06
P[see B↑ and B↓ in B] = 2/33 = 0.06
P[see C↑ and C↓ in C] = 5/33 = 0.15
b
C↑ C↓ B↑ B↓ A↑ A↓ A↑ C↑ C↓ B↑ B↓ A↑ A↓ A↑ C↑
P[see A↑ and A↓ in A] = 2/18 = 0.11
P[see B↑ and B↓ in B] = 2/18 = 0.11
P[see C↑ and C↓ in C] = 2/18 = 0.11

Fig. 39: Example for upper and lower specification arrival curves and two examples for generated traces with probabilities corresponding to the intervals A, B and C. (a) randomly generated trace and (b) deterministically generated trace. The dotted lines give the limits after which the generation pattern is repeated in the examples.

value I:

1. Set next switching time $T = 0$
2. Increase T by the chosen step size.
3. Generate events according to ON state while $t < T$.
4. If $t = T$ then switch to OFF state, set $t = 0$, and generate events while $t < T$.
5. If T exceeds window size L, go to 1.
6. Switch to ON state, set $t = 0$, and go to 2.

The above event generation optimises two conflicting goals: (1) all the intervals should be generated as often as possible to increase the individual probability P_τ and (2) all the intervals should be generated as soon as possible, because we have no a priori knowledge about the desired length of the trace.

In Figure 39(bottom,a) the example of a randomly generated trace is given. Below the probabilities to detect the upper and lower specification curve for A, B or C are depicted. In part (b) of the figure, we show a trace that was generated according to the deterministic algorithm. Again the probabilities for the interval sizes are given. Note that the largest interval (in the example A) has to be generated 3 times in order to equalise the probabilities P_τ.

The algorithm shown above generates a deterministic trace which is often not wanted for performance evaluation. To avoid this problem, the deterministic algorithm is randomised. Instead of stepwise increasing the dwell times in the generator states deterministically, we determine the dwell times randomly, according to a uniform distribution of the dwell times between the minimum interval size and the window size L. With this procedure, the event trace becomes unpredictable and especially multiple event traces are independent and uncorrelated.

3.7.3 Implementation

3.7.3.1 Trace Generation

In this section, we present an implementation of the algorithms described in Section 3.7.2. As discussed above, event traces can be generated starting from arrival curves using an ON/OFF state machine as traffic generator. The times at which we switch from ON to OFF state and vice versa can be determined at random or deterministically.

We can use arbitrary specification arrival curves for the event generator. Of course, the specification curves have to represent a valid characterisation, i.e. it must be possible to find an event stream that complies with both the upper and the lower specification curve. The only additional constraining factor for the curves is the memory demand. To keep the needed memory bounded, we use a periodically extended finite representation of the arrival curves in the following form:

$$\alpha(\Delta) = \left\lfloor \frac{\Delta}{W} \right\rfloor \alpha(W) + \alpha(\Delta - \left\lfloor \frac{\Delta}{W} \right\rfloor W)$$

In order to achieve a finite representation, we introduce a specification size W, up to which an arrival curve is defined. For arguments larger than W, arrival curves are periodically extended.

Algorithm 1 describes our general approach to generate event traces starting from upper and lower arrival curves $[\alpha^l(\Delta), \alpha^u(\Delta)]$. The algorithm contains calls to several functions which are further described in Table 8.

In the implementation, the algorithm to determine the next switching time between ON and OFF state is the only component that can be freely chosen by the user, whereas the generation pattern in the ON and the OFF state is fixed by the upper and lower specification arrival curves. We have implemented the deterministic switching time generator as proposed in

Algorithm 1 Algorithm for trace generation in pseudo-code.

```
/* initialise variables */
t = 0;
generate = false;
state = 0;
swt = getNextSwitchingTime(t);
/* generate event at time t */
generateEvent(t);
while (!stopGeneration) {
  while ( t < swt ) {
    if (state == 0) {
      if ( canIGenerateNow(t) )
        generate = true;
    }
    else{
      if ( !canIStillWait(t) )
        generate = true;
    }
    if (generate) {
      /* generate event at time t */
      generateEvent(t);
      updateHistoryWithEvent(t);
    }
    t = t + timeStep;
    generate = false;
  }
  swt = getNextSwitchingTime(t);
  state = (state + 1) mod 2;
}
```

Name	**Description**
`getNextSwitchingTime(`t`)`	Determines the next point in time based on the time stamp t at which the state of the generation mode should be changed.
`generateEvent(`t`)`	Generates event at time t
`canIGenerateNow(`t`)`	Returns a Boolean value denoting whether it is possible to generate event at time t without violating the specification curves.
`canIStillWait(`t`)`	Returns a Boolean value denoting whether it is possible to wait for `timeStep` time units after t with the event generation without violating the specification curves.
`updateHistoryWithEvent(`t`)`	Update the history of already generated events. Add the newest time stamp t and remove the oldest time stamp if the size of the history is > `WindowSize`.

Tab. 8: Functions used in Algorithm 1.

Section 3.7.2 and two randomised versions. One is based on a uniform distribution of the switching times between an upper and lower bound that can be set as a parameter. The generator with a uniform distribution represents a randomised version of the deterministic algorithm. The other one is based on a Weibull-distribution. We chose the Weibull-distribution here, because it is often used for traffic generation (e.g. in [ALM98] and [BC98]). We use it in our experiments to have control runs that we can compare with the other runs. The Weibull probability distribution function of a random variable X,

$$P\{X \leq x\} = 1 - e^{-(x/\beta)^{\alpha}}, x \geq 0$$

has two parameters $\alpha > 0$ and $\beta > 0$.

Note the difference between the event trace generation presented in Section 3.7.2 and the implementation. Although we stated that in the ON state, we will generate events as greedy as allowed by the upper arrival curve, this is not always possible without violating the specification $[\alpha^l(\Delta), \alpha^u(\Delta)]$.

Assume that the generator was in the ON state for some time t_0, the generated trace snippet represents the upper specification curve. In other words, the upper curve derived from the generated trace snippet matches the upper specification curve. Then, we switch for a very short time to the OFF state, and do not generate events at all. After coming back to the ON state, we are not necessarily allowed to again generate events as specified in the upper curve up to t_0. This is because we then would generate the maximum number of events allowed by the specification curves in t_0 twice within an interval of length $2t_0$. This could potentially violate the specification. See Figure 40 for an example. To solve this problem, we use in the algorithm the guard functions `CanIStillWait()` and `CanIGenerateNow()`. They ensure that a generated event does neither violate the upper nor the lower specification curve at any time.

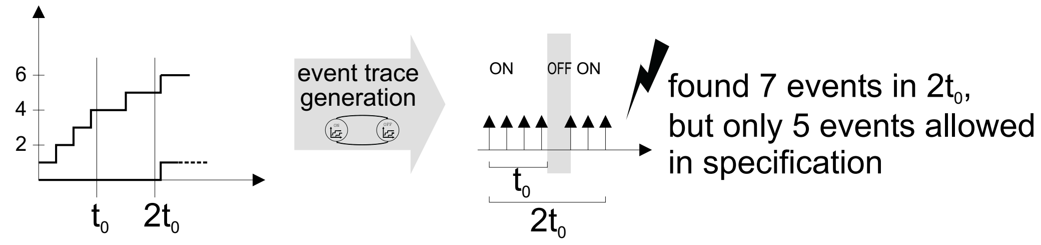

6
4
2
t0
2t0
event trace generation
ON
OFF
ON
t0
2t0
found 7 events in 2t0, but only 5 events allowed in specification

Fig. 40: Example in which greedy event generation leads to violation of the upper specification arrival curve.

3.7.4 Results

To compare different switching time estimation algorithms, we generated traces of 100'000 events each for different switching strategies. The results of the comparison are shown in Figure 41. On the y-axis the indicator values I are given. To be able to compare the different algorithms for two different specifications, we use the indicator values achieved for the deterministic algorithm as a base line and give the relative indicator values for the other algorithms. On the x-axis, we show the result bars for different switching time algorithms.

We have performed the tests for two different pairs of specification curves leading to two bars for all different switching time algorithms. The parameters corresponding to the different switching time estimation algorithms are specified in Table 9. For the Weibull distribution, α was fixed and the parameter β was then calculated such that the expectation value of the dwell time was as desired.

The deterministic switching time algorithm (left-most bars) achieves the highest indicator value, as expected. It leads to the event trace that represents well the desired worst-case behaviour. The randomised algorithms all perform worse than the deterministic algorithm. Nevertheless, the randomised algorithm using switching times uniformly distributed between 0 and $2L$, achieves also reasonably good results (within 65 %

of the deterministic algorithm on average), while being non-predictable. Weibull-distributed switching times lead to worse indicator values and should not be used for our event trace generator as a consequence. The indicator value I for a uniformly distributed switching time with expectation $\frac{L}{2}$ varies much between the two specifications (cf. Figure 41, column 10). The reason for this is that intervals of length L are hardly generated using this switching strategy and that the specifications consist of two different arrival curve pairs, where specification 1 is more regular and therefore easier to fulfill than specification 2.

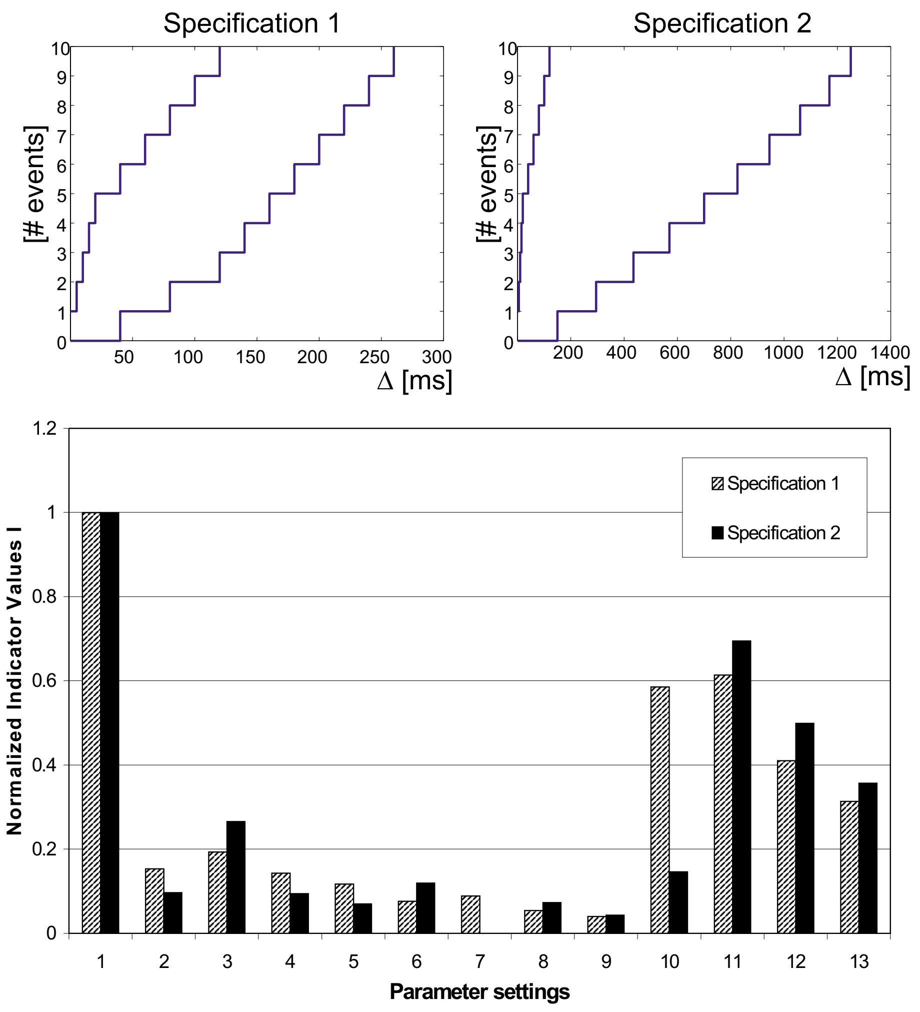


Fig. 41: Normalised indicator values I for different switching strategies between ON and OFF state specified in Table 9.

Nr.	Switching Time Determination
1	Deterministic algorithm as presented in Section 3.7.2 up to window size L
2	Weibull-distributed with expectation $\frac{L}{2}$ and $\alpha = 0.5$
3	Weibull-distributed with expectation L and $\alpha = 0.5$
4	Weibull-distributed with expectation $2L$ and $\alpha = 0.5$
5	Weibull-distributed with expectation $3L$ and $\alpha = 0.5$
6	Weibull-distributed with expectation $\frac{L}{2}$ and $\alpha = 0.3$
7	Weibull-distributed with expectation L and $\alpha = 0.3$
8	Weibull-distributed with expectation $2L$ and $\alpha = 0.3$
9	Weibull-distributed with expectation $3L$ and $\alpha = 0.3$
10	Uniformly distributed with expectation $\frac{L}{2}$
11	Uniformly distributed with expectation L
12	Uniformly distributed with expectation $2L$
13	Uniformly distributed with expectation $3L$

Tab. 9: Parameters for the different switching strategies for the event trace generator.

3.7.4.1 Application for Measurement System

The event generation has been applied in an industrial case study. In particular, a tool chain for validating a complex packet processing device with several quality of service classes was implemented. The framework consists of an arrival curve based event trace generator as presented in this section, a packet generator that generates IP (Internet protocol) packets at exactly the times specified in the event trace, and finally a data collector tool that collects the packets that have been processed by the system under test. In addition, a high precision synchronisation device allowed us to measure end-to-end delays of the system up to a precision of 1μs (see [Blu05]).

The graphical components of the tool chain have been implemented in Java, the packet generator and the data collector tool as Linux kernel modules in order to achieve good timing accuracy.

The block diagram of the tool chain is shown in Figure 42, a screen-shot of the data collector tool is given in Figure 43. In Figure 43, the top row shows the measured arrival curve at the receiver (left) and the measured per packet delay (right) for a real-time packet flow (RT), the bottom row for a flow with no real-time requirements (NRT). Note that the maximum delay measured for the RT flow is 95μs, whereas for the NRT flow the maximum delay is ca. 60 ms. With this framework, we have a tool chain at hand that can be used to validate system properties such as packet throughput or processing delays, starting from a formal specification of the system input with arrival curves.

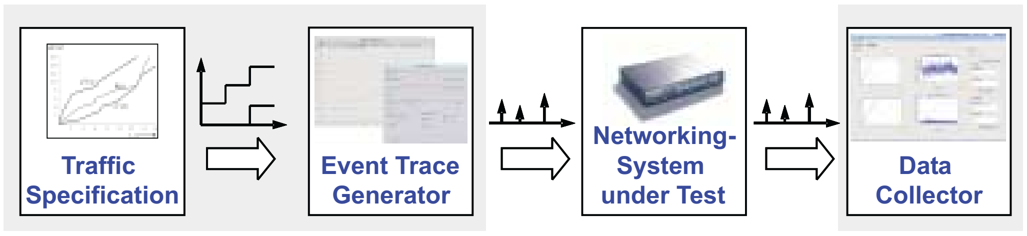


Fig. 42: Tool chain of the measurement tool.

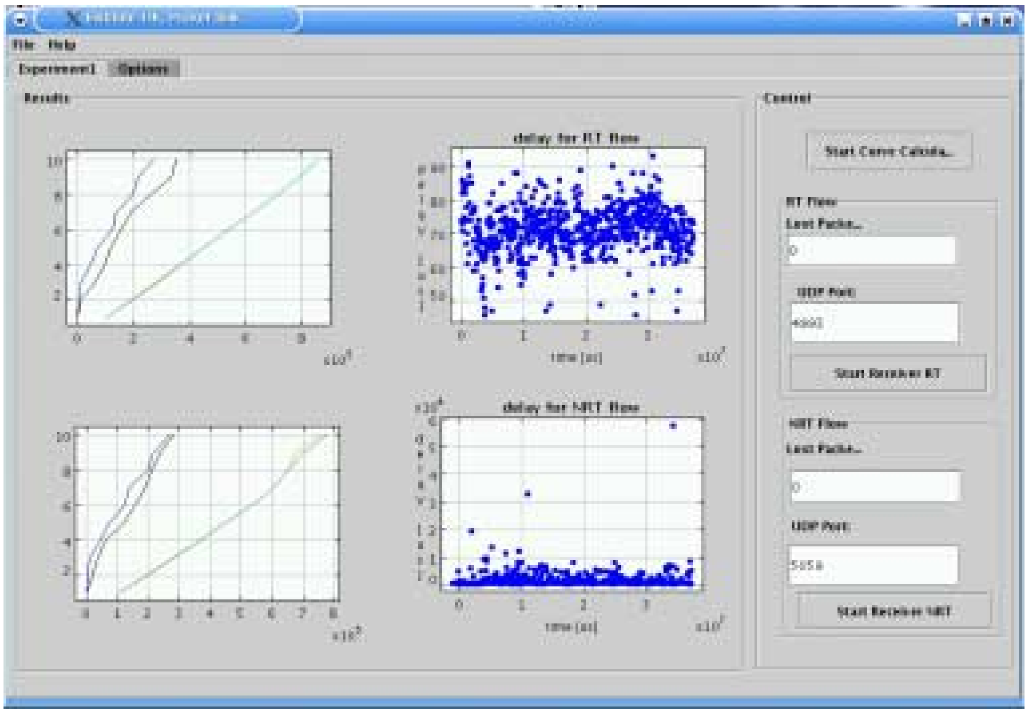


Fig. 43: Screen-shot of the measurement tool.

3.8 Summary

In this chapter, we revised the problem of design evaluation of a single design point in design space exploration. We discussed and classified existing approaches for system-level performance evaluation. We introduced a formal performance evaluation framework as well as simulation-based approaches written in SystemC. The analytical performance evaluation method introduced in Section 3.3 can be adapted such that it can be used for design space exploration. Further, we presented two comparative studies in which we compare the formal analysis approach used for design space exploration with existing simulation-based approaches. From these studies we can conclude that the results obtained with the formal analysis method are good enough and produced fast enough to enable the use of the method for design space exploration.

Finally, we presented a new hybrid approach for system-level performance evaluation of embedded systems that combines formal analysis methods with a simulation framework. We defined the interfaces needed for this combination and showed the applicability and the benefits of the approach using a case study. One of the core elements of the hybrid approach, the event trace generation based on arrival curves was presented and discussed in Section 3.7. We discussed the requirements for artificial traces as well as proposed a new quality indicator capturing these requirements.

In future, we would like to apply the hybrid approach to analyse larger systems. Furthermore, we believe that especially for the hybrid approach presented in this chapter there are more and more application scenarios, as for the design of embedded systems an increasing number of reusable components exist and the systems tend to become more and more complex.

4

Search Strategies

Search strategies are an important part in a design space exploration framework. Recent studies show that the quality of a design space exploration run heavily depends on the search strategy used (cf. [PSZ05],[KBTZ04]). In contrast to Section 2.4 where we discussed many different exploration methods used for design space exploration and classified them, we will concentrate on the use of multi-objective evolutionary algorithms (MOEAs) in the remainder of this chapter. The use of MOEAs proved to be very successful for design space exploration of embedded systems, as various application reports by other researchers show.

For examples, see chapters in a book edited by Drechsler and Drechsler [DD02] for system-level design methods and test generation with evolutionary algorithms. Palesi and Givargis explored the design space for a system-on-a-chip (SoC) in [PG02]. In [HE05], Hamann and Ernst employ evolutionary algorithms to identify optimal time slot sizes and turn lengths for TDMA-scheduled resources. These references are given just as examples for applications of EAs and are not complete.

In the next section, we introduce the concept of multi-objective evolutionary algorithms in detail. We give guidelines how to use MOEAs in the context of design space exploration. Then, performance indicators to assess the quality of the various search algorithms are presented in Section 4.2, before we introduce as a main contribution of this chapter a new class of multi-objective evolutionary algorithms that incorporate directly the user preferences. These new algorithms named IBEA are described in Section 4.3, where we discuss the working principle of the new algorithms. Last, we compare the results achieved by IBEA with other popular evolutionary algorithms. The new proposed algorithms show a good performance on many test problems, including the design space exploration problem proposed in Chapter 5.

4.1 Multi-Objective Evolutionary Algorithms

Design space exploration is often a multi-objective optimisation problem. In the cache example from Section 2.1, there is a trade-off between the performance (measured in CPI for an example application) and the area on silicon needed for the implementation. It is not clear how to relate performance and area, therefore already for this simple example, the problem really deals with multiple objectives.

Evolutionary algorithms are a good choice for design space exploration of embedded systems, because:

1. The design space is usually too large to allow us the use of exhaustive search methods.

2. The evaluation of a design point can be seen as a black box, where we do not have knowledge about the internal structure of the objective functions. This prohibits the use of specialised solvers as e.g. CPLEX [CPL], where we need knowledge about the internals of the search space.

3. The problem usually exposes a multi-objective optimisation problem for which evolutionary algorithms are well suited because they keep a population which can be used to approximate the front of optimal solutions in one run [Deb01].

Figure 44 shows the general principle of an evolutionary algorithm. Evolutionary algorithms (EA) are inspired by natural evolution. In other words, they are based on populations that evolve over several generations. In each generation, the EA selects from the population several interesting solutions, these are then so-called parent solutions. From the parents, new solutions are determined, using variation operators — mutation and recombination. The new solutions are evaluated to determine the fitness and added to the population. The last step in the loop is the selection for survival, in which the population is shrunk to the initial size by eliminating the worst (least-fit) solutions. In this section, we will discuss selection and variation in more detail. The evaluation of a solution was already covered in Chapter 3.

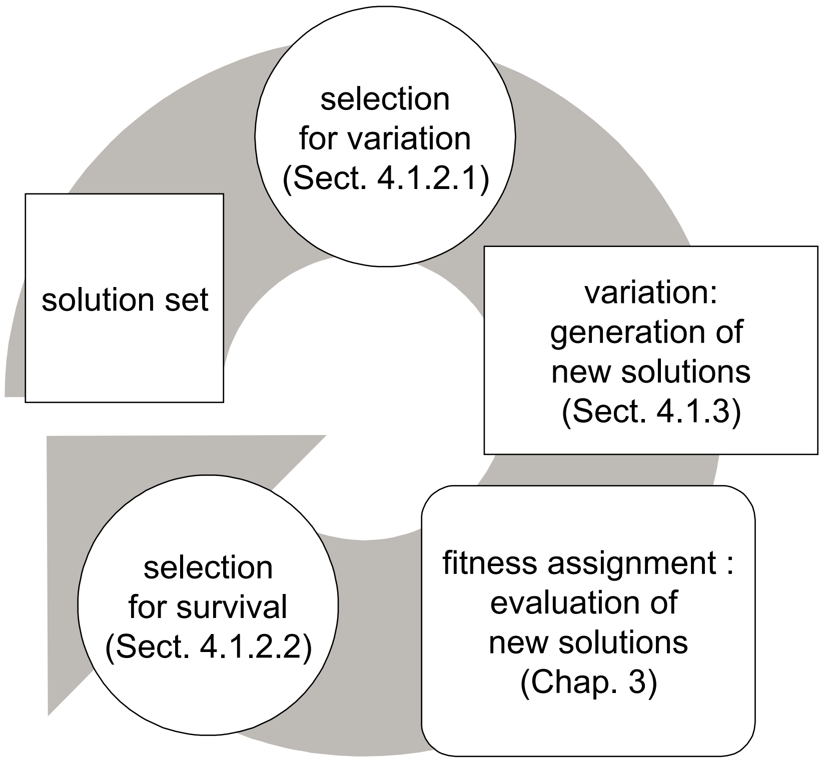


Fig. 44: General exploration cycle involving an evolutionary algorithm.

4.1.1 Pareto dominance

In order to introduce the components of evolutionary algorithms, we first have to clarify how two found solutions can be compared and somehow ranked. Let us consider a general multi-objective optimisation problem that is defined by a decision space X, an objective space Z, and n objective functions $f_1, f_2, \ldots, f_n$ that assign to each decision vector $\vec{x} \in X$ a corresponding objective vector $\vec{z} = (f_1(\vec{x}), f_2(\vec{x}), \ldots, f_n(\vec{x})) \in Z$. Without loss of generality, it is assumed that all objective functions are to be minimised and that $Z \subseteq \mathbb{R}^n$.

Assume that by some exploration method, we find the four solutions depicted in Figure 45 for our cache example from Section 2.1. To be able to categorise the solutions, in multi-objective optimisation the concept of Pareto-dominance is used. A found solution, represented by its decision vector $\vec{x}$ dominates an other solution if it is at least as good in all objective values and better in at least one. If a solution $\vec{x}^1$ dominates a solution $\vec{x}^2$ we also write $\vec{x}^1 \succ \vec{x}^2$. Formally, Pareto-dominance can be expressed like this:

$$\vec{x}^1 \succ \vec{x}^2, \quad \text{if} \quad \forall i, 1 \le i \le dim : f_i(\vec{x}^1) \le f_i(\vec{x}^2) \text{ and}$$

$$\exists i, 1 \leq i \leq dim : f_i(\vec{x}^1) < f_i(\vec{x}^2),$$

with *dim* the number of problem dimensions. Further, we write that a decision vector $\vec{x}^1$ weakly dominates another one $\vec{x}^2$, written as $\vec{x}^1 \succeq \vec{x}^2$, if $\vec{x}^1$ dominates $\vec{x}^2$ or the corresponding objective vectors are equal.

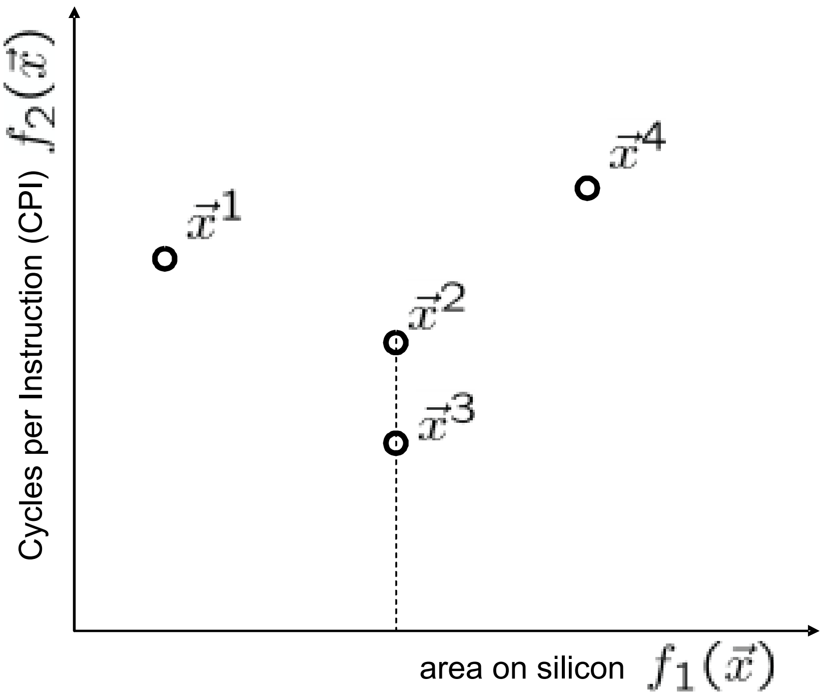


Fig. 45: Four solutions found for the cache example with two objectives involved.

Using the Pareto-dominance definition from above, we can now compare the four solutions found for the cache example shown in Figure 45. It is a minimisation problem, so the optimal point is the origin. Solution $\vec{x}^3$ dominates $\vec{x}^4$ as it is better in both criteria. $\vec{x}^3$ also dominates $\vec{x}^2$, it is at least as good in all criteria and better in one. Solution $\vec{x}^1$ is incomparable to $\vec{x}^2$ and $\vec{x}^3$, but clearly dominates $\vec{x}^4$.

The outcome of a MOEA is defined as a set of incomparable decision vectors, i.e., no decision vector dominates any other decision vector in the set. Such a set will also be denoted as Pareto set approximation, and the entirety of all Pareto set approximations is represented by the symbol Ω, where $\Omega \subseteq 2^Z$. The set of all Pareto-optimal solutions is called the Pareto set S with $S \in \Omega$.

4.1.2 Selection

Selection in evolutionary algorithms implements two distinct phases: selection for variation and selection for survival. The former type of selection chooses the most promising designs from the set of previously generated designs that will be varied in order to create new designs.

For practical reasons, not all of the generated designs will be kept in memory. While, e.g., simulated annealing and tabu search only store one solution in the working memory (in this case, selection for variation simply returns the single, stored solution), evolutionary algorithms operate on a population of solutions, which is usually of fixed size. As a consequence, another selection phase is necessary in order to decide which of the currently stored designs and the newly created ones will remain in the working memory. This phase is often called selection for survival or environmental selection, in analogy to the biological terminology used in the context of evolutionary algorithms.

4.1.2.1 Selection for Variation

Selection for variation is usually implemented in a randomised fashion. One possibility to choose N out of M designs is to hold tournaments between two solutions that are picked at random from the working memory based on a uniform probability distribution. For each tournament, the better design is copied to a temporary set which is also denoted as mating pool — again a term mainly used within the field of evolutionary computation. By repeating this procedure, several designs can be selected for variation, where high-quality designs are more likely to have one or multiple copies in the mating pool. This selection method is known as binary tournament selection; many alternative schemes exist as well (see [BFM97]).

Most of these selection algorithms assume that the usefulness or quality of a solution is represented by a scalar value, the so-called fitness value. While fitness assignment is straight forward in case of a single objective function, the situation is more complex in a multi-objective scenario. Here, one can distinguish between three conceptually different approaches:

- *Aggregation:* Traditionally, several optimisation criteria are aggregated into a single objective by, e.g., summing up the distinct objective function values, where weight coefficients are used to control the influence of each criterion. The difficulty with this approach, though, is the appropriate setting of the weights. This usually requires more knowledge about the design space than is actually available. Furthermore, optimising a particular weight combination yields one Pareto-optimal solution. To obtain several optimal trade-off designs, multiple weight combinations need to be explored either in parallel or subsequently. Nevertheless, not necessarily all Pareto-optimal designs can be found as illustrated in Figure 46. The weighted-sum approach is only able to detect all solutions if the front of Pareto-optimal solutions is convex. Similar problems occur with many other aggregation methods, see [Mie99].

- *Objective Switching:* The first papers using evolutionary algorithms to approximate the Pareto set suggested to switch between the different objectives during the selection step. For instance, Schaffer [Sch85] divided selection for variation into n selection steps where n corresponds to the number of optimisation criteria; in the ith step, designs in the working memory were chosen according to their ith objective function value.

- *Dominance-based Ranking:* Nowadays, most popular schemes use fitness assignments that directly make use of the dominance relation or extensions of it. By pairwise comparing all the designs in the working memory, different types of information can be extracted. The dominance rank gives the number of solutions by which a specific solution is dominated, the dominance count represents the number of designs that a particular design dominates, and the dominance depth denotes the level of dominance when the set of designs is divided into non-overlapping non-dominated fronts (see [Deb01] for details).

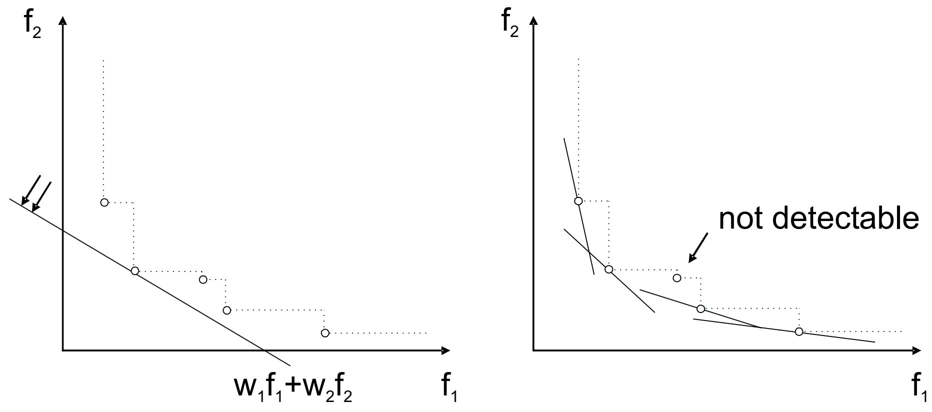


Fig. 46: Illustration of the weighted-sum approach for two objectives. The left hand side shows how a particular weight combination (w_1, w_2) uniquely identifies one Pareto-optimal design. The right hand side demonstrates that not for all Pareto-optimal designs such a weight combination exists.

