TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Video signal processor mapping

Citation for published version (APA):
de Kock, E. A. (1999). *Video signal processor mapping*. [Phd Thesis 2 (Research NOT TU/e / Graduation TU/e), Mathematics and Computer Science]. Technische Universiteit Eindhoven. https://doi.org/10.6100/IR529864

DOI:
10.6100/IR529864

Document status and date:
Published: 01/01/1999

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 07. Jul. 2024

Video Signal Processor Mapping

On the cover: a composition of photographs of the video signal processor system showing the programming environment, a processor board, and the display of a result that has been produced by executing a video algorithm in real-time. The programming environment contains implementations of the techniques to map video algorithms onto processor boards that are presented in this thesis.

Video Signal Processor Mapping

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof.dr. M. Rem, voor een commissie aangewezen door het College voor Promoties in het openbaar te verdedigen op woensdag 15 december 1999 om 16.00 uur

door

Erwin de Kock

geboren te Tilburg

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. E.H.L. Aarts
en
prof.dr. P.A.J. Hilbers

Copromotor:
dr.ir. J.H.M. Korst

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

de Kock, Erwin

Video Signal Processor Mapping /
Erwin de Kock. -
Eindhoven: Eindhoven University of Technology
Thesis Eindhoven. - With index, ref. - With summary in Dutch
ISBN 90-74445-48-9
Subject headings: mapping, scheduling, partitioning, lifetime minimization,
combinatorial optimization, code generation, video signal processing

© Philips Electronics N.V. 1999
All rights are reserved. Reproduction in whole or in part is
prohibited without the written consent of the copyright owner.

Preface

The video signal processor project has been carried out at the Philips Research Laboratories from 1987 to 1997. I have been involved in this project on several occasions both as a student and as an employee. This thesis completes my contribution to the project. I would like to express my gratitude towards the many people that made this work possible.

First of all, I would like to thank my promoters. I thank Emile Aarts for his guidance during the seven years that we have been working together. In this period he has taught me his way of doing research which is reflected in this thesis. I appreciate our cooperation very much and look forward to it in the future. I thank Peter Hilbers and Jan Korst for carefully reading drafts of this thesis. Peter's extensive knowledge of parallel systems has been of great help in generalizing this work. Jan's excellent mathematical skills have revealed many inconsistencies. Their advice has led to many improvements.

I am also grateful to my colleagues Gerben Essink, Peter van Gerwen, Wim Smits, and Kees Vissers. Without their contribution this thesis would not have existed. I thank them for their friendship and the sharing of their ideas. In particular I want to thank Gerben for the discussions on mapping techniques and software engineering, and Wim for evaluating the mapping tools and for disseminating the results of the project.

Furthermore, I would like to thank the students that helped in tackling the mapping problem. In chronological order they are René van Dongen, Babette de Fluiter, Peter Vink, Roger Jansen, Mark Smeets, and Angelica Verstraten. Without their assistance we would not have been able to study the mapping problem as thoroughly as we did.

Next, I thank the members of the 'Algoritmen Club' and many other colleagues not already mentioned, for the pleasant atmosphere and the interesting discussions on a variety of topics. I thank Eric van Utteren, Cees Niessen, and Rob Woudsma for giving me the opportunity to carry out the research described in this thesis.

Finally, I give special thanks to my family and friends for their interest in my work and to Patricia for sharing her enthusiastic attitude towards life with me.

Eindhoven, October 1999 Erwin de Kock

Contents

1 Introduction **1**
1.1 Video Signal Processing 1
1.2 Programming Trajectory 4
1.3 Mapping Trajectory 6
1.4 Related Work 7
1.5 Outline 10

2 Problem Formulation **11**
2.1 Processor Networks 11
2.2 Signal Flow Graphs 18
2.3 Mappings 23
2.4 Constraints 26
2.5 Mapping Problem 31
2.6 Problem Instances 31
2.7 Summary 34

3 Complexity Analysis **35**
3.1 Computational Complexity 35
3.2 Type Constraints 38
3.3 Array Constraints 38
3.4 Storage Constraints 38
3.5 Computation and Communication Constraints 40
3.6 Periodicity Constraints 43
3.7 Precedence Constraints 44
3.8 Connectivity Constraints 48
3.9 Summary 52

4 Problem Decomposition **53**
4.1 Mapping Sets 54
4.2 Constraint Relaxations 55
4.3 Decomposition Strategy 61
4.4 Delay Management Problem 64

4.5 Partitioning Problem 65
4.6 Scheduling Problem 66
4.7 Results 67
4.8 Summary 68

5 Delay Management **71**
5.1 Problem Decomposition 71
5.2 Delay Minimization 75
5.3 Delay Assignment 81
5.4 Results 87
5.5 Summary 91

6 Partitioning **93**
6.1 Problem Decomposition 93
6.2 Processor Assignment 97
6.3 Type Assignment 108
6.4 Results 108
6.5 Summary 113

7 Scheduling **115**
7.1 Problem Decomposition 115
7.2 Time Assignment 120
7.3 Processing Element Assignment 127
7.4 Channel Assignment 130
7.5 Results 132
7.6 Summary 136

8 Conclusion **137**

Bibliography **141**

Symbol Index **148**

Author Index **152**

Subject Index **155**

Samenvatting **158**

Curriculum Vitae **160**

1

Introduction

This thesis is concerned with the mapping of video algorithms onto parallel processors. Mapping considers the scheduling over time of the execution of operations onto processing elements, the storage of variables into memories, and the communication of variables between processing elements and memories. Our interest in mapping onto parallel processors originates from the field of digital video signal processing. More precisely, it originates from the problem of compiling a video algorithm description into micro code that can be executed by a network of video signal processors.

