TECHNISCHE UNIVERSITÄT WIEN
VIENNA UNIVERSITY OF TECHNOLOGY

DISSERTATION

Heuristic Optimisation Methods for System Partitioning in HW/SW Co-Design

Conducted for the purpose of receiving the academic title
'Doktor der technischen Wissenschaften'

Submitted at
Vienna University of Technology
Faculty of Electrical Engineering and Information Technology

by

Dipl.-Ing. Bastian Knerr
Talgasse 8/13, 1150 Vienna, Austria
born in Püttlingen, Germany, September 3, 1976
Matriculation number: 0327662

Vienna, July 2008 .....

Advisor

Univ.Prof. Dipl.-Ing. Dr.techn. Markus Rupp
Technische Universität Wien
Institut für Nachrichtentechnik und Hochfrequenztechnik

Examiner

Univ.Prof. Dr.habil. Christoph Grimm
Technische Universität Wien
Institut für Computertechnik

ABSTRACT

Nowadays, the design of embedded systems is confronted with the combination of complex signal processing algorithms on the one hand and a variety of computational intensive multimedia applications on the other hand, while time to product launch has been extremely reduced. Especially in the wireless domain those challenges are stacked with tough requirements on power consumption and chip size. Unfortunately, design productivity did not undergo a similar progression and therefore fails to cope with the heterogeneity of modern hardware architectures. Until now, electronic design automation do not provide for complete coverage of the design flow. In particular crucial design tasks as high level characterisation of algorithms, floating-point to fixed-point conversion, automated hardware/software partitioning, and automated virtual prototyping are not sufficiently supported or completely absent. In recent years a consistent design framework named *Open Tool Integration Environment* (OTIE) has been established to address the most crucial shortcomings of the wide spread design problems in this field. As integral part of the OTIE framework powerful tool chains exist that support high level estimation techniques for algorithm characteristics, static code analysis, automatic generation of virtual prototypes, floating-point to fixed-point conversion, and so forth. A very substantial ingredient of OTIE was missing until now: a rich library for architecture modelling of embedded system and algorithms for their precise partitioning and scheduling. Therefore, this thesis examines the research field of *system partitioning* of embedded systems in the wireless design domain.

This field started to find strong advertence of scientists about fifteen years ago. Since a multitude of formulations for the partitioning problem exist, the same multitude could be found in the number of strategies that address this problem. Their feasibility is highly dependent on the platform abstraction and the degree of realism that it features. This thesis identifies the most mature and powerful approaches for system partitioning and to some degree task scheduling in order to integrate them into the OTIE framework. The contribution of this work involves a detailed platform abstraction that combines a high degree of realism with the flexibility to compose arbitrary multi-core multi-bus structures and the theoretical underpinning of the system partitioning in wireless embedded system design as combinatorial optimisation problem. Furthermore, a thorough analysis of the properties of typical system graphs is undertaken. Eventually, the implementation and improvement of the most popular strategies, and the introduction of entirely new algorithms for the system partitioning and scheduling problem is accomplished.

ZUSAMMENFASSUNG

Der Entwurf von eingebetteten Systemen ist heutzutage konfrontiert mit einer Kombination aus hochkomplexen Signalverarbeitungsalgorithmen und einer Vielzahl von rechenintensiven multimedialen Anwendungen. Erschwerend kommt hinzu, dass die Entwicklungszeit bis zum fertigen Produkt dramatisch verkürzt wurde. Besonders innerhalb der mobilen Sparte, die Mobiltelefone, PDAs, und Kameras umfasst, werden diese grundsätzlichen Widrigkeiten noch erschwert durch beträchtliche Anforderungen bezüglich Leistungsaufnahme und Baugröße. Leider konnte die Entwurfsproduktivität nicht mit den ansteigenden Anforderungen Schritt halten, und kämpft bis heute mit der Heterogenität moderner Hardwarearchitektur. Werkzeuge für die Entwurfsautomatisierung offenbaren breite Lücken in ihrer Abdeckung der Entwurfsabfolge, insbesondere wurden bisher folgende Aufgaben nicht zufriedenstellend gelöst: Algorithmencharakterisierung auf höchster Abstraktionsebene, Konvertierung von Fließkomma- zu Fixpunktdarstellung, Systempartitionierung, und Virtual Prototyping. In den letzten Jahren wurde im Christian Doppler Labor für Designmethodik von Signalverarbeitungsalgorithmen eine Entwurfsumgebung, OTIE, entwickelt, die in konsistenter Weise die kritischsten Mängel des Entwurfsprozesses in diesem Bereich zu beheben versucht. Bis auf eines wurden die zuvor genannten Aufgaben mit OTIE in bemerkenswerter Weise gelöst. Der fehlende Entwurfsschritt Systempartitionierung vereint mit einer flexiblen Architekturmodellierung stellt das Thema dieser Dissertation dar.

Systempartitionierung ist ein Forschungsgegenstand, der in den letzten 15 Jahren beträchtliche Aufmerksamkeit von Forschungsgruppen im elektronischen Systementwurf erhielt. Aus diesem Grund existiert eine ebenso große Anzahl spezifischer Problemformulierungen wie jeweiliger Lösungsstrategien. Ihre Anwendbarkeit variiert stark mit dem gewählten Abstraktionsgrad des Plattformmodells und dessen Wirklichkeitstreue. In dieser Arbeit werden die tauglichsten Ansätze für die Partitionierung von Prozessgraphen und in kleinerem Ausmaß jene für deren Ablaufplanung identifiziert, um diese dann in OTIE zu integrieren. Ein detailliertes Architekturmodell wird vorgestellt, das außergewöhnliche Wirklichkeitstreue mit großer Flexibilität vereint. Mit diesem ist es nun möglich beliebige heterogene Plattformstrukturen zu modellieren, in denen z.B. mehrere Prozessoren, FPGAs, und ASICs mittels mehrerer Busse oder anderer Datenkanäle verbunden werden. Basierend auf diesem Fundament wird das Partitionierungsproblem analysiert und als kombinatorische Mehrzieloptimierung definiert. Weitergehend werden jene Graphen, die für eingebettete Systeme typisch sind, analysiert und deren Eigenschaften herausgearbeitet. Mit Hilfe der erlangten Erkenntnisse werden in dieser Arbeit neue spezialisierte Algorithmen für Partitionierung und Ablaufplanung entwickelt und bestehende Konzepte und Techniken verbessert.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor Professor Markus Rupp for his persistent encouragement, support, and understanding of my research. I also would like to thank Professor Christoph Grimm for accepting to act as the examiner and, of course, for his valuable comments that greatly improved the quality of this thesis.

Whenever you start thinking about doing research, writing theses, or discuss decent music and the passage of being, you will tremendously benefit from sharing your office with Dr Martin Holzer. Great friend, great colleague!

This work has been funded by the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

On average every human being living in the industrialised world is almost permanently interacting with electronic systems and - usually - totally unaware of this fact. The last two decades of the past millennium witnessed an irresistible flood of electronics covering every aspect of modern life. Devices with a high *visibility* for the end user like notebooks, mobile phones, and PDAs mark only a small fraction of those electronic equipment the average person comes in touch with. Their overwhelming portion is represented by special purpose processors being embedded into larger devices, hence unifying omnipresence with invisibility. For example, the modern household is pervaded by processors controlling the dishwasher, the washing machine, the toaster, the vacuum cleaner, television, radio, etc. The proliferation of microelectronics in general, and embedded systems in particular, has percolated not only the personal but all commercial and industrial sectors, including logistics, communications, energy, transportation, security, mass media and others. As a result a multitude of design obstacles for embedded systems especially in the wireless domain popped up, persistently deeming their timely production an ordeal. The notion of the ubiquity of these devices shall facilitate the reader to comprehend the relevance of the design hurdles manufacturers usually encounter in the design process of embedded systems. This thesis deals with one of these obstacles, the system partitioning problem, and discusses how it can be overcome.

1.1 Motivation

When scrutinising the evolution of modern electronic systems, it becomes apparent, that not only their circulation but also their complexity has undergone a tremendous growth and is still not losing steam. The next generation of mobile devices for 3G UMTS systems is expected to be based on processors containing more than 40 million transistors [55]. Hence, during a relatively short period of time of about 10 years, a staggering increase in complexity of more than six orders of magnitude has taken place [130].
In comparison to this extremely fast-paced growth in algorithmic complexity, the concurrent increase in the complexity of silicon integrated circuits proceeds according to the well-known Moore's Law [109], famously predicting the doubling of the number of transistors integrated onto a single integrated circuit every 18 months. Hence, it can be concluded that the

growth in silicon complexity lags behind the extreme growth in the algorithmic complexity of wireless communication systems. This is also known as the *algorithmic complexity gap* depicted in Figure 1.1 on the left. The regularly published International Technology Roadmap

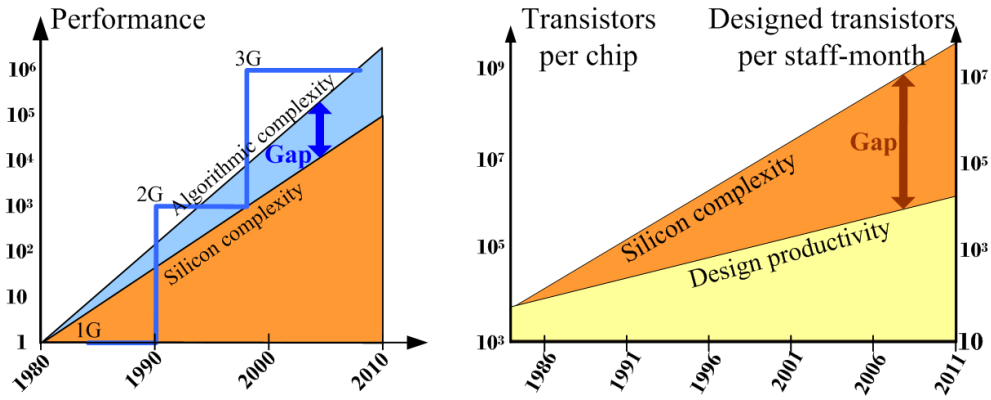


Fig. 1.1: Algorithmic complexity gap and design productivity gap.

for Semiconductors [74, 75] reported a growth in design productivity, expressed in terms of designed transistors per staff-month, of approximately 21% compounded annual growth rate, which lags behind the growth in silicon complexity. This is known as the *design gap* or *productivity gap* depicted in Figure 1.1 on the right. Thus, the abilities of underlying silicon platforms on which wireless communication systems are built have to be exploited with increasing efficiency, i.e. more functionality has to be gained from each individual transistor. In other words, the *quality* of the design process, i.e. effective functionality per unit of raw silicon achieved through both hardware and software parts of the system, needs to increase. From this it follows that it is increasingly difficult to design entire integrated circuits - although ever more transistors can be designed over some period of time, over the same period of time the total number of transistors in the circuit increases by an even higher factor. Hence, the *speed* of the design process has to increase significantly.

The existence of both the algorithmic and the productivity gap points to inefficiencies in the design process. At various stages in the process, these inefficiencies form bottlenecks, impeding increased productivity which is needed to keep up with the mentioned algorithmic demand. When putting the focus on the wireless systems domain, additionally the time-to-market for a new product dropped from about three years for the first GSM phones below 18 months for smart phones of the latest UMTS generations [10]. And in this domain launching a product six months early triples profits, whereas being six months late results in breaking even [11].

Naturally, the identification of inefficiencies and novel design strategies have been subject

to extensive research of the electronic design automation (EDA) industry in general and the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms in particular. The major problems are the severe fragmentation of the design process [123], which is briefly surveyed in Appendix A, and the existence of *hard* design tasks, i.e. the hot spots of the design process [63, Knerr et al.][1], for which a feasible solution has not yet been integrated into any commercial EDA tool.

As a direct consequence of the steep requirements on the design cycle, the investigation of synthesised implementation alternatives is disqualified. A trial synthesis even for a small subset of the design would be far too time consuming, thus causing a strong bias in the importance of the design tasks towards those located in the early stages. Their relevance is even more emphasised by the impact on the final system performances. About 90% of the overall costs are determined in the first design stages [111]. Figure 1.2 illustrates the potential of the design decision to influence the cost while the development proceeds. The

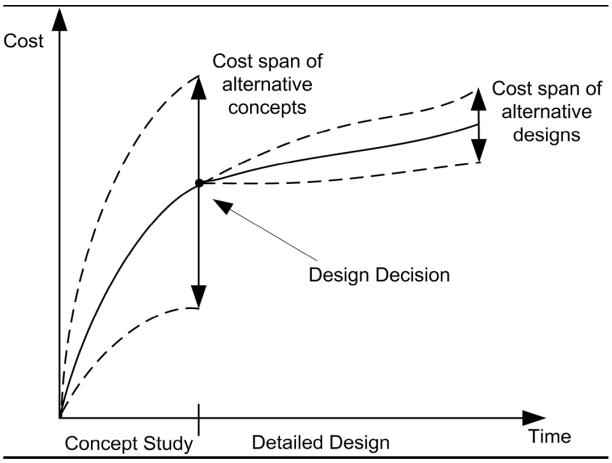


Fig. 1.2: Evolution of the cost span over the development time [8].

most important design tasks located in the *early* region of the design process are system level analysis, system partitioning, floating-point to fixed-point conversion, high level synthesis, and virtual prototyping [16, 63, Knerr et al.].

Since system level analysis is a necessary prerequisite of system partitioning, it shall be briefly concretised. Ideally, crucial decisions are based on a profound knowledge about the intricacies of the design. As soon as the core algorithms, that make up a system's functionality, have been assembled, their internal structure has to be quantised and reliably extrapolated regarding the probable set of future implementation alternatives. For any part of the design a multi-dimensional design space is spanned, in which estimations of crucial design parameters

[1] Cited work which I authored or co-authored is indicated as such by [♯,Knerr et al.]

like timing, area, consumed power, throughput, latency, cost, signal-to-quantisation noise ratio, etc., are incorporated. System level analysis typically comprises a set of different tasks like static code analysis [1, 42], profiling [21], compiler optimisation [9], complexity analysis for verification [102], cost predictions [10], etc. These values serve as system graph enrichments with respect to the targeted platform abstraction, which will be explained in detail in Section 3.4.

System partitioning can be considered as first *constructive* design task beyond pure analyis and characterisation. It consists in the component selection of the underlying platform carrying out the desired applications and the following binding of functional parts to these components. Hence, it constitutes the very core of hardware/software codesign. Its purpose is the identification of the optimal architecture out of many platform alternatives based on computation models for processing elements as general-purpose processors, DSPs, μControllers, application specific instruction set processors (ASIPs), application specific integrated circuits (ASICs), networks, busses, memories, etc. In Figure 1.3 a small cut out from a data flow

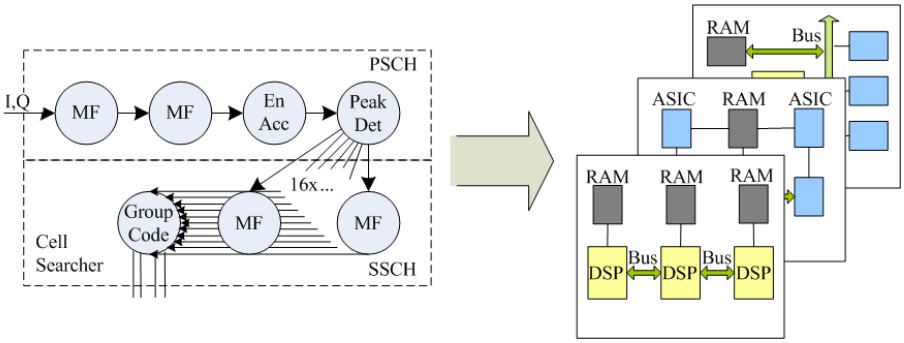


Fig. 1.3: Partitioning: Map functionality to platforms.

oriented system graph from the cell searcher component of an UMTS receiver is depicted. It consists of matched filters (MF), energy accumulation (En Acc), peak detection (Peak Det), and the group code table (Group Code). On the right side some platform models are sketched. The decision which part of the functionality of the system graph is implemented on which resource is not straightforward. Although this graph seems rather simple, some constraints on available silicon area may exist that do not allow for a complete ASIC implementation, and some others e.g. on latency may hinder a complete software implementation on a DSP. The trade-off between competing objectives in larger scenarios with hundreds of communicating functional blocks represents the core of the partitioning problem. Hence, system partitioning is concerned with the formulation of such a multi-objective optimisation scenario, typically with many different objective functions that may comprise complex inter-

nal problems such as multi-resource scheduling with precedence constraints.
The next section highlights the contributions that have been achieved in this field and that are demonstrated in this thesis.

1.2 Contributions

The research work in the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms addresses both recipes to overcome the fragmentation of the design process as well as the development of powerful integrated solutions for the aforementioned design tasks. Concretely, the work covered in this thesis is focused on the research area of system partitioning with respect to the wireless embedded system domain and is designed as integral part of the Open Tool Integration Environment (OTIE). For a brief overview of this environment refer to Appendix A. Although not mainly in the focus of this thesis, it has to be exposed, that significant efforts regarding floating-point to fixed-point conversion, automatic virtual prototyping, automatic verification, system level analysis, and scheduling of SDF graphs with multi-frequency domains have been undertaken in the context of OTIE:

- B. Knerr, P. Belanović, M. Holzer, G. Sauzon, and M. Rupp, "Design Flow Improvements for Embedded Wireless Receivers", in Proc. of the 12th European Signal Processing Conference (EUSIPCO), pages 2015 - 2018, Vienna, Austria, 2004.

- B. Knerr, P. Belanović, M. Holzer, G. Sauzon, and M. Rupp, "Advanced UMTS Receiver Chip Design Using Virtual Prototyping", in Proc. of the 2004 International Symposium on Signals, Systems and Electronics (ISSSE), Linz, Austria, 2004.

- P. Belanović, B. Knerr, M. Holzer, G. Sauzon, and M. Rupp, "A Consistent Design Methodology for Wireless Embedded Systems", EURASIP Journal on Applied Signal Processing, Vol. 2005(16) , pages 2598 - 2612, 2005.

- B. Knerr, M. Holzer, and M. Rupp, "Task Scheduling for Power Optimisation of Multi Frequency Synchronous Data Flow Graphs", in Proc. of the 18th Annual Symposium on Integrated Circuits and System Design, pages 50 - 55, Florianapolis, Brazil, September, 2005.

- M. Holzer, B. Knerr, P. Belanović, and M. Rupp, Efficient Design Methods for Embedded Communication Systems, EURASIP Journal on Embedded Systems, vol. 2006, Article ID 64913, 18 pages, 2006. doi:10.1155/ES/2006/64913.

- P. Belanović, B. Knerr, M. Holzer, and M. Rupp, A Fully Automated Environment for Verification of Virtual Prototypes, EURASIP Journal on Applied Signal Processing, vol. 2006, Article ID 32408, 12 pages, 2006. doi:10.1155/ASP/2006/32408.

The goal of this thesis is to present a set of strategies for the system partitioning problem as they typically appear in wireless embedded systems. This goal necessitates a solid base for the problem formulation that on the one hand accommodates a high degree of flexibility to be utilised in realistic scenarios and that on the other hand exhibits sufficient mathematical rigour to enable the applicability of powerful algorithmic concepts. The predominant contributions of this thesis in the research field of embedded system design can be grouped into three scientific claims as they are listed in the following enumeration:

- **Claim 1**: Analysis of the properties of typical system graphs in embedded systems.
  The variety of existing graph representations in the field is illuminated and the influencing factors, when a certain graph representation is considered to be beneficial, are distinguished. Common terms and definitions of general and applied graph theory are introduced. In this thesis a thorough revision of properties which are very typical for graphs describing signal-processing systems is undertaken. This deeper knowledge finally leads to improvements of existing optimisation methods and eventually to the development of entirely new strategies. The characteristics of these graph properties have been discussed (alongside new algorithms for system partitioning) in the following publications:
  - B. Knerr, M. Holzer, and M. Rupp, RRES: A Novel Approach to the Partitioning Problem for a Typical Subset of System Graphs, EURASIP Journal on Embedded Systems, vol. 2008, Article ID 259686, 13 pages, 2008. doi:10.1155/2008/259686.
  - B. Knerr, M. Holzer, and M. Rupp, "Novel Genome Coding of Genetic Algorithms for the System Partitioning Problem", Proc. of IEEE 2nd Int. Symposium on Industrial Embedded Systems (SIES), pages 134 - 141, Lisbon, Portugal, July, 2007.
- **Claim 2**: A flexible platform abstraction matching the heterogeneity of embedded systems.
  The architecture model constitutes the first of the two necessary parts to assemble a system partitioning scenario. The thesis reviews classical and modern platform concepts and develops a flexible description that offers the designer a large degree of freedom in the number and type of architectural components. In opposite to existing partitioning approaches, an architecture library has been developed in this work that permits *arbitrary* platform compositions and a very detailed communication and processing model.

Such a flexibility is obligatory in the field of wireless embedded systems, since herein the heterogeneity and variety of architectures is protruding. The formulation of the platform abstraction allows for a non-ambiguous and distinct mapping of system graphs to the architecture models. The features and advantages of this very flexible framework have been discussed (alongside the automatic virtual prototyping framework and new algorithms for partitioning) in the following publications:

- B. Knerr, M. Holzer, and M. Rupp, "Extending the GCLP Algorithm for HW/SW Partitioning: A Detailed Platform Model and Performance Improvements", in Proc. IEEE Austrochip, pages 89 - 95, Vienna, Austria, 2006.
- M. Holzer, B. Knerr, P. Belanović, and M. Rupp, "Efficient Design Methods for Embedded Communication Systems", EURASIP Journal on Embedded Systems, 2006.

- **Claim 3**: Analysis and development of classical and new heuristic methods for the system partitioning problem.
  The process of mapping the functional objects to the composed platform model constitutes the second part of the system partitioning problem and occupies the major portion of this thesis. Having analysed typical system graphs in this field and exploiting the high degree of detail of the newly developed platform abstraction, the partitioning problem can now be formulated in a very distinct manner that offers a higher degree of representativeness for wireless embedded systems than any approach before. On this level system partitioning is defined as combinatorial multi-objective optimisation problem and a variety of algorithmic strategies is evaluated with respect to their applicability. The thesis discusses, analyses, and tests deterministic and randomised algorithms based on classical and entirely new approaches that are outlined in the following list:

  - exhaustive search,
  - gradient search [68],
  - global criticality/local phase algorithm [77] and a modified version with substantial improvements published in:
    B. Knerr, M. Holzer, and M. Rupp, "Improvements of the GCLP Algorithm for HW/SW Partitioning of Task Graphs", in Proc. of the 4th IASTED Int. Conf. on Circuits, Signals, and Systems (CSS), pages 107 - 113, San Francisco, CA, USA, November, 2006.
  - simulated annealing [7, 82]
  - *penalty reward* tabu search [142]

- genetic algorithm [35, 147] with a novel genome coding published in:
  B. Knerr, M. Holzer, and M. Rupp, "Novel Genome Coding of Genetic Algorithms for the System Partitioning Problem", Proc. of IEEE 2nd Int. Symposium on Industrial Embedded Systems (SIES), pages 134 - 141, Lisbon, Portugal, July, 2007.
- Kernighan-Lin min-cut [81, 140] and a modified version to be applicable for embedded system partitioning of which an early version has been published in:
  B. Knerr, M. Holzer, and M. Rupp, "HW/SW Partitioning Using High Level Metrics", in Proc. of the Int. Conf. on Computing, Communications and Control Technologies (CCCT), pages 33 - 38, Austin, Texas, August, 2004.
- an entirely new heuristic: restricted range exhaustive search (RRES):
  B. Knerr, M. Holzer, and M. Rupp, RRES: A Novel Approach to the Partitioning Problem for a Typical Subset of System Graphs, EURASIP Journal on Embedded Systems, vol. 2008, Article ID 259686, 13 pages, 2008. doi:10.1155/2008/259686.
  B. Knerr, M. Holzer, and M. Rupp, "Restricted Range Exhaustive Search: A New Heuristic for HW/SW Partitioning of Task Graphs", in Proc. of XXII Conf. on Design of Circuits and Integrated Systems (DCIS), Sevilla, Spain, 2007.
- and a fast scheduling algorithm in comparison with two classical list scheduling techniques published in:
  B. Knerr, M. Holzer, and M. Rupp, "A Fast Rescheduling Heuristic of SDF Graphs for HW/SW Partitioning Algorithms", in Proc. of COMSWARE, New Delhi, India, January, 2006.

1.3 Outline

The thesis is structured as follows:

Chapter 2 reviews the evolution and state-of-the-art in hardware/software codesign. General concepts and terms are introduced and a survey of the technological advances in this area is presented. The variety of existing tool sets is listed and the most significant commercial and academic approaches are outlined. Examples of industry-designed target platforms for modern signal-processing/multimedia systems in the wireless domain are discussed.

Chapter 3 formulates the system partitioning problem as combinatorial multi-objective optimisation problem. The graph representations that are common in signal processing systems are reviewed and the general synchronous data flow graph calculus is highlighted. The architecture library to assemble arbitrary platform models, and the precise communication model is introduced. Then, the mapping problem between system graph and architecture subject to a set of objectives is formulated and cost functions and performance metrics to assess the quality for a feasible mappings are defined. Finally, the embedded multi-resource scheduling problem is briefly outlined.

Chapter 4 deals initially with system graphs for wireless embedded systems with typical properties concerning sparsity, locality, parallelism, etc. It then describes the algorithms that address the system partitioning problem. A smaller part concerns itself with the inherent scheduling problem and two classical and a new technique to solve this problem efficiently. The major part comprises detailed descriptions of a variety of partitioning techniques, partly taken from the related literature and partly representing beneficial modifications and entirely novel approaches. Based on the task graph set the results obtained from extensive test runs of the implemented techniques are demonstrated. Their performance, robustness, and computation time is discussed. Judgements of the applicability of the specific approaches according to system graph properties are given.

Chapter 5 concludes the thesis commenting on the probable impact of the obtained results, the evolution of underlying models of computation with respect to system representations as graphs, and gives perspectives to research fields still open to be investigated.

2 STATE OF THE ART IN HW/SW CO-DESIGN

This chapter reviews the evolution and current state of the embedded system design relating to typical target architectures. Recalling the technological developments mentioned in the introductory chapter, the design flow is compartmentalised and dedicated descriptions of the single components are given. Systems assembled by a technological mixture of hardware and software parts exist for more than 15 years. These scenarios occur in the context of *general-purpose systems* (PCs, workstations), in which the joint development of processor, compiler, and operating system is addressed. The instruction set selection, the exploitation of parallelism by pipelining and scalar units, and the implementation of different caching strategies are typical topics for these systems. A slightly different context is opened by embedded systems, which contains a similar component, namely the joint development of one ore more special-purpose processors and their respective compilers, whereas the other major component in co-design for embedded systems arises from the naturally strong dependency on its *specific purpose* that is captured in the system description. This description of the system's functionality drives the design process, hence the term *system level design* is often utilised in this field [118]. Focusing on the latter scenario a proper definition of the term *embedded systems* is mandatory. Although being around for more than 20 years, a unique, commonly accepted definition is very hard to find. Still recently, Henzinger and Sifakis felt hustled to publish a work about the 'embedded systems design challenge' [58], in which this term has been tried to be captured without losing generality:

"An *embedded system* is an engineering artifact involving computation that is subject to physical constraints. The physical constraints arise through two kinds of interactions of computational processes with the physical world: (1) reaction to a physical environment, and (2) execution on a physical platform."

This definition does not necessarily entail a better palpability of the matter, since it embraces any *built thing that computes*. Besides, this definition completely embezzles the very part of the term, which makes it distinguishable in the first place: *embedded*. Therefore, we adhere to the following definition.

Definition 1 (Embedded System)**.** An **embedded system** is a computing device in general subject to a specific purpose and its implementation is predominantly determined by this

purpose, usually entailing a complete encapsulation into the environment where this purpose is located at.

Unfortunately, even this description becomes blurry, as we explicitly included for instance PDAs and modern mobile phones in the aforementioned examples. To be precise these products are rather general-purpose devices, but on a smaller scale compared to desktop PCs.
The design methodology for embedded systems is intended to offer efficient and comprehensive mechanisms to explore a variety of implementation alternatives. An executable description of the system's functionality is inherently capable to be simulated and to be formally verified. The description has to be complete, i.e. all relevant design traits have to be present at any stage of the design process.

As illustrated by Figure 2.1 the concept of system level design consists of strongly inter-

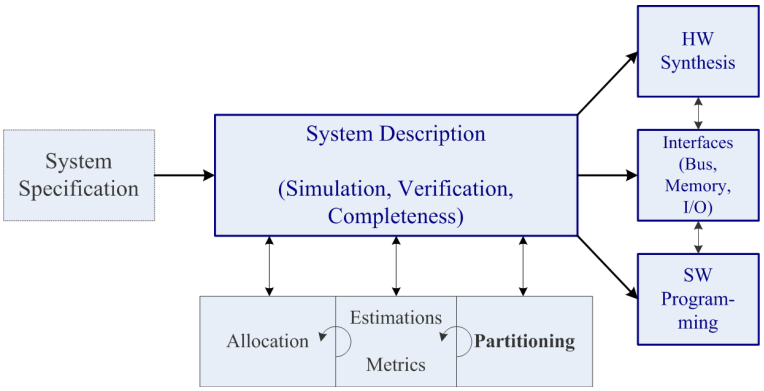


Fig. 2.1: Concept of System Level Design.

connected tasks. The allocation phase comprises the component selection being appropriate for the execution of the system, e.g. processors, memory units, ASICs, etc. Any of those is characterised by a multitude of parameters, e.g. consumed power, number of operations per second, size of the silicon, read and write access times for memory units, etc. Processor structures comprise DSPs, RISCs (reduced instruction set computer), ASIPs, or microcontrollers (μC). According to the simulated, measured, or estimated system characteristics the behavioural components are partitioned onto the chosen architectural components.
These tasks are strongly interleaved. As every specific allocation and partitioning solution generates a new implementation alternative of the system with a new set of mostly estimated characteristics featuring a higher degree of detail, the preceding assumptions have frequently to be reconsidered. Once the design decisions have reached a mature state, the synthesis of the hardware parts, of the software parts, and of the interfaces eventually begins.
The following section gives a more detailed view on a typical platform composition in embedded systems.

2.1 Target Architectures in Embedded Systems

The most typical architectural structures for embedded systems concentrate essentially onto a range of processing units: relevant for software implementations are μCs and DSPs, or even more specific ASIPs, typical candidates for hardware implementation are programmable logic and dedicated data-paths. A mixture of these components is either assembled onto a single chip for which the term System-on-Chip (SoC) has prevailed, or is composed by several chips onto a board system.

Figure 2.2 visualises the common notion of the trade-off between hardware and software

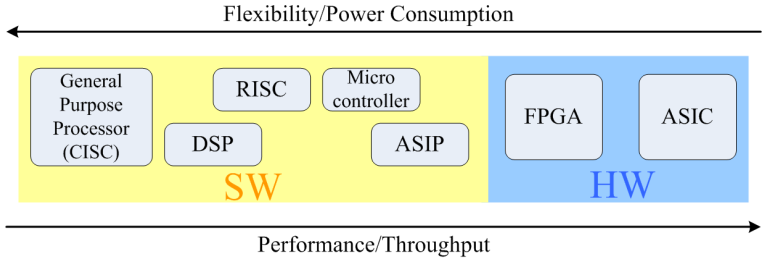


Fig. 2.2: Architectural components and their affiliation to hardware and software.

architectural components. From the left to the right the complexity of the underlying component is decreasing in terms of instruction set, sophisticated memory access, and pipelining strategies. This is counterbalanced by the increase of the computational speed towards ASICs, mostly measured in throughput or number of operations per time unit. The grouping of these processor classes into hardware and software systems has not been clearly defined but is generally understood [31, 118]. More detailed descriptions of the individual components are given in Appendix B.

2.1.1 Platform Composition

The assembly of state-of-the-art systems from the range of available processing elements can be in general be separated in two categories: system-on-a-chip (SoC) and board-level (or multi-chip) system. Typical board-level systems are for instance desktop computers and laser printers. To be precise, SoCs do not comprise every single peripheral on one die, but nearly any portable wireless system encapsulates vital parts on a single silicon substrate, since then the advantages of minor size and low power consumption can be combined. Moreover, the reliability of the circuit benefits well from the assembly on a single chip, such that in many cases even analogue parts (sensor, actuator, power amplifier) are put onto the same chip.

Again a commonly acknowledged definition for SoC is hard to obtain. We adhere to the

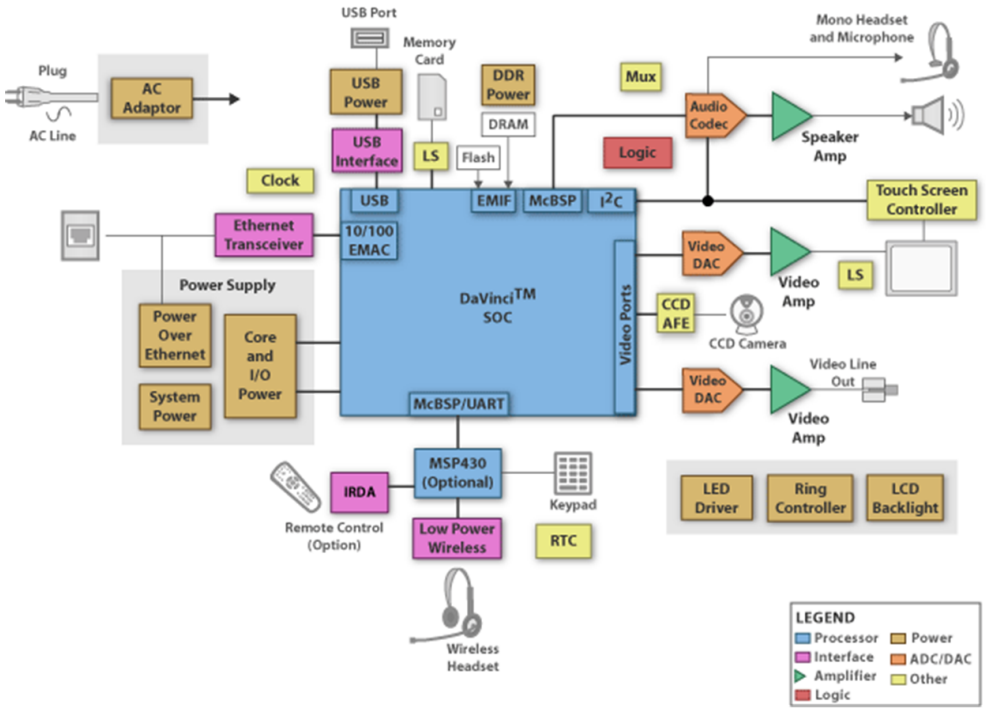


Fig. 2.3: Block structure of an System-on-Chip based design for a video phone [136].

notion of very typical components, of which a selection is assembled on an SoC:

- At least one μP (μController, DSP, RISC, etc.).
- A selection of memory units (RAM, EEPROM, ROM, etc.)
- Serial and parallel ports and interfaces (UART, JTAG, USB, etc.)
- Analogue circuitry (phase locked loops, sensors, oscillators, etc.)
- Analogue interfaces as analogue-digital or digital-analogue converters.
- Power management circuitry (clock gating, voltage/frequency regulation).
- Internal busses (CAN, AMBA, etc.).
- Direct memory access (DMA) to disburden the μProcessor.

In Figure 2.3 a block based structure of a state-of-the-art board level system containing a large SoC is depicted, which is comprised of a selection of components very typical for a modern video phone. The multiplicity of components integrated onto the same die is an immediate consequence from the dramatic advances in microelectronics. In consequence,

the non-recurring engineering costs for SoCs are much higher than for multi-chip (board-level) systems, whereas the production cost for SoCs is lower, once the floor planning is finalised. The overwhelming part of embedded systems in the wireless domain is built around System-on-Chips and in the following section the traditional design flow for such systems is described.

2.2 Embedded Systems Design Flow

The flow of the overall embedded design process, starting with the initial conceptual idea of the system and finishing in the final product, is traditionally divided into a number of abstraction levels. Of the many design methodologies in existence, each prescribes a different set of abstraction levels to make up the design process. As a result, there is no clear and universally accepted division of the design process into a well-defined set of abstraction levels. Rather, there exists a great number of overlapping or even synonymous definitions of abstraction levels, some of which are broad in scope, while others cover small and very specific parts of the design flow, and again some of which enjoy wide recognition in academic and industrial circles, while others are referred to less commonly [45, 50, 75].

The right side of Figure 2.4 shows a collection of some of the most commonly used abstraction levels in academic literature and/or industrial practice, given in their relative order within the overall design process (from high to low level of abstraction). On the left side a co-design flow is depicted accordingly, as it is commonly described in embedded systems. On highest abstraction level an informal specification of the product is put together that does not contain any information regarding its realisation but only a rough sketch of the desired behaviour. Consequently, first refinements intend to identify algorithmic solutions for this functionality, for instance whether a complex computation *could* be performed and not how much resources the computation required. Exploration of different algorithmic variants with respect to precision, computational effort, and robustness is located here indicated by the cycle item in the co-design flow.

The next lower transaction/architecture level is typically occupied by platform modelling, the allocation of architectural components and the partitioning into hardware and software domains. Static code analysis, profiling and co-simulation deliver estimations of timing, silicon area, latency, throughput, regarding the chosen platform composition. Communication models for intra-platform data transfers are applied to simulate the behaviour of memory units and bus structures. According to the platform setup, interface functions are provided that handle the data transfers and transactions between architectural components. Within a single component usually untimed code, e.g. plain `C` is deployed, whereas the interface functions

modelling bus and memory accesses feature a more sophisticated, so called *bus-cycle true* behaviour. Performance estimations are fed back in the co-design flow to alter the platform setup, the partitioning or even fundamental parts in the executable system specification.

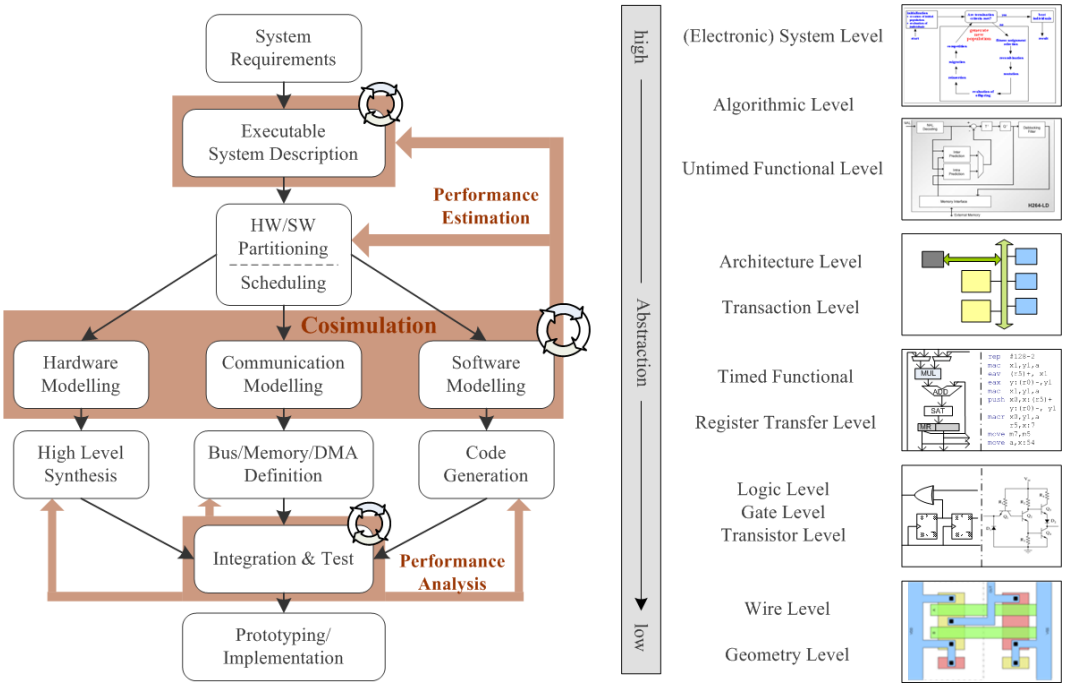


Fig. 2.4: Common abstraction levels and co-design flow for embedded systems.

The hardware assembly of ASICs, ASIPs and FPGA blocks is performed on register transfer level, for which hardware description languages (HDL) like VHDL [33] and Verilog [34] are vicarious. Exact, so-called *bit/cycle true* simulations are possible at this stage, hence is verification of the hardware behaviour. The software assembly onto chosen CISC, DSP, or RISC cores belongs to the same level of abstraction. Once assembly code for these cores exist, the software part does not undergo a further refinement in terms of abstraction levels, whereas the hardware described by VHDL or Verilog does so on its way to final synthesis (via the wire and the geometry level). The integration and verification of both hardware and software components is the final step towards manufacturing the product. In embedded systems it is in general not possible to revise upper levels in the design flow once the integration and verification stage below register transfer level has been reached, due to the harsh time-to-market requirements. Therefore, the performance analysis and verification step may only affect the immediate preceding implementation of the actual function block.

Such an iterative reduction of the abstraction level narrows the design space and increases the accuracy of the model. To avoid the introduction of malfunctions due to the perma-

nent transformation of the system description while decreasing the level of abstraction, the *correct-by-construction* paradigm [37, 125] is of major importance. Formal methods and automated synthesis have to be incorporated whenever possible. For the hardware synthesis from register transfer down to geometry, this goal has already been accomplished. Powerful synthesis tools exist, which enable the designer to automate the implementation process for VHDL/Verilog specified hardware blocks to a very large degree, e.g. Design Compiler from Synopsys [131], HDL coder toolbox from Matlab/Simulink [137], or proprietary tools of FPGA providers like Xilinx [145] and Altera [3]. This success in raising the *lowest* abstraction level to register transfer level by automation of the subsequent refinements is considered to be exemplarily for the future path of electronic system design [32].

With respect to the platform setup and partitioning stage a similar success has not yet been achieved. Paragraphs dedicated to related work in platform abstraction, problem formulation, and partitioning algorithms reside in their respective chapters. In Appendix B.8 a review of commercial and academic co-design frameworks and their respective design languages is given.

3 SYSTEM PARTITIONING

In embedded system design the term *system partitioning* usually comprises the compound of two synthesis tasks: *allocation* and *mapping*. The selection of architectural components is meant by *allocation*, whereas the binding of the functional code of the system to these components is performed during the *mapping*. Note, that often *mapping* and *(system) partitioning* are used synonymously, when the architectural traits of the platform are fixed beforehand and are therefore not part of the optimisation. Usually a variety of constraints exists that aggravate the process of finding a suitable solution. Amongst others there are quantifiable properties for timing, power consumption, compiled code size, silicon area, throughput, latency, implementation cost, etc. and unquantifiable properties as flexibility, humble maintenance effort, testability, reusability, and many more. In general it can be stated that most formulations of this optimisation problem are marked by a huge solution space and analytically intractable relations between the characteristic values that influence the feasibility of a candidate solution.

The following chapter introduces classical graph concepts and fundamental terms. Specific graph structures that are commonly used to describe signal processing systems are surveyed with respect to hierarchy and granularity in Section 3.1. The classical platform model predominantly used for partitioning scenarios is surveyed in Section 3.2. How the underlying architecture for the allocation phase can be modelled to give consideration to modern heterogeneous architectures is described in Section 3.3. Herein, a detailed and flexible component library is introduced, which allows for arbitrary architecture composition, as they are typically found in embedded systems. Section 3.5 sketches the origin of the system partitioning problem in an NP-complete problem known from graph theory. Based upon these classical terms the problem is formulated to map the system graph to components from the new architecture library as a combinatorial multi-objective optimisation problem. Eventually, it is shown that this formulation accommodates an enclosed NP-complete problem in Section 3.5.3, namely the multi-resource scheduling problem.

3.1 Typical Graphs in Embedded System Design

System partitioning of a system relies on its representation in various graph forms. The following basic definitions prepare the ground for the further discussion of these forms.

Definition 2 (Graph)**.** A **graph** $G(\mathcal{V}, \mathcal{E})$ is defined as an ordered pair of a set $\mathcal{V} = \{v_1, v_2, \ldots, v_{|\mathcal{V}|}\}$ of vertices and a set $\mathcal{E} = \{e_1, e_2, \ldots, e_{|\mathcal{E}|}\}$ of edges. The elements of the set $\mathcal{E}$ correspond to unordered pairs of vertices. The vertices belonging to an edge are called endpoint or end vertex of the edge.

Definition 3 (Directed Graph)**.** A **directed graph** $G(\mathcal{V}, \mathcal{E})$ is defined as an ordered pair of a set $\mathcal{V} = \{v_1, v_2, \ldots, v_{|\mathcal{V}|}\}$ of vertices and a set $\mathcal{E} = \{e_1, e_2, \ldots, e_{|\mathcal{E}|}\}$ of edges. Here, $\mathcal{E}$ is defined as set of 2-tuples of vertices $\mathcal{E} = \{(v, w) \,|\, v, w \in \mathcal{V}\}$. The operation **beg** returns the source (tail) vertex, and the operation **end** returns the sink (head) vertex of an edge e as follows: $\forall\, e = (v, w) \in \mathcal{E} : \mathrm{beg}(e) = v, \mathrm{end}(e) = w$. The vertex v is called a direct predecessor of w and w is a direct successor of the vertex v.

Definition 4 (Path/Simple Path/Cycle)**.** A **path** $\mathbf{p}$ from a vertex v to a vertex w in a directed graph is a sequence of vertices $v = s_1,\ s_2,\ \ldots,\ s_n = w$ that satisfies: $\forall\, i,\ i = 2 \ldots n\, \exists\, (s_{i-1}, s_i) \in \mathcal{E}$. The vertex s_1 is the initial vertex of that path and s_n is the terminal vertex of the path. A **simple path** $\mathbf{p}_s$ additionally fulfills the condition: $\forall\, s_i, s_j \in \mathbf{p}, i \neq j : s_i \neq s_j$. If the initial and the terminal vertices of a path are the same, that is, $s_0 = s_n$, then the path is called a **cycle**.

Definition 5 (Directed Acyclic Graph)**.** A **directed acyclic graph** (DAG) is a directed graph that does not contain any cycles. (Or equivalently, all possible paths of a DAG are simple paths.)

Definition 6 (Indegree/Outdegree)**.** The operation **indegree**(v) returns the number of incoming edges to the vertex $v \in \mathcal{V}$ of a directed graph. The operation **outdegree**(v) returns the number of outgoing edges from the vertex $v \in \mathcal{V}$ of a directed graph.

Definition 7 (Rank)**.** The operation **rank**(v) of a vertex $v \in \mathcal{V}$ in a directed graph $G(\mathcal{V}, \mathcal{E})$ is defined as:

$$\forall\, v \in \mathcal{V} :\ \mathrm{rank}(v) \triangleq \begin{cases} 0 & ,\ \mathrm{indegree}(v) = 0 \\ 1 + \max\limits_{u}\ \mathrm{rank}(u) & ,\ \mathrm{indegree}(v) > 0 \end{cases}, \tag{3.1}$$

with $u \in \mathcal{V}$, $e \in \mathcal{E} : u = \mathrm{beg}(e) \wedge v = \mathrm{end}(e)$.

3.1.1 Process Graphs

A common approach for reaching a high perceivability of the functionality within a large and complex system is to use a hierarchical decomposition together with graphical representation. Hierarchical decomposition into subsystems provides a structured view to the system for a group of different designers. In Figure 3.1 a common graphical representation for a system (e.g. in communications) is shown. This starts on the left with a process or task graph describing a modern signal processing system in dependence on its nature as data flow oriented system on a macroscopic level, in which vertices represent processes or functions and edges represent data transfers between them. Nearly every signal processing work suite offers a graphical block-based design environment, which mirrors the movement of data, streamed or blockwise, while it is being processed [12, 98, 133, 137].