These fitness assignment schemes can also be extended to handle design constraints. For dominance-based approaches, the dominance relation can be modified such that feasible solutions by definition dominate infeasible ones, while among infeasible designs the one with the lower constraint violation is superior. For feasible solutions, the definition of dominance remains unchanged (cf. [Deb01]). An alternative is the penalty approach which can be used with all of the above schemes. Here, the overall constraint violation is calculated and summarised by a real value. This value is then added to the original fitness value (assuming that fitness is to be minimised); thereby, infeasible solutions are penalised.

Finally, another issue that is especially important in the presence of multiple objectives is maintaining diversity among the designs stored. If the goal is to identify a set of Pareto optima, special techniques are necessary in order to prevent the search algorithm from converging to a single trade-off solution. Most modern multi-objective optimisers integrate some diversity preservation technique that estimates the density of solutions in the space defined by the objective functions. For instance, the density around a solution can be estimated by calculating the Euclidean distance to the next closest solution. This density information can then be incorporated into the fitness, e.g., by adding original fitness value and density estimate.

4.1.2.2 Selection for Survival

When approximating the Pareto set, it is desirable not to loose promising designs due to random effects. Therefore, selection for survival is usually realised by a deterministic algorithm. Similar issues as with selection for variation come into play here; however, almost all search methods make sure that designs not dominated among those in the working memory are preferred over dominated ones with respect to environmental selection. If there are too many non-dominated solutions, then additional diversity information is used to further discriminate between these designs. Furthermore, as many randomised search algorithms only keep a single solution in the working memory, often a secondary memory, a so-called archive, is maintained that stores the current approximation of the Pareto set. For instance, PAES [KC00], a randomised local search method for multi-objective optimisation, checks for every generated design whether it should be added to the archive, i.e., whether it is dominated by any other archive member. If the design was inserted, dominated designs are removed. If the archive size is exceeded after insertion, a design with the highest density estimate is deleted.

A theoretical issue that has been investigated recently by different researchers [LTDZ02, Kno02] addresses the loss in quality per iteration.

Optimally, the current set of designs represents the best Pareto set approximation among all solutions ever considered during the optimisation run—given the actual memory constraints. This goal is difficult to achieve in general, but Laumanns *et al.* [LTDZ02] proposed an archiving method by which the loss can be bound and kept arbitrarily small by adjusting the memory usage accordingly.

4.1.3 Variation

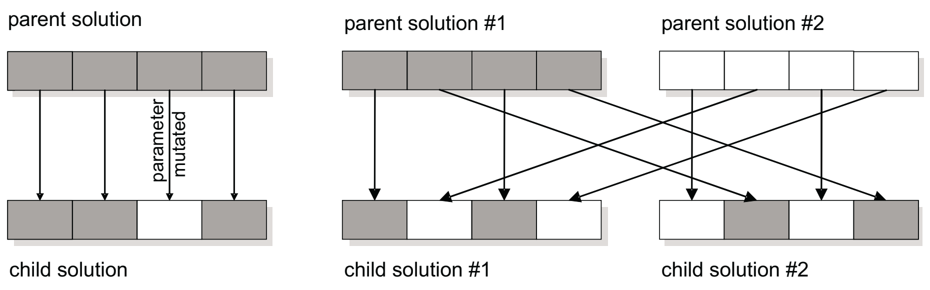


Fig. 47: Variation operators used in the cache example: mutation (left), recombination (right).

The purpose of the variation operators used with evolutionary algorithms is to determine new design points given a set of selected previously evaluated design points. There are several objectives for selecting appropriate variation operators:

- The variation operators operate on the design representation and generate a local neighbourhood of the selected design points. The construction of the variation operators is problem-dependent and a major possibility to include domain-knowledge.
- The constructed neighbourhood should not contain infeasible design points, if possible.
- In case of design points that are infeasible because non-functional properties are outside of given constraints, one may use a feedback loop in order to correct.
- The variation operator may also involve problem-dependent local search (e.g. by optimising certain parameters or hidden optimisation criteria) in order to relieve the randomised search from optimisation tasks that can better be handled with domain-knowledge.

In principle, different variation operators can be distinguished according to the number of solutions they operate on. Most randomised search algorithms generate a single new design point by applying a randomised operator to a known design point. For simulated annealing and randomised local search algorithms this operator is called neighbourhood function, whereas for evolutionary algorithms this operator is denoted as mutation operator. The term mutation will be used in the remainder of this section.

In the context of evolutionary algorithms there also exists a second type of variation, in addition to mutation. Since evolutionary algorithms maintain a population of solutions, it is possible to generate one or more new solutions based on two or more existing solutions. The existing designs selected for variation are often referred to as parents, whereas the newly generated designs are called children. The operator that generates ≥ 1 children based on ≥ 2 parents is denoted as recombination.

4.1.3.1 Mutation

The assumption behind mutation is that it is likely to find better solutions in the neighbourhood of good solutions. Therefore, mutation operators are usually designed in such a way that the probability of generating a specific solution decreases with increasing distance from the parent. There exist several approaches to implement mutation. It is, e.g., possible to always change exactly one parameter in the representation of a solution and keep all other parameters unchanged. A different mutation operator changes each of n parameters with probability $1/n$, which leads to the fact that one parameter is changed in expectation.

Changing a parameter means changing its value, i.e., flipping a bit in a binary representation, or choosing new parameter values according to some probability distribution for an integer- or real-valued representation. For representations based on permutations of vector elements the mutation operator changes the permutation by exchanging two elements. If the specification is based on lists of possible values, the mutation operator selects a new element according to some probability distribution.

In general, a mutation operator should on the one hand produce a new solution that is “close” to the parent solution with a high probability, but on the other hand be able to produce any solution in the design space, although with very small probability. This is to prevent the algorithm from being stuck in a local optimum.

The cache example from Section 2.1 uses the following mutation operator: Each of the design parameters is mutated with probability 0.25 (as there are 4 different parameters). The change that is applied to each of the parameters is normally distributed, i.e., the value of a parameter is increased by a value that is normally distributed around 0 inside the ranges given in Table 1; e.g. the block size parameter change is normally distributed between -4 and +4. Note, that in the example changes of size 0 are also allowed, i.e. the parameter remains unchanged.

4.1.3.2 Recombination

Recombination takes two or more solutions as input, and then generates new solutions that represent combinations of the parents. The idea behind recombination is to take advantage of the good properties of each of the parent to produce even better children. In analogy to the mutation operator, a good recombination vector should produce solutions that lie "between" the parents either with respect to the parameter space or to the objective space.

For vectors in general, recombination of two parents can be accomplished by cutting both solutions at randomly chosen positions and rearranging the resulting pieces. For instance, one-point crossover creates a child by copying the first half from the first parent and the second half from the second parent. If the cut is made at every position, i.e., at each position randomly either the value from the first or the second parent is copied, the operator is called uniform recombination.

A further approach for real-valued parameters is to use the average of the two parents' parameter values, or some value between the parents' parameter values. A detailed overview of various recombination operators for different representation data structures can be found in [BFM97].

In general, a good recombination operator should fulfil the following guideline: If we assume to have a distance metric $d(a,b)$ with a and b parent solutions, the recombination of a and b should lead to a solution c, such that $d(a,c) < d(a,b)$ and $d(b,c) < d(a,b)$.

For the cache example from Section 2.1 uniform recombination was used, i.e., for each of the parameters like cache block size it was randomly decided from which parent solution the parameter for the first child solution should be used, where all unused parameters of the parent solutions are then used for the second child solution. See Figure 47 on the right hand side for a graphical representation of uniform recombination.

4.1.3.3 Infeasible Solutions

It may happen that after mutation or recombination a generated solution is not feasible, i.e., the solution represented by the parameters doesn't describe a valid system. To solve this problem there are different possibilities. First, one could ensure that the variation operators do not create infeasible solutions by controlling the construction of new solutions, one can call this approach "valid by construction". Second, one could implement a repair method, that turns constructed solutions that are infeasible into feasible ones by fixing the infeasible parameters. Finally, one can use the concept of penalty functions in order to guide the search away from areas with infeasible design points.

4.2 Performance Indicators for Search Algorithms

In a multi-objective scenario, the goal of the optimisation process is to find a good approximation of the Pareto-set S. The difficulty, though, is that there is no general definition of what a good approximation of the Pareto set is. Each particular definition represents specific preference information that depends on the user.

We assume that the preferences of the decision maker can be given in terms of a binary quality indicator $I : \Omega \times \Omega \rightarrow \mathbb{R}$. A quality indicator in general is a function that maps k Pareto set approximations to a real number; most common are unary quality indicators where $k = 1$ (cf. [ZTL+03]). Binary quality indicators can be used to compare the quality of two Pareto set approximations relative to each other.

In this section we revise several existing indicators to assess the performance of optimisation algorithms. We do not cover all existing performance indicators here, other indicators can be found e.g. in Hansen and Jaszkiewicz's study [HJ98] or in [ZTL+03]. The indicators presented here are used later in this chapter and in the next chapter.

4.2.1 Additive ε^+-indicator

The additive ε-quality measure $I_{\varepsilon^+}(A, B)$ denotes the maximum value d, which has to be subtracted from all objective values for all points in the set of solutions A, such that the solutions in the shifted set A' equal or dominate any solution in set B in terms of the objective values. If the value is negative, the solution set A entirely dominates the solution set B. In other words, if the value $d = I_{\varepsilon^+}$ is negative, we can even add $|d|$ to all objective values for all points in set A and still the shifted set A' dominates the solution set B. Formally, this measure can be stated as follows:

$$I_{\varepsilon^+}(A, B) = \max_{\vec{y} \in B} \left\{ \min_{\vec{x} \in A} \left\{ \max_{1 \le i \le dim} \{ f_i(\vec{x}) - f_i(\vec{y}) \} \right\} \right\}, \qquad (4.1)$$

where *dim* denotes the problem dimension. Figure 48 shows a graphical interpretation of the additive ε-quality measure. The additive ε-indicator was first proposed in [ZTL+03].

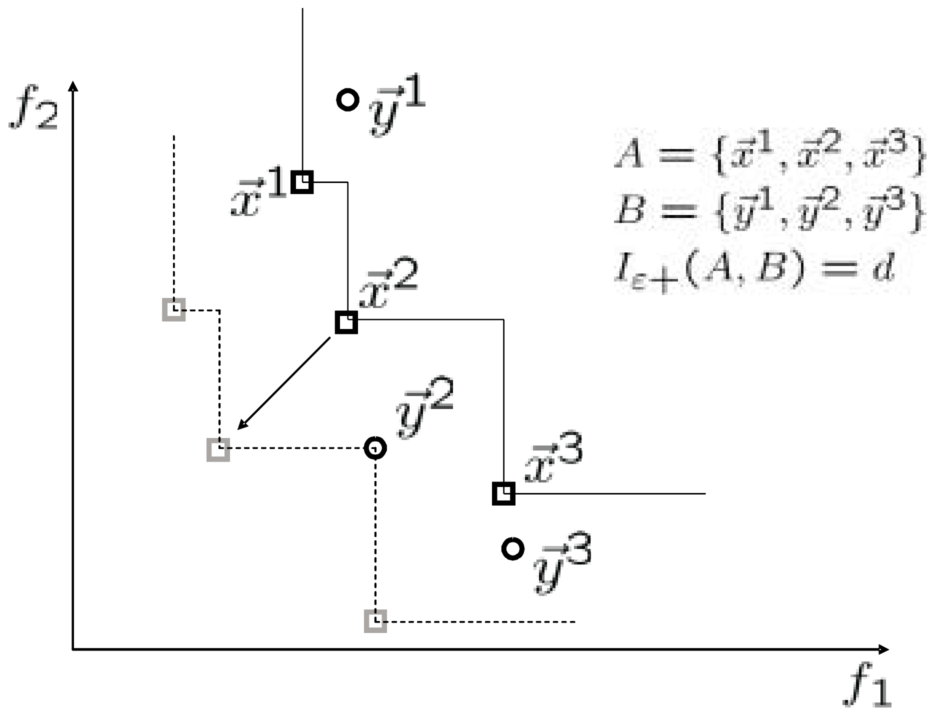

f_2
f_1
$\vec{y}^1$
$\vec{x}^1$
$\vec{x}^2$
$\vec{y}^2$
$\vec{x}^3$
$\vec{y}^3$
$A = \{\vec{x}^1, \vec{x}^2, \vec{x}^3\}$
$B = \{\vec{y}^1, \vec{y}^2, \vec{y}^3\}$
$I_{\varepsilon+}(A, B) = d$

Fig. 48: Illustration of the additive ε-quality measure I_{ε^+}, here $I_{\varepsilon^+}(A, B) = d$.

4.2.2 Multiplicative ε-indicator

The multiplicative ε-quality measure $I_\varepsilon(A, B)$ returns the maximum value f, by which all objective values for all points in the set of solutions A can be divided, such that the solutions in the shifted set A' equal or dominate any solution in set B in terms of the objective values. If the value is < 1, the solution set A entirely dominates the solution set B. Formally, this measure can be stated as follows:

$$I_\varepsilon(A, B) \;=\; \max_{\vec{y}\in B}\left\{\min_{\vec{x}\in A}\left\{\max_{1\le i\le dim}\left\{\frac{f_i(\vec{x})}{f_i(\vec{y})}\right\}\right\}\right\}, \tag{4.2}$$

where *dim* denotes the problem dimension. The multiplicative ε-indicator was proposed in [ZTL+03]. It allows us to claim that solution set A is at least $\frac{1}{I_\varepsilon(A,B)}$ times better than solution set B in all objectives, which is an interesting property especially when comparing two sets consisting of only one solution each.

4.2.3 Coverage Indicator

The coverage measure $I_C(A, B)$ returns the percentage of solutions in B which are dominated by or equal to at least one solution in A. It was introduced as a binary quality indicator in [ZT99]. More formally, we can define the coverage by the following formula:

$$I_C(A,B) = \frac{|\vec{y} \in B; \exists \vec{x} \in A : \vec{x} \geq \vec{y}|}{|B|} \tag{4.3}$$

Figure 49 shows a graphical example for the coverage indicator.

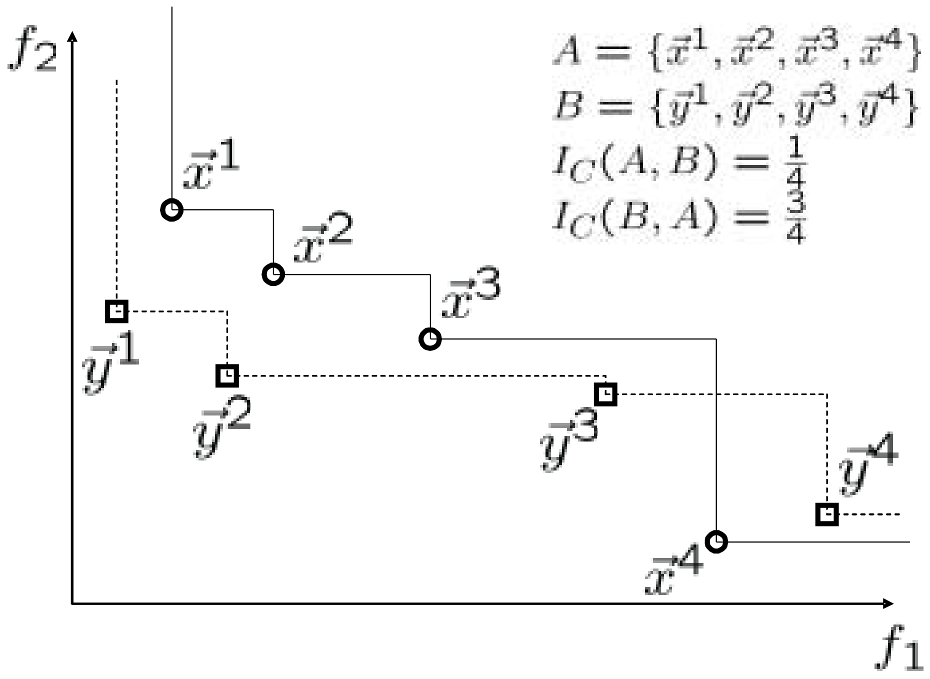


Fig. 49: Illustration of the coverage quality measure.

4.2.4 Hypervolume Indicator

The hypervolume indicator was introduced in [ZT99]. In contrast to the other quality measures presented in this section, it is an unary indicator. I.e. it is based on a single set of solutions only. For this set, we compute the hypervolume that is spanned by all the solutions $\vec{x}$ in the set A. The hypervolume can be seen as the union of hypercubes, each spanned between an individual solution $\vec{x}$ and a reference point $\vec{Z}$. We denote this indicator with $I_H(A)$:

$$I_H(A) = \bigcup_{\forall \vec{x} \in A} \Big\{ \prod_{1 \leq i \leq dim} (Z_i - f_i(\vec{x})) \Big\} \tag{4.4}$$

Figure 50 shows the hypervolume covered by 4 solutions in a 2 criteria optimisation problem. The criteria both have to be minimised. The choice of the reference point $\vec{Z}$ influences the values for $I_H(A)$. This may lead to problems if the reference point is chosen such that points lying on the edge of the objective space hardly contribute to the hypervolume.

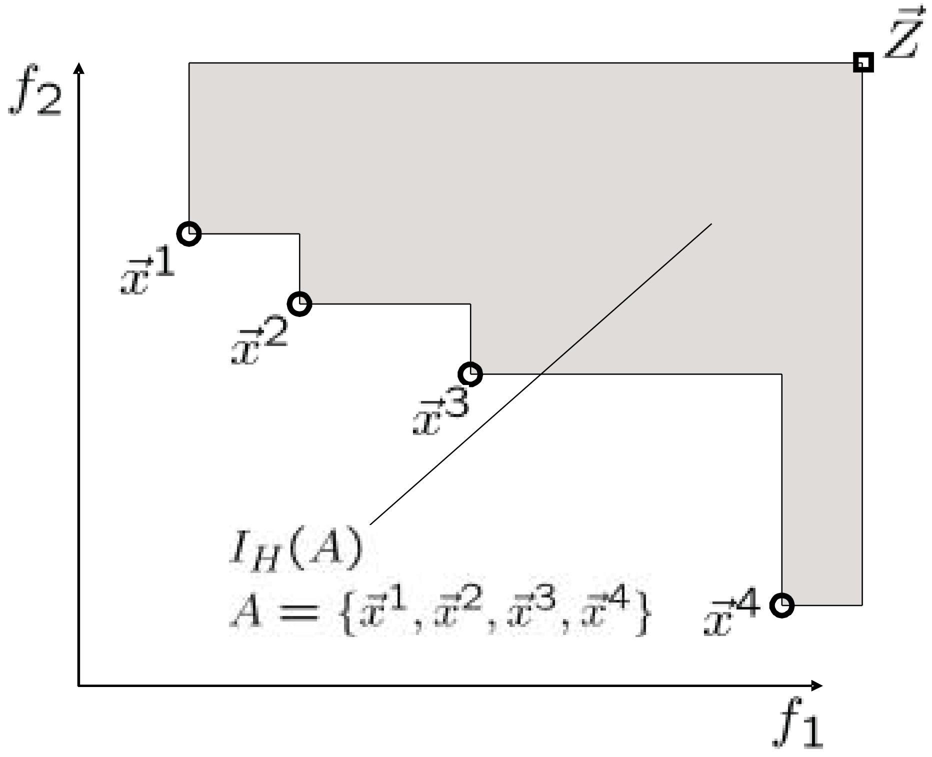


Fig. 50: Illustration of the hypervolume quality measure.

4.2.5 Binary Hypervolume Indicator

In contrast to the unary hypervolume indicator presented in the previous section, this indicator can be used to compare two solution sets. We define this indicator with $I_{HD}(A,B)$ as follows:

$$I_{HD}(A,B) = \begin{cases} I_H(B) - I_H(A) & \text{if } \forall \vec{x}^2 \in B \, \exists \vec{x}^1 \in A \,:\, \vec{x}^1 \succ \vec{x}^2 \\ I_H(A+B) - I_H(A) & \text{else} \end{cases} \tag{4.5}$$

Here, $I_H(A)$ gives the hypervolume of the objective space dominated by A (as defined in Section 4.2.4), and accordingly $I_{HD}(A,B)$ measures the volume of the space that is dominated by B but not by A with respect to a predefined reference point $\vec{Z}$.

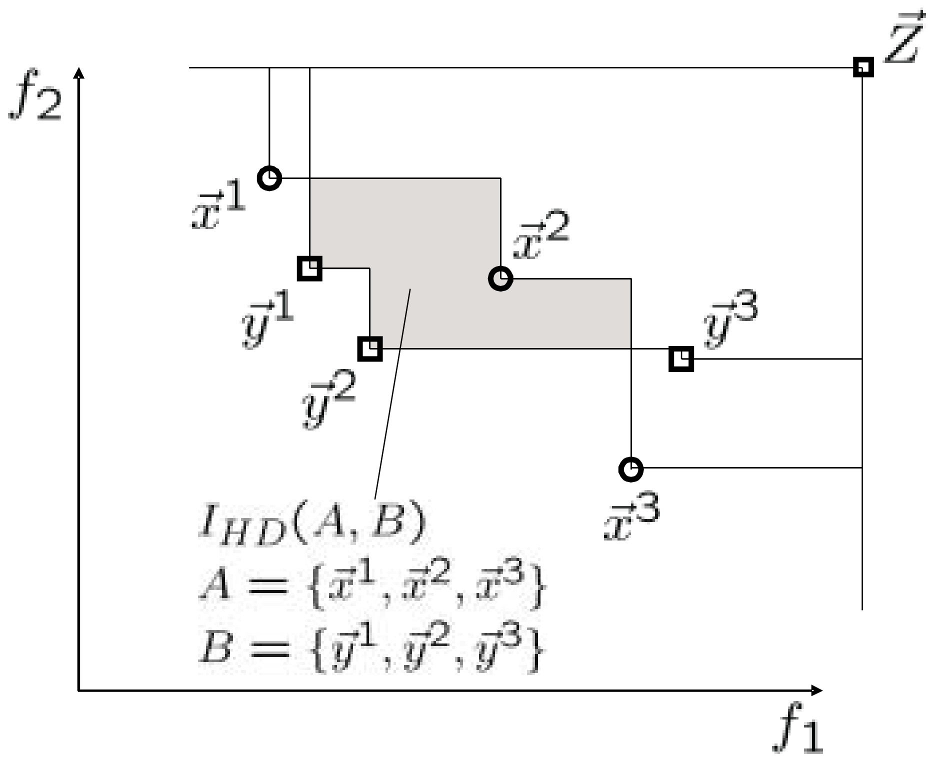


Fig. 51: Illustration of the binary hypervolume quality measure.

4.3 Indicator-Based Evolutionary Algorithm

This section discusses how preference information of the decision maker can in general be integrated into multi-objective search. The main idea is to first define the optimisation goal in terms of a binary performance measure (indicators as the ones introduced in Section 4.2) and then to directly use this measure in the selection process. To this end, we propose a general indicator-based evolutionary algorithm (IBEA) that can be combined with arbitrary indicators. In contrast to existing algorithms, IBEA can be adapted to the preferences of the user and moreover does not require any additional diversity preservation mechanism such as fitness sharing to be used. It is shown on several continuous and discrete benchmark problems that IBEA can substantially improve on the results generated by two popular algorithms, namely NSGA-II and SPEA2, with respect to different performance measures.

4.3.1 Motivation

If we consider the criteria that guided the design of MOEAs in the last decade, we make two observations:

1. The basis of most MOEAs is the assumption that there are two conflicting goals: (i) to minimise the distance to the Pareto-optimal set, and (ii) to maximise the diversity within the approximation of the Pareto-optimal set [Deb01]. However, recent studies [KC02, ZTL$^+$03] have shown that this assumption is problematic; to our best knowledge, there exists no formal definition of two separate objectives, one for convergence and one for diversity, that is compliant with the Pareto dominance relation. Furthermore, there are also practical problems related to this issue as discussed in [BT03].

2. In most popular MOEAs, the above assumption is implemented in terms of a Pareto-based ranking of the individuals that is refined by additional density information in objective space. The algorithms, though, differ in various aspects, and therefore each of them realises a slightly different optimisation goal, which is usually not explicitly defined. That means current approaches have not been designed for flexibility with respect to the preference information used; instead, they directly implement one particular type of preference information.

As to the first aspect, the alternative is to use Pareto-compliant formalisations of the decision maker's preferences (cf. [HJ98, KC02, ZTL$^+$03]). This, in turn, leads to a question that is directly related to the second aspect: How to design MOEAs with respect to arbitrary preference information?

The issue of integrating preference information into multi-objective search has been addressed by different researchers, see [CVL02] for an overview. For instance, Fonseca and Fleming [FF98] proposed an extended dominance relation that integrates predefined priorities and goals; however, the two observations stated above also apply to the algorithm introduced by them, similarly to many other algorithms used in this context: a diversity preservation mechanism is implemented that implicitly encodes unspecified preference information. In contrast, Knowles [Kno02] presented a multi-objective hill climber that can be combined with arbitrary unary performance measures and does not require niching methods. This approach, though, is—depending on the performance measure used—computationally expensive, and it is not clear how to extend it to population-based multi-objective optimisers that implement both mating and environmental selection.

In this section, we extend the idea of flexible integration of preference information by Fonseca and Fleming [FF98] and Knowles [Kno02] and propose a general indicator-based evolutionary algorithm, IBEA for short. The main idea is to formalise preferences in terms of *continuous* generalisations of the dominance relation, which leads to a simple algorithmic concept. Thereby, IBEA not only allows adaptation to arbitrary preference information and optimisation scenarios, but also does not need any diversity preservation techniques, in contrast to [FF98]. In comparison to [Kno02], IBEA is more general, since the population size can be arbitrary, and faster, because it only compares pairs of individuals and not entire approximation sets. As will be shown, the proposed approach can significantly improve the quality of the generated Pareto set approximation with respect to the considered optimisation goal—in comparison to prominent Pareto-based MOEAs.

4.3.2 Preliminaries

We consider binary quality indicators here because they represent a natural extension of the Pareto dominance relation. They can directly be used for fitness calculation similarly to the common Pareto-based fitness assignment schemes. One requirement, though, is that the considered indicator I is compliant with Pareto dominance as defined as follows.

Def. 3: *A binary quality indicator I is denoted as* dominance preserving *if (i) $\vec{x}^1 \succ \vec{x}^2 \Rightarrow I(\{\vec{x}^1\},\{\vec{x}^2\}) < I(\{\vec{x}^2\},\{\vec{x}^1\})$ and (ii) $\vec{x}^1 \succ \vec{x}^2 \Rightarrow I(\{\vec{x}^3\},\{\vec{x}^1\}) \geq I(\{\vec{x}^3\},\{\vec{x}^2\})$ for all $\vec{x}^1, \vec{x}^2, \vec{x}^3 \in X$.*

We will see later how these properties ensure that the proposed fitness assignment scheme is also Pareto dominance compliant. Note that the I_{ε^+}-indicator (cf. Section 4.2.1) is dominance preserving; for instance, the indicator values become negative as soon as $\vec{x}^1$ dominates $\vec{x}^2$ (cf. [ZTL$^+$03]).

Now, given an arbitrary optimisation problem and a corresponding binary quality indicator I, we can define the goal of the optimisation process as minimising $I(A, S)$ for $A \in \Omega$ where S is the Pareto set. If I is dominance preserving, then $I(A, S)$ is minimal for $A = S$; in the case of the additive ε-indicator, $I_{\varepsilon^+}(S, S) = 0$.

4.3.3 Indicator-Based Selection

Taking the scenario described in Section 4.3.2, the question is how I can be integrated in an MOEA to minimise $I(A, S)$, where A is the generated Pareto set approximation. This section deals with this issue.

4.3.3.1 Fitness Assignment

The population P represents a sample of the decision space, and fitness assignment tries to rank the population members according to their usefulness regarding the optimisation goal. Among the different ways how to exploit the information given by P and I, one possibility is to simply sum up the indicator values for each population member with respect to the rest of population, i.e.: $F'(\vec{x}^1) = \sum_{\vec{x}^2 \in P \setminus \{\vec{x}^1\}} I(\{\vec{x}^2\}, \{\vec{x}^1\})$ This fitness value F', which is to be maximised, is a measure for the "loss in quality" if $\vec{x}^1$ is removed from the population. For I_{ϵ^+}, e.g., $F'(\vec{x}^1)$ divided by the population size N equals the average ϵ needed to cover $\vec{x}^1$ by other population members. However, we will use a slightly different scheme in the following that amplifies the influence of dominating population members over dominated ones:

$$F(\vec{x}^1) = \sum_{\vec{x}^2 \in P \setminus \{\vec{x}^1\}} -e^{-I(\{\vec{x}^2\}, \{\vec{x}^1\})/\kappa}$$

We use one property of dominance preserving indicators here, namely that $I(\{\vec{x}^1\}, \{\vec{x}^2\}) < I(\{\vec{x}^2\}, \{\vec{x}^1\})$ if $\vec{x}^1 \succ \vec{x}^2$. Thereby, the influence of small indicator values contributes much more to the overall fitness than large values. The parameter κ is a scaling factor depending on I and the underlying problem; κ needs to be greater than 0. The proposed fitness assignment scheme is illustrated in Figure 52 for the I_{ϵ^+}-indicator. Furthermore, the following theorem shows that this fitness scheme is compliant with the Pareto dominance relation.

Thm. 2: *Let I be a binary quality indicator. If I is dominance preserving, then it holds that* $\vec{x}^1 \succ \vec{x}^2 \Rightarrow F(\vec{x}^1) > F(\vec{x}^2)$.

Proof: The fitness values of $\vec{x}^1$ and $\vec{x}^2$ are defined as

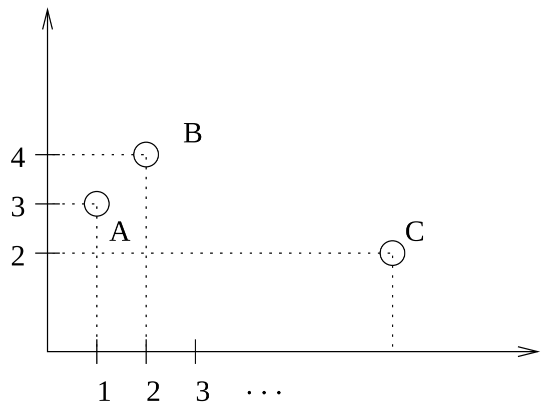
$$F(\vec{x}^1) = -e^{-I(\{\vec{x}^2\}, \{\vec{x}^1\})/\kappa} + \sum_{\vec{x}^3 \in P \setminus \{\vec{x}^1, \vec{x}^2\}} -e^{-I(\{\vec{x}^3\}, \{\vec{x}^1\})/\kappa} \tag{4.6}$$

$$F(\vec{x}^2) = -e^{-I(\{\vec{x}^1\}, \{\vec{x}^2\})/\kappa} + \sum_{\vec{x}^3 \in P \setminus \{\vec{x}^1, \vec{x}^2\}} -e^{-I(\{\vec{x}^3\}, \{\vec{x}^2\})/\kappa} \tag{4.7}$$

From Definition 3 and property (*i*) it follows that the indicator value $I(\{\vec{x}^1\}, \{\vec{x}^2\}) < I(\{\vec{x}^2\}, \{\vec{x}^1\})$. Since $-e^{-x/\kappa} > -e^{-y/\kappa}$, if $x < y$ and $\kappa > 0$, the left addend of (4.6) is strictly greater than the left addend of (4.7). Due to property (*ii*) of Definition 3, we find that the right addend of (4.6) is always greater or equal than the right addend of (4.7), following the same reasoning. Hence, it follows that $F(\vec{x}^1) > F(\vec{x}^2)$. □

4.3.3.2 Example Indicators

We show here, how two of the performance indicators presented in Section 4.2 can be used for the fitness assignment presented in this section.



$$I_{\varepsilon+}(A,B) = -1$$
$$I_{\varepsilon+}(B,A) = 1$$
$$I_{\varepsilon+}(A,C) = 1$$
$$I_{\varepsilon+}(C,A) = 6$$
$$I_{\varepsilon+}(B,C) = 2$$
$$I_{\varepsilon+}(C,B) = 5$$

$F'(x) = \sum_{y \in P \setminus \{x\}} I(y,x)$	$F(x) = \sum_{y \in P \setminus \{x\}} -e^{-I(y,x)}$
$F(A) = 1 + 6 = 7$	$F(A) = -e^{-1} - e^{-6} = -0.3703$
$F(B) = -1 + 5 = 4$	$F(B) = -e^{1} - e^{-5} = -2.7249$
$F(C) = 1 + 2 = 3$	$F(C) = -e^{-1} - e^{-2} = -0.5031$
Solution C is weakest	Solution B is weakest

Fig. 52: Consider three solutions A, B and C. Using the indicator $I_{\varepsilon+}$ we compute the fitness using a sum (left) and the amplified sum (right). The latter assignment shows the desired behaviour.