In this introductory chapter we discuss the following aspects. In Section 1.1 we give the necessary background information on the field of video signal processing. In Section 1.2 we present the programming trajectory for a specific video signal processor architecture. In Section 1.3 we present the mapping trajectory for this specific architecture. In Section 1.4 we provide an overview of related work. Finally, in Section 1.5 we present an outline of this thesis.

1.1 Video Signal Processing

Signal processing typically concerns the transformation of a stream of input samples into a stream of output samples. Examples of signal processing are correlation techniques, spectrum analyses, and image and sound improvements [Pratt,

1991]. Industrial application of signal processing are found in the field of geophysics, telecommunications, medical imaging, television studios, and process control. Consumer applications are found mainly in the field of audio, video, and mobile telephony.

Each application domain uses its own specific type of signals, but each signal can usually be described in an abstracted form as a function of time. The function value is generally known as *amplitude*. Signals can be either *continuous* or *discrete* in time and in amplitude. A signal that is continuous in time and in amplitude is called an *analog* signal, whereas a signal that is discrete in time and in amplitude is called a *digital* signal. Digitization of a continuous signal in time is called *sampling*. Digitization of a continuous signal in amplitude is called *quantization*.

The processing of digital signals has many advantages compared to the processing of analog signals. One advantage is that one can decide on the computation precision by choosing the sampling frequency and the quantization precision. This makes the amplitude domain finite and allows a unique representation of each amplitude value. A binary representation is a well-known example of such a representation which is used in the implementation of digital signal processing applications on *integrated circuits*. Integrated circuits can be manufactured in large numbers against low cost, they are small and reliable, and they can perform complex computations. The advantage of a binary representation is that small electrical deviations do not influence the amplitude values, as long as the bit values are correctly interpreted. For this reason, digital systems can be reproduced with identical behavior and are less sensible to factors such as aging, noise, and temperature compared to analog systems. Another important advantage is that one can apply error detection and error correction on digital signals, for instance, by adding redundant information to the representation of the amplitude values. Alternatively, one can compress streams of digital signals by using relations between the samples, since streams often have a more compact representation than the sum of their individual sample representations. An important disadvantage of digital signal processing systems is that they cannot handle arbitrarily high frequencies. However, this is becoming less of a limit due to the advanced integrated circuit technology. In the last decade, this technology has opened the way to digital video signal processing.

In order to process digital video signals in real time with an acceptable resolution on television, one requires high sampling frequencies. Television sets contain cathode ray tubes in which an electron beam impinges on light emitting particles located at the inside of the tube. The particles emit red, green, or blue light. Each picture element or *pixel* consists of one red, one green, and one blue light emitting particle. The energies of the electrons that are fired on the particles determine the resulting color of the pixel. While firing, the electron beam moves horizontally from left to right. This results in the display of a sequence of pixels which is called

a video *line*. At the end of each line, the electron beam moves to the beginning of the next line without firing electrons. The time between the last pixel of the previous line and the beginning of the next line is called the *line blanking*. In this way, the electron beam moves vertically from top to bottom. This results in the display of a sequence of lines which is called a video *frame*. The time between the last line of the previous frame and the first line of the next frame is called the *frame blanking*. The motion of the electron beam is called *scanning*. Figure 1.1 visualizes the entire scanning process. The number of frames per second, the number of

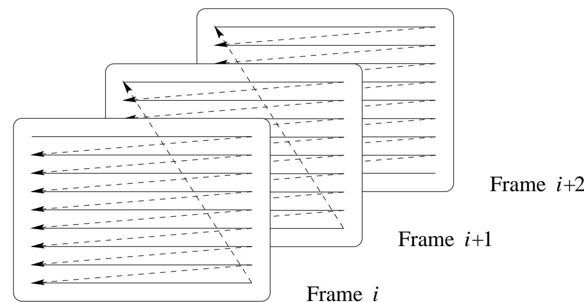


Figure 1.1. The scanning process.

lines per frame, and the number of pixels per line depend on the video standard. In the Phase Alternating Line (PAL) standard one uses 25 frames per second, 625 lines per frame, and 864 pixels per line. This gives a typical sampling frequency of 13.5 megahertz.

An application that contains several hundreds of operations per sample on a stream of 13.5 mega samples per second requires a computation capacity of more than 5 giga operations per second. Typically the time that is required to process one sample is much longer than the time between the arrival of two consecutive samples such that the operations need to be executed in parallel. Consequently, real-time processing of digital video signals requires parallelism to handle the high data throughput. Moreover, two consecutive samples are horizontally adjacent. In order to process two vertical adjacent samples, one has to store a complete video line of 864 samples which typically takes 24 bits per sample since each sample is composed out of a red, a green, and a blue component whose amplitudes are usually represented in 8 bit values. So the storage of a video line takes more than 2 kilobyte and the storage of a video frame takes even more than 1 megabyte. The corresponding communication requirements to transport a single second of video is more than 25 megabyte per second. These amounts of computation, storage, and, especially, communication capacities cannot be delivered by current state-of-the-art general-purpose microprocessors at reasonable cost. Therefore, one uses more specialized processors which are called video signal processors.