Definition 8 (Process or Task Graph)**.** A **process** or **task graph** is a directed graph, in which the vertices represent functional elements performing data processing and in which the edges represent data transfers between those processing elements. The term process (or function) is used synonymously for the vertices of a process graph throughout the thesis.

Furthermore, a detailed view of one vertex may reveal several function calls, which can be assembled to a single larger process (or may be represented as access or call graph if necessary). By zooming in one of the processing vertices, a control flow graph is dismantled as shown in the middle of Figure 3.1, which is in turn assembled by so called basic blocks (Def. 10). And finally, within these basic blocks algebraic expressions are represented with a data flow graph on operational level, a so called expression tree on the right. Beside its purpose of

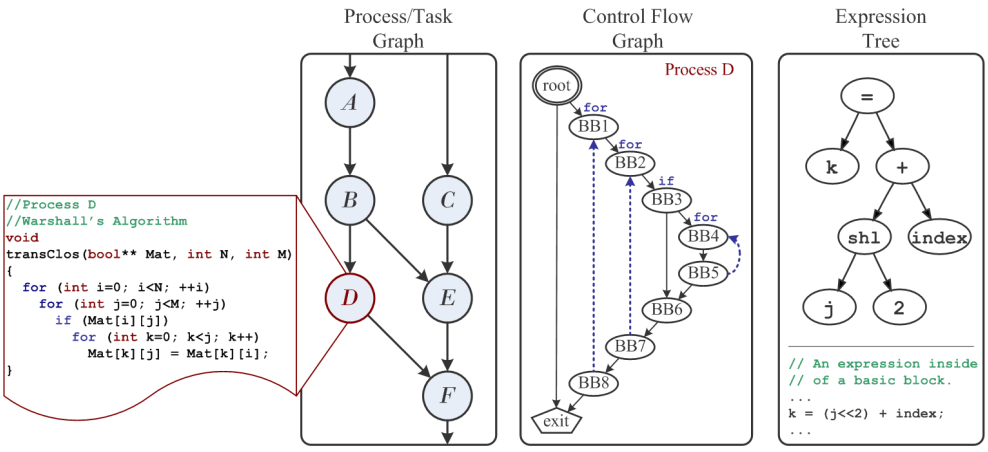


Fig. 3.1: System decomposed into hierarchical graph structures.

a structured view to a system, these graphs and subgraphs allow for automatic analysis in

order to derive characteristic properties and successively for automated partitioning as well. The definition of those graph structures is presented in the following paragraphs.

Definition 9 (Control Flow Graph)**.** A **control flow graph** (CFG) is a directed graph $G(\mathcal{V}, \mathcal{E}, \mathrm{root}, \mathrm{exit})$. It represents a notation of all paths that can be traversed throughout a process during its execution. The control flow of a process enters only at one vertex (root vertex) and leaves the process only at one vertex (exit vertex). The $root$ vertex does not have any incoming and the exit vertex does not have any outgoing edge ($\mathrm{indegree}(\mathrm{root}) = \mathrm{outdegree}(\mathrm{exit}) = 0$).

Definition 10 (Basic Block)**.** A **basic block** (BB) is a vertex of a CFG and contains a sequence of data operations ended by a control flow statement as last instruction.

The statements implementing the control flow are for instance for `C` based languages `if`, `case`, `goto`, `for`, `while`, `do`, `continue`, and `break`. These statements divide the program flow into separable basic blocks and establish the control dependencies between them. Due to programming constructs like loops a CFG is in general not cycle-free. The middle part of Figure 3.1 depicts the resulting CFG structure from the three `for`-loops in process D.

Control flow graph analysis contains many more characteristic values like reducibility of loops, dominance trees [2], and loop cascading [102], feasibility of paths and linear independent paths [119], etc. Partitioning techniques working on these fine-grain graphs are less common, since basic blocks contain normally only a few operations, so that the superposed communication overhead to interconnect different partitions is very high. Still, some approaches exist that partition on CFG level [57].

The sequence of data operations inside of one BB forms itself a data flow graph (DFG) or equivalently one or more expression trees (Figure 3.2 on the right). To briefly sketch how source code is transformed into these graph representations, consider the following example of a `C`-like expression: `factor = offset + period * 60;`.
After lexical analysis, which generates a sequence of symbols and analyses their classification as identifier, operator, constant, expression, etc., a parse tree is assembled as in Figure 3.2 on the left. This parse tree corresponds to the more comprehensive expression tree in the same figure on the right. Usually within a BB more than one expression exists that reads from and writes to the variables. The assembly of all expression trees within a BB then forms a data flow graph on operational level. In such a DFG edges correspond to variables or constants holding information and vertices correspond to operations performed on these variables. For more detailed information on compiler techniques refer to the literature [2].

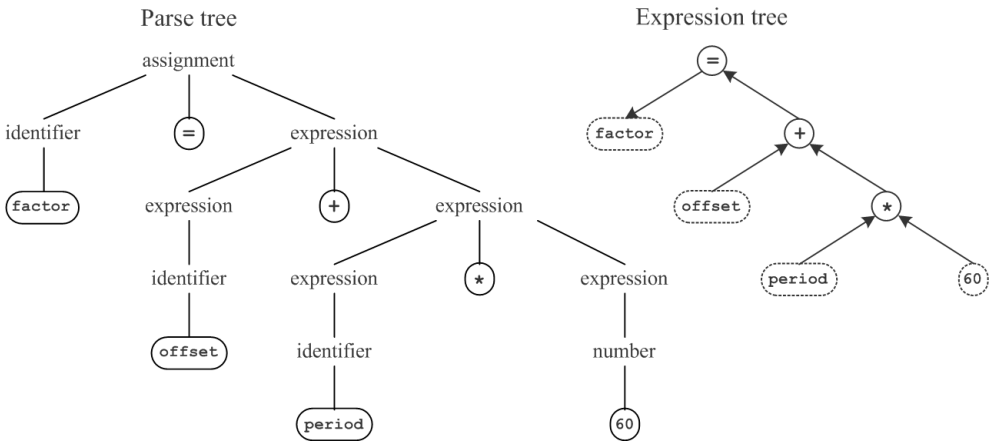


Fig. 3.2: Code fragment representation as parse and expression tree.

3.1.2 Synchronous Data Flow Graphs

To be in accordance with most of the partitioning approaches in the field, a graph representation to be in the form of synchronous data flow (SDF) graphs can be assumed. This model of computation has been firstly introduced in 1987 by Lee and Messerschmitt [99] at UC Berkeley. This model established the backbone of renowned signal processing work suites, e.g. Ptolemy [98] or SPW [29]. SDF captures precisely multiple invocations of processes and their data dependencies and thus is very suitable to serve as a system model for data stream oriented signal processing systems. An indicator for the persistent relevance of the SDF graph representation is provided by the fact that SystemC and its most recent extensions natively support the SDF domain. For instance, the official analog mixed signal (AMS) extension of SystemC [135] just introduced a computation model now providing SDF computation models for all AMS modules.

Definition 11 (Synchronous Data Flow Graph)**.** A **synchronous data flow** (SDF) **graph** is a directed graph $G(\mathcal{V}, \mathcal{E})$. Any edge $e_i \in \mathcal{E}$ is annotated with two numbers $p_i, c_i \in \mathbb{Z}^+$, of which p_i is assigned to the tail of e_i, and c_i to the head of e_i. The numbers p_i represent the number of samples produced per invocation of the vertex at the edge's tail, $\mathrm{out}(e_i)$. The numbers c_i indicate the number of samples consumed per invocation of the vertex at the edge's head, $\mathrm{in}(e_i)$.

In Figure 3.3a, an example of an SDF graph $G(\mathcal{V}, \mathcal{E})$ is depicted, composed of a set of vertices $\mathcal{V} = \{a, .., e\}$ and a set of edges $\mathcal{E} = \{e_1, .., e_5\}$. According to the data rates at the edges such a graph can be uniquely transformed into a single activation graph (SAG) in Figure 3.3b. Every vertex in a SAG stands for exactly one invocation of the process,

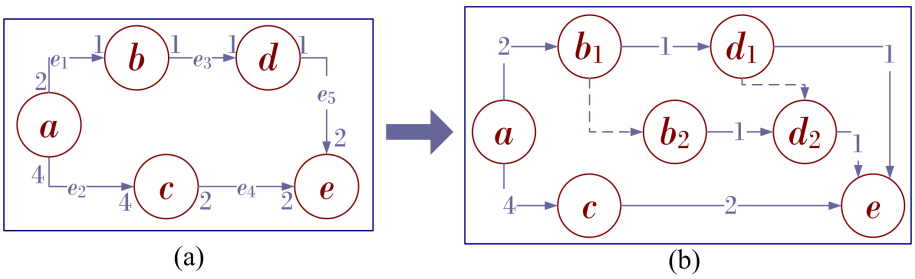


Fig. 3.3: Example of a synchronous data flow graph and its decomposition into a single activation graph.

thus the complete parallelism in the design becomes now visible. Here vertex b and d occur twice in the SAG to ensure a valid graph execution, i.e. every produced data sample is also consumed.
An SDF graph can be formally described by a topology matrix $\mathbf{\Gamma}$, in which any vertex is assigned to a column and any edge is assigned to a row:

$$\mathbf{\Gamma} = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ 4 & 0 & -4 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 2 & 0 & -2 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix}. \tag{3.2}$$

This matrix represents the topology matrix for the SDF graph in Figure 3.3 with the columns corresponding to the vertices in order a to e and the rows corresponding to edges in order e_1 to e_5. This formalism allows for instance for the detection of inconsistent sample rates, when $\mathrm{rank}(\mathbf{\Gamma}) \neq |\mathcal{V}| - 1$, and for periodic scheduling analysis [99]. Many partitioning approaches premise the *homogeneous* form of SDF graphs.

Definition 12 (Homogeneous SDF Graph)**.** A **homogeneous** SDF graph is an SDF graph, if $\forall e_i \in \mathcal{E} : \mathrm{out}(e_i) = \mathrm{in}(e_i)$, or equivalently, if the SDF graph and the single activation graph exhibit an isomorphic graph structure.

In this homogeneous form the connection to process or task graphs can be easily established, as a single activation graph is commonly considered as general process graph.
Although widely accepted for signal processing systems, SDF graphs are restricted to static dataflow behaviour. Therefore, modern SoC applications are often not completely amenable to SDF. Parameterised or cyclo-static dataflow provides for dynamic behaviour by means of structured, dynamic parameter changes in the base model that it is applied to. These modern approaches have not yet made their way into any academic or commercial EDA tool

and their integration into OTIE remains an open issue. A brief survey of these graphs can be found in Appendix C.3.

In this thesis the partitioning backbone supports both general process graph structures as well as general SDF graph representations. According to Lee's work [99], a calculus is provided that validates SDF graphs with respect to their consistency, that resolves feedback edges and multiple invocations and that performs the transformation into single activation graphs, which can be treated equivalently to process graphs. The granularity of the vertices adheres to the common notion of a *partitionable* size that covers the encapsulated functionality of FIRs, DCTs, quicksort, Walsh-Hadamard transform, or similar procedures in consideration of comparable work by other authors in this field [25, 35, 78, 142].

3.2 Classical Platform Model for Hardware/Software Partitioning

When the hardware/software partitioning problem came to be recognised as a hard optimisation problem being encountered in system design at the beginning of the last decade, the perception of such a hardware/software platform has been rather uniform throughout the following ten years.

In 1993 Ernst and Henkel published an early work on the partitioning problem within the COSYMA system [39]. The underlying architecture model has been composed of a programmable processor core, memory, and customised hardware (Figure 3.4). Its composition

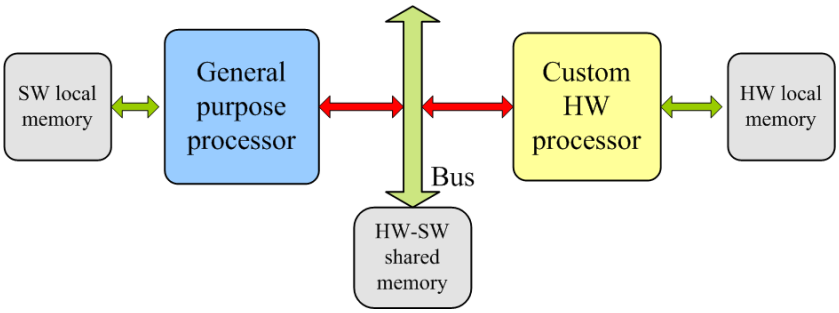


Fig. 3.4: Common implementation architecture.

entailed from the fundamental notion of the existence of two different processing elements: a programmable *software* processor that executes functional code sequentially and in a rather slow manner and a non-programmable *hardware* processor that allows for different function blocks to be carried out concurrently in a rather fast manner. In the beginning this setup was appropriate to excogitate fundamental strategies to cope with problems like minimising

execution time with limited silicon area. The distinction between local and shared memory resources with different access times according to the number of data being transferred offered a flavour of realism but completely neglected the occurrence of collisions on the system bus and the difference between on-chip and off-chip memory access. Still, this platform model was widely adapted by research groups around the world as it can be found by Kalavade et al. [77,78], Eles et al. [38], Vahid et al. [140], Chatha et al. [25], and Srinivasan [129]. However, the crudeness of the model prevented its deployment to any realistic scenario, since too many details affecting execution time and area through communication and control overhead had been neglected.

In the late nineties the first approaches were developed that spent more effort to model communication between processes and different resources more accurately. In 1997 Hardt and Rosenstiel modelled a Sparc CPU based architecture featuring a direct memory access (DMA) controller, cache structures and different execution times for load and store accesses on all communication links [54]. However, in their codesign approach the performance of a system executed on this architecture was crudely estimated, i.e. completely unaware of a possible concurrent process execution on different resources and unaware of colliding data transfers on communication links accessed simultaneously by different resources. In 2002 Wiangtong et al. [142] were one of the first authors that improved this model with a proper scheduling technique avoiding any packet collisions on the bus structures, but neglecting complexity in the variety of communication links, the bus arbitration schemes and the number of available resources.

Realistic communication includes a multitude of connections from direct I/O especially suited for high data rate connections, point-to-point communication via dual-port RAM (DPRAM) or FIFOs , to complex bus structures (CAN, AMBA) with single or pipelined data transfers with optional data protection (parity or cyclic redundancy checks). In 2000 Renner et al. published an in-depth analysis of the mentioned communication resources and how they can be effectively modelled [122]. This work delivered the underpinning of the communication model applied in this thesis, whose intricacies are described in the next section.

3.3 Flexible Platform Model for Heterogeneous Embedded Systems

Until now, publications addressing partitioning for embedded systems base their considerations on the very basic platform described in the preceding section. This model does not have much resemblance with modern heterogeneous architectures as they are used in embedded systems especially in the wireless domain. Therefore, a lot of effort has been undertaken to develop a sustainable fundament for an arbitrary composition of heterogeneous platforms

including DSPs, ASIPs, FPGAs, ASICs, busses, memories, etc.

Consider the following examples of platform architectures taken from industrial designs. The

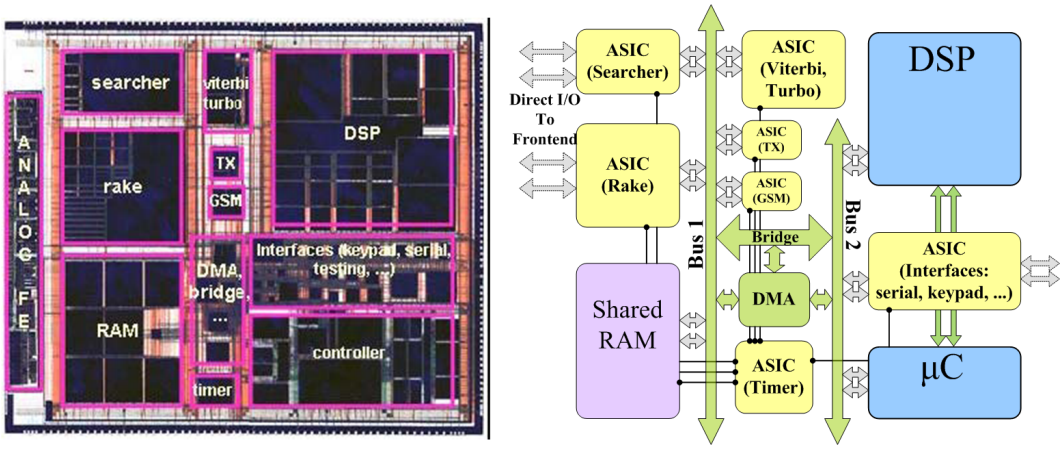


Fig. 3.5: UMTS+GSM baseband transceiver chip [53] and its platform abstraction.

first example originates from our experience with an industry-designed dual-mode UMTS+-GSM baseband transceiver chip [53]. The real reference chip is composed of a DSP core and a microcontroller for the signalling subsystem (the multimedia subsystem has not yet been integrated on the same die in the first version). Even without the multimedia components, i.e. video processing, camera management, etc., the chip features, apart from both microprocessors, several hardware accelerating units (ASICs), for the more data oriented and computation intensive signal processing, two system busses connected by a bridge, a shared RAM for mixed resource communication, direct memory access controllers (DMA) and a direct I/O frontend to peripheral subsystems (the antenna) to disburden the bus. In Figure 3.5 the chip and its abstraction are depicted.

Another industrial example consists in a rapid prototyping board for real-time MIMO OFDM channel emulation [103]. This platform basically consists of a Xilinx Virtex 2 FPGA, a TI C6416 DSP, and a Motorola Coldfire μP. Figure 3.6 depicts a block diagram of the board and its abstracted model. The FPGA provides digital baseband interfaces for input and output signals, as well as for communication over the backplane. The baseband signals are implemented by means of LVDS (Low Voltage Differential Signalling) and using standard Channel Link connectors. The Motorola Coldfire μP is used for configuration of the board and communication with a PC. Although the classical concept of 'one DSP and one FPGA' is adhered to in this example, a realistic platform abstraction necessitates a much more sophisticated approach compared to the classical model, especially with respect to mixed resource communication.

The generalisation of both the baseband receiver chip and the rapid prototyping board com-

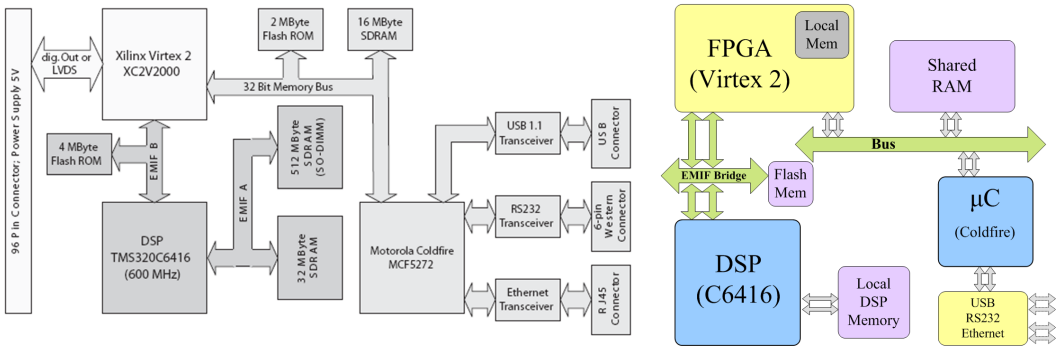


Fig. 3.6: Rapid prototyping board for MIMO OFDM channel emulation [103] and its platform abstraction.

bined with the detailed communication model proposed by Renner [122] resulted in the development of a C++ based architecture library for the models of processing elements DSP, μC, DMA, ASIC, FPGA, bus, FIFO, and direct interconnect. Their individual characteristics and how tasks and data transfers express themselves on those are described in the remaining section.

Four base classes build the backbone of the architecture library: platform, processing element, channel, and memory.

- `Platform`
  This class constitutes the managing object for all following resource classes. It handles lists, vectors and tables for the residing Processor, Channel and Memory objects, as well as the link characteristics about the available connections between those.

- `Processor`
  Processing elements are further subdivided into sequential types (μPs executing compiled code) and concurrent types (FPGAs or ASIC blocks). Processors occupy chip area measured in gates and consume power according to their processing load.

  - `Sequential`
    A sequential processor model manages a schedule of its processes and is restricted in the size of available memory for compiled code. The processor model determines a static schedule avoiding any collisions.

  - `Concurrent`
    A concurrent processor, as an FPGA, has a specific capacity measured in gates of being able to accommodate functional blocks that may run concurrently. Runtime reconfigurability of the FPGA is not yet supported, but currently worked on [65, Knerr et al.].

The architecture supports internal memory structures for both processor types with customisable access times, to allow for e.g. levelled caches with a fast one-cycle read/write accesses or SDRAM components.

- `Channel`
  Processing elements and memory units possess ports to connect to channels over which the data are transferred. The channels subdivide into three types:
  - `Direct`
    A direct channel is an unmanaged resource that introduces area overhead, but avoids additional data overhead and features a high throughput (≈ 100Mbits/s ... 10GBits/s). But in this model direct data transfers are not buffered, hence two directly connected processes have to be executed in immediate succession.
  - `FIFO`
    FIFO channels introduce a larger area overhead depending on the offered FIFO depth and FIFO access logic (busy/idle), and medium available read and write access times ($\approx 1 \ldots 100$MBits/s).
  - `Bus`
    A bus channel offers customisable data overhead for framing, identifier, routing, and error checksums (≈ 4bytes/packet, packet of $0 \ldots 8$bytes) and adjustable arbitration schemes (priority scheduling). Busses are modelled with direct memory access (DMA) controller to decouple process execution and process communication. A typical throughput is 1Mbits/s. A static schedule is created that allows for sequential or concurrent read and write accesses via the bus.
- `Memory`
  A memory block features a number of ports that can be connected to busses or directly to DSPs. It serves typically as shared memory, when data are transferred between processes that run on different resources, or when, while running on the same resource, the internal memory is exceeded. A memory block has customisable access times, silicon area, and power consumption, so that it is also possible to model e.g. off-chip memory.

Any of the characteristic values for a distinct resource is parameterisable, such that a designer has the possibility to specify his component library according to his needs and the available resources in his design flow. A platform example with all the components contained is depicted in Figure 3.7.

To the best of our knowledge, there is no other partitioning framework, which permits arbitrary platform assembly with comparable precision and flexibility. In Chapter 4 this flexibility

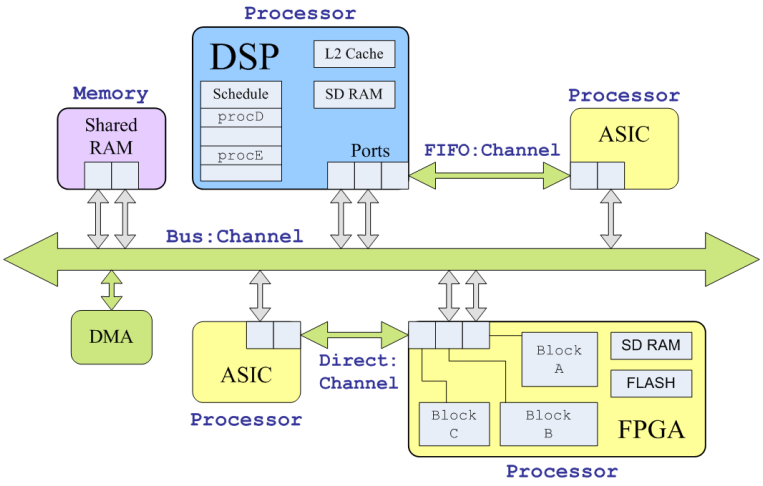


Fig. 3.7: Example of a heterogeneous architecture model.

is applied to evaluate the partitioning algorithms on different platform models and thus to obtain a clearer picture of their performance.

Moreover, although such a mechanism has not yet been deployed for this thesis, the implemented class library allows for the run time modification of the platform traits during a partitioning process, hence enabling a further extension towards automated architecture exploration.

3.4 System Graph Enrichments

With the knowledge about the platform abstraction described in the last section, the system graph is enriched with additional information about execution time, silicon area, amount of data to be transferred between processes, power consumption, etc. The majority of the approaches assigns a set of characteristic values to every vertex representing a process. Most common are execution time et for any available resource, silicon area measured in gates gc both for ASICs and FPGA and compiled code size cs for μPs. To a minor degree power consumption pc is incorporated as well.

Those values are mostly obtained by two different techniques: Static code analysis is based on CFGs and expression trees [67, Knerr et al.], [1, 127] and investigates longest and shortest paths, value lifetimes, memory accesses, tree depths, etc. This technique abstains from time consuming compilation or synthesis steps, but yields usually the least reliable values.
The second technique is dynamic profiling [22, 120], which necessitates representative test data for the application on the one hand and a code compilation step for a μP target or a

VHDL/Verilog description for an ASIC or FPGA target on the other hand. For these dynamic techniques supportive EDA tools exist as e.g. VisualDSP++ from Analog Devices for μPs, that supports profiling and, for some cores, profile guided optimisation with an code size-execution time trade-off. In general, sets of implementation alternatives can be created by varying the compiler options. For instance the minimisation of DSP stall cycles is traded off against the code size for a lower execution time, loops are unrolled, or dedicated memory maps are created. Profiling for FPGA or ASIC implementations is supported by ModelSim [105] from Mentor Graphics. High level synthesis from `C` functions to VHDL is supported by CatapultC [104] from the same vendor, an EDA tool that allows additionally for a fast and reliable analysis by a manual variation of parameters, e.g. the unfolding factor, pipelining, or register usage. It is possible to generate a set of implementation alternatives of every single process for a specific resource such as e.g. a Virtex IV FPGA from Xilinx or any dedicated ASIC library that a chip manufacturer provides.

In this thesis a set of available implementation alternatives $\mathcal{I}(v_i)$ for any process visualised by a vertex v_i of the design graph is assumed that characterises how the process expresses itself on a distinct resource $r \in \mathcal{R}$. For instance, for a design assembled by an Virtex FPGA (VX), an ARM processor and a StarCore (SC), the elements $A^i_{r,j}$ of the implementation set can be listed as follows:

$$\begin{aligned} \forall\, v_i \in \mathcal{V}\, \exists\, \mathcal{I}(v_i) = \{\ & A^i_{\mathrm{VX},1}, A^i_{\mathrm{VX},2}, \ldots, A^i_{\mathrm{VX},k}, \\ & A^i_{\mathrm{ARM},1}, A^i_{\mathrm{ARM},2}, \ldots, A^i_{\mathrm{ARM},l}, \\ & A^i_{\mathrm{SC},1}, A^i_{\mathrm{SC},2}, \ldots, A^i_{\mathrm{SC},m}\ \}. \end{aligned} \tag{3.3}$$

In case of the usual deployed values for execution time, code size, gate count, and power consumption, the jth implementation alternative on resource r would yield a four-tuple: $A^i_{r,j} = (et, cs, gc, pc)$.

In a similar fashion the transfer times tt for the data transfer, visualised by edges e_i, are considered, because several communication channels exist in the design: the bus access to shared memory or another processor (bus), the direct connect (dir), the FIFO connect (ff), or the access to the local memory of resource $r \in \mathcal{R}$ if present:

$$\forall\, e_i \in \mathcal{E}\ \exists\ \mathcal{I}(e_i) = \{\ tt^i_{\mathrm{bus}}, tt^i_{\mathrm{dir}}, tt^i_{\mathrm{ff}}, tt^i_{r_1}, \ldots, tt^i_{r_{|\mathcal{R}|}}\ \}. \tag{3.4}$$

The next section finally surveys the origin of the partitioning problem and introduces the formulation for the given system graph and the platform model under consideration of a set of constraints.

3.5 Problem Formulation

With respect to solution strategies for combinatorial optimisation problems, the term heuristic is frequently used, and in particular in this thesis. In a very general sense a heuristic algorithm is a consistent algorithm for an optimisation problem that is based on some transparent strategy of searching in the set of feasible solutions, and that does not guarantee finding any optimal solution. In a stricter form this term can be defined as follows [68]:

Definition 13 (Heuristic)**.** A **heuristic** is a robust[1] technique for the design of (randomised) algorithms for optimisation problems, and it provides (randomised) algorithms for which one is not able to guarantee at once the efficiency and the quality of the computed feasible solutions, even not with any bounded constant probability $P > 0$.

Frequently, the term *randomised* is omitted, hence e.g. local searches or the Kernighan-Lin min-cut can be viewed as heuristics as well. In the following, randomisation is not explicitly demanded, when an algorithmic technique is denoted as *heuristic*.

Combinatorial optimisation problems that are difficult to solve are said to be NP-hard. It is generally assumed that algorithms do not exist having their run times bounded by a polynomial in the size of the input. For a detailed survey of algorithmic complexity categorised in the classes P, NP, NP-complete etc. consider the work of Garey and Johnson [46]. Herein, only brief definitions of these complexity classes are given:

Definition 14 (Algorithmic Complexity Classes)**.** A decision problem P_d takes an input I (an instance of the problem) and yields as output **yes** or **no**. If an algorithm exists, which is capable to produce the correct answer for any input of length n_I in a polynomially bounded number of steps, P_d is said to be solvable in polynomial time. P denotes the class of all decision problems for which such a polynomial-time algorithm exists. The class of problems for which given answers to this problem can be verified by an algorithm, with a run time that is polynomial in the size of the input, is denoted NP. A decision problem $P_d \in \mathrm{NP}$ is said to be NP-complete, if it is possible to transform every other decision problem $Q_d \in \mathrm{NP}$ to P_d in polynomial time. A decision problem is called NP-hard, if it is at least as hard as every problem in NP. An optimisation problem is therefore already NP-hard if its underlying decision problem (i.e. deciding whether a solution exists with the optimisation objective being better than a given constant K) is NP-complete.

[1] Robust is loosely defined to be 'applicable to a large class of optimisation problems with possibly very different combinatorial structures' [68].

3.5.1 The Classical Graph Partitioning Problem

This section briefly introduces the fundamental problem formulation from which all following considerations are derived. The problem instance is represented by a graph $G(\mathcal{V}, \mathcal{E})$ with vertices' weights $w(v) \in \mathbb{Z}^+ \, \forall \, v \in \mathcal{V}$ and edges' lengths $l(e) \in \mathbb{Z}^+ \, \forall \, e \in \mathcal{E}$, and two positive integers W_{lim} and L_{lim}, as depicted in Figure 3.8 on the left.

The problem is to find a partition of $\mathcal{V}$ into m disjoint sets $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_m$ such that the sum

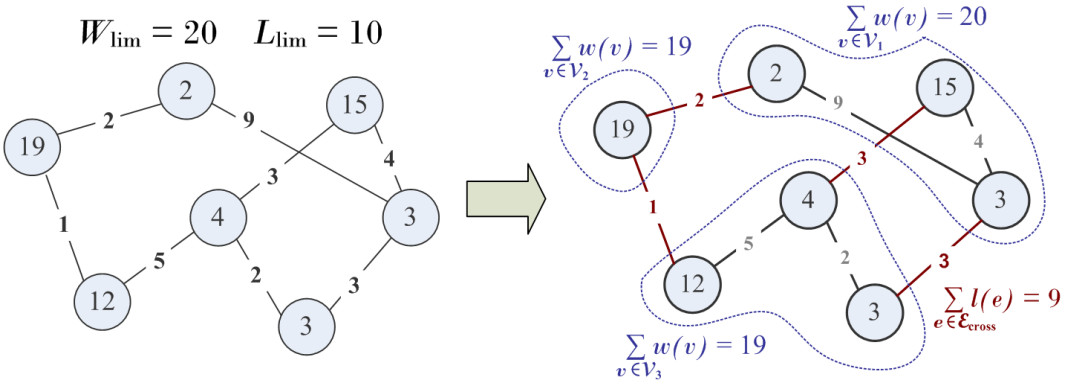


Fig. 3.8: Classical partitioning subject to constraints : $W_{\text{lim}} \leq 20$ and $L_{\text{lim}} \leq 10$.

over the weights of each set's vertices $\sum_{v \in \mathcal{V}_i} w(v) \leq W_{\text{lim}}$ and such that, if $\mathcal{E}_{\text{cross}} \subseteq \mathcal{E}$ is the set of edges that have their endpoints in two different sets $\mathcal{V}_i, \mathcal{V}_j$ with $i, j = 1 \ldots m, i \neq j$, then the sum over the lengths of the crossing edges is $\sum_{e \in \mathcal{E}_{\text{cross}}} l(e) \leq L_{\text{lim}}$.

On the right side in Figure 3.8 such a partition into three disjoint sets has been established such that the constraints are fulfilled. This problem has been proven to be NP-complete by Hyafil and Rivest in 1973 [71] by a transformation from the *partition into triangles* problem, which has also been discussed by Garey and Johnson [46].

For further discussions it is important to note that the origin of the system partitioning problem is in essential aspects identical to this classical problem. In system partitioning the assignment of vertices to disjoint partitions corresponds to the implementation of functionality onto disjoint processing elements (e.g. a set of DSPs), the weights correspond to the resource consumption (e.g. code size or power) of a vertex, and the partition-crossing edges correspond to the data transfers, that may not exceed a certain limit measured in bus utilisation or throughput. Certainly, some objectives, for instance the bus utilisation, cannot be evaluated with simple sums, but necessitate a more elaborate computation model for scheduling and collision arbitration. Still, it is very important to perceive that the hardness of the system partitioning problem does not ensue from this superposed complexity of the objective functions or of the computation model, but rather is an immanent trait of its nature. The

exact details are going to be discussed in the next section of this chapter.

3.5.2 The System Partitioning Problem

In Figure 3.9 the *mapping* problem of a process graph is depicted. The left side shows the system graph, the right side shows an exemplary platform model. With the connecting arcs in the middle, the system graph and the architecture graph compose the *mapping* specification. All possible realisations of a mapping span the design space $\mathbb{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$,

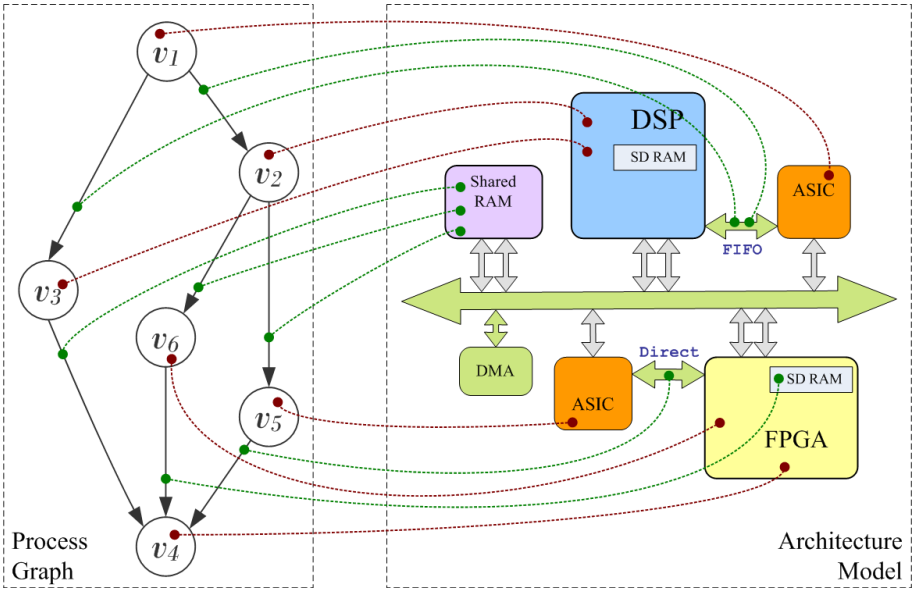


Fig. 3.9: Mapping between task graph and architecture model.

where $\mathbf{x} = (x_1, \ldots, x_n)^T$, $n = |\mathcal{V}| + |\mathcal{E}|$, represents the decision (variable) vector with $x_1 \ldots x_{|\mathcal{V}|}$ identifying the relations $v_i \rightarrow A^i_{r,j}$, $i = 1 \ldots |\mathcal{V}|$ and $x_{|\mathcal{V}|+1} \ldots x_n$ the relations $e_i \rightarrow tt^i_j$, $i = |\mathcal{V}| + 1 \ldots n$. The mapping rules for a *feasible* solution are defined as follows:

Definition 15 (Feasible Mapping)**.** A mapping $\mathbf{x} \in \mathbb{S}_\mathrm{f}(I)$ for a problem instance I with $\mathbb{S}_\mathrm{f}(I) \subseteq \mathbb{S}(I)$ is named **feasible**, if

- all vertices of the system graph are mapped to a `Processor` resource r of the architecture graph for which they possess at least one implementation alternative: $v_i \rightarrow A^i_{r,j} \in \mathcal{I}(v_i)$,
- and all edges of the system graph are mapped to a `channel` resource that is accessible by both processing resources to which the edge's head and tail vertex are mapped to: $e_i \rightarrow tt^i_j \in \mathcal{I}(e_i)$.

The structure of the space of feasible solutions $\mathbb{S}_f(I) \subseteq \mathbb{S}(I)$ has to be organised in terms of a neighbourhood to be accessible by optimisation algorithms as local or gradient searches [68].

Definition 16 (Neighbourhood on $\mathbb{S}_\mathrm{f}(I)$)**.** For every problem instance I, the **neighbourhood on** $\mathbb{S}_\mathrm{f}(I)$ is defined by an operation[2] $n_I : \mathbb{S}_\mathrm{f}(I) \rightarrow \mathrm{Pot}(\mathbb{S}_\mathrm{f}(I))$ such that

- $\forall\, \mathbf{x}_\alpha \in \mathbb{S}_\mathrm{f}(I) : \mathbf{x}_\alpha \in n_I(\mathbf{x}_\alpha)$,
- if $\mathbf{x}_\beta \in n_I(\mathbf{x}_\alpha)$ for any $\mathbf{x}_\alpha \in \mathbb{S}_\mathrm{f}(I)$, then $\mathbf{x}_\alpha \in n_I(\mathbf{x}_\beta)$, and
- $\forall\, \mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathbb{S}_\mathrm{f}(I)\, \exists\, k > 0, \chi_1, \ldots, \chi_k \in \mathbb{S}_\mathrm{f}(I) : \chi_1 \in n_I(\mathbf{x}_\alpha), \chi_{i+1} \in n_I(\chi_i)$ for $i = 1 \ldots k-1$, and $\mathbf{x}_\beta \in n_I(\chi_k)$.

If $\mathbf{x}_\alpha \in n_I(\mathbf{x}_\beta)$ for any $\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathbb{S}_\mathrm{f}(I)$, then $\mathbf{x}_\alpha$ and $\mathbf{x}_\beta$ are called **neighbours in** $\mathbb{S}_\mathrm{f}(I)$.

In other words, n_I is a local transformation on a *feasible* solution $\mathbf{x}_\alpha$ that generates a new *feasible* solution $\mathbf{x}_\beta$ by some local changes of the specification of $\mathbf{x}_\alpha$. As a consequence, a condition for the platform assembly arises with the necessary existence of at least one communication channel between any two processing resources. Then it is ensured that a transformation $n_I(\mathbf{x}_\alpha)$, which changes exactly one vector element x_j, $j = 0 \ldots |\mathcal{V}|$ to obtain a new solution $\mathbf{x}_\beta$ adheres to the aforementioned conditions. In further discussions the explicit reference to the problem instance I is omitted, when the solution spaces are considered.

Feasibility does not ensure validity. Among all feasible solutions $\mathbb{S}_\mathrm{f}$ a subspace of the *valid* solutions $\mathbb{S}_v \subseteq \mathbb{S}_\mathrm{f}$ exists that is determined by a set of constraints b_i like for instance maximum area, maximum response time, and limited memory capacity of the resources given by the requirements of the system. With those constraints, which can be grouped into a vector $\mathbf{b} = (b_1, \ldots, b_k)^T$ and corresponding objectives functions $\mathbf{f} = (f_1, \ldots, f_k)^T$, the term validity can be defined.

Definition 17 (Valid Mapping)**.** A mapping $\mathbf{x}$ is named **valid**, if it is a feasible mapping that fulfills a number of constraints $\mathbf{b}$ for a number of objective functions $\mathbf{f}(\mathbf{x})$ given by a set of inequalities:

$$\mathbf{f}(\mathbf{x}) \leq \mathbf{b}. \tag{3.5}$$

These constraints can be described in a formal manner to establish a multi-objective optimisation (MOO) problem [28]:

$$\mathbf{x}_\mathrm{opt} = \underset{\mathbf{x} \in \mathbb{S}}{\mathrm{argmin}}\{\mathbf{f}(\mathbf{x})\} \tag{3.6}$$

[2] Pot is the powerset operator.

where $\mathbf{f}(\mathbf{x})$ defines a vector involving $k\,(\geq 2)$ conflicting objective functions $f_i : \mathbb{S} \rightarrow \mathbb{R}$, $i = 1..k$.

More concretely, in our case these functions f_i can be obtained as follows:

- For the silicon area constraint $b_{\mathrm{A}} \equiv A_{\mathrm{limit}}$ it is:

$$f_{\mathrm{A}}(\mathbf{x}) = \sum_{i=1}^{n} g(x_i) + \sum_{r \in \mathcal{R}} g(r), \tag{3.7}$$

  where $g(x_i) \neq 0$ is the gate count identified by the relation x_i (in case of an ASIC implementation) and $g(r)$ yields the gate count of any resource present (DSP, bus, FPGA, etc.).

- Similarly, any resource r may possess a limited capacity $b_{\mathrm{A},r} \equiv A_{\mathrm{limit},r}$ measured in gates (e.g. FPGAs):

$$f_{\mathrm{A},r}(\mathbf{x}) = \sum_{i=1}^{n} g(x_i). \tag{3.8}$$

- Any resource r may possess a limited memory capacity $b_{\mathrm{C},r} \equiv C_{\mathrm{limit},r}$ measured in bytes (e.g. DSPs):

$$f_{\mathrm{C},r}(\mathbf{x}) = \sum_{i=1}^{n} c(x_i), \tag{3.9}$$

  where $c(x_i) \neq 0$ is the code size identified by the relation x_i.

- The objective function $f_{\mathrm{T}}(\mathbf{x})$ for the makespan (the time required for a complete system execution) of the system subject to $b_{\mathrm{T}} \equiv T_{\mathrm{limit}}$ cannot be written down in a similar manner, since for any mapping a precedence and resource constrained multi-resource scheduling problem has to be solved. This objective has to be distinguished, since the computation of $f_{\mathrm{T}}(\mathbf{x})$ results in solving an embedded NP-complete problem. This special case is therefore discussed in a dedicated section at the end of this chapter (Section 3.5.3).

As stated before it is in general not possible to find a solution that optimises *all* the objectives simultaneously. A very common description of optimality for a multi-objective optimisation is given by Vilfredo Pareto [117].

Definition 18 (Pareto-optimality)**.** A vector $\mathbf{x}_1$ is Pareto-optimal if there does not exist another vector $\mathbf{x}_2$ such that $\mathbf{x}_2$ dominates $\mathbf{x}_1$. A vector $\mathbf{x}_2$ dominates another vector $\mathbf{x}_1$, $\mathbf{x}_2 \succ \mathbf{x}_1$, iff $\forall\, i = 1 \ldots k\ \mathbf{f}_i(\mathbf{x}_2) < \mathbf{f}_i(\mathbf{x}_1)$.

The subspace of Pareto optimal points $\mathbf{x}_\mathrm{p}$ is called Pareto optimal subspace $\mathbb{S}_\mathrm{p} \subseteq \mathbb{S}_\mathrm{f}$ or shortly Pareto front. More definitions and terms on Pareto-optimality are given in Appendix D.3. Initial work to represent solution spaces with respect to the area and timing trade-off has been formulated in terms of Pareto fronts [62, Knerr et al.]. However, in this thesis a different approach has been chosen to offer a more striking comparison in form of a single value between the optimisation algorithms.
Without further specification any member of $\mathbb{S}_\mathrm{p}$ is an equally preferable solution for the optimisation problem. To resolve the ambiguity between those Pareto optimal solutions $\mathbf{x}_\mathrm{p}$ and in order to calculate a palpable value for their solution quality, the designer formulates his design preferences with the means of weight factors for the k objective functions f_i, represented by the weight vector $\mathbf{w} = (w_1, \ldots, w_k)^T$ with $\sum_{i=1}^{k} w_i = 1$. As the objective functions usually return values subject to their specific dimension (cycle counts, number of gates, number of bytes, etc.) a normalisation has to take place. In our case of existing upper bounds, the respective constraint b_i, and *natural* lower bounds, i.e. the minimum possible value for a specific objective function $\min f_i$, the normalisation is: $f_{i,\mathrm{norm}}(\mathbf{x}) = \frac{f_i(\mathbf{x}) - \min f_i}{b_i - \min f_i}$.
The scalar product of the normalised objective functions with the weight vector yields a scalar value Ω_p for the solution quality as in (3.10):

$$\Omega_\mathrm{p}(\mathbf{x}) = \mathbf{f}_\mathrm{norm}(\mathbf{x}) \cdot \mathbf{w}. \tag{3.10}$$

This weighted sum of normalised metrics is a classical approach to transform a MOO into an single objective optimisation [36]. The minimisation of this weighted sum of normalised objectives $\Omega_\mathrm{p}(\mathbf{x})$ is then the ultimate goal of the algorithm:

$$\min_{\mathbf{x} \in \mathbb{S}_\mathrm{p}} \Omega_\mathrm{p}(\mathbf{x}). \tag{3.11}$$

Please notify, that for those algorithms that comprise a randomised structure in Chapter 4, the outcome naturally varies for different runs of the same algorithm. For those a set of problem instances of at least 30 different runs over any graph is performed, returning 30 cost values, $\Omega_1(G) \ldots \Omega_{30}(G)$ as defined in (3.10), a mean cost value $\overline{\Omega}(G)$, and standard deviation $\sigma(G)$ for any graph G. However, since many different graphs are analysed, these values are averaged again over the respective graph set $\mathcal{G} = \{G_1, G_2, ..., G_{|\mathcal{G}|}\}$, yielding globally averaged values for cost $\overline{\Omega}(\mathcal{G})$ and for standard deviation $\overline{\sigma}(\mathcal{G})$. Henceforth, if not stated otherwise, these globally averaged values are referred to when $\overline{\Omega}$ and $\overline{\sigma}$ are listed omitting $(\mathcal{G})$ for brevity.
In case one of the objective functions f_i exceeds its limit b_i, a penalty function can be applied

to enforce solutions within the limits:

$$\widetilde{f}_{i,\mathrm{norm}} = \begin{cases} f_{i,\mathrm{norm}} & , f_i \leq b_i \\ (f_{i,\mathrm{norm}})^{\eta} & , f_i > b_i \end{cases} , \tag{3.12}$$

with $\eta > 1.0$. Certainly, there are a lot more approaches to apply penalties to invalid solutions with constant, linear, and exponential progression. Two aspects are important for the penalty function: first, the penalties shall allow for a proper comparison of two invalid solutions, hence a continuous differentiable function is preferable; second, the applied penalty shall generate a strong pressure to favour valid solutions to avoid convergence to a solution that erroneously accepts a few *slightly* invalid objectives as long as there are enough objectives with very low valid objectives. In this thesis η has been typically set in the range $2..4$. Additionally, in all algorithms the found valid solutions are managed separately from the invalid ones during the optimisation process. That will be explained in detail in Chapter 4.

For objective functions that are additive in nature, the computation of $\min f_i$ (or $\max f_i$ if required) is a trivial task. More difficult is the generation of a reasonable lower bound for the system's execution time, which depends heavily on the underlying scheduling technique and is hence not analytically ascertainable. To find the exact achievable minimum is equivalent to solving a hard optimisation problem. In the last section of this chapter the scheduling problem is briefly introduced and the consequences for the objective function will become evident.

Since the evaluation of algorithms includes the amount of found valid mappings as substantial metric, a meaningful validity ratio has to be defined:

Definition 19 (Validity Ratio)**.** The **validity ratio** Ψ^A of an algorithm A is defined as the number of valid mappings $|\mathbb{S}^A_{\mathrm{valid}}|$ found by A divided by the overall number of mappings $|\mathbb{S}^A|$ found by A: $\Psi_A \triangleq |\mathbb{S}^A_{\mathrm{valid}}|/|\mathbb{S}^A|$.