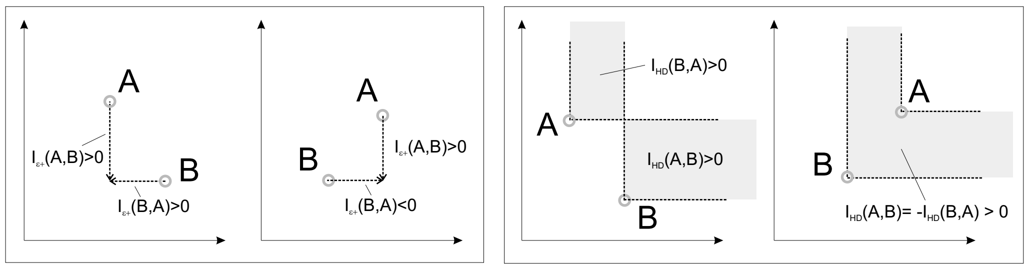


Fig. 53: Illustration of the two binary quality indicators used in this section where A and B contain one decision vector each (left: $I_{\varepsilon+}$-indicator; right: I_{HD}-indicator) .

In the example, we have seen how the additive ε-indicator can directly be incorporated into the algorithm to assign fitness values to the population members. The binary hypervolume indicator can be used in a similar way.

While the calculation of the $I_{HD}(A,B)$-values is computationally expensive for approximations containing several decision vectors, it is of order $\mathcal{O}(n)$ if two decision vectors are compared, with n denoting the number of objectives. The running-time complexity of computing $I_{\epsilon^+}(A,B)$ is also of order $\mathcal{O}(n)$, if the sets consist only of one solution each. Both indicators will be used later in this section. A graphical interpretation for I_{ϵ^+} and I_{HD} can be found in Figure 53.

We present only two binary indicators here. However, many other dominance preserving indicators can be defined that could be used instead. Other examples for binary quality indicators that could be used here are described in Hansen and Jaszkiewicz's study [HJ98].

4.3.3.3 Basic Algorithm

Based on the above fitness assignment scheme, we propose a general indicator-based evolutionary algorithm (IBEA) that performs binary tournaments for mating selection and implements environmental selection by iteratively removing the worst individual from the population and updating the fitness values of the remaining individuals. Its running-time complexity is $\mathcal{O}(n\alpha^2)$ with regard to the population size α and the number of objectives n. The fitness of each individual is based on the indicator values (of complexity $\mathcal{O}(n)$) computed for all other individuals in the population, therefore the complexity of computing the fitness for a single individual is $\mathcal{O}(n\alpha)$. Details of the algorithm are given below; note that it represents only the basic version of IBEA (denoted B-IBEA in the following), an extended version will be specified later.

Alg. 1: (Basic IBEA)

Input:	α	*(population size)*
	N	*(maximum number of generations)*
	κ	*(fitness scaling factor)*
Output:	A	*(Pareto set approximation)*

Step 1: *Initialisation: Generate an initial population P of size α; set the generation counter m to* 0.

Step 2: *Fitness assignment: Calculate fitness values of individuals in P, i.e., for all $\vec{x}^1 \in P$ set $F(\vec{x}^1) = \sum_{\vec{x}^2 \in P\setminus\{\vec{x}^1\}} -e^{-I(\{\vec{x}^2\},\{\vec{x}^1\})/\kappa}$.*

Step 3: *Environmental selection: Iterate the following three steps until the size of population P is smaller or equal to α:*

1. *Choose an individual $\vec{x}^* \in P$ with the smallest fitness value, i.e., $F(\vec{x}^*) \leq F(\vec{x})$ for all $\vec{x} \in P$.*
2. *Remove $\vec{x}^*$ from the population.*
3. *Update the fitness values of the remaining individuals, i.e., $F(\vec{x}) = F(\vec{x}) + e^{-I(\{\vec{x}^*\},\{\vec{x}\})/\kappa}$ for all $\vec{x} \in P$.*

Step 4: *Termination: If $m \geq N$ or another stopping criterion is satisfied then set A to the set of decision vectors represented by the non-dominated individuals in P. Stop.*

Step 5: *Mating_selection: Perform binary tournament selection with replacement on P in order to fill the temporary mating pool P'.*

Step 6: *Variation: Apply recombination and mutation operators to the mating pool P' and add the resulting offspring to P. Increment the generation counter ($m = m + 1$) and go to Step 2.*

4.3.3.4 Experimental Results

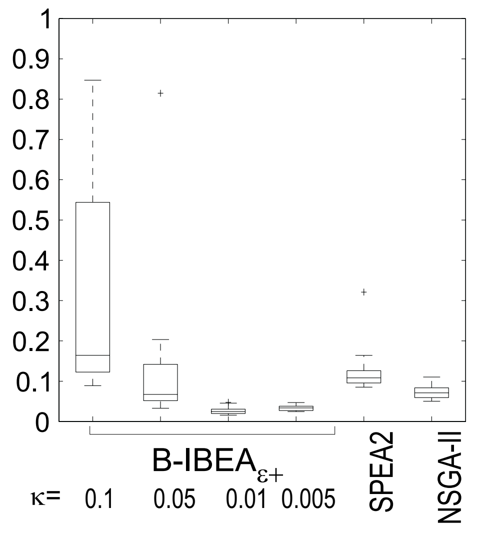
The proposed algorithm was tested on several well-known benchmark problems: the 2-dimensional knapsack problem instance from [ZT99] with 100 items, the design space exploration problem for network processors further described in Chapter 5 comprising problem instances with two (EXPO2), three (EXPO3), and four (EXPO4) objectives (cf. [TCGK02a]), and four continuous test functions, namely ZDT6 [ZDT00] and KUR [Kur91] with two objectives as well as DTLZ2 and DTLZ6 [DTLZ02] with three objectives each. For all problems, the population size α was set to 100 and the maximum number of generations N to 200. Overall, 30 runs with different initial populations were carried out per algorithm and per benchmark problem.

For the continuous problems, the individuals are coded as real vectors, where the SBX-20 operator is used for recombination and a polynomial distribution for mutation [DA95]. The recombination and mutation probabilities were set to 1.0 and to 0.01, resp., according to [DTLZ05]. For the knapsack problem, an individual is represented by a bit string, recombination is performed as one-point crossover with probability 0.8 according to [ZT99], and point mutations are performed with bit-flip probability 0.04, as suggested in [LZT01]. For the design space exploration problems EXPO, the representation of individuals and the operators are described in [TCGK02a]. The recombination probability was set to 0.5 and the probability for mutation was set to 0.8. (These are the same parameter settings as proposed in [TCGK02a].)

To assess the performance values, we have compared the solutions found by the two new algorithms B-IBEA$_{\varepsilon+}$ and B-IBEA$_{HD}$ with NSGA-II [DAPM00] and SPEA2 [ZLT02]. The performance comparison was carried out using the quality indicators $I_{\varepsilon+}$ and I_{HD}, i.e., we have computed 30 indicator values $I(A, R)$ for different seeds for all the tested algorithms. In this formula, A stands for the output that the evolutionary algorithm produced; the reference set R was determined by merging all solutions found by all the different algorithms into one set and keeping the non-dominated solutions. R was used instead of the Pareto set S, because S is usually unknown.

For the results obtained using B-IBEA$_{\varepsilon+}$, B-IBEA$_{HD}$, NSGA-II and SPEA2, we can observe in the comparison that B-IBEA$_{\varepsilon+}$ and B-IBEA$_{HD}$ perform significantly better than the other algorithms regarding both performance indicators and for appropriately chosen parameter κ. For the variation parameter settings described above, the choice for the parameter κ does not influence the performance of the algorithm much. However, other parameter settings indicate that the optimal choice of κ can vary and is dependent on the problem and the indicator used. In Figure 54 (left), the influence of different values κ for the performance of B-IBEA$_{\varepsilon+}$ on the problem ZDT6 is given. The performance of B-IBEA$_{HD}$ not only depends on the choice of κ but also on the choice of the reference point. In Figure 54 (right), we can see that for a particular choice of both κ and the reference point, the performance of B-IBEA$_{HD}$ for problem ZDT6 is better than SPEA2 and NSGA-II, but for other choices for κ and the reference point the performance is substantially worse (For the experiment in Figure 54 we set both the mutation and recombination probability to 1).

The question that arises inspecting the results obtained so far is how we can improve the algorithms such that (i) the same κ value can be used for different problems and indicators, and (ii) B-IBEA$_{HD}$ becomes less sensitive to the choice of the reference point for I_{HD}.



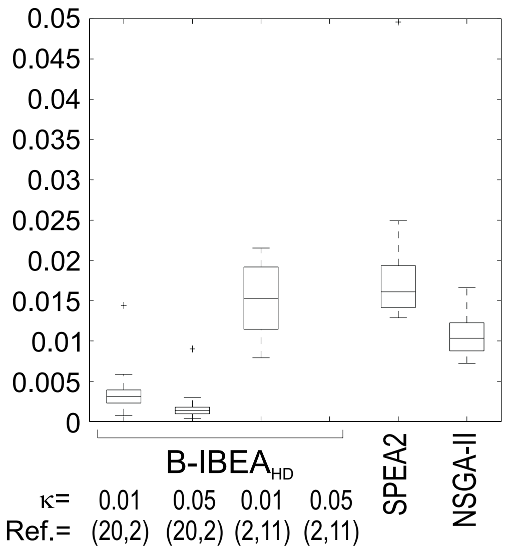


Fig. 54: **(left)** The indicator values $I_{\varepsilon+}$ for SPEA2, NSGA-II and B-IBEA$_{\varepsilon+}$ for different values of κ. For all the different algorithms one outlier was removed from the result sample for improved readability. **(right)** The indicator values I_{HD} for SPEA2, NSGA-II and B-IBEA$_{HD}$ for different values of κ and the reference point. In the 4th column, no values are given because they are about 10 times greater than the values given.

4.3.4 Improving Robustness

4.3.4.1 Adaptive IBEA

The values for the indicators $I(A, B)$ can be widely spread for different problems. This makes it difficult to determine an appropriate value for κ. We can ease this task by adaptively scaling the indicator values such that they lie in the interval $[-1, 1]$ for all points in the population. Thereby, we can use the same value κ for all the problems.

To tackle the problem of determining a good reference point for the I_{HD} indicator, we propose to use adaptive scaling not only for the indicator values, but also for the objective values. After scaling, the objective values lie in the interval $[0, 1]$. Like this, we can choose the worst values for each objective found in the population as reference point to calculate I_{HD}, i.e. the reference point would be set to 1 for all objectives. If we use this strategy, the only problem remaining is that the corner points found in a population do not add to the hypervolume. To overcome this problem, for the reference point we used a value of 2 for all objectives in the experiments with IBEA$_{HD}$.

Alg. 2: (Adaptive IBEA)

...

Step 2: *Fitness assignment: First scale objective and indicator values, and then use scaled values to assign fitness values:*

1. *Determine for each objective f_i its lower bound $\underline{b_i} = \min_{\vec{x} \in P} f_i(\vec{x})$ and its upper bound $\overline{b_i} = \max_{\vec{x} \in P} f_i(\vec{x})$.*
2. *Scale each objective to the interval $[0, 1]$, i.e., $f_i'(\vec{x}) = (f_i(\vec{x}) - \underline{b_i})/(\overline{b_i} - \underline{b_i})$.*
3. *Calculate indicator values $I(\vec{x}^1, \vec{x}^2)$ using the scaled objective values f_i' instead of the original f_i, and determine the maximum absolute indicator value $c = \max_{\vec{x}^1, \vec{x}^2 \in P} |I(\vec{x}^1, \vec{x}^2)|$.*
4. *For all $\vec{x}^1 \in P$ set $F(\vec{x}^1) = \sum_{\vec{x}^2 \in P \setminus \{\vec{x}^1\}} -e^{-I(\{\vec{x}^2\},\{\vec{x}^1\})/(c \cdot \kappa)}$.*

Step 3: *Environmental selection: ...*

1. ...
2. ...
3. *Update the fitness values of the remaining individuals, i.e., $F(\vec{x}) = F(\vec{x}) + e^{-I(\{\vec{x}^*\},\{\vec{x}\})/(c \cdot \kappa)}$ for all $\vec{x} \in P$.*

...

The algorithms IBEA$_{\varepsilon+}$ and IBEA$_{HD}$ denote the adaptive versions of the basic algorithms. For these versions, the choice of κ only marginally depends on the problem and the indicator under consideration. The changes in the initial algorithm are shown in Algorithm 2. For the experiments discussed in Section 4.3.4.2, we have used a fixed $\kappa = 0.05$ for all the problems and algorithms.

Preliminary tests have shown that this value for κ produced good results on the problems considered. Furthermore, the value for κ was chosen such that in the implementation no numerical problems occur, because smaller values led to fitness values larger than the maximum allowed double value in the PISA-specification ($= 10^{99}$).

4.3.4.2 Simulation Results

In Figure 55, the comparison results for the problems DTLZ6 and EXPO2 are shown. For both problems, the proposed algorithms IBEA$_{\varepsilon+}$ and IBEA$_{HD}$ perform significantly better than SPEA2 and NSGA-II with respect to the performance indicators $I_{\varepsilon+}$ and I_{HD}. Note that these IBEA versions all work with the same value for κ.

In addition to SPEA2, NSGA-II and the proposed IBEA$_{\varepsilon+}$ and IBEA$_{HD}$, we have implemented an adaptive version of SPEA2 to see the impact of adaptive objective-value scaling as such. The performance of the adaptive version of SPEA2 is similar to the performance of to the original algorithm on the test problems, and the Wilcoxon rank test returns false for all the problems investigated, i. e. the distributions of $I(A, R)$ for SPEA2 and the adaptive version of SPEA2 are not significantly different.

An overview of the results for $I_{\varepsilon+}$ is given in Table 10. Overall, we can see that for the continuous problems DTLZ2, DTLZ6 and ZDT6, the proposed algorithms IBEA$_{\varepsilon+}$ and IBEA$_{HD}$ perform significantly better than SPEA2 or NSGA-II; only for KUR, the latter provide better performance than the two IBEA instances. For the discrete knapsack problem, the significance tests return false, i.e. the indicator value distributions generated by the different search algorithms are statistically not different from each other. In contrast, the indicator-based algorithms show significantly better performance for the design-space exploration problem EXPO in two, three and four dimensions (EXPO2, EXPO3, EXPO4).

4.4 Summary

In this chapter we motivated the use of evolutionary algorithms for design space exploration. Using EAs, the design space exploration problem can be handled as multi-objective optimisation problem where we do not have to know internals of the problem but can treat it as black box. The selection and variation operators used with EAs were also discussed. In

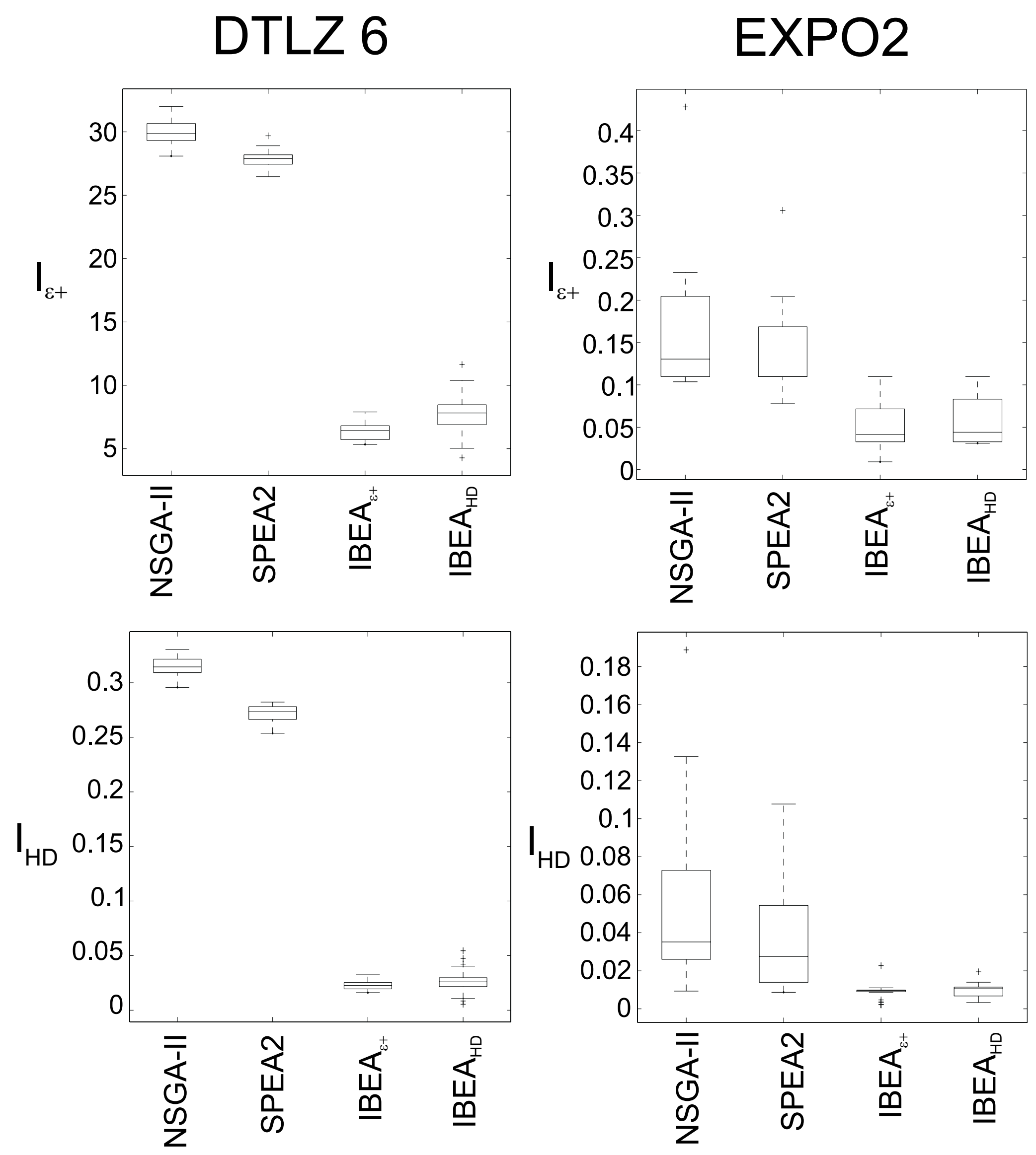


Fig. 55: Performance comparison for adaptive IBEA$_\varepsilon$, IBEA$_{HD}$, SPEA2 and NSGA-II solving problems DTLZ6 (left) and EXPO2 (right). On top, values for $I_{\varepsilon+}$, below for I_{HD} are given.

order to be able to compare the performance of different EAs for design space exploration, we introduced five indicators to assess the quality of an approximated Pareto-front found by a specific EA.

Further, we proposed a new evolutionary algorithm IBEA. Every MOEA implementation makes assumptions about the decision maker's preferences which are usually hard coded in the algorithm. These preferences may vary for each user and application. Therefore, we have argued that ideally MOEAs would be designed and evaluated with regard to the specific preferences of the user, formalised in terms of a performance measure. We have proposed a general indicator-based evolutionary algorithm (IBEA) that, contrarily to existing population-based MOEAs, allows to adapt the search according to arbitrary performance measures. For two different performance measures, this approach has be shown to generate significantly better results on six of eight benchmark problems in comparison to SPEA2 and NSGA-II, while no statistically significant performance difference could be observed on one of the test function.

All evolutionary algorithms used in this chapter (including the new indicator-based algorithm IBEA), programs implementing the performance indicators, and the statistical test programs can be obtained at the PISA website at `http://www.tik.ee.ethz.ch/pisa`.

	SPEA2		NSGA-II		SPEA2$_{adap}$		IBEA$_{\varepsilon,adap}$	
	P value	T	P value	T	P value	T	P value	T
ZDT6								
NSGA-II	$5.6073 \cdot 10^{-4}$	$\uparrow$						
SPEA2$_{adap}$	$> 5\%$	$\rightleftharpoons$	$8.1975 \cdot 10^{-6}$	$\downarrow$				
IBEA$_{\varepsilon,adap}$	$8.1014 \cdot 10^{-9}$	$\uparrow$	$2.0023 \cdot 10^{-5}$	$\uparrow$	$1.9568 \cdot 10^{-9}$	$\uparrow$		
IBEA$_{HD,adap}$	0.0095	$\uparrow$	$> 5\%$	$\rightleftharpoons$	$5.4620 \cdot 10^{-5}$	$\uparrow$	$1.3853 \cdot 10^{-5}$	$\downarrow$
DTLZ2								
NSGA-II	$3.0199 \cdot 10^{-10}$	$\downarrow$						
SPEA2$_{adap}$	$> 5\%$	$\rightleftharpoons$	$3.0199 \cdot 10^{-10}$	$\uparrow$				
IBEA$_{\varepsilon,adap}$	$3.0199 \cdot 10^{-10}$	$\uparrow$	$3.0199 \cdot 10^{-10}$	$\uparrow$	$3.0199 \cdot 10^{-10}$	$\uparrow$		
IBEA$_{HD,adap}$	$3.0199 \cdot 10^{-10}$	$\uparrow$	$3.0199 \cdot 10^{-10}$	$\uparrow$	$3.0199 \cdot 10^{-10}$	$\uparrow$	$5.5329 \cdot 10^{-7}$	$\downarrow$
DTLZ6								
NSGA-II	$8.1014 \cdot 10^{-9}$	$\downarrow$						
SPEA2$_{adap}$	$> 5\%$	$\rightleftharpoons$	$6.1210 \cdot 10^{-9}$	$\uparrow$				
IBEA$_{\varepsilon,adap}$	$3.0199 \cdot 10^{-10}$	$\uparrow$	$3.0199 \cdot 10^{-10}$	$\uparrow$	$3.0199 \cdot 10^{-10}$	$\uparrow$		
IBEA$_{HD,adap}$	$3.0199 \cdot 10^{-10}$	$\uparrow$	$3.0199 \cdot 10^{-10}$	$\uparrow$	$3.0199 \cdot 10^{-10}$	$\uparrow$	$3.5923 \cdot 10^{-4}$	$\downarrow$
KUR								
NSGA-II	$> 5\%$	$\rightleftharpoons$						
SPEA2$_{adap}$	$> 5\%$	$\rightleftharpoons$	$> 5\%$	$\rightleftharpoons$				
IBEA$_{\varepsilon,adap}$	$3.0199 \cdot 10^{-10}$	$\downarrow$	$3.0199 \cdot 10^{-10}$	$\downarrow$	$6.6955 \cdot 10^{-10}$	$\downarrow$		
IBEA$_{HD,adap}$	$3.0199 \cdot 10^{-10}$	$\downarrow$	$3.0199 \cdot 10^{-10}$	$\downarrow$	$4.9752 \cdot 10^{-10}$	$\downarrow$	$> 5\%$	$\rightleftharpoons$
Knapsack								
NSGA-II	$> 5\%$	$\rightleftharpoons$						
SPEA2$_{adap}$	$> 5\%$	$\rightleftharpoons$	$> 5\%$	$\rightleftharpoons$				
IBEA$_{\varepsilon,adap}$	$> 5\%$	$\rightleftharpoons$	$> 5\%$	$\rightleftharpoons$	$> 5\%$	$\rightleftharpoons$		
IBEA$_{HD,adap}$	$> 5\%$	$\rightleftharpoons$	$> 5\%$	$\rightleftharpoons$	$> 5\%$	$\rightleftharpoons$	$> 5\%$	$\rightleftharpoons$
EXPO2								
NSGA-II	$> 5\%$	$\rightleftharpoons$						
SPEA2$_{adap}$	$> 5\%$	$\rightleftharpoons$	0.0189	$\uparrow$				
IBEA$_{\varepsilon,adap}$	$1.0837 \cdot 10^{-8}$	$\uparrow$	$2.6753 \cdot 10^{-9}$	$\uparrow$	$6.4048 \cdot 10^{-8}$	$\uparrow$		
IBEA$_{HD,adap}$	$1.9638 \cdot 10^{-7}$	$\uparrow$	$1.2260 \cdot 10^{-8}$	$\uparrow$	$6.6261 \cdot 10^{-7}$	$\uparrow$	$> 5\%$	$\rightleftharpoons$
EXPO3								
NSGA-II	$> 5\%$	$\rightleftharpoons$						
SPEA2$_{adap}$	$> 5\%$	$\rightleftharpoons$	$> 5\%$	$\rightleftharpoons$				
IBEA$_{\varepsilon,adap}$	$4.3165 \cdot 10^{-8}$	$\uparrow$	$5.0801 \cdot 10^{-8}$	$\uparrow$	$3.1159 \cdot 10^{-7}$	$\uparrow$		
IBEA$_{HD,adap}$	$2.4189 \cdot 10^{-7}$	$\uparrow$	$1.5732 \cdot 10^{-7}$	$\uparrow$	$1.1653 \cdot 10^{-6}$	$\uparrow$	$> 5\%$	$\rightleftharpoons$
EXPO4								
NSGA-II	$> 5\%$	$\rightleftharpoons$	-					
SPEA2$_{adap}$	$> 5\%$	$\rightleftharpoons$	$9.4209 \cdot 10^{-4}$	$\downarrow$				
IBEA$_{\varepsilon,adap}$	$1.8546 \cdot 10^{-10}$	$\uparrow$	$6.9754 \cdot 10^{-10}$	$\uparrow$	$1.8390 \cdot 10^{-10}$	$\uparrow$		
IBEA$_{HD,adap}$	$1.9883 \cdot 10^{-10}$	$\uparrow$	$1.0221 \cdot 10^{-9}$	$\uparrow$	$1.9716 \cdot 10^{-10}$	$\uparrow$	$> 5\%$	$\rightleftharpoons$

Tab. 10: Comparison of different MOEAs for the $I_{\varepsilon+}$-indicator using the Wilcoxon rank test. The “P value” columns give the adjusted P value of the corresponding pairwise test that accounts for multiple testing; it equals the lowest significance level for which the null-hypothesis (the medians are drawn from the same distribution) would still be rejected. The “T” columns give the outcome of the test for a significance level of 5%: either the algorithm corresponding to the specific row is significantly better ($\uparrow$) resp. worse ($\downarrow$) than the algorithm associated with the corresponding column or there is no significant difference between the results ($\rightleftharpoons$).

5 Tools and Applications

In this chapter, we present applications of the work discussed in the previous chapters. As main new result, we introduce EXPO, a general tool for design space exploration. It is easily customisable such that it can be used to explore the solution space for virtually any user's problem specification. We describe all parts that have to be implemented for such a customisation. The advantages of using the framework is that the implementation effort for a user is minimised and that it can be coupled with various popular search algorithms that implement the PISA interface.

In a next section, we present the software architecture of the EXPO tool and introduce the PISA interface. Then, we present a case study, in which we used EXPO to explore the design space for complex packet processor architectures.

Finally, we present a comparative study in which we used the packet processor design space exploration problem as a benchmark application. With EXPO, we can assess the quality of different search algorithms. The results of the study are presented and discussed in Section 5.3.

5.1 EXPO: A General Framework for Design Space Exploration

The EXPO tool is a general framework for design space exploration. The tool performs the steps common to all design space exploration problems that are solved with evolutionary algorithms. It implements the variator part of the PISA interface (see Section 5.1.2). Further, the tool implements the population handling and offers a graphical interface to the user. The parts that are specific to the exploration problem are accessed by EXPO through a set of well-defined interfaces. Problem-specific parts include, e.g., the representation of a design point, or how a design point is evaluated. The structure of the tool is discussed in the next section. Section 5.1.2 covers the protocol used between the EXPO tool and an evolutionary search algorithm.

5.1.1 Software Architecture

The EXPO tool is completely written in Java and therefore platform-independent. Because it implements the text-based PISA interface, it needs access to a file system. There are no other restrictions on the execution platform.

Figure 56 gives the general structure of the EXPO tool framework. In the centre the main module is given. This module implements the following functionalities:

- **Control unit:** The tool consists of a control unit that enables a user to initiate, supervise and stop a design space exploration run.
- **Graphical user interface:** The user can start and stop exploration runs through a GUI. Further, the user can display details about the active population during the exploration as well as details about a single design point.
- **Population management:** The tool takes care about the population of active design points during an exploration run. It removes design points selected by the search algorithm as well as it adds new design points to the active population.
- **Solution generation and evaluation:** The tool invokes problem-specific software modules through interfaces: first for generation of new design points and second for their evaluation. These new design points are then added to the active population.
- **Communication with optimiser:** The main module handles all the communication needed with a PISA-compliant optimiser.

The purpose of the main module is to implement the functionality that is common to all design space exploration problems that are solved using evolutionary algorithms. A user of the framework can rely on already implemented common functions and therefore can concentrate on the software modules that are specific to his problem.

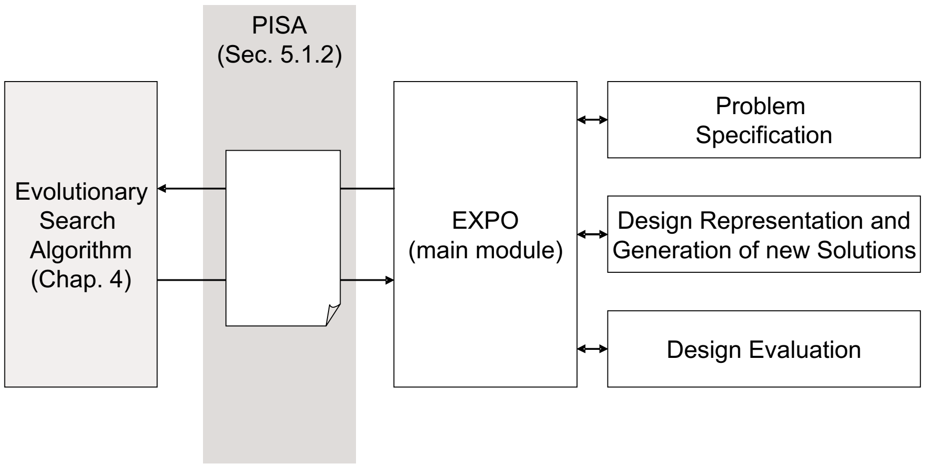

PISA
(Sec. 5.1.2)
Evolutionary Search Algorithm (Chap. 4)
EXPO (main module)
Problem Specification
Design Representation and Generation of new Solutions
Design Evaluation

Fig. 56: Block diagram for the EXPO-tool with the main module (centre), the problem-specific modules (right) and the search algorithm attached through the PISA interface (left).