Many video signal processors are designed to execute one specific algorithm because that results in a more efficient implementation. The implementation is often in silicon in the form of an application-specific integrated circuit. The design and manufacturing of integrated circuits is time consuming, even with the aid of high-level synthesis tools. This makes them of little use in the development stages of video algorithms when the real-time behavior is to be evaluated.

Programmable video signal processors save time because it is possible to execute many different algorithms on the same hardware such that the fine-tuning of algorithms requires no hardware modification. Another advantage of programmable processors is that they reduce the time-to-market, which is of utmost importance when the competition launches new popular features. Important disadvantages of programmable processors are that their size and their power consumption exceeds that of application-specific circuits.

The mapping of video algorithms onto video signal processors is a complex task due to the high degree of parallelism that is involved. For instance, a processor with a clock frequency of 50 megahertz must execute 20 operations in parallel in order to execute an application that requires 1 giga operations per second in real-time. In order to reduce the development cost and time-to-market of consumer products containing embedded video signal processors, much effort is spent on design automation and code generation for these processors. As examples that have been developed at the Philips Research Laboratories we mention the Phideo design methodology [Van Meerbergen et al., 1995] which supports high-level synthesis of video algorithms into application-specific circuits, and the VSP programming environment [Vissers et al., 1995] which supports code generation of video algorithms for programmable video signal processors. The latter contains implementations of many of the mapping techniques presented in this thesis and has been used to develop many industrially relevant video algorithms.

1.2 Programming Trajectory

The goal of the VSP programming environment is to map video algorithms onto networks of programmable video signal processors. The programming environment is specifically targeted to one video signal processor architecture, called *the VSP architecture*. The architecture contains a number of in parallel operating processing elements with computation and storage capabilities that are fully interconnected by a switch matrix. Each processing element repeatedly executes one sequence of instructions that is determined at compile time. There is no conditional execution of instructions. This model of execution is called *cyclostatic* execution. It limits the set of executable video algorithms, but it ensures the real-time execution of any video algorithm that can be mapped onto a network of video signal

processors. The video algorithms are represented by *signal flow graphs* in which the nodes represent elementary operations from the instruction set and the arcs represent dependencies between the operations. Figure 1.2 graphically visualizes the programming trajectory. Below, we discuss each of the programming steps in more detail.

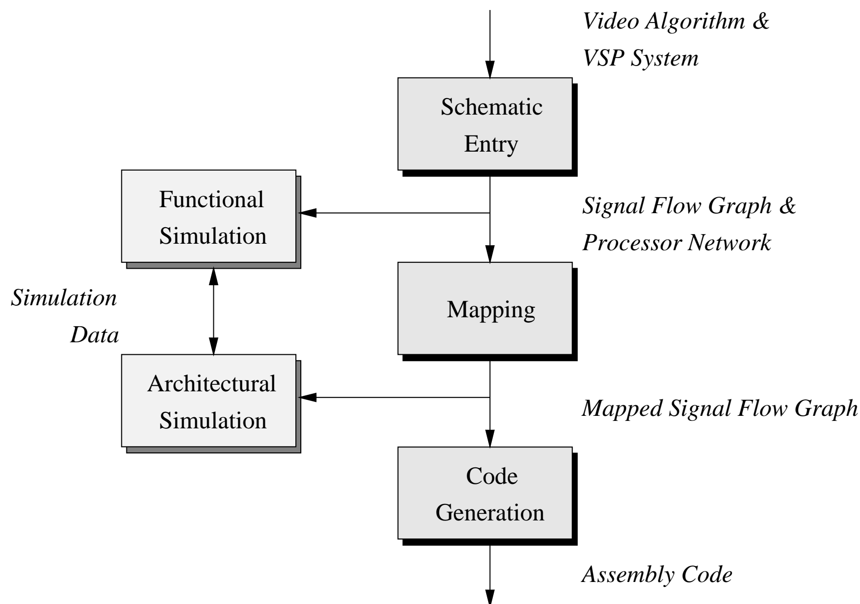


Figure 1.2. The VSP programming trajectory consisting of three steps. The first step is the schematic entry of a video algorithm represented as a signal flow graph and of an architecture instance represented as a processor network. The second step is the mapping of the signal flow graph onto the processor network. The third step is the generation of assembly code.

Schematic entry. In the first step the programmer must represent a video algorithm in the form of a signal flow graph. Often one has specified a video algorithm using a *sequential* programming language that can be executed on a general-purpose microprocessor to do the initial development and check the functional requirements. The sequential representation of the algorithm is now translated into a signal flow graph representation at the level of processing element instructions that explicitly contains the potential *parallelism*. The programming environment contains a functional simulator that can be used to verify the correctness of the translation. In addition, the programmer must describe a system configuration in the form of a video signal processor network. Often one has developed a network for the prototyping of video algorithms. After prototyping one can decide to im-

plement the algorithm on another network, for instance to minimize the number of video signal processors.