Typically, algorithms are tested over graph sets containing a large number of different graphs, and when the algorithm comprises a randomised element, additionally many instances for every single graph are analysed.

Definition 20 (Run Time)**.** The computational **run time** Θ^A of an algorithm A is evaluated by the number of clock cycles consumed by A until termination measured with the high-precision timer `QueryPerformanceCounter` on an AMD ATHLON 64 3000+ Dual Core 1.8GHz PC.

Analogue to the globally averaged cost values, an globally averaged run time $\overline{\Theta}$ is computed, whenever randomised algorithms are applied to many instances of many graphs.

The individual constraints of a mapping scenario can further be specified by the ratios C_i in (21) to give a better understanding of their strictness with respect to the present resources in a mapping scenario.

Definition 21 (Constraint Ratio)**.** A **constraint ratio** $C_i \in [0, 1]$ is defined by the following equation:

$$C_i \triangleq \frac{b_i - \min f_i}{\max f_i - \min f_i}. \tag{3.13}$$

Small values for $C_i < 0.5$ correspond to rather strict constraints, as the objective gets closer to its minimum, and accordingly larger values $C_i > 0.5$ correspond to rather loose constraints.

3.5.3 Embedded Scheduling Problem

The objective function $f_{\mathrm{T}}(\mathbf{x})$ for the makespan of a solution $\mathbf{x}$ necessitates a more elaborate description, since it is not educible with a closed formula as the objective functions for area $f_{\mathrm{A}}(\mathbf{x})$ and code size $f_{\mathrm{C}}(\mathbf{x})$.
Due to the presence of architectural components with sequential character, the mapping problem includes another hard optimisation challenge: the generation of (near-)optimal schedules for every mapping instance. For example, for any two processes mapped to a DSP or data transfers mapped to a bus that overlap in time, a collision has to be resolved. Assume a schedule for the mapping example in Figure 3.9 has to be generated. In Figure 3.10 a time table is depicted, in which the processing times and the read and write accesses are visible. Herein, concurrent devices (FPGA) are indicated by a wide track accommodating parallel tasks, whereas sequential devices (all others) feature a narrow track accommodating tasks in a distinct consecution. Typically, for a single mapping instance many different schedules exist. This graph structure allows for the concurrent execution of the processes v_2 and v_3. But as the current mapping instance assigns both processes to the DSP, a collision has to be arbitrated. The processing order of v_2 and v_3 could as well be vice versa, affecting the succeeding timing of processes and data transfers, and thus in general the schedule length. Similar situations arise for any sequential device on many occasions in a mapping instance of a realistic size.

Basically, such a mapping scenario, for which a schedule has to be produced, includes the problem characteristics of two classical NP-complete scheduling problems, multiprocessor scheduling [46] and precedence constrained scheduling [139]. In Appendix D these problems are shortly described, as well as how the terms used in their formulation can be comprehended regarding the scheduling scenario encountered in this thesis.

Our main interest in this aspect is focused on very fast scheduling techniques accepting sub-optimal results with a reasonable quality for two reasons: this problem has to be evaluated in

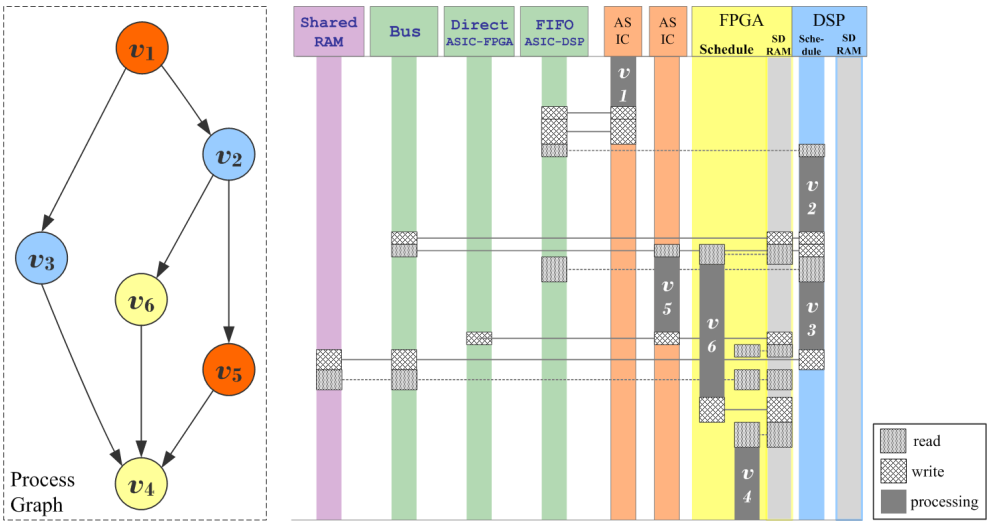


Fig. 3.10: Multi-resource schedule for a simple process graph.

the very core of a partitioning algorithm that naturally visits a huge number of solutions, and the system execution time is, although typically important, just one constraint among others. In this thesis two very common list scheduling techniques are utilised: Hu's Highest Level First (HLF) scheduling [69] and Hwang's Earliest Task First (ETF) scheduling [70]. Considering partitioning problems, in which the system time has outstanding significance, a new scheduling algorithm is introduced that is more complex but achieves a better performance in Section 4.2.2.

The objective function $f_{\mathrm{T}}(\mathbf{x})$ that contributes to the quality of the solution naturally depends on the chosen scheduling technique: $f_{\mathrm{T}}^{\mathrm{HLF}}(\mathbf{x}) \neq f_{\mathrm{T}}^{\mathrm{ETF}}(\mathbf{x})$. With respect to the minimum (or maximum) values that could be returned by this function, only a lower (or upper) bound can be calculated. The global lower bound on the system's execution time $\min f_{\mathrm{T,global}}$ is then the length of the critical path through the process graph, with every process featuring its minimum possible execution time and assuming full parallelism (i.e. neglecting any collisions). The data transfer times are analogously assumed to be handled by the resource with the highest throughput, again neglecting any collisions.

4. ALGORITHMS FOR SCHEDULING AND PARTITIONING

This chapter is subject to the range of algorithms developed and implemented in the course of this thesis and hence constitutes its main goal: the enrichment of the Open Tool Integration Environment with powerful partitioning techniques. The gained insights over the implemented approaches with respect to their applicability and performance will be presented. The first part highlights important system graph properties that shed some light on beneficial traits of typical system graphs in the embedded design domain. These will be of relevance for the discussion of the new approaches with respect to partitioning as well as to scheduling. The second part then addresses the deployed scheduling techniques in detail. Two of these are renowned algorithms, a third one represents a new approach developed for this thesis. The last part of this chapter is concerned with a range of algorithms for the system partitioning problem, their advantages, drawbacks and several new aspects about how these algorithms can be applied in a beneficial manner. Several techniques will simply serve as benchmarks and are merely adapted to the given problem formulation, other existing techniques are substantially improved with dedicated operators and codings, and finally completely new approaches will be introduced and judged by their performance compared to each other.
In this chapter the algorithmic complexity of optimisation techniques is mentioned several times. In these cases we measure this complexity as asymptotic efficiency of the algorithms in relation to the input size for a given problem instance [68]. The standard notation in algorithmics for this form of complexity is defined as follows:

Definition 22 (Asymptotic Efficiency)**.** Let $f : \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$ be a function. We define

$$\mathrm{O}(f(n)) = \{t : \mathbb{N} \rightarrow \mathbb{R}^{\geq 0} | \, \exists \, c, n_0 \in \mathbb{N}, \text{such that} \, \forall \, n \in \mathbb{N}, n \geq n_0 : t(n) \leq c \cdot f(n)\}. \quad (4.1)$$

If $t(n) \in \mathrm{O}(f(n))$, we say that **t does not grow asymptotically faster than f**.

In that sense $f(n)$ serves as upper bound for the notion of algorithmic efficiency of an optimisation technique for a problem instance of input size n.

4.1 Specific Properties of Typical Process Graphs

The very first step to analyse an existing or to design a new algorithm for partitioning lies in the acquisition of a profound knowledge about the problem formulation. Here, it consists of two components: the target architecture and the system graph. Both have already been discussed in the last chapter, but for the latter a few more considerations shall be made. General random graphs, if not further specified, can differ dramatically from the specific properties found in a distinct application domain. Graphs in electronic system design, in which programmers capture their understanding of the functionality and of the data flow, can be isolated by their value ranges for specific graph properties, which will be illustrated in this section.
A review of the literature in the field of partitioning and electronic system design in general, regarding realistic and generated system graphs has been performed. The value ranges of the properties discussed below have been extracted from the following sources:

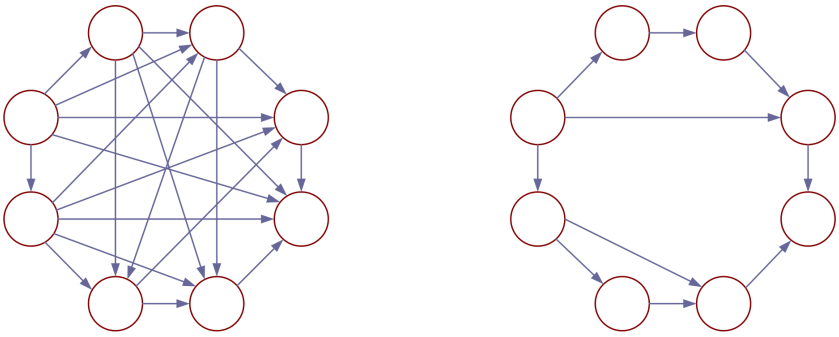
- a UMTS baseband receiver system from Infineon Technologies [53],
- an xDSL transceiver modem for combined classical and wide band telecommunication form Infineon Technologies,
- the realistic examples taken from the benchmark library for multiprocessor scheduling algorithms of the Kasahara Laboratory [79], also called the standard task graph set,
- the graph structures analysed by Kwok et al. [94] and Wiangtong et al. [142].

These examples are described in detail in Appendix C.1.

Depending on the granularity of the graph representation, the vertices may stand for a single operational unit (MAC, Add, Shift) [39] or have the rich complexity of an MPEG or H.264 decoder. In the considered partitioning scenario the majority of the published algorithms [26, 35, 78, 142] decide for medium sized vertices that cover the functionality of FIRs, IDCTs, Walsh-Hadamard transforms, FFTs, sorting algorithms or similar procedures. This size is commonly considered as *partitionable*, i.e. the trade-off between introduced data transfer overhead and the gain in performance may yield an overall improvement. Quantised, the *partitionable* size represents 20 to 50 lines of hand written code of $\approx 5 \ldots 15$kbyte compiled C-code.

The following graph properties are related to system graphs with such a granularity.

Definition 23 (Density/Sparsity)**.** The **density** or **sparsity** ρ of a graph is the ratio of number of edges divided by the number of vertices: $\rho \triangleq \frac{|\mathcal{E}|}{|\mathcal{V}|}$.



(a) dense ($\rho = 21/8$) (b) sparse ($\rho = 10/8$)

Fig. 4.1: Density of graph structures.

A directed graph is considered as *dense*, if $|\mathcal{E}| \sim |\mathcal{V}|^2$, and as *sparse*, if $|\mathcal{E}| \sim |\mathcal{V}|$, see Figure 4.1. Here, an edge corresponds to a directed data transfer, which is either existing between two vertices or not. The possible values for the number of edges calculate to $0 \leq |\mathcal{E}| \leq (|\mathcal{V}| - 1)|\mathcal{V}|$, and for directed *acyclic* graphs to $0 \leq |\mathcal{E}| \leq \frac{(|\mathcal{V}|-1)|\mathcal{V}|}{2}$ [126]. The considered system graphs are biased towards *sparse* graphs with a density ratio of about $\rho = \frac{|\mathcal{E}|}{|\mathcal{V}|} = 0 \ldots c|\mathcal{V}|$, with $c \ll |\mathcal{V}|$.

Definition 24 (Degree of Parallelism)**.** The **degree of parallelism** γ is defined as $\gamma \triangleq \frac{|\mathcal{V}|}{|\mathcal{V}_{\mathrm{CP}}|}$, with $|\mathcal{V}_{\mathrm{CP}}|$ being the number of vertices on the critical path [110].

In a weighted graph scenario this definition can easily be modified towards the fraction of the overall sum of the vertices' (and edges') weights divided by the sum of the weights of the vertices (and edges) encountered on the critical path. Apparently, this modification fails when the vertices and edges feature a set of varying weights, as it is in our case for the execution times et and transfer times tt.
Hence, for every vertex and every edge an average is built over their possible execution and transfer times, et_{avg} and tt_{avg}. These averaged values then serve as unique weights for the time weighted degree of parallelism γ_{T}:

Definition 25 (Time Weighted Degree of Parallelism)**.** The **time weighted degree of**

parallelism γ_{T} is defined as follows:

$$\gamma_{\mathrm{T}} \triangleq \frac{\sum\limits_{v_i \in \mathcal{V}} et^i_{\mathrm{avg}} + \sum\limits_{e_j \in \mathcal{E}} tt^j_{\mathrm{avg}}}{\sum\limits_{v_i \in \mathcal{V}_{\mathrm{CP}}} et^i_{\mathrm{avg}} + \sum\limits_{e_j \in \mathcal{E}_{\mathrm{CP}}} tt^j_{\mathrm{avg}}}. \tag{4.2}$$

This property may vary to a higher degree, since many chain-like signal processing systems exist as well as graphs with a medium, although rarely high, inherent parallelism, $\gamma_{\mathrm{T}} = 1 \ldots \sqrt{|\mathcal{V}|}$.

However, when respecting precedence graphs to be sequenced or scheduled on a set of resources, the aforementioned property for parallelism does not sufficiently reflect the graph's capability to rearrange its vertices. Consider the two graphs in Figure 4.2. Both feature the very same γ (and ρ), however the left graph is 'less parallel' in such a way, that e.g. the number of parallel vertices is much lower, as indicated by the highlighted vertex v_5 with its parallel counterparts accentuated in blue. A structural metric to cover this property is here

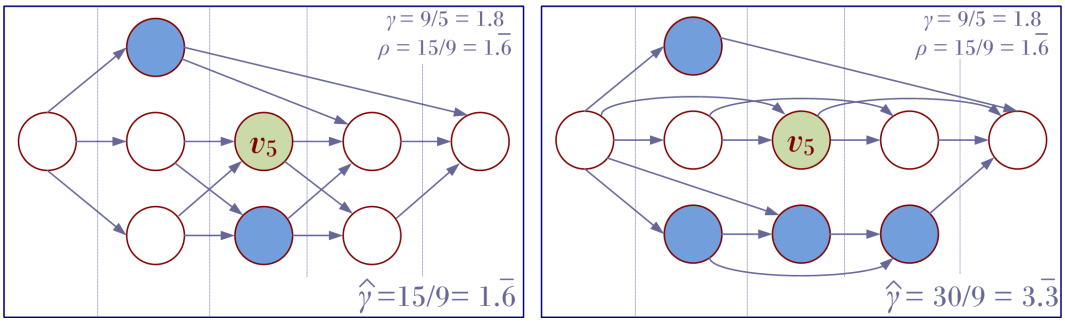


Fig. 4.2: Parallel vertices seen by vertex v_5.

introduced.

Definition 26 (Average Number of Parallel Nodes)**.** The **average number of parallel nodes** $\widehat{\gamma}$ is defined as

$$\widehat{\gamma} \triangleq \frac{\sum_{v_i \in \mathcal{V}} \mathrm{par}(v_i)}{|\mathcal{V}|}, \tag{4.3}$$

where $\mathrm{par}(v)$ yields the number of vertices that are neither successors nor predecessors of vertex v.

The value range of $\widehat{\gamma}$ lies between 0 for chain-like graphs with a single valid sequence and $|\mathcal{V}| - 1$ for completely disconnected graphs ($|\mathcal{E}| = 0$). This structural feature affects dramatically the size of the search space when trying to schedule these precedence graphs on one

or more resources.

Definition 27 (k-Locality)**.** The k-**locality** $0 < k_{\mathrm{loc}} \leq |\mathcal{V}| - 1$ is defined as follows: when all vertices of a graph are written as elements of a vector with indices $i = 1 \ldots |\mathcal{V}|$, then for a graph with $k_{\mathrm{loc}} = j$ edges may only exist between vertices whose indices do not differ by more than j. The arrangement of the vertices is such that k_{loc} is always the smallest possible [126].

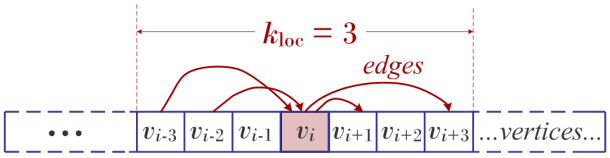


Fig. 4.3: The k-locality graph property with $k_{\mathrm{loc}} = 3$ shown as vector.

For a better understanding this property is depicted in Figure 4.3. The k-locality affects the density in such a way that is serves as an upper bound for ρ, since the maximum number of edges in a directed k-locality graph is $\rho \leq k_{\mathrm{loc}}|\mathcal{V}|/|\mathcal{V}| = k_{\mathrm{loc}}$.

Human made graphs in electronic system design reveal a strong affinity to this locality property for rather small k_{loc} values compared to its number of vertices $|\mathcal{V}|$. The generation of a k-locality graph is simple, but the computation of the k-locality for a given graph is a hard optimisation problem itself [126]. Hence, we introduce a related property to describe the locality of a given graph:

Definition 28 (rank-Locality)**.** The rank-**locality** r_{loc} is defined as

$$r_{\mathrm{loc}} \triangleq \frac{1}{|\mathcal{E}|} \sum_{e_i \in \mathcal{E}} \mathrm{rank}(\mathrm{end}(e_i)) - \mathrm{rank}(\mathrm{beg}(e_i)). \tag{4.4}$$

For a better understanding this property is depicted in Figure 4.4. At the bottom of this figure the rank levels are annotated.

For sparse graphs $\rho = c|\mathcal{V}|$, with $c \ll |\mathcal{V}|$, there is a nearly linear dependency between the product $\gamma_{\mathrm{T}} r_{\mathrm{loc}}$ and k_{loc}, here plotted for graph sizes $|\mathcal{V}| = 50$ averaged over 50 graphs for any given k_{loc}: The rank-locality can be calculated very easily for a given graph. Very low values, $r_{\mathrm{loc}} \in [1.0 \ldots 3.0]$, are reliable indicators for system graphs in signal processing. It is hence possible to assess the locality property of a sparse graph by this product without being

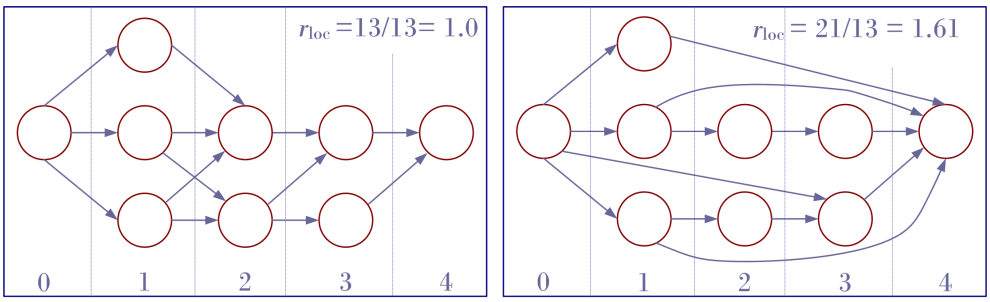


Fig. 4.4: Examples for the rank-locality of two different graphs according to (4.4).

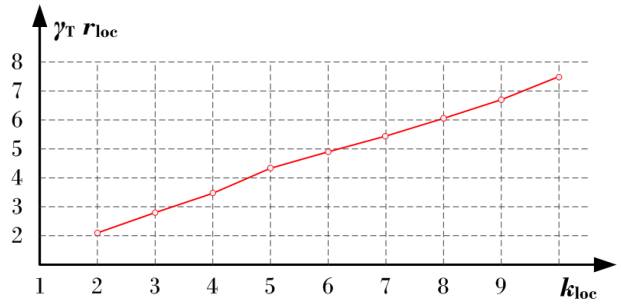


Fig. 4.5: Linear dependency between $\gamma_{\mathrm{T}} r_{\mathrm{loc}}$ and k_{loc}.

forced to solve a hard optimisation problem to derive the minimum k_{loc}.

All of the aforementioned properties can be efficiently computed before any scheduling or partitioning algorithm starts: $r_{\mathrm{loc}}, \rho, \gamma_{\mathrm{T}}$ can be obtained with linear efficiency $\mathrm{O}(|\mathcal{V}| + |\mathcal{E}|)$ for directed graphs with no (or at least reducible [56]) loops. The property $\widehat{\gamma}$ can be obtained via the transitive closure in $\mathrm{O}(|\mathcal{V}|^3)$ [126].

A large number of graphs of different sizes and characteristics have been generated to provide a fundament for a reliable performance analysis and comparison between different techniques. In Appendix C.2 the generation of these graphs is described in more detail.

4.2 Algorithms for Scheduling

Although the discussion of existing partitioning algorithms occupies the major portion of this thesis, the scheduling problem has to be addressed as well, since it is integral part of any of those partitioning algorithms. Therefore, a comparably brief discussion of two renowned scheduling techniques is given, as these are applied in the core of the partitioning algorithms. Moreover, fundamental ideas resulting from the consideration of typical graph properties in

embedded systems and how these affect the workings of partitioning algorithms also lead to novel ideas for the applied scheduling techniques. Eventually, a new scheduling technique is presented in this thesis that exploits specific properties of typical system graphs.

4.2.1 Classical Scheduling Techniques

A very common strategy to solve occurring collisions in a fast and easy-to-implement manner is the deployment of a priority list. In the Highest Level First (HLF) approach every process is annotated with a priority value, such that, in case of a collision, that process is scheduled earlier, which features the higher priority value. The computation of the priority levels is performed according to the processes' critical path as proposed by Hu [69], that is the largest sum of execution times along any directed path from vertex v_i to an exit vertex, over all exit vertices of the graph.
In Figure 4.6 an example graph is depicted, which illustrates how the priority values, or *Hu levels*, are calculated before the mapping algorithm starts. The execution times of vertices and edges are summed along the critical path (the dashed arrows) from any vertex to the exit vertex. As in our case the execution and transfer times vary depending on their current

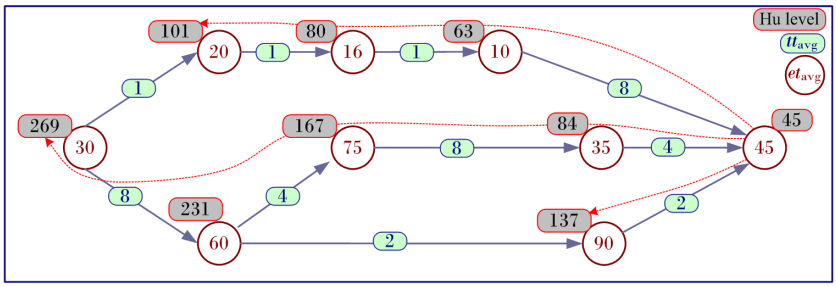


Fig. 4.6: Computation of Hu priority levels based on critical path analysis.

mapping, it is either possible to obtain a single execution time by averaging over all possible implementation alternatives of a vertex and calculate the vertex' priority level before the partitioning starts, or to calculate the priority levels anew for any distinct mapping solution that is visited during the partitioning algorithm proceeds. Let's name the first case HLF scheduling with *static* priorities and the latter HLF scheduling with *dynamic* priorities.

In Figure 4.6 the underlying execution and transfer times, et_{avg} and tt_{avg}, are averaged over all implementation alternatives for any vertex and data transfer for a *static* HLF scheduling. In this case the priority list has to be generated just once and is not altered during the mapping algorithm, hence collision arbitration is a constant time look-up. The dynamic priorities have to updated for every alteration of the mapping, but the inaccuracies imported by the averaging can be avoided. The asymptotic efficiency of this update is $\mathrm{O}(|\mathcal{V}| + |\mathcal{E}|)$

for a directed graph, being acyclic or with reducible loops. The scheduling process can then be implemented by a breadth first search, in which the queue is filled with ready vertices ordered according to their priorities. When the queue has a maximum length $L_{\mathrm{Q}} \leq |\mathcal{V}|$, then the asymptotic efficiency of HLF for a given mapping is $\mathrm{O}((|\mathcal{V}| + |\mathcal{E}|)\, L_{\mathrm{Q}} \log(L_{\mathrm{Q}}))$.

Another simple but powerful approach is the Earliest Task First (ETF) algorithm that uses static priorities and assumes also a bounded number of processors [70]. However, a vertex with a higher priority may not necessarily get scheduled before the vertices with lower priorities. This is because at each scheduling step, the ETF algorithm first computes the earliest start times for all the ready vertices and then selects the one with the smallest value of the earliest start time. A vertex is ready if all its direct predecessors have been scheduled. The earliest start time of a vertex is computed by examining the start time of the vertex on all possible processing elements exhaustively. When two vertices have the same value of the earliest start times, the ETF algorithm breaks the tie by scheduling the one with a higher Hu priority. The efficiency of the ETF algorithm is described to be $\mathrm{O}((|\mathcal{V}| + |\mathcal{E}|)\, |\mathcal{R}|\, |L_{\mathrm{Q}}|)$, where $|\mathcal{R}|$ is the number of processors given. Note, that ETF alters the mapping of the processes to resources, and is in that sense a partitioning algorithm instead of a pure collision arbiter. Since this alteration is only subject to execution time and ignores all other objectives, ETF has to be redefined for this thesis to be applied to a given mapping. Whenever two processes collide, that process is scheduled first, which features the lower start time. ETF can be implemented analogically to HLF with a breadth first search queue that is ordered by increasing start times, thus featuring the same asymptotic efficiency. When colliding processes possess the same start time, the current Hu priority levels can serve as fallback mechanism.

As many more scheduling techniques exist in the literature, a complete discussion is far beyond the scope of this thesis. A good overview of a large number of popular scheduling techniques is given by Kwok et al. [94]. Therein six scheduling algorithms are discussed of which the ETF reveals a good performance, a very good asymptotic efficiency, and which is easily applicable in our scenario. Although HLF itself is not explicitly discussed in Kwok's work, it constitutes the core of four out of the six discussed algorithms, and its dynamic version reveals a better performance than the dynamic ETF algorithm. This will be shown in the following sections. For these reasons ETF and HLF have been chosen to serve as benchmarks for the newly developed scheduling algorithm.

4.2.2 Local Exploitation of Parallelism

The main idea of this new approach can be sketched as the optimisation of the exploitation of parallelism in the system graph for a limited part of the schedule. The idea has emerged in order to develop a scheduling algorithm that takes the parallelism offered by the system

graph and the platform into account, while preserving a fast collision arbitration.
In Figure 4.7 a typical situation is demonstrated, in which the scheduler is responsible whether the returned value for the system's execution time of the considered partitioning solution is better or worse depending on the collision arbitration 'C before B' (Schedule 1) or 'B before C' (Schedule 2). The vertices are annotated with their execution times and the resulting Hu priority levels are depicted in the vector on the right side of the figure. In this example both the HLF and ETF scheduling would come to the wrong decision, as it does not consider the parallelism in the design. A first intuitive notion is that the schedule decision should ensure that the inherent parallelism is exploited. Based on this concept, a new strategy has been developed, whose feasibility for a partitioning algorithms with an incremental search space traversal, such as simulated annealing and tabu search, could already be demonstrated [90, Knerr et al.]. In this thesis a completely revised version of the LEP algorithm is presented that does not require an incremental search space traversal of the partitioning algorithm. It is generally applicable within e.g. genetic algorithms, which typically create new solutions being very dissimilar from older ones.

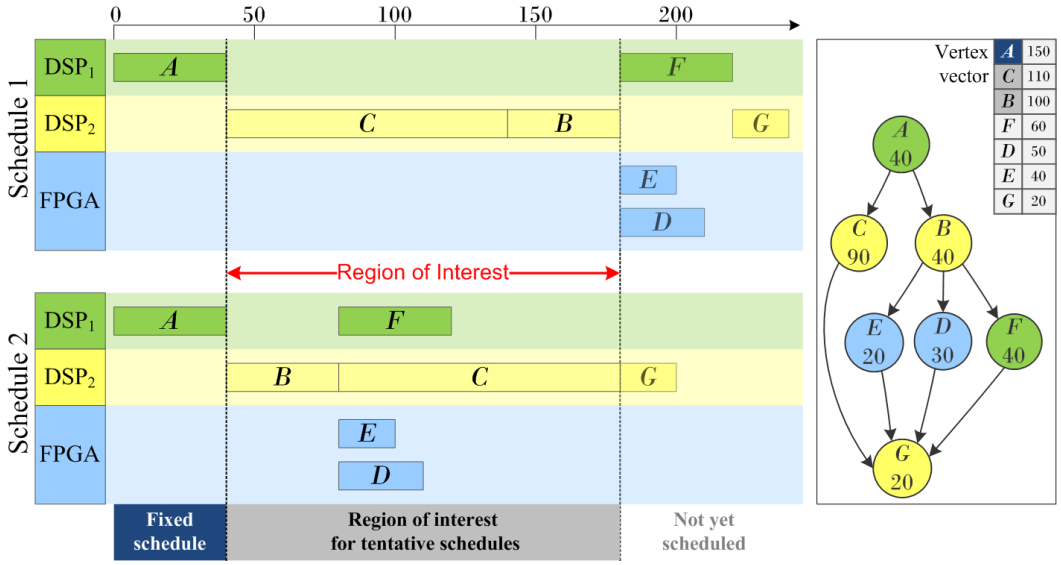


Fig. 4.7: LEP algorithm: two tentative schedules for the decision B first or C first.

The fundamental idea can be characterised by a *divide and conquer* principle. It is assumed that for specific graph structures a concatenation of locally optimal solutions is likely to compose globally a near-optimal solution. The basic mechanism of the LEP is a priority ordered breadth first search (BFS), i.e. vertices ready to be scheduled are inserted in a BFS queue with descending priority levels. Then, LEP tries to schedule the first vertex in this queue, appending its successors, in case they are ready, and finally this vertex is popped

from the queue. But whenever such a scheduling attempt produces a collision, the following sophisticated arbitration is invoked.

In Figure 4.7 a process graph with a current mapping to a 2-DSP plus FPGA platform is depicted. On the right side a vertex vector is given, ordered according to the priority levels for the current execution times. Process A owns the first position and is scheduled on resource DSP_1, thereafter process B and C are ready to be scheduled. Process C is assigned to its resource. But then process B causes the first collision during this scheduling process. According to HLF, C features the highest priority, but as it can be seen from the left side of this figure, this decision would lead to a worse schedule (upper schedule). Instead, LEP computes tentative *local* schedules for both ready vertices C, B to be scheduled first. The local range is defined by the current region of interest RoI $= [40, 170]$, which is chosen to be the maximum time range covered by the involved vertices for both local schedules. The region marked as 'Not yet scheduled' is not explored at this stage of the LEP and is here only depicted for clarity.

For both tentative schedules the exploitation of the inherent parallelism is measured by summing the execution times of all processes that lie within the RoI. The arbiter decides then for the vertex with the maximum enabled parallelism to be scheduled first. In the illustrated example it decides B to be scheduled first and marks it as locked, then C is removed from its schedule position and inserted again in the BFS queue. The vertices D, E, F are now ready and also inserted in the queue according to their priority. Since none of C, D, E, F causes a collision henceforth, the LEP scheduling proceeds unimpeded down to the exit vertex G.

However, the programmer of the LEP has to deliberate on several implementation details. The HLF ordered vertex vector establishes the backbone of the LEP in more than one aspect. Whenever situations eventuate, in which more than two vertices are ready to be scheduled and would collide on a resource, not all permutations are tentatively scheduled, but only those resulting from the two vertices with the highest priority. Whenever such a collision has been arbitrated, the 'winning' vertex is locked on the schedule, whereas the 'losing' vertex (and all its successors in case they have already been scheduled) is added again to the BFS queue. It is the typical case that only a subset of vertices ever wins a collision and thus receives the locked status. All vertices forced to be reinserted into the BFS queue because their predecessor lost a collision arbitration naturally lose a possible locked status, they might have achieved in a prior competition. This status does only prevent the algorithm to enter infinite loops, in which three or more vertices cause an alternating winning-losing arbitration. Therefore, a strong and desired affinity to the fast and very efficient HLF scheduling is maintained. The introduced run time overhead can be estimated quite accurately.

Listing 4.1: Pseudocode for the LEP scheduling algorithm

```
0 LEP() {
1  createPriorityVector();  // Critical path search to get
2                           // current priorities.
3  pushToBFSQueue(startVertices);  // Add start vertices to
4                                  // BFS queue.
5  while (Queue is not empty) {    // Start BFS.
6
7   if (detectCollision(curV) == true) // COLLISION.
8    winV = tentativeSchedules(V1,V2); // Try both schedules.
9    lockVertex(winV);                 // Lock winning vertex.
10   prependToBFS(loseV);              // Prepend losing
11                                     // vertex to BFS.
12   appendReadySuccsToBFS(winV);      // Push ready success-
13  }                                  // ors of winV in BFS.
14  else {
15   schedule(curV);                   // NO COLLISION.
16   appendReadySuccsToBFS(curV);      // Push ready success-
17  }                                  // ors to BFS.
18  popFromBFSQueue();
19 } // end while (...)
20 } // end LEP()
```

If we set the RoI to be limited to the maximum range spanned by both tentative schedules and assume that the execution times of the involved vertices do not differ by more than one order of magnitude from the average vertex execution time, then both tentative schedules cover at most two ranks of successors of the two involved vertices. In the simple example shown in Figure 4.7 even only one level of successors (D, E, F after process B) causes a tentative local schedule. As we already specified the graphs to have a rather low density $\rho = 1 \dots c|\mathcal{V}|$, with $c \ll |\mathcal{V}|$, we see $\frac{\rho}{2}$ successor edges per vertex. Then, when descending at most two ranks downwards the graph, we introduce computational overhead of scheduling $\frac{\rho}{2} + \left(\frac{\rho}{2}\right)^2$ vertices in every local schedule at most. The asymptotic efficiency can hence be approximated to be $\mathrm{O}((2(\frac{\rho}{2} + \left(\frac{\rho}{2}\right)^2))C_{\mathrm{avg}}) = \mathrm{O}(c^2 C_{avg}) = \mathrm{O}(C_{\mathrm{avg}})$, with C_{avg} being the average number of collisions per scheduling. Since sparse precedence graphs are considered and typically mapping solutions distribute their vertices beneficially among the available resources, the number of collisions is in general much lower than the number of processes $C_{\mathrm{avg}} \ll |\mathcal{V}|$, and for the worst case $C_{\mathrm{avg}} = |\mathcal{V}|$. Hence, the approximate efficiency eventuates to be basically linear $\mathrm{O}(c^2(|\mathcal{E}| + |\mathcal{V}|)) = \mathrm{O}(|\mathcal{E}| + |\mathcal{V}|)$.

To compare the behaviour and performance of the scheduling algorithms, they are applied to a large number of system graphs. For any specific graph a huge number of different mappings

exist, which can be scheduled with the presented techniques. Hence, we simulate the search space traversal of partitioning algorithms in such a way, that a lot of different mappings per graph are randomly generated and scheduled with the three candidates. Thus, we obtain for any new partitioning solution $\mathbf{x}$ the three schedule lengths $f_{\mathrm{T}}^{\mathrm{HLF}}(\mathbf{x})$, $f_{\mathrm{T}}^{\mathrm{ETF}}(\mathbf{x})$, $f_{\mathrm{T}}^{\mathrm{LEP}}(\mathbf{x})$. Furthermore, we calculate the lower bound $\min f_{\mathrm{T}}(\mathbf{x})$ for the schedule length of each visited $\mathbf{x}$. This lower bound is determined by the sum of the process execution times et and data transfer times tt along the critical path. For all visited partitioning solutions $\mathbb{S}_{\mathrm{vis}} \subseteq \mathbb{S}_{\mathrm{f}}$, the specific schedule lengths $f_{\mathrm{T}}(\mathbf{x})$ of one partitioning solution $\mathbf{x} \in \mathbb{S}_{\mathrm{vis}}$ are summed up, as well as the specific lower bounds, $f_{\mathrm{T}}^{\mathrm{LB}}(\mathbf{x}) = \min f_{\mathrm{T}}(\mathbf{x})$. Thus, we obtain for all three algorithms and the lower bound a global sum over all considered schedule lengths, e.g. for LEP:

$$F_{\mathrm{T}}^{\mathrm{LEP}}(\mathbb{S}_{\mathrm{vis}}) = \sum_{\mathbf{x} \in \mathbb{S}_{\mathrm{vis}}} f_{\mathrm{T}}^{\mathrm{LEP}}(\mathbf{x}) \tag{4.5}$$

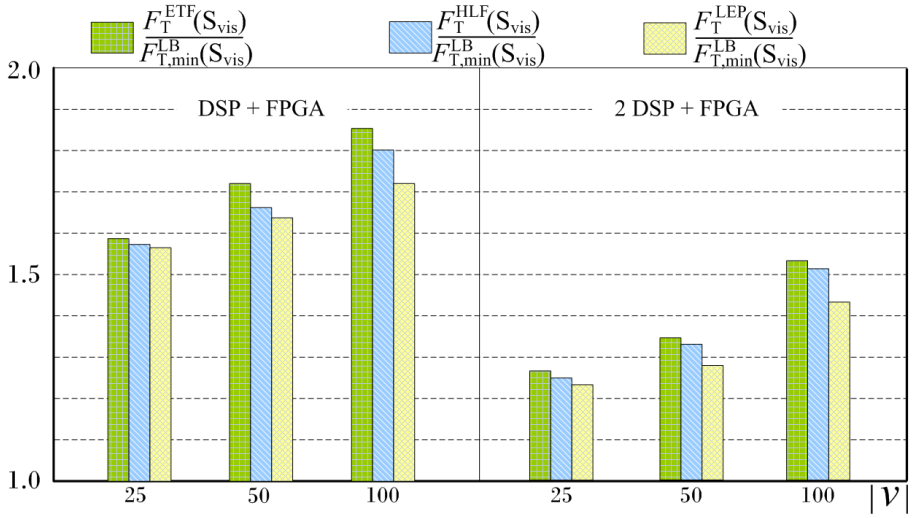


Fig. 4.8: Averaged global schedule lengths normalised to the global lower bound schedule lengths over different mappings and graph sizes.

Figure 4.8 shows a bar chart, in which the x-axis shows different graph sizes for two different platforms described by their main resources. The y-axis shows the global sums over schedule lengths for the three algorithms normalised to the global sum over the lower bound schedule lengths. The results have been averaged over 30 different graphs for each size and for over $10,000$ randomly generated mappings for any single graph. It can be seen that the proposed algorithm creates better results for all depicted graph sizes with a larger margin going towards larger graphs. Smaller graphs with fewer vertices $|\mathcal{V}| \leq 15$ did not show in average any remarkable difference for the chosen schedules. The picture becomes clearer

when we plot the rescheduling algorithms not over their respective sizes but over their timed degree of parallelism γ_{T}, as shown in Figure 4.9. The higher the inherent parallelism of the graph, the more opportunities to exploit this trait exist. Due to the system graph generation procedure the larger graphs feature on average a higher degree of parallelism. Therefore, the growing performance difference with the growing number of vertices has been observed in this setting. It can be stated that the proposed LEP algorithm benefits well from a higher

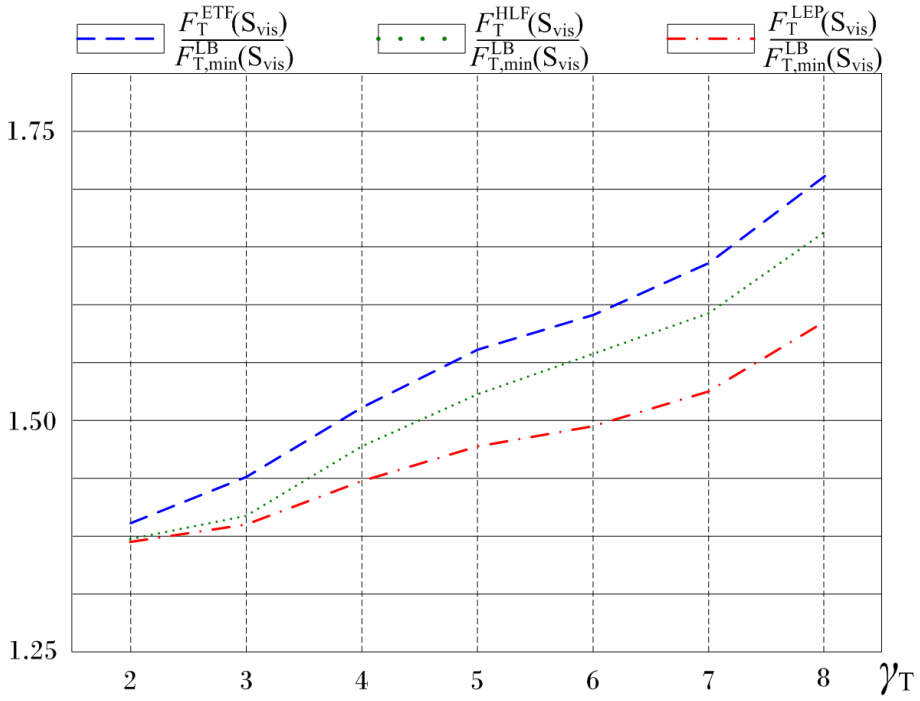


Fig. 4.9: Averaged global schedule lengths normalised to the global lower bound schedule length over degree of parallelism γ_{T}.

degree of parallelism, whereas the HLF algorithm is focused on the current critical path, in other words, its decision is lead by a successor evaluation of the current vertex. The ETF algorithm obtains its decision from evaluating the predecessors of the current vertex, with an HLF fallback mechanism that breaks the ties. None of them considers in any way the vertices that lie in parallel to the current vertex, which is remarkable as parallelism is present both in the graph as also in the architecture. For graphs with $\gamma_{\mathrm{T}} \leq 2.5$ the proposed LEP algorithm did not show significant performance improvements in comparison to HLF.

4.3 Algorithms for System Partitioning

In this section the complete range of all the implemented approaches for OTIE is described. For every technique a brief discussion is given individually and simulation results are listed. Eventually, the chapter is concluded with a consolidated comparison of all algorithms and an interpretation of their appropriateness is given.

4.3.1 Exhaustive Search

As long as the problem size allows for evaluating all possible solutions, an exhaustive search (ES) is applied. In the given scenario the search space for the system partitioning problem grows rampantly. When we assume a_v to be the average number of implementation alternatives for any vertex $v \in \mathcal{V}$, and further assume that any edge's implementation is hence determined, then the size of the search space is approximately $|\mathbb{S}| = a_v^{|\mathcal{V}|}$. Moreover, when we consider the possibilities to be multiplied coming from the alternative data transfer implementations, a similar factor a_e for all edges $e \in |\mathcal{E}|$ comes into play. In such a case we end up with a search space size of $|\mathbb{S}_\mathrm{f}| \approx a_v^{|\mathcal{V}|} a_e^{|\mathcal{E}|}$.
The exhaustive search has been implemented to cover this case, such that any possible solution is iteratively generated and evaluated by its cost. Still manageable search space sizes lie approximately with $|\mathbb{S}_\mathrm{f}| = 2^{30} = 4^{15}$ for 30 vertices in the classical binary partitioning case or for 15 vertices in a more elaborated setting with four implementation alternatives per vertex. The computational run time Θ on a common PC with a dual core AMD Athlon 64 3000+ is considered as manageable, when not exceeding a few minutes. That is due to the size of the utilised graph sets with $100 - 200$ graphs, so that roughly one exhaustive search over a complete graph set can be performed per day.
In the plot depicted by Figure 4.10 it has been tried to illustrate the multi-modality of the search space for a small binary partitioning problem of $|\mathcal{V}| = 20$ processes and $a_v = 2$ implementation alternatives, such that $|\mathbb{S}_\mathrm{f}| = 2^{20} = 1,048,576$. Over the x-axis a fraction of the solution space is plotted, with any solution being indicated by an integer between $160,000$ and $200,000$. The integers are the decimal representation of a Gray coded string corresponding to a single solution: for instance solution number $c_\mathrm{Gray} = 195,000$ would correspond to the Gray string $0011\,1000\,0101\,0110\,0100_G$, in which any bit x_i, $i = 1..20$ is the implementation identifier for a vertex v_i. The y-axis shows the cost Ω of the solutions. The rather complicated encoding for the solutions on the x-axis just ensues from the effort to preserve a minimum of a neighbourhood relations for this plot. Since the encoded solutions have $|\mathcal{V}|$ dimensions, and hence $|\mathcal{V}|$ equally distant neighbours, it is naturally impossible to plot all neighbours in vicinity to each other in only two dimensions. In this plot, a solution with number c_Gray is plotted next to at least *two* of its real neighbours $c_\mathrm{Gray} - 1$ and $c_\mathrm{Gray} + 1$. Of

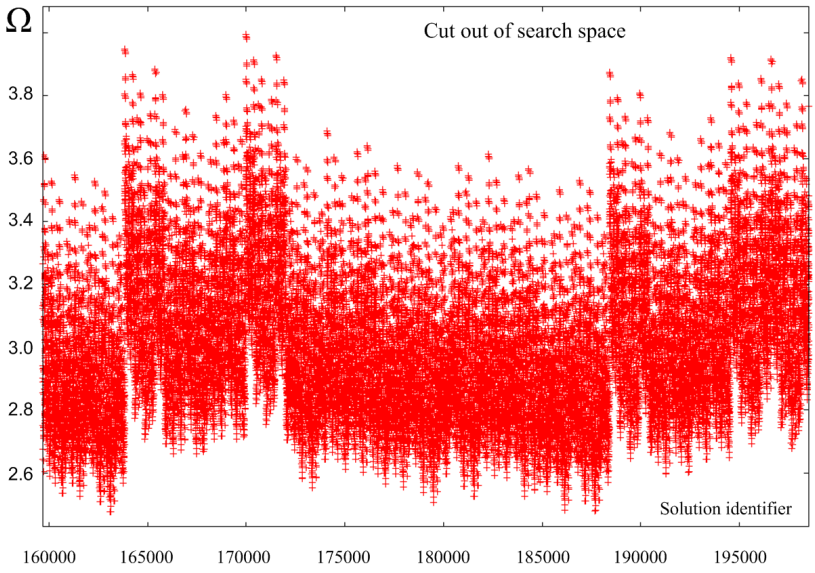


Fig. 4.10: A first impression of the multi-modality of the search space.

course, due to the aforementioned difficulty to interpret the vicinity relations of the solutions here depicted on the x-axis, this figure cannot serve as proof for the multi-modality of the cost function, but rather shall give a first impression of the solution space. In the next section several direct neighbourhood searches will be applied, from whose analysis the multi-modality can be reliably concluded.

Another substantial contribution to the search space size, which could not be taken into account in the exhaustive search, is made by the embedded scheduling problem, whenever system execution time is part of the solution quality. As it had been shown in Section 3.5.3, for any member of the search space many different schedules can be created in general. The exact calculation of the additional degrees of freedom is not possible with a closed formula. Even the very simplified case of sequencing $|\mathcal{V}|$ vertices with precedence constraints, i.e. given as a graph, on a *single* processor can only be approximated by lower and upper bounds. A lower bound is obtained from a rank ordering of the corresponding graph, from which $\forall\ v_r \in \mathcal{V}_r \subseteq \mathcal{V}$ vertices featuring the same r can be sequenced in $|\mathcal{V}_r|!$ ways. Performed for any rank level r, the lower bound to the number of sequencing possibilities is then $\prod_{r=0}^{r_{\max}} |\mathcal{V}_r|!$. It is a lower bound only, because it is in general possible to switch the sequence of vertices with different ranks as well in case they are parallel vertices.
The latter observation enables us to create an upper bound. In order to do so, for any vertex the number of parallel vertices is computed via the transitive closure. The vertices are aligned on an array in rank ascending order and indexed with $i = 1..|\mathcal{V}|$ and for any vertex a

multiplier $m_1(i) = \text{par}(v_i)$ is stored, with $\text{par}(v_i)$ being the number of parallel vertices of v_i. This array is then processed from $j = 1$ to $j = |\mathcal{V}|$ and in each step j those vertices v_k with index $k > j$, which are parallel to v_j, diminish their multiplier by one $m_j(k) = m_{j-1}(k) - 1$. When the last vertex has been processed, the upper bound is then $\prod_{i=1}^{|\mathcal{V}|} m_{|\mathcal{V}|}(i)$. Both upper and lower bound are precise only for vertex orders without any precedence constraints.
It has to be stated, that for sequences with more than one processor present and varying execution times, even the calculation of reasonable upper and lower bounds turns out to be a very complicated task. The most important observation in this respect is the non-existence of an algorithm, which generates all possible sequences in a controlled manner, neither for the simple case nor for the elaborate architecture and process model deployed herein. Hence, this exhaustive search disregards the possibilities introduced by the embedded scheduling. Nonetheless, all three scheduling algorithms presented in Section 4.2 are applied to any visited partitioning solution during the exhaustive search, hence being at least 'exhaustive' in the given set of fast scheduling algorithms.