On the right hand side of Figure 56, three problem-specific modules are shown. These modules are accessed from within the EXPO main module through interfaces. Note that for each new design space problem, these modules have to be implemented, whereas the main module remains the same. The following functionality is implemented by the modules:

- **Problem specification:** This module holds the specification for the design space exploration problem. The specification spans the solution space of all possible solutions and contains the constraints.
- **Design representation:** This module holds the parameters that represent a design point in the search space. Further, it implements the variation operators to generate new design points.
- **Design evaluation:** This module is invoked by the EXPO tool to evaluate a design point. It contains means to assign one or more values to a solution representing the desired properties of the design.

The PISA interface shown on the left hand side in Figure 56 is introduced in the next section. The description of the interfaces between EXPO and the problem-specific modules and their definition in Java is given in Appendix A.

5.1.2 PISA

PISA is a platform- and programming-language-independent interface for search algorithms. It is the purpose of PISA [BLTZ03] to make state-

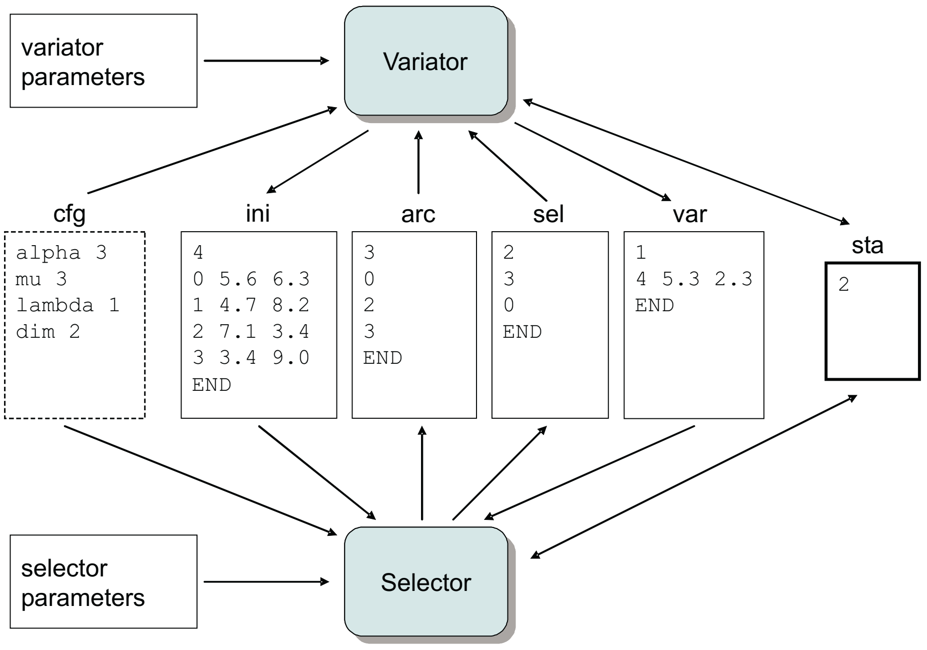


Fig. 57: Communication between modules through text files as defined by the PISA protocol. The files contain sample data.

of-the-art randomised search algorithms for multi-objective optimisation problems readily available. Therefore, for a new design space exploration task in embedded system design, one can concentrate on the problem-dependent aspects, where the domain-knowledge comes in. The protocol has to be implemented by any design space exploration tool, such as EXPO, that would like to benefit from pre-compiled and ready-to-use search algorithms available at `http://www.tik.ee.ethz.ch/pisa`. The detailed protocol including file formats and data type definitions is given in [BLTZ03]. In the protocol description, the application-specific part is called 'variator' and the search algorithm is denoted 'selector', according to Figure 57.

The protocol has been designed with several objectives in mind:

- Small amount of data that need to be communicated between the two different processes (selector and variator).
- The communicated data should be independent of the problem domain in order to enable a generic implementation of the selector process.
- Separation into problem-independent (selector) and problem-dependent (variator) processes.

- The implementation of the protocol should be as much as possible independent of the programming languages, hardware platforms and operating systems. It should enable a reliable (delay-independent) execution of the design space exploration.

The protocol defines the sequence of actions performed by the selector and variator processes. The communication between the two processes is done by exchange of text files over a common file system. The handshake protocol is based on states and ensures that during the optimisation process only one module is active at any time. During the inactive period a process polls the state file for changes. Whenever a module reads a state that requires some action on its part, the operations are performed and the next state is set.

The core of the optimisation process consists of state 2 and state 3: In each iteration, the selector chooses a set of parent individuals (design points) and passes them to the variator. The variator generates new child solutions on the basis of the parents, computes the objective function values of the new individuals, and passes them back to the selector.

In addition to the core states two more states are necessary for normal operation. State 0 and state 1 trigger the initialisation of the variator and the selector, respectively. In state 0 the variator reads the necessary parameters. Then, the variator creates an initial population, determines the objective values of the individuals and passes the initial population to the selector. In state 1, the selector also reads the required parameters, then selects a sample of parent individuals and passes them to the variator.

The four states 0–3 provide the basic functionality of the PISA-protocol. To add some flexibility the PISA-protocol defines a few more states which are mainly used to terminate or reset both the variator process and the selector process. Table 11 gives an overview over all defined states. The additional states 4–11 are not mandatory for a basic implementation of the protocol.

The data transfer between the two modules introduces some overhead compared to a traditional monolithic implementation. Thus, the amount of data exchange for each individual should be minimised. Since all representation-specific operators are located in the variator, the selector does not have to know the representation of the individuals. Therefore, it is sufficient to convey only the following data to the selector for each individual: an identifier and its objective vector. In return, the selector only needs to communicate the identifiers of the parent individuals to the variator. The proposed scheme allows to restrict the amount of data exchange between the two modules to a minimum.

For PISA-compliant search algorithms to work correctly, a designer has to ensure, that all objectives are to be *minimised*. In addition the variator

State	**Action**	**Next State**
State 0	Variator reads parameters and creates initial solutions	State 1
State 1	Selector reads parameters and selects parent solutions	State 2
State 2	**Variator generates and evaluates new solutions**	**State 3**
State 3	**Selector selects solutions for variation**	**State 2**
State 4	Variator terminates.	State 5
State 6	Selector terminates.	State 7
State 8	Variator resets. (Getting ready to start in state 0)	State 9
State 10	Selector resets. (Getting ready to start in state 0)	State 11

Tab. 11: States for the PISA-protocol. The main states of the protocol are printed in bold face.

and selector have to agree on a few common parameters: (i) the population size α, (ii) the number of parent solutions μ, (iii) the number of child solutions λ and (iv) the number of objectives `dim`. These parameters are specified in the parameter file with suffix `cfg`, an example file is shown in Figure 57.

The selector and the variator are normally implemented as two separate processes. These two processes can be located on different machines with possibly different operating systems. This complicates the implementation of a synchronisation method. Most common methods for interprocess communication are therefore not applicable.

In PISA, the synchronisation problem is solved using a common state variable which both modules can read and write. The two processes regularly read this state variable and perform the corresponding actions. If no action is required in a certain state, the respective process sleeps for a specified amount of time and then rereads the state variable. The state variable is an integer number stored to a text file with suffix `sta`. The protocol uses text files instead of, e.g., sockets, because file access is completely portable between different platforms and familiar to all programmers.

All other data transfers between the two processes besides the state are also performed using text files. The initial population is written by the variator to the file with suffix `ini`, the population is written by the selector to a file with suffix `arc`. In a text file with suffix `sel` the selector stores the parent solutions that are selected for variation. The newly generated solutions are passed from the variator to the selector through a file with suffix `var`. All text files for data transfer have to begin with the number of elements that follow and to end with the keyword `END`. Once the receiving process has completely read a text file, it has to overwrite the file with 0, to indicate that it successfully read the data.

5.2 Applications of Framework

The tool framework with the generic EXPO tool for design space exploration together with the PISA interface for easy coupling of search algorithms was used for two example design space explorations described in this thesis: the cache exploration example introduced in Section 2.1 and the example presented in this chapter. To perform both the design space exploration runs, we only had to implement the problem-specific modules for EXPO introduced in Section 5.1.1 and decide on a optimisation algorithm to use. In the next section, we introduce the exploration problem for packet processor architectures.

5.2.1 Packet Processors

Packet processors are high-performance, programmable devices with special architectural features that are optimised for network packet processing. In this area of application, also the term network processor is used. They are mostly embedded within network routers and switches and are designed to implement complex packet processing tasks at high line speeds such as routing and forwarding, firewalls, network address translators, means for implementing quality-of-service (QoS) guarantees to different packet flows, and also pricing mechanisms.

Other examples of packet processors would be media processors which have network interfaces. Such processors have audio, video and packet-processing capabilities and serve as a bridge between a network and a source/sink audio/video device. They are used to distribute (real-time) multimedia streams over a packet network like wired or wireless Ethernet. This involves receiving packets from a network, followed by processing in the protocol stack, forwarding to different audio/video devices and applying functions like decryption and decompression of multimedia streams. Similarly, at source end, this involves receiving multimedia streams from audio/video devices (e.g., video camera, microphone, stereo systems), probably encrypting, compressing and packetising them, and finally sending them over a network.

Following the above discussion, there are major constraints to satisfy and conflicting goals to optimise in the design of packet processors:

- **Delay Constraints:** In case of packets belonging to a multimedia stream, there is very often a constraint on the maximal time a packet is allowed to stay within the packet processor. This upper delay must be satisfied under all possible load conditions imposed by other packet streams that are processed simultaneously by the same device.

- **Throughput Maximisation:** The goal is to maximise the maximum possible throughput of the packet processing device in terms of the number of packets per second.

- **Cost Minimisation:** One is interested in a design that uses a small amount of resources, e.g., single processing units, memory and communication networks.

- **Conflicting Usage Scenarios:** Usually, a packet processor is used in several, different systems. For example, one processor will be implemented within a router, another one is built into a consumer device for multimedia processing. The requirements from these different applications in terms of throughput and delay are typically in conflict to each other.

All of the above constraints and conflicting goals will be taken into account in the design space exploration for packet processors presented in this section.

5.2.1.1 Design Space Exploration

Complex embedded systems like packet processors are often comprised of a heterogeneous combination of different hardware and software components such as CPU cores, dedicated hardware blocks, different kinds of memory modules and caches, various interconnections and I/O interfaces, run-time environment and drivers, see e.g. Figure 58. They are integrated on a single chip and they run specialised software to perform the application.

Typically, the analysis questions faced by a designer during a system-level design process are:

- **Allocation:** Determine the hardware components of the packet processor like microprocessors, dedicated hardware blocks for computationally intensive application tasks, memory and busses.

- **Binding:** For each task of the software application choose an allocated hardware unit which executes it.

- **Scheduling Policy:** For the set of tasks that are mapped onto a specific hardware resource choose a scheduling policy from the available run-time environment, e.g., a fixed priority.

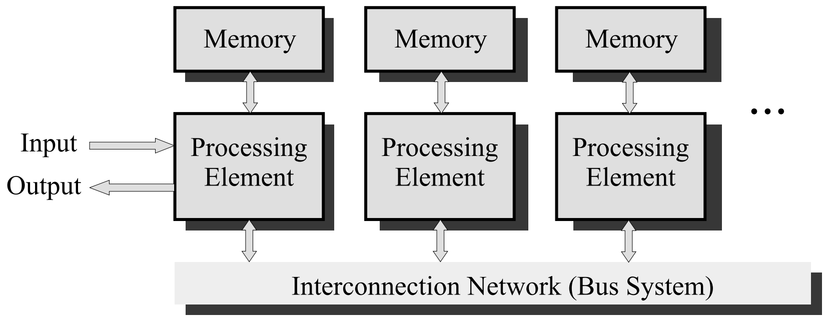


Fig. 58: Template of a packet processor architecture as used for the design space exploration.

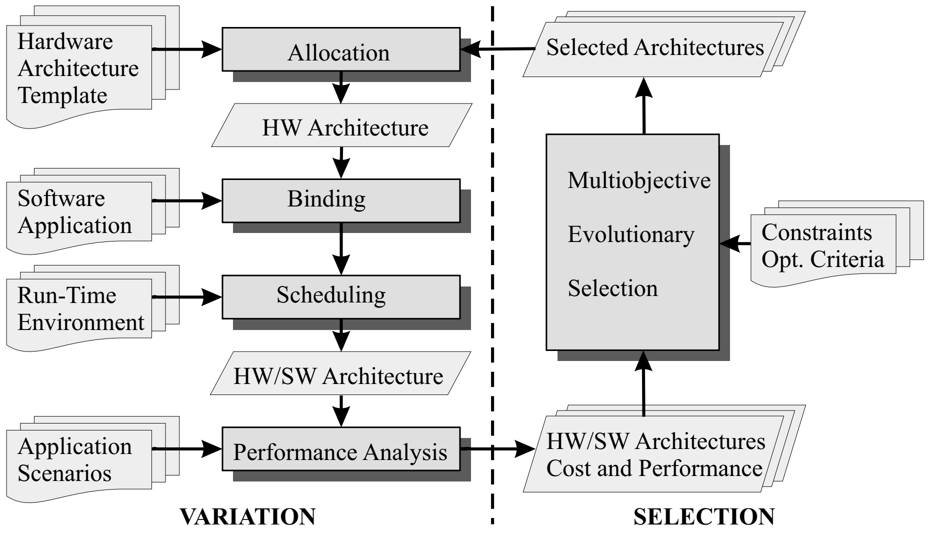


Fig. 59: Design space exploration methodology used for packet processors.

Most of the available design methodologies start with an abstract specification of the application and the performance requirements (cf. Chapter 1). These specifications are used to drive a system-level design space exploration [PLvdW$^+$01], which iterates between performance evaluation and exploration steps, see also Thiele *et al.* [TCGK02b, TCGK02a], and Blickle *et al.* [BTT98]. Finally, appropriate allocations, bindings, and scheduling strategies are identified. The methodology used for this exploration is shown in Figure 59. In the following, we describe how we implemented the 3 problem-specific modules needed for the EXPO tool (cf. Figure 56) to perform the design space exploration.

5.2.1.2 Problem Specification

According to Figure 59, basic prerequisites of the design space exploration are models for the architecture, the application, the run-time scheduling, and the application scenarios. Based on these models, the method presented in Section 3.3 can be used for performance analysis.

Architecture Template and Allocation. Following Figure 58, the model for a packet processor consists of a set of computation units or processing elements which perform operations on the individual packets. For the exploration problem presented here, we do not model the communication between the processing elements, i.e., packets can be moved from one memory element to the next one without constraints.

Def. 4: *We define a set of resources R. To each resource $r \in R$ we associate a relative implementation cost $cost(r) \geq 0$. The allocation of resources is described by the function $alloc(r) \in \{0, 1\}$. To each resource r there are associated two functions $\beta_r^u(\Delta) \geq 0$ and $\beta_r^l(\Delta) \geq 0$, denoted as upper and lower service curves, respectively.*

Initially, we specify all *available* processing units as our resource set R and associate the corresponding costs to them. For example we may have the resources R = {ARM9, MEngine, Classifier, DSP, Cipher, LookUp, CheckSum, PowerPC}. During the allocation step (see Figure 59), we select those which will be in a specific architecture, i.e., if $alloc(r) = 1$, then resource $r \in R$ will be implemented in the packet processor architecture.

The upper and lower service curves specify the available *computing units* of a resource r in a relative measure, e.g., processor cycles or instructions. In particular, $\beta_r^u(\Delta)$ and $\beta_r^l(\Delta)$ are the maximum and minimum number of available processor cycles in any time interval of length Δ. In other words, the service curves of a resource determine the best case and worst case computing capabilities.

Software Application and Binding. The purpose of a packet processor is to simultaneously process several streams of packets. For example, one stream may contain packets that store audio samples and another one contains packets from an FTP application. Whereas the different streams may be processed differently, each packet of a particular stream is processed identically, i.e., each packet is processed by the same sequence of tasks.

Def. 5: *We define a set of streams $s \in S$ and a set of tasks $t \in T$. To each stream s there is an ordered sequence of tasks $V(s) = [t_0, ..., t_n]$ associated. Each packet of the stream is first processed by task $t_0 \in T$, then successively by all other tasks until $t_n \in T$.*

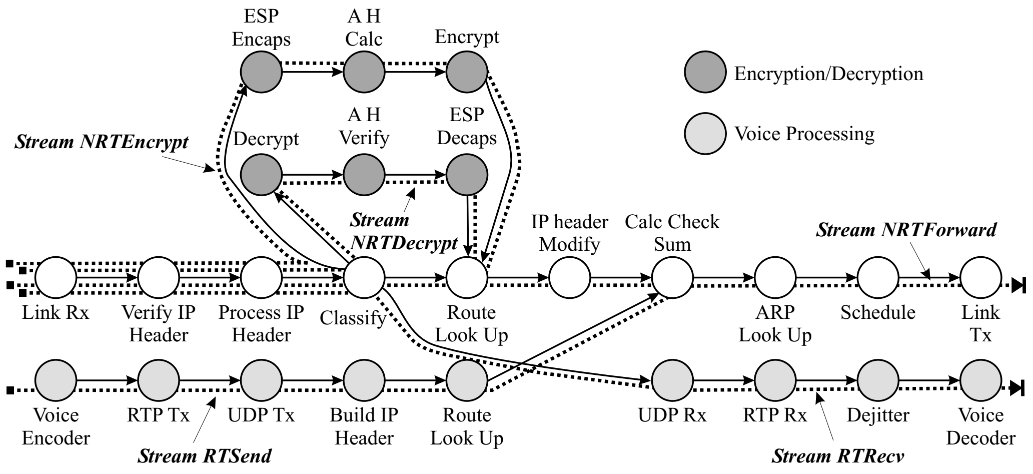


Fig. 60: Task graph of a packet processing application.

As an example we may have five streams $S = \{\text{RTSend, NRTDecrypt, NRTEncrypt, RTRecv, NRTForward}\}$. According to Figure 60, the packets of these streams when entering the packet processor undergo different sequences of tasks, i.e., the packets follow the paths shown. For example, for stream s = NRTForward we have the sequence of tasks $V(s)$ = [LinkRX, VerifyIPHeader, ProcessIPHeader, Classify, RouteLookUp, ... , Schedule, LinkTx].

Def. 6: *The mapping relation $M \subseteq T \times R$ defines all possible bindings of tasks, i.e., if $(t, r) \in M$, then task t could be executed on resource r. This execution of t for one packet would use $w(r, t) \geq 0$ computing units of r. The binding Z of tasks to resources $Z \subseteq M$ is a subset of the mapping such that every task $t \in T$ is bound to exactly one allocated resource $r \in R$, $alloc(r) = 1$. We also write $r = bind(t)$ in a functional notation.*

In a similar way as *alloc* describes the selection of architectural components, *bind* defines a selection of the possible mappings. Both *alloc* and *bind* will be encoded using an appropriate representation described later. The ‘load’ that a task t puts onto its resource $r = bind(t)$ is denoted as $w(r, t)$.

Figure 61 represents an example of a mapping between tasks and resources. For example, task ‘Classify’ could be bound to resource ‘ARM9’ or ‘DSP’. In a particular implementation of a packet processor we may have $bind(\text{Classify}) = \text{DSP}$, i.e., the task ‘Classify’ is executed on the resource ‘DSP’ and the corresponding execution requirement for each packet is $w(\text{DSP}, \text{Classify}) = 2.9$. Of course, this is possible only if the resource is allocated, i.e., $alloc(\text{DSP}) = 1$.

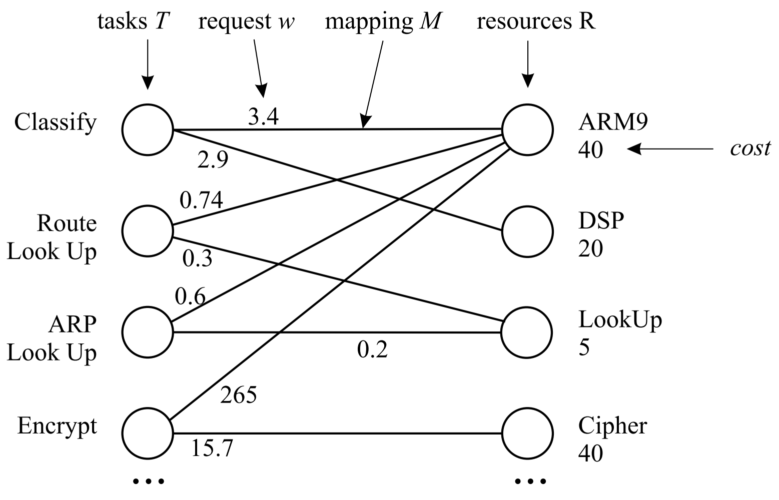


Fig. 61: Example of a mapping of task to resources.

Run-time Environment and Scheduling. According to Figure 58, there is a memory associated to each processing element. Within this memory, all packets are stored that need to be processed by the respective resource. The run-time environment now has different scheduling policies available that determine which of the waiting packets will be processed next.

Def. 7: *To each stream s there is associated an integer priority prio*$(s) > 0$. *There are no streams with equal priority.*

In the benchmark application, we suppose that only preemptive fixed-priority scheduling is available on each resource. To this end, we need to associate to each stream s a fixed priority $prio(s) > 0$, i.e., all packets of s receive this priority. From all packets that wait to be executed in a memory, the run-time environment chooses one for processing that has the highest priority among all waiting packets. If several packets from one stream are waiting, then it prefers those that are earlier in the task chain $V(s)$.

Application Scenarios. A packet processor will be used in several, possibly conflicting application scenarios. Such a scenario is described by the properties of the input streams, the allowable end-to-end delay (deadline) for each stream and the available total memory for all packets (sum of all individual memories of the processing elements).

Def. 8: *The properties of each stream s are described by upper and lower arrival curves* $\alpha_s^u(\Delta)$ *and* $\alpha_s^l(\Delta)$. *To each stream* $s \in S$ *there is associated the maximal total packet memory* $m(s) \geq 0$ *and an end-to-end deadline* $d(s) \geq 0$, *denoting the maximal time by which any packet of the stream has to be processed by all associated tasks* $V(s)$ *after its arrival.*

The upper and lower arrival curves specify upper and lower bounds on the number of packets that arrive at the packet processor. In particular, $\alpha_s^u(\Delta)$ and $\alpha_s^l(\Delta)$ are the maximum and minimum number of packets in *any time interval* of length Δ.

Def. 9: *The packet processor is evaluated for a set of scenarios $b \in B$. The quantities of Definition 8 are defined for each scenario independently.*

In addition, whereas the allocation *alloc* is defining a particular hardware architecture, the quantities that are specific for a software application are also specific for each scenario $b \in B$ and must be determined independently, for example the binding *bind* of tasks to processing elements and the stream priorities *prio*.

5.2.1.3 Design Evaluation

It is not obvious how to determine for any memory module, the maximum number of stored packets in it waiting to be processed at any point in time. Neither is it clear how to determine the maximum end-to-end delays experienced by the packets, since all packet flows share common resources. As the packets may flow from one resource to the next one, there may be intermediate bursts and packet jams, making the computations of the packet delays and the memory requirements non-trivial.

As introduced in Chapter 3, there exists a computationally efficient method to derive worst-case estimates on the end-to-end delays of packets and the required memory for each computation and communication. For this, we construct a scheduling network and apply Real-Time Calculus in order to derive the desired bounds.

As we know for each scenario the delay and memory in comparison to the allowed values $d(b,s)$ and $m(b,s)$, we can increase the input traffic until the constraints are just about satisfied. In particular, we do not use the arrival curves $\alpha^u_{(b,s)}$ and $\alpha^l_{(b,s)}$ directly in the scheduling network, but linearly scaled amounts $\psi_b \cdot \alpha^u_{(b,s)}$ and $\psi_b \cdot \alpha^l_{(b,s)}$, where the scaling factor ψ_b is different for each scenario. Now, binary search is applied to determine the maximal throughput such that the constraints on delay and memory are just about satisfied.

For the design evaluation, the following fact holds:

- Given the specification of a packet processing design problem by the set of resources $r \in R$, the cost function for each resource $cost(r)$, the service curves β_r^u and β_r^l, a set of streams $s \in S$, a set of application tasks $t \in T$, the ordered sequence of tasks for each stream $V(s)$, and the computing requirement $w(r,t)$ for task t on resource r;

- given a set of application scenarios $b \in B$ with associated arrival curves for each stream $\alpha^u_{(b,s)}$ and $\alpha^l_{(b,s)}$, and a maximum delay and memory for each stream $d(b,s)$ and $m(b,s)$;
- given a specific HW/SW architecture defined by the allocation of hardware resources $alloc(r)$, for each scenario b a specific priority of each stream $prio(b,s)$ and a specific binding $bind(b,t)$ of tasks t to resources;
- then we can determine — using the concepts of scheduling network, Real-Time Calculus and binary search — the maximal scaling factor ψ_b such that under the input arrival curves $\psi_b \cdot \alpha^u_{(b,s)}$ and $\psi_b \cdot \alpha^l_{(b,s)}$ the maximal delay of each packet and the maximal number of stored packets is not larger than $d(b,s)$ and $m(b,s)$, respectively.

As a result, we can define the criteria for the optimisation of packet processors.

Def. 10: *The quality measures for packet processors are the associated cost* $cost = \sum_{r \in R} alloc(r) cost(r)$ *and the throughput* ψ_b *for each scenario* $b \in B$*. These quantities can be computed from the specification of a HW/SW architecture, i.e.,* $alloc(r)$*,* $prio(b,s)$ *and* $bind(b,t)$ *for all streams* $s \in S$ *and tasks* $t \in T$*.*

5.2.1.4 Design Representation and Generation of new Solutions

Following Figure 59 and Definition 10, a specific HW/SW architecture is defined by $alloc(r)$, $prio(b,s)$ and $bind(b,t)$ for all resources $r \in R$, streams $s \in S$ and tasks $t \in T$. For the representation of architectures, we number the available resources from 1 to $|R|$; the tasks are numbered from 1 to $|T|$, and each stream is assigned a number between 1 and $|S|$. The allocation of resources can then be represented as integer vector $A \in \{0,1\}^{|R|}$, where $A[i] = 1$ denotes, that resource i is allocated. To represent the binding of tasks on resources, we use a two-dimensional vector $Z \in \{1,\ldots,|R|\}^{|B| \times |T|}$, where for all scenarios $b \in B$ it is stored which task is bound to which resource. $Z[i][j] = k$ means that in scenario i task j is bound to resource k. Priorities of flows are represented as a two-dimensional vector $P \in \{1,\ldots,|S|\}^{|B| \times |S|}$, where we store the streams according to their priorities, e.g., $P[i][j] = k$ means that in scenario i, stream k has priority j, with 1 being the highest priority. Obviously, not all possible encodings A, Z, P represent feasible architectures. Therefore, a repair method has been developed that converts infeasible solutions into feasible ones.

Recombination. The first step in recombining two individuals is creating exact copies of the parent individuals. With probability $1 - P_{cross}$, these individuals are returned as offspring and no recombination takes place.

Otherwise, crossing over is performed on either the allocation, the task binding or the priority assignment of flows.

With probability $P_{cross-alloc}$, a one-point crossover operation is applied to the allocation vectors A_1 and A_2 of the parents: First we randomly define the position j where to perform the crossover, then we create the allocation vector A_{new1} for the first offspring as follows:

$$\begin{aligned} A_{new1}[i] &= A_1[i], \text{ if } 1 \leq i < j \\ A_{new1}[i] &= A_2[i], \text{ if } j \leq i \leq |R| \end{aligned}$$

Similarly, A_{new2} is created. After this exchange in the allocation of resources, the repair method is called, to ensure, that for all tasks there is at least one resource allocated on which the task can be performed. If there exists a task in the specification for which there is no resource allocated on which it can be run, a feasible resource is randomly selected and allocated. If there exist mappings of tasks to resources which point to resources no longer allocated, the resources are re-allocated.

If the crossover is not done within the allocation vector, it is performed with probability $P_{cross-bind}$ within the binding of tasks to resources. In detail, a scenario $b \in B$ is randomly determined, for which the crossover of the binding vectors should happen. Then, a one point crossover for the binding vectors $Z_1[b]$ and $Z_2[b]$ of the parents according to the following procedure is performed, where j is a random value in the interval $[1, |T|]$.

$$\begin{aligned} Z_{new1}[b][i] &= Z_1[b][i], \text{ if } 1 \leq i < j \\ Z_{new1}[b][i] &= Z_2[b][i], \text{ if } j \leq i \leq |T| \end{aligned}$$

The binding Z_{new2} can be determined accordingly. After this step, the repair method is called. If a binding of a task maps the task to a resource which is not allocated, the binding is changed, such that it points to a resource which is already allocated.

Finally, if the crossover is neither in the allocation nor in the binding of tasks to resources, the crossover happens in the priority vector. For a randomly selected scenario b, the priority vectors $P_1[b]$ and $P_2[b]$ are crossed in one point to produce new priority vectors P_{new1} and P_{new2} following a similar procedure as described above.

Mutation. First, an exact copy of the individual to be mutated is created. With probability $1-P_{mut}$, no mutation takes place and the copy is returned. Otherwise, the copy is modified with respect to either the allocation, the task binding, or the priority assignment.

We mutate the allocation vector with probability $P_{mut-alloc}$. To this end, we randomly select a resource i and set $A_{new}[i] = 0$ with probability

$P_{mut-alloc-zero}$, otherwise we set $A_{new}[i] = 1$. After this change in the allocation vector, the repair method is called, which changes infeasible bindings such that they all map tasks to allocated resources only.

In case the mutation does not affect the allocation vector, with probability $P_{mut-bind}$ we mutate the binding vector $Z_{new}[b]$ for a randomly determined scenario $b \in B$. That is we randomly select a task and map it to a resource randomly selected from the specification. If the resource is not yet allocated in this solution, we additionally allocate it.

If we do neither mutate the allocation nor the binding, we mutate the priority vector for a randomly selected scenario b. We just exchange two flows within the priority list $P_{new}[b]$.

Now, the design space exploration problem is formally defined and we described the problem specification, the module for the representation of a single design point and the design evaluation module for the EXPO tool framework.

5.3 Different Search Algorithms

In this section, we discuss the use of the packet processor design space exploration problem, implemented within the EXPO tool framework, as a benchmark application. Among the various benchmark problems designed to compare and evaluate the performance of multi-objective optimisers, there is a lack of real-world applications that are commonly accepted and, even more important, are easy to use by different research groups. The main reason is, in our opinion, the high effort required to re-implement or adapt the corresponding programs. We address this problem with the EXPO tool framework, used to solve a real-world problem, namely the design space exploration for packet processors.

5.3.1 Benchmark Applications

The field of evolutionary multi-objective optimisation (EMO) has been growing rapidly since the first pioneering works in the mid-1980's and the early 1990's. Meanwhile numerous methods and algorithmic components are available, and accordingly there is a need for representative benchmark problems to compare and evaluate the different techniques.

Most test problems that have been suggested in the literature are artificial and abstract from real-world scenarios. Some authors considered multi-objective extensions of NP-hard problems such as the knapsack problem [ZT99], the set covering problem [Jas03], and the quadratic assignment problem [KC03]. Other benchmark problem examples are the Pseudo-Boolean functions introduced by Thierens [Thi03] and Laumanns *et al.* [LTZ04] that were designed mainly for theoretical investigations. Most popular, though, are real-valued functions [Deb01, CVL02]. For instance, several test functions representing different types of problem difficulties were proposed by Zitzler *et al.* [ZDT00] and Deb *et al.* [DTLZ02].