Mapping. In the second step the given signal flow graph is mapped onto the given processor network. The main tasks during mapping are to decide on which processing elements the operations are executed, in which clock cycles the operations are executed, how long the variables are stored, and along which routes the variables are transported between the processing elements. To implement the communication of variables between processors and the storage of variables in processing elements, additional operations have to be inserted into the signal flow graph. The goal is to find a mapping that satisfies all functional and architectural constraints.

Code generation. In the third step assembly code is generated. This is a straightforward task once the mapping step has been completed successfully. The programming environment contains an architectural simulator that can be used to verify the correctness of the mapping of the signal flow graph onto the processor network. The results of the functional simulator and the architectural simulator can be compared to verify consistency. Subsequently, the assembly code is translated into binary code that can be downloaded into the physical video signal processor system.

In this thesis we concentrate on mapping because this is the key issue in the programming trajectory. The development of software that supports schematic entry and code generation is an engineering task rather than a research issue.

1.3 Mapping Trajectory

The objective of the mapping problem is to find a mapping of a given signal flow graph onto a given network of video signal processors. We restrict ourselves to processors with cyclostatic execution models that allow off-line nonpreemptive periodic scheduling. The partitioning of algorithms onto multiple processors and the storage of variables into multiple memories are part of the mapping problem.

We develop a decomposition strategy to handle the mapping problem because we believe that the problem is too complex to handle in its entirety. The problem decomposition is based on the results of a complexity analysis using the theory of NP-completeness in order to determine suitable techniques to handle the mapping constraints. In the problem decomposition we distinguish between three subproblems that are called *the delay management problem*, *the partitioning problem*, and *the scheduling problem*. These subproblems can be handled by well-known combinatorial optimization techniques, although the subproblems are hard in the formal sense.

Delay management refers to the problem of determining for each variable how long and in which type of memory it is to be stored. This has to be done in such a way that the storage times do not contradict the data dependencies between the operations and such that the storage capacities of the different memory types are sufficient. To minimize the global storage requirements we apply *network flow* techniques. Subsequently, we apply *bin packing* techniques to find a memory type assignment and we insert additional code in the video algorithm to store the variables in the designated memory types. An additional objective is to balance the utilization of the different memory types in order to simplify the subsequent partitioning and scheduling steps.

Partitioning refers to the problem of determining for each operation on which processor it is to be executed. This has to be done in such a way that the computation, storage, and communication capacities of the processors are sufficient. To handle the communication of variables between nonadjacent processors we apply a recursive bipartitioning approach. The bipartitioning algorithm is based on *local search*. After each bipartitioning step we insert additional code in the video algorithm to communicate the variables between the designated processors. An additional objective is to balance the utilization of the different processors in order to simplify the subsequent scheduling step.

Scheduling refers to the problem of determining for each operation on which processing element and in which clock cycle it is to be executed. This has to be done in such a way that operations on the same processing element and accesses to the same memory do not overlap in time. To handle the assignment of operations to clock cycles we apply a *constraint satisfaction* approach. To handle the assignment of operations to processing elements and the assignment of accesses to memories we apply *graph coloring* techniques.

We refer to the literature for more details on the above-mentioned mapping trajectory. De Kock et al. [1998] present a formal model of the mapping problem and its decomposition in the delay management problem, the partitioning problem, and the scheduling problem. For delay management we refer to Smeets et al. [1997]. For partitioning we refer to De Kock et al. [1995] and Aarts et al. [1996]. For scheduling we refer to Essink et al. [1991a].

1.4 Related Work

The discrete nature of the mapping problem enables the use of a broad range of solutions techniques originating from the field of combinatorial optimization. Introductions to this field are given by Papadimitriou and Steiglitz [1982], Schrijver [1986], Nemhauser and Wolsey [1988], and Cook et al. [1997]. Furthermore, one can investigate the computational complexity of the problem using the theory of

NP-completeness. For an overview of this theory we refer to Garey and Johnson [1979].

The problem of mapping algorithms onto programmable processors is gaining interest in the literature due to the need for flexibility and short design times in a growing number of electronic systems. Many of these systems [Kalouptsidis, 1997] contain embedded processors that are often specifically tuned towards the application area in order to increase efficiency. The conventional compiler techniques for general-purpose microprocessors are not applicable due to the highly irregular architectures of many embedded processors. Consequently, code generation for embedded processors is an emerging research subject. In addition, one aims to broaden the application of developed techniques by retargeting the code generation trajectory to specific processor instances within a given class of architectures. For a survey into the field of retargetable code generation for embedded processors we refer to Marwedel and Goossens [1995]. This survey describes among others the retargetable code generation environments FlexWare [Paulin et al., 1995] and Chess [Lanneer et al., 1995].