$\lvert\mathcal{V}\rvert$	$(C_\mathrm{T}, C_\mathrm{A}, C_\mathrm{C})$	$\overline{\Omega}$	Ψ	$\overline{\Theta}$
20	(0.4,0.4,0.5)	2.534	95.1%	301M
	(0.5,0.5,0.5)	1.988	100%	301M
	(0.6,0.6,0.6)	1.659	100%	301M

$(C_\mathrm{T}, C_\mathrm{A}, C_\mathrm{C})$: Constraint ratios (3.13), Ω: Cost (3.10), Ψ: Validity (Def. 19), Θ: Run time (Def. 20)

Tab. 4.1: Results obtained for exhaustive searches.

Table 4.1 contains the simulation results for the smallest graph sets for different constraints. However, interesting problem sizes lie well beyond those with 20 vertices, and as the ES run time at least doubles with every single vertex added, it is not applicable for interesting sizes. As it will be shown, the more sophisticated techniques, like GA, TS, and RRES, draw very near the optimal values for this problem size in most cases, and hence a meaningful comparison of them is inhibited for $|\mathcal{V}| = 20$. For this purpose these are judged by a comparison of their performance for problem sizes up to $|\mathcal{V}| = 200$.

4.3.2 Gradient Search

Among the very first candidates for any optimisation problem are the gradient based algorithms evaluating the neighbourhood of a given solution. Typically, highly multi-modal functions cannot be optimised with a good quality, since hill-climbing algorithms converge very quickly to the nearest local optimum, and are hence not appropriate for system partitioning. But the results obtained from neighbourhood searches yield very valuable insights to the shape of solution space. Moreover, they are very simple to implement, can serve as initial benchmark provider for more sophisticated approaches, and may be applied as final

stage of the best solution obtained by an optimisation heuristic.

A hill-climbing search chooses one solution from the neighbourhood set and evaluates its quality. If the neighbour solution has the same or a higher quality, the current solution is replaced, if not, the next neighbour is analysed. This procedure iterates as long as another neighbour can be identified that improves the current solution. But the way, in which the neighbourhood is scanned for better solutions, may differ. Classically, three approaches can be distinguished.

- *Random neighbour* (RN) applies iteratively a random local transformation $n_I(\mathbf{x})$ on solution $\mathbf{x}$, accepts any better solutions immediately, and terminates when every neighbour or a specific number of neighbours has been visited without improvement.

- *Next neighbour* (NN) applies a procedure, which scans the neighbourhood in a deterministic way and chooses the first neighbour that does not decrease the quality of the current solution. It terminates when the complete neighbourhood has been scanned without improvement.

- *Best neighbour* (BN) performs in any iteration a complete neighbourhood scan and chooses that neighbour, which yields the best improvement. When a better solution cannot be found, the search terminates.

Table 4.2 lists the results we obtain from these rather trivial approaches (best values are in bold font). The BNS search delivers the lowest cost values, but with a smaller margin going to larger graphs accompanied by rampantly increasing run time. RNS and NNS searches bare a very similar performance, becoming more and more competitive to BNS for larger graphs, and the run time difference becomes more and more significant. The most important fact to be remembered is the rather large standard deviation σ of the obtained solutions due to the well-known sensitivity to the randomly generated initial solution. Quality variations for different runs of up to 30% are not rare, since the underlying cost function shows an extreme multi-modal shape. This observation follows from the number of different solutions obtained from consecutive runs starting from random initial solutions. This approach returns for consecutive 100 runs of the smallest problem $|\mathcal{V}| = 20$ between 20 and 40 different solutions. For the medium sized problem $|\mathcal{V}| = 50$ the algorithm returns persistently 200 different solutions out of 200 runs. Of course, a multiple start approach provides a performance boost compared to single start, but the basic problem lies in the vast number of local optima for

$\|\mathcal{V}\|$	$(C_\mathrm{T}, C_\mathrm{A}, C_\mathrm{C})$	Neighbour	$\overline{\Omega}$	$\overline{\sigma}$	Ψ	$\overline{\Theta}$
20	(0.4,0.4,0.5)	Random	3.026	0.0338	16.1%	71k
		Next	3.026	0.0330	16.9%	**65k**
		Best	**2.936**	0.0314	**22%**	147k
	(0.5,0.5,0.5)	Random	2.341	0.0211	84.7%	66k
		Next	2.338	0.0221	83.9%	**60k**
		Best	**2.269**	0.0194	**89.2%**	153k
50	(0.4,0.4,0.5)	Random	**3.018**	0.0235	**16.6%**	167k
		Next	3.043	0.0231	13.9%	**145k**
		Best	3.044	0.0250	13.2%	606k
	(0.5,0.5,0.5)	Random	2.417	0.0163	99.1%	163k
		Next	2.424	0.0164	99.1%	142k
		Best	**2.389**	0.0161	**98.8%**	621k
100	(0.4,0.4,0.5)	Random	**2.899**	0.0160	**39.1%**	2.47M
		Next	2.921	0.0157	30.5%	**1.95M**
		Best	2.949	0.0178	18.2%	13.8M
	(0.5,0.5,0.5)	Random	2.404	0.0132	100%	2.43M
		Next	2.412	0.0133	100%	**1.93M**
		Best	**2.387**	0.0132	100%	12.6M

$(C_\mathrm{T}, C_\mathrm{A}, C_\mathrm{C})$: Constraint ratios (3.13), Ω: Cost (3.10), σ: Standard deviation, Ψ: Validity (Def. 19), Θ: Run time (Def. 20)

Tab. 4.2: Results obtained for neighbourhood searches.

realistic problem sizes. A multi start approach returns steadily medium quality output and does not learn from the traits of the few high quality solutions it produces. The heuristic algorithms in the following sections try to circumvent this drawback.

4.3.3 Global Criticality/Local Phase (GCLP) Algorithm

One of the leading research groups to address the difficulties in modern system design established the Ptolemy Project (1991 - now) at the University of California, Berkeley [98]. The Global Criticality/Local Phase algorithm has been integrated into Ptolemy in 1995 [77]. In the following years the authors enhanced this method to solve the *extended partitioning problem* [78], which incorporates the existence of several implementation alternatives, or *bins* in their terms, for both hardware (HW) and software (SW). Due to its fine reputation to be a fast technique, i.e. with a good efficiency of $\mathrm{O}(|\mathcal{V}|^2)$, while yielding reasonably good results compared to *Integer Linear Programming* [78], the Open Tool Integration Environment (OTIE) has been enriched with a version of the GCLP algorithm. The analysis and evaluation of the original algorithm disclosed several possibilities to save computation time and to improve quality. The contribution of this thesis comprises a thorough analysis of the GCLP algorithm and the introduction of several modifications to increase the performance of this approach with respect to the solution quality, the computation time and the probability of valid results [91, Knerr et al.].

In the case of the binary (SW, HW) partitioning problem for GCLP, the characteristic values of the processes v_i being part of the implementation alternatives A^i_{HW} and A^i_{SW} compose a three-tuple: $A^i_{r,j} = (\text{execution time, code size, gate count})$. The mapping of the task graph to the given architecture in Figure 4.11b is performed by the GCLP algorithm with the objective to meet constraints for time, area, and code size. The platform model features a general purpose processor, which allows for sequential execution of the assigned processes, and an FPGA or a set of ASICs for a custom data path, which allows for concurrent execution of the assigned processes. A model for HW to SW communication via shared memory is provided, whereas HW to HW and SW to SW communication is neglected. The following paragraphs present a short discussion of the basic concepts of the GCLP approach. For complete detail, please refer to the Kalavade's dissertation [77].

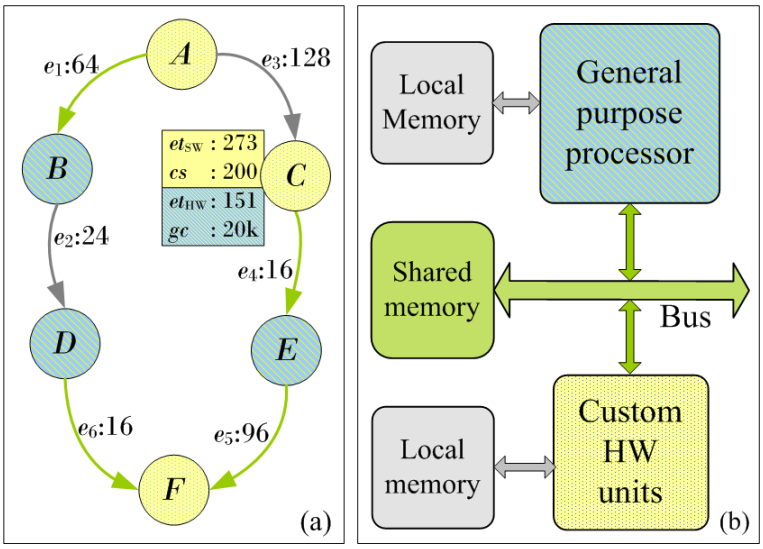


Fig. 4.11: (a) Process graph, annotated with characteristic values. (b) Typical platform model.

Essentially the GCLP is a greedy approach, which visits every vertex exactly once, and decides where to map it based on two different values: the *Global Criticality* (GC) measure and the *Local Phase* (LP) measure. The GC value is a *global* look-ahead measure that estimates whether time, code size or area is most critical at the current stage of the algorithm and then decides which of these targets shall be minimised. The LP value is calculated for every single process before the main algorithm starts and is based on intrinsic properties that represent the individual mapping preferences of this process. For instance, when a specific process prefers an implementation in SW, because of its very large bit level instruction mix, the LP value reflects this preference, or when a process stands out by its extraordinary HW size

and a rather small SW execution time, then the LP value takes this into account. By the superposition of the global GC value and the local LP value the greediness of the approach is moderated and a balanced mapping, which meets all constraints, shall be ensured.

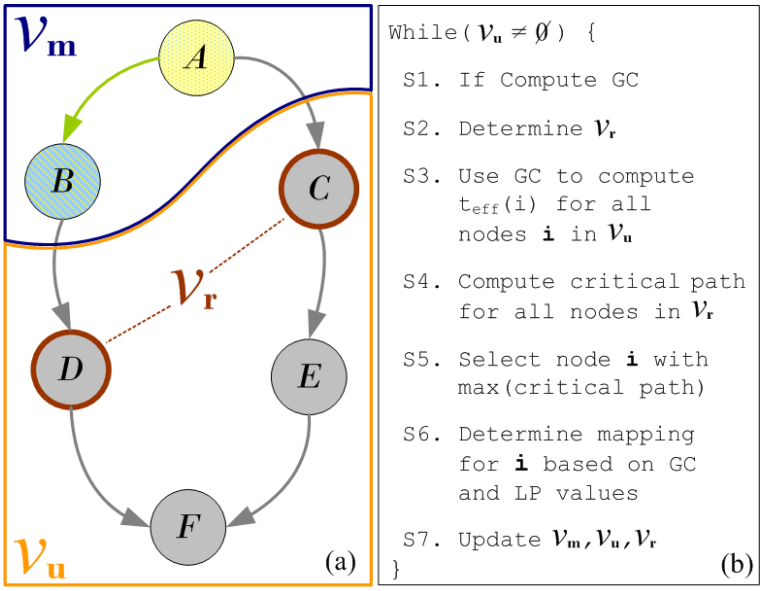


Fig. 4.12: (a) Process graph at a distinct stage of the GCLP algorithm. (b) Pseudo code for a single GCLP iteration.

In Figure 4.12a the process graph is depicted and in Figure 4.12b pseudo code of one GCLP iteration is listed. The upper two vertices have been already **m**apped ($\mathcal{V}_\mathrm{m} = \{A, B\}$), all others are still **u**nmapped ($\mathcal{V}_\mathrm{u} = \{C, D, E, F\}$), of which two are **r**eady ($\mathcal{V}_\mathrm{r} = \{C, D\}$) to be mapped next. In step S1 the current GC value is calculated. Within S1 a provisional yet *complete* mapping is performed such that the time constraint is surely met. The GC value is then calculated based on this preliminary mapping and is normalised to lie in the interval $[0, 1]$ ($0 =$ lowest time criticality, $1 =$ highest time criticality). In step S2 the ready processes $\mathcal{V}_\mathrm{r} = \{C, D\}$ are determined. The steps S3 and S4 shall decide which of both vertices C, D will be mapped next. In step S3, an *effective* execution time $t_\mathrm{eff} = GC\ et_\mathrm{HW} + (1 - GC)\ et_\mathrm{SW}$ is assigned to all yet unmapped vertices. In step S4, t_eff serves as the base for a critical path search from every vertex in $\mathcal{V}_\mathrm{r}$ to the exit process f. In step S5, the vertex with the maximum critical path value is selected to be mapped next. In step S6 the final mapping of this vertex is performed based on the superposition of the *global* GC value and the *local* LP value. In step S7, all sets, lists and intermediate values are updated. These seven steps are repeated until all vertices have been finally mapped ($\mathcal{V}_\mathrm{u} = \emptyset$).

In the following sections substantial modifications applied to this algorithm are introduced and simulation results for all GCLP versions will be compared to each other.

GCLP Modification 1 - Revision of Step 3 and Step 4

Consider the steps S3 and S4 in the listing in Figure 4.12b. Note, that their single purpose is the decision *which* process is going to be mapped next, neither *where* it is going to be mapped, nor *when* exactly it will be scheduled when *all* processes have been finally mapped. For all the graph sets, a positive impact on the solution quality by these two steps could not be observed. A comparison to a random selection of the process from $\mathcal{V}_\mathrm{r}$, which should be mapped next, did not show any significant difference, as Table 4.3 indicates. The reason for this result is two-fold: the calculation of the critical paths in S4 is based on *effective* execution times. The critical path searches yield correct values for all vertices in $\mathcal{V}_\mathrm{r}$, if and only if $GC = 1$, or in other words in case of a complete HW solution of the remaining vertices, given the HW processor allows for **concurrent** execution of tasks. For a complete SW solution, the critical path calculations lose their relation to the graph completely, since the SW processor is a **sequential** device, and all processes have to run on it consecutively anyway. Thus, for small GC values this calculation does not have significance, and for balanced GC values, the execution times are averaged between et_HW and et_SW and lack precision due to this averaging. Only for large GC values S4 delivers approximately correct results, which is not enough to compensate the imbalance of this mechanism.
To overcome this malfunction we propose two modifications, M1a or M1b:

- M1a: Omit the steps S3 and S4 completely to save run time of about 15%. That is only of interest for very large graphs ($|\mathcal{V}| \geq 200$), in which the run time for each graph becomes a matter of minutes instead of seconds.
- M1b: Calculate the critical path searches for all vertices in $\mathcal{V}_\mathrm{r}$ based on the provisional partitioning just generated in step S1. Recall, that step S1 comprises a full partitioning and scheduling to compute the current GC value and thus represents a precise snapshot of the present partitioning situation: all processes apply either their correct et_SW or et_HW instead of a mixture of both and a full schedule exists also. Hence, the critical path search in S4 returns correct values to determine the vertex in $\mathcal{V}_\mathrm{r}$ that currently lies on the critical path. S3 can be omitted.

Table 4.3 shows the impact for all graph sets on run time, cost, and validity. M1a saves about 15% run time without any degradation of the obtained solutions. Modification M1b improves the result quality by about 1.5% to 2% in cost, *and* reduces the run time, *and* features an almost 3% higher Ψ, as listed.

$\lvert\mathcal{V}\rvert$		$\overline{\Omega}$	$\overline{\sigma}$	Ψ	$\overline{\Theta}$
20	GCLP	2.377	0.0253	88.9%	35k
	M1a	2.385	0.0264	88.9%	**24k**
	M1b	**2.348**	0.0252	**95.6%**	33k
50	GCLP	2.452	0.0163	89.8%	91k
	M1a	2.476	0.0177	90.8%	**71k**
	M1b	**2.400**	0.0162	**96.3%**	89k
100	GCLP	2.413	0.0134	89.0%	210k
	M1a	2.428	0.0138	92.3%	**166k**
	M1b	**2.367**	0.0131	**99.4%**	202k
200	GCLP	2.593	0.0125	85.5%	1.1M
	M1a	2.621	0.0124	87.8%	**810k**
	M1b	**2.468**	0.0123	92.3%	1.1M

Ω: Cost (3.10), σ: Standard deviation,
Ψ: Validity (Def. 19), Θ: Run time (Def. 20)

Tab. 4.3: Impact of proposed modifications M1a and M1b compared with the original GCLP algorithm.

GCLP Modification 2 - Initial Solution

Another substantial gain in performance is possible by a more sophisticated choice of the initial solution. Although the preparation phase of GCLP comprises the individual characterisation of processes with respect to their preferred implementation type, GCLP assumes a complete SW solution as starting point. Neither the constraints given by the designer nor the just calculated *local phase* values affect this assumption in any manner. A strong potential to enhance the quality of the final result without increasing the run time can be put forth. The

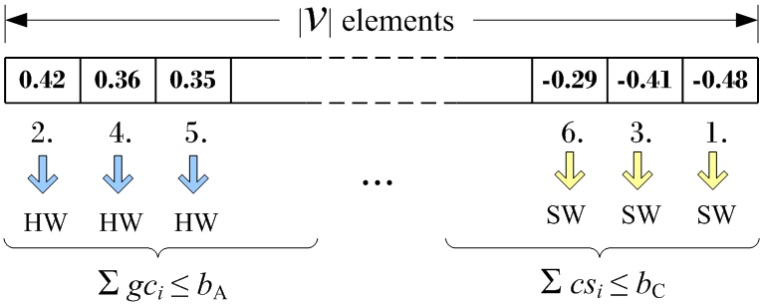


Fig. 4.13: Modification 2 (M2): Constructing the initial solution.

initial configuration for GCLP is a graph, in which every vertex has an LP value in $[-0.5, 0.5]$ indicating whether it is more suited for a SW (-0.5) or a HW (0.5) implementation. The generation of these values is described in detail in the literature [77,78], due to limited space it has to be omitted here. A simple and fast strategy to construct a better initial solution is to build an ordered list of these individual values, which can be achieved very efficiently (on average in $\mathrm{O}(|\mathcal{V}| \log |\mathcal{V}|)$ with the quicksort algorithm).
We process this list simultaneously from both ends, depending on the absolute value of the

contained measure, as depicted in Figure 4.13. We proceed as long as the initially mapped processes do not reach the area limit b_{A} for those mapped to HW or the code size limit b_{C} for those mapped to SW. The remaining processes in the middle of this list are flagged to be considered preferentially in step S1 of the GCLP algorithm. The efficiency of this operation is $\mathrm{O}(|\mathcal{V}|)$. The computational overhead is smaller than 0.3% and was only observable during the simulations for the largest graphs ($|\mathcal{V}| \geq 200$) averaged over 180 graphs. Table 4.4 con-

$\|\mathcal{V}\|$		$\overline{\Omega}$	Ψ
20	GCLP	2.377	96.9%
	M2	**2.353**	**98.1%**
50	GCLP	2.452	97.8%
	M2	**2.420**	**97.2%**
100	GCLP	2.413	96.0%
	M2	**2.368**	**99.6%**
200	GCLP	2.593	91.3%
	M2	**2.537**	**94.9%**

Ω: Cost (3.10), Ψ: Validity (Def. 19)

Tab. 4.4: Impact on cost and validity percentage of M2.

tains the results obtained while applying this modification (M2) to the graph sets compared to the original algorithm. Another 2% quality gain and a higher yield in valid solutions can be achieved.

GCLP Modification 3 - Precocious Breaks

A third modification (M3) is the insertion of precocious breaks as soon as all constraints are met. Although the design of the GCLP algorithm is focused on low run time, a mechanism to stop the algorithm as soon as possible is surprisingly not provided. As stated before, step S1 generates a full partitioning solution, even though being provisional, it makes perfect sense to evaluate this solution as well. The partitioning with the lowest cost seen is stored and when the constraints happen to be met, the algorithm stops. In the case of rather loose constraints the run time drops dramatically. When the constraints are rather strict, so that the original algorithm would finalise returning an *invalid* solution, the run time stays exactly the same, with a possibly better cost obtained by one of the provisional mappings. When the constraints are strict, but the original algorithm would finalise with a *valid* solution, the run time will drop very likely by at least a small margin. For a profound understanding of the last case, it is mandatory to demonstrate the functionality of S1 in detail.
As stated before, in step S1 it is always assumed that all processes in $\mathcal{V}_{\mathrm{u}}$ are implemented in SW. Then processes are tentatively moved to HW until the time constraint b_{T} is met. Of course this mechanism is sensitive to the chosen order in which the processes in $\mathcal{V}_{\mathrm{u}}$ are moved. The GCLP designers proposed a priority list for the processes ordered by their best *gain* in time measured by the quotient $et_{\mathrm{SW}}/et_{\mathrm{HW}}$. A large *gain* means that its mapping from SW

to HW results very likely in a large reduction of the system's execution time. Consider a

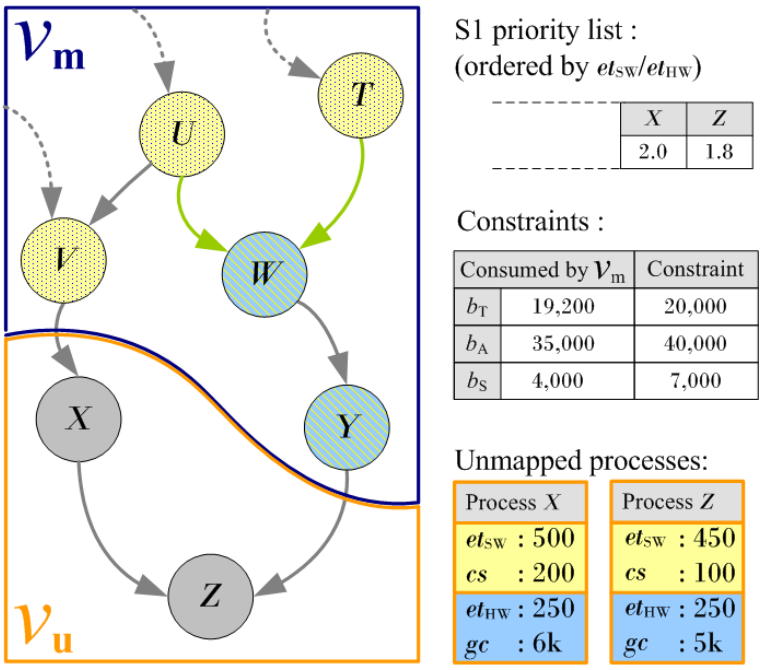


Fig. 4.14: Modification 3(M3): Precocious breaks.

situation, which adheres to the mentioned case: a *valid* solution exists, that would be found by the original algorithm and rather strict constraints prevented a precocious break up to the current stage of the algorithm. In Figure 4.14 the tail of a graph is depicted with the exit vertex Z. The preceding iteration, in which process Y has been finally mapped, did not break precociously, i.e. not all constraints had been fulfilled in S1 of the last iteration. Since S1 ensures a provisional partitioning, in which the constraint b_T is met, only b_A and/or b_S could have been exceeded. But this is only possible when the order of the priority list, that guides the tentative mapping, in S1 does *not* cause a valid mapping. On the top right of Figure 4.14 the entries for X and Z in the priority list are shown. Hence, S1 does always map X to HW at first, detects that b_T is met and thus leaves Z in SW. In this example b_A is then exceeded by this combination $(35,000 + 6,000 \geq 40,000)$, so a precocious break is not possible. The following final mapping of X chooses a SW implementation, since b_A is exceeded whereas b_T is met and proceeds the very last process Z in the graph.

This scenario demonstrates the only case in which the modified version is not capable of finishing at least a short time earlier than the original algorithm. In *all* other scenarios, when the tail of the priority list matches a *valid* partitioning solution, a precocious break will occur. Table 4.5 lists the impacts of this last modification on the run time for loose, medium, and strict constraints. The run time improvement for large graphs and loose constraints is

$\|\mathcal{V}\|$	$(C_\mathrm{T}, C_\mathrm{A}, C_\mathrm{C})$		$\overline{\Omega}$	$\overline{\Theta}$
100	(0.4,0.4,0.5)	GCLP	**2.933**	1.1M
		M3	2.939	**1.0M**
	(0.5,0.5,0.5)	GCLP	**2.413**	1.1M
		M3	2.462	**968k**
	(0.6,0.6,0.6)	GCLP	**1.908**	1.1M
		M3	1.970	**821k**
200	(0.4,0.4,0.5)	GCLP	**3.211**	5.3M
		M3	3.332	**5.1M**
	(0.5,0.5,0.5)	GCLP	**2.593**	5.3M
		M3	2.655	**4.3M**
	(0.6,0.6,0.6)	GCLP	**2.051**	5.3M
		M3	2.126	**3.9M**

$(C_\mathrm{T}, C_\mathrm{A}, C_\mathrm{C})$: Constraint ratios (3.13), Ω: Cost (3.10)
Θ: Run time (Def. 20)

Tab. 4.5: Effect of modification M3 on the run time.

substantial with up to 25%. The validity percentage is even improved by about 0.5% for larger graphs ($|\mathcal{V}| \geq 100$), as there are rare occasions, when a provisional mapping is detected to be valid and the modified algorithm ends precociously, whereas the original algorithm would yield an invalid result with one of the constraints narrowly missed.
It has to be mentioned that the third modification evidently leads to a degradation of the quality for the *valid* partitioning solutions, as a precocious break is surely valid but will often have a higher cost than an algorithm with this option disabled, whereas the quality of *invalid* solutions will increase, as the provisional mappings are considered additionally.

Results for Combined Modifications

Eventually, two promising combinations of the proposed modifications are build. The first combination incorporates M1a and M3 to obtain an algorithm with a substantially lower run time Θ, a slightly better validity percentage Ψ, and minor degradations of the solutions cost Ω. The second combination incorporates M1b and M2 to obtain an algorithm, which concentrates on cost improvements and higher validity percentages nearly without affecting the run time. The subsequent Table 4.6 presents a comparison with the original algorithm for all graph sets and different sets of constraints: Naturally, large graphs with rather loose constraints lead to a dramatic drop in computation time of up to 27%. Additionally combination M1a+M3 causes a measurable increase in the validity percentage of about 1%. These improvements are paid by a rise in averaged cost $\overline{\Omega}$ of about 3-4%. The second combination M1b+M2 is a more balanced improvement. The predominant part is the boost in validity percentage Ψ, with about 4% most noticeable for strict constraints on smaller graphs. This performance is accompanied by a quality improvement of up to 3%, while the run time even drops slightly.

$\lvert\mathcal{V}\rvert$	$(C_\mathrm{T}, C_\mathrm{A}, C_\mathrm{C})$		$\overline{\Omega}$	Ψ	$\overline{\Theta}$
50	(0.4,0.4,0.5)	GCLP	3.096	13.3%	91k
		M1a+M3	3.101	13.9%	**90k**
		M1b+M2	**3.021**	**19.4%**	89k
	(0.5,0.5,0.5)	GCLP	2.452	89.8%	91k
		M1a+M3	2.482	**90.2%**	**82k**
		M1b+M2	**2.394**	**94.5%**	89k
	(0.6,0.6,0.6)	GCLP	1.920	100%	91k
		M1a+M3	2.011	100%	**66k**
		M1b+M2	**1.873**	100%	90k
100	(0.4,0.4,0.5)	GCLP	2.933	57.6%	1.1M
		M1a+M3	2.937	59.1%	**1.0M**
		M1b+M2	**2.898**	**88.0%**	1.0M
	(0.5,0.5,0.5)	GCLP	2.413	89.0%	1.1M
		M1a+M3	2.451	**90.2%**	**923k**
		M1b+M2	**2.367**	**95.3%**	1.1M
	(0.6,0.6,0.6)	GCLP	1.908	100%	1.1M
		M1a+M3	1.988	100%	**768k**
		M1b+M2	**1.865**	100%	1.1M
200	(0.4,0.4,0.5)	GCLP	3.111	11.6%	5.3M
		M1a+M3	3.137	12.3%	**5.1M**
		M1b+M2	**3.034**	**18.2%**	5.2M
	(0.5,0.5,0.5)	GCLP	2.493	88.8%	5.3M
		M1a+M3	2.544	89.8%	**4.3M**
		M1b+M2	**2.420**	94%	5.3M
	(0.6,0.6,0.6)	GCLP	1.951	100%	5.3M
		M1a+M3	2.030	100%	**3.7M**
		M1b+M2	**1.901**	100%	5.3M

$(C_\mathrm{T}, C_\mathrm{A}, C_\mathrm{C})$: Constraint ratios (3.13), Ω: Cost (3.10)
Ψ: Validity (Def. 19), Θ: Run time (Def. 20)

Tab. 4.6: Impact of combined modifications M1a+M3 and M1b+M2 GCLP performance.

Both combinations cover different areas of problem instances, while both prove to be better than the original algorithm in these areas. The first combination M1a+M3 is recommended for problem instances with very large graphs ($|\mathcal{V}| \geq 200$) or a graph set containing a large number of different graphs, for which valid results shall be produced, as its benefits lie predominantly in a run time reduction. The second combination M1b+M2 can simply replace the implementation of the original GCLP algorithm, as it yields better results in every aspect with the largest margin in increasing on average the number of valid results.

Finally, it has to be clarified that the GCLP approach was not designed and is not capable to compete with time-consuming approaches based on genetic algorithms, tabu search, simulated annealing or even integer linear programming, when the aim is to find a near-optimal solution. The run time of these approaches is $10^3 - 10^4$ times higher [78, 142], while tens of thousands of solutions are generated and a cost reduction of up to 25% can be observed.

4.3.4 Simulated Annealing

Simulated annealing (SA) is a generic probabilistic heuristic approach for the global optimisation problem, namely locating a good approximation to the global optimum of a given function in a large search space [82]. Over the years SA gained a lot of popularity among research groups to tackle many combinatorial optimisation problems. Due to its comprehensive structure, and its easy-to-use, it is one of the first candidates to be implemented, when the solution space is large and the problem is analytically intractable. As in many works before, SA will mainly serve as a typical benchmark provider for the more elaborate approaches [7, 38, 142].The following paragraph summarises the basics of the simulated annealing mechanism.

Simulated annealing is essentially a simple modification to a hill-climbing neighbourhood search. It traverses iteratively through the search space and accepts better solutions. But unlike a simple gradient search algorithm, which always rejects moves that lead to worse solutions, its control structure avoids to be trapped in local optima by also accepting worse solutions with a certain probability. This probability decreases as the algorithms proceeds and eventually the algorithm converges. As the roots of SA lie in metallurgy, the classical terms energy E and temperature T are common in algorithmic topics. Note, that energy E is a synonym for cost, because genuinely in metallurgy SA tries to decrease the energy of an alloy on a molecular level [106] as we try to decrease cost in this optimisation problem. The probability of accepting the move of a vertex to another implementation type that creates a new (worse) kth solution $\mathbf{x}_k$, is given by the function $P(\delta E, T_i) = \exp(-\delta E/(k_\mathrm{B} T_i))$ of the energy/cost difference $\delta E = E(\mathbf{x}_n) - E(\mathbf{x}_\mathrm{best})$ and of a global time-varying parameter called the temperature T_i, which is in metallurgy multiplied with the Boltzmann constant k_B to yield energy. As stated before, P is defined non-zero, if δE is positive. While the optimisation proceeds, $T_i \longrightarrow 0$, so that $P(\delta E, T_i) \longrightarrow 0$ for worse solutions. Obviously, a crucial parameter is T_i, since it defines the annealing strategy, i.e. the decreasing probability of accepting worse solutions. Generally, the initial temperature has to be large enough to allow *all* transitions to be accepted, because the physical analogy is the heating of the alloy until all particles are randomly arranged. The corresponding candidate is the maximal difference in cost between any neighbouring solutions. However, this value is typically costly to compute, thus a pragmatic approach is to compute T_init large enough, that the probability to accept any neighbouring solution, $\mathbf{x} \in n_I(\mathbf{x}_\mathrm{init})$, is approximately 1. To do so for the first $|n_I(\mathbf{x}_\mathrm{init})| = a_v|\mathcal{V}|$ iterations is sufficient to obtain a reasonable initial temperature, which adheres to the heating analogy [68]. Wiangtong et al. [142] elaborated on different annealing strategies and identified the classical geometric cooling to be most appropriate. Geometric cooling means that every u iterations the temperature will be updated as follows: $T_{i+1} = \vartheta\, T_i$, with $\vartheta = 0.9 \ldots 0.99$, and $u = f(|n_I(\mathbf{x}_\mathrm{init})|) = a_v|\mathcal{V}|$. By decreasing values of

T_i the algorithm is forced to converge by degenerating into a gradient search. The size of the problem instance I expressed by the neighbourhood operation $|n_I(\mathbf{x}_{\text{init}})|$ is taken into account by the parameter u.

$\|\mathcal{V}\|$	$(C_{\text{T}}, C_{\text{A}}, C_{\text{C}})$	ϑ	$\overline{\Omega}$	$\overline{\sigma}$	Ψ	$\overline{\Theta}$
20	(0.4,0.4,0.5)	0.99	2.735	0.0185	44.9%	692k
		0.95	**2.661**	0.0148	55.5%	720k
		0.90	2.678	0.0154	**55.8%**	**611k**
	(0.5,0.5,0.5)	0.99	2.187	0.0105	99.2%	633k
		0.95	**2.146**	0.0113	**100%**	796k
		0.90	2.152	0.0120	99.8%	**587k**
50	(0.4,0.4,0.5)	0.95	2.964	0.0141	36.2%	40.1M
		0.90	3.003	0.0139	29.5%	31.9M
	(0.5,0.5,0.5)	0.95	**2.317**	0.0129	100%	61.1M
		0.90	2.335	0.0134	100%	**41.9M**
100	(0.4,0.4,0.5)	0.90	3.127	0.0243	6.2%	312M
	(0.5,0.5,0.5)		2.270	0.0115	100%	281M

$(C_{\text{T}}, C_{\text{A}}, C_{\text{C}})$: Constraint ratios (3.13), ϑ: Cooling factor, Ω: Cost (3.10), σ: Standard deviation, Ψ: Validity (Def. 19), Θ: Run time (Def. 20)

Tab. 4.7: Results obtained for simulated annealing.

Its rather large run time is a significant disadvantage compared to the following implementations of tabu search and genetic algorithm. However, its advantage lies in the reliable generation of good quality solutions, if the problem instance is not large $|\mathcal{V}| \leq 50$ or design time is not important. Simulated annealing has been tested for different values of the geometric cooling factors $\vartheta = 0.9, 0.95, 0.99$ and temperature level updates for every $u = |\mathcal{V}|a_v$. It can immediately stated that for medium $|\mathcal{V}| = 50$ and large graphs $|\mathcal{V}| = 100$, these values had to be limited to $\vartheta = 0.95$ for medium and $\vartheta = 0.90$ for large graphs due to the extraordinary run time of often several hours for one problem instance. Table 4.7 gives an impression of cost values $\overline{\Omega}$ averaged over all instances of all graphs and the standard deviations $\overline{\sigma}$ averaged over all graphs. The second column yields two different constraint sets $(C_{\text{T}}, C_{\text{A}}, C_{\text{C}})$ for the three following objectives: area C_{A} on the FPGA, code size C_{C} on the DSP, and the schedule length C_{T} of a complete execution.

4.3.5 Tabu Search

The tabu search (TS) combines the concept of a neighbourhood search method with a memory in which the search history of the algorithm is tracked [48]. From a current solution $\mathbf{x}$ a subset of the neighbourhood $\mathbb{S}_{N,\text{TS}} \subseteq n_I(\mathbf{x})$ is generated at any iteration of the algorithm, of which the solution with the best cost serves as the base for the next iteration. Inspired by artificial intelligence techniques this neighbourhood search incorporates a short-term memory of recent moves through the search space and stigmatises these moves as *tabu*. A move

with the *tabu* status is prohibited thus forcing the algorithm to traverse different regions of the search space. After a certain number of steps, these moves are destigmatised and these regions of the search space are allowed to be visited anew. An *aspiration* criterion may be provided that can override the *tabu* status of a move, for instance in case the *tabu* move creates a better solution than the best one seen so far. An additional long term memory keeps track on the number of visits to certain regions of the search space and may even store which regions feature high quality solutions more frequently than others. Based on the long term memory, two complementary mechanisms are often applied: *diversification*, which influences the algorithm to broaden the spectrum of visited solutions, and *intensification*, which influences the algorithm to reward regions that more frequently produce high quality solutions.

The work of Wiangtong [142] introduces a tabu search with both *diversification* and *in-*

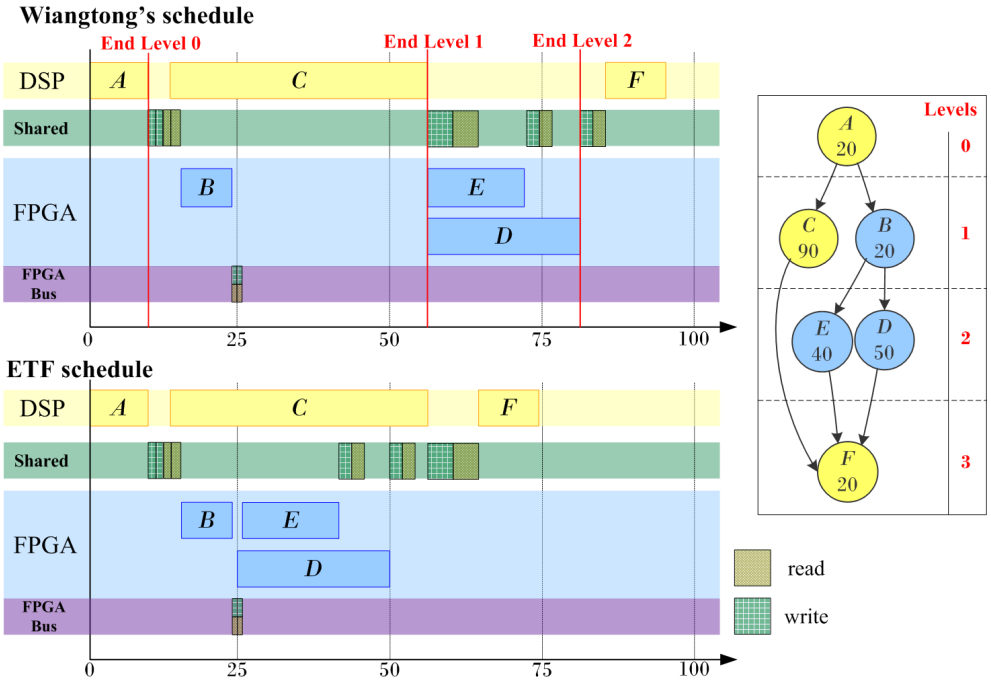


Fig. 4.15: Wiangtong's scheduling compared to a ETF.

tensification strategies called *penalty reward* and demonstrates its superior performance over simulated annealing and genetic algorithm implementations. In the scope of this thesis this very promising technique has been implemented to be incorporated into OTIE and to benchmark against other approaches. An additional modification of the original work has taken place, since the scheduling technique chosen by Wiangtong revealed inferior performance in comparison with HLF and LEP. Basically the vertices are annotated with their precedence level (or rank) as in the upper part of Figure 4.15. For all vertices on a distinct rank, the

scheduler first extracts those mapped to a sequential device. Those causing a collision are ordered according to decreasing utilisation of the shared communication resource. In other words two processes, whose execution would collide on a DSP core, are scheduled such, that the process comes first, which uses the shared bus most. When all processes for sequential resources of the current rank have been scheduled, all remaining processes to be mapped to concurrent resources are considered. These processes are then mapped onto the concurrent device and whenever a collision on the shared communication resource occurs, the process with the higher execution time is allowed to communicate first.
This scheduling technique has a severe drawback: first and foremost process execution is always delayed until all processes on the previous precedence level have finished their execution. Consider the upper schedule in Figure 4.15 resulting for the simple graph on the right. In fact, the precedence or rank levels as they are defined in the Wiangtong's PhD thesis [141] do not reflect the individual precedence constraints of a vertex. This scheduling technique results in a consecution of maximum execution and transfer times for any precedence level, inevitably delaying processes unnecessarily, as they are forced to wait not only for their own predecessors but also for the predecessors of all vertices with the same rank. The collision arbitration within a single precedence level according to the bus utilisation becomes then irrelevant.

In Table 4.8 the results are listed for the original TS algorithm of Wiangtong ($\mathrm{TS}_{\mathrm{org}}$) and the TS algorithm enhanced by the proposed LEP scheduling ($\mathrm{TS}_{\mathrm{LEP}}$). The second column again

$\lvert\mathcal{V}\rvert$	$(C_{\mathrm{T}}, C_{\mathrm{A}}, C_{\mathrm{C}})$		$\overline{\Omega}$	$\overline{\sigma}$	Ψ	$\overline{\Theta}$
20	(0.4,0.4,0.5)	$\mathrm{TS}_{\mathrm{org}}$	2.631	0.0171	83.2%	**213k**
		$\mathrm{TS}_{\mathrm{LEP}}$	**2.589**	0.0176	**84.4%**	270k
	(0.5,0.5,0.5)	$\mathrm{TS}_{\mathrm{org}}$	2.145	0.0129	100%	**199k**
		$\mathrm{TS}_{\mathrm{LEP}}$	**2.108**	0.0111	100%	249k
50	(0.4,0.4,0.5)	$\mathrm{TS}_{\mathrm{org}}$	2.919	0.0174	62.8%	**611k**
		$\mathrm{TS}_{\mathrm{LEP}}$	**2.849**	0.0173	**67.7%**	743k
	(0.5,0.5,0.5)	$\mathrm{TS}_{\mathrm{org}}$	2.333	0.0114	100%	**530k**
		$\mathrm{TS}_{\mathrm{LEP}}$	**2.257**	0.0126	100%	616k
100	(0.4,0.4,0.5)	$\mathrm{TS}_{\mathrm{org}}$	2.862	0.0136	93.5%	**20.3M**
		$\mathrm{TS}_{\mathrm{LEP}}$	**2.779**	0.0139	98.4%	25.9M
	(0.5,0.5,0.5)	$\mathrm{TS}_{\mathrm{org}}$	2.336	0.0111	100%	**26.8M**
		$\mathrm{TS}_{\mathrm{LEP}}$	**2.231**	0.0104	100%	34.4M

$(C_{\mathrm{T}}, C_{\mathrm{A}}, C_{\mathrm{C}})$: Constraint ratios (3.13), Ω: Cost (3.10), σ: Standard deviation, Ψ: Validity (Def. 19), Θ: Run time (Def. 20)

Tab. 4.8: Results obtained for tabu search, original and with LEP scheduling.

yields the aforementioned constraint ratio set $(C_{\mathrm{T}}, C_{\mathrm{A}}, C_{\mathrm{C}})$ for the three following objectives: area C_{A} on the FPGA, code size C_{C} on the DSP, and the schedule length C_{T} of a complete execution.
In fact the tabu search is improved by the better suited scheduling technique up to 2% for larger graphs and feature an up to 4% higher validity value for strict constraints. Note, that

herein the scheduling contributes only to a third to the overall cost besides area and code size, hence the improvement is significant. Due to the added complexity of the LEP scheduling the run time increases but with an admissible growth rate. For further comparisons, the original Wiangtong scheduling is always replaced by either HLF, ETF, or LEP according to what is chosen for competing algorithms.
The complete parameter set to obtain results with a good quality are chosen according to Wiangtong's original work: size of neighbourhood subset $|\mathbb{S}_{\mathrm{N,TS}}| = 0.5\sqrt{|\mathcal{V}|}\ldots\sqrt{|\mathcal{V}|}$, tabu list length $L_{\mathrm{tabu}} = 7\ldots20$ but typically such that $|\mathbb{S}_{\mathrm{N,TS}}|L_{\mathrm{tabu}} \leq 0.5|\mathcal{V}|$, region size for intensification and diversification strategy $s_{\mathrm{reg}} \leq 100,000$, which corresponds to 16 elements of $\mathbf{x}$ in the binary case ($s_{\mathrm{reg}} = 2^{16}$) and only six elements of $\mathbf{x}$, when there are six implementation alternatives per element ($s_{\mathrm{reg}} = 6^6$).

4.3.6 Genetic Algorithm

This section briefly introduces fundamental terms, sketches how the GA concept is typically applied to system partitioning, reveals where of the flaws of such a typical deployment lie and finally demonstrates how to significantly improve the GA's performance [92, Knerr et al.].

The inspiration of GA originates from the modifications to the chromosomes of a species

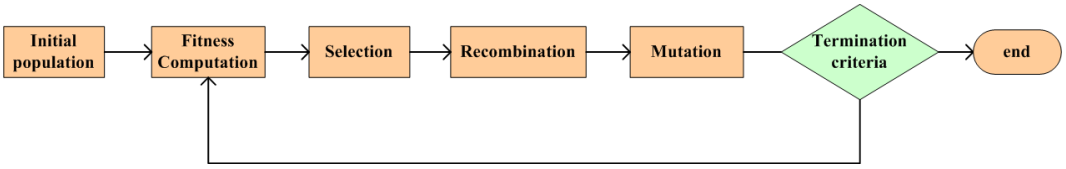


Fig. 4.16: 3-operator genetic algorithm.

caused by natural reproduction that iteratively improve the fitness of the species. According to the natural mechanisms, abstracted concepts of *selection*, *recombination*, and *mutation* exist in the algorithm as depicted in Figure 4.16. A population of individuals (or solutions) exists in a generation $\mathcal{P} \subset \mathbb{S}$, of which a subset of individuals is chosen to serve as the parents of the population of the next generation. This *selection* process is guided by the fitness (quality of solution) of the individuals. The creation of a new individual for the next generation by mating of the parent individuals is called *recombination*. The concept of *mutation* is a random mechanism that affects parts of a chromosome of an individual with a certain probability. *Mutation* ensures a persistent diversity in the number of individuals in any population. Depending on the problem formulation, there exists a large variety of concrete implementations for all these mechanisms, that cannot be covered within the scope of this paper. In the following we adhere to the classical terms and definitions used by Goldberg [49] and Michalewicz [107].

Chromosome Coding

The fundament of any GA is the genome, which captures all necessary information to derive a solution for a problem instance. Many different approaches exist, but most common in general and for system partitioning in particular is a string representation. An intuitive and comprehensive way to represent a solution for the partitioning problem in form of a genome is depicted in Figure 4.17. Assume a system graph with $|\mathcal{V}|$ processes shall be partitioned. A vector of length $|\mathcal{V}|$ is provided, in which every entry, a *gene*, corresponds to a specific process:

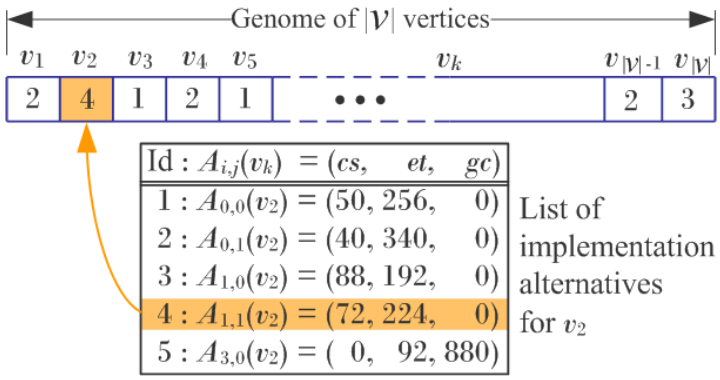


Fig. 4.17: Chromosome coding for the system partitioning problem.