Although there exists no commonly accepted set of benchmark problems as, e.g., the SPEC benchmarks [SPE] in computer engineering, most of the aforementioned functions are used by different researchers within the EMO community. The reason is that the corresponding problem formulations are simple which in turn keeps the implementation effort low. However, the simplicity and the high abstraction level come along with a loss of information: various features and characteristics of real-world applications cannot be captured by these artificial optimisation problems. As a consequence, one has to test algorithms also on actual applications in order to obtain more reliable results. Complex problems in various areas have been tackled using multi-objective evolutionary algorithms, and many studies even compare two or several algorithms on a specific application [Deb01, CVL02]. The restricted reusability, though, has prohibited so far that one or several applications have established themselves as benchmark problems that are used by *different* research groups. Re-implementation is usually too work-intensive and error-prone, while re-compilation is often not possible because either the source code is not available, e.g., due to intellectual property issues, or particular software packages are needed that are not publicly available.

To overcome the problems discussed above, we present EXPO as a benchmark application for evolutionary algorithms that

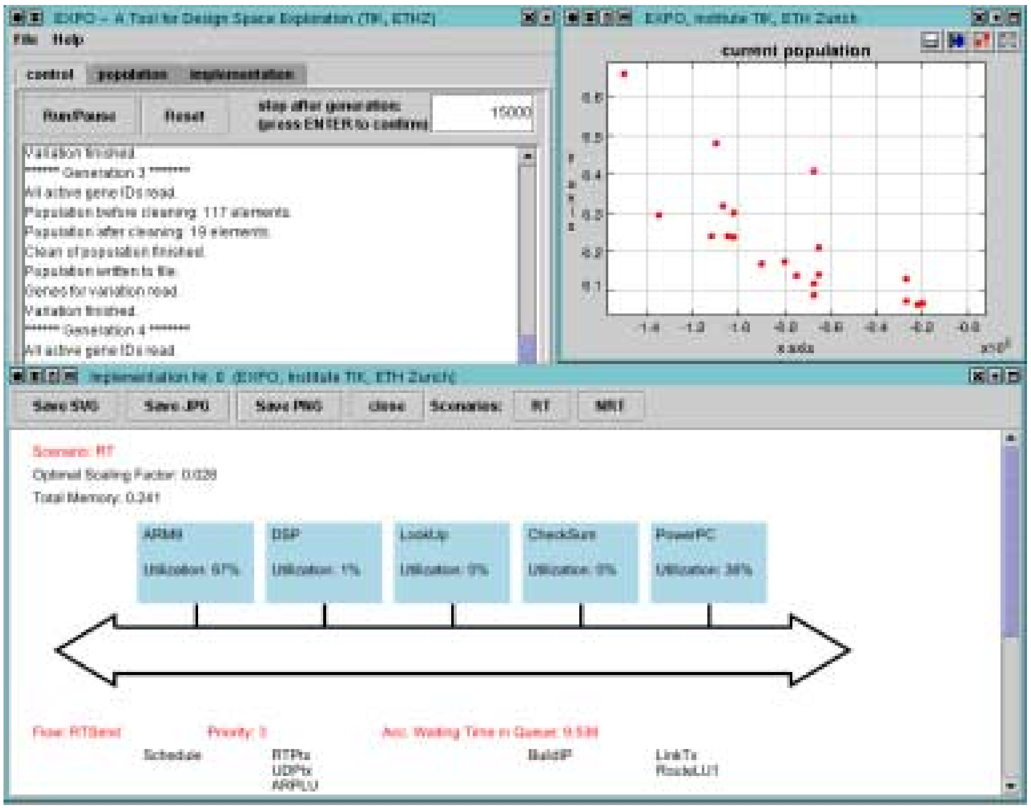
- provides a platform- and programming-language-independent interface that allows the usage of pre-compiled and executable programs and therefore circumvents the problem mentioned above,

- is scalable in terms of complexity, i.e., problem instances of different levels of difficulty are available, and

- is representative for several other applications in the area of computer design [BTT98, DJ98, ZTB00a].

5.3.2 Test Cases

The implementation of EXPO, which has been tested under Solaris, Linux and Microsoft Windows, provides a graphical user interface that allows to control the program execution (see Figure 62): the optimisation run can be halted, the current population can be plotted and individual processor designs can be inspected graphically.


EXPO – A Tool for Design Space Exploration (TIK, ETHZ)
File Help
control population implementation
Run/Pause
Reset
stop after generation: (press ENTER to confirm)
15000
EXPO, Institute TIK, ETH Zurich
current population
x axis
Implementation Nr. 0 (EXPO, Institute TIK, ETH Zurich)
Save SVG
Save JPG
Save PNG
close
Scenarios:
RT
NRT
ARM9
DSP
LookUp
CheckSum
PowerPC
Schedule
RTPtx
UDPtx
ARPLU
BuildIP
LinkTx
RouteLU1

```
Variation finished
****** Generation 3 *******
All active gene IDs read
Population before cleaning: 117 elements
Population after cleaning: 19 elements
Clean of population finished
Population written to file
Genes for variation read
Variation finished
****** Generation 4 *******
All active gene IDs read
```

Scenario: RT
Optimal Scaling Factor: 0.028
Total Memory: 0.341

ARM9 Utilization: 67%
DSP Utilization: 1%
LookUp Utilization: 0%
CheckSum Utilization: 0%
PowerPC Utilization: 38%

Flow: RTSend Priority: 3 Acc. Waiting Time in Queue: 9.538

Fig. 62: The user interface of the benchmark application: The main control window in the upper left, a plot the current population in the upper right and a graphical representation of a network processor configuration in the lower part.

In the following, we will present three problem instances for the packet processor application and demonstrate how to compare different evolutionary multi-objective optimisers on these instances.

5.3.2.1 Problem Instances

We consider three problem instances that are based on the packet processing application depicted in Figure 60. The set of available resources is the same for all the problem instances. The three problem instances differ in the number of objectives. We have defined a problem with 2 objectives, one with 3 and a scenario including 4 objectives. For all the different instances one objective is in common, the total cost of the allocated resources. The remaining objectives in a problem instance are the performance ψ_b of the solution packet processor under a given load scenario $b \in B$. In Table 12 the different load characteristics for the remaining objectives are shown. Overall, three different loads can be identified; in

Load scenario		RT send	RT receive	NRT encrypt	NRT decrypt	NRT forward
2 Objectives	Load 1	√	√	√	√	√
3 Objectives	Load 2	√	√	-	-	√
	Load 3	-	-	√	√	√
4 Objectives	Load 1	√	√	√	√	√
	Load 2	√	√	-	-	√
	Load 3	-	-	√	√	√

Tab. 12: Loads for the different scenarios for which the architecture should be optimised.

Load 1 all flows have to be processed; in *Load 2* there are only the three flows real-time voice receive and send, and non-real-time (NRT) packet forwarding present; in *Load 3*, the packet processor has to forward packets of flow 'NRT forward' and encrypt/decrypt packets of flows 'NRT encryption' and 'NRT decryption', respectively.

The size of the search space for the given instance can be computed as follows. In the problem setting, there are 4 resource types on which all the tasks can be performed. Therefore, we have more than 4^{25} possibilities to map the tasks on the resources. Furthermore, the solution contains a priority assignment to the different flows. There are 5! possibilities to assign priorities to the flows. So, if we take into account that there are other specialised resources available, the size of the search space is $S > 4^{25} \times 5! > 10^{17}$ already for the problem instance with 2 objectives and even larger for the instances with 3 or 4 objectives.

As an example, an approximated Pareto front for the 3-objective instance is shown in Figure 63 — the front has been generated by the optimiser SPEA2 [ZLT02]. The x-axis shows the objective value corresponding to ψ_{Load2} under *Load 2* (as defined in Table 12), the y-axis shows the objective value corresponding to ψ_{Load3}, whereas the z-axis shows the normalised total cost of the allocated resources.

The two example architectures shown in Figure 63 differ only in the allocation of the resource 'Cipher', which is a specialised hardware for encryption and decryption of packets. The performance of the two architectures for the load scenario with real-time flows to be processed is more or less the same. However, the architecture with a cipher unit performs around 30 times better for the encryption/decryption scenario, at increased cost for the cipher unit. So, a designer of a packet processor that should have the capability of encryption/decryption would go for the solution with a cipher unit (solution on the left in Figure 63), whereas one would decide for the cheaper solution on the right, if there is no need for encryption.

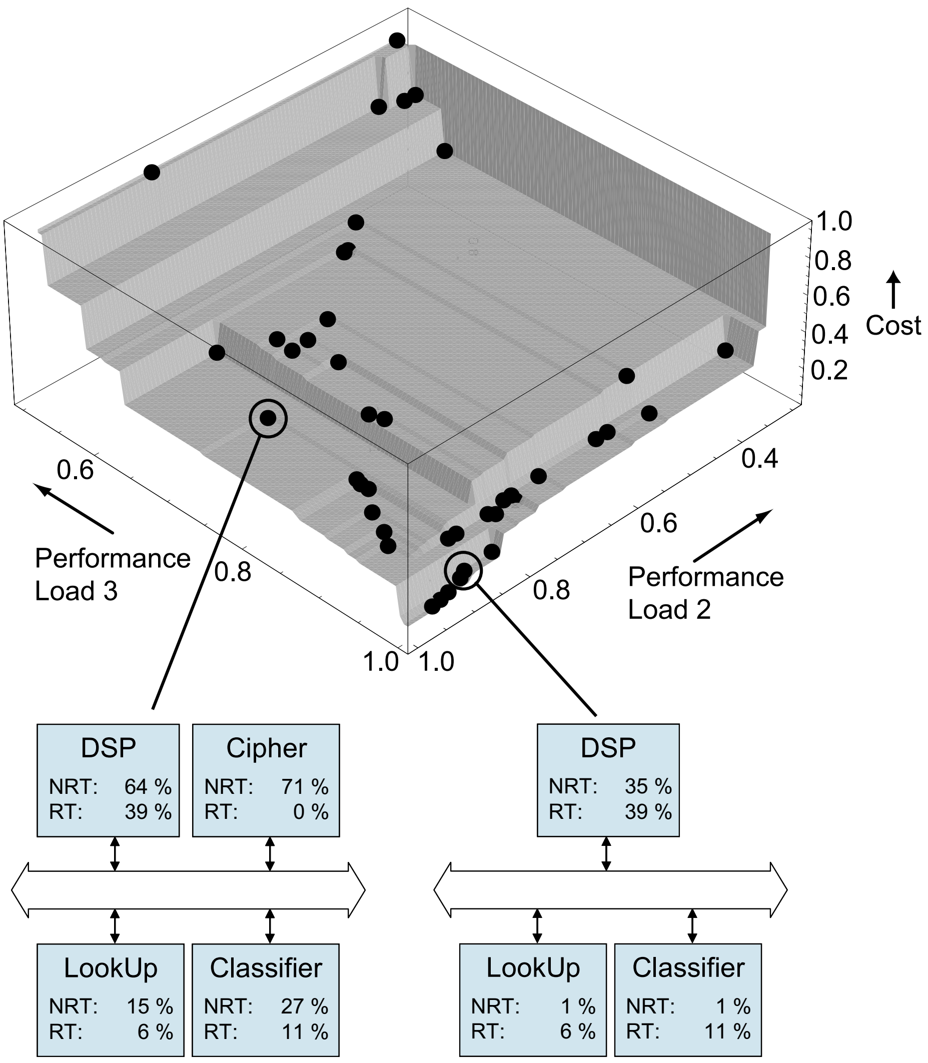


Fig. 63: Two solution packet processor architectures annotated with loads on resources for the different loads specified in Table 12.

5.3.2.2 Experimental Results

To evaluate the difficulty of the proposed benchmark application, we compared the performance of five evolutionary multi-objective optimisers, namely SPEA2 [ZLT02], NSGA-II [DPAM02], SEMO [LTZ04], FEMO [LTZ04], and IBEA (see Chapter 4) on the three aforementioned problem instances.

For each algorithm, 30 runs were performed using the parameter set-

# of objectives	population size	# of generations
2	100	200
3	150	300
4	200	400

Tab. 13: Parameters for population size and duration of runs dependent on the number of objectives.

Mutation		P_{mut}	=	0.8
$\rightarrow$	Allocation	$P_{mut-alloc}$	=	0.3
		$P_{mut-alloc-zero}$	=	0.5
$\rightarrow$	Binding	$P_{mut-bind}$	=	0.5
Crossover		P_{cross}	=	0.5
$\rightarrow$	Allocation	$P_{cross-alloc}$	=	0.3
$\rightarrow$	Binding	$P_{cross-bind}$	=	0.5

Tab. 14: Probabilities for mutation and crossover.

tings listed in Tables 13 and 14. These parameters were determined based on extensive, preliminary experiments. Furthermore, all objective functions were scaled such that the corresponding values lie within the interval $[0, 1]$. Note that all objectives are to be minimised, i.e., the performance values are reversed (smaller values correspond to better performance). The different runs were carried out on a Sun Ultra 60. A single run for 3 objectives, a population size of 150 individuals in conjunction with SPEA2 takes about 20 minutes to complete.

Figure 64 gives the populations of solutions of the 2-dimensional design space exploration problem found by two search algorithms in 30 runs after 50 generations each. Here, the optimal point is in the lower left corner, and it is quite “easy” to deduce from the figure that IBEA’s performance seems to be superior to SPEA2’s, because the points found by IBEA are closer to the optimal point than the ones found by SPEA2. Nevertheless, in general it is impossible to find such a “clear” situation, and more involved techniques to compare the performance of different algorithms have to be used.

Therefore, in the following we have used three binary performance measures for the comparison of the EMO techniques: (1) the additive ε-quality measure $I_{\varepsilon+}(A, R)$ [ZTL+03] presented in Section 4.2.1. In this formula, A stands for the output that the evolutionary algorithm produced. The reference set R was determined by merging all solutions found by all the different algorithms into one set and keeping the non-dominated solutions. R was used instead of the Pareto set S, because S is usually unknown.

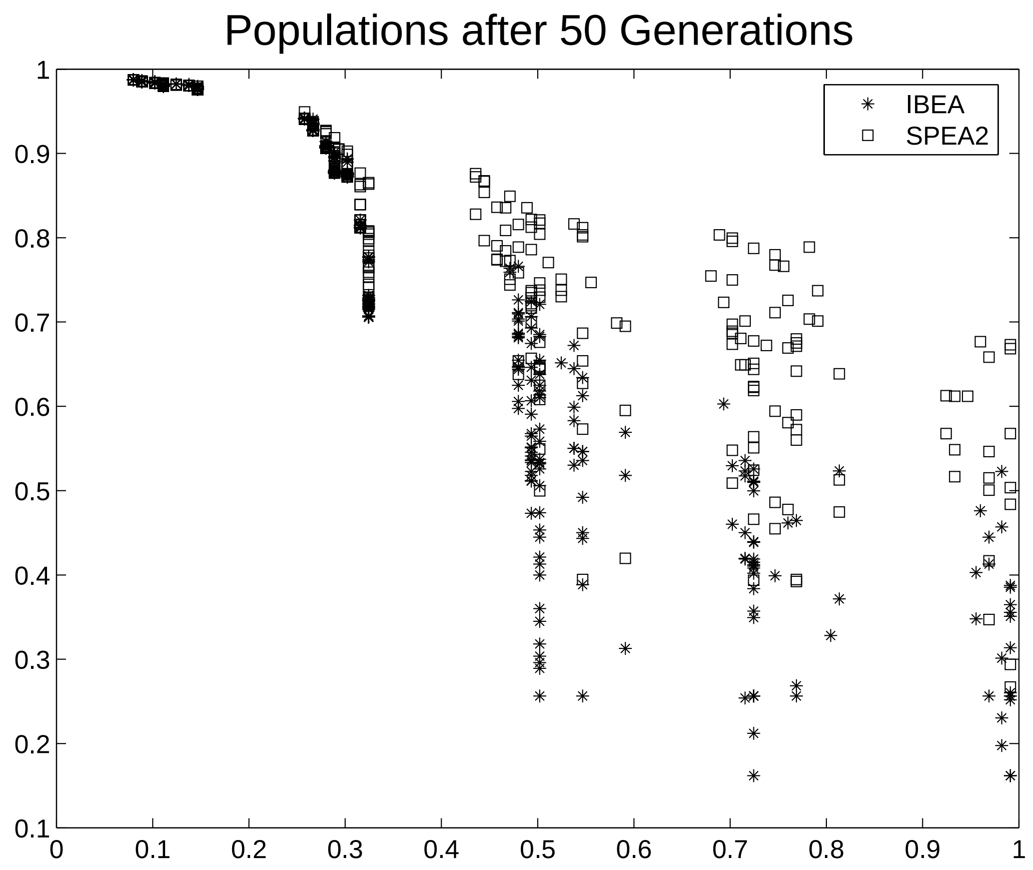


Fig. 64: Populations found by IBEA and SPEA2, for 30 different runs to solve the 2-dimensional problem. The x-axis gives the normalised cost of the solution whereas the y-axis gives the normalised inverse of the performance.

(2) The hypervolume indicator $I_{HD}(A, R)$ presented in Section 4.2.5. This indicator measures the difference in hypervolume that is spanned by the points in set A compared to the hypervolume that is spanned by the reference set R. (3) The coverage measure [ZT99] $I_C(A, B)$ presented in Section 4.2.3. In this formula A and B stand for the output sets of the two evolutionary algorithms for which to compute the coverage indicator.

Overall, we can state that there exist differences in the performance of different evolutionary algorithms on the packet processor design space exploration benchmark problem. FEMO and IBEA outperform all other optimisers for the two-dimensional problem with respect to the coverage measure. In the case with two objectives, SEMO performs better than SPEA2 and NSGA-II, while SPEA2 achieves a higher coverage compared to NSGA-II. The box plots with all results for the coverage measure in case of the problem with 2 objectives are given in Figure 65.

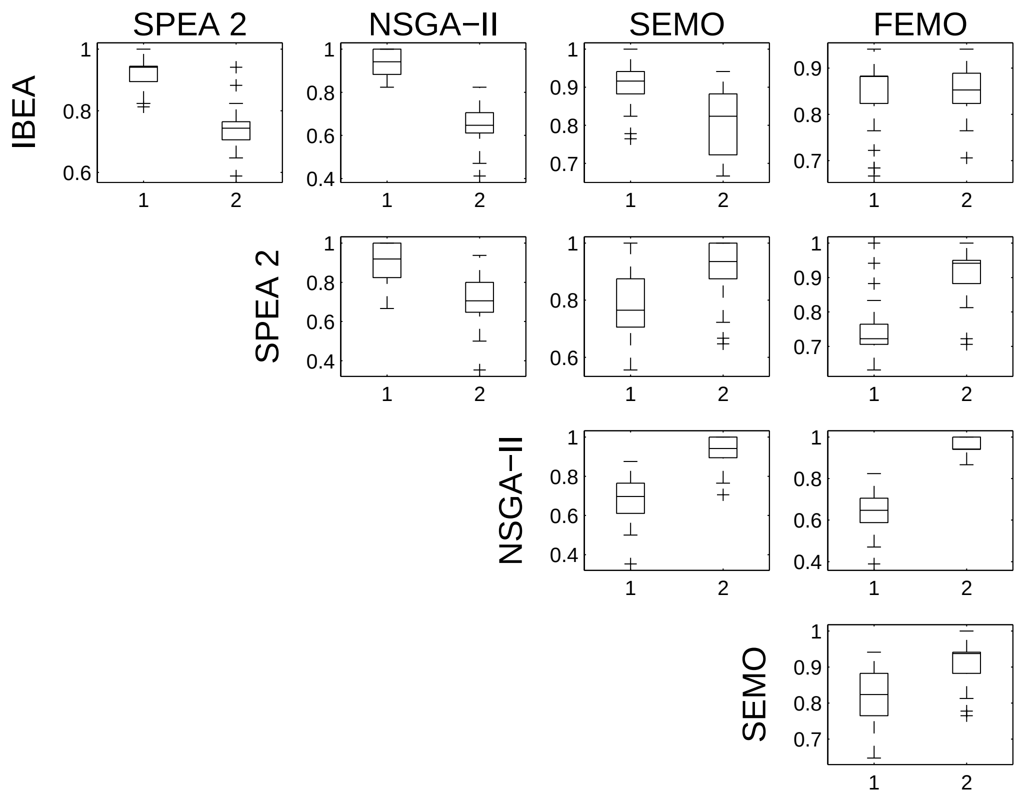


Fig. 65: Coverage plots for the five search algorithms used for the experiments. The plots are given for the 2-dimensional design space exploration problem. The plots are to be read in the following way: the left box plot gives the coverage of the solution set found by the search algorithm indicated by the row over solution set found by the search algorithm marked at the column. The second box plot gives the coverage for the column-based search algorithm over the row-based algorithm.

For the problem with three objectives (cf. Figure 66), the results for the coverage measure are different. In this case, IBEA performs worst of all algorithms investigated. FEMO achieves the best coverage values of all algorithms, SPEA2 now achieves better coverage than NSGA-II. These results differ from the results presented in [KBTZ04]. This is due to the difference in the comparison method. In [KBTZ04] the active populations at certain points in time were compared, whereas we compare the archive of all Pareto-optimal solutions found so far. For 4 dimensions, FEMO still performs best with respect to the coverage measure. IBEA, SEMO, and SPEA2 show a slightly better performance than NSGA-II for this problem (cf. Figure 67).

If we compare the results for the coverage measure for the different problems, we can notice that the optimisers perform differently for the

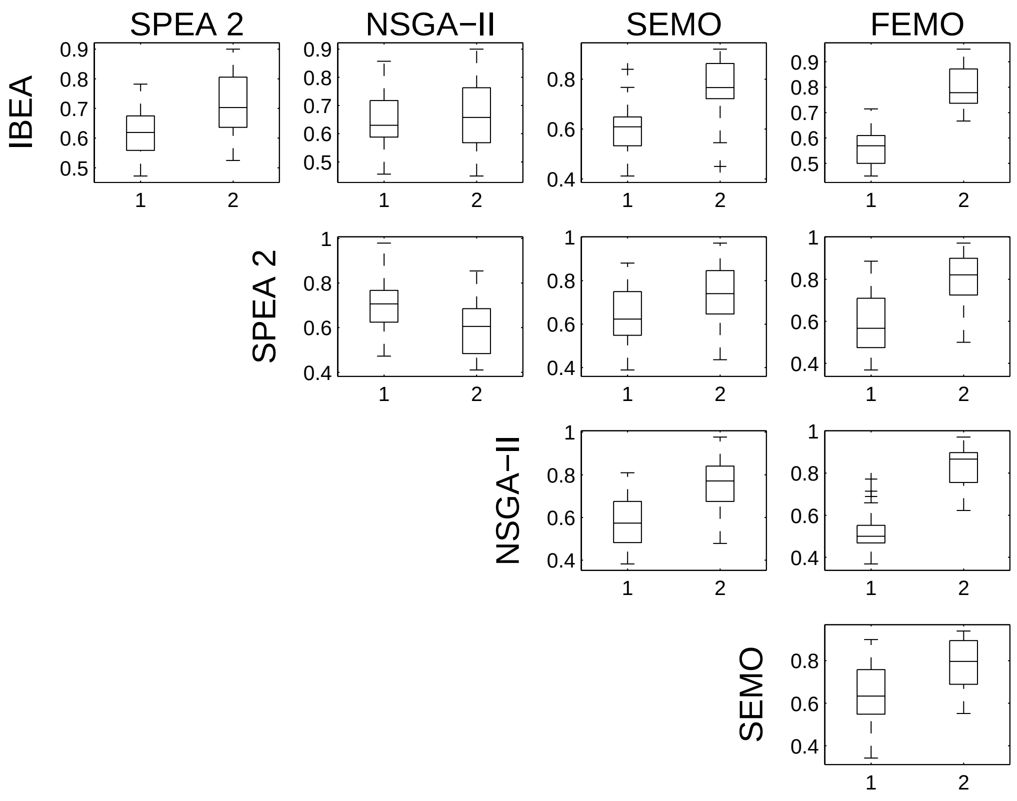


Fig. 66: Coverage plots for the five search algorithms used for the experiments. The plots are given for the 3-dimensional design space exploration problem. The plots are to be read in the following way: the left box plot gives the coverage of the solution set found by the search algorithm indicated by the row over solution set found by the search algorithm marked at the column. The second box plot gives the coverage for the column-based search algorithm over the row-based algorithm.

different problems (EXPO2, EXPO3, or EXPO4). Especially IBEA's performance measured with the coverage indicator depends much on the problem (good performance for EXPO2 and EXPO4 w.r.t. the coverage measure, worse performance for EXPO3). The reason for this behaviour seems not to be the increase in the number of objectives as IBEA shows a good performance for 4 dimensions. IBEA's weaker performance for 3 dimensions could lie in the nature of the problem. Here, we assess the fitness of a solution by computing the cost, and the performance for two different load scenarios. In contrast to the 2-dimensional case with a single load scenario for performance assessment, and the 4-dimensional case with 3 partly overlapping usage scenarios, the 2 load scenarios used for the 3-dimensional problem are almost disjoint, as can be seen in Table 12.

Figure 68 shows the box plots for the indicators $I_{\varepsilon+}$ and I_{HD} for the three

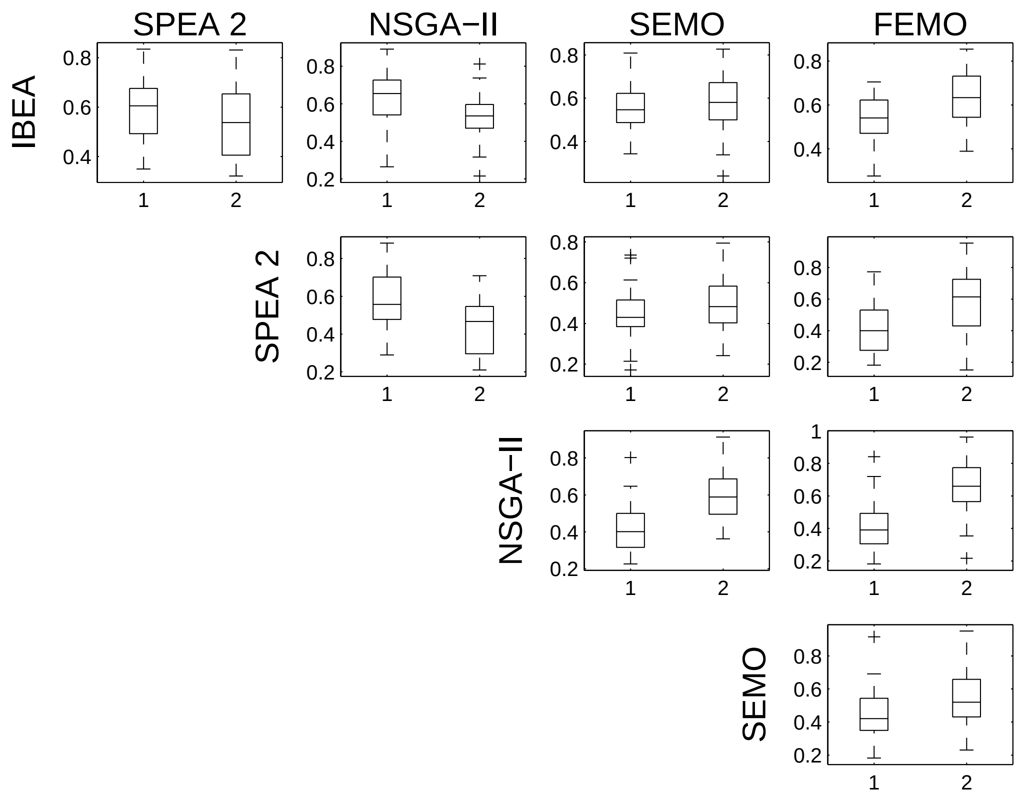


Fig. 67: Coverage plots for the five search algorithms used for the experiments. The plots are given for the 4-dimensional design space exploration problem. The plots are to be read in the following way: the left box plot gives the coverage of the solution set found by the search algorithm indicated by the row over solution set found by the search algorithm marked at the column. The second box plot gives the coverage for the column-based search algorithm over the row-based algorithm.

problem instances. As for the coverage measure used before, the search algorithms perform with different success with respect to the indicators used. From the figure, we can intuitively conclude that IBEA outperforms all other algorithms, this is at least true for the problem with 4 objectives. For the other two problems, it can already be difficult to decide whether one optimiser really shows better performance than another, or the difference in the result is just caused by random effects. To solve this problem, we used statistical significance tests as proposed in [KTZ05a].

For the tests, the populations found by two different algorithms are compared. To do so, we first calculate the indicator values that correspond to the populations, this leads to two collections of indicator values. In the case of the experiments in this section, there are 30 values for each of the algorithms. For the tests we would now like to know, whether the

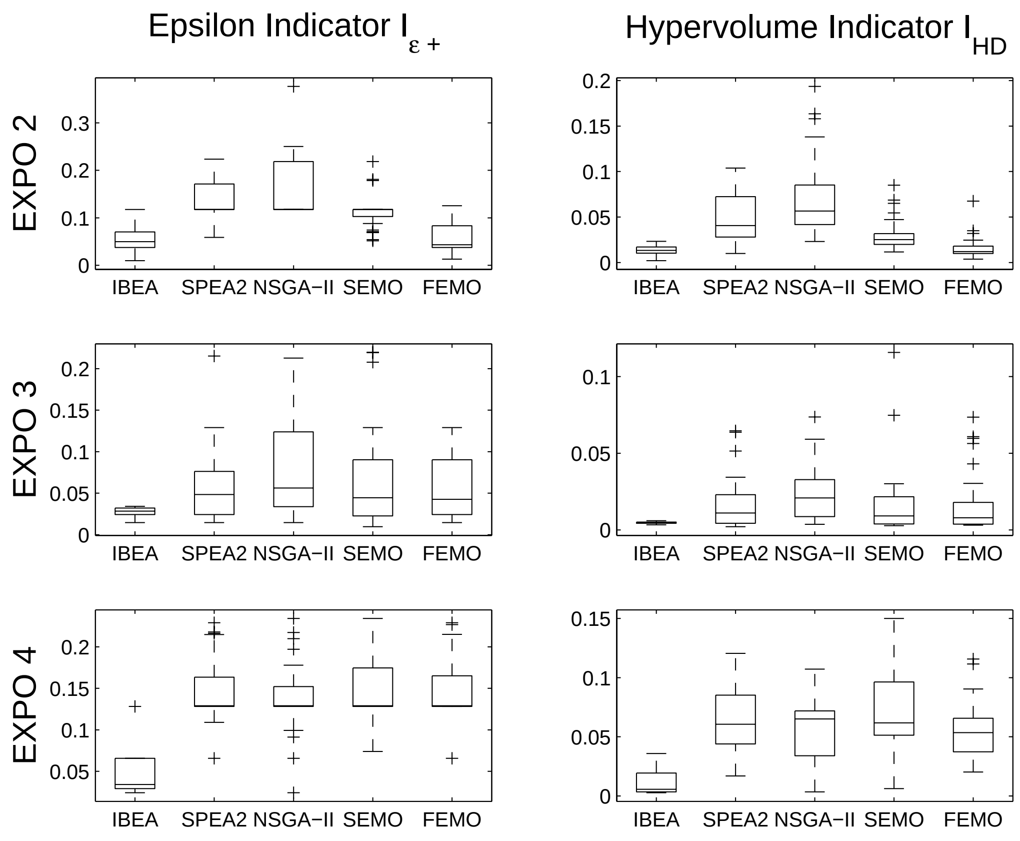


Fig. 68: $I_\varepsilon+$ and I_{HD} for the 5 search algorithms on all 3 problems after the maximum number of generations per experiment.

two collections could be derived from the same data, or whether different data sets are more likely to produce these two collections. The tests are based on the so-called null hypothesis, or short H_0. The null hypothesis H_0 claims, that two samples are drawn from the same distribution. In our case this means that both the collections of indicator values are derived from the same population.