Signal processors are an important class of embedded processors that come in a large variety of architectures. Well-known digital signal processor families are the Texas Instruments TMS 320 family [Balmer et al., 1994], the Motorola DSP 56000 and 96000 families [Kloker, 1987; Sohie, 1989], and the Analog Devices ADSP 2100 family [Cavigioli, 1987]. Digital signal processors are often equipped with special features for signal processing such as multiply/accumulate instructions, heterogeneous register sets, alternative memory addressing modes, and multiple arithmetic and logic units, which make them more efficient but also more difficult to program. The most important mapping techniques that are applied in this area are list scheduling [Goossens et al., 1990], constraint satisfaction, [Timmer and Jess, 1993] integer linear programming [Wilson et al., 1995], and genetic algorithms [Grewal and Wilson, 1997]. Additional work in this area has been reported recently among others by Mesman et al. [1998], Hwang and Hwang [1997], Wess et al. [1995], and Desmet and Genin [1993]

The above-mentioned signal processors are not very well suited to handle high-throughput applications such as video algorithms because they do not provide sufficient parallelism. Because of the regular and repetitive nature of video signal processing, we restrict ourselves to more regular and parallel architectures which contain multiple computation and memory resources such as the VSP architecture. Comparable architectures have been reported in the literature by Yeung and Rabaey [1992], Bove and Watlington [1995], and Theis [1996]. These architectures typically support computation models that allow off-line nonpreemptive scheduling. Pioneer work in this area has been performed by Lee and Bier [1990] and Buck [1994]. Korst [1992] and Verhaegh [1995] have extensively studied the more spe-

cific area of off-line nonpreemptive periodic scheduling. They assume that the communication between the resources is unconstrained. Korst [1992] studies the scheduling of video algorithms onto multiprocessors assuming that the communication can be implemented on a programmable interconnection network. Verhaegh [1995] studies the scheduling of video algorithms in high-level synthesis assuming that the communication is implemented on a dedicated interconnection network which is generated after scheduling. For an introduction to scheduling we refer to Coffman [1976], French [1982], and Pinedo [1995].

There are multiprocessor system in which partitioning is an important topic in order to handle the communication constraints. The subject of partitioning digital signal processing algorithms onto multiprocessor systems is studied extensively in the literature. Lengauer [1990] has given an extensive overview of partitioning problems and corresponding solution techniques in integrated circuit layout. Sheu and Chen [1995], Palenichka and Lutsyk [1996], and Gumuskaya et al. [1994] report partitioning problems for fixed multiprocessor networks with linear, mesh, and ring topologies, respectively. The large variety of processor architectures and associated communication structures has led to many different partitioning strategies. Bokhari [1988] and Ashraf and Bokhari [1995] present efficient algorithms for the partitioning of chain-structured digital signal processing algorithms onto a linear array of processors. Stewart [1988] presents mapping strategies of two dimensional digital signal processing graphs onto a triangular systolic array of processors. Koch et al. [1993] study the mapping of digital signal processing algorithms onto a small number of regularly interconnected signal processors. Saha and Krisnamurthy [1994] describe a methodology that can be used to map digital signal processing algorithms onto field programmable gate arrays. Chen et al. [1994] map digital signal processing algorithms in the form of control data flow graphs onto application-specific integrated circuits.

Finally, we mention work on memory optimization that can be classified into code optimization and lifetime optimization. Code optimization refers to the problem of minimizing the amount of memory that is needed to store assembly or binary code. Lifetime optimization refers to the problem of minimizing the amount of memory that is needed to store intermediate values which are generated during the computation. Clearly, lifetime optimization plays a more dominant role in video signal processing than code optimization because of the high data throughput. From the recently reported code optimization techniques we mention Chang et al. [1997] and Gebotys [1997]. Recent work on lifetime optimization is reported by Depuydt et al. [1994], Denk and Parhi [1994], Araujo and Malik [1995], and Cheng and Lin [1995] in the field of programmable digital signal processors, and by Hu et al. [1994] in the field of pipelined integrated circuit design.

1.5 Outline

The objective of this thesis is to study the problem of mapping video algorithms onto networks of programmable video signal processors that have a cyclostatic execution model, and to develop solution techniques to handle this problem. The outline of this thesis is as follows.

In Chapter 2 we mathematically model the problem of mapping video algorithms onto networks of video signal processors. This includes the representation of architecture instances in the form of processor networks and the representation of video algorithms in the form of signal flow graphs. Furthermore, we present a benchmark set of industrially relevant video algorithms to indicate the nature of the problem instances. We use this benchmark set throughout this thesis to evaluate the results of the presented solution approach.

In Chapter 3 we analyze the computational complexity of the mapping problem. We study the complexity of the mapping problem in combination with different subsets of the constraints in order to determine suitable solution techniques.

In Chapter 4 we present a decomposition strategy to handle the mapping problem. The problem decomposition is based on the results of the complexity analysis. There we introduce the three subproblems, i.e., the delay management problem, the partitioning problem, and the scheduling problem.

In Chapters 5, 6, and 7 we present solution approaches to handle the delay management, partitioning, and scheduling problem, respectively. We evaluate the results of the proposed approaches on the benchmark set.

Finally, in Chapter 8 we summarize the main results and provide suggestions for further research.

2

Problem Formulation

In this chapter we formally state the problem of mapping signal flow graphs onto networks of video signal processors. In Section 2.1 we describe the video signal processor architecture and the formal representation of processor networks. In Section 2.2 we describe the video algorithms and the formal representation of signal flow graphs. In Section 2.3 we mathematically formulate a mapping of a signal flow graph onto a processing element network. In Section 2.4 we present the mapping constraints. In Section 2.5 we formulate the mapping problem. In Section 2.6 we present a set of industrially relevant video algorithms to indicate the nature of real-life problem instances. We use these instances in the subsequent chapters to evaluate our solution techniques. Finally, in Section 2.7 we summarize the contents of this chapter.