The value of a gene (*allele*) identifies the implementation alternative for the respective process. A specific partitioning solution is then coded as such a vector filled with the implementation types for all its processes. A coding in this form is very beneficial, since recombination and mutation schemes can be easily defined, as can be seen later on. Nearly all publications in this field adhere to this concept. However, the question, in which order $(v_1, v_2, .., v_{|\mathcal{V}|})$ the processes are aligned in the genome vector, is hardly ever raised and almost always said to be arbitrary. In fact, this is problematic, because of the consequences of the fundamental schema of genetic algorithms [49]. The theorem states explicitly that short, low-order, and highly fit schemata are sampled, recombined, and resampled to form strings of potentially higher fitness. When precedence graphs are considered and optimisation is subject to time, a trivial observation consists in a mapping to be very beneficial for *two* reasons: first, if - in general - process implementations are mapped such that they feature comparably low execution times, which is a *combinatorial* trait; second, if the system graph is mapped to the architecture graph such that the inherent parallelism in both matches very beneficially, which is a *structural* trait. It is the latter aspect, which causes serious performance differences depending on the chosen vertex order in the genome.
Assume the graph in Figure 4.18 features a highly fit mapping with respect to its timing for

the substring containing the vertices e, f, g, h due to a clever exploitation of the parallelism by mapping these vertices to *different* resources in the architecture graph. The schema representing this substring is $(3 * * * *1 * * * 2 * 2*)$ for Coding 1 and $(* * * * 2312 * * * **)$ for Coding 2, with $*$ representing wild cards in a schema. Both have the same low order $o = 4$, which is the number of fixed values in a schema, and a very different defining length δ, which is the maximum distance between any two fixed values of a schema. In the Coding

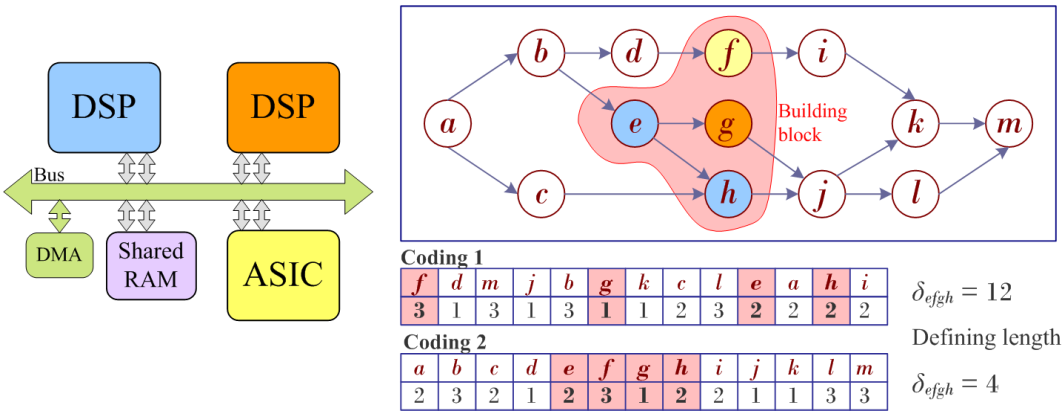


Fig. 4.18: Examples for bad (1) and good (2) chromosome codings.

1 it is almost certain that a beneficial mapping for e, f, g, h will be destroyed during the next recombination cycle, since the *defining length* δ of this schema is very high for standard recombination operators such as one-point, multi-point or uniform crossover. Hence, the probable persistent destruction of these schemata hinders an efficient exploitation of this structural trait, which inevitably affects the convergence and overall performance of the genetic algorithm. Naturally, when such a structural component does not exist, e.g. when there are no precedence constraints or execution time is not of interest, this consideration is irrelevant.

But in general execution time is a predominant optimisation objective and contributes largely to cost function. Due to this strong relation, it is of major importance to align neighbouring (with respect to the time dimension) vertices in the system graph preferentially next to each other in the chromosome. This trait, although being a fundamental neighbourhood property of this problem formulation, is hardly ever considered in the field of genetic algorithms for system partitioning. The only work known to us elaborating on this trait has been published by Dick et al. [35]. They conclude to order their task vector according to a depth first search through the graph concatenating the vertices as they are visited. This is an astonishing surmise, since such an order destroys the parallel vicinity of vertices in an one-dimensional array instead of emphasising it. A depth first search could result in vertex order $(a, c, \mathbf{h}, j, l, m, k, b, \mathbf{e}, \mathbf{g}, d, \mathbf{f}, i)$ for the example in Figure 4.18. Trivially, a breadth first

search delivers then the more promising candidate resulting in $(a, c, b, \mathbf{e}, d, \mathbf{h}, \mathbf{g}, \mathbf{f}, j, i, l, k, m)$ and even much better candidates exist, as it is shown in the following. The theoretical underpinning of the survival of schemata and neighbourhood examinations can be found in the literature [49, 107].

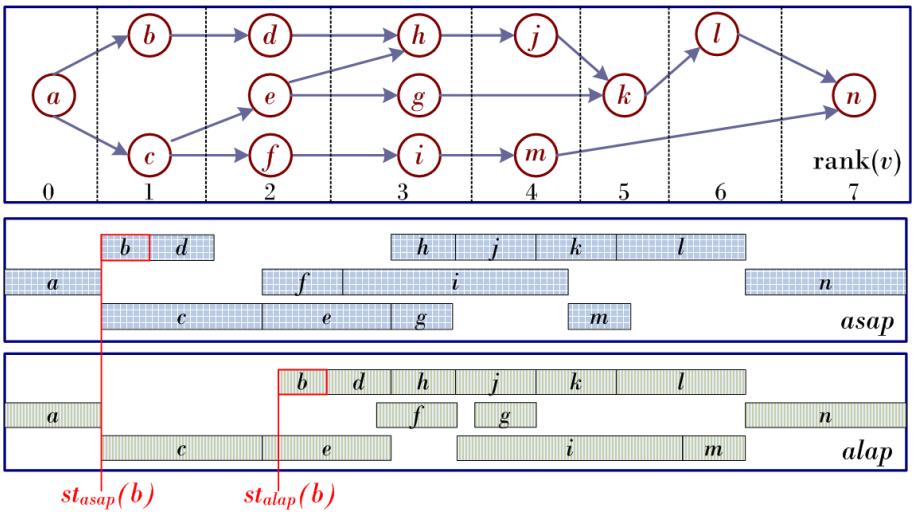


Fig. 4.19: Example graph with annotated ranks, asap and alap schedule.

The remaining section evaluates different chromosome codings and points out the strong dependency between the chromosome coding and the timing payoff. Additionally it is shown, that these considerations do not in any way affect other objective values like code size or area, since these are *combinatorial* and thus independent from the graph structure.
Three different codings are proposed: a random order (rand) of genes, an order based on the vertices' rank in the graph (rank), and a more elaborate order based on the medium start times of an *as soon as possible* (asap) and *as late as possible* (alap) schedule of the graph (med). In Figure 4.19 a small example graph is depicted, in which the ranks of the vertices are annotated. Additionally, the two schedules, asap and alap, are depicted. In the interest of clarity, communication is omitted in this figure and the execution times are averaged over all implementation alternatives per process.
It becomes obvious that a genome, in which the genes are ordered according to the rank of the corresponding vertex, mirrors the vicinity relations of the graph in a reasonable way. A further intensification of this relation lies in the integration of the time dimension and the vertices' dynamic range in the graph. This information can be imported by the application of the two indicated schedules, of which the start times serve as base for the genome ordering. For instance, process a features in both schedules the same start times $st_{\text{asap}}(a) = st_{\text{alap}}(a)$, whereas process b exhibits different start times $st_{\text{asap}}(b) \neq st_{\text{alap}}(b)$. In the latter case the mean is taken as base for the genome ordering. It has to be stated, that for all processes

more than one execution time exists, so the generation of the two schedules has to rely on the average execution time. Nevertheless, a *med* based ordering of the genome preserves the locality of the processes in the system graph more effectively. The plot in Figure 4.20

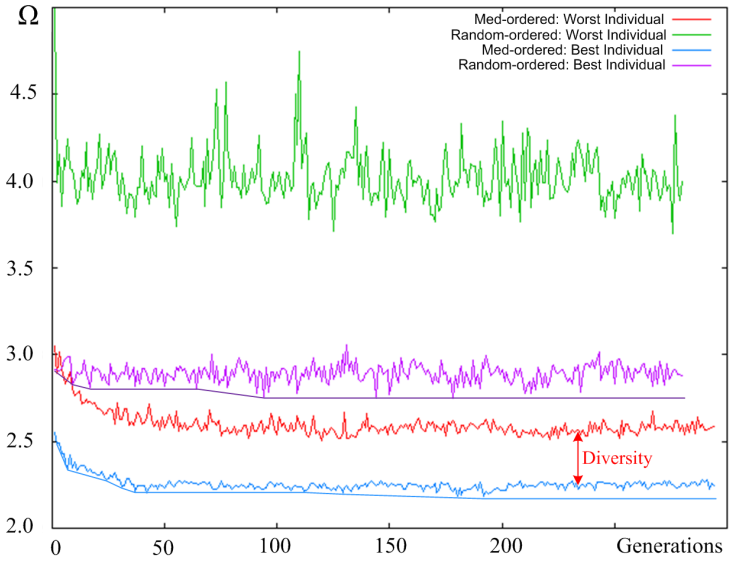


Fig. 4.20: Convergence behaviour for GAs with different genome codings.

illustrates the convergence behaviour of two genetic algorithms with identical parameters but different chromosome codings. The four plots show the worst and best member of a generation for a GA with randomly ordered chromosomes in the upper part and those for a GA with an ordering depending on the asap and alap schedules in the lower part. The difference between the quality of the best and worst individual of a population is an indicator for the population's diversity, which is basically adjusted by the mutation operator in Section 4.3.6. It is apparent in this example that the convergence of the upper two curves is significantly inferior, since promising candidates in the population are persistently destroyed by the recombination and mutation operators. The empirical demonstration of this trait for many instances of many graphs can be seen in the bar chart in Figure 4.21 with averaged cost $\overline{\Omega}$ on the y-axis. The bar groups a, $b2$, and c result from GA runs with identical weight elements in the weight vector $\mathbf{w} = (1, 1, 1)$ (3.10). The effect of the chromosome coding is dramatic with the biggest difference for large graphs: up to 20% better than random coding. This is a reasonable result, since small graphs mean implicitly short chromosomes, in which a disorder has only limited impact.

For $|\mathcal{V}| = 50$ two more tests are depicted in bar groups $b1$ and $b3$ with a variation in the fitness function. When optimisation is only subject to time $\mathbf{w} = (1, 0, 0)$, a GA with randomly ordered genome performs tremendously worse. In opposition, when neglecting time completely $\mathbf{w} = (0, 1, 1)$, the ordering does not matter at all, leading to identical

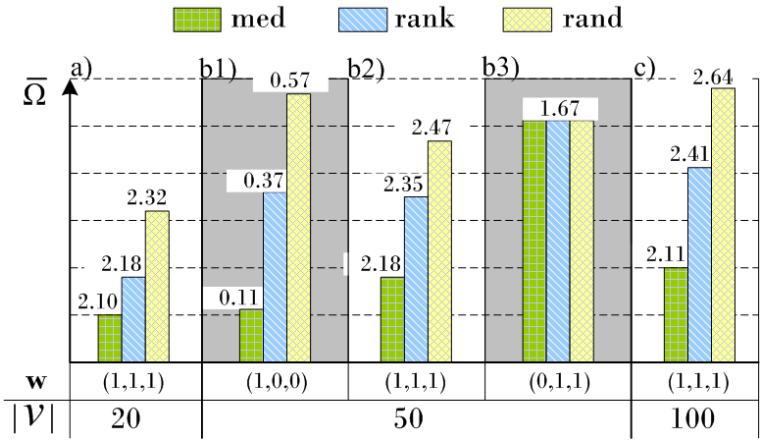


Fig. 4.21: Averaged cost $\overline{\Omega}$ for different genome codings on all graph sizes $|\mathcal{V}| = 20, 50, 100$.

results. Note, that the demonstrated effect is persistent, even when varying the following three operators *selection*, *recombination*, and *mutation*. Further parameters for this test have been: binary tournament selection, uniform crossover, and mutation disabled.

1st Operator - Selection

At any stage of the genetic algorithm, i.e. *generation*, among the individuals present in the population, some have to be selected to serve as *parents* for the individuals of the next generation. Again a multitude of selection criteria exist with varying effects on convergence, robustness and solution quality. Although the focus of this work is not primarily set on an evaluation of this feature, three classical schemes have been examined to complete the picture: survival of the fittest (SOTF), binary tournament (BT), and roulette wheel (RW) selection.
Selection based on survival of the fittest means that from a population consisting of $|\mathcal{P}|$ individuals the best $|\mathcal{P}|/2$ individuals are chosen to serve as parents for the next generation. Binary tournament means to select consecutively random pairs out of the population, whose fitness values are compared. The fitter one gets the parent status, whereas the other is discarded. Both are removed from the population to avoid multiple selections of the same individual. Roulette wheel selection distributes probabilities proportional to the fitness values among the individuals. Since our fitness function Ω returns the lower values the better the individual is, the cost-to-fitness transformation $\hat{\Omega}(\mathbf{x}) = \Omega_{\max} - \Omega(\mathbf{x})$ is applied, with $\Omega_{\max}$ being the worst fitness of the last iteration. Hence, the selection probability $P_{\text{sel}}(\mathbf{x})$ of an individual $\mathbf{x}$ in the current population $\mathcal{P}$ calculates to:

$$\forall\, \mathbf{x} \in \mathcal{P} : P_{\text{sel}}(\mathbf{x}) = \frac{\hat{\Omega}(\mathbf{x})}{\sum_{\mathbf{y}\in\mathcal{P}} \hat{\Omega}(\mathbf{y})}. \tag{4.6}$$

Note, that for the RW selection the highly non-linear character of the fitness function when evaluating invalid solutions, due to the penalty exponent η in (3.12), leads to an undesirable effect: the probabilities for invalid solutions become small very quickly. This is circumvented by scaling η with the number of invalid individuals in the population $|\mathcal{P}_{\text{inv}}|$ to $\eta = \eta_0 - (\eta_0 - 1)\frac{|\mathcal{P}_{\text{inv}}|}{|\mathcal{P}|}, \eta_0 = 4$.

The evaluation of this main operator exposes an interesting interdependence with the mutation operator. The plot in Figure 4.22 depicts the three classical selection schemes [49, 68] binary tournament (BT), roulette wheel (RW), and survival of the fittest (SOTF) for different values of common one-gene mutation on the x-axis. The graph size is $|\mathcal{V}| = 50$ with uniform crossover and med-ordered genome. BT and RW selection exhibit an almost

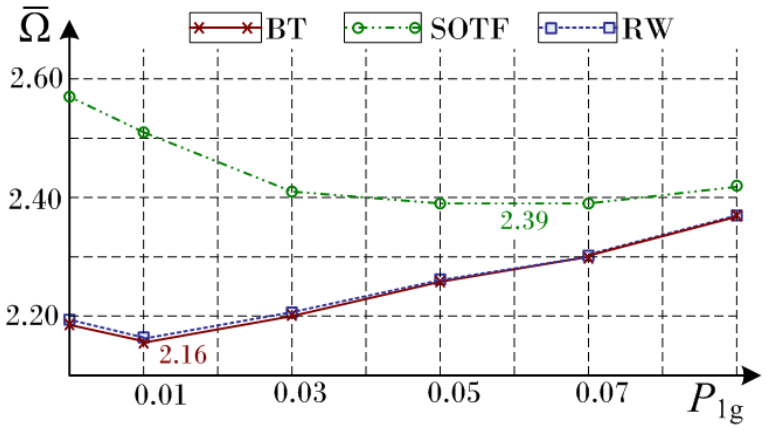


Fig. 4.22: Result for different selection schemes over varying mutation probabilities.

identical performance with minimum cost for low mutation probabilities. In fact, mutation improves the results by only about 1% if $P_{1g} = 0.01$ but degrades the outcome for larger values. The SOTF selection shows a very different behaviour improving $\overline{\Omega}$ substantially with best values for $0.05 \geq P_{1g} \geq 0.07$. A cause of this result may lie in the relatively high implicit diversity of the BT and RW selection opposing the SOTF selection. Hence, mutation, which is another guarantor of diversity, degenerates in the first case rather quickly, but leads in the second case to substantial improvements. For mutation probabilities higher than 0.1 the degradation in cost becomes quickly outrageous for all three selection schemes.

2nd Operator - Recombination

Once a subset of individuals has been selected, the so-called *mating* takes place, that is the creation of new offspring individuals from this subset. Combined with the aforementioned chromosome coding, single- and multi-point as well as uniform crossover are very simple to implement and ensure the creation of offspring solutions that are always feasible avoiding the often costly implementation of a repair mechanism for infeasible solutions. In Figure 4.23 a

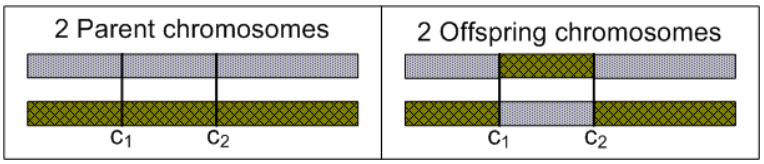


Fig. 4.23: Recombination via 2-point crossover with cut points c_1, c_2.

2-point crossover is illustrated. The two randomly chosen individuals from the parent subset on the left are cut at two points c_1 and c_2 and recombined by permuting the two substrings between the cut points. One- and multi-point crossover recombination are performed analogously. Uniform crossover means a simple iteration over the genes of the parent chromosomes and selecting the allele from one of the two parents with a certain probability. Normally this probability is set to 0.5. In this work, uniform as well as multi-point crossover, reaching from one cut point to $|\mathcal{V}|/10$ cut points, are evaluated and a significant difference can be observed especially for larger graphs. To keep the population size constant, any parent individual is used twice in the crossover scheme with alternating partners.

In this section large graphs with $|\mathcal{V}| = 100$ are tested with four different recombination schemes (uniform and 10-,5-,1-point crossover) on two genome orderings (med and rand. Parameters are: BT selection, and no mutation. The analysed schemes in Figure 4.24 reveal

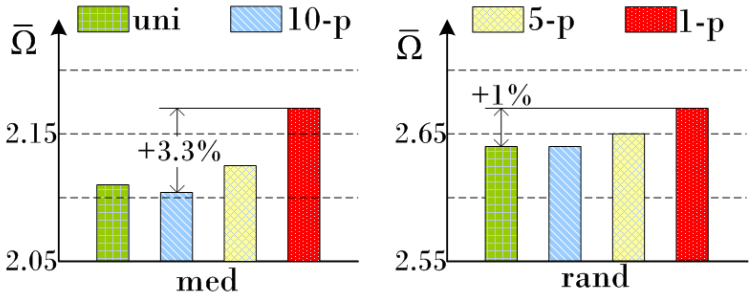


Fig. 4.24: Result for different recombination schemes for two genome orderings.

negligible differences for the GA with randomly ordered genome. But in the med-case on

the left a quite remarkable gap of more than 3% can be observed, which is accompanied by a 15% shorter run time for the GA with 10-point (or uniform) crossover. This effect is still observable to a minor degree for medium graphs ($|\mathcal{V}| = 50$), and is negligible for small graphs ($|\mathcal{V}| = 20$).

3rd Operator - Mutation

The last major operator is a randomised mechanism, which processes the offspring generation and alters small portions of the chromosome with a certain probability. Its main purpose is to provide a chance to (re)introduce new or lost regions of the solution space, and thus to ensure a persistent diversity in the solution subspace covered by the population. Almost omnipresent for the string coding of the chromosome is a simple one-gene mutation (M_{1g}), that is the alteration of an allele typically with a low probability $P_{1g} = 0.01 \ldots 0.05$. A related scheme especially in symmetric multi-processor scenarios is a swap mutation (M_{swap}), that is the exchange of two process assignments to different processors. In this work one-gene and swap mutation are evaluated for varying probabilities. Due to similar deliberations as in Section 4.3.6, it can be reasoned that a one-gene mutation does not tap the full potential, especially with respect to the late stages of the genetic algorithm. Assume, a GA has proceeded through several generations, so that the short low-order building blocks in Figure 4.17 represent on average partial solutions with a rather good quality. As described before, the solution quality then depends to a large degree on the strong exploitation of the parallelism in system and architecture graph, as illustrated in Figure 4.25. One-gene muta-

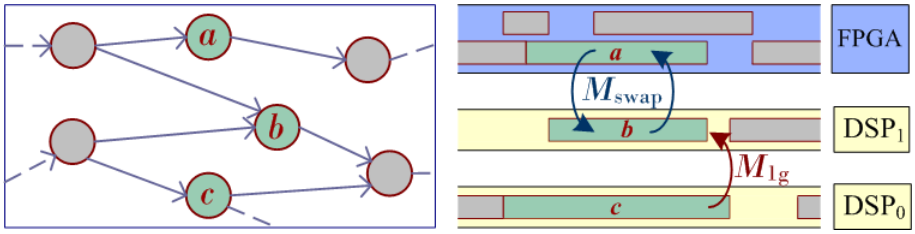


Fig. 4.25: Partial system graph and schedule: one-gene versus swap mutation.

tion (M_{1g}) is likely to destroy the beneficial combination of process assignment to *different* processors, when mutating c onto DSP$_1$ behind b, even if the new implementation alternative of c is better suited to DSP$_1$. Consequently, M_{swap} seems to be much more appropriate, e.g. when exchanging a and b. Since we allow for platform abstractions with more than two processor units, we extend the swap mutation towards a building block mutation (M_{bb}): on a number of consecutive genes $|\mathcal{R}|$ swaps are applied with a certain probability, $\mathcal{R}$ being the set of available processors. The result is a permutation in a limited range of the chromosome that corresponds to a local region (subgraph) of the system graph. The latter is

implicitly true, if a chromosome order is chosen that reflects the locality of the processes in the graph. The next paragraph lists the obtained results for the proposed mutation operators.

Unlike before the results in this section are related to different platform models to demonstrate the dependency of the building block mutation with the number of available processors $|\mathcal{R}| = 2, 3, 4$ with local memories and connected by a system bus to a shared memory. Again the outcome for the largest graphs is considered, as the differences turned out to be most perceptible. Further operators are *med*-ordered genome, binary tournament selection, and 10-point crossover. From Figure 4.26 it can be seen, that a mutation, that permutes the

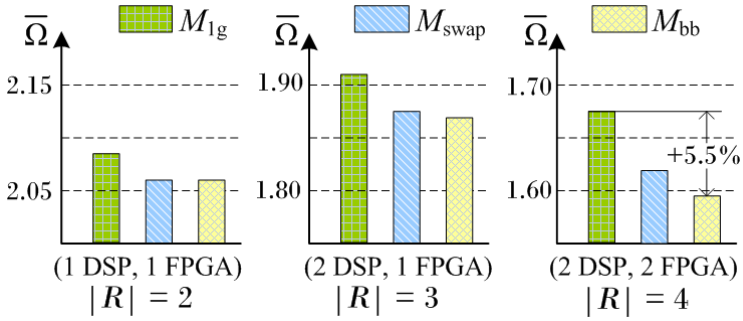


Fig. 4.26: Result for different mutation schemes M_{1g}, M_{swap}, and M_{bb} on three different platforms.

assignment of processes among the available processors, is very beneficial in comparison with the most common one-gene mutation. Remember, that the permuted processes should lie in the same region on the time scale, which is (very likely) implicitly true for adjacent genes in an med-ordered genome. Hence, the building block mutation is very simple to implement and only causes a negligible run time overhead.

Miscellaneous

This section completes the description of the genetic algorithm with the discussion of the parameters population size $|\mathcal{P}|$ and termination criterion. The first of which is of major importance, since it has a dramatic impact on the solution quality in a direct trade-off with the GA's run time: the bigger the population, the better the solution quality. Up to a certain degree the algorithm designer can choose freely depending on his project's time frame. However, it is obligatory to consider certain policies: $|\mathcal{P}|$ has to be large enough to yield a sufficient diversity in the initial population in order to guarantee a good search space coverage. In general, it is reasonable to bind $|\mathcal{P}|$ to the same parameters that determine the problem size and to ensure that any possible allele per gene is present in the initial

population. In this scenario, the population size is then a function, $|\mathcal{P}| = f(|\mathcal{V}|, a_v)$, with the latter parameter being the average number of implementation alternatives per process. We found the product $|\mathcal{V}|\ a_v$ to be an appropriate value.
The termination criterion, i.e. when the GA ceases to breed further generations, scales typically in a similar fashion. However, we found, that terminating after $|\mathcal{V}|/2$ generations without improvement gave enough room to evaluate the operators and showed sufficient convergence.
As stated before, there exist many more parameters and mechanisms for genetic algorithms: elitism, crowding model, overlapping generations, variable neighbourhoods to name just a few. A complete discussion of those would be far beyond the scope of this thesis. We concentrated on the main operators and tried to give interpretations of their performance.

4.3.7 Restricted Range Exhaustive Search

This section introduces the new strategy to exploit the properties of graph structures described at the beginning of this chapter [93, Knerr et al.]. Recall the concept of local optimality that is the fundament of the LEP scheduling technique. Consider the simplified problem of mapping precedence constrained processes to a platform with the single objective to minimise execution time, and additionally data transfer edges are neglected. In that case the graph structure in Figure 4.27 allows for the concatenation of three locally optimal schedules $f_{\mathrm{T,opt}}(\mathbf{x}) = f_{\mathrm{T,opt}}(\mathbf{x}_u) + f_{\mathrm{T,opt}}(\mathbf{x}_m) + f_{\mathrm{T,opt}}(\mathbf{x}_l)$ with f_{T} being the objective function for time, and $\mathbf{x}_u, \mathbf{x}_m, \mathbf{x}_l$ being the upper, middle, and lower partial solution vectors of the complete solution $\mathbf{x}$, respectively. The vertex v_j for relation x_j is identifiable by $\mathrm{par}(v_j) = 0$, i.e. it does not have any parallel vertices[1]. For any vertex featuring this property, the graph can be split into smaller subgraphs, which can be optimised individually. Certainly, as soon as the other objective functions, like area or code size, affect the cost Ω, the prerequisites for these cuts are not fulfilled anymore. But as long as the system's execution time reveals a high relevance in the composed objective function, we accept the trade-off between the restriction of the solution space and the obtainable optimal solutions regarding the timing. Additionally, the majority of the graphs does not contain these cut vertices (also called articulation points [126]). However, from the analysis of typical

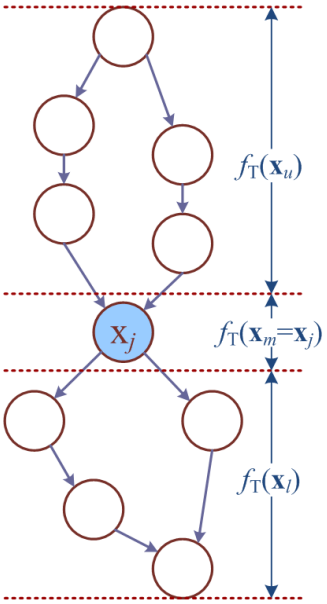


Fig. 4.27: Global optimality by locally optimal solutions.

[1] Equivalently, this property v_j is also called articulation point [126].

system graphs in this field, we observed in general rather low values for $\widehat{\gamma}$, i.e. on average for any vertex the number of its parallel candidates is small. Hence, if we can cover the major part of the neighbourhood of any vertex in a local optimisation, i.e. its direct successors, predecessors, and parallel vertices, we may obtain a concatenation of locally optimal solutions that can be composed to a near-optimal global solution.

Thus, instead of finding proper cuts in the graph, which is rarely possible, we consider a contiguous subset of vertices, or in other words, a moving window over the topologically sorted vertices of the graph, and apply exhaustive searches on these subsets, as depicted in Figure 4.28. The annotations of the vertices refer to the graph in Figure 4.19. The window

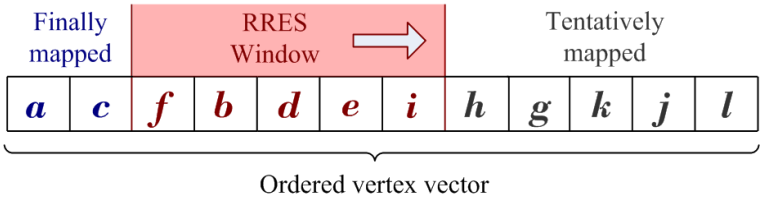


Fig. 4.28: Moving window for the RRES on an ordered vertex vector.

is moved incrementally along the graph structure from the start vertices to the exit vertices while locally optimising the subset of the RRES window.
A crucial part is certainly the identification of the order, in which the vertices are aligned in the vector to be *visited* by the moving window, since the window shall reflect a good neighbourhood coverage for any vertex. The main requirement for the ordering is that adjacent elements in the vector mirror the vicinity of readily mapped processes in the schedule. Recall, that this consideration is identical to that for the creation of a good chromosome coding for the genetic algorithm in Section 4.3.6. Similarly, different schemes to order the vertices have been tested: a simple rank ordering that neglects the annotated execution and transfer times; an ordering according to ascending HLF priority levels that incorporates the critical path of every vertex; and the more elaborate approach guided by the averaged start times of the *as soon as possible* schedule and the *as late as possible* schedule as depicted in Figure 4.19. The order is identical to that of the favoured chromosome coding.
The vertex alignment according to this last scheme yielded the best results compared to rank and HLF ordering, since the dynamic range of possible schedule positions is hence incorporated. It has to be stated, that this vector has to be generated before the algorithm starts and is thus forced to average the possible execution times of any vertex for the asap and alap schedule creation. Note, that this averaging takes place *before* the RRES algorithm starts to ensure a reasonable vertex order and hence a good exploitation of its potential. It shall

not be mistaken as the method to calculate the graph's execution time *during* the RRES algorithm. Within RRES and all other algorithms, any generated partitioning solution is properly mapped and scheduled onto a platform with any process and data transfer featuring its individual unaveraged properties.

Once the vertex vector has been generated, the main algorithm starts. In Listing 4.2 pseudocode is given for the basic steps of the proposed algorithm. In Line 1 the initial solution is created, the details of which are discussed in the next paragraph. Line 2 assembles the vertex vector according to the aforementioned scheme. The loop in lines 4-6 is the windowing across the vertex vector with window length W.

Listing 4.2: Pseudocode for the RRES scheduling algorithm

```
0 RRES() {
1   createInitialSolution();
2   createOrderedVector();
3
4   for(i=1; i <= |V|-W; i++) {
5     windowedExhaustiveSearch(i, i+W);
6   }
7 }
8
9 windowedExhaustiveSearch(int v_i, int v_j) {
10  while (! exhausted) {
11    createNextMapping(v_i, v_j);
12
13    if (constraints fulfilled) { valid = true; }
14    if (cost < bestCost) { storeSolution(); }
15    if (cost < bestValCost && valid) { storeValSolution(); }
16  }
17  mapVertex(v_i, bestSolution);
18 }
```

From within the loop, the exhaustive search in Line 9 is called with parameters for the window from v_i to v_j. In Lines 10-16 any possible mapping of vertices $v_i, \ldots, v_j$ is created, combined with the final mapping for $v_1, \ldots, v_{i-1}$ and the tentative mapping for $v_{j+1}, \ldots, v_{|\mathcal{V}|}$, and then evaluated. Any of these solutions is properly scheduled, avoiding any collisions, and its cost is computed. In Lines 13-15, the checks for the best and the best valid solution are performed. The current final mapping of the *oldest* vertex in the window v_i takes place in line 17. Here, that mapping of v_i is chosen that is part of the best solution seen so far. When the window reaches the end of the vector, the algorithm terminates.

The initial solution in Line 1, i.e. the very first assignment of vertices to an implementation type, has an impact on the achieved quality, although we can observe that this effect is negligible for fast and reasonable techniques to create initial solutions. In Table 4.9 the obtained cost values for a RRES (window length $W = 8$, constraint ratios $(0.5, 0.5, 0.5)$) is depicted starting from different initial solutions for a classical binary partitioning: pure software, pure hardware, random assignment, a more sophisticated but still very fast construction heuristic is described in Section 4.3.3, and when applying RRES on the partitioning solutions obtained by an preceding run with the aforementioned construction heuristic. Apparently,

	$\overline{\Omega}$ obtained when started from				
$\lvert\mathcal{V}\rvert$	pure SW	pure HW	random	heuristic	heuristic and RRES
20	2.241	2.267	2.153	2.101	**2.085**
50	2.569	2.566	2.279	2.185	**2.170**
100	2.700	2.655	2.300	2.202	**2.188**

Ω: Cost (3.10)

Tab. 4.9: Averaged cost $\overline{\Omega}$ obtained for RRES starting from different initial solutions.

the local optima reached via the pure HW and pure SW initial solutions are substantially worse than the others. The random assignment already improves the outcome significantly. The construction heuristic discussed for GCLP in the fourth column considers each vertex' traits individually and incorporates a sorting algorithm with efficiency $\mathrm{O}(|\mathcal{V}|\log(|\mathcal{V}|))$. In the last column RRES has been applied twice, the second time on the solutions obtained for an RRES run with the custom heuristic. The improvement is marginal opposing the doubled run time. These examples shall demonstrate that RRES is quite robust when proceeding from a reasonable point of origin. Further on, RRES is always applied starting from the construction heuristic, since it provides good solutions introducing only a small run time overhead.

Naturally, the most crucial parameter of RRES is the window length W, which has strong effects on both the run time and the quality of the obtained solutions. In Figure 4.29, the first result is given for the graph set with the least number of vertices $|\mathcal{V}| = 20$, since a complete exhaustive search (ES) over all 2^{20} solutions is still computationally manageable. The constraint ratios are strict $(0.4, 0.4, 0.5)$. The vertical axes show the range of the validity percentage Ψ and the best obtained cost values $\overline{\Omega}$ averaged over the 180 graphs. Over the possible window lengths W, shown on the x-axis, the performance of the RRES algorithm is plotted. The dotted lines show the ES performance. For a window length of 20, the obtained values for RRES and ES naturally coincide. The algorithm's performance is scalable with the window length parameter W. The trade-off between solution quality and run time can hence directly be adjusted by the number of calculated solutions $|\mathbb{S}_{\mathrm{RRES}}| = (|\mathcal{V}| - W){a_v}^W$.

Another perspective uncloses an even better comprehension of the RRES algorithm and its workings. This algorithm is now applied to a graph set containing sparse k-locality graphs

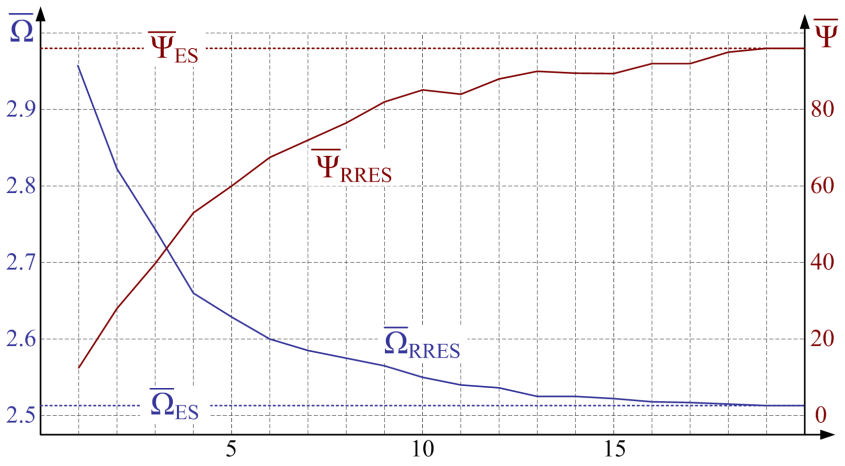
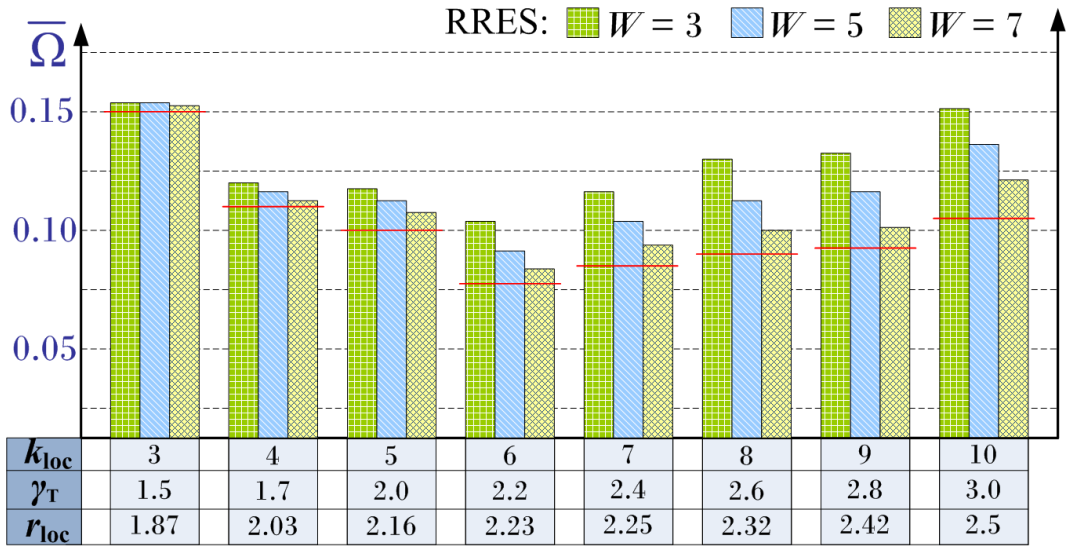


Fig. 4.29: Validity Ψ, and cost $\overline{\Omega}$ for RRES and ES plotted over the window length W.

with $k_{\mathrm{loc}} = 3 \ldots 10$ and $|\mathcal{V}| = 50$, which shall be mapped to a platform featuring three identical DSP cores interconnected by a parallel read/write system bus. We set the objective to minimise for time exclusively, thus suppressing the strictly combinatorial objectives like area or code size in doing so. In Figure 4.30 a bar chart is plotted to visualise the results obtained for the mentioned architecture depicted at the right side. For very low k_{loc} the graphs exhibit rather chain-like structures and the RRES returns near-optimal solutions for any window length ≥ 3. The red lines crossing the bar groups indicate the allegedly near-optimal solutions, generated by extensive test runs with a genetic algorithm featuring a very large population size $|\mathcal{P}| = 600$, binary tournament selection, 2-point crossover recombination and basic block mutation. The bathtub-like appearance of the bar groups results from the relative performance improvement of RRES for increasing values of k_{loc}, as more parallelism is present in the graph. This structural trait is then exploited by a good match to the *three*-core architecture. Thus, the system's execution time is approaching the length of the critical path, until a certain border, $k_{\mathrm{loc,T}} \approx 6$, is trespassed, when the ever increasing parallelism in the graph cannot be exploited anymore by the limited number of resources. For values $k_{\mathrm{loc}} \geq 7$ the gap between the system time returned by RRES and the estimated optimal time is widening rapidly.

From this behaviour the general relation between the architecture traits and the graph structure becomes visible: as long as a nicely fitting subgraph (semi-)isomorphism between platform graph and and system graph is present, the execution time can draw near the minimum. Another RRES specific observation is the strong and reliable performance of this algorithm for a the subset of sparse graphs featuring a high locality (i.e. small k_{loc}). For graphs with higher k_{loc} values or when the objectives are strictly combinatorial, RRES is outperformed by the genetic algorithms and tabu search, as it will be shown in Section 4.3.9.



k_{loc}	3	4	5	6	7	8	9	10
γ_{T}	1.5	1.7	2.0	2.2	2.4	2.6	2.8	3.0
r_{loc}	1.87	2.03	2.16	2.23	2.25	2.32	2.42	2.5

Fig. 4.30: Dependency between graph locality k_{loc} (or $\gamma_{\text{T}} r_{\text{loc}}$) and performance for RRES.

In Table 4.10 simulation results are listed for different window length and constraint sets. RRES returns high quality solutions even for rather small window lengths especially for the analysed subset of graphs. For small graphs even RRES with $W \leq 6$ yields very low cost values for both listed constraint sets, substantially outperforming all neighbourhood searches, GCLP plus modifications, and most notably simulated annealing, without imposing mentionable run time overhead. Tabu search and genetic algorithms prove to be more competitive as it will be shown in the concluding section.

Regarding the asymptotic efficiency of RRES consider the plot in Figure 4.31. It shows the quality dependency of RRES over the window length W and the run time $\overline{\Theta}$ in clock cycles for the graph set with $|\mathcal{V}| = 100$ in comparison to the GA. The constraint set is $(0.5, 0.5, 0.5)$. The shaded area illustrates where RRES outperforms GA both in quality and run time for the analysed process graph set. Apparently the window length should lie below 14 for the binary mapping problem in order to achieve a still feasible run time.

Consequently, a relevant aspect is the consideration of the extended mapping problem, when more than two implementation alternatives exist. It is obvious that the run time of the RRES algorithm suffers greatly from an increasing number of implementation alternatives. Assume for every process in the design four implementation alternatives exist: for instance another DSP is made available and two different FPGA implementations for any process exist trading off area versus execution time. As the run time is then proportional to $(|\mathcal{V}| - W)4^W$, the window length has to be halved to yield the same run time as in the case of two implemen-

$\lvert\mathcal{V}\rvert$	$(C_\mathrm{T}, C_\mathrm{A}, C_\mathrm{C})$	RRES	$\overline{\Omega}$	Ψ	$\overline{\Theta}$
20	(0.4,0.4,0.5)	$W = 6$	2.675	68.9%	**104k**
		$W = 8$	2.612	78.7%	357k
		$W = 10$	**2.586**	**86.8%**	1.2M
	(0.5,0.5,0.5)	$W = 6$	2.121	100%	**144k**
		$W = 8$	2.104	100%	462k
		$W = 10$	**2.094**	100%	1.8M
50	(0.4,0.4,0.5)	$W = 6$	2.813	75.4%	**1.3M**
		$W = 8$	2.774	86.9%	5.0M
		$W = 10$	**2.769**	**90.2%**	25.7M
	(0.5,0.5,0.5)	$W = 6$	2.224	100%	**1.3M**
		$W = 8$	2.208	100%	4.9M
		$W = 10$	**2.199**	100%	25.5M
100	(0.4,0.4,0.5)	$W = 6$	2.776	95.1%	**22.9M**
		$W = 8$	2.76	98.3%	92M
		$W = 10$	**2.743**	**100%**	358M
	(0.5,0.5,0.5)	$W = 6$	2.20	100%	**26.9M**
		$W = 8$	2.198	100%	108M
		$W = 10$	**2.190**	100%	369M

$(C_\mathrm{T}, C_\mathrm{A}, C_\mathrm{C})$: Constraint ratios (3.13), W: Window length, Ω: Cost (3.10), Ψ: Validity (Def. 19), Θ: Run time (Def. 20)

Tab. 4.10: Results obtained for the RRES.

tation alternatives. From Figure 4.31 such a bisection (from $W = 10$ to $W = 5$) may still look acceptable, but it is clear that for an average number of implementation alternatives greater than four per process, RRES becomes quickly infeasible.

4.3.8 Kernighan-Lin Min-Cut

The classical Kernighan/Lin (KL) heuristic for graph partitioning [81] seeks to improve a given two-way graph partition by reducing the edges crossing between parts, known as the cut, as depicted in Figure 4.32. Vahid et al. [140] modified the algorithm and applied it to an access graph representation of a signal processing system. Chatha and Vemuri [25] demonstrated an algorithm, which features the same control mechanism as KL to traverse the search space. In both approaches KL includes system time as objective and both of them have to rely on rough estimations for the execution time in their respective objective functions. A similar KL adaptation based on estimation techniques was also the first system partitioning approach that has been implemented for OTIE [86, Knerr et al.]. In the following, this implementation is adjusted to include precise execution times based on collision-free schedules, Moreover, the impact on KL is exposed, when the composition of the objective function and the definition of the neighbourhood do not allow for an efficient incremental cost computation. Hence, the application of KL to the problem formulation used in this thesis is not straightforward and necessitates a careful consideration of the cost function and how it is computed. Initially, the original version of KL for balanced bipartitioning is described,

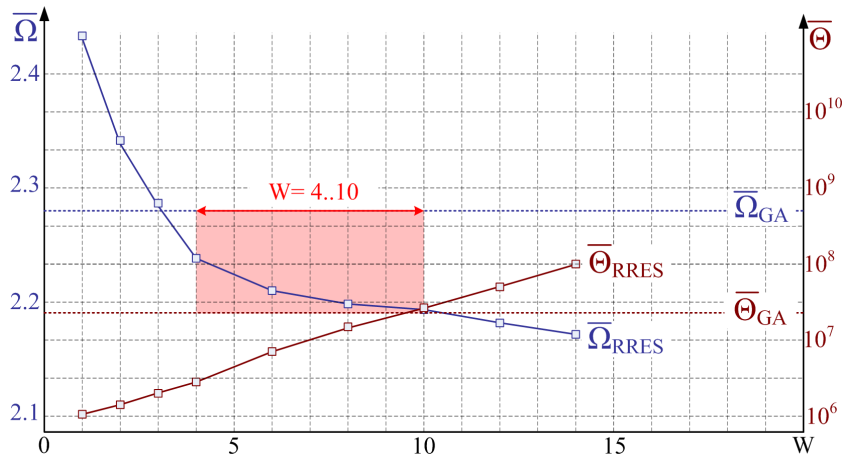


Fig. 4.31: Quality and run time of RRES and GA over window length for graphs with $|\mathcal{V}| = 100$.

followed by the required modifications to be applicable to the chosen setting, their consequences for the KL run time, and eventually its performance regarding different problem sizes.

The essence of the heuristic consists in the simple yet powerful control strategy, which overcomes many local minima without using excessive moves. This strategy can be summarised as follows. The classical algorithm for a balanced binary partitioning starts from an initial solution with the same number of vertices in every partition (Line 1) and tentatively swaps every vertex from one part with every vertex from the other part (Line 4, 12-18). For every swap, i.e. a new temporary solution, the change in cost is evaluated and put in an ordered list, the *change* list (Lines 14-16). The deeper sense of this list is to establish a memory for already performed swaps in order to efficiently manage a search space traversal visiting all possible neighbouring solutions within one iteration.
After all possible swaps have been performed (and again undone), the change list is being processed in the order of increasing cost changes (Line 5, 20-26). The best (or least worse) swap of the list is then *realised* (Line 23), and the swapped vertices are locked (Line 24). As a consequence of this realised swap, in the classical problem only some entries in the *change* list have to be updated (Line 25). This can be achieved very efficiently, since the cut value (its cost) counts the crossing of edges between the two parts, and the movement of a vertex to and fro only affects those list entries that correspond to swaps involving connected neighbours of this vertex. Thus, only very few entries have to recalculated. After locking the swapped vertices and updating the *change* list, the *next best* swap in the list is realised, even when it actually causes a degradation.

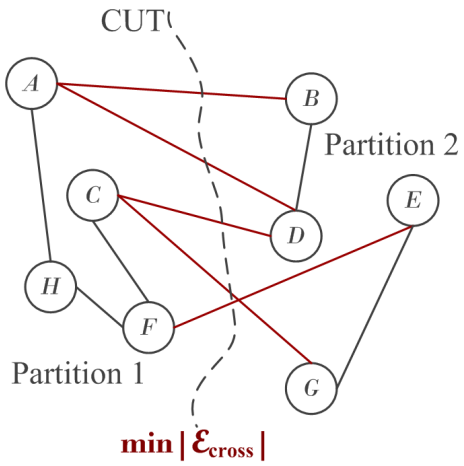


Fig. 4.32: Cut problem for a two-way partitioned graph.