A $p-$value lower than a certain significance level α denotes that the null hypothesis can be rejected in favour of the alternative hypothesis H_A, at the significance level α. The alternative hypothesis H_A is of the form: 'sample A comes from a better distribution than sample B'. Usually, for this sort of tests, a significance level of $\alpha = 0.05$ is used. In Tables 15 ($I_{\varepsilon+}$) and 16 (I_{HD}), we give the results of the Kruskal-Wallis test with a significance level $\alpha = 0.05$. For this test given in the table, the alternative hypothesis is of the form: 'the indicator values for the algorithm given in the row are drawn from a better distribution than the indicator values for the algorithm given in the column'. Note that for the experimental

EXPO 2, $I_{\varepsilon+}$					
	IBEA	SPEA2	NSGA-II	SEMO	FEMO
IBEA	-	1.810^{-17}	5.610^{-19}	1.210^{-09}	> 0.05
SPEA2	> 0.05	-	> 0.05	> 0.05	> 0.05
NSGA-II	> 0.05	> 0.05	-	> 0.05	> 0.05
SEMO	> 0.05	0.00079	0.00010	-	> 0.05
FEMO	> 0.05	7.610^{-16}	2.410^{-17}	2.810^{-08}	-
EXPO 3, $I_{\varepsilon+}$					
	IBEA	SPEA2	NSGA-II	SEMO	FEMO
IBEA	-	0.0029	2.210^{-05}	0.011	0.0071
SPEA2	> 0.05	-	> 0.05	> 0.05	> 0.05
NSGA-II	> 0.05	> 0.05	-	> 0.05	> 0.05
SEMO	> 0.05	> 0.05	0.030	-	> 0.05
FEMO	> 0.05	> 0.05	0.042	> 0.05	-
EXPO 4, $I_{\varepsilon+}$					
	IBEA	SPEA2	NSGA-II	SEMO	FEMO
IBEA	-	$1.21-^{-15}$	5.810^{-12}	1.010^{-15}	1.210^{-13}
SPEA2	> 0.05	-	> 0.05	> 0.05	> 0.05
NSGA-II	> 0.05	> 0.05	-	> 0.05	> 0.05
SEMO	> 0.05	> 0.05	> 0.05	-	> 0.05
FEMO	> 0.05	> 0.05	> 0.05	> 0.05	-

Tab. 15: Kruskal-Wallis test applied to the additive $\varepsilon-$indicator $I_{\varepsilon+}(A,R)$ for the 2-, 3- and 4-dimensional exploration problem. The values indicate the p-values to accept the null hypothesis based on the assumption that optimiser 1 (row) is better than optimiser 2 (column).

results given in this section no adjustment of the significance level is necessary, e.g., according to Bonferroni correction, as the Kruskal-Wallis test is used [KTZ05a].

From the results of the statistical tests, we can see that there exists a significant difference in the solutions found by the different evolutionary algorithms. The performance of IBEA is very good compared to the other algorithms with respect to the indicator values $I_{\varepsilon+}$ and I_{HD} with a significance level of $\alpha = 5\%$. FEMO shows a comparably well performance for the 2-dimensional problem, but for the problems with higher dimension, FEMO is inferior to IBEA. NSGA-II performs significantly worse than the other algorithms for 2 and 3 dimensions with respect to the hypervolume indicator I_{HD}, where there is no significant difference between NSGA-II, SPEA2 and SEMO for 4 dimensions.

The development of a performance indicator for an algorithm over time leads to another interesting observation. Figure 69 shows the addi-

EXPO 2, I_{HD}					
	IBEA	SPEA2	NSGA-II	SEMO	FEMO
IBEA	-	9.810^{-19}	7.610^{-25}	2.610^{-09}	> 0.05
SPEA2	> 0.05	-	0.010	> 0.05	> 0.05
NSGA-II	> 0.05	> 0.05	-	> 0.05	> 0.05
SEMO	> 0.05	8.410^{-05}	2.910^{-09}	-	> 0.05
FEMO	> 0.05	3.210^{-17}	2.710^{-23}	4.610^{-08}	-

EXPO 3, I_{HD}					
	IBEA	SPEA2	NSGA-II	SEMO	FEMO
IBEA	-	0.00028	1.810^{-07}	0.0046	0.0063
SPEA2	> 0.05	-	0.036	> 0.05	> 0.05
NSGA-II	> 0.05	> 0.05	-	> 0.05	> 0.05
SEMO	> 0.05	> 0.05	0.0039	-	> 0.05
FEMO	> 0.05	> 0.05	0.0028	> 0.05	-

EXPO 4, I_{HD}					
	IBEA	SPEA2	NSGA-II	SEMO	FEMO
IBEA	-	1.910^{-16}	1.210^{-13}	1.410^{-16}	2.810^{-12}
SPEA2	> 0.05	-	> 0.05	> 0.05	> 0.05
NSGA-II	> 0.05	> 0.05	-	> 0.05	> 0.05
SEMO	> 0.05	> 0.05	> 0.05	-	> 0.05
FEMO	> 0.05	0.048	> 0.05	0.042	-

Tab. 16: Kruskal-Wallis test applied to the hypervolume indicator $I_{HD}(A, R)$ for the 2-, 3- and 4-dimensional exploration problem. The values indicate the p-values to accept the null hypothesis based on the assumption that optimiser 1 (row) is better than optimiser 2 (column).

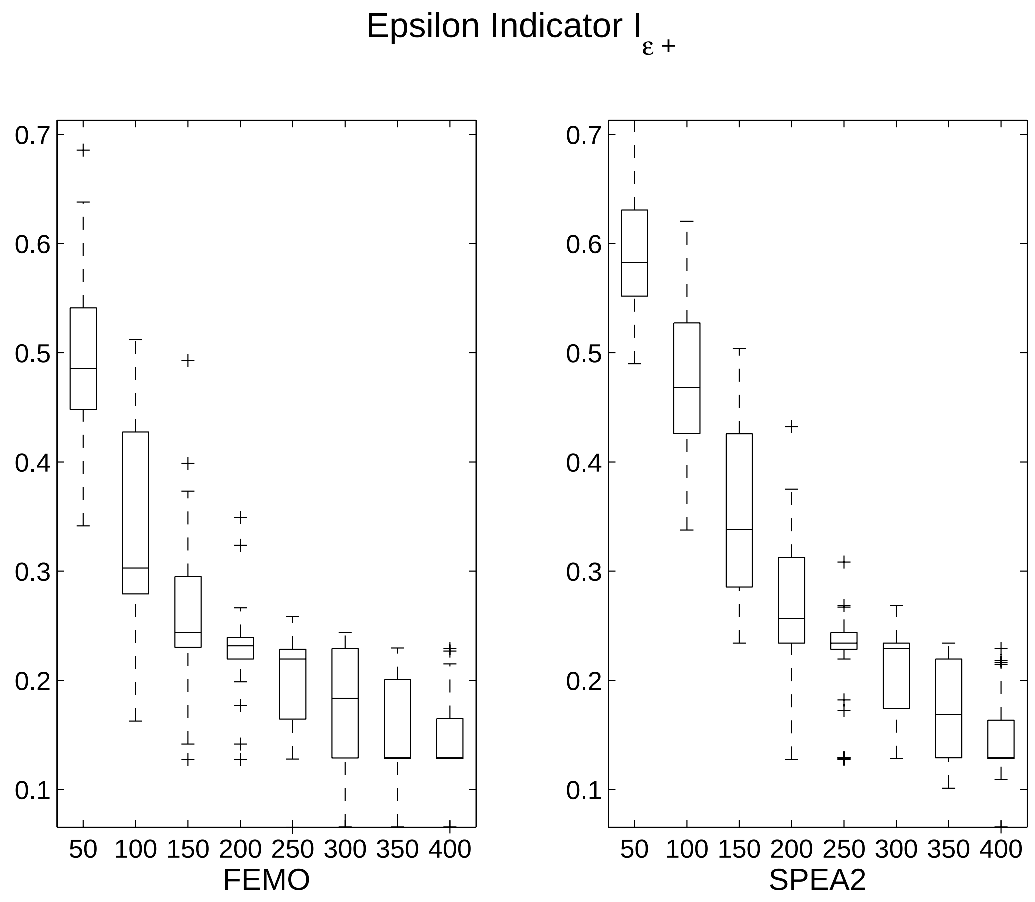


Fig. 69: Indicator value $I_{\varepsilon+}$ vs. number of generations for FEMO and SPEA2 on the problem with 4 objectives.

tive ε-indicator for FEMO and SPEA2 after 50, 100,. . . , 400 generations. It can be noted that FEMO seems to find good solutions faster than SPEA2, leading to a smaller indicator value $I_{\varepsilon+}$ for 50 generations. After 400 generations SPEA2 catches up and there is no significant difference between the performance of the two algorithms any more. The question of when, or after how many generations, to compare algorithms therefore also influences the results of the experimental study. The number of generations that we propose for our benchmark problems was found by performing different preliminary experiments.

5.4 Summary

In this chapter we presented EXPO, a general framework for design space exploration. It consists of a main module that can be coupled with search algorithms through the PISA interface. Moreover, well defined interfaces exist that must be implemented by the user for each design space exploration problem. We presented two different applications of the framework and discussed the implementation of the problem-specific modules.

The packet processor design space exploration problem was used as benchmark application for search algorithms. We presented a comparative study with 5 different evolutionary algorithms. The differences in the performance of the search algorithms are significant and therefore the design space exploration problem is suitable as benchmark problem to assess the quality of different search algorithms.

The EXPO tool, with the packet processor application implemented and several evolutionary algorithms can be downloaded from the PISA web page at `http://www.tik.ee.ethz.ch/pisa`.

6

Conclusions

In this chapter, we summarise our contributions and discuss potential extensions of the work for future research.

6.1 Contributions

In this thesis we described our contributions to address the task of design space exploration for embedded systems. Namely, we presented the results summarised in the following list.

- We described a new performance evaluation method. The method makes use of a combination of existing performance analysis approaches. We proposed the needed interfaces between the different methods, and showed the applicability of the approach using a case study.

- We presented a new evolutionary search algorithm that directly incorporates the user's preference. The algorithm is adaptive to the fitness values and therefore needs only little configuration, i.e. only a few parameters have to be set. For several test problems, including the design space exploration for packet processors problem, the new algorithm IBEA shows a good performance.

- A software framework for design space exploration was developed. For a new DSE problem, a user has only to implement the problem-specific parts, i.e. the problem specification, the representation of a solution, and the method for design point evaluation, but not the parts that are common to all DSE problems. These common parts can be re-used from the EXPO tool. It is fully written in Java and is therefore platform-independent. It implements the PISA-interface for communication with popular randomised search algorithms.

6.2 Future Work

For future research, we could think of the following extensions of the presented work:

- Extend the newly proposed performance evaluation method. On the one hand, we should investigate how we can increase the possible range of applications, e.g. by allowing feedback loops. On the other hand, we could define the proposed interfaces not only for the existing performance evaluation methods used for this work, but also other existing analysis techniques. Having a generic definition of the interfaces, we could come up with component-based performance evaluation models, where we could decide per component which performance evaluation method/model to use.

- Apply the presented performance evaluation method to larger example systems. Doing so, we could get information about the scalability, and applicability of the approach also for systems including more processing elements and more complex communication structures than the ones presented.

- Extend the proposed trace generation algorithm such that not only traces can be generated that either comply with the upper or the lower specification curve. But in addition to this behaviour also an algorithm exposing variable “greediness” should be investigated, e.g. an algorithm that produces a trace that is at ε-distance from the upper and lower curves. Like this more complex simulation traces could be generated.

- Investigate other performance indicators that could be adapted for the use with IBEA. With the existing PISA framework these new adapted IBEA versions could be easily tested against several test problems and statistics about their performance could be gathered.

A

The EXPO Tool Framework

In this chapter the EXPO tool framework for design space exploration and its components are described in more detail. Figure 70 gives an overview over all parts of the framework.

The framework is written in Java, which makes it platform independent. The main components are the main module for control and population handling, the graphical user interface, and the interfaces to the search algorithm (PISA-compliant) and the exploration-problem specific parts (both grey-shaded). All the design space exploration problems introduced in this thesis were solved using the EXPO tool framework. Like this, all the software for the file handling and population handling could be reused and didn't have to be recoded for all the individual problems.

To perform a design space exploration using EXPO in conjunction with PISA compliant search algorithms it is sufficient to implement the representation of a solution, how new solutions are found and how the fitness of solution is evaluated. To do so, a user just has to write Java classes that implement the interfaces presented in the next sections. These classes are then accessed by the main module of the EXPO framework to perform the design space exploration with a PISA compliant search algorithm.

In the following subsections the interfaces that have to be implemented by the user are described in more detail.

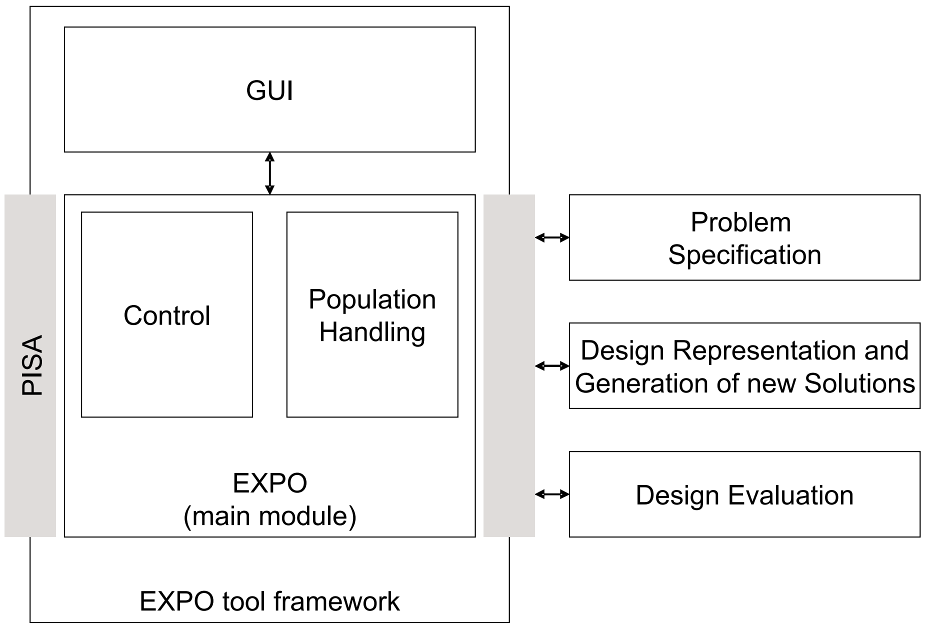


Fig. 70: Overview over the components of the EXPO tool framework.

A.1 Interface Specification

This interface separates the EXPO tool framework from the problem specification. It consists of four methods which have to be implemented for the tool to work correctly.

`String simpleFileInput(File file)`

The method will be called by the EXPO tool framework during the initialisation phase. It can be used to read all specification data that is given for instance in a text file. The file handle is passed as an argument to the method. The method has to return an error code as a string. The implementation of this method is optional.

`int getProblemDimension()`

This method has to return the number of objectives the full problem deals with. This method will be called by the controller and the graphical user interface of EXPO to be able to plot the population.

`double getMutationProbability()`

This method has to return the probability with which an individual has to be mutated. This probability has to be a double value between 0.0 and 1.0, whereas 1.0 means that all the genes will have to be mutated.

`double getCrossoverProbability()`

This method has to return the probability with which two individuals have to be crossed. This probability has to be a double value between 0.0 and 1.0, whereas 1.0 means that all the genes will have to be crossed.

A.2 Interface Gene

The following methods are defined in the interface Gene. A solution for the design space exploration problem has to implement this interface. The EXPO tool framework accesses a solution with its user-defined code only via these interfaces.

`String getFitnessString()`

This method has to return the fitness values of the gene, by returning a String with space-separated double values. If the values are not yet known, they have first to be computed and then returned as String.

`ArrayList getFitnessValues()`

This method has to return an ArrayList which contains all fitness values of the Gene. If this method is called and these values do not yet exist then they have to be computed first and then returned.

`Gene mutateGene()`

This method is called if the evolutionary algorithm has decided that the solution should be mutated. It has to create a new object of type Gene, which is returned by this method, based on the actual solution.

`Gene[] crossOverGene(Gene geneToCross)`

This method is called if the evolutionary algorithm decided that the actual Gene should be recombined with the Gene given as argument to this method. After this crossover is performed, two new solutions exist and they have to be returned in an array of two objects of type Gene.

`String getFullDescription()`

This method has to return a detailed description of the gene in plain text form, if implemented. The implementation of this method is optional.

`String getShortDescription()`

This method has to return a short description of the gene in plain text form, if implemented. The implementation of this method is optional.

`Object clone()`

This method has to be overwritten. It is called by the framework to produce deep copies of a gene.

A.3 Interface Analyzer

The EXPO framework accesses the design point evaluation module through this interface.

`ArrayList analyze(Gene gene)`

This method of the Analyzer interface has to analyze a gene which is given by parameter gene and has to return an ArrayList with all the fitness values of the gene as doubles. The larger a fitness value is, the weaker is the gene in that objective.

`String getReport(Gene gene)`
This method has to return a report in a textual form. In the string, all facts of the gene, such as e.g. allocated resources etc. are written. The implementation of this method is optional.

`String drawReport(Gene gene)`
This method has in some sense to graphically show all the parameters of the gene specified in the parameter gene. If there is no graphical representation this method should return a string containing an error message. The implementation of this method is optional.

B

Approximations for Real-Time Calculus

In this chapter, we give approximated formulas for the remaining arrival and service curves as introduced in Chapter 3. The results given here correspond to Equations (3.3) – (3.6) by approximating the input and output curves using three line segments. For ease of reading, we replicate the equations from Chapter 3.

$$\begin{aligned}
\alpha^{l\prime}(\Delta) &= \min\{\inf_{0\le\mu\le\Delta}\{\sup_{\lambda>0}\{\alpha^l(\mu+\lambda)-\beta^u(\lambda)\}+\beta^l(\Delta-\mu)\},\beta^l(\Delta)\} \\
\alpha^{u\prime}(\Delta) &= \min\{\sup_{\lambda>0}\{\inf_{0\le\mu<\lambda+\Delta}\{\alpha^u(\mu)+\beta^u(\lambda+\Delta-\mu)\}-\beta^l(\lambda)\},\beta^u(\Delta)\} \\
\beta^{l\prime}(\Delta) &= \sup_{0\le\lambda\le\Delta}\{\beta^l(\lambda)-\alpha^u(\lambda)\} \\
\beta^{u\prime}(\Delta) &= \max\{\inf_{\lambda>\Delta}\{\beta^u(\lambda)-\alpha^l(\lambda)\},0\}
\end{aligned}$$

Figure 71 shows the arrival and service curves consisting of three linear pieces. This form allows us to exactly model an arrival curve in the form of a T-SPEC [SW97]. In the case of an arrival curve, here q_1^u may represent the maximum workload because of a single packet, r^u can be interpreted as the burst rate and s^u the long term arrival rate. The approximation enables an efficient implementation of the formulas.

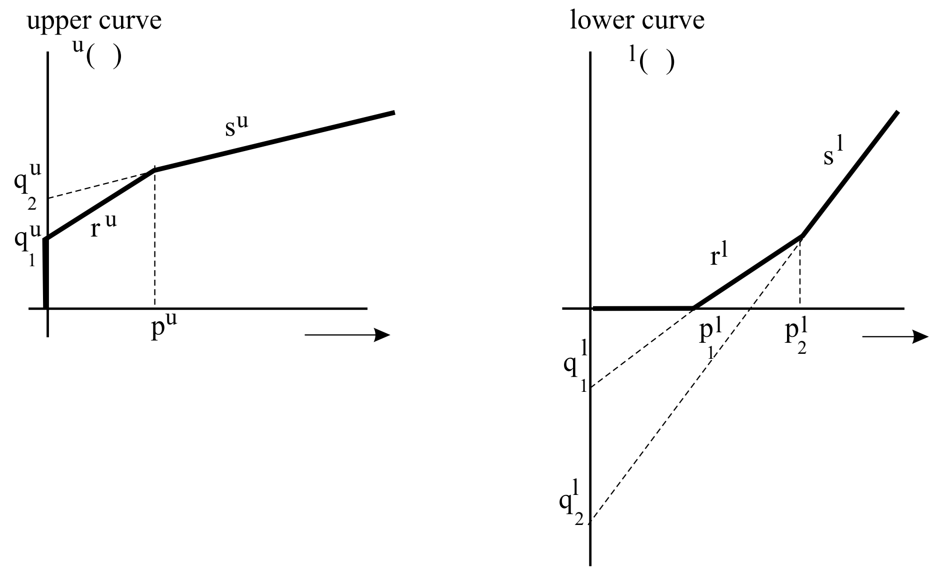


Fig. 71: Three-piece linear approximation of upper and lower curves.

The upper and the lower curves in this case can be written as:

$$
\begin{aligned}
\gamma^u(\Delta) &= \begin{cases} \min\{q_1^u + r^u\Delta, q_2^u + s^u\Delta\} & \text{if } \Delta > 0 \\ 0 & \text{if } \Delta = 0 \end{cases} \\
\gamma^l(\Delta) &= \max\{q_2^l + s^l\Delta, q_1^l + r^l\Delta, 0\}
\end{aligned}
$$

where,

$$
\begin{array}{lll}
q_2^u \geq q_1^u \geq 0, & r^u \geq s^u \geq 0, & r^u = s^u \Leftrightarrow q_1^u = q_2^u \\
q_2^l \leq q_1^l \leq 0, & 0 \leq r^l \leq s^l, & r^l = s^l \Leftrightarrow q_1^l = q_2^l
\end{array}
$$

The values of p^u and p_1^l, p_2^l (see Figure 71) can be calculated as:

$$
p^u = \begin{cases} \frac{q_2^u - q_1^u}{r^u - s^u} & \text{if } r^u > s^u \\ 0 & \text{if } r^u = s^u \end{cases}
$$

$$
p_1^l = \begin{cases} -\frac{q_1^u}{r^l} & \text{if } r^l > 0 \\ 0 & \text{if } r^l = 0 \end{cases}, \qquad p_2^l = \begin{cases} \frac{q_2^l - q_1^l}{r^l - s^l} & \text{if } r^l < s^l \\ p_1^l & \text{if } r^l = s^l \end{cases}
$$

We denote the curves γ^u and γ^l in this case by $U(q_1, q_2, r, s)$ and $L(q_1, q_2, r, s)$ respectively.

Prop. 3: **(Simple Lower Arrival Curve)** *Given the lower arrival and upper and lower service curves $\alpha^l = L(q_{1\alpha}, q_{2\alpha}, r_\alpha, s_\alpha)$, $\beta^u = U(q_{1\beta u}, q_{2\beta u}, r_{\beta u}, s_{\beta u})$, and $\beta^l = L(q_{1\beta l}, q_{2\beta l}, r_{\beta l}, s_{\beta l})$ respectively, the approximate outgoing lower arrival curve $\alpha^{l\prime} = L(q_1, q_2, r, s)$ can be given as the following:*

If $s_\alpha \leq s_{\beta u}$:

$$\begin{aligned}
r &= \begin{cases} r_\alpha, & if\, r_\alpha \leq r_{\beta l} \\ r_{\beta l}, & if\, r_\alpha > r_{\beta l} \end{cases} \\
s &= \begin{cases} s_\alpha, & if\, s_\alpha \leq s_{\beta l} \\ s_{\beta l}, & if\, s_\alpha > s_{\beta l} \end{cases} \\
q_1 &= \begin{cases} \frac{r_\alpha}{r_{\beta l}} q_{1\beta l} + q_{1\alpha}, & if\, r_\alpha \leq r_{\beta l} \\ \frac{r_{\beta l}}{r_\alpha} q_{1\alpha} + q_{1\beta l}, & if\, r_\alpha > r_{\beta l} \end{cases} \\
q_2 &= \begin{cases} q_{2\beta l} + q_{1\alpha} + (r_\alpha - s_{\beta l}) \frac{q_{2\alpha} - q_{1\alpha}}{r_\alpha - s_\alpha}, & if\, r_\alpha < s_{\beta l} \leq s_\alpha \\ q_{2\beta l} + \frac{s_{\beta l}}{r_\alpha} q_{1\alpha}, & if\, s_{\beta l} \leq r_\alpha \\ q_{2\alpha} + \frac{s_\alpha}{r_{\beta l}} q_{1\beta l}, & if\, s_\gamma \leq r_{\beta l} \\ q_{2\alpha} + q_{1\beta l} + (r_{\beta l} - s_\alpha) \frac{q_{2\beta l} - q_{1\beta l}}{r_{\beta l} - s_{\beta l}}, & if\, r_{\beta l} < s_\alpha \leq s_{\beta l} \end{cases}
\end{aligned}$$

If $s_\alpha > s_{\beta u}$:

$$\begin{aligned}
r &= r_{\beta l} \\
s &= s_{\beta l} \\
q_1 &= q_{1\beta l} \\
q_2 &= q_{2\beta l}
\end{aligned}$$

Derivation: Following [BT01], we can rewrite $\alpha^{l\prime}(\Delta) = \min\{\inf_{0 \leq \mu \leq \Delta}\{\sup_{\lambda > 0}\{\alpha^l(\mu + \lambda) - \beta^u(\lambda)\} + \beta^l(\Delta - \mu)\}, \beta^l(\Delta)\}$ as $\alpha^{l\prime} = \min\{(\alpha^l \oslash \beta^u) \otimes \beta^l, \beta^l\}$. First of all, we calculate the innermost parentheses $\alpha^l \oslash \beta^u$ and set $\gamma = \alpha^l \oslash \beta^u$.

We have to distinguish two cases for this calculation. In the first case, the long term arrival rate s_α for the lower arrival curve is less or equal compared to the long term service rate of the upper service curve $s_{\beta u}$. In this first case, the formula $\alpha^l \oslash \beta^u$ results in α^l, as we find the maximum of $\alpha^l(\mu + \lambda) - \beta^u(\lambda)$ for $\lambda = 0$, because β^u is larger than α^l for all $\Delta \geq 0$. In the second case, where $s_\alpha > s_{\beta u}$, $\alpha^l \oslash \beta^u$ tends to infinity, i.e. $\gamma(0) = 0$, $\gamma(\Delta > 0) \to \infty$.

The next formula to evaluate is $\gamma \otimes \beta^l$. In the first case described above, where $\gamma = \alpha^l$, we can reuse a result from [BT01]. It states that in the case of piecewise linear convex curves α^l and β^l the resulting curve is the

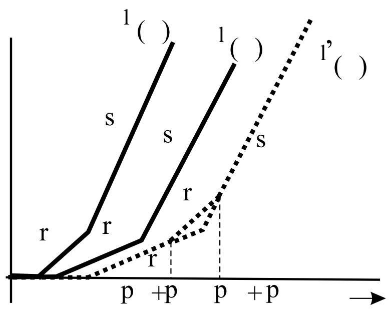


Fig. 72: Approximation of $\alpha^{l'}$.

concatenation of the linear pieces, ordered according to their slope. As $r_\alpha \leq s_\alpha$ and $r_\beta \leq s_\beta$, the slope of $\inf_{0\leq u\leq\Delta}\left\{\alpha^l(u)+\beta^l(\Delta-u)\right\}$ for $\Delta \rightarrow 0$ is $r = \min\{r_\alpha, r_\beta\}$. In a similar way, we find $s = \min\{s_\alpha, s_\beta\}$ if we consider $\Delta \rightarrow \infty$.

The offset q_1 can be determined considering two cases. If $r_\alpha \leq r_\beta$ then $q_1 = -r_\alpha(p_{1\alpha} + p_{1\beta})$ because the resulting curve crosses the x-axis at point $p_{1\alpha} + p_{1\beta}$. If $r_\alpha > r_\beta$ then $q_1 = -r_\beta(p_{1\alpha} + p_{1\beta})$.

The offset q_2 can be calculated considering four cases. If $s_\beta < r_\alpha$ the resulting curve is just the initial curve β^l shifted by $p_{1\alpha}$ to the right. Therefore, q_2 is determined by $q_{2\beta} - s_\beta p_{1\alpha}$. In the same way, if $s_\alpha < r_\beta$ then $q_2 = q_{2\alpha} - s_\alpha p_{2\alpha}$. The remaining two cases are more involved. In general, if we calculate a resulting lower arrival curve from curves with three linear pieces, it will consist of four linear pieces. To again obtain a curve with three linear pieces, we have to approximate the resulting curve. How we do this here can be seen in Figure 72.

If $r_\alpha < s_\beta < s_\alpha$, then $q_2 + s_\beta(p_{2\alpha} + p_{2\beta}) = r_\alpha(p_{2\alpha} - p_{1\alpha}) + r_\beta(p_{2\beta} - p_{1\beta})$. This case is shown in Figure 73. If $r_\beta < s_\alpha < s_\beta$, then $q_2 + s_\alpha(p_{2\alpha} + p_{2\beta}) = r_\alpha(p_{2\alpha} - p_{1\alpha}) + r_\beta(p_{2\beta} - p_{1\beta})$.

In the second case, with $\gamma(0) = 0$ and $\gamma(\Delta > 0) \rightarrow \infty$ the formula $\gamma \otimes \beta^l$ is simpler to solve, the resulting curve is just β^l.

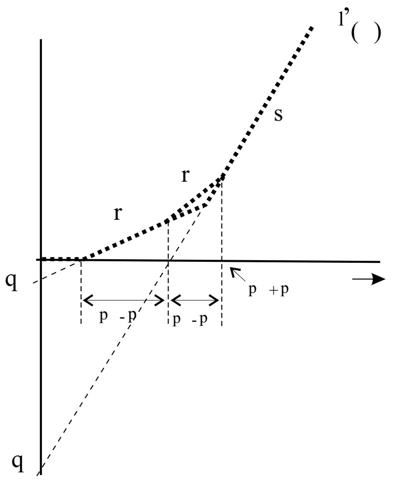


Fig. 73: Approximation of $\alpha^{l'}$ in the case $r_\alpha < s_\beta < s_\alpha$.