2.1 Processor Networks

We consider architectures that are represented as networks of video signal processors. They typically contain a number of interconnected video signal processors that interact with surrounding systems. We abstract from these surrounding systems by assuming that input processors produce incoming sample streams and that output processors consume outgoing sample streams. Examples of input and output processors are analog-to-digital and digital-to-analog converters. The intercon-

nections between the processors are fixed and directed from processor outputs to processor inputs. Each input is connected to at most one output.

Internally, the video signal processors contain processing elements that are connected to a switch matrix. Both the processing elements and the switch matrix are programmable. There are four types of processing elements, i.e., arithmetic and logic elements (ALEs), memory elements (MEs), buffer elements (BEs), and output elements (OEs); see Figure 2.1. The processing elements are pipelined in such a

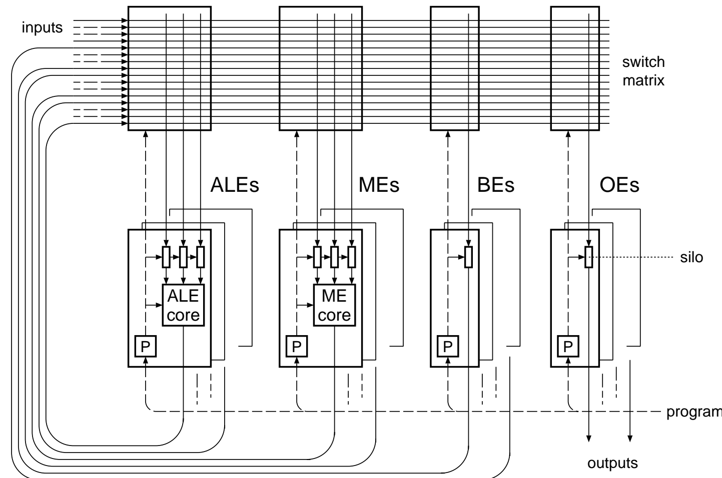


Figure 2.1. Video signal processor architecture.

way that they can start a new instruction in each clock cycle. The instructions are stored in program memories. The computation results are transferred back to the switch matrix or, in case of output elements, to the outputs. Between the outputs of the switch matrix and the inputs of the processing elements, the computation results are stored in programmable delay elements called silos.

The program memories are loaded with instructions via a serial download. The instructions control the switch matrix, the silos, and the cores of the processing elements. After initialization, the programs are executed cyclostatically, which means that they are repeated infinitely and that each instruction is executed unconditionally. This guarantees real-time execution of any algorithm.

The silos consist of a random access memory and some address calculation logic. The storage capacity of the random access memory is thirty-two words. The write addresses are calculated by the address calculation logic in such a way that data from the switch matrix is written cyclically in the random access memory in each clock cycle. Hence, the lifetime of data in a silo is thirty-one clock cycles. The read addresses are calculated by a modulo subtraction of the write address and the required delay length. They are stored in the program memories.

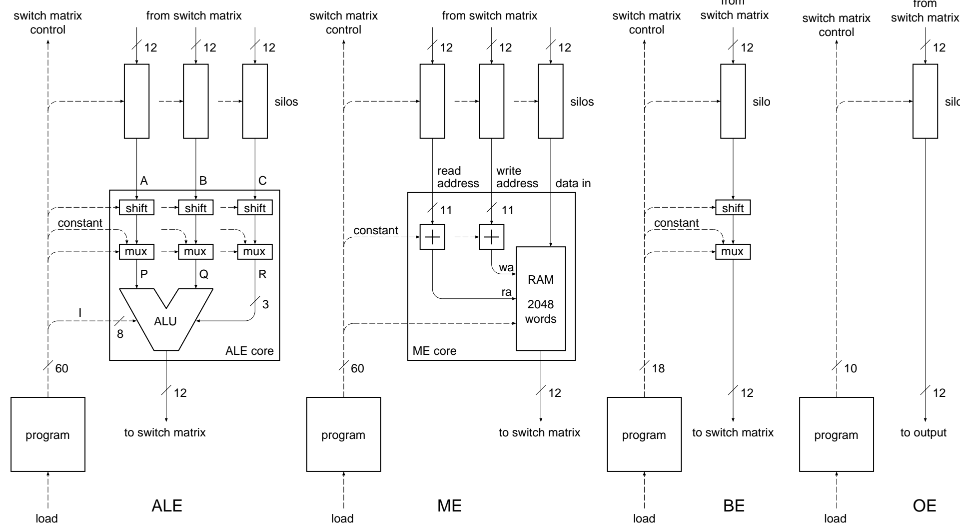


Figure 2.2. Processing element architectures.