Listing 4.3: Pseudocode for the classical Kernighan Lin min-cut

```
0 ClassicalKernighanLin() {
1   X = initialSolution; // serves as first best solution
2   while(bestSolutionFound) {
3     GenerateChangeList(X);  // see line 12
4     ProcessChangeList();    // see line 20
5     UnlockAllVertices();
6     ReturnToBestSeenSolution(X=X_best);
7   }
8 }
9
10 GenerateChangeList() {
11   for (all pairs x_i,x_j being in different partitions) {
12     SwapPartitions(x_i,x_j);
13     SortIntoChangeList(cost(X_new)-cost(X));
14     UndoSwap(x_i,x_j);
15   }
16 }
17
18 ProcessChangeList() {
19   while (ChangeList not empty) {
20     <x_i,x_j> = SelectNextSwap();
21     if (<x_i,x_j> unlocked) { RealiseSwap();
22                               LockSwappedVertices(); }
23     UpdateChangeListEntries();
24   }
25 }
```

Again, the swapped vertices are locked and the list is updated again, and so forth. When all vertices have been swapped and received the locked status, one pass ends and the algorithm returns to the best partitioning solution seen during the last pass (Line 8). This solution then serves as the initial solution for the next pass, and so forth. After a number of iterations, typically less than five, without global improvement, the algorithm terminates.
There are several simple modifications that have to be applied. We allow for more than two partitions, according to the number of resources in the platform. Another trivial observation is that there is no need for balanced partitions in our scenario, so a swap of two vertices can be redefined as a move of one vertex to another partition. The objective function is indeed affected by 'crossing' edges by the introduced inter-resource communication, which is in general more expensive than intra-resource communication. But the major part of the cost is caused by the individual exploitation of the resources and the overall system time. For the classical algorithm, additional improvements exist that are concerned with sophisticated data structures for constant-time *change* list look-ups and updates [40], etc. These modifications are likewise inapplicable in our scenario as it will be explained in the next paragraph.

The substantial drawback regarding the system partitioning, being binary or extended, lies in the computation of the cost and how it is affected by the move of one vertex. Any cost function that integrates the overall system execution time, that shall *not* be based on crude estimations but on a proper scheduling, hinders the implementation of an *efficient* neighbourhood scan and an efficient *change* list update. In general, every tentative swap or move has consequences for the complete schedule of all resources: collisions on the bus or on a DSP core may newly occur and/or time slots are freed. The execution interval of many processes will naturally change whether or not they are directly connected or in vicinity of each other. Thus, a complete schedule update for *every* new partitioning solution created by the tentative move would be necessary in Line 14 of Listing 4.3. And we encounter the same situation, as soon as the best (or least worse) move in the *change* list is about to be realised in Line 23. Instead of just recalculating a few entries in the list, **all** remaining entries have to be recalculated in Line 25, since it is not possible to determine beforehand which of them are affected by the just realised move. In that sense the *change* list looses its qualification to avoid 'unnecessary' cost function calculations. Hence, this data structure is obsolete and is removed from the algorithm.
The search space traversal strategy of KL can still be utilised, although its property of being extremely fast is hence lost. Listing 4.4 shows the basic steps of the algorithm with the required modifications to be applicable for the scenario chosen in this thesis. After an initial solution has been generated and all vertices are flagged to be unlocked, the current neighbourhood is analysed in Lines 6-10, and the best move is stored. This best move

corresponds to first entry in the change list in the original algorithm, and is as such realised and locked (Lines 12-13). At this point, the short cut offered by a change list to determine the next move is not viable, as the cost in change for **all** neighbours might be affected by the just realised move. Hence, **all** neighbours of the just realised solution $\mathbf{x}_{new}$ have to analysed again to find the next best (or least worst) move. Certainly, a moved vertex is locked, so with every iteration the neighbourhood to be analysed is decreased by one. When all vertices have been moved once, and are thus locked, the algorithm unlocks all vertices, returns to the best solution found during the last pass, and proceeds from there. The termination criterion is met, when the last pass did not yield a new best solution.

Listing 4.4: Pseudocode for the Kernighan Lin partitioning algorithm

```
0 ModifiedKernighanLin() {
1   X = initialSolution; // serves as first best solution
2
3   while(bestSolutionFound) {
4
5     while (Unlocked vertices exist) {
6       for (all x in X) {
7         MoveToAnotherPartition(x); // Tentative move.
8         StoreBestMove(x_best);
9         UndoMove(x);
10      }
11      X_new = RealiseMove(x_best); // Best (or least worse).
12      LockVertex(x_best);
13
14      if (cost(X_new) < bestCost) StoreBestSolution(X_new);
15    }
16    UnlockAllVertices();
17    ReturnToBestSeenSolution(X=X_best);
18  }
19 }
```

With respect to the run time of the modified KL, consider the following scenario. Assume, a graph with $|\mathcal{V}|$ vertices and a_v implementation alternatives per vertex shall be partitioned. Hence, the first tentative neighbourhood scan (Lines 6-10) visits and evaluates $(a_v-1)|\mathcal{V}|$ new solutions. Before the next run, one vertex receives the locked status, and then $(a_v-1)(|\mathcal{V}|-1)$ solutions are computed, in the third run $(a_v-1)(|\mathcal{V}|-2)$, and so forth. When all vertices have been locked, $(a_v-1)(\frac{|\mathcal{V}|}{2})(|\mathcal{V}|+1)$ solutions have been visited. As long as global best solution could be identified during the last iteration, the algorithm proceeds. From the results listed in Table 4.11, we can observe, that the run time lies well in the region of simulated annealing and genetic algorithms. The tremendous degradation becomes more evident, when we recall

that KL is reputed as extremely fast algorithm with an asymptotic efficiency of $\mathrm{O}(|\mathcal{V}|^3)$ or even $\mathrm{O}(|\mathcal{V}|^2 \log(|\mathcal{V}|))$ in some formulations [100]. The main reason can be easily identified, a cost change from neighbour to neighbour is originally a constant-time operation, and the change list a very efficient way to guide the search space traversal. The modified KL has to recalculate the cost change very expensively via a complete schedule update for every possible neighbour and every iteration of the algorithm. Apart from this drawback, the

$\lvert\mathcal{V}\rvert$	$(C_\mathrm{T}, C_\mathrm{A}, C_\mathrm{C})$	$\overline{\Omega}$	$\overline{\sigma}$	Ψ	$\overline{\Theta}$
20	(0.4,0.4,0.5)	2.648	0.0661	69.6%	110k
	(0.5,0.5,0.5)	2.118	0.0219	100%	102k
50	(0.4,0.4,0.5)	2.762	0.0423	73.2%	3.18M
	(0.5,0.5,0.5)	2.253	0.0174	100%	3.31M
100	(0.4,0.4,0.5)	2.758	0.0346	91.6%	100M
	(0.5,0.5,0.5)	2.240	0.0143	100%	110M

$(C_\mathrm{T}, C_\mathrm{A}, C_\mathrm{C})$: Constraint ratios (3.13), σ: Standard deviation, Ω: Cost (3.10), Ψ: Validity (Def. 19), Θ: Run time (Def. 20)

Tab. 4.11: Results obtained for Kernighan-Lin.

control structure of the modified Kernighan-Lin still offers a very beneficial characteristic. Every vertex is forced to leave its initial state in every iteration and can hence contribute to the global cost reduction. Its performance lies well in the region of simulated annealing and tabu search for the chosen scenario. Additionally, it represents a good candidate for the extraordinary case that system time (and hence scheduling) is not of importance for the objective functions, since then the original implementation utilising a change list is again permitted.
In the last section the performance of all algorithms is visualised by bar charts in order to facilitate the direct comparison and evaluation of their advantages and disadvantages.

4.3.9 Discussion

This section reviews the algorithms' performance in direct comparison. In Figure 4.33 a bar chart is depicted for three different graph sizes for the classical binary partitioning problem with one DSP and one FPGA connected by system bus. As this is the formulation for which the original versions of the applied algorithms have been supposed by their respective authors, it is fair to evaluate at first the performance in this exact scenario. Exhaustive search (ES) could only be deployed for the graph set with the least number of vertices. The general parameters are a balanced constraint ratio set of $(C_\mathrm{T} = 0.5, C_\mathrm{A} = 0.5, C_\mathrm{C} = 0.5)$ and dynamic HLF scheduling being applied.
The specific parameters for the algorithms are:

- RRES: Window length $W = 10$.

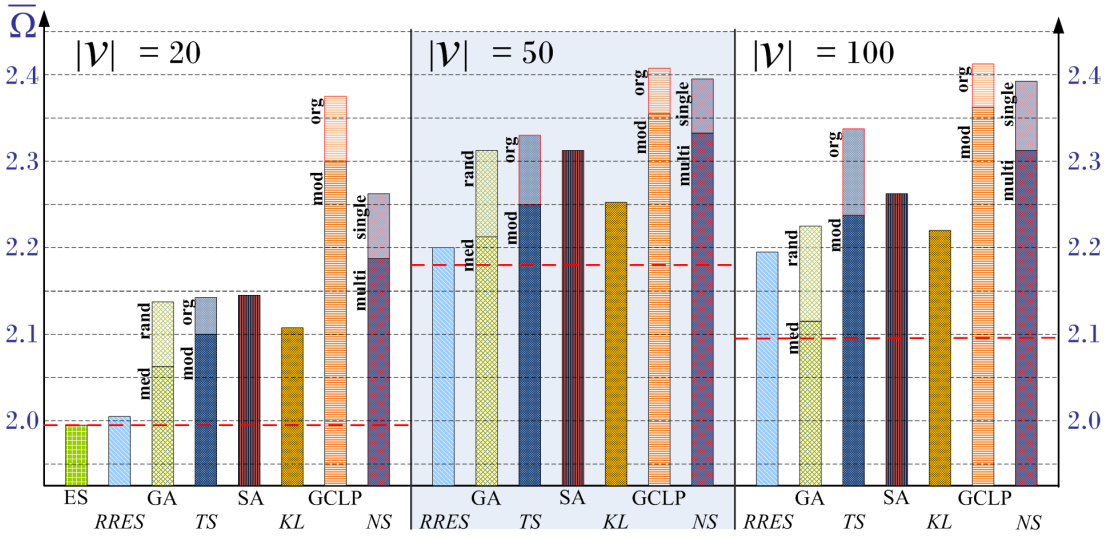


Fig. 4.33: Quality of all algorithms for different graph sizes for binary partitioning.

- GA: Population size $= 2|\mathcal{V}|$, med-ordered and rand-ordered chromosome, binary tournament selection, multi-point crossover recombination with $\frac{|\mathcal{V}|}{10}$ points, swap mutation as the number of resources is $|\mathcal{R}| = 2$, termination criterion after $|\mathcal{V}|/2$ generations without improvement.
  The results for a randomly ordered chromosome are not depicted as they would lie well beyond the value range of this bar chart.

- TS: Neighbourhood size $|\mathbb{S}_{\mathrm{N,TS}}| = \max(5, 0.7\sqrt{|\mathcal{V}|})$, tabu list length $L_{\mathrm{tabu}} = 0.7\sqrt{|\mathcal{V}|}$, region for intensification and diversification covers $s_{\mathrm{reg}} \leq 100,000$ different solutions, here 16 elements of $\mathbf{x}$. The termination criterion is reached after $10|\mathcal{V}|$ analysed solutions without improvement.
  The **mod** subscripted bar indicates the TS with HLF scheduling, whereas the **org** stands for the original version by Wiangtong.

- SA: Annealing factor $\vartheta = 0.95$, temperature update after $u = |\mathcal{V}|$ iterations, termination after $|\mathcal{V}|/2$ updates without improvement.

- KL: No parameters.

- GCLP: **org** indicates the original version, and **mod** indicates the version with the modifications M1b and M3 deployed.

- NS: A trivial *best* neighbourhood search is indicated by **single**, and the multi-start version with $|\mathcal{V}|$ consecutive runs is indicated by **multi**.

In summary, it can be stated that the well reputed *global criticality local phase* algorithm is not a favourable technique for the system partitioning problem. Although being indeed a very fast approach, it is on average easily outperformed by direct neighbourhood searches, when they are started from the same initial solution. However, the proposed modifications attenuate its poor performance, so that at least the direct neighbourhood searches drop behind. Of course, the run time of the GCLP is very low, for the graph size of $|\mathcal{V}| = 100$ it turns out to be already 50 times smaller than that of the *best* neighbour gradient search as it is depicted in Figure 4.34. This fact discloses its application to large and very large graphs, for which all other algorithms would need many hours to terminate successfully.

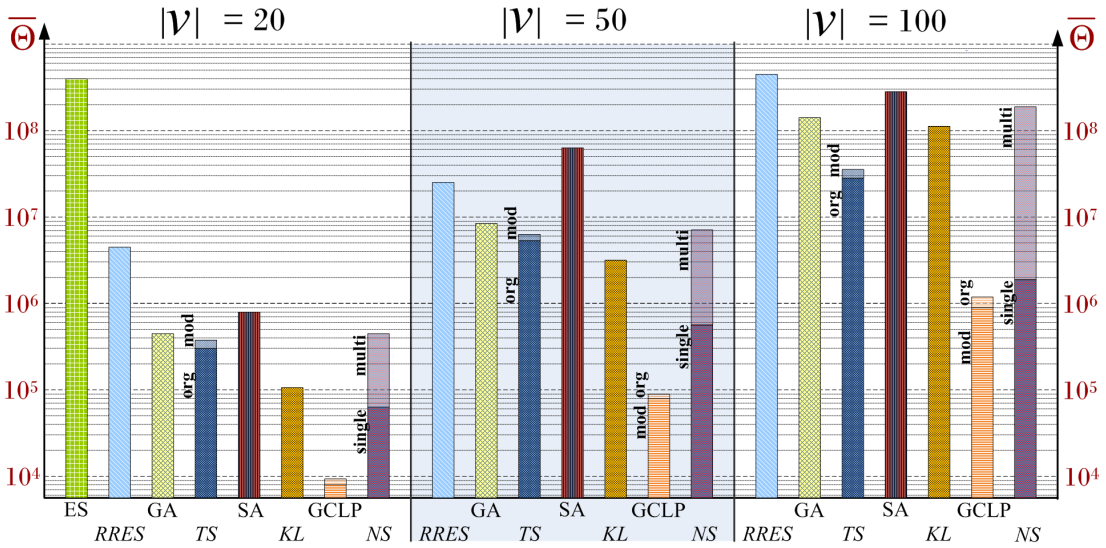


Fig. 4.34: Run time of all algorithms for different graph sizes for binary partitioning.

Comparably, simulated annealing yields a first significant performance gain, but it is accompanied by a large run time of about 300Mcycles for $|\mathcal{V}| = 100$ in this scenario. Wiangtong's tabu search, improved by a more powerful scheduling, offers a run time of about a tenth of SA, while it still exposes better quality results. The adjusted Kernighan-Lin is obviously competitive with respect to solution quality, although concerning the run time it is far from the extremely fast and efficient original approach.

Restricted range exhaustive search and genetic algorithm prove to be significantly better than the aforementioned approaches. For small graphs, in which the windows length of RRES covers a substantial part of the solution vector, it draws near the optimal solution. Its performance drops back behind the genetic algorithm with the proposed chromosome coding with a growing margin for larger graphs. The run time of RRES with 369Mcycles and GA with 429Mcycles is recognisable but not significant.

When the system graphs are untypical with respect to locality and sparsity, and the platform to be partitioned for is of very heterogeneous nature with many different resources $|\mathcal{R}| \geq 6$, and the objectives incorporate many combinatorial metrics, then the performance evaluation is still inclined towards genetic algorithms. Although it is notable, that all algorithms seem to be in reachable distance. For this test a platform with four DSPs and two FPGAs interconnected by three system busses has been composed for a graph set containing general graphs without distinct locality property (i.e. $k = |\mathcal{V}|$), as explained in Section C.2. The average graph parameters are: $|\mathcal{V}| = 100$, $\rho = 6.58$, $r_{\text{loc}} = 5.21$, $\gamma = 5.66$, $\widehat{\gamma} = 25.4$. The optimisation has been subject to the complete set of objectives, including code size and gate count constraints for any resource, and a deadline until which the complete graph had to be executed. The constraint set has been set to $(C_{\text{T}}, C_{\text{C}}, C_{\text{A}}) = (\frac{1}{5}, \frac{1}{5}, \frac{1}{5})$, which is a rather loose constraint set, since there are six resources present, and hence to every resource $\frac{1}{6}$ of the vertices is assigned. The dashed red line in the bar chart in Figure 4.35 indicates the averaged cost

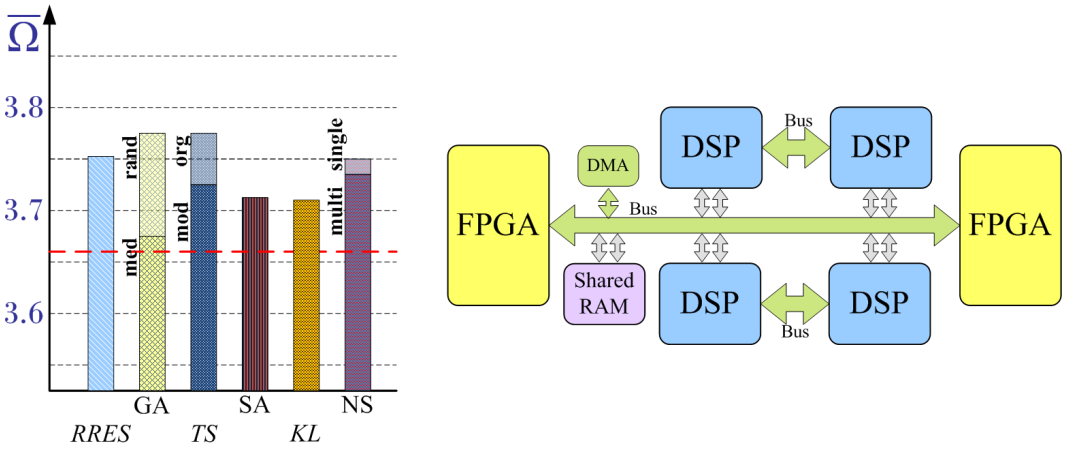


Fig. 4.35: Quality of all algorithms for different graph sizes for a heterogeneous platform.

over all graphs computed over their respective minimum cost regardless by which of the algorithms it has been returned. The chosen scenario entails that the graphs' parallelism can be easily matched to the strong parallelism of the platform. With respect to the cost function, it is of importance that there are now six individual resource constraints and one general execution time constraint. Hence we have seven additive terms in (3.10) of which time contributes only a fraction of $\frac{1}{7}$, if the weight vector is balanced, $\mathbf{w} = (1, 1, 1, 1, 1, 1, 1)^T$. Therefore, the structural advantages of RRES due to local optimality and of GA due to the structural chromosome coding are of minor relevance, because $\frac{6}{7}$ of the cost is influenced by combinatorial objectives. For all algorithms the standard deviation of the obtained solutions showed to be very low, being highest for the *next* neighbour gradient search.

The specific parameters of the algorithms for this experiments are:

- RRES: Window length $W = 4$.
- GA: Population size $= 2|\mathcal{V}|$, *med*-ordered chromosome, binary tournament selection, multi-point crossover recombination with $\frac{|\mathcal{V}|}{10}$ points, basic block mutation every fourth individual ($P_{\mathrm{mut}} = 0.25$), termination criterion after $|\mathcal{V}|/3$ generations without improvement.

- TS: The tabu search did not work well for many parameter sets in which $|\mathbb{S}_{\mathrm{N,TS}}| \leq 10$, with growing margin towards smaller tabu list length L_{tabu}. The best results have been obtained for a large neighbourhood of $|\mathbb{S}_{\mathrm{N,TS}}| = 20$, and a short tabu list of $L_{\mathrm{tabu}} = 3$, hence prohibiting 60 vertices to be moved while analysing 20 of the remaining vertices per iteration. Region for intensification and diversification is $s_{\mathrm{reg}} \leq 6^6 = 46,656$ and covers 6 elements of $\mathbf{x}$. The termination criterion is reached after $10|\mathcal{V}|$ analysed solutions without improvement
  The **mod** subscripted bar indicates the TS with HLF scheduling, whereas the **org** stands for the original version by Wiangtong [142].
- SA: Annealing factor $\vartheta = 0.95$, temperature update after $u = |\mathcal{V}|$ iterations, termination after $|\mathcal{V}|/2$ updates without improvement.
- KL: No parameters.
- GCLP: Not applicable to extended partitioning in its current form.
- NS: A *next* neighbour gradient search has been applied, which is indicated by **single**, and the multi-start version with $|\mathcal{V}|$ consecutive runs is indicated by **multi**. In this setting the *best* neighbour gradient search performed significantly worse than the *random* and *next* neighbour gradient searches.

However, the GA turned out to be the best technique with the same parameter set as in the previously described binary partitioning and proved to be robust against its own parameter variation. The doubling of the population size simply lead to less generations and very much the same averaged cost values. Basic block mutation showed a much better impact than 'bit flip' mutation, but whether the mutated individuals per population covered 10%, 15%, or 25% of the population does not seem to have a strong impact.
Simulated annealing and tabu search yielded basically the same performance with now a similar run time. Simulated annealing revealed the same trait as GA regarding parameter variation. If the temperature update u occurred after $|\mathcal{V}|, 2|\mathcal{V}|$ or $4|\mathcal{V}|$ affected the run time

but not the average solution quality. Of course, for values lower than $|\mathcal{V}|$ the obtained results started to degrade.
Higher effort had to be undertaken to tune the tabu search to this scenario. The proposed values of Wiangtong for tabu degrees of $L_{\text{tabu}} = 7..20$, and $|\mathbb{S}_{\text{N,TS}}| \approx \sqrt{|\mathcal{V}|}$ turned out to be not competitive. Recall, that the tabu region of the forbidden vertices per iteration calculates to $N_{\text{tabu}} = L_{\text{tabu}}|\mathbb{S}_{\text{N,TS}}|$. For any combination $L_{\text{tabu}}|\mathbb{S}_{\text{N,TS}}| \leq |\mathcal{V}|/3$, i.e. a low number of prohibited vertices per iteration, the results deteriorated quickly. The best results have been found for $L_{\text{tabu}}|\mathbb{S}_{\text{N,TS}}| \approx |\mathcal{V}|/2$, with a comparably large $|\mathbb{S}_{\text{N,TS}}| = 20$ and a short $L_{\text{tabu}} = 3$. This setting is rather opposite to that proposed by the original authors. Hence, we conclude that the analysed neighbourhood per iteration is of major importance and has to scale with its growing size.
In opposite to TS and GA, the Kernighan-Lin based approach does not require a tedious parameter adjustment, which can turn out as very beneficial trait, when the designer varies frequently the platform composition. Then, Kernighan-Lin is immediately ready to be applied, whereas tabu search and genetic algorithms undergo typically an adjustment of their parameters. This advantage shares KL with RRES and the NS approaches.
The *next* neighbour gradient search performs surprisingly well, being even better than RRES, with only a fraction of its run time. When the multi-start option is set to 100, the performance does not further improve significantly. The run time of a single start is approximately a $\frac{1}{20}$ of that of GA, SA, and TS, and $\frac{1}{50}$ of RRES.
Naturally, the window length of RRES had to be diminished to four to offer a manageable run time, and its good exploitation of the structural local fit between architecture and graph is obscured by the number of combinatorial objectives, that do not possess such a locality trait. Even a gradient search that can repeatedly visit the same vertex over and over again is hence more capable to cope with the given scenario.

4.4 Criticism

Although the degree of realism that the proposed framework with respect to architecture detail offers is significantly higher than in any other framework that has been referred to, a major drawback still exists. The question, how realistic the metrics and estimations for timing, code size, gates, power, etc. of the individual processes are, is very hard to answer, and research teams struggle hard to give a reliable feedback [61]. As an immediate consequence even a thoroughly decomposed system graph and a true to detail architecture model, on which this graph is partitioned and scheduled, can only be as good as the underlying estimations of the graph components. Until now, the proposed framework does not accomplish any means to include uncertainty parameters such as a reliability metric for any estimated value, let

alone a sophisticated calculus to combine these values to a confidence level of a returned partitioning solution that scales comprehensibly with these individual reliability values.
Related to this rather general aspect, a concrete problem shall be highlighted. One of the reasons for the focus being set rather on data driven signal processing systems, as the receiver circuits of a UMTS baseband chip, and xDSL transceivers, etc. lies in the fact of the unpredictability of control oriented code in terms of execution time. According to the data being processed functional parts, whether they are compiled code on a DSP or a custom data-path burned into silicon, could vary their behaviour to a large degree. Assume for instance, WCDMA correlator banks instantaneously vary their filter length according to the signal to noise ratio of the current communication link. Such a scenario inhibits the assignment of a *single* execution time for the process carrying out the filter function, but rather introduces an execution time profile for this functional part. Such a profile outlines all possible values the execution time can adopt depending on the current control parameters and the data being processed. Static code analysis yields typically feedback on the best and worst execution times of such a function or code segment, and a partitioning and scheduling approach may either work with average or conservatively with the worst case times. But the larger the number of these control depending system components, the less the chance to obtain a reasonable schedule and thus a good partitioning solution based on static values. In the considered designs made available by our industrial partner, the most time critical parts of these designs are in fact highly data-driven and the worst case *is* the usual real scenario. Still, it is foreseeable that the complexity of electronic system design is ever-increasing and approaches based on conservative estimates do very likely not tap the full potential of embedded systems. The incorporation of reliability metrics and confidence levels of the obtained solutions is hence a task for the near future.
Although all the features are present in the architecture and algorithm library to extend the system partitioning to a completely automated design space and architecture exploration, this step has not yet been accomplished and has thus not found its way into this thesis. Solutions, at present encoded as a vector of implementation alternatives for a *fixed* set of resources, could encode the current platform composition as well, for which then optimisation operators had to be excogitated. Or in a different scenario, an outer algorithmic structure could be applied to analyse the outcome of the inner partitioning framework for instance regarding processor load, bus load, number of postponed processes and data transfers (arbitrated collisions), power consumption and so forth, to modify the platform composition by removal of a poorly utilised DSP or insertion of another communication channel. However, the underpinning for these very interesting and promising approaches, in which not only the *mapping*, but also the *allocation* can be automatically optimised within in a single algorithmic procedure, has been successfully established.

5 CONCLUSIONS

"Ich bin voller Gedanken und äußere daher nicht immer den richtigen [1]*."*
KATZ UND GOLDT

This thesis presents a system partitioning framework for embedded systems in wireless communication incorporated into OTIE, the open tool integration environment. OTIE has been developed in the Christian-Doppler Laboratory for Design Methodology of Signal Processing Algorithms to overcome the obstacles in embedded system design caused by the fragmentation of the design process and the multiplicity of inherent hard optimisation problems. System partitioning for heterogeneous platforms is amongst the most challenging problems in this field. This thesis establishes a detailed C++ library within OTIE to compose arbitrary architecture models that allow for the exact representation of a wide range of heterogeneous platforms being composed of manifold processing and communication resources. Based on this fundament, a variety of existing and newly developed partitioning and scheduling algorithms has been implemented in this library to provide a powerful means to map the given functionality of an electronic system onto a dedicated platform subject to multiple competing objectives. The unique contributions of this thesis can be summarised as follows:

- A major contribution that enabled the development of any partitioning algorithm is constituted by the thorough and realistic modelling of architecture components as well as the platforms being assembled by them. The huge majority of the partitioning engines known from literature base their insights on very simple platform models and highly unrealistic assumptions about the processing elements and intra-platform communication, if considered at all. In this thesis the focus has been set on embedded systems in the wireless domain, which is the embodiment of platform heterogeneity. Arbitrary structures including DSPs, μPs, ASICs, FPGAs, and busses etc. are available for the designer and can be composed as desired. Great care has been taken to model static schedules and communication overhead precisely. According to the designer's needs all resources are parameterisable with respect to occupied chip area, base power consumption, and latency, DSPs can scale their dynamic power consumption according to their processor load etc.

[1] I am filled with thoughts and do therefore not always express the right one.

- The analysis of system graphs typical for this design domain delivered valuable insights to the structure of the targeted problem formulation. Based on these insights and on the architectural backbone the biggest attainment has been made with the implementation of several partitioning and scheduling algorithms. The multiplicity of implemented techniques for partitioning include exhaustive searches, hill-climbing neighbourhood searches, simulated annealing, the global criticality/local phase algorithm, tabu searches, genetic algorithms, and the restricted range exhaustive search algorithm. The scheduling problem has been tackled to a minor degree with Hu's dynamic highest level first, with Hwang's dynamic earliest task first, and a scheduling guided by the local exploitation of parallelism. The following list highlights briefly the achievements for any algorithm individually.

  - The system partitioning formulation embeds a multi-processor scheduling problem at its very core. The focus has been set on very fast list scheduling techniques, namely HLF and ETF. The analysis of the graph and architecture properties revealed potential for a better exploitation of the parallelism present in both. A new scheduling algorithm has been developed that shows superior performance in terms of returned schedule length that scales with the parallelism factor in the graphs. This performance gain of up to 6% is paid by the introduction of an admissible run time overhead of about 30%.
  - Besides exhaustive search for small problem sizes, three hill-climbing algorithms with a multi-start option have been implemented in order to assess the structure of the search space and to obtain first benchmarks for more sophisticated optimisation techniques. A simulated annealing strategy constituted the natural extension of these direct neighbourhood search algorithms. It has been chosen to serve as more competitive benchmark provider, since one of its first deployments was the classical graph bipartitioning.
  - Another technique has been developed, which is based on the Kernighan-Lin min-cut algorithm for graph bipartitioning. Being in its classical form well-known as extremely fast approach, the precise computation of the objective functions disables this property. Still, it was demonstrated that its control structure is applicable and shows a competitive performance compared with simulated annealing and tabu search, with a similar run time. The biggest advantage is the absence of any complicated operator, which demands for a tedious adjustment before the partitioning can start.
  - The *penalty reward* tabu search based on Wiangtong's work has also been chosen as competitive candidate for this optimisation problem. Its original scheduling technique turned out to be inferior and had to be replaced by one of the three

techniques described before. The replacement yielded a performance boost of up a 10% reduction in cost, when the system's execution time was the only objective, and still up to 4%, when multiple objectives were considered.

- The well reputed *global criticality local phase* (GCLP) algorithm has been added for the solution of the binary HW/SW partitioning problem. It has been thoroughly analysed and several modifications to increase its performance have been introduced. Depending on the problem instances and the designer's intentions, two versions of GCLP advancements have been presented, either of which yielding significantly better results than the original algorithm with the focus set on different objectives. The first modification set yielded up to 25% run time reduction with negligible impact on the number of returned valid results. The second applied modification set improved both the solution quality and the validity percentage between 2% and 5%.
- The classical three-operator genetic algorithm has been implemented and thoroughly analysed in application to the system partitioning problem. The significant relevance of the underlying genome coding for typical problem formulations was demonstrated, which has been completely neglected in a large number of publications in this field. Hence, the standard GA implementations performed misleadingly worse than for example dedicated tabu search implementations. Proposals for a better exploitation of the GA's potential with respect to genome coding and mutation were made without imposing additional complexity. In extensive test runs the superior performance of the proposed problem-oriented GA in comparison with the most common GA version and the other heuristic methods has been revealed.
- Eventually, an entirely new heuristic for the HW/SW partitioning problem has been introduced. A thorough analysis of its behaviour related to graph properties locality and parallelism revealed a strong performance for a distinct subset of system graphs that are typical in the field of embedded system design. For this subset and the mapping problem with a limited number of implementation alternatives per process, the proposed RRES algorithm even outperforms more sophisticated techniques like genetic algorithms and tabu search. Its dependency on the locality property of the system graph and the parallelism present in the architecture graph has been outlined.

A number of interesting topics for the future based on the framework accomplished in this thesis can be identified.

With respect to very recent developments in the graph representations dedicated to the signal processing domain of embedded systems, namely the parameterisable cyclo-static SDF graphs proposed by Saha [124], it has to be awaited whether these will be adopted by the electronic system design community and perhaps become integral part of commercial EDA tools. Since they offer dynamic and run time reconfigurable data rates, they seem to be appropriate to represent the increasing complexity of modern signal processing systems. But to what extent a reliable scheduling and partitioning engine based on a detailed platform model can be developed for such graph representations has to answered in the future.

As already stated at the end of the last chapter, the consideration of the uncertainties imported by the estimated characteristics of the design and its graph-like representation is certainly of significant relevance. Concretely, it shall be possible to provide a reliability metric and/or a possible variance for any estimated value in the partitioning scenario. Another aspect in this context is the scalability of some code segments especially with respect to hardware implementations. Whenever a code segment with loops offers adjustment of the unrolling factor, then any integral division of the loop count into parallel paths is theoretically possible. Hence, the trade-off between execution time and occupied gates is rather a function, than a fixed set of value pairs. How these degrees of freedom can be incorporated into the model of computation for any design space exploration framework in a tractable manner is yet unsolved.

The most pressing extension constitutes the deployment of an automated platform optimisation. In the framework as it is, a designer can already assemble arbitrary heterogeneous architectures, a feature entirely absent in any other partitioning engine, and then analyse how a system's functionality expresses itself on this platform. However, there is still a manual step involved: the assembly of the next architecture, in case the last attempt turned out to be unfavourable. The development of a regulation and control apparatus, in which the architectural degrees of freedom can be assigned a-priori, and the implementation of an optimisation engine that gears the outcomes of the partitioning and scheduling algorithms towards a more favourable platform, embodies the vision of a completely automated design space exploration on system level.

APPENDICES

A. THE OPEN TOOL INTEGRATION ENVIRONMENT

To obtain a good impression of the bigger context, in which the contributions of this work are residing, it can be of interest to survey the related research of this Christian-Doppler Laboratory.

Hardware/software systems is the common term generally used to identify the heterogeneity of aforementioned electronic designs [31]. Basically, it refers to systems composed of programmable devices as general-purpose or digital signal processors (software part) and dedicated custom data paths (hardware part). Hardware performs a specific function, usually with a much higher throughput, but is expensive and inflexible, whereas software features low development efforts, is easily adjustable, but is rather slow and often much power consuming. The term hardware/software codesign has been introduced by Franke and Purvis [43] as "the system design process that combines the hardware and software perspectives from the earliest stages to exploit design flexibility and efficient allocation of function". Wolf emphasised early that "the hardware and software must be designed together to make sure that the implementation not only functions properly but also meets performance, cost, and reliability goals" [144]. The following paragraph indicates that the phrase "..must be designed together.." is persistently hard to achieve.
In Figure A.1 a rough breakdown of the design process for hardware/software systems is depicted to illustrate the aforementioned fragmentation. Due to the large scope and the extremely heterogeneous nature of modern wireless communication devices, their development suffers from many incompatible system descriptions. On its way to the final product the design meanders towards completion passing very dissimilar development stages. At the very beginning the research team excogitates new algorithms and applications, that are described and tested in high level languages as C/C++, UML, or Matlab. The system design team elaborates on the platform structure and IP selection. In this stage a conglomerate of tool suites, languages, and standards has to be considered: instruction set simulators (ISS) for microprocessors (μPs), bus and bridge models, memory models, power simulators,

etc. Herein, a besetting variety of dedicated tools ensures a variety of experts. The last

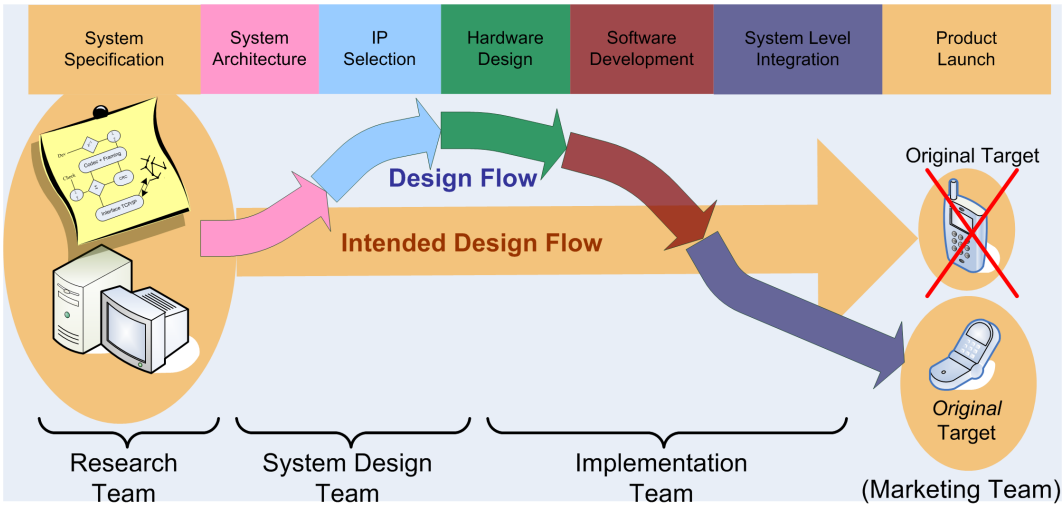


Fig. A.1: The meandering of the electronic design process.

stages are embraced by the implementation team undertaking the actual programming, i.e. rewriting parts of the system's functionality in VHDL or Verilog for application specific integrated circuits (ASICs) or custom data paths on an field programmable gate array (FPGA). VHDL code is then synthesised to register transfer level (RTL) followed by a place and route process. Other parts are transformed and assembled onto μPs, as DSPs or RISCs (reduced instruction set computer). Platform drivers and register tables have to be developed. Due to the heterogeneity, real time operating systems (RTOS) have often to be programmed from scratch. Eventually, the assembled product may deviate from the original target and a fourth team has to step in to explain that it essentially does not so.
Apparently, system descriptions are herein constantly rewritten by corresponding experts and are converted into other, locally more suitable, forms. The outcome are serious communication obstacles between design teams in backward and forward direction leading to a dramatic increase of verification effort of up to 70% of the overall development time [10, 59].

The initial framework concept to tackle the flaws in current system design manifests in the establishment of the Open Tool Integration Environment (OTIE) [13, 16]. Therein, a flexible, scalable, robust, and secure implementation of OTIE is presented, based on a design database (MySQL [112], CVS [41]), providing a single, central repository for all refinement information in the design process. Instead of non-realistically aiming at the development of a complete stand-alone signal processing work suite and thus competing with large-scale companies like Synopsys, CoWare, Mentor Graphics, and The MathWorks, this framework in-

terfaces the popular *existing* EDA tools on the market, connects their domains with powerful tools targeting the missing design tasks, and additionally opens paths to mutual interconnections. In Figure A.2 this concept is illustrated. The pictograph in the centre represents the

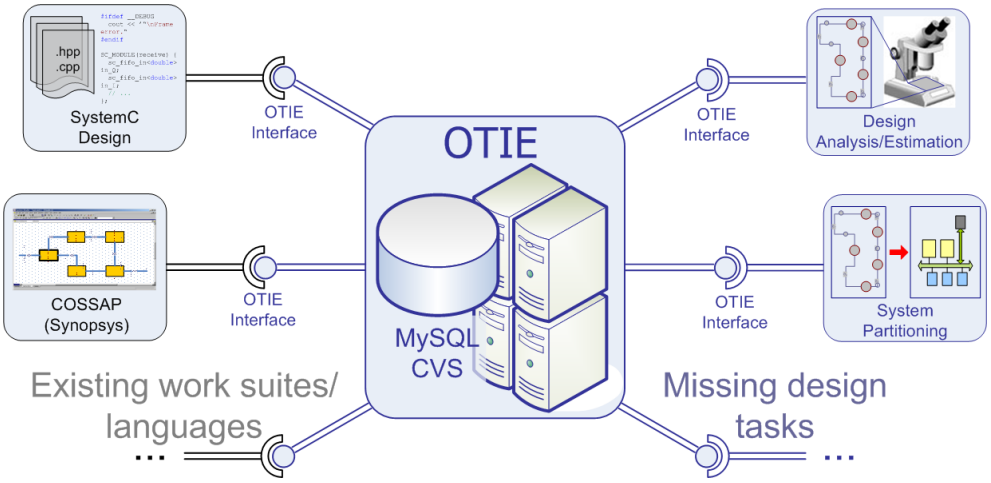


Fig. A.2: The concept of the Open Tool Integration Environment (OTIE).

design database that captures the information of the complete design process. On the left side existing tools/languages, COSSAP and SystemC [116], are depicted, which have already been interfaced [83, Knerr et al.], [14] (COSSAP has been replaced by Synopsys System Studio [133], which is also not longer available). On the right side two of the missing or insufficiently covered design tasks are depicted.

The most fundamental ingredient for a consistent design methodology establishes a Single System Description (SSD). An elaborated solution of an SSD is the implementation in the form of a MySQL [112] based design database. A database representation is not bound to specific language constraints and thus offers great flexibility in capturing the miscellaneous aspects of a design. Additional advantages of the database approach are fast access, data security by the capability to grant permissions to the developers, a high popularity as well as compatibility with major Data Base Management Systems (DBMS) from Microsoft, IBM, and Oracle.

In our framework OTIE, these obligations are illustrated by Figure A.2. It depicts the database and concurrent version system (CVS) surrounded by the required tools each with dedicated interfaces to incorporate the various existing EDA tools and languages, while being open for incorporation of tools for missing design tasks. During the design flow various design teams provide inputs, such as desired system behaviour and structure, constraints, tool options etc. Also, the designers receive outputs, like status of the system description, results of simula-

tions, estimates of hardware cost, timing and so on. Typically, the outputs of the database are handed to the tools, which present them in form of their graphical user interfaces (GUI) to the designer. Some of the tools supported by OTIE are commercially available, favoured by the various design teams, while others are specially written to perform missing tasks, usually performed manually by designers in the past. As long as some design steps are not covered by available tools, for example HW/SW partitioning, a database modification tool is available, simply allowing the designer to enter manually derived values. The database is thus enriched and the system description is refined on its way to implementation. Note, that the database system does not require a specific order of which various tools need to be performed. For example, some designers prefer to perform floating-point to fixed-point conversion after the HW/SW partitioning, while others apply it first. As long as the succeeding tool is provided with sufficient information, it can be started. Such an open environment has not only the advantage that new commercial tools can be incorporated, but it also provides a realistic platform to investigate the performance of new research tools. A possible design flow sequence for example would be loading a `SystemC` description into the database. Furthermore, design/analysis and estimation would be performed, which enriches the content of the database with the results of the characterisation process. Finally, system partitioning may be performed, which exploits the properties of the system analysis.

At the moment several design tools have been developed, that are integrated within this design environment: automatic partitioning of the system into HW/SW parts [86, Knerr et al.], automatic generation of virtual prototypes [83, Knerr et al.], an environment called *fixify* is available for performing the task of fixed-point to floating-point optimisations [17], and several design space exploration tools [62, 64, 66].

B. TYPICAL EXAMPLES OF ARCHITECTURAL COMPONENTS

Throughout the thesis, architectural components are referred to, such DSPs, FPGAs, ASICs, etc. This chapter sheds some light on their specifics, in case the reader is not familiar with their architectural implications.

B.1 General-Purpose Processors

Although GPPs are not considered as viable choice in embedded systems a short description is given to round up the picture. These processor types are all-rounders on which nearly any application can be executed with a medium performance instead of being optimal for just a single one. Workstations, PCs, servers, etc. are typical candidates for a deployment of these processors. The steep requirements on flexibility and processing speed necessitate very complex circuit structures with (super-)pipelining, branch prediction, hierarchical caching structures, and superscalar scheduling by prefetching instructions. The execution time of a characteristic code block varies therefore, as it is dependent on a number of dynamic effects. For real time systems with strict deadlines on certain parts of the functionality, these processors are normally inappropriate. Another obstacle for the deployment of GPPs in embedded systems is the large power consumption and the tedious and time-consuming interface design for I/O and memory access due to the aforementioned circuit complexity.

B.2 Digital Signal Processors

DSPs are processors dedicated to a specific application domain of digital signal processing, e.g. mobile communication, image processing, audio/video applications. With respect to the general instruction set, they offer very much the same possibilities as general-purpose processors but with less facets and simpler circuitry. Their big advantage consists in *additional* instructions for which its circuitry is optimised. Relevant traits for DSPs are amongst:

- Combined multiply-accumulate (MAC) operations.
  In a single instruction cycle a multiply operation of two operands is interlinked with a subsequent accumulation of the result. This instruction has a direct realisation in hardware circuitry in a DSP for floating-point or fixed point number formats.

- High jump predictability and zero-overhead loops.
  A humble level of code branching and fixed loop count variable is exploited by special registers, in which start and end address and the loop counter is stored. Every iteration through the loop body triggers the counter's increment or decrement and the subsequent comparison with the end condition, thus not imposing any overhead due to loop controlling.

- Specialised addressing techniques.
  DSPs provide address generators that are capable to increment or decrement the address pointer by a programmable step width in parallel to the actual instruction processing. Two relevant applications are the circular address scheme, which facilitates filter implementations and bit-reverse address schemes for e.g. Walsh-Hadamard or Fast Fourier Transforms.

Many embedded systems comprise DSPs with fixed-point numeric formats, since a fixed-point arithmetic logic unit (ALU) is much faster than a floating-point ALU given the same chip area. However, the transition towards fixed-point formats additionally complicates the design due to quantisation noise, rounding and overflow errors.
Nowadays, C-compilers exist for most of the DSPs on the market, but crucial functions may still be designed in assembler to ensure a better exploitation of the specific architectural features of the DSP. For many applications, as e.g. in the image processing domain, time critical code parts that have been manually optimised in assembler can be embedded into C routines.
The digital signal processing domain gained significant attention due to the revolution in mobile communications. Therefore, a large variety of different DSP cores emerged with manyfold innovative architectures [47]: for instance multiple DSP platforms, very large instruction word (VLIW) DSPs, and desktop DSPs.
The multi-DSP strategy earns special attention, since it is conceptualised for the multitude of present standards and protocols in the embedded system domain. For instance, the meanwhile common combination of multimedia applications, comprising huge amounts of data and complex algorithms, merged with parallel signal processing for mixed UMTS, GSM, and Bluetooth standards easily overstrains the capability of a single DSP. Examples for DSP structures that enable immediate bidirectional communication to other DSPs have been released ten years ago, thus bypassing the need for external buffers or additional synchronisation,

e.g. ADSP21060 from Analog Devices or TMS320C40 from Texas Instruments. These DSPs prepared the ground for more flexible systems for specific purposes.

B.3 Microcontrollers

As the name suggests a microcontroller (μC) is dedicated to control flow dominated applications like protocols that are characterised by a large number of branches, internal states, and boolean logic operations. The data throughput as well as the arithmetic operations do

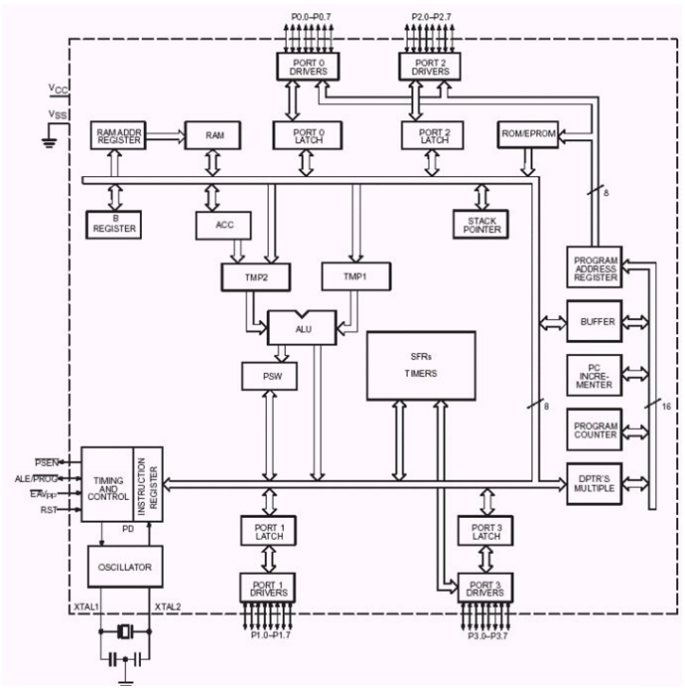


Fig. B.1: Block diagram of a state-of-the-art NXP 80C51 microcontroller [115].

not play a major role. Typically, μCs are used for interrupt handling and support a very fast context switching often seen in protocol state machines. In other words, the current program context is realised completely in the RAM, so that in the case of an interrupt the program address pointer is simply set to a new address.