Prop. 4: (Simple Upper Arrival Curve) *Given arrival curves and service curves $\alpha^u = U(q_{1\alpha}, q_{2\alpha}, r_\alpha, s_\alpha)$, $\beta^l = L(q_{1\beta l}, q_{2\beta l}, r_{\beta l}, s_{\beta l})$ and $\beta^u = U(q_{1\beta u}, q_{2\beta u}, r_{\beta u}, s_{\beta u})$. Then the processed arrival curve can be approximated by the curve*

$$\alpha^{u\prime} = U(q_1, q_2, r, s)$$

where

$$\text{if } s_\alpha \geq s_{\beta u} \wedge q_{1\alpha} \geq q_{1\beta u} \quad \begin{cases} q_{1\gamma} &= q_{1\beta u} \\ q_{2\gamma} &= q_{2\beta u} \\ r_\gamma &= r_{\beta u} \\ s_\gamma &= s_{\beta u} \end{cases}$$

$$\text{if } s_\alpha \geq s_{\beta u} \wedge q_{1\alpha} < q_{1\beta u} \quad \begin{cases} q_{1\gamma} &= q_{1\alpha} \\ q_{2\gamma} &= q_{2\beta u} \\ r_\gamma &= r_\alpha \\ s_\gamma &= s_{\beta u} \end{cases}$$

$$\text{if } s_\alpha < s_{\beta u} \wedge q_{1\alpha} < q_{1\beta u} \quad \begin{cases} q_{1\gamma} &= q_{1\alpha} \\ q_{2\gamma} &= q_{2\alpha} \\ r_\gamma &= r_\alpha \\ s_\gamma &= s_\alpha \end{cases}$$

$$\text{if } s_\alpha < s_{\beta u} \wedge q_{1\alpha} \geq q_{1\beta u} \quad \begin{cases} q_{1\gamma} &= q_{1\beta u} \\ q_{2\gamma} &= q_{2\alpha} \\ r_\gamma &= r_{\beta u} \\ s_\gamma &= s_\alpha \end{cases}$$

$$\text{if } p_\gamma < p_{1\beta l} \wedge s_\gamma \le r_{\beta l} \qquad \begin{cases} q_{1\delta} &= q_{2\gamma} + s_\gamma p_{1\beta l} \\ q_{2\delta} &= q_{2\gamma} + s_\gamma p_{1\beta l} \\ r_\delta &= s_\gamma \\ s_\delta &= s_\gamma \end{cases}$$

$$\text{if } p_{1\beta l} \le p_\gamma \wedge r_\gamma \le r_{\beta l} \qquad \begin{cases} q_{1\delta} &= q_{1\gamma} + r_\gamma p_{1\beta l} \\ q_{2\delta} &= q_{2\gamma} + s_\gamma p_\gamma \\ r_\delta &= r_\gamma \\ s_\delta &= s_\gamma \end{cases}$$

$$\text{if } p_{1\beta l} \le p_\gamma \le p_{2\beta l} \wedge s_\gamma \le r_{\beta l} < r_\gamma \qquad \begin{cases} q_{1\delta} &= q_{2\gamma} + s_\gamma p_\gamma - (q_{1\beta l} + r_{\beta l} p_\gamma) \\ q_{2\delta} &= q_{2\gamma} + s_\gamma p_\gamma - (q_{1\beta l} + r_{\beta l} p_\gamma) \\ r_\delta &= s_\gamma \\ s_\delta &= s_\gamma \end{cases}$$

$$\text{if } p_{2\beta l} < p_\gamma \wedge r_{\beta l} < r_\gamma \le s_{\beta l} \qquad \begin{cases} q_{1\delta} &= q_{1\gamma} + r_\gamma p_{2\beta l} - (q_{1\beta l} + r_{\beta l} p_{2\beta l}) \\ q_{2\delta} &= q_{1\gamma} - q_{1\beta l} + (r_\gamma - r_{\beta l}) p_{2\beta l} + \\ & \quad (r_\gamma - s_\gamma)(p_\gamma - p_{2\beta l}) \\ r_\delta &= r_\gamma \\ s_\delta &= s_\gamma \end{cases}$$

$$\text{if } p_\gamma \le p_{2\beta l} \wedge r_{\beta l} < s_\gamma \le s_{\beta l} \qquad \begin{cases} q_{1\delta} &= q_{2\gamma} + s_\gamma p_{2\beta l} - (q_{1\beta l} + r_{\beta l} p_{2\beta l}) \\ q_{2\delta} &= q_{2\gamma} + s_\gamma p_{2\beta l} - (q_{1\beta l} + r_{\beta l} p_{2\beta l}) \\ r_\delta &= s_\gamma \\ s_\delta &= s_\gamma \end{cases}$$

$$\text{if } p_{2\beta l} < p_\gamma \wedge s_\gamma \le s_{\beta l} < r_\gamma \qquad \begin{cases} q_{1\delta} &= q_{2\gamma} + s_\gamma p_\gamma - (q_{2\beta l} + s_{\beta l} p_\gamma) \\ q_{2\delta} &= q_{2\gamma} + s_\gamma p_\gamma - (q_{2\beta l} + s_{\beta l} p_\gamma) \\ r_\delta &= s_\gamma \\ s_\delta &= s_\gamma \end{cases}$$

$$\text{if } s_{\beta l} < s_\gamma \qquad \begin{cases} q_{1\delta} &= \infty \\ q_{2\delta} &= \infty \\ r_\delta &= \infty \\ s_\delta &= \infty \end{cases}$$

and finally

$$\text{if } s_\delta \geq s_{\beta u} \wedge q_{1\delta} \geq q_{1\beta u} \quad \begin{cases} q_1 &= q_{1\beta u} \\ q_2 &= q_{2\beta u} \\ r &= r_{\beta u} \\ s &= s_{\beta u} \end{cases}$$

$$\text{if } s_\delta \geq s_{\beta u} \wedge q_{1\delta} < q_{1\beta u} \quad \begin{cases} q_1 &= q_{1\delta} \\ q_2 &= q_{2\beta u} \\ r &= r_{\delta} \\ s &= s_{\beta u} \end{cases}$$

$$\text{if } s_\delta < s_{\beta u} \wedge q_{1\delta} < q_{1\beta u} \quad \begin{cases} q_1 &= q_{1\delta} \\ q_2 &= q_{2\delta} \\ r &= r_{\delta} \\ s &= s_{\delta} \end{cases}$$

$$\text{if } s_\delta < s_{\beta u} \wedge q_{1\delta} \geq q_{1\beta u} \quad \begin{cases} q_1 &= q_{1\beta u} \\ q_2 &= q_{2\delta} \\ r &= r_{\beta u} \\ s &= s_{\delta} \end{cases}$$

Derivation: The remaining upper arrival curve $\alpha^{u\prime}$ is calculated in three steps from the formula $\alpha^{u\prime} = \min\{\sup_{\lambda\geq 0}\{\inf_{0\leq\mu<\lambda+\Delta}\{\alpha^u(\mu) + \beta^u(\lambda + \Delta - \mu)\} - \beta^l(\lambda)\}, \beta^u(\Delta)\}$, which can be written as $\alpha^{u\prime} = \min\{(\alpha^u \otimes \beta^u) \oslash \beta^l, \beta^u\}$. First, we calculate the innermost parentheses only and get an upper curve $\gamma^u = U(q1, q2, r, s)$, with $\gamma^u = \alpha^u \otimes \beta^u$. To compute γ^u, from [BT01], we know that for two concave curves passing through the origin, $\alpha^u \otimes \beta^u$ equals to the minimum of the two curves. Therefore, to compute γ^u, we have to distinguish the 4 cases depicted in Figure 74.

In a next step, we compute $\delta^u = \gamma^u \oslash \beta^l$. For this calculation, we have to look at 18 different cases. All different combinations of γ^u and β^l have to be inspected. As a result, we can distinguish 7 different cases for δ^u. In Table 18 all cases with corresponding δ^u is provided.

In Table 19 all the different cases (A, B, ...) are further described. In the figures in Table 18 the vertical arrow indicates the value for $q_{1\delta}$ of δ^u which is the largest vertical distance between γ^u and β^l. All other points of the δ^u curve are obtained by shifting the γ^u curve in such a way, that the lower end of this arrow lies at the origin. Some of the cases can be combined to regions which are described with capital letters. A representation of these regions is given in Figure 75.

Once we found the values for $q_{1\delta}$, $q_{2\delta}$, r_δ and s_δ for the δ^u curve, to compute $\alpha^{u\prime}$ we only need δ^u and the initial upper service curve β^u. From

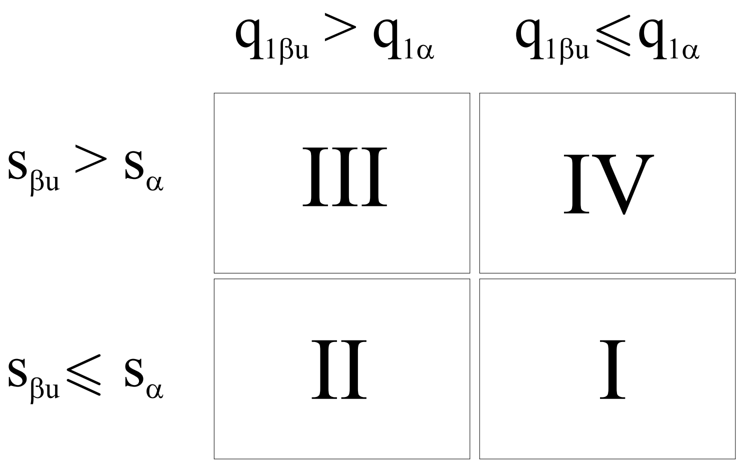
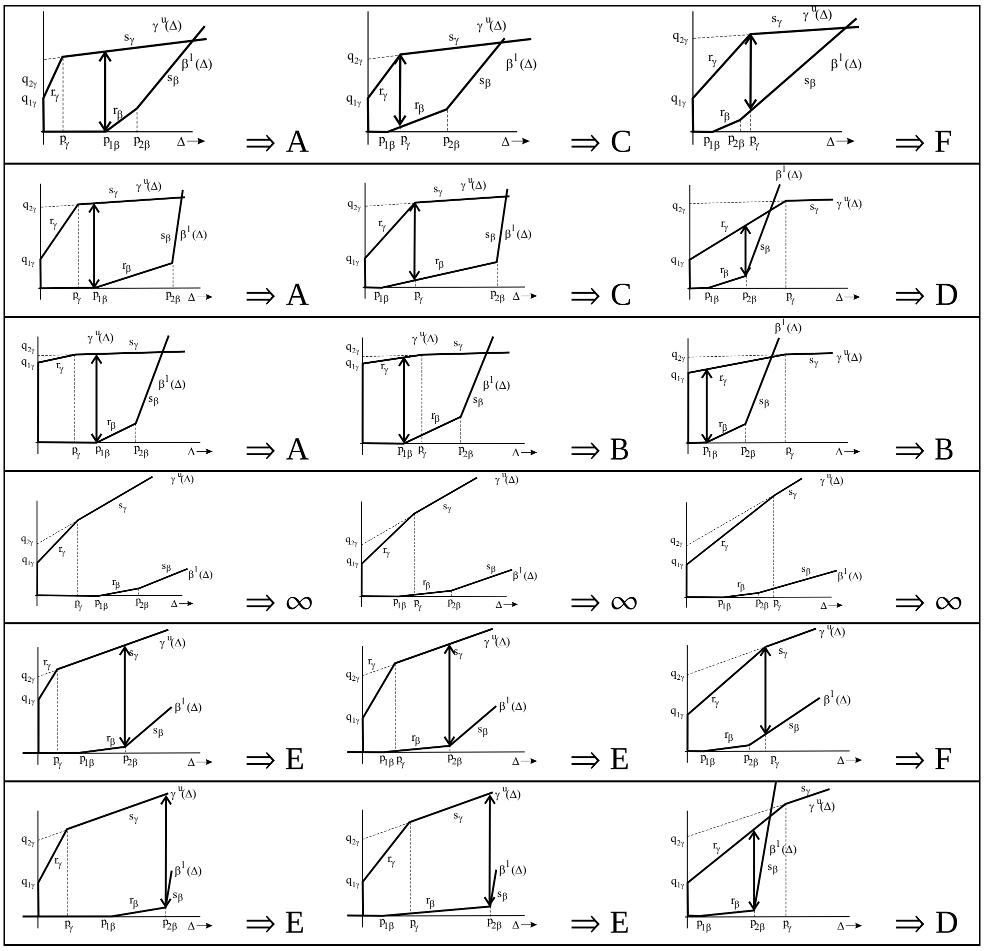

$q_{1\beta u} > q_{1\alpha}$
$q_{1\beta u} \leqslant q_{1\alpha}$
$s_{\beta u} > s_{\alpha}$
$s_{\beta u} \leqslant s_{\alpha}$
III
IV
II
I

Fig. 74: Different cases for $\gamma^u = \alpha^u \wedge \beta^u$.

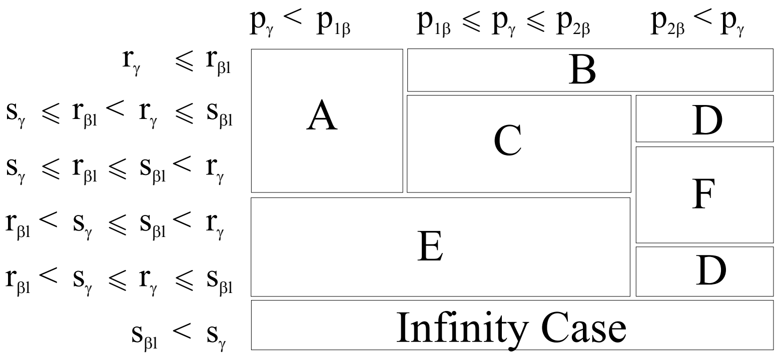
Case I	$q_{1\gamma}$	$=$	$q_{1\beta u}$
	$q_{2\gamma}$	$=$	$q_{2\beta u}$
	r_{γ}	$=$	$r_{\beta u}$
	s_{γ}	$=$	$s_{\beta u}$
Case II	$q_{1\gamma}$	$=$	$q_{1\alpha}$
	$q_{2\gamma}$	$=$	$q_{2\alpha}$
	r_{γ}	$=$	r_{α}
	s_{γ}	$=$	$s_{\beta u}$
Case III	$q_{1\gamma}$	$=$	$q_{1\alpha}$
	$q_{2\gamma}$	$=$	$q_{2\alpha}$
	r_{γ}	$=$	r_{α}
	s_{γ}	$=$	s_{α}
Case IV	$q_{1\gamma}$	$=$	$q_{1\beta u}$
	$q_{2\gamma}$	$=$	$q_{2\alpha}$
	r_{γ}	$=$	$r_{\beta u}$
	s_{γ}	$=$	s_{α}

Tab. 17: Four different cases for $\alpha^u \wedge \beta^u$.



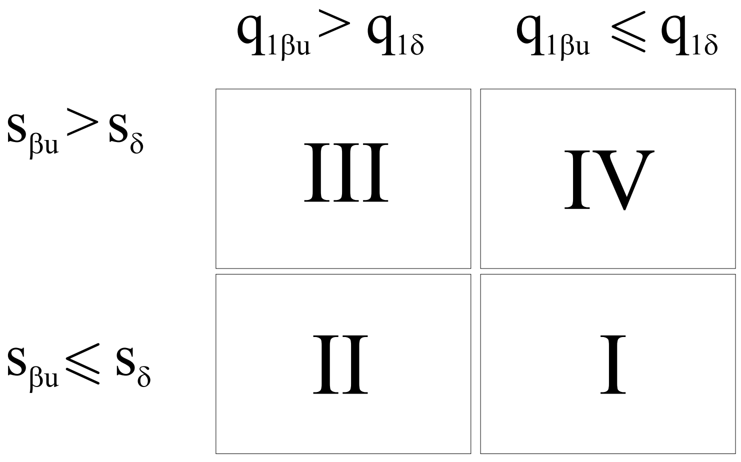
Tab. 18: All combinations of γ^u and β^l to compute δ^u.

the two curves, we have to calculate the minimum, which is again done by distinguishing 4 cases as shown in Figure 76. The resulting values for q_1, q_2, r and s are given in Table 20.



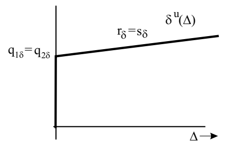
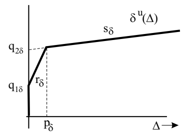
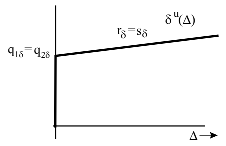
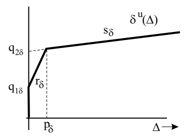
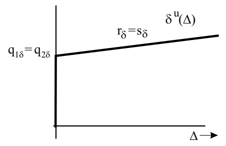
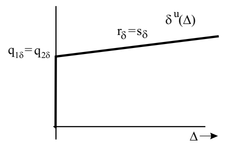
$p_\gamma < p_{1\beta}$
$p_{1\beta} \leqslant p_\gamma \leqslant p_{2\beta}$
$p_{2\beta} < p_\gamma$
$r_\gamma \leqslant r_{\beta 1}$
$s_\gamma \leqslant r_{\beta 1} < r_\gamma \leqslant s_{\beta 1}$
$s_\gamma \leqslant r_{\beta 1} \leqslant s_{\beta 1} < r_\gamma$
$r_{\beta 1} < s_\gamma \leqslant s_{\beta 1} < r_\gamma$
$r_{\beta 1} < s_\gamma \leqslant r_\gamma \leqslant s_{\beta 1}$
$s_{\beta 1} < s_\gamma$
A
B
C
D
F
E
D
Infinity Case

Fig. 75: Regions for different cases to compute δ^u.



$q_{1\beta u} > q_{1\delta}$
$q_{1\beta u} \leqslant q_{1\delta}$
$s_{\beta u} > s_\delta$
$s_{\beta u} \leqslant s_\delta$
III
IV
II
I

Fig. 76: Different cases for $\delta^u \wedge \beta^u$.

Case A	 $\delta^u(\Delta)$ $r_\delta=s_\delta$ $q_{1\delta}=q_{2\delta}$ $\Delta\rightarrow$	$\begin{aligned} q_{1\delta} &= q_{2\gamma}+s_\gamma p_{1\beta l} \\ q_{2\delta} &= q_{2\gamma}+s_\gamma p_{1\beta l} \\ r_\delta &= s_\gamma \\ s_\delta &= s_\gamma \end{aligned}$
Case B	 $\delta^u(\Delta)$ s_δ $q_{2\delta}$ r_δ $q_{1\delta}$ p_δ $\Delta\rightarrow$	$\begin{aligned} q_{1\delta} &= q_{1\gamma}+r_\gamma p_{1\beta l} \\ q_{2\delta} &= q_{2\gamma}+s_\gamma p_\gamma \\ r_\delta &= r_\gamma \\ s_\delta &= s_\gamma \end{aligned}$
Case C	 $\delta^u(\Delta)$ $r_\delta=s_\delta$ $q_{1\delta}=q_{2\delta}$ $\Delta\rightarrow$	$\begin{aligned} q_{1\delta} &= (q_{2\gamma}+s_\gamma p_\gamma)-(q_{1\beta l}+r_{\beta l}p_\gamma) \\ q_{2\delta} &= (q_{2\gamma}+s_\gamma p_\gamma)-(q_{1\beta l}+r_{\beta l}p_\gamma) \\ r_\delta &= s_\gamma \\ s_\delta &= s_\gamma \end{aligned}$
Case D	 $\delta^u(\Delta)$ s_δ $q_{2\delta}$ r_δ $q_{1\delta}$ p_δ $\Delta\rightarrow$	$\begin{aligned} q_{1\delta} &= (q_{1\gamma}+r_\gamma p_{2\beta l})-(q_{1\beta l}+r_{\beta l}p_{2\beta l}) \\ q_{2\delta} &= q_{1\gamma}-q_{1\beta l}+(r_\gamma-r_{\beta l})p_{2\beta l} \\ &\quad +(r_\gamma-s_\gamma)(p_\gamma-p_{2\beta l}) \\ r_\delta &= r_\gamma \\ s_\delta &= s_\gamma \end{aligned}$
Case E	 $\delta^u(\Delta)$ $r_\delta=s_\delta$ $q_{1\delta}=q_{2\delta}$ $\Delta\rightarrow$	$\begin{aligned} q_{1\delta} &= (q_{2\gamma}+s_\gamma p_{2\beta l})-(q_{1\beta l}+r_{\beta l}p_{2\beta l}) \\ q_{2\delta} &= (q_{2\gamma}+s_\gamma p_{2\beta l})-(q_{1\beta l}+r_{\beta l}p_{2\beta l}) \\ r_\delta &= s_\gamma \\ s_\delta &= s_\gamma \end{aligned}$
Case F	 $\delta^u(\Delta)$ $r_\delta=s_\delta$ $q_{1\delta}=q_{2\delta}$ $\Delta\rightarrow$	$\begin{aligned} q_{1\delta} &= (q_{2\gamma}+s_\gamma p_\gamma)-(q_{2\beta l}+s_{\beta l}p_\gamma) \\ q_{2\delta} &= (q_{2\gamma}+s_\gamma p_\gamma)-(q_{2\beta l}+s_{\beta l}p_\gamma) \\ r_\delta &= s_\gamma \\ s_\delta &= s_\gamma \end{aligned}$
Case ∞		$\begin{aligned} q_{1\delta} &= \infty \\ q_{2\delta} &= \infty \\ r_\delta &= \infty \\ s_\delta &= \infty \end{aligned}$

Tab. 19: Explicit values for $q_{1\delta}, q_{2\delta}, r_\delta$ and s_δ.

Case I	q_1	$=$	$q_{1\beta u}$
	q_2	$=$	$q_{2\beta u}$
	r	$=$	$r_{\beta u}$
	s	$=$	$s_{\beta u}$
Case II	q_1	$=$	$q_{1\delta}$
	q_2	$=$	$q_{2\beta u}$
	r	$=$	r_{δ}
	s	$=$	$s_{\beta u}$
Case III	q_1	$=$	$q_{1\delta}$
	q_2	$=$	$q_{2\delta}$
	r	$=$	r_{δ}
	s	$=$	s_{δ}
Case IV	q_1	$=$	$q_{1\beta u}$
	q_2	$=$	$q_{2\delta}$
	r	$=$	$r_{\beta u}$
	s	$=$	s_{δ}

Tab. 20: Four different cases for $\delta^u \wedge \beta^u$.

Prop. 5: **(Remaining Lower Service Curve)** *Given the upper arrival and lower service curves $\alpha^u = U(q_{1\alpha}, q_{2\alpha}, r_\alpha, s_\alpha)$ and $\beta^l = L(q_{1\beta}, q_{2\beta}, r_\beta, s_\beta)$ respectively, the approximate remaining service curve $\beta^{l'} = L(q_1, q_2, r, s)$ can be given as the following.*

Case 1: For some $\Delta' > 0$, $q_{2\beta} + s_\beta\Delta' = q_{2\alpha} + s_\alpha\Delta'$, and for all $\Delta < \Delta'$, $\alpha^u(\Delta) > \beta^l(\Delta)$.

In this case,

$$\begin{aligned} r &= 0, & s &= s_\beta - s_\alpha \\ q_1 &= 0, & q_2 &= q_{2\beta} - q_{2\alpha} \end{aligned}$$

Case 2: For some $\Delta' > 0$, $q_{1\beta} + r_\beta\Delta' = q_{2\alpha} + s_\alpha\Delta'$, and for all $\Delta < \Delta'$, $\alpha^u(\Delta) > \beta^l(\Delta)$.

In this case,

$$\begin{aligned} r &= r_\beta - s_\alpha, & s &= s_\beta - s_\alpha \\ q_1 &= q_{1\beta} - q_{2\alpha}, & q_2 &= q_{2\beta} - q_{2\alpha} \end{aligned}$$

Case 3: For some $\Delta' > 0$, $q_{1\beta} + r_\beta\Delta' = q_{1\alpha} + r_\alpha\Delta'$, and for all $\Delta < \Delta'$, $\alpha^u(\Delta) > \beta^l(\Delta)$.

In this case,

$$\begin{aligned} r &= r_\beta - r_\alpha, & s &= s_\beta - s_\alpha \\ q_1 &= q_{1\beta} - q_{1\alpha}, & q_2 &= q_{2\beta} - q_{2\alpha} \end{aligned}$$

Case 4: For some $\Delta' > 0$, $q_{2\beta} + s_\beta\Delta' = q_{1\alpha} + r_\alpha\Delta'$, and for all $\Delta < \Delta'$, $\alpha^u(\Delta) > \beta^l(\Delta)$.

In this case,

$$\begin{aligned} r &= s_\beta - r_\alpha, & s &= s_\beta - s_\alpha \\ q_1 &= q_{2\beta} - q_{1\alpha}, & q_2 &= q_{2\beta} - q_{2\alpha} \end{aligned}$$

If $\alpha^u(\Delta) \geq \beta^l(\Delta)$ for all $\Delta \geq 0$ then $r = s = 0$ and $q_1 = q_2 = 0$

Derivation: To prove that $\beta^{l'} = L(q_1, q_2, r, s)$ is a valid lower remaining service curve, we shall as before show that $L(q_1, q_2, r, s)(\Delta) \leq \sup_{0\leq u\leq\Delta}\{\beta^l(u) - \alpha^u(u)\}$ for all $\Delta \geq 0$.

Firstly, it may be noted that $\beta^l(\Delta)$ and $\alpha^u(\Delta)$ are convex and concave respectively. Therefore, $\beta^l(\Delta) - \alpha^u(\Delta)$ and $\sup_{0\leq u\leq\Delta}\{\beta^l(u) - \alpha^u(u)\}$ are convex. Here we have to consider four different cases.

Case 1 is when the last segment of $\beta^l(\Delta)$ intersects the last segment of $\alpha^u(\Delta)$, at say $\Delta = \Delta'$ (see Figure 77). Therefore, for all $\Delta < \Delta'$, $\beta^l(\Delta) < \alpha^u(\Delta)$.

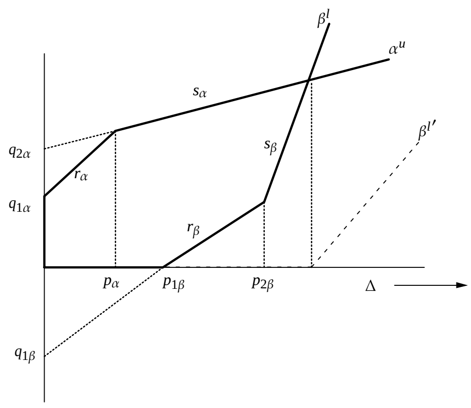


Fig. 77: Remaining Lower Service Curve (Case 1).

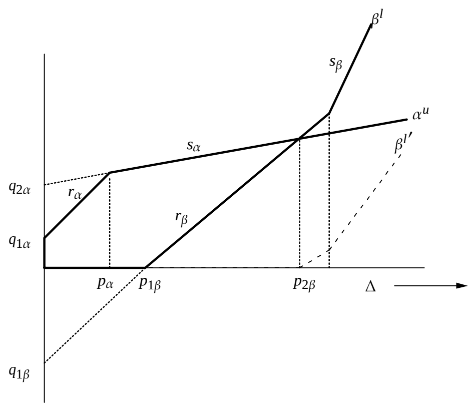


Fig. 78: Remaining Lower Service Curve (Case 2).

Hence, $\sup_{0\le u\le\Delta}\{\beta^l(u)-\alpha^u(u)\}=0$ for all $\Delta\le\Delta'$, and therefore $r=0$ and $q_1=0$. When $\Delta\to\infty$, $\sup_{0\le u\le\Delta}\{\beta^l(u)-\alpha^u(u)\}=\sup_{0\le u\le\Delta}\{q_{2\beta}+s_\beta u-q_{2\alpha}-s_\alpha u\}$. Therefore, we have $s=s_\beta-s_\alpha$ and $q_2=q_{2\beta}-q_{2\alpha}$.

Case 2 is when the middle segment of $\beta^l(\Delta)$ intersects the last segment of $\alpha^u(\Delta)$. If this intersection is at $\Delta=\Delta'$, then for all $\Delta<\Delta'$, $\beta^l(\Delta)<\alpha^u(\Delta)$ and $\sup_{0\le u\le\Delta}\{\beta^l(u)-\alpha^u(u)\}=0$ for all $\Delta\le\Delta'$. This case is shown in Figure 78. Clearly, $r=r_\beta-s_\alpha$, $s=s_\beta-s_\alpha$, $q_1=q_{1\beta}-q_{2\alpha}$ and $q_2=q_{2\beta}-q_{2\alpha}$.

Case 3 is when the middle segment of $\beta^l(\Delta)$ intersects the middle segment of $\alpha^u(\Delta)$ (see Figures 79 and 80). In this case, $\sup_{0\le u\le\Delta}\{\beta^l(u)-\alpha^u(u)\}$ is made up of four linear segments. But we approximate it using $L(q_1,q_2,r,s)(\Delta)$, which is made up of three segments. Figures 79 and 80 show two possible subcases: when $p_{2\beta}\ge p_\alpha$ and when $p_{2\beta}<p_\alpha$ respectively. If $\beta^l(\Delta)$ and $\alpha^u(\Delta)$ intersect at Δ', then the four segments that make up $\sup_{0\le u\le\Delta}\{\beta^l(u)-\alpha^u(u)\}$ span the intervals $\Delta\in[0,\Delta')$, $[\Delta',p_\alpha)$, $[p_\alpha,p_{2\beta})$, $[p_{2\beta},\infty)$ (as shown in Figure 79) and $\Delta\in[0,\Delta')$, $[\Delta',p_{2\beta})$, $[p_{2\beta},p_\alpha)$, $[p_\alpha,\infty)$

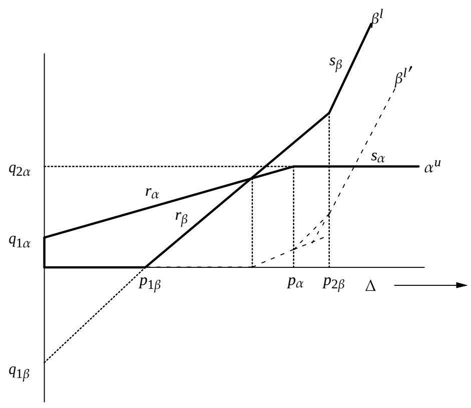


Fig. 79: Remaining Lower Service Curve (Case 3, $p_{2\beta} \geq p_\alpha$).

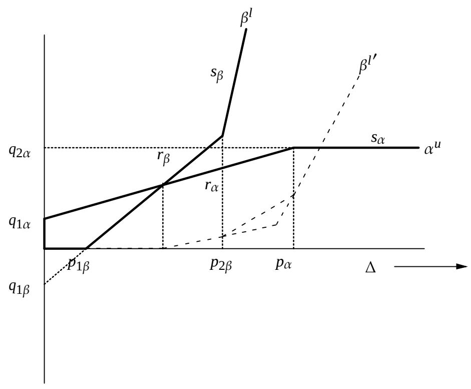


Fig. 80: Remaining Lower Service Curve (Case 3, $p_{2\beta} < p_\alpha$).

(as shown in Figure 80). To obtain $L(q_1, q_2, r, s)(\Delta)$, we neglect the segment of $\sup_{0 \leq u \leq \Delta}\{\beta^l(u) - \alpha^u(u)\}$ corresponding to the interval $[p_\alpha, p_{2\beta})$ in Figure 79 and the interval $[p_{2\beta}, p_\alpha)$ in Figure 80, and instead approximate this segment by the segments preceding and following it. It may be noted that $L(q_1, q_2, r, s)(\Delta)$ is a valid lower curve, since $L(q_1, q_2, r, s)(\Delta) \leq \sup_{0 \leq u \leq \Delta}\{\beta^l(u) - \alpha^u(u)\}$ for all $\Delta \geq 0$. Therefore, $r = r_\beta - r_\alpha$, $s = s_\beta - s_\alpha$, $q_1 = q_{1\beta} - q_{1\alpha}$, $q_2 = q_{2\beta} - q_{2\alpha}$.

Case 4 is when the last segment of $\beta^l(\Delta)$ intersects the middle segment of $\alpha^u(\Delta)$ (see Figure 81). It can be seen that $r = s_\beta - r_\alpha$, $q_1 = q_{2\beta} - q_{1\alpha}$, and as before, $s = s_\beta - s_\alpha$ and $q_2 = q_{2\beta} - q_{2\alpha}$.

Lastly, if $\beta^l(\Delta) \leq \alpha^u(\Delta)$ for all $\Delta \geq 0$, then $\sup_{0 \leq u \leq \Delta}\{\beta^l(u) - \alpha^u(u)\} \leq 0$ for all $\Delta \geq 0$. Hence, $r = s = 0$ and $q_1 = q_2 = 0$.

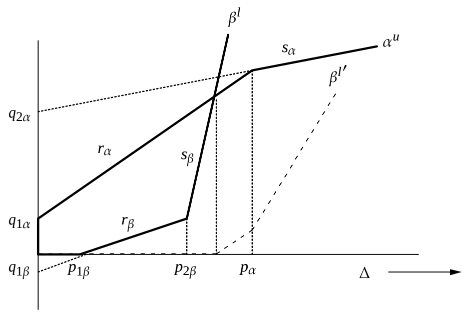


Fig. 81: Remaining Lower Service Curve (Case 4).