Beside these common resources, the various processing element types contain distinct resources which are shown in Figure 2.2. Arithmetic and logic elements contain an arithmetic and logic unit which has a data-dependent instruction set. This instruction set consists of additions, subtractions, logical operations, comparisons, and multiplications. The data-dependent part of the instruction words enters the arithmetic and logic unit via its condition input. For each instruction there exists a symmetrical variant such that it is always possible to maintain the functionality if the contents of the two data inputs are swapped. The arithmetic and logic elements also contain programmable shifters and multiplexers. The shifters are used to execute arithmetical and logical shifts and the multiplexers are used to select operands from either the switch matrix or the program memories. Buffer elements also contain these shifters and multiplexers. Memory elements contain a random access memory with two data ports and either one or two address ports; Figure 2.2 only shows a dual address ported memory element. Single address ported memory elements (ME1S) can start at most one read or write instruction per clock cycle, while dual address ported memory elements (ME2S) can start both a read and a write instruction per clock cycle. Memory elements also contain adders which are positioned in front of the address ports to add offsets that are stored in the program memory to the addresses that arrive from the switch matrix. Dual address ported memory elements furthermore contain additional address calculation logic that supports the implementation of multiple first-in-first-out buffers. This feature is called compact silo. In compact silos the write address is incremented

each instruction cycle whereas in silos the write address is incremented in each clock cycle. The storage requirement needed by a compact silo equals $2^{\lceil {}^2\log(1+n) \rceil}$ words from the random access memory, where the integer n represents the total number of samples that must be stored. For more information about the compact silo feature the reader is referred to Dijkstra et al. [1989].

Table 2.1. Number of pipeline stages.

Input nr.	ALE	ME1	ME2	BE	OE
0	4	4	4	3	1
1	4	3	6		
2	5		3		

The number of pipeline stages varies for the types of processing elements and the types of input terminals; see Table 2.1 in which the inputs are consecutively numbered starting from zero in relation to Figure 2.2. Each pipeline stage takes one clock cycle to complete. For dual ported memory elements that can execute read and write instructions in parallel holds that the read instruction accesses the random access memory before the write instruction but that both accesses occur in the single clock cycle of the last pipeline stage.

Table 2.2. VSP1 and VSP2 characteristics.

Characteristic	VSP1	VSP2
Circuit Technology	1.2μ CMOS	0.8μ CMOS
# Transistors	206,000	1,150,000
Max. Clock Frequency	27 MHz	54 MHz
# Inputs	5 × 12 bit	6 × 12 bit
# Outputs	5 × 12 bit	6 × 12 bit
Program memory size	16 × 60 bit	32 × 60 bit
Silo memory size	32 × 12 bit	32 × 12 bit
ME memory size	512 × 12 bit	2048 × 12 bit
ME memory style	single port	dual port
# ALES	3	12
# MES	2	4
# BES	-	6
# OES	5	6

The above-mentioned video signal processor architecture has been implemented in two integrated circuit versions that are called VSP1 and VSP2. The characteristics of both instances are listed in Table 2.2. In the VSP1 architecture each processing element has a program memory and a program counter, but in the VSP2 architecture buffer elements and output element share program memories and pro-

gram counters in order to save circuit area. The instruction width of arithmetic and logic elements and memory elements equals sixty bits while the instruction width of buffer elements and output elements equals eighteen and ten bits, respectively; see Figure 2.2. So three buffer elements share one program memory of sixty bits width. In the same way six output elements share one program memory of sixty bits width. This saves nine program counters in total, but as a consequence the programs of the processing elements that share a program memory must have the same length. For more information about the VSP1 and VSP2 architectures the reader is referred to Van Roermund et al. [1989] and Veendrick et al. [1994], respectively.

In the remainder of this section we present a formal architecture model that is based the above-mentioned video signal processor architecture. We do this according to increasing granularity of the architecture components, i.e., terminals, processing elements, processors, and processor networks.

Terminals are the smallest components we consider in our architecture model. They constitute the interfaces of the processing elements and the operations that are executed on the processing elements. We model them to formulate type constraints for the mapping of these interfaces. The processing elements of the video signal processor architecture contain six different types of terminals. There are data inputs (DI) and data outputs (DO). Furthermore, the memory elements have address inputs that are dedicated for read addresses (RI), write addresses (WI), or that can handle both (AI). Finally, the arithmetic and logic elements have a condition input (CI) that is dedicated for the data-dependent part of the instruction words.

Definition 2.1 (Terminal Types). The set $T_\mathrm{t} = \{\mathrm{AI}, \mathrm{CI}, \mathrm{DI}, \mathrm{DO}, \mathrm{RI}, \mathrm{WI}\}$ defines the set of *terminal types*. □

Processing elements are the operators in our architecture model. The video signal processor architecture contains five different types of processing elements, i.e., arithmetic and logic elements (ALE), single ported memory elements (ME1), dual ported memory elements (ME2), buffer elements (BE), and output elements (OE). We introduce two additional processing element types (IN and OUT) that are located on input and output processors to model the surrounding system of a processor network. Each processing element type has a set of terminals that each have their own type. The processing element types are graphically depicted in Figure 2.3. With each processing element type and each terminal type we associate a computation delay that indicates when input is required and when output is available at a terminal. Since our processing element types have at most one output terminal we express the computation delay in the pipeline depth, i.e., the computation delay for the output terminals is equal to zero and the computation delay for the input terminals is equal to the number of pipeline stages between the input terminals and the output terminal. They are listed in Table 2.1 where the consecutive

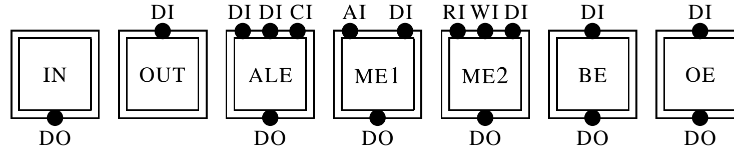


Figure 2.3. Graphical representation of the processing element types. The double squares denote the processing elements, the closed circles above denote the input terminals, and the closed circles below denote the output terminals. The texts denote the types of the processing elements and terminals.

numbering of the input terminals corresponds with the position of the terminals in Figure 2.3 from left to right. Finally, we associate a size with each processing element type that indicates the size of the corresponding random access memory. Only the memory elements have nonzero sizes which are listed in Table 2.2.