A classical example is the micro controller 80C51, optimised for very small code size, in Figure B.1. Although, this design has been superseded by more innovative versions from all big manufacturers like Texas Instruments, NXP, Maxim IC, or Intel, their characteristic traits remained widely identical. The main part of its chip area is reserved for memory blocks. It provides many units integrated in one chip: CPU, RAM, ROM, serial ports, analog-digital

converters, timers, etc. The word width is 8bit and it offers several power saving modes and two level interrupt priorities.

B.4 Application Specific Instruction Set Processors

These μPs are even more customised to their specific application domain than DSPs and micro controllers. The key idea is the application-directed generation of a *programmable* device, whose instruction set and data word widths have undergone a fierce optimisation towards its purpose. As indicated in Figure 2.2, ASIPs occupy the location with the least flexibility and the highest performance in the software domain.
Since ASIPs are by definition application specific, it is difficult to classify them by their commonalities. Usually their instruction set includes operator concatenation as MAC operations, or vector arithmetics. Similar to their larger siblings, the DSPs, their circuitry exploits parallelism of address calculation and data operations. On the contrary, ASIPs usually dispense complicated caching schemes and reduce the pin number as far as possible to enable smaller chip sizes. In Figure B.2 a small ASIP is depicted with a classical Harvard architecture, i.e.

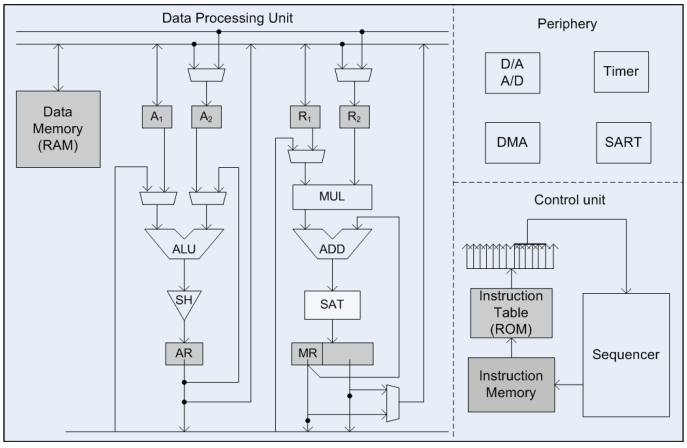


Fig. B.2: ASIP with Harvard architecture.

separated data and instruction storage to enable parallel access to both (in contrast to a von Neumann architecture). This ASIP features a multiply-accumulate circuitry and a single ALU and is designed to process any instruction within a single cycle.
The development of optimising compilers, debuggers, and linkers for ASIPs has long been subject to intense research. In recent years, a design group from RWTH Aachen developed a mature tool suite for ASIP design called LISA [60, 146], which is now commercially available in the portefeuille of CoWare [30].

B.5 Field Programmable Gate Arrays

Field programmable gate arrays belong in our notion to the hardware domain, although being programmable as the name suggests. A regular arrangement of configurable logic blocks (CLB) is programmable by adjusting the interconnects between them in order to duplicate basic logic gates as AND, OR, XOR, memory or more complex combinatorial functions. The CLBs contain look-up tables, multiplexers, and flip-flops, whose structure usually differs

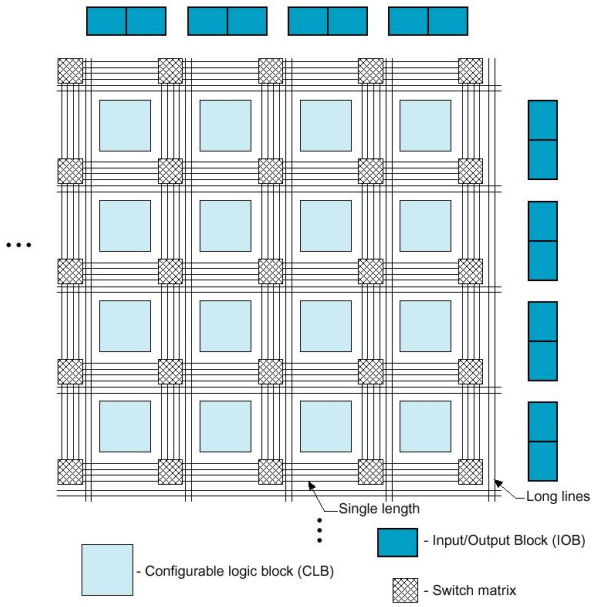


Fig. B.3: Structure of an FPGA.

widely to offer high flexibility on a single FPGA. The interconnection network occupies the major portion of the chip area of up to 90%. I/O blocks surround the CLB grid.
The FPGA programming is undertaken by the system designer in the *field*, thus the name. It is in general distinguished between one time only programming of FPGAs with anti-fuse switches and reconfigurable programming of FPGAs with SRAM switches. In the first case the interconnects and configuration of the multiplexers are burned onto the die to establish a connection (thus *anti*-fuse). Eventually, these FPGAs resemble ASICs, as their behaviour is permanently determined. The configuration of SRAM based FPGAs is accomplished by setting variables in the SRAM units that determine the interconnects and multiplexers. In modern FPGAs at every power up of the FPGA the configuration is loaded from an EEPROM.
The development of FPGA circuitry resembles very much the development of ASICs. Classical hardware design tools are utilised to develop schematics and netlists of integrated circuit ele-

ments (gates, flip-flops). The FPGA vendor usually offers integrated tools for the schematics, which automatically transpose the netlist into the configuration data and eventually configures the FPGA.
Rapid prototyping is popular deployment of FPGAs. Since verification of integrated circuits via simulation on a PC or a work station is normally a very slow process, FPGAs offer the possibility to emulate the designed circuitry in a very fast manner. In embedded systems SRAM based FPGAs found strong usage in the end product as they offer a much higher throughput than microprocessors and are reconfigurable in the field. Base stations for mobile communications are popular examples, as communication standards and protocols are constantly rewritten and new versions pop up nearly once a year. Additionally, the signalling of these protocols requires computationally intensive processing.

B.6 Application Specific Integrated Circuits

Application specific integrated circuits have been the answer to any application that demanded either high performance computation and vast throughput and/or that is sold in large quantities. ASICs are completely customised to a single product and are sold by the manufacturer normally only to the ordering customer. Digital ASICs integrate a large number of logical and arithmetical functions, which are permanently burned on a single die. ASICs outperform any other architectural component with respect to latency, throughput, power consumption and chip size. Naturally, such a functional component can never be used for anything else. In mobile communications ASICs adopt typically the most computation intensive functions like Viterbi or turbo codecs.

Figure B.4 shows the top level schematic of a rather small ASIC for Viterbi decoding with about two thousand gates. The interconnects and word widths are fixed as well as the internals of the eight computation blocks. The lower left part contains the traceback, the path metric unit, and the add-compare-select (ACS) circuitry and the lower right block contains the branch metric unit with the squarer-adder circuitry. The interconnect of this circuitry is depicted to illustrate the rather simple IC composition strictly geared towards a fast streamed computation within a single cycle.

B.7 Communication Infrastructure

The interconnection of heterogeneous structures assembled by a selection of the aforementioned processing elements is likely to be similarly multi-faceted. A brief overview shall illuminate the most common types:

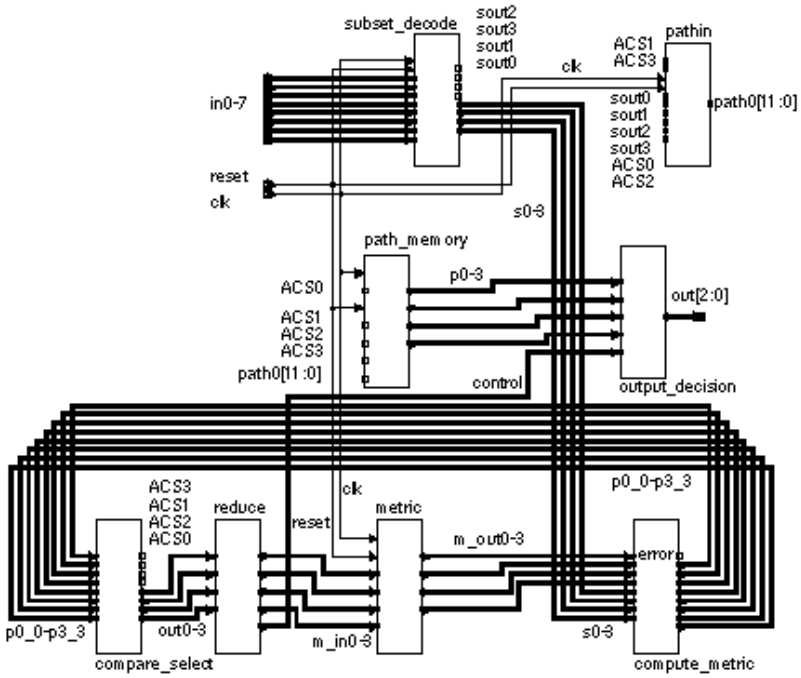


Fig. B.4: Core logic of an Viterbi decoder ASIC [128].

1.) The simplest connection to transfer data is performed over a signal wire on gate level, herein referred to as *direct* link. Candidates are the data heavy connections with a high throughput ($\approx$ Gbytes/s) and low latency between the antenna subsystem and the first digital signal processing units, typically implemented as hardware devices such as ASIC or FPGA. The synchronisation between transmitter and receiver is not managed and has to be preconceived in the unit design, but the area and control logic overhead is naturally minimal.
2.) A *FIFO* based interconnection permits to push bits into a memory structure that are not immediately processed by the receiving device but stored for a later time instant. Additional silicon area is required for the memory according to the FIFO depth and for the glue logic to control (non-)blocking read/write accesses. FIFOs are very versatile to connect hardware as well as software devices due to relative small overhead and fast access times ($\approx$ Mbytes/s).
3.) On-chip *point-to-point* (PTP) connections superpose another layer of complexity as a protocol manages the handshake between the communicating resources including message framing, control flags, data bits, and often error correction. PTP is considered to connect ports of DSPs, μCs and rarely for FPGAs, but usually not for ASIC blocks.

When the number of ports of one or several processing elements is limited, or the number of direct or FIFO interconnections causes a dramatic increase in chip area, bus based solutions

come into play to integrate non-critical communication links. A bus structure collects several data connections on the same bundle of wires, dispatches the messages from transmitter to receiver and arbitrates collisions when two or more resources start transmitting simultaneously. A popular bus structure in embedded systems is the Controller Area Network (CAN) [73] for noisy industrial environments. The introduced control overhead increases due to logic for bus arbitration, scheduling, message acknowledgement and error recovery. Transmitted messages have to contain information concerning their priority, destination, current size and cyclic redundancy check code. A typical throughput for a CAN bus 2.0 lies in the range of up to $1\text{Mbit}/s$. Another popular on-chip bus architecture is the Advanced Microcontroller Bus Architecture (AMBA) from ARM Techn. [6] that offers a multitude of versions varying in throughput, bandwidth, gate count and power consumption.

B.8 Academic and Commercial Co-Design Frameworks

At the beginning of the co-design process, a design language is generally used to embody the semantics of a system prior to mapping it onto an architecture. The design language must provide a way of describing the behaviour of the components in the system irrespective of whether they will be mapped to software or hardware. It should support structure and hierarchy, and should include the ability to incorporate the description of design constraints such as dependency, timing, concurrency, and task scheduling between hardware and software. Different design frameworks use different system design languages.

B.8.1 Design Languages

At the highest abstraction level, languages like the Unified Modelling Language (UML, `www.uml.org`) and the Specification and Description Language (SDL, `www.sdl-forum.org`) are available, providing modelling support for nearly any application type, from pure software engineering to microelectronics. Traditionally, systems are not formally modelled at these high levels, but the modelling starts when all the parts of the system and their interactions have already been explored analytically to some extent. Hence, the `C` and/or `C++` languages have a more established base in algorithmic modelling, starting at a slightly lower level of abstraction. It is interesting to note that the `C/C++` languages are used in two parts of the design process. At the algorithmic level, they are used to model the behaviour of the system, i.e. the algorithm itself. At the timed functional level, `C/C++` is widely used for implementation of the software components of the embedded system, usually running on a standard processor.

SystemVerilog [134] is an extension to the Verilog HDL, enhancing the abilities of the core language into higher abstraction levels. For instance, typical concepts from object oriented programming have been adopted. Unlike Verilog, *SystemVerilog* possesses constructs for unions, structures, classes, inheritance, dynamic arrays, sophisticated loop control, and new/-custom data types. With these enrichments, the aim of *SystemVerilog* is to help designers describing the relatively abstract architectural structure through conceptual interfaces, rather than describing concrete, implementable functions connected through registers and implementable data types. This enables the designer to refine the abstract model down to the customary register transfer level design without having to migrate languages.

`SystemC` [116] is based directly on the `C/C++` languages and hence inherits their strengths both at the algorithmic and implementation levels, while attempting to create new support (which was so far missing) at the architectural level. The `SystemC` language was divided from its conception into three parts: `SystemC` 1.x for modelling at the implementation level, `SystemC` 2.x for architectural modelling, and `SystemC` 3.x, for system level modelling. The latest available version of SystemC is version 2.2, released in April 2007. The underlying `C++` library supports hardware related concepts as concurrently running processes, a sophisticated timing, and communication channels like FIFOs, buses, memory models, etc.

SpecC [138] is an alternative approach to describe structures on system level. It is basically an ANSI-C extension, enriched with the same concepts for concurrency, hierarchy, communication and timed behaviour. Similar levels as in `SystemC` are covered, like untimed functional, architectural, transaction, and register transfer level.

Esterel [18], SDL, and SpecCharts [113] are real-time capable languages that are based upon a model for conditional event-based state transitions. Hence, it is often the method of choice to represent control dominated system components. The realisation of algorithmic structures is usually performed via encapsulated `C/C++` constructs that are embedded in the distinct states.

Handel-C [24] is a programming language also based on ANSI-C. A large number of the types, operators and statements in ANSI-C are also available in Handel-C. However, it is a programming language that offers concurrency in the system abstraction and that can be used to compile directly to configurable hardware - e.g. FPGAs - by creating the information needed by FPGA implementation tools. For this purpose it contains additional statements and expressions handling e.g. synchronisation, timed computation, bit pattern formats, deterministic parallelism of operations, timed signals and channels, multiple clock domains, and memory units (RAM, ROM).

A variety of other languages are available at the implementation level, each associated with a particular implementation option. Assembler should be added as it has still not gone extinct

in embedded system design. It should however be noted that the abstraction level offered in assembler is very low, and hence any large parts of code (such as entire applications) are not modelled well in this language. Nevertheless, the precise control, which this low abstraction level brings, affords the designers the opportunity to write small, hardware-close pieces of software (such as device drivers) which are nearly optimal.

B.8.2 Co-design Frameworks

A number of academic design environments emerged over the years in the field of embedded systems. Early mentionable approaches originating from academia are POLIS, VULCAN, COSYMA, Ptolemy, CASTLE and Chinook.

The CASTLE environment [143] is basically a comfortable profiling tool to support the designer's decision regarding the system structure. CASTLE yields feedback on execution time spent in specific functions, operations, and memory accesses. The designer iteratively alters the system structure towards the desired behaviour, hence not being released from the optimisation process.

Chinook [27] is a hardware/software co-synthesis tool for control dominated, reactive systems under timing constraints. Chinook focuses on the synthesis of the communication infrastructure. Allocation and partitioning is supposed to be performed by the designer at system level, and Chinook transforms the source code to a lower level, namely register transfer level.

COSYMA [39] is a co-design system that is targeted to systems based around a standard RISC processor core. Initially, the design, described in `Cx`, a `C` derivative featuring the concept of concurrency, is considered to be mapped completely to the RISC. Iteratively, time critical function blocks are retargeted to dedicated hardware blocks that act as coprocessors for which the HDL code is automatically generated. Vice versa, the VULCAN co-design system [51] starts from a virtual hardware setup captured in `HardwareC`. Silicon area is reduced by moving non-critical parts to a software implementation.

Esterel is the design language accommodated in the POLIS environment [12]. Designers are guided by feedback mechanisms provided by POLIS to partition the functional units in hardware and software parts. As the designer's interaction is required for any decision, this process is unviable for large designs. Moreover, the POLIS design flow is not unified in a sense that it requires mechanisms located in other environments, e.g. adopting Ptolemy features.

A first version of Ptolemy has been published in the late eighties [98], called Ptolemy classic, which has been replaced by Ptolemy II later on. Ptolemy classic has been designed in `C++` to

support a variety of models of computation for heterogeneous designs, such as synchronous data flow (SDF), process networks (PN), and discrete events (DE). Ptolemy II supports even more models of computation and is Java-based to exploit remote control and concurrent processing on several platforms.

Not all the features offered by academic approaches found their way into the commercial world. The following approaches survey those that are most commonly used by the industry.

The Matlab Simulink environment of The MathWorks [137] can be used for functional specification and algorithmic analysis in the co-design flow as well as for hardware synthesis via the HDL Coder toolbox. For the first aspect, a library of Simulink blocks accompanied by a corresponding analysis toolbox is used to evaluate parameter choices. The library components are parameterised to provide a high flexibility and comfortable implementation. Automatic partitioning routines do not yet exist. Software tasks are transferred to the The MathWorks Real Time Workshop to generate C codes, while the hardware tasks are processed by the HDL Coder's Module Generator to generate VHDL netlists.

Synopsys is one of the major EDA tool vendors offering automated support for many parts of the design process. Synopsys developed a commercial environment for tool integration, the Galaxy Design Platform [132], which is based on a single description of the system, implemented as a database. It eliminates the need for rewriting system descriptions at various stages of the design process, covers both the design and the verification processes and is capable of integrating a wide range of Synopsys commercial EDA tools. An added bonus of this approach is the open nature of the interface format to the database, allowing third-party EDA tools to be integrated into the tool chain, if these adhere to the interface standard. A serious limitation of this approach is the lack of support for any other part of the design process than the implementation level, and only for hardware components.

The Virtual Component Co-design (VCC) from Cadence [23] is a design framework built around POLIS to enable the simultaneous development of hardware and software modules as well as the integration of IP. An abstract infrastructure is provided that allows for the integration of system parts modelled in different design languages like C/C++, SDL, Matlab m-files, etc. by encapsulating those in virtual components. The communication interfaces between these components are automatically generated thus facilitating the design space exploration. However, all decisions with respect to the platform infrastructure have to be performed manually. Another restriction is the lack of static code analysis and profiling features within this environment.

CoWare is another major player in the EDA tool market. In the late nineties the tool suite Napkin-to-Chip (N2C) emerged as first release dedicated to hardware/software co-design. In close collaboration with ARM a framework was assembled that focused on the assembly of

bus (AMBA) and memory structures for heterogeneous systems and data transfer analysis. In 2003 the N2C's successor ConvergenSC has been released in which also instruction set simulators for the ARM 7 and ARM 9 core families were incorporated. Still this release focused on the assembly of the platform infrastructure and did not support the design of the functional parts themselves. After CoWare acquired the Signal Processing Worksuite (SPW) from Cadence and LISA [60] from LisaTek in 2004 another tool suite has been released, the CoWare System Designer. In principle, the CoWare portefeuille now covers a huge range of abstraction levels in the design process. However, the seamless integration of these manifold concepts that originate from completely separated design frameworks into a mature tool has still not be accomplished. For instance the transition from algorithmic level (SPW) to architecture level (ConvergenSC) is still weakly automated and introduces significant coding overhead for the designer.

C GRAPHS IN EMBEDDED SYSTEM DESIGN

This chapter provides information necessary to reproduce and verify the algorithms on a similar graph set. In the first section a few real examples of task graphs in embedded systems are given to underpin the considerations about the typical properties found in this field. The following section describes the graph generation method.

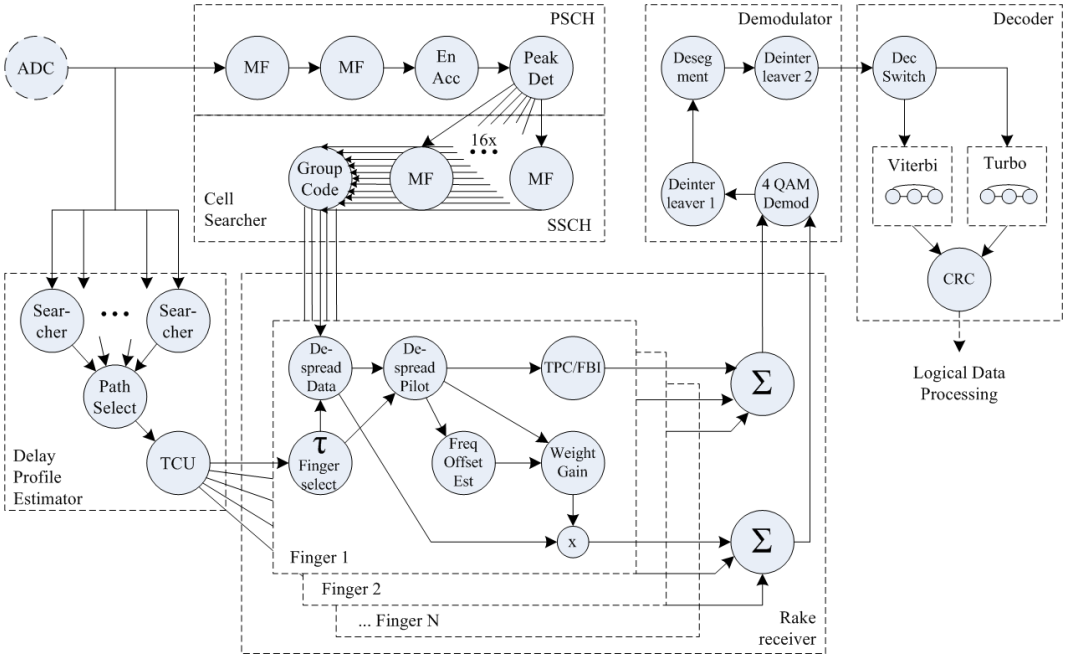


Fig. C.1: Part of the signal processing for an UMTS receiver.

C.1 Typical Graph Structures in Embedded Systems

In this section a few graph examples are given to illustrate that system graphs that are typical for human made graphs in embedded systems can be isolated by distinct properties described in Section 4.1.

The first example is a part of the digital signal processing of a UMTS baseband receiver chip

after the analogue to digital converter. The data flow of the most computation intensive parts is depicted including the cell searcher with primary (PSCH) and secondary synchronisation channel (SSCH), the delay profile estimator, the rake receiver, demodulator and the two deinterleaver stages, followed by the decoder. The Viterbi and turbo decoder are not presented in detail due to limited space. Although, some parts could not be clearly outlined, like the 16 matched filters for SSCH module, and the complete array of fingers of the rake receiver, it becomes obvious how digital signal processing systems express themselves as graphs with distinct properties when assembled by human designers. The graph properties of this example are: $|\mathcal{V}| = 127$, $|\mathcal{E}| = 226$, $\rho = 1.78$, $r_{\text{loc}} = 1.23$, $\gamma = 5.77$, $\widehat{\gamma} = 16.2$.
Relevant figures were obtained for this design to enrich parts of the system graph for gate count and code size. These values and their respective ranges in industrial deployment are important for the generation of realistic system graphs in the next section. A typical value set for the Viterbi block implemented as ASIP clocked with 100MHz is 26.5kgates in NAND2 equivalents, a code size of 2.6kbytes of optimised code for this ASIP, and a core power dissipation of 48mW. For instance the first two matched filters in the cell searcher are both 16 tap accumulators of 16bit values accompanied by $256 \times os$ registers of length 32bit, with os being the oversampling factor ($os = 2..8$). The latter 16 matched filters in the cell searcher correlate for the group code on a half-chip basis. Table C.1 and C.2 list some more values for the Cell Searcher and the Delay Profile Estimator sub modules. Note, that the huge cycle counts results, as they are measured per frame, which equals $= 15$ slots, with 2560 I,Q pairs (chips) per slot.

Cell Searcher	gc_{op}	gc_{reg}	gc	et_{HW}	et_{SW}
2 MF	9,850	6,895	16,742	100k	146k
Energy Accumulate	2,761	924	3,685	14,752	24,970
Peak Detection	659	352	1,011	5,637	12,044
16 MF	24,855	17,398	42,253	252k	353k
Group Code	377	263	640	3,854	4,817

Tab. C.1: Some characteristic values for the Cell Searcher.

Delay Profile Estimator	gc_{op}	gc_{reg}	gc	et_{HW}	et_{SW}
16 Searcher	5872	4112	9984	58,816	77,648
Path Select	266	156	523	2,675	3,852
TCU	98	67	166	529	677

Tab. C.2: Some characteristic values for the Delay Profile Estimator.

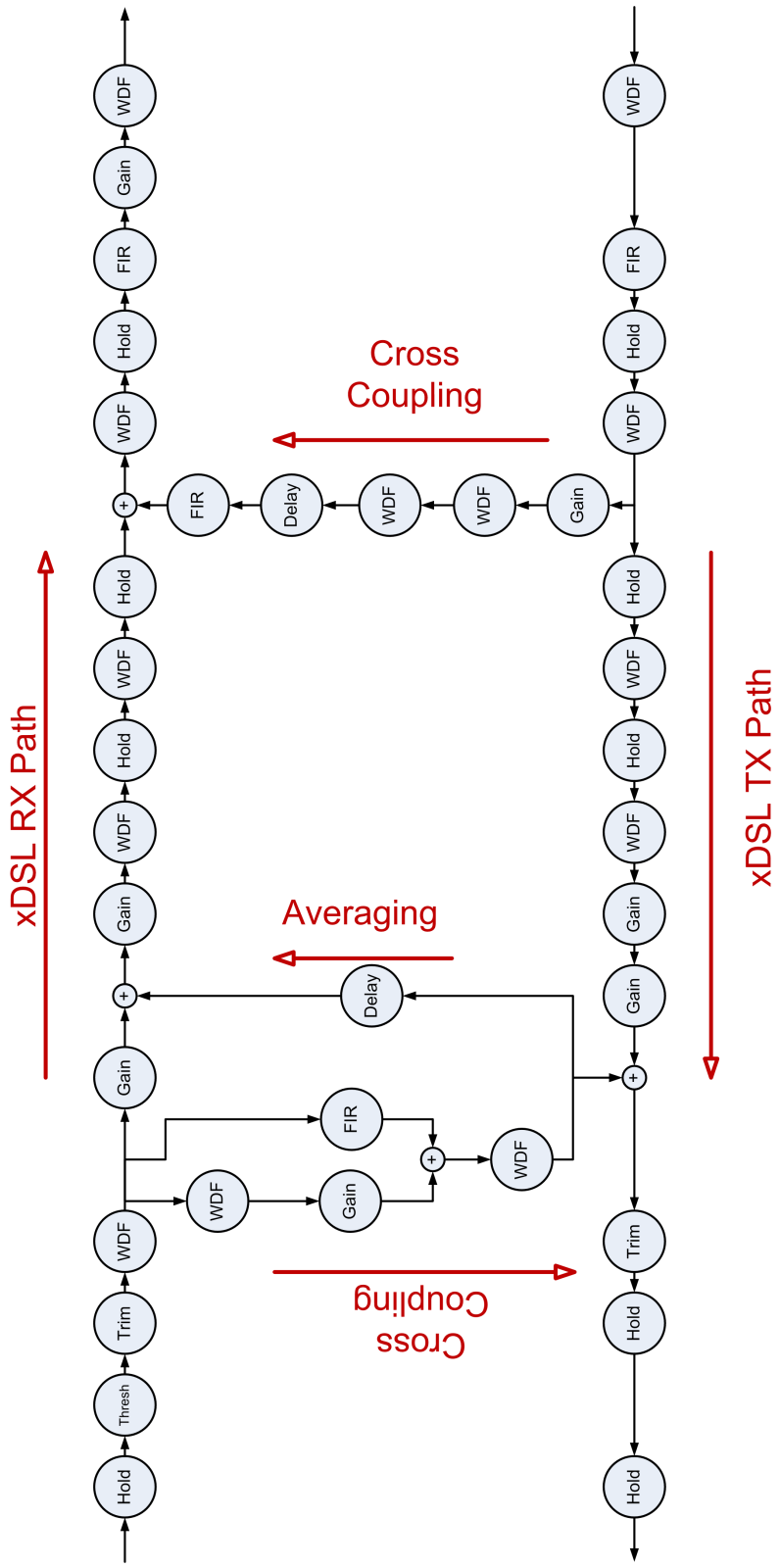


Fig. C.2: Part of a data flow graph of an xDSL Modem.

The second example is a cut out of the data flow diagram of the digital signal processing part of an industry designed xDSL (symmetric and asymmetric DSL) transceiver [72]. This modem handles the high speed wide band transmission utilising the classical two-wire copper connection of the telephone network. *Symmetric* DSL exploits the complete frequency band for DSL transmission, hindering any classical analogue utilisation, whereas *asymmetric* DSL allows for combined classical telecommunication and digital high speed transmission. Such a modem that has to be installed with the customer is a typical example for an embedded system of a small form factor. In Figure C.2 the TX and RX path are depicted. To the left side the front-end to the network has to be imagined, and to the right the DSL protocol processing, depackaging and packet interpretation, has to be assumed. Nearly half of the depicted vertices represent either finite impulse response (FIR) filters and Wigner distribution function (WDF) filters. The remaining vertices represent of gain control, thresholds, decimators, hold-and-sample, and so forth. The graph properties of this example are: $|\mathcal{V}| = 38$, $|\mathcal{E}| = 44$, $\rho = 1.16$, $r_{\mathrm{loc}} = 1.23$, $\gamma = 2.11$, $\widehat{\gamma} = 18.4$.
This design has been originally implemented on a single StarCore DSP with a clock frequency of 300MHz for which the following cycle counts have been annotated:

Function	Taps	Aliases	et_{SW}
MAC		Trim, Gain	6
SUM		Thresh	6
WDF	1		9
WDF	2		13
WDF	3		16
WDF	5		22
WDF	7		28
WDF	9		34
FIR	2		11
FIR	7		21
FIR	11		29
MOV		Hold	5
MUX		Delay	6

Tab. C.3: Typical cycle counts for filter code segments on a DSP.

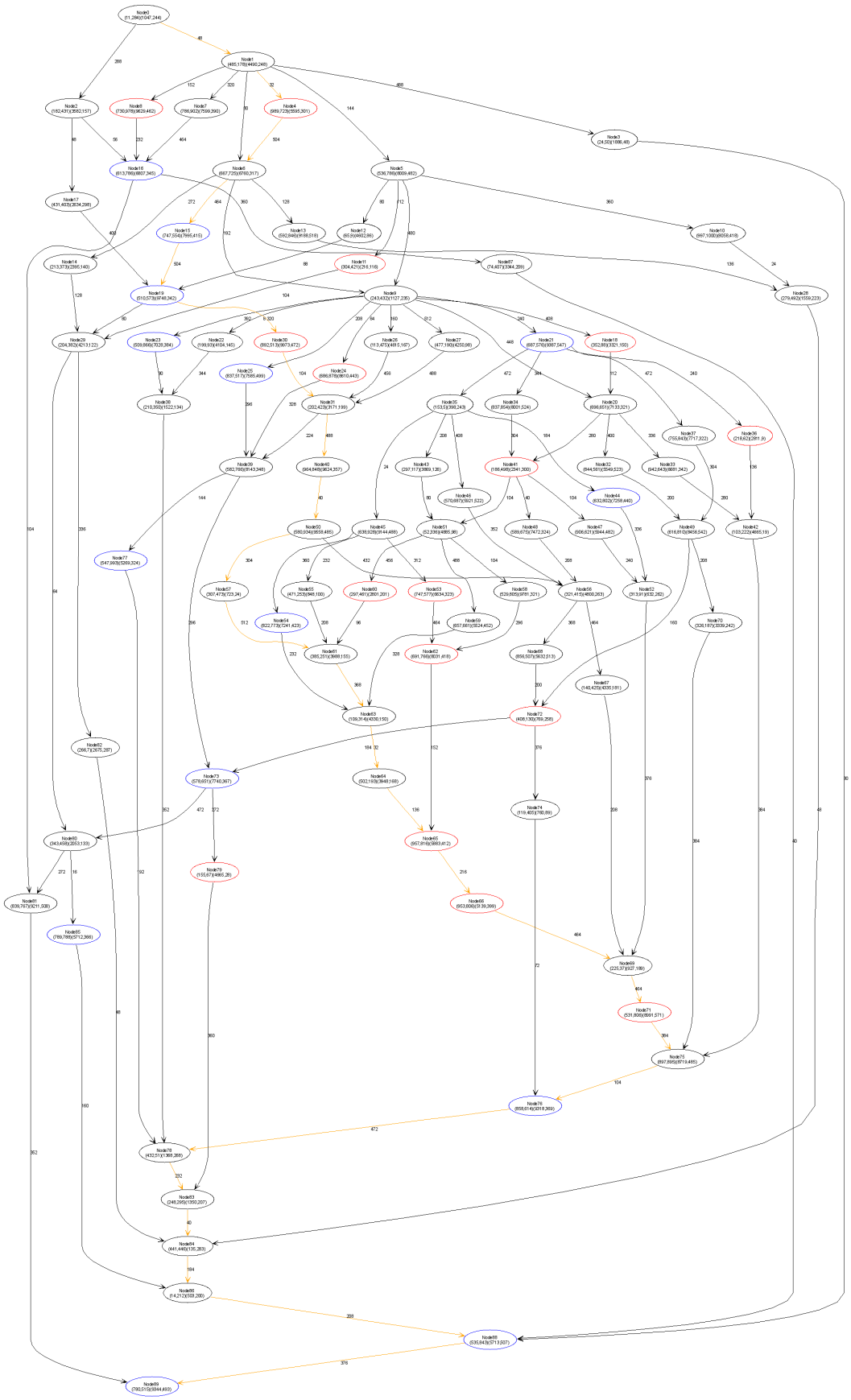


Fig. C.3: A realistic robot control process graph [79].

The Kasahara laboratory established a benchmark graph library, the standard task graph (STG) set, dedicated to analyse the performance of multi-resource scheduling and partitioning algorithms. The majority of these graphs are generated according to their proprietary ruleset, but three realistic examples are given as well: a robot control system, a sparse matrix solver, and a SPEC process graph. In Figure C.3 the smallest of the three realistic examples is depicted. The graph properties of this example are: $|\mathcal{V}| = 88$, $|\mathcal{E}| = 131$, $\rho = 1.49$, $r_{\text{loc}} = 2.55$, $\gamma = 4.36$, $\widehat{\gamma} = 11.4$.

This example includes cycle counts as well, although the exact implementation type has not been specified. The annotated values for the single processes lie between 5 and 111 cycles with an average cycle count of 28.2. Unfortunately, the three realistic examples do not contain the communication between processes in such a form that they could reliably expressed according to their execution time.

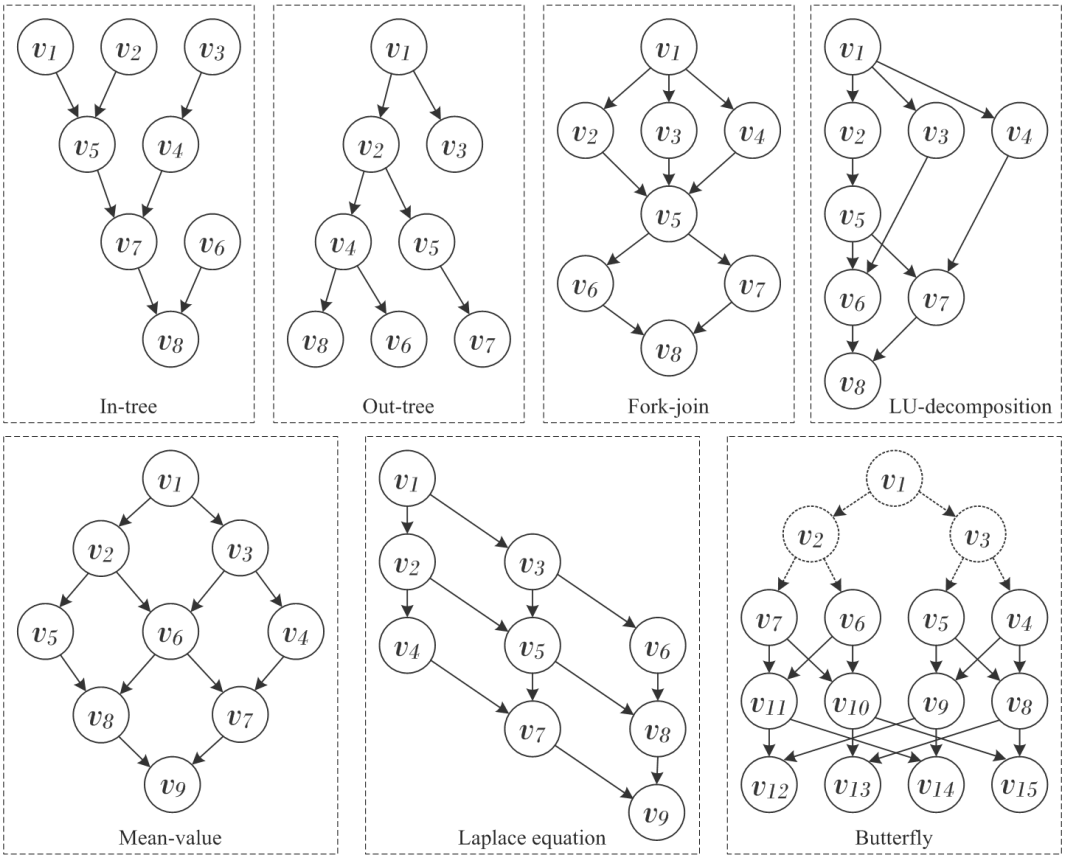


Fig. C.4: Seven task graph categories for signal processing defined in the literature [94, 142].

Wiangtong et al. [142] and Kwok et al. [94] specify the task graphs resolutely in the following

seven very typcial categories: in-tree, out-tree, fork-join, LU-decomposition, mean-value computation, a Laplace equation solver, and an FFT. In Figure C.4 small examples for these categories are depicted. In their work the referred authors varied the number of vertices and the architecture dependent parameters: hardware and software execution time, number of bytes per edge, bus speed, local memory access, the communication to computation ratio, etc.
For our considerations, it is just of importance to extract the graph properties that characterise these seven groups. Due to the very regular structure of some graphs, their graph properties can be given as functions of their chosen size. For others the size has been chosen to be $|\mathcal{V}| = 50$, and the properties are evaluated accordingly.

- In-tree and Out-tree
  The in-tree structure represents data collection, consolidation, and generalisation while processing, whereas the out-tree represents data distribution and specification while processing. Typically, for the in-tree $|\mathcal{V}_{\text{start}}| \sim \log_2 |\mathcal{V}|$, i.e. the number of start vertices (data collecting points) increases logarithmically with the graph size (and analogically for the end vertices of the out-tree).
  For all sizes is $\rho \approx 1.0$, and the $r_{\text{loc}} \approx 1 \ldots 2$ is low, but increases with the graph size, and $\gamma \sim \sqrt{|\mathcal{V}|}$ increases as well with the graph size but more rapidly.

- Fork-join
  This category is a concatenation of in-tree and out-tree structures and obeys as such the same rules.

- Mean-value and Laplace Equation Solver
  These two 'categories' exhibit identical graph structures, which raises the question why they are separated in two categories. For all sizes $\rho \approx 1.5$, $r_{\text{loc}} \approx 1$ and γ is simple to calculate for this graph with depth d: $\gamma = \left(\frac{d+1}{2}\right)^2 / d$. For $|\mathcal{V}| = 49$ ($d = 13$) is $\gamma = 49/13 = 3.77$.

- Butterfly Graph (FFT or Walsh-Hadamard Transform)
  Butterfly graphs belong to the most regular graph structures. The number of vertices is given via the dimension of the input vector n (here $n = 4$ with input vertices $v_{4\ldots7}$), to be $|\mathcal{V}| = n \log_2(n)$. The density is constant for all sizes $\rho = 2$, as well as $\gamma = n \log_2(n)/(n-1) \approx \log_2(n)$ and $r_{\text{loc}} = 1$.

- LU Decomposition
  The LU decomposition refers to an $n \times n$ nonsingular matrix [101]. The number of vertices is $|\mathcal{V}| = (n+1)n/2$ and the length of the critical path through such a graph (or its depth) is $|\mathcal{V}_{\text{CP}}| = 2(n-1)$, hence for $n = 7$ is $\gamma = 28/12 = 2.33$. The density

is $\rho = (2(n-1)n/2) \,/\, ((n+1)n/2) = 2(n-1)/(n+1) \approx 2$. The rank-locality is $r_{\mathrm{loc}} \approx 1.5$.

Without exception any of the aforementioned examples and categories exhibits the characteristic values for the graph properties introduced in Section 4.1.

These graphs originating from different sources in embedded system design did not contain all the information regarding varying implementation types, which would have been required to directly serve as application examples of the partitioning algorithms. However, for considerable parts of the first two industrial design from IFX a lot of measurements and high level synthesis values were available, so that it was possible to derive typical value ranges for execution times, compiled code size and gate counts. Henceforth, the following section surveys the graph generation and enrichment engine by which means the utilised graph sets have been created.

C.2 Generation of System Graph Sets

An intuitive and trivial graph generator constructs a random graph just by adding random pairs of integers from the interval $[1, |\mathcal{V}|]$ until the intended density ρ is reached. This strategy either leads to many loops and parallel edges (multi-graphs) when the density is high or leads to forests when the density is low. Although it is easily possible to eliminate self-loops, multi-edges, and forests, these graphs do not have any specialised structure and may not resemble typical graphs for the considered problem space.

From the literature many other graph generators are known [126] as the combinatorial nature of graphs is typically intended to be structured yet randomised. The structured graph generators are rather diverse: Euclidean neighbour graphs, transaction graphs, function call graphs, interval graphs, de Bruijn graphs, etc. The k-neighbour graph, already described in Section 4.1 has been chosen to serve as base model for our graph sets, as it is easy to gear them towards the desired properties concerning sparsity, locality, and to a minor degree towards parallelism. The sparsity is limited to $\rho < k_{\mathrm{loc}} < |\mathcal{V}| - 1$ for k-locality graphs. The vertices are then aligned on a vector, which processed element-wise. For any vector element i, the following k_{loc} elements are visited iteratively and an edge is inserted between two vector elements with edge generation probability P_e. This probability is determined by the intended sparsity $P_e = \rho / k_{\mathrm{loc}}$. To obtain acyclic graphs the edges are always directed from lower indices of the vector to higher indices. In Figure C.5 an example of such an acyclic k-locality graph is depicted.

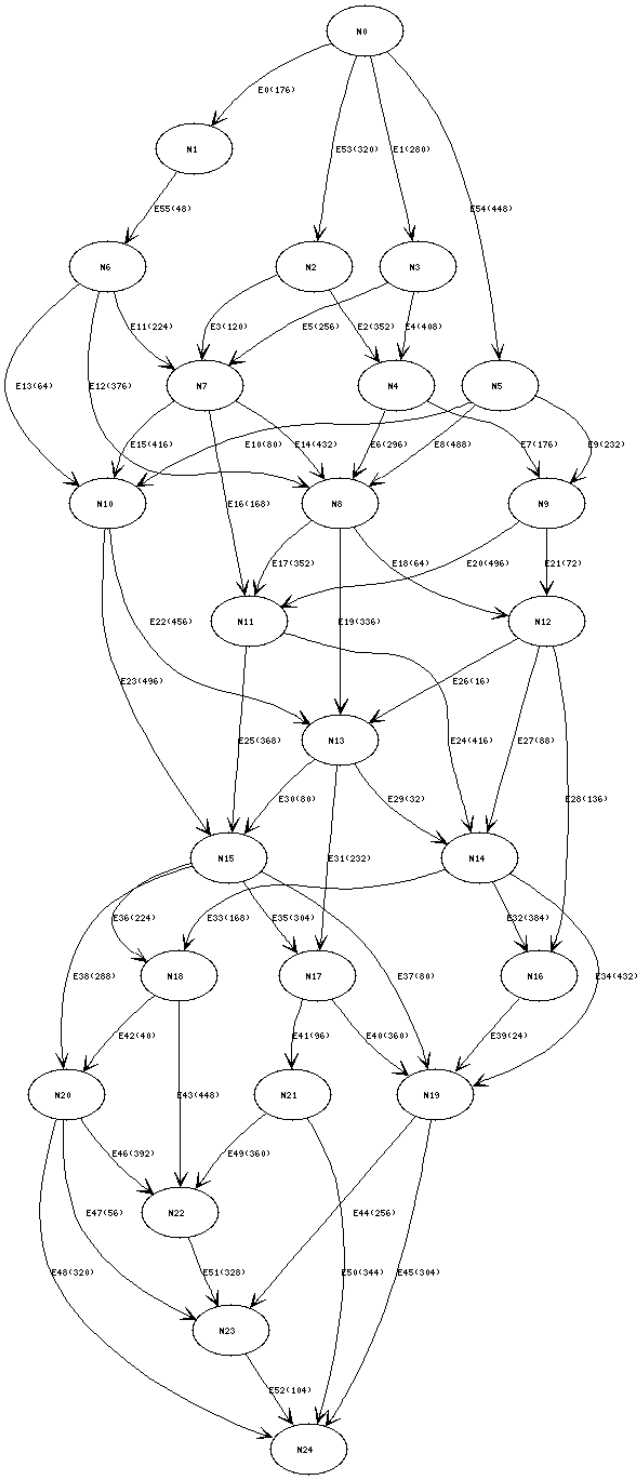


Fig. C.5: Acyclic k-locality graph with $k_{\mathrm{loc}} = 5$, $|\mathcal{V}| = 25$, $\rho = 3$

.

The other substantial ingredient is the enrichment of the system graph with realistic values for code size, area, power consumption, and execution time. Apart from the individual generation of reasonable values in typical ranges for leaf functions of a system graph, two additional aspects have to be considered. The first aspect is the rather crude relation that functions featuring many operations, operands, and control statements offer comparably large implementation alternatives on all resources. It is in general very unlikely that implementation alternatives with very low values for *all* parameters coexist with alternatives with rather high values for *all* parameters.

Concretely, the following table lists the value ranges utilised in the graph generation engine for execution time measured in cycles, code size measured in bytes, area measured in gates (NAND2 equivalents), and power measured in mW. The listed properties have been extracted from the aforementioned realistic examples as well as from the literature [77, 95, 142]. These value ranges are of course supposed to be modified by the designer depending on his knowledge about the chip manufacturing process, favoured FPGA or DSP type, and of course the high level estimation techniques utilised to obtain these values. For instance, the power

property	unit	Minimum	Maximum
execution time (et)	cycles	10	200
code size (cs)	bytes	8	$1,024$
gate count (gc)	NAND2 gate equivalents	500	$10,000$
power consumption (pc)	mW	10	$1,000$

Tab. C.4: Possible ranges for process properties utilised in the graph generation engine.

consumption ranges may vary a lot depending on the intended DSP or CPU categories: the Itanium®2 processor consumes about 130Watt running at 1GHz under full load with about $10,000$MOPS (million operations per second), whereas the StrongARM110 processor works at 160MHz under full load with about 500MOPS drawing only a power of about 900mW [95].

Another relevant aspect to be considered is that the generated subset of implementation alternatives $\mathcal{I}_r(v_i) \subseteq \mathcal{I}(v_i)$, of a process v_i for any specific resource r contains K Pareto-optimal elements $A_i(r,k)$, $i = 1 \ldots K$. Consider the example depicted in Figure C.6. The graph generation engine tries to create eight implementation alternatives for a single process v_i for a specific resource $r \in \mathcal{R}$ with the two dimensions execution time et and area gc. Three of these generated alternatives are not Pareto-optimal and have to be erased from the subset of alternatives for this resource, since they only contribute to the search space size without offering any possibility to improve the overall quality of the solution.