Prop. 6: **(Simple Upper Service Curve)** *Given the lower arrival and upper service curves $\alpha^l = L(q_{1\alpha}, q_{2\alpha}, r_\alpha, s_\alpha)$ and $\beta^u = U(q_{1\beta}, q_{2\beta}, r_\beta, s_\beta)$ respectively, then the remaining upper service curve $\beta^{u\prime} = U(q_1, q_2, r, s)$ is determined as follows:*

$$r = \begin{cases} 0, & if\, s_\beta < s_\alpha \\ r_\beta, & if s_\beta \geq s_\alpha \wedge p_\beta < p_{1\alpha} \\ r_\beta - r_\alpha, & if\, s_\beta \geq s_\alpha \wedge p_\beta \geq p_{1\alpha} \end{cases}$$

$$s = \begin{cases} 0, & if\, s_\beta < s_\alpha \\ s_\beta - s_\alpha, & if\, s_\beta \geq s_\alpha \end{cases}$$

$$q_1 = \begin{cases} 0, & if\, s_\beta < s_\alpha \\ q_{1\beta}, & if\, s_\beta \geq s_\alpha \wedge p_\beta < p_{1\alpha} \\ q_{1\beta} - q_{1\alpha}, & if\, s_\beta \geq s_\alpha \wedge p_\beta \geq p_{1\alpha} \end{cases}$$

$$q_2 = \begin{cases} 0, & if\, s_\beta < s_\alpha \\ q_{2\beta} - q_{2\alpha}, & if\, s_\beta \geq s_\alpha \end{cases}$$

Derivation: Because the remaining upper service curve $\beta^{u\prime}(\Delta)$ is defined as the smallest difference between the initial upper service curve $\beta^u(u)$ and the initial lower arrival curve of packets $\alpha^l(u)$ for all $u \geq \Delta$, it is obvious that $\beta^{u\prime}(\Delta) = 0$, if $s_\alpha > s_\beta$.

If $s_\alpha \leq s_\beta$, there are three possible cases, how the incoming curves α^l and β^u can look like. These different cases are shown in Figures 82–84. In general, if we calculate the resulting upper service curve $\beta^{u\prime}$ using formula , we obtain a curve with four linear pieces. To obtain only three linear pieces we will have to approximate this resulting curve with three linear segments. The four piece linear approximation is shown in the figures as small dotted lines, the approximation is given as bold solid lines.

The approximation in the first case is shown in Figure 82. In this case the burst rate of the resulting upper service curve $\beta^{u\prime}(\Delta)$ equals to the burst rate of the initial upper service curve β^u, because there is no load offered by the lower arrival curve α^l at all. Similarly the initial offset q_1 of the upper service curve remains the same. The long term rate s in this first case results to the difference between the long term rates of the initial curves, therefore $s = s_\beta - s_\alpha$. To compute q_2 we will have to solve the equation $q_2 + p_{2\alpha}(s_\beta - s_\alpha) = q_{2\beta} + s_\beta p_{2\alpha} - q_{2\alpha} - s_\alpha p_{2\alpha}$. This results to $q_2 = q_{2\beta} - q_{2\alpha}$.

In the second (Figure 83) and third (Figure 84) case the final remaining upper service curve is derived from the four piece curve in the same way. That's the reason why, in the formulas, we only distinguish between the case $p_\beta < p_{1\alpha}$ and the case where $p_{1\alpha} \leq p_\beta$. In the latter case, the burst rate is determined by calculating $r_\beta - r_\alpha$, this means, that the burst rate of the final approximation curve is determined by the second line segment

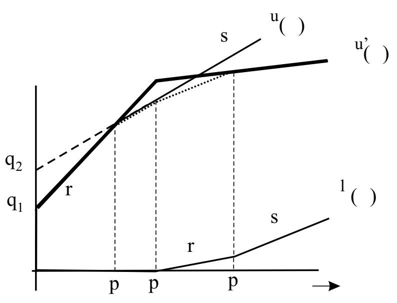


Fig. 82: Approximation of $\beta^{u\prime}$ in the case $p_\beta < p_{1\alpha}$.

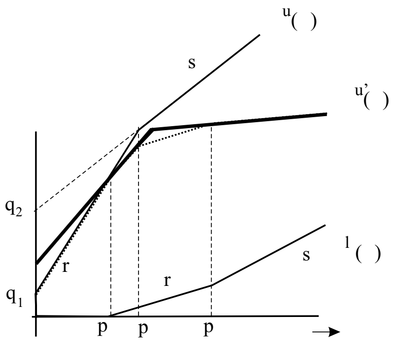


Fig. 83: Approximation of $\beta^{u\prime}$ in the case $p_{1\alpha} \leq p_\beta \leq p_{2\alpha}$.

of the intermediate (four segment) approximation curve. To compute the initial offset, we have to solve the equation $q_{1\beta} + r_\beta = q_1 + (r_\beta - r_\alpha)p_{1\alpha}$. With $r_\alpha p_{1\alpha} + q_{1\alpha} = 0$, we find $q_1 = q_{1\beta} - q_{1\alpha}$.

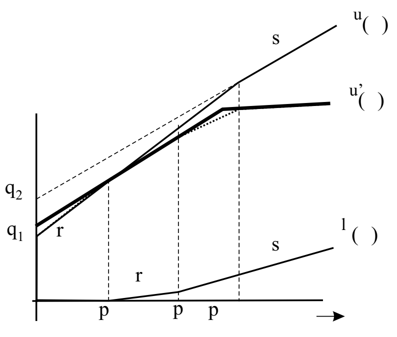


Fig. 84: Approximation of $\beta^{u\prime}$ in the case $p_\beta > p_{2\alpha}$.

Prop. 7: (Backlog and Delay) *Given the upper arrival and lower service curves $\alpha^u = U(q_{1\alpha}, q_{2\alpha}, r_\alpha, s_\alpha)$ and $\beta^l = L(q_{1\beta}, q_{2\beta}, r_\beta, s_\beta)$ respectively, then the maximum backlog B and the maximum delay D of a packet can be bounded as follows:*

$$B \leq \begin{cases} q_{2\alpha} - \frac{s_\alpha}{r_\beta} q_{1\beta} & \text{if } p_\alpha \leq p_{1\beta} \wedge s_\alpha < r_\beta \\ q_{2\alpha} - q_{1\beta} + (s_\alpha - r_\beta)\frac{q_{2\alpha}-q_{1\alpha}}{r_\alpha - s_\alpha} & \text{if } p_{1\beta} < p_\alpha < p_{2\beta} \wedge s_\alpha < r_\beta < r_\alpha \\ q_{2\alpha} - q_{2\beta} + (s_\alpha - s_\beta)\frac{q_{2\alpha}-q_{1\alpha}}{r_\alpha - s_\alpha} & \text{if } p_{2\beta} \leq p_\alpha \wedge s_\alpha \leq s_\beta < r_\alpha \\ q_{1\alpha} - q_{1\beta} + (r_\alpha - r_\beta)\frac{q_{2\beta}-q_{1\beta}}{r_\beta - s_\beta} & \text{if } p_{2\beta} \leq p_\alpha \wedge r_\beta < r_\alpha \leq s_\beta \\ q_{1\alpha} - \frac{r_\alpha}{r_\beta} q_{1\beta} & \text{if } p_{1\beta} < p_\alpha \wedge s_\alpha < r_\alpha \leq r_\beta < s_\beta \\ q_{2\alpha} - q_{1\beta} + (s_\alpha - r_\beta)\frac{q_{2\beta}-q_{1\beta}}{r_\beta - s_\beta} & \text{if } p_\alpha < p_{2\beta} \wedge r_\beta \leq s_\alpha \leq s_\beta \\ \infty & \text{if } s_\beta < s_\alpha \end{cases}$$

$$D \leq \begin{cases} \infty & \text{if } s_\beta < s_\alpha \\ \frac{\frac{q_{2\alpha}-q_{1\alpha}}{r_\alpha - s_\alpha} r_\alpha + q_{1\alpha} - q_{2\beta}}{s_\beta} - \frac{q_{2\alpha}-q_{1\alpha}}{r_\alpha - s_\alpha} & \text{if } s_\alpha \leq s_\beta \leq r_\alpha \\ \frac{q_{1\alpha}-q_{2\beta}}{s_\beta} & \text{if } r_\beta < r_\alpha < s_\beta \wedge q_{2\beta} + p_{2\beta}s_\beta \leq q_{1\alpha} \text{ or } r_\alpha \leq r_\beta \\ -\frac{\frac{q_{2\beta}-q_{1\beta}}{r_\beta - s_\beta} r_\beta + q_{1\beta} - q_{1\alpha}}{r_\alpha} + \frac{q_{2\beta}-q_{1\beta}}{r_\beta - s_\beta} & \text{if } r_\beta < r_\alpha < s_\beta \wedge q_{1\alpha} < q_{2\beta} + p_{2\beta}s_\beta \leq q_{1\alpha} + p_\alpha r_\alpha \\ \frac{\frac{q_{2\alpha}-q_{1\alpha}}{r_\alpha - s_\alpha} r_\alpha + q_{1\alpha} - q_{1\beta}}{r_\beta} - \frac{q_{2\alpha}-q_{1\alpha}}{r_\alpha - s_\alpha} & \text{if } s_\alpha \leq r_\beta < r_\alpha < s_\beta \wedge q_{1\alpha} + p_\alpha r_\alpha < q_{2\beta} + p_{2\beta}s_\beta \\ -\frac{\frac{q_{2\beta}-q_{1\beta}}{r_\beta - s_\beta} r_\beta + q_{1\beta} - q_{2\alpha}}{s_\alpha} + \frac{q_{2\beta}-q_{1\beta}}{r_\beta - s_\beta} & \text{if } r_\beta < s_\alpha < r_\alpha < s_\beta \wedge q_{1\alpha} + p_\alpha r_\alpha < q_{2\beta} + p_{2\beta}s_\beta \end{cases}$$

Derivation: The derivation of this proposition can be found by inspecting the graphs for $\alpha^u = U(q_{1\alpha}, q_{2\alpha}, r_\alpha, s_\alpha)$ and $\beta^l = L(q_{1\beta}, q_{2\beta}, r_\beta, s_\beta)$ and calculating the maximal horizontal and vertical distances.

Bibliography

[ABD+04] U. Anliker, J. Beutel, M. Dyer, R. Enzler, P. Lukowicz, L. Thiele, and G. Tröster. A systematic approach to the design of distributed wearable systems. *IEEE Transactions on Computers*, 53(8):1017–1033, Aug 2004.

[Aga92] A. Agarwal. Performance tradeoffs in multithreaded processors. *IEEE Transactions on Parallel and Distributed Systems*, 3(5):525–539, September 1992.

[ALM98] G. Anastasi, L. Lenzini, and E. Mingozzi. Stability and performance analysis of hiperlan. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '98)*, March 1998.

[Alp92] *Alpha Architecture Reference Manual, Digital Press*, 1992.

[AMB] ARM AMBA 2.0. `http://www.arm.com/products/../solutions/AMBAOverview.html`.

[APS04] Giovanni Agosta, Gianluca Palermo, and Cristina Silvano. Multi-objective co-exploration of source code transformations and design space architectures for low-power embedded systems. In Hisham Haddad, Andrea Omicini, Roger L. Wainwright, and Lorie M. Liebrock, editors, *Proceedings of the 2004 ACM symposium on Applied computing*, pages 891–896, New York, NY, USA, 2004. ACM Press.

[ARS98] Santosh Abraham, B. Ramakrishna Rau, and Robert Schreiber. Fast design space exploration through validity and quality filtering of subsystem designs. Technical Report HPL-2000-98, HP Labs Technical Reports, 1998.

[BA97] Doug Burger and Todd M. Austin. The simplescalar tool set, version 2.0. *SIGARCH Comput. Archit. News*, 25(3):13–25, 1997.

[BBB01] Davide Bruni, Alessandro Bogliolo, and Luca Benini. Statistical design space exploration for application-specific unit synthesis. In *Proceedings of the 38th Design Automation Conference*, pages 641–646, New York, NY, USA, 2001. ACM Press.

[BBTZ01] Neal K. Bambha, Shuvra Bhattacharyya, Juergen Teich, and Eckart Zitzler. Hybrid search strategies for dynamic voltage scaling in embedded multiprocessors. In Jan Madsen, Jörg Henkel, and Xiaobo Sharon Hu, editors, *Proceedings of the Ninth International Symposium on Hardware/Software Codesign*, pages 243–248, New York, NY, USA, April 2001. ACM Press.

[BC98] Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 151–160. ACM Press, 1998.

[BFM97] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. IOP Publishing and Oxford University Press, Bristol, UK, 1997.

[BLTZ03] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA — a platform and programming language independent interface for search algorithms. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization (EMO 2003)*, Lecture Notes in Computer Science, pages 494–508, Berlin, 2003. Springer.

[Blu05] Philipp Blum. *Guaranteed Time Synchronization in Wireless and Ad-Hoc Networks*. PhD thesis, Swiss Federal Institute of Technology (ETH), 2005.

[Boe88] Barry W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.

[BPN$^+$04] Alex Bobrek, Joshua J. Pieper, Jeffrey E. Nelson, JoAnn M. Paul, and Donald E. Thomas. Modeling shared resource contention using a hybrid simulation/analytical approach. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition Volume II (DATE'04)*, pages 1144–1149. IEEE Computer Society, 2004.

[BT01] J.Y. Le Boudec and P. Thiran. *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*. LNCS 2050, Springer Verlag, 2001.

[BT03] Peter A. N. Bosman and Dirk Thierens. The balance between proximity and diversity in multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(2):174–188, 2003.

[BTT98] Tobias Blickle, Jürgen Teich, and Lothar Thiele. System-level synthesis using evolutionary algorithms. *Journal on Design Automation for Embedded Systems*, 3(8):23–58, January 1998.

[BZZ04] A. Bona, V. Zaccaria, and R. Zafalon. System level power modeling and simulation of high-end industrial network-on-chip. In *Design and Test in Europe Conference (DATE)*, pages 318–323, 2004.

[CB02] P. Crowley and J-L. Baer. A modeling framework for network processor systems. In *Proc. 1st Workshop on Network Processors, held in conjunction with the 8th International Symposium on High-Performance Computer Architecture*, February 2002. An enhanced version of this paper appeared in the book "Network Processor Design: Issues and Practices, Volume 1", Morgan Kaufmann Publishers, October 2002.

[CdGS98] Francky Catthoor, Eddy de Greef, and Sven Suytack. *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[CG03] Lukai Cai and Daniel Gajski. Transaction level modeling: an overview. In *CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 19–24, New York, NY, USA, 2003. ACM Press.

[CJ98] P. Czyzak and A. Jaszkiewicz. Pareto-simulated annealing – a metaheuristic technique for multi-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47, January 1998.

[CKT03a] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. 6th Design, Automation and*

Test in Europe (DATE), pages 190–195, Munich, Germany, March 2003.

[CKT$^+$03b] S. Chakraborty, S. Künzli, L. Thiele, A. Herkersdorf, and P. Sagmeister. Performance evaluation of network processor architectures: Combining simulation with analytical estimation. *Computer Networks*, 41(5):641–665, April 2003.

[CPL] CPLEX optimizer. `http://www.cplex.com/`.

[CSH00] C. Chantrapornchai, E.H.-M. Sha, and X. S. Hu. Efficient acceptable design exploration based on module utility selection. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 19(1):19–29, 2000.

[CVL02] C. A. Coello Coello, D. A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, 2002.

[DA95] Kalyanmoy Deb and Ram Bhushan Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9:115–148, 1995.

[DAPM00] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In M. Schoenauer et al., editors, *Parallel Problem Solving from Nature (PPSN VI)*, Lecture Notes in Computer Science Vol. 1917, pages 849–858. Springer, 2000.

[DD02] Rolf Drechsler and Nicole Drechsler. *Evolutionary Algorithms for Embedded System Design*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[Deb01] K. Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley, Chichester, UK, 2001.

[DJ98] R. P. Dick and N. K. Jha. MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Co-synthesis of Hierarchical Heterogeneous Distributed Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):920–935, 1998.

[DPAM02] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm : NSGA-II. *IEEE Trans. on Evolutionary Computation*, 6(2):182–197, April 2002.

[DTLZ02] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. In *Congress on Evolutionary Computation (CEC)*, pages 825–830. IEEE Press, 2002.

[DTLZ05] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable test problems for evolutionary multi-objective optimization. In A. Abraham, L. C. Jain, and R. Goldberg, editors, *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. Springer, 2005.

[Ern98] Rolf Ernst. Codesign of embedded systems: Status and trends. *IEEE Design & Test*, 15(2):45–54, 1998.

[Ern03] Rolf Ernst. Putting it all together. *Queue*, 1(2):50, 2003.

[ETZ00] Michael Eisenring, Lothar Thiele, and Eckart Zitzler. Handling conflicting criteria in embedded system design. *IEEE Design and Test of Computers*, 17(2):51–59, April 2000.

[FF98] C. M. Fonseca and Peter J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms—part ii: Application example. *IEEE Transactions on Systems, Man, and Cybernetics*, 28(1):38–47, 1998.

[FW02] M.A. Franklin and T. Wolf. A network processor performance and design model with benchmark parameterization. In *Proc. 1st Workshop on Network Processors, held in conjunction with the 8th International Symposium on High-Performance Computer Architecture*, Cambridge, Massachusetts, February 2002. An enhanced version of this paper appears in the book “Network Processor Design: Issues and Practices, Volume 1”, Morgan Kaufmann Publishers, October 2002.

[GG03] Arijit Ghosh and Tony Givargis. Analytical design space exploration of caches for embedded systems. In Norbert Wehn and Diederik Verkest, editors, *Design, Automation and Test in Europe Conference and Exhibition (DATE 03)*, pages 650–655, Los Alamitos, CA, USA, March 2003. IEEE Press.

[GLMS02] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, Boston, MA, USA, May 2002.

[GVH02] Tony Givargis, Frank Vahid, and Joerg Henkel. System-level exploration for Pareto-optimal configurations in parameterized system-on-a-chip. *IEEE Trans. Very Large Scale Integr. Syst.*, 10(4):416–422, 2002.

[GVNG98] Daniel D. Gajski, Frank Vahid, Sanjiv Narayan, and Jie Gong. System-level exploration with SpecSyn. In *Proceedings of the 35th Design Automation Conference*, pages 812–817, New York, NY, USA, 1998. ACM press.

[GZD+00] Daniel D. Gajski, Jianwen Zhu, Rainer Dömer, Andreas Gerstlauer, and Shuqing Zhao. *SPEC C:Specification Language and Design Methodology*. Kluwer Academic Publishers, 2000.

[HE05] Arne Hamann and Rolf Ernst. TDMA time slot and turn optimization with evolutionary search techniques. In *Proc. of Design, Automation and Test in Europe (DATE '05)*, March 2005.

[HHBS99] G.J. Hekstra, G.D. La Hei, P. Bingley, and F.W. Sijstermans. TriMedia CPU64 Design Space Exploration. In Andreas Kuehlmann and Craig Chase, editors, *1999 IEEE International Conference on Computer Design*, pages 593–598, Los Alamitos, CA, USA, October 1999. IEEE Press.

[HHJ+05] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis – the SymTA/S approach. *IEE Proceedings - Computers and Digital Techniques*, 152(02), March 2005.

[HJ98] Michael P. Hansen and Andrzej Jaszkiewicz. Evaluating the quality of approximations of the non-dominated set. Technical report, Institute of Mathematical Modeling, Technical University of Denmark, 1998. IMM Technical Report IMM-REP-1998-7.

[IBMa] Blue Logic technology, IBM. `http://www.chips.ibm.com/bluelogic/`.

[IBMb] Coreconnect bus architecture, IBM. `http://www.chips.ibm.com/products/coreconnect/`.

[Jas03] A. Jaszkiewicz. Do multiple-objective metaheuristics deliver on their promises? a computational experiment on the set-covering problem. *IEEE Transactions on Evolutionary Computation*, 7(2):133–143, 2003.

[KBTZ04] Simon Künzli, Stefan Bleuler, Lothar Thiele, and Eckart Zitzler. A computer engineering benchmark application for multiobjective optimizers. In Carlos Coello Coello and Gary B. Lamont, editors, *Application of Multi-Objective Evolutionary Algorithms*, pages 269–294. World Scientific, 2004.

[KC00] J. D. Knowles and D. W. Corne. Approximating the non-dominated front using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[KC02] J. Knowles and D. Corne. On metrics for comparing non-dominated sets. In *Congress on Evolutionary Computation (CEC 2002)*, pages 711–716, Piscataway, NJ, 2002. IEEE Press.

[KC03] J. D. Knowles and D. W. Corne. Instance generators and test suites for the multiobjective quadratic assignment problem. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization (EMO 2003)*, Lecture Notes in Computer Science, pages 295–310, Berlin, 2003. Springer.

[KM98] A. Kalavade and P. Moghé. A tool for performance estimation of networked embedded end-systems. In *35th DAC*, 1998.

[KMC$^+$00] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M.F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[KMN$^+$00] K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli. System level design: Orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design*, 19(12), December 2000.

[Kno02] J. D. Knowles. *Local-Search and Hybrid Evolutionary Algorithms for Pareto Optimization*. PhD thesis, University of Reading, 2002.

[KPBT06] S. Künzli, F. Poletti, L. Benini, and L. Thiele. Combining simulation and formal methods for system-level performance analysis. In *Proc. Design, Automation and Test in Europe (DATE)*, March 2006.

[KT06] Simon Künzli and Lothar Thiele. Generating event traces based on arrival curves. In *Proc. of 13th GI/ITG Conference* *on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB)*, March 2006.

[KTZ05a] J. Knowles, L. Thiele, and E. Zitzler. A tutorial on the performance assessment of stochastic multiobjective optimizers. 214, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, July 2005.

[KTZ05b] S. Künzli, L. Thiele, and E. Zitzler. Modular design space exploration framework for embedded systems. *IEE Proceedings - Computers and Digital Techniques*, 152(02):183–192, March 2005.

[KTZ06] Simon Künzli, Lothar Thiele, and Eckart Zitzler. Multi-criteria decision making in embedded system design. In Bashir Al-Hashimi, editor, *System On Chip: Next Generation Electronics*, pages 3–28. IEE Press, January 2006.

[Kur91] F. Kursawe. A variant of evolution strategies for vector optimization. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature (PPSN)*, pages 193–197. Springer, 1991.

[LAB$^+$04] Mirko Loghi, Federico Angiolini, Davide Bertozzi, Luca Benini, and Roberto Zafalon. Analyzing on-chip communication in a MPSoC environment. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition Volume II (DATE'04)*, pages 752–757. IEEE Computer Society, 2004.

[LNA] National Laboratory for Applied Network Research (NLANR), Traces collected in June 2000, `http://pma.nlanr.net/pma/`.

[LPB04] M. Loghi, M. Poncino, and L. Benini. Cycle-accurate power analysis for multiprocessor systems-on-a-chip. In *GLSVLSI04: Great Lake Symposium on VLSI*, pages 401–406, 2004.

[LRD04] K. Lahiri, A. Raghunathan, and S. Dey. Design space exploration for optimizing on-chip communication architectures. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 23(6):952–961, June 2004.

[LTDZ02] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary Computation*, 10(3):263–282, 2002.

[LTWW93] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of Ethernet traffic. In Deepinder P. Sidhu, editor, *ACM SIGCOMM*, pages 183–193, San Francisco, California, 1993.

[LTZ04] M. Laumanns, L. Thiele, and E. Zitzler. Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions. *IEEE Transactions on Evolutionary Computation*, 8(2):170–182, April 2004.

[LZT01] Marco Laumanns, Eckart Zitzler, and Lothar Thiele. On the effects of archiving, elitism, and density based selection in evolutionary multi-objective optimization. In *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, pages 181–196. Springer-Verlag, 2001.

[Mar03] Peter Marwedel. *Embedded System Design*. Kluwer Academic Publishers, 2003.

[MAS$^+$05] Shankar Mahadevan, Federico Angiolini, Michael Storgaard, Rasmus Grondahl Olsen, Jens Sparso, and Jan Madsen. A network traffic generator model for fast network-on-chip simulation. In *DATE '05: Proceedings of the Design, Automation and Test in Europe (DATE'05) Volume 2*, pages 780–785. IEEE Computer Society, 2005.

[Max05] Alexander Maxiaguine. *Modeling Multimedia Workloads for embedded system design*. PhD thesis, Swiss Federal Institute of Technology (ETH), 2005.

[Mie99] K. Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer, Boston, 1999.

[MKT04] A. Maxiaguine, S. Künzli, and L. Thiele. Workload characterization model for tasks with variable execution demand. In *Proc. 7th Design, Automation and Test in Europe (DATE)*, pages 1040–1045, Paris, France, February 2004.

[MSB$^+$02] Bart Mesman, Ben Spaanenburg, Ed Brinksma, Ed Deprettere, Eric Verhulst, Floris Timmer, Hans van Gageldonk, Ludwig D.J. Eggermont, René van Leuken, Thijs Krol, and

Wim Hendriksen. Embedded systems roadmap 2002. Technical report, STW Technologiesichting, Utrecht, Netherlands, March 2002.

[NSK02] Sandeep Neam, Janos Sztipanovits, and Gabor Karsai. Design-space construction and exploration in platform-based design. Technical Report ISIS-02-301, Vanderbilt University, Institute for Software Integrated Systems, 2002.

[PAB+05] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa. The design and implementation of a first-generation CELL processor. In *Proceedings of ISSCC 2005*, February 2005.

[PEP05] Paul Pop, Petru Eles, and Zebo Peng. Analysis and optimization of heterogeneous multiprocessor soc. *IEE Proceedings - Computers and Digital Techniques*, 152(02), March 2005.

[PG02] Maurizio Palesi and Tony Givargis. Multi-objective design space exploration using genetic algorithms. In Jörg Henkel, Xiaobo Sharon Hu, Rajesh Gupta, and Sri Parameswaran, editors, *Proceedings of the 10th international symposium on Hardware/software codesign*, pages 67–72, New York, NY, USA, 2002. ACM Press.

[PLvdW+01] A.D. Pimentel, P. Lieverse, P. van der Wolf, L.O. Hertzberger, and E.F. Deprettere. Exploring embedded-systems architectures with artemis. *IEEE Computer*, 34(11):57–63, November 2001.

[Pow] IBM PowerNP NPe405 Embedded Processors. `http://www-3.ibm.com/chips/techlib/techlib.nsf/products/PowerNP_NPe405_Embedded_Processors`.

[PSZ03] G. Palermo, C. Silvano, and V. Zaccaria. A flexible framework for fast multi-objective design space exploration of embedded systems. In Jorge Juan-Chico and Enrico Macii, editors, *PATMOS 2003- International Workshop on Power and Timing Modeling*, volume 2799 of *Lecture Notes in Computer Science*, pages 249–258, Berlin, Germany, 2003. Springer.

[PSZ05] Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. Multi-objective design space exploration of embedded systems. *Journal of Embedded Computing*, 1(3), 2005.

[RCR04] S. Rajagopal, J. Cavallaro, and S. Rixner. Design space exploration for real-time embedded stream processors. *IEEE Micro*, 24(4):54–66, July/August 2004.

[RE02] K. Richter and R. Ernst. Event model interfaces for heterogeneous system analysis. In *Proc. 5th Design, Automation and Test in Europe (DATE)*, page 506. IEEE Computer Society, March 2002.

[RJE03] Kai Richter, Marek Jersak, and Rolf Ernst. A formal approach to MpSoC performance verification. *Computer*, 36(4):60–67, 2003.

[RZJE02] Kai Richter, Dirk Ziegenbein, Marek Jersak, and Rolf Ernst. Model composition for scheduling analysis in platform design. In *Proceedings 39th Design Automation Conference (DAC)*, June 2002.

[SC99] Wen-Tsong Shiue and Chaitali Chakrabarti. Memory exploration for low power, embedded systems. In *Proceedings of the 36th ACM/IEEE Design Automation Conference*, pages 140–145, New York, NY, USA, 1999. ACM Press.

[Sch85] J. D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In John J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 93–100, 1985.

[SCK04] R. Szymanek, F. Catthoor, and K. Kuchcinski. Time-energy design space exploration for multi-layer memory architectures. In Georges Gielen and Joan Figueras, editors, *Proc. of 7th ACM/IEEE Design, Automation and Test in Europe Conference*, page 10318, New York, NY, USA, 2004. ACM Press.

[Sea] Seamless Hardware/Software Co-Verification, Mentor Graphics. `http://www.mentor.com/seamless/`.

[SJ01] Premkishore Shivakumar and Norman P. Jouppi. Cacti 3.0:an integrated cache timing, power and area model. Technical Report WRL Research Report 2001/2, Compaq, Western Research Laboratory, August 2001.

[SPE] Standard performance evaluation corporation. http://www.spec.org/.

[SW97] S. Shenker and J. Wroclawski. General characterization parameters for integrated service network elements. RFC 2215, IETF, September 1997.

[SYS] SystemC homepage. http://www.systemc.org.

[TCGK02a] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. Design space exploration of network processor architectures. In Patrick Crowley, Mark A. Franklin, Haldun Hadimioglu, and Peter Z. Onufryk, editors, *Network Processor Design: Issues and Practices*, volume 1, chapter 4, pages 55–90. Morgan Kaufmann Publishers, San Francisco, CA, USA, October 2002. A preliminary version of this paper appeared in the Proc. 1st Workshop on Network Processors, held in conjunction with the 8th International Symposium on High-Performance Computer Architecture, Cambridge, Massachusetts, 2002.

[TCGK02b] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *Proc. 39th Design Automation Conference (DAC)*, pages 880–885, New York, NY, USA, June 2002. ACM Press.

[Thi03] D. Thierens. Convergence time analysis for the multi-objective counting ones problem. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization (EMO 2003)*, Lecture Notes in Computer Science, pages 355–364, Berlin, 2003. Springer.

[TW05] Lothar Thiele and Ernesto Wandeler. Performance analysis of distributed embedded systems. In Richard Zuwarski, editor, *Embedded Systems Handbook*, chapter 15, pages 15–1–15–18. CRC Press, Boca Raton, FL, USA, 2005.

[VCC] The Cadence Virtual Component Co-design (VCC). http://www.cadence.com/products/vcc.html.

[VG01] Frank Vahid and Tony Givargis. *Embedded System Design: A Unified Hardware/Software Introduction*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[WF00] T. Wolf and M. Franklin. CommBench - A telecommunications benchmark for network processors. In *Proc. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 154–162, Austin, Texas, 2000.

[WMT04] Ernesto Wandeler, Alexander Maxiaguine, and Lothar Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)*, page 450. IEEE Computer Society, 2004.

[WMY$^+$01] C. Wong, P. Marchal, P. Yang, A. Prayati, F. Catthoor, R. Lauwereins, D. Verkest, and H.D. Man. Task concurrency management methodology to schedule the mpeg4 im1 player on a highly parallel processor platform. In *CODES '01: Proceedings of the ninth international symposium on Hardware/software codesign*, pages 170–177, New York, NY, USA, 2001. ACM Press.

[Wol01] Wayne Wolf. *Computers as components: principles of embedded computing system design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[Wor01] F. Worm. A performance evaluation of memory organizations in the context of core based network processor designs. Master's thesis, Institut Eurécom, Sophia-Antipolis, France, This work was done at IBM Research Laboratory Zürich, 2001.

[ZDT00] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[ZK04] E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *Proc. 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, volume 3242 of *Lecture Notes in Computer Science*, Berlin, Germany, September 2004. Springer.

[ZLT02] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In K.C. Giannakoglou et al., editors, *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001)*, pages

95–100. International Center for Numerical Methods in Engineering (CIMNE), 2002.

[ZSXS03] Qingfeng Zhuge, Zili Shao, Bin Xiao, and Edwin H.-M. Sha. Design space minimization with timing and code size optimization for embedded DSP. In Rajesh Gupta, Yukihiro Nakamura, Alex Orailoglu, and Pai H. Chou, editors, *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 144–149, New York, NY, USA, 2003. ACM Press.

[ZT99] Eckart Zitzler and Lothar Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, November 1999.

[ZTB00a] E. Zitzler, J. Teich, and S. S. Bhattacharyya. Multidimensional exploration of software implementations for DSP algorithms. *Journal of VLSI Signal Processing*, 24(1):83–98, February 2000.

[ZTB00b] Eckart Zitzler, Juergen Teich, and Shuvra Bhattacharyya. Evolutionary algorithms for the synthesis of embedded software. *IEEE Transactions on VLSI Systems*, 8(4):452–456, April 2000.

[ZTL$^+$03] E. Zitzler, L. Thiele, M. Laumanns, C. M. Foneseca, and V. Grunert da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.