The last essential consideration for the task graph generation involves the fact that the values for code size, area, power consumption, and execution time are not completely independent. A common dependency is the area-execution time trade-off for an FPGA or ASIC implementation of a function: the higher the consumed silicon area, the lower the execution time. That is due to design parameters as pipelining, unrolling factor, operator reuse, etc. [66, Knerr et al.].

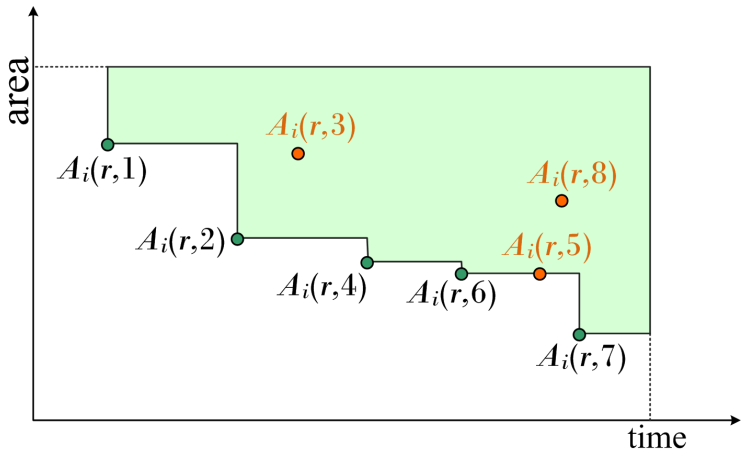


Fig. C.6: Pareto-optimal implementation alternatives of a process for a single resource r.

Additionally, the power consumption of an implementation alternative is naturally correlated both with the occupied chip area (ASIC or FPGA) and the code size (DSP). Therefore, the task graph generation engine creates the first two dimensions of the implementation alternatives for processes on any resource, $A_i(r,k) = (et, cs, 0, 0)$ for DSPs, $A_i(r,k) = (et, 0, gc, 0)$ for ASICs or FPGAs, in a Pareto-optimal manner. Then, the global maximum and minimum values for cs or gc for any resource are known: $\min\limits_r cs, \max\limits_r cs, \min\limits_r gc, \max\limits_r gc, r \in \mathcal{R}$. The power consumption of any process value is then correlated to these values, e.g. for a process implementation on a DSP D:

$$pc = \frac{cs - \min\limits_D cs}{\max\limits_D cs - \min\limits_D cs} (1 + v_{cs}) (\max_D pc - \min_D pc), \tag{C.1}$$

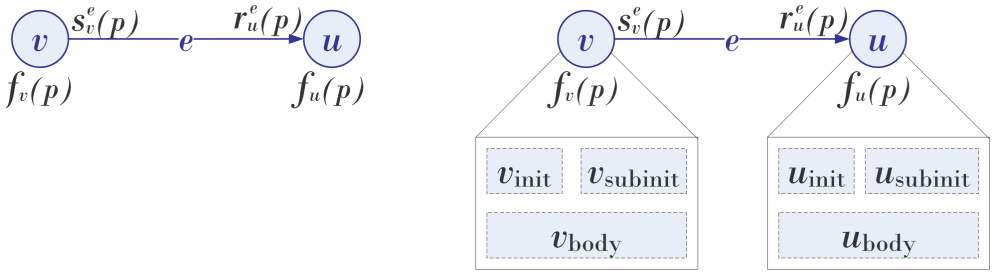
with v_{cs} being randomly taken out of the interval $[-0.2, 0.2]$ with uniform distribution and analogously for FPGAs with gc instead of cs. By these means, both the correlation between the characteristic values as well as the Pareto-optimality of the implementation alternatives can be ensured.

C.3 Parameterisable SDF Graphs

Although widely accepted for signal processing systems, SDF graphs are restricted to static dataflow behaviour, thus modern SoC applications are often not completely amenable to SDF. Parameterised dataflow, published in 1996 [20], provides the dynamic behaviour by means of structured, dynamic parameter changes in the base model that it is applied to. A

parameterised SDF (PSDF) graph is composed of PSDF actors and PSDF edges. A PSDF actor is characterised by a set of parameters that can control the actors functionality, including the actors dataflow behaviour that is, the numbers of tokens consumed and produced at its input and output ports. Similarly, a PSDF edge also has associated notions of parameterisations and configuration. A PSDF subsystem consists of three distinct PSDF graphs the init graph, the subinit graph and the body graph. Intuitively, the body graph models the main functional behaviour of the specification, whereas the init and subinit graphs control the behaviour of the body graph by appropriately configuring the body graph parameters.

Another more recent extension of SDF took place in 2001 with cyclo-static dataflow (CSDF) [19]. In CSDF, token production and consumption can vary between actor firings as long as the variation forms a certain type of periodic pattern. Each component of this periodic pattern is called a phase of the actor. Actor behaviour, including dataflow properties of the actor, can vary between phases $p = 1 \ldots P_v$, as long as the sequence of phases is periodic. Formally, in



(a) Cyclo-static SDF graph

(b) Parameterised cyclo-static SDF graph

Fig. C.7: SDF graph extensions.

CSDF, every actor $v \in \mathcal{V}$ has an underlying execution sequence $f_v(1), f_v(2), \ldots, f_v(P_v)$ of length P_v, where each $f_v(p)$ is called a phase as depicted in Figure C.7(a). Conceptually, at every P_vth time instant an actor v periodically iterates again over its firing phase sequence $f_v(1), f_v(2), \ldots, f_v(P_v)$. As a consequence, in a CSDF graph sample production $s_v^e(p)$ and consumption $r_u^e(p)$ rates follow periodic sequences.

Most recently, Saha et al. published a work in which the concepts of parameterised and cyclo-static SDF graphs has been combined [124], hence creating a new class of graphs. Parameterised cyclo-static SDF (PCSDF) combines powerful optimisation capabilities with strong expressibility properties. The syntax and semantics for PCSDF graphs are similar to that of PSDF graphs. However, unlike a PSDF actor, a PCSDF actors functionality as well as its dataflow behaviour are not parameterised directly. Instead, they vary cyclically and this cyclic pattern is parameterised.

The two fundamental parameters involved in the dataflow properties of a PCSDF actor are the period of the cycle of phases, and the data rates (i.e., the rates of token production and consumption) associated with each phase. Values of these parameters must remain constant throughout any given iteration of the underlying (parameterised) CSDF graph; however, there is significant flexibility in reconfiguring the parameters between iterations. Dynamic behaviour of a CSDF graph is therefore modelled by allowing dynamic parameterisations of the period length, and of the individual phases associated with each actor. Thus, it is possible to have behaviours in which the sequence of phases is fixed for a single iteration, or for a group of successive iterations, while the sequences can generally vary between iterations. Figure C.7(b) shows an example of a PCSDF subsystem.

Up to now no commercial or academic EDA tools exists supporting any of the latter very sophisticated models of computation in a comfortable way. As a consequence the major part of embedded system domain still relies on a diverse set of models of computations: discrete-event (or finite state machines) for the control-heavy system parts and process or SDF graphs for computationally intensive signal processing.

D. NP-COMPLETE ALGORITHMS AND OPTIMALITY

This chapter lists two classical NP-complete problems, both of which can alternatively serve as problem description for the scheduling (and partitioning) problem. Their description can be found in the literature by the authors Garey and Johnson [46]. Furthermore, in the last section the term Pareto-optimality is described in more detail.

D.1 Multi-processor Scheduling

Given is the problem instance with a set $\mathcal{T}$ of tasks, a number $m \in \mathbb{Z}^+$ of processors, a length $l(t) \in \mathbb{Z}^+$ for each $t \in \mathcal{T}$, and a deadline $D \in \mathbb{Z}^+$.
The problem is to find an m-processor schedule σ for $\mathcal{T}$ that meets the overall deadline D, i.e. a function $\sigma(t) : \mathcal{T} \rightarrow \mathbb{Z}_0^+$, such that $\forall\, u \geq 0$: the number of tasks $t \in \mathcal{T}$ for $\sigma(t) \leq u < \sigma(t) + l(t)$ is no more than m and such that $\forall\, t \in \mathcal{T} : \sigma(t) + l(t) \leq D$.

As in our scenario unconnected graphs are uncommon but certainly allowed, in other words precedence constraints may not be present, the aforementioned classical problem represents a special case of our combined partitioning and scheduling (PS) problem. The unconnected set of vertices $\mathcal{V}$ corresponds to the task set $\mathcal{T}$, the execution time $et(v)$ of a vertex $v \in \mathcal{V}$ corresponds to the length $l(t)$ of the tasks $t \in \mathcal{T}$, the set of resources $\mathcal{R}$ corresponds to m processors, and eventually the deadline D maps to the timing constraint b_{T}. When there are not any other constraints involved, as in Section 4.3.7, in which the impact of the k-locality on the performance of the RRES algorithm is described for a platform of three identical DPSs, this classical formulation is equivalent to the special case: $k = 0 \Longleftrightarrow \mathcal{E} = \emptyset$. Thus, a known NP-complete problem is a special case of the PS problem formulated in this thesis, and hence, by restriction [46](pg 63), PS is also NP-complete.

D.2 Precedence Constrained Scheduling

Given is the problem instance with a set $\mathcal{T}$ of tasks, each having the same length $l(t) = 1$, a number $m \in \mathbb{Z}^+$ of processors, a partial order $\lessdot$ on $\mathcal{T}$[1], and a deadline $D \in \mathbb{Z}^+$.
The problem is to find an m-processor schedule σ for $\mathcal{T}$ that meets the overall deadline D and obeys the precedence constraints, i.e. such that $t \lessdot t'$ implies $\sigma(t') \geq \sigma(t) + l(t) = \sigma(t) + 1$.

This problem can serve as well as a corresponding formulation of a special case to our problem description. In this case the precedence constraints (represented by directed edges between tasks) are explicitly included. Herein the task lengths are set to equal values in this formulation, which is as well uncommon but certainly allowed in our scenario. Even for the case $m = 2$, Ullman proved the problem to be NP-complete, if only two different task lengths are allowed [139]. Thus, this classical problem is as well a special case for the problem PS considered in this thesis, and consequently, by restriction, PS has also to be NP-complete.

D.3 Pareto Optimality

Pareto optimal solutions are characterised by their dominance relation. The following dominance relations between two design points $\mathbf{x}_1$ and $\mathbf{x}_2$ can be identified:

$$\begin{gathered}\mathbf{x}_1 \succ \mathbf{x}_2 \text{ (dominates) if } \mathbf{f}(\mathbf{x}_1) < \mathbf{f}(\mathbf{x}_2),\\ \mathbf{x}_1 \succeq \mathbf{x}_2 \text{ (weakly dominates) if } \mathbf{f}(\mathbf{x}_1) \leq \mathbf{f}(\mathbf{x}_2),\\ \mathbf{x}_1 \sim \mathbf{x}_2 \text{ (is indifferent to) if } \mathbf{f}(\mathbf{x}_1) \not\leq \mathbf{f}(\mathbf{x}_2) \wedge \mathbf{f}(\mathbf{x}_1) \not\geq \mathbf{f}(\mathbf{x}_2).\end{gathered}$$

The relation for vectors is defined in the following.

$$\begin{gathered}\mathbf{u} = \mathbf{v} \text{ if for all } i = 1, \ldots, k : u_i = v_i,\\ \mathbf{u} \leq \mathbf{v} \text{ if for all } i = 1, \ldots, k : u_i \leq v_i,\\ \text{and } \mathbf{u} < \mathbf{v} \text{ if for all } i = 1, \ldots, k : u_i < v_i.\end{gathered}$$

The relations greater and greater equal are defined symmetrically.

A more detailed picture is given by the following considerations:

Definition 29 (Pareto-optimality)**.** A vector $\mathbf{x}_1$ is Pareto optimal if there does not exist another vector $\mathbf{x}_2$ such that $\mathbf{x}_2 \succ \mathbf{x}_1$. The set of Pareto optimal points is called Pareto

[1] The operator $\lessdot$ for partial order is an equivalent description of precedence constraints to directed edges in graphs.

optimal set X_{p} or short Pareto front. Furthermore, the approximation of the Pareto set X_{p} is will be called quality set X_{q}.

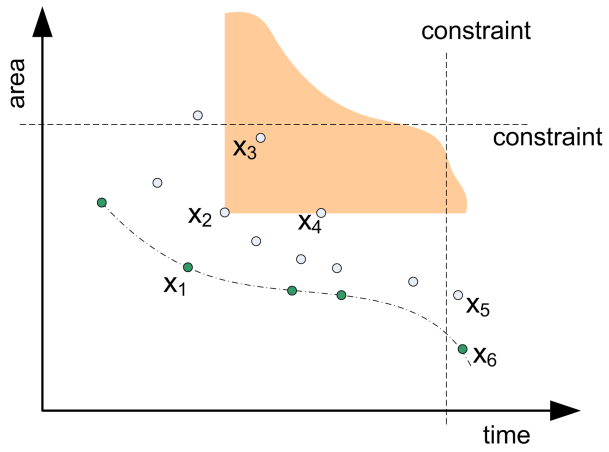


Fig. D.1: Pareto front for implementation alternatives with area-time trade-off.

In Figure D.1 a two-dimensional objective space for area and timing is depicted. The design point $\mathbf{x}_1$ is Pareto optimal and it dominates $\mathbf{x}_2$, $\mathbf{x}_3$, $\mathbf{x}_4$ and $\mathbf{x}_5$, ($\mathbf{x}_1 \succ \mathbf{x}_2$, $\mathbf{x}_1 \succ \mathbf{x}_3$, $\mathbf{x}_1 \succ \mathbf{x}_4$, $\mathbf{x}_1 \succ \mathbf{x}_5$), whereas $\mathbf{x}_3$ is indifferent to $\mathbf{x}_4$ ($\mathbf{x}_3 \sim \mathbf{x}_4$). The design point $\mathbf{x}_2$ weakly dominates $\mathbf{x}_3$($\mathbf{x}_2 \succeq \mathbf{x}_3$). Also some constraints for maximum timing and area are shown, so that the design points $\mathbf{x}_5$ and $\mathbf{x}_6$ become invalid. Mathematically, all the Pareto optimal points are equally acceptable solutions. However, it is generally desirable to obtain only one solution, which has been solved by the linear weighted sum described in Section 3.5.2.

E NOTATION, VARIABLES, AND ACRONYMS

E.1 Notation

a,b,c	scalars
$\mathbf{a}$,$\mathbf{b}$,$\mathbf{c}$	vectors
$\mathbf{A}$,$\mathbf{D}$,$\mathbf{C}$	matrices
a_i	element of vector
$\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$	sets
$\lvert\mathcal{A}\rvert$	cardinality of a set
$(.)^T$	Transpose operator
$\min f(x)$	Minimum of the function $f(x)$
$\max f(x)$	Maximum of the function $f(x)$
$\operatorname{argmin} f(x)$	Argument x for which the scalar function $f(x)$ is minimised
$\operatorname{argmax} f(x)$	Argument x for which the scalar function $f(x)$ is maximised
$\mathrm{beg}(e)$	Returns the vertex from which the edge e emerges in a directed graph.
$\mathrm{end}(e)$	Returns the vertex which the edge e points to in a directed graph.
$\mathrm{indegree}(v)$	Returns the number of incoming edges of a vertex v in a directed graph.
$\mathrm{outdegree}(v)$	Returns the number of outgoing edges of a vertex v in a directed graph.
$\mathrm{out}(e)$	Returns the number of samples produced by vertex $v = \mathrm{beg}(e)$ in an SDF graph.
$\mathrm{in}(e)$	Returns the number of samples consumed by vertex $v = \mathrm{end}(e)$ in an SDF graph.
$\mathrm{rank}(v)$	Rank of a vertex v in a graph structure.
$\mathrm{par}(v)$	Returns the number of parallel vertices of v.
$\preccurlyeq$	Partial-order operator, equivalent to directed edges in graph representations.
$\succ, \succeq$	Domination operator for Pareto relations.
$\mathrm{P}(\mathcal{A})$	Powerset of set $\mathcal{A}$.

E.2 List of Variables

$\mathcal{V}$	Set of vertices.
v_i	ith vertex.
$\mathcal{E}$	Set of edges.
e_i	ith edge.
$G(\mathcal{V}, \mathcal{E})$	Graph.
$\mathbf{p}$	Path through a graph.
$\mathbf{p}_s$	Simple path (without cycles).
$\mathbf{\Gamma}$	Topology matrix of an SDF graph.
k_{loc}	Classic locality property used for generation of graphs.
r_{loc}	Rank locality property used for characterising existing graphs.
ρ	Graph density/sparsity.
γ	Degree of parallelism based on number of vertices.
γ_T	Degree of parallelism based on vertex' weights.
$\widehat{\gamma}$	Average number of parallel vertices in a graph.
$\mathcal{G}$	Set of system graphs.
$\mathcal{R}$	Set of platform resources.
$A^i_{r,j}$	jth implementation alternative for vertex v_i on resource r.
a_v	Average number of implementation alternatives for vertices.
a_e	Average number of implementation alternatives for edges.
$\mathcal{I}(v_i)$	Set of implementation alternatives for vertex v_i.
$\mathcal{I}(e_i)$	Set of implementation alternatives for edge e_i.
$\mathbf{x}$	A partitioning solution represented by a vector.
x_i	Element of $\mathbf{x}$, identifier for a distinct mapping of vertex v_i.
$\mathbf{x}_p$	Pareto-optimal partitioning solution.
I	A problem instance.
$\mathbb{S}(I)$	Solution space for a problem instance I.
$\mathbb{S}_f(I)$	Subspace of all *feasible* solutions.
$\mathbb{S}_p(I)$	Subspace of all Pareto-optimal solutions.
$n_I(\mathbf{x})$	Neighbourhood transformation for solution $\mathbf{x}$.
et	Execution time.
gc	Gate count when implemented as hardware.
cs	Compiled code size when implemented as software.
pc	Power consumption.
tt	Communication transfer time.
b_i	ith constraint for a partitioning problem (absolute).
$b_{A,r}$	Area constraint for resource r.
$b_{C,r}$	Code size constraint for resource r.

b_T	Time constraint.
$f_i(\mathbf{x})$	ith objective function for solution $\mathbf{x}$.
$f_{i,norm}$	ith objective function normalised to its constraint and minimum values.
$\widetilde{f}_{i,norm}$	ith normalised objective function penalised when constraint is not met.
$\mathbf{w}$	weight vector for objective functions.
Ω	Cost value of a partitioning solution.
$\overline{\Omega}$	Cost value averaged over many runs of a single instance and/or over many problem instances.
σ	Standard deviation of cost for many runs of a single problem instance.
$\overline{\sigma}$	Standard deviation, averaged over many problem instances.
Ψ	Validity ratio.
$\overline{\Psi}$	Validity value averaged over many problem instances.
Θ	Run time measured in clock cycles on an AMD ATHLON 64 3000+ Dual Core 1.8GHz PC.
$\overline{\Theta}$	Run time averaged over many problem instances.
C_i	ith constraint ratio for a partitioning problem (relative to min, max values).
C_A	Area constraint ratio.
C_T	Time constraint ratio.
C_C	Code size constraint ratio.
$\mathrm{O}(f(n))$	Asymptotic efficiency of an algorithm.
L_Q	Average length of a breadth first search queue (LEP).
RoI	Region of interest (LEP).
C_{avg}	Average number of collision per schedule (LEP).
LP	Local phase value (GCLP).)
GC	Global criticality value (GCLP).)
ϑ	Geometric cooling factor (SA).
T	Temperature (SA).
T_{init}	Initial temperature (SA).
$\mathbb{S}_{N,TS}$	Size of visited neighbourhood (TS).
L_{tabu}	Tabu list length (TS).
s_{reg}	Region size for diversification and intensification (TS).
$\mathcal{P}$	Population set (GA).
$st_{asap}(v)$	Start time of vertex v in an ASAP schedule (GA).
$st_{alap}(v)$	Start time of vertex v in an ALAP schedule (GA).
$P_{sel}(\mathbf{x})$	Selection probability for roulette wheel selection (GA).
$P_{1g}(\mathbf{x})$	Mutation probability for one-gene mutation (GA).
W	Window length (RRES).

E.3 List of Acronyms

ACS	Add Compare Select
ALAP	As Late As Possible
ALU	Arithmetic Logic Unit
AMBA	Advanced Microcontroller Bus Architecture
ASAP	As Soon As Possible
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction Set Processor
BB	Basic Block
BFS	Breadth First Search
BNS	Best Neighbour Search
BT	Binary Tournament
CAN	Controller Area Network
CFG	Control Flow Graph
CISC	Complex Instruction Set Computer
CLB	Configurable Logic Blocks
CPU	Central Processing Unit
CVS	Concurrent Version System
DAG	Directed Acyclic Graph
DCT	Discrete Cosine Transform
DE	Discrete Event
DFG	Data Flow Graph
DFS	Depth First Search
DMA	Direct Memory Access
DPRAM	Dual Port RAM
DSL	Digital Subscriber Line
DSP	Digital Signal Processor
DRL	Dynamic Reconfigurable Logic
EDA	Electronic Design Automation
EEPROM	Electrically Erasable Programmable ROM
ES	Exhaustive Search
ETF	Earliest Task First
FIFO	First In, First Out
FIR	Finite Impulse Response (Filter)
FPGA	Field Programmable Gate Array
GA	Genetic Algorithm
GCLP	Global Criticality Local Phase

GPP	General Purpose Processor
GSM	Global System for Mobile Communication
GXL	Graph Exchange Library
HDL	Hardware Description Language
HLF	Highest Level First
HLS	High Level synthesis
HW	Hardware
IP	Intellectual Property
ISS	Instruction Set Simulator
LEP	Local Exploitation of Parallelism
LSB	Least Significant Bit
LSI	Large Scale Integration
MAC	Multiply Accumulate
MOC	Model of Computation
MOO	Multi-Objective Optimisation
NNS	Next Neighbour Search
OTIE	Open Tool Integration Environment
PC	Personal Computer
PDA	Personal Digital Assistant
PTP	Point To Point
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RNS	Random Neighbour Search
ROM	Read Only Memory
RRES	Restricted Range Exhaustive Search
RTL	Register Transfer Level
RTOS	Real Time Operating System
RW	Roulette Wheel
SA	Simulated Annealing
SAG	Single Activation Graph
SDF	Synchronous Data Flow
SDL	Specification and Description Language
SDRAM	Synchronous Dynamic RAM
SoC	System-on-Chip
SOTF	Survival Of The Fittest
SQL	Structured Query Language
SRAM	Static RAM
SW	Software

TLM	Transaction Level Model
TS	Tabu Search
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunication System
VC	Virtual Component
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLIW	Very Large Instruction Word
VLSI	Very Large Scale Integration
VSIA	Virtual Socket Interface Alliance
VP	Virtual Prototyping
XML	eXtensible Markup Language
μC	Microcontroller
μP	Microprocessor

BIBLIOGRAPHY

[1] G. Agosta, F. Bruschi, and D. Sciuto. Static Analysis of Transaction-Level Models. In *Proc. of the Design Automation Conference (DAC)*, pages 448–453, June 2003.

[2] A. Aho, R. Sethi, and J. Ullmann. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1985.

[3] Altera Corporation. Altera Devices. `http://www.altera.com/products/devices/dev-index.jsp`.

[4] T. Anderson, R. Schutten, and F. Thoen. Virtual prototypes cut software bottleneck. Technical report, Wireless Systems Design Online Magazine, February 2005. `http://www.wsdmag.com/Articles/ArticleID/9821`.

[5] ARM Corporate Communications. ARM7 Family - 32 bit RISC processor. `http://www.arm.com/products/CPUs/ARM720T.html`.

[6] ARM Corporate Communications. Advanced Microcontroller Bus Architecture (AMBA) 3.0, 1996–2007. `http://www.arm.com/products/solutions/AMBA3AXI.html`.

[7] J. Axelsson. Architecture Synthesis and Partitioning of Real-Time Systems: A Comparison of Three Heuristic Search Strategies. In *Proc. of the 5th Int. Workshop on HW/SW Codesign, (CODES/CASHE)*, pages 161–165, 1997.

[8] J. Axelsson. Cost Model for Electronic Architectures Trade Studies. In *Proc. of the 6th Int. Conf. on Engineering of Complex Computer Systems*, 2000.

[9] N. Azeemi. A Multiobjective Evolutionary Approach for Constrained Joint Source Code Optimization. In *Proc. of the 19th International Conference on Computer Application in Industry (ISCA)*, pages 175–180, Las Vegas, Nevada, USA, November 2006.

[10] B. Bailey. The Waking of the Sleeping Giant – Verification, April 2002. `http://www.mentor.com/consulting/techpapers/mentorpaper_8226.pdf`.

[11] R. Baines and D. Pulley. A Total Cost Approach to Evaluating Different Reconfigurable Architectures for Baseband Processing in Wireless Receivers. In *IEEE Communications Magazine*, volume 41, pages 105–128, January 2003.

[12] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara. Hardware-software co-design of embedded systems: The polis approach. 1997.

[13] P. Belanović. *An Open Tool Integration Environment for Efficient Design of Embedded Systems in Wireless Communications*. PhD thesis, Institute for Communications and Radio Frequency Engineering, Vienna University of Technology, 2006.

[14] P. Belanović, M. Holzer, D. Mičušík, and M. Rupp. Design Methodology of Signal Processing Algorithms in Wireless Systems. In *Int. Conf. on Computer, Communication and Control Technologies CCCT'03*, pages 288–291, July 2003.

[15] P. Belanović, B. Knerr, M. Holzer, and M. Rupp. A Fully Automated Environment for Verification of Virtual Prototypes. *EURASIP Journal on Applied Signal Processing*, 2006. Article ID 32408, 12 pages.

[16] P. Belanović, B. Knerr, M. Holzer, G. Sauzon, and M. Rupp. A Consistent Design Methodology for Wireless Embedded Systems. *EURASIP Journal on Applied Signal Processing*, 2005(16):2598–2612.

[17] P. Belanović and M. Rupp. Automated Floating-point to Fixed-point Conversion with the *fixify* Environment. In *Proc. of the In. Workshop on Rapid System Prototyping (RSP)*, pages 172–178, June 2005.

[18] G. Berry. The esterel v5 language primer, version 5.21 release 2.0, April 1999. `ftp://ftpsop.inria.fr/meije/esterel/papers/primer.pdf`.

[19] B. Bhattacharya and S. Bhattacharyya. Parameterized dataflow modeling for dsp systems. *IEEE Trans. on Signal Processing*, 49(10):2408–2412, 2001.

[20] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cyclostatic dataflow. *IEEE Trans. on Signal Processing*, 44(2):397–408, 1996.

[21] P. Bjuréus, M. Millberg, and A. Jantsch. FPGA Resource and Timing Estimation from Matlab Execution Traces. In *Proc. of the Int. Workshop on Hardware/Software Co-Design*, pages 31–36, May 2002.

[22] C. Brandolese, W. Fornaciari, and F. Salice. An Area Estimation Methodology for FPGA Based Designs at SystemC-Level. In *Design Automation Conference*, pages 129–132, June 2004.

[23] Cadence Design Systems Inc. Cadence Virtual Component Co-Design (VCC). http://www.cadence.com.

[24] Celoxica Ltd. Handel-c language reference manual. http://www.celoxica.com/techlib/.

[25] K. Chatha and R. Vemuri. An Iterative Algorithm for Hardware-Software Partitioning, Hardware Design Space Exploration and Scheduling. (5):281–293, 2000.

[26] K. Chatha and R. Vemuri. Magellan: multiway hardware-software partitioning and scheduling for latency minimization of hierarchical control-dataflow task graphs. In *Proceedings of the ninth international symposium on Hardware/software codesign (CODES)*, pages 42–47, New York, NY, USA, 2001. ACM Press.

[27] P. Chou, R. Ortega, and G. Borriello. The chinook hardware/software co-synthesis system. pages 22–27, 1995.

[28] Y. Collete and P. Siarry. *Multiobjective Optimization. Principles and Case Studies (Decision Engineering)*. Springer-Verlag, Berlin Heidelberg New York, 2003.

[29] CoWare Design Systems. SPW 4. Technical report, 2004. http://www.coware.com/products/spw4.php.

[30] CoWare Inc. Processor Designer, 2005. http://www.coware.com/products/processordesigner.php.

[31] G. de Micheli, R. Ernst, and W. Wolf. *Readings in Hardware/Software Co-Design*. Morgan Kaufman Publishers, Academic Press, San Francisco, CA, USA, 2002.

[32] G. de Micheli and M. Sami. *Hardware-Software Co-Design*. Kluwer Academic Publishers, 1996.

[33] Design Automation Standards Committee. *IEEE Std 1076-2000, IEEE Standard VHDL Language Reference Manual*. IEEE Computer Society, December 2000.

[34] Design Automation Standards Committee. *IEEE Std p1364-2001, IEEE Standard Verilog Hardware Description Language*. IEEE Computer Society, 345 East 47th Street, New York, NY 10017-2394, USA, March 2001.

[35] R. Dick and N. Jha. MOGAC: A Multiobjective Genetic Algorithm for the Co-synthesis of HW/SW Embedded Systems. In *Proc. of the IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pages 522–529, Washington, DC, USA, 1997. IEEE Computer Society.

[36] Y. Donoso and R. Fabregat. *Mult-Objective Optimization in Computer Networks Using Metaheuristics*. Auerbach Publications, 6000 Broken Sound Parkway NW, Boca Raton, FL, 2007.

[37] S. Edwards, L. Lavagno, E. Lee, and A. Sangiovanni-Vincentelli. Design of embedded systems: formal models, validation, and synthesis. pages 86–107, 2002.

[38] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli. System level hardware/software partitioning based on simulated annealing and tabu search. *Design Automation for Embedded Systems*, 2:5–32, 1997.

[39] R. Ernst, J. Henkel, and T. Benner. Hardware-software cosynthsis for microcontrollers. *IEEE Design & Test of Computers*, 10(4):64–75, 1993.

[40] C. Fiduccia and R. Mattheyses. A linear-time heuristic for improving network partitions. In *25 years of DAC: Papers on Twenty-five years of electronic design automation*, pages 241–247, New York, NY, USA, 1988. ACM.

[41] K. Fogel and M. Bar. *Open Source Development with CVS*. Paraglyph Press, 3rd edition, 2003.

[42] W. Fornaciari, P. Gubian, D. Sciuto, and C. Silvano. Power Estimation of Embedded Systems: A Hardware/Software Codesign Approach. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6:266–275, 1998.

[43] D. Franke and M. Purvis. Hardware/Software Codesign: A Perspective. In *Proc. of the 13th Int. Conf. on Software Engineering*, pages 344–352, 1991.

[44] Gaisler Research. LEON3 SPARC V8 Processor Core. `http://www.gaisler.com/leonmain.html`.

[45] D. Gajski, S. Narayan, L. Ramachandran, F. Vahid, and P. Fung. System Design Methodologies: Aiming at the 100h Design Cycle. *IEEE Transactions on Very Large Scale Integration Systems*, 4(1):70–82, March 1996.

[46] M. Garey and D. Johnson. *Computers and Intractability: A Guide to NP-Completeness*. W.H. Freeman, San Francisco, California, 1979.

[47] L. Geppert. *High-flying DSP architectures*. IEEE Spectrum, 1998.

[48] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Ann. Oper. Res.*, 41(1-4):3–28, 1993.

[49] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[50] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[51] R. Gupta and G. D. Micheli. Hardware-software cosynthesis for digital systems. *IEEE Design & Test of Computers*, 10(3):29–41, 1993.

[52] S. Gupta. SPARK: A high-level synthesis framework for applying parallelizing compiler transformations. In *Proc. of the Int. Conf. on VLSI Design*, January 2003.

[53] W. Haas, M. Hofstaetter, T. Herndl, and A. Martin. UMTS Baseband Chip Design. In *Informationstagung Mikroelektronik*, pages 261–266, Vienna, October 2003.

[54] W. Hardt and W. Rosenstiel. Prototyping of tightly coupled hardware/software systems. *Design Automation for Embedded Systems*, 2:283–317, 1997.

[55] J. Hausner and R. Denk. Implementation of Signal Processing Algorithms for 3G and Beyond. *IEEE Microwave And Wireless Components Letters*, 13(8), 2003.

[56] M. Hecht and J. Ullman. Flow graph reducibility. In *STOC '72: Proceedings of the fourth annual ACM symposium on Theory of computing*, pages 238–250, New York, NY, USA, 1972. ACM.

[57] J. Henkel and R. Ernst. A hardware/software partitioner using a dynamically determined granularity. In *Proc. of the 34th Annual Conf. on Design Automation (DAC)*, pages 691–696, New York, NY, USA, 1997. ACM Press.

[58] T. Henzinger and J. Sifakis. *The Embedded Systems Design Challenge*. Lecture Notes in Computer Science. Springer, 2006.

[59] A. Hoffmann, T. Kogel, and H. Meyr. A Framework for Fast Hardware-Software Co-simulation. In *Proc. of the Design, Automation and Test in Europe DATE'01*, Munich, 2001.

[60] A. Hoffmann, H. Meyr, and R. Leupers. *Architecture Exploration for Embedded Processors with LISA*. Kluwer Academic Publishers, December 2002.

[61] M. Holzer. *Design Space Exploration for Embedded Systems in Wireless Communications*. PhD thesis, Institut für Nachrichtentechnik und Hochfrequenztechnik, Vienna University of Technology, 2008.

[62] M. Holzer and B. Knerr. Pareto Front Generation for a Tradeoff between Area and Timing. In *Austrochip 2006 Tagungsband*, pages 131–134, Messegelände Wien, Austria, October 2006.

[63] M. Holzer, B. Knerr, P. Belanović, and M. Rupp. Efficient Design Methods for Embedded Communication Systems. *EURASIP Journal on Embedded Systems*, 2006. Article ID 64913, 18 pages.

[64] M. Holzer, B. Knerr, and M. Rupp. Structural Verification in Minimal Time. In *International Symposium on System-on-Chip*, pages 151–154, Tampere, Finland, November 2006.

[65] M. Holzer, B. Knerr, and M. Rupp. Design Space Exploration for Real-Time Reconfigurable Computing. In *Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, USA, November 2007.

[66] M. Holzer, B. Knerr, and M. Rupp. Design Space Exploration with Evolutionary Multi-Objective Optimisation. In *IEEE Symposium on Industrial Embedded Systems (SIES)*, pages 126–133, July 2007.

[67] M. Holzer and M. Rupp. Static Estimation of the Execution Time for Hardware Accelerators in System-on-Chips. In *International Symposium on System-on-Chip*, November 2005.

[68] J. Hromkovic and W. Oliva. *Algorithmics for Hard Problems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[69] T. Hu. Parallel Sequencing and Assembly Line Problems. Technical Report 6, Operations Research, 1961.

[70] J.-J. Hwang, Y.-C. Chow, F. Anger, and C.-Y. Lee. Scheduling Precedence Graphs in Systems with Interprocessor Communication Times. *SIAM J. Comput.*, 18(2):244–257, 1989.

[71] L. Hyafil and R. Rivest. Graph partitioning and constructing optimal decision trees are polynomial complete problems. Technical Report 33, IRIA-Laboria, Rocquencourt, France, 1973.

[72] Infineon Technologies AG. Vinetic-xm, voice and internet enhanced telephony if circuit, 2004.

[73] International Organization for Standardization. ISO 11898-1:2003, Controller Area Network (CAN), 2003. `http://www.iso.org/iso/iso_catalogue/`.

[74] International SEMATECH. International Technology Roadmap for Semiconductors, 1999. http://www.sematech.org.

[75] International SEMATECH. International Technology Roadmap for Semiconductors, 2005. http://www.sematech.org.

[76] A. Jantsch and H. Tenhunen. *Networks on Chip*. Springer, 2003.

[77] A. Kalavade. *System-level Codesign of Mixed Hardware-software Systems*. PhD thesis, University of California, Berkeley, CA, USA, 1995.

[78] A. Kalavade and E. Lee. The Extended Partitioning Problem: Hardware/software Mapping, Scheduling, and Implementation-bin Selection. *Readings in hardware/software co-design*, pages 293–312, 2002.

[79] Kasahara Laboratory. Standard task graph set, 2006. Dept. of Electrical Engineering, Waseda University.

[80] H. Keding, M. Willems, M. Coors, and H. Meyr. FRIDGE: A Fixed-Point Design and Simulation Environment. In *Proc. of the Design, Automation and Test In Europe (DATE)*, Feb 1998.

[81] B. Kernighan and S. Lin. An Efficient Heuristic Procedure in Partitioning Graphs. *Bell System Technical Journal*, February 1970.

[82] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[83] B. Knerr, P. Belanovic, M. Holzer, G. Sauzon, and M. Rupp. Design Flow Improvements for Embedded Wireless Receivers. In *Proc. of the 12th European Signal Processing Conference (EUSIPCO)*, pages 2015–2018, Wien, September 2004.

[84] B. Knerr, M. Holzer, P. Belanovic, G. Sauzon, and M. Rupp. Advanced UMTS Receiver Chip Design Using Virtual Prototyping. In *Proceedings of the 2004 International Symposium on Signals, Systems and Electronics ISSSE 04*, Linz, Austria, August 2004.

[85] B. Knerr, M. Holzer, and M. Rupp. RRES: A Novel Approach to the Partitioning Problem for a Typical Subset of System Graphs. *EURASIP Journal on Embedded Systems*, 2008. Article ID 259686, 13 pages.

[86] B. Knerr, M. Holzer, and M. Rupp. HW/SW Partitioning Using High Level Metrics. In *Proceedings of the International Conference on Computing, Communications and Control Technologies*, pages 33–38, Austin, Texas, August 2004.

[87] B. Knerr, M. Holzer, and M. Rupp. Fast rescheduling of multi-rate systems for hw/sw partitioning algorithms. In *Proc. of Thirty-Ninth Annual Asilomar Conference on Signals, Systems, and Computers*, Monterey, CA, USA, October 2005.

[88] B. Knerr, M. Holzer, and M. Rupp. Task scheduling for power optimisation of multi frequency synchronous data flow graphs. In *Proceedings of the 18th Annual Symposium on Integrated Circuits and System Design*, pages 50–55, Florianapolis, Brazil, September 2005. ACM Press.

[89] B. Knerr, M. Holzer, and M. Rupp. Extending the GCLP algorithm for HW/SW partitioning: A detailed platform model and performance improvements. In *Austrochip 2006 Tagungsband*, pages 89–95, Messezentrum Wien, Austria, October 2006.

[90] B. Knerr, M. Holzer, and M. Rupp. A fast rescheduling heuristic of SDF graphs for HW/SW partitioning algorithms. In *Proceedings of COMSWARE 2006*, New Delhi, India, January 2006.

[91] B. Knerr, M. Holzer, and M. Rupp. Improvements of the gclp algorithm for HW/SW partitioning of task graphs. In *Proceedings of the 4th IASTED Int. Conf. on Circuits, Signals, and Systems (CSS)*, pages 107–113, San Francisco, CA, USA, November 2006.

[92] B. Knerr, M. Holzer, and M. Rupp. Novel Genome Coding of Genetic Algorithms for the System Partitioning Problem. In *Proc. of IEEE 2nd Int. Symposium on Industrial Embedded Systems (SIES)*, pages 134–141, Lisboa, Portugal, July 2007.

[93] B. Knerr, M. Holzer, and M. Rupp. Restricted Range Exhaustive Search: A New Heuristic for HW/SW Partitioning of Task Graphs. In *Proc. of XXII Conf. on Design of Circuits and Integrated Systems (DCIS)*, Sevilla, Spain, November 2007.

[94] Y.-K. Kwok and I. Ahmad. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. *IEEE Trans. Parallel Distrib. Syst.*, 7(5):506–521, 1996.

[95] L. Benini. ESSES'03: System-Level Power Optimization, Design Techniques & CAD Tools, 2003. DEIS University' di Bologna, Italy.

[96] P. Lapsley, J. Bier, A. Shoham, and E. Lee. *DSP Processor Fundamentals*. IEEE Press, 1997.

[97] M. Le. 8-bit microcontrollers: still going ..., June 2004. `http://www.eetimes.com/showAricle.jhtml?articleID=54202120`.

[98] E. Lee. Overview of the Ptolemy Project. Technical report, University of Berkeley, March 2001. `http://ptolemy.eecs.berkeley.edu`.

[99] E. Lee and D. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, 36(1):24–35, 1987.

[100] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. JohnWiley and Sons Ltd., Chichester, England, 1990.

[101] R. Lord, J. Kowalik, and S. Kumar. Solving linear algebraic equations on an mimd computer. *J. ACM*, 30(1):103–117, 1983.

[102] T. McCabe. A Complexity Measure. In *IEEE Transaction of Software Engineering*, volume SE-2, pages 308–320, December 1976.

[103] C. Mehlführer, F. Kaltenberger, M. Rupp, and G. Humer. A Scalable Rapid Prototyping System for Real-time MIMO OFDM Transmissions. In *Proc. of 2nd IEE/EURASIP Conf. on DSP enabled Radio*, Southampton, UK, September 2005.

[104] Mentor Graphics. High Level Synthesis with CatapultC. `http://www.mentor.com/products/esl/high_level_synthesis/index.cfm`.

[105] Mentor Graphics. ModelSim. `http://www.model.com/`.

[106] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[107] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, Berlin, Heidelberg, New York, 2nd edition, 1994.

[108] MIPS Technologies. MIPS Hard IP Cores. `http://www.mips.com/content/Products/Cores/HardIPCores/content_html`.

[109] G. Moore. Cramming More Components Onto Integrated Circuits. *Electronics Magazine*, 38 (8):114–117, April 1965.

[110] Y. Moullec, N. B. Amor, J.-P. Diguet, M. Abid, and J.-L. Philippe. Multi-Granularity Metrics for the Era of Strongly Personalized SOCs. In *Design, Automation and Test in Europe*, pages 674–679, March 2003.

[111] Y. L. Moullec, P. Koch, J.-P. Diguet, and J. Philippe. Design Trotter: Building and Selecting Architectures for Embedded Multimedia Applications. In *Proc. of the IEEE Int. Symposium on Consumer Electronics*, December 2003.

[112] MySQL Database Products. `http://www.mysql.com/products/database/`.

[113] S. Narayan, F. Vahid, and D. Gajski. System specification with the specchartslanguage. *IEEE Design & Test of Computers*, 9(4):6–13, December 1992.

[114] Y. Neuvo. Cellular Phones as Embedded Systems. In *Proc. of the IEEE International Solid-State Circuits Conference ISSCC'04*, pages 32–37, February 2004.

[115] NXP Semiconductors. P80C51 8-bit microcontroller family 128/256 byte RAM ROM-less low voltage 2.7V. Technical report, 2007. `http://www.nxp.com/Products/Microcontrollers/8-bit80C51microcontrollers/Standard80C51(12clock)`.

[116] Open SystemC Initiative. `http://www.systemc.org`.

[117] V. Pareto. *Cours D'Economie Politique*, volume I and II. 1896.

[118] M. Platzner and L. Thiele. Hardware/software codesign. Lecture, 2005.

[119] J. Poole. A Method to Determine a Basis Set of Paths to Perform Program Testing. U.S. Department of Commerce/National Institute of Standards and Technology, November 1995.

[120] H. Posadas, F. Herrera, P. Sanchez, E. Villar, and F. Blasco. System-Level Performance Analysis in SystemC. In *Design, Automation and Test in Europe*, pages 378–383, February 2004.

[121] Radioscape. Radiolab 3g, 1999. Licensable Block Set for The MathWorks MATLAB and Simulink products.

[122] F. Renner, J. Becker, and M. Glesner. Communication Performance Models for Architecture-Precise Prototyping of Real-Time Embedded Systems. *Design Automation for Embedded Systems*, 5:351–363, 2000.

[123] M. Rupp, A. Burg, and E. Beck. Rapid Prototyping for Wireless Designs: the Five-Ones Approach. *Signal Processing Europe 2003*, 83:1427–1444, July 2003.

[124] S. Saha, S. Puthenpurayil, and S. Bhattacharyya. Dataflow transformations in high-level dsp system design. In *IEEE International Symposium on System-on-Chip*, pages 131–136, Tampere, Finland, November 2006.

[125] A. Sangiovanni-Vincentelli, M. Sgroi, and L. Lavagno. Formal models for communication-based design. In *Proc. of the 11th Int. Conf. on Concurrency Theory (CONCUR'00)*, pages 29–47, London, UK, 2000. Springer-Verlag.

[126] R. Sedgewick. *Algorithms in C++, Part 5: Graph Algorithms*. Addison-Wesley, 3rd edition, January 2002.

[127] J. P. Singh, A. Kumar, and S. Kumar. A Multiplier Generator for Xilinx FPGA's. In *International Conference on VLSI Design: VLSI in Mobile Communications*, pages 322–323, 1996.

[128] M. Smith. *Application-Specific Integrated Circuits*. Addison-Wesley, 1997.

[129] V. Srinivasan, S. Radhakrishnan, and R. Vemuri. Hardware software partitioning with integrated hardware design space exploration. In *Proceedings of Design, Automation & Test in Europe (DATE)*, 1998.

[130] R. Subramanian. Shannon vs. Moore: Driving the Evolution of Signal Processing Platforms in Wireless Communications. In *Proc. of the IEEE Workshop on Signal Processing Systems SIPS'02*, October 2002.

[131] Synopsys Inc. Design Compiler. `http://www.synopsys.com/products/logic/design_compiler.html`.

[132] Synopsys Inc. Galaxy Design Platform. `http://www.synopsys.com/products/solutions/galaxy_platform.html`.

[133] Synopsys Inc. System Studio. `http://www.synopsys.com/products/designware/system_studio/system_studio.html`.

[134] System Verilog. `http://www.systemverilog.org`.

[135] SystemC AMS and Design of Embedded Mixed Signal Systems. `http://www.systemc-ams.org`.

[136] Texas Instruments. IP Video Phone, DaVinci SoC based. `http://focus.ti.com/docs/solution/folders/print/206.html`.

[137] The MathWorks Inc. Simulink. `http://www.mathworks.com/products/simulink`.

[138] The SpecC System - Center for Embedded Computer Systems. `http://www.cecs.uci.edu/~specc/`.

[139] J. Ullman. Np-complete scheduling problems. *J. Comput. Syst. Sci.*, 10(3):384–393, 1975.

[140] F. Vahid and T. D. Le. Extending the Kernighan/Lin Heuristic for Hardware and Software Functional Partitioning. *Design Automation for Embedded Systems Journal*, (2):237–261, 1997.

[141] T. Wiangtong. *Hardware/Software Partitioning and Scheduling for Reconfigurable Systems*. PhD thesis, Dept. of Electrical and Electronic Engineering, University of London, 2004.

[142] T. Wiangtong, P. Cheung, and W. Luk. Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware-Software Codesign. *Design Automation for Embedded Systems*, 6(4):425–449, Sept 2002.

[143] J. Wilberg, A. Kuth, R. Camposano, W. Rosenstiel, and T. Vierhaus. Design exploration in castle. In *Workshop on High Level Synthesis Algorithms Tools and Design (HILES)*, 1995.

[144] W. Wolf. Hardware-Software Co-design of Embedded Systems. In *Proc. of the IEEE*, volume 82, pages 965–989, 1994.

[145] Xilinx Inc. Xilinx: The Programmable Logic Company. `http://www.xilinx.com/`.

[146] V. Zivojnovic, S. Pees, and H. Meyr. Lisa - machine description language and generic machine model for hw/sw co-design. In *Proc. of the IEEE Workshop on VLSI Signal Processing*, San Francisco, October 1996.

[147] Y. Zou, Z. Zhuang, and H. Chen. Hw-sw partitioning based on genetic algorithm. In *Congress on Evolutionary Computation (CEC)*, pages 628–633, Portland, Oregon, June 2004.