Definition 2.2 (Processing Element Type Set). The 6-tuple $(T_\mathrm{p}, I_\mathrm{p}, O_\mathrm{p}, t_\mathrm{p}, d_\mathrm{p}, s_\mathrm{p})$ defines a *processing element type set*, where

- $T_\mathrm{p} = \{\mathrm{IN}, \mathrm{OUT}, \mathrm{ALE}, \mathrm{ME1}, \mathrm{ME2}, \mathrm{BE}, \mathrm{OE}\}$ denotes the set of *processing element types*,
- $I_\mathrm{p}(t)$ denotes a finite set of *input terminals* for all $t \in T_\mathrm{p}$,
- $O_\mathrm{p}(t)$ denotes a finite set of *output terminals* for all $t \in T_\mathrm{p}$,
- $t_\mathrm{p}(t,n) \in T_\mathrm{t}$ denotes the *terminal type* for all $t \in T_\mathrm{p}$ and $n \in I_\mathrm{p}(t) \cup O_\mathrm{p}(t)$,
- $d_\mathrm{p}(t,n) \in \mathbb{Z}$ denotes the *computation delay* for all $t \in T_\mathrm{p}$ and $n \in I_\mathrm{p}(t) \cup O_\mathrm{p}(t)$, and
- $s_\mathrm{p}(t) \in \mathbb{N}$ denotes the *memory size* for all $t \in T_\mathrm{p}$. □

Processors have a cyclostatic execution model. We distinguish between two video signal processor types, namely VSP1 and VSP2, and two additional processor types, namely INPUT and OUTPUT, to model the surrounding system of a processor network. Each processor type has a set of input ports and a set of output ports through which a processor communicates with other processors. Furthermore, a processor type has a set of program memories to store the instructions for its set of processing elements. All the program memories of a processor have the same size. Each processing element has its own type and fetches its instructions from one designated program memory. The number and type of processing elements of the video signal processor types are listed in Table 2.2. The input and output processors contain only one processing element of type IN and OUT, respectively. A set of directed intraprocessor connections indicates which processing elements and ports have point-to-point communication possibilities within a processor. The connections in the video signal processor architecture are graphically depicted in

Figure 2.1. The switch matrix defines a fully connected subgraph. All intraprocessor connections have the same communication delay. All memories of the silos that are located between the outputs of the switch matrix and the inputs terminals of the processing elements have the same size.

Definition 2.3 (Processor Type Set). The 11-tuple $(T_\text{v}, I_\text{v}, O_\text{v}, M_\text{v}, p_\text{v}, P_\text{v}, t_\text{v}, m_\text{v}, C_\text{v}, d_\text{v}, n_\text{v})$ defines a *processor type set*, where

- $T_\text{v} = \{\text{INPUT}, \text{OUTPUT}, \text{VSP1}, \text{VSP2}\}$ denotes the set of *processor types*,
- $I_\text{v}(t)$ denotes a finite set of *input ports* for all $t \in T_\text{v}$,
- $O_\text{v}(t)$ denotes a finite set of *output ports* for all $t \in T_\text{v}$,
- $M_\text{v}(t)$ denotes a finite set of *program memories* for all $t \in T_\text{v}$,
- $p_\text{v}(t) \in \mathbb{N}$ denotes the *program memory size* for all $t \in T_\text{v}$,
- $P_\text{v}(t)$ denotes a finite set of *processing elements* for all $t \in T_\text{v}$,
- $t_\text{v}(t,e) \in T_\text{p}$ denotes the *processing element type* for all $t \in T_\text{v}$ and $e \in P_\text{v}(t)$,
- $m_\text{v}(t,e) \in M_\text{v}(t)$ denotes the *program memory* for all $t \in T_\text{v}$ and $e \in P_\text{v}(t)$,
- $C_\text{v}(t) \subseteq (I_\text{v}(t) \cup Y) \times (X \cup O_\text{v}(t))$ denotes a finite set of *intraprocessor connections* for all $t \in T_\text{v}$, where $Y = \{(e,n) \mid e \in P_\text{v}(t) \wedge n \in O_\text{p}(t_\text{v}(t,e))\}$ denotes a set of output terminals and $X = \{(e,n) \mid e \in P_\text{v}(t) \wedge n \in I_\text{p}(t_\text{v}(t,e))\}$ denotes a set of input terminals,
- $d_\text{v}(t) \in \mathbb{N}$ denotes the *communication delay* for all $t \in T_\text{v}$, and
- $n_\text{v}(t) \in \mathbb{N}$ denotes the *silo size* for all $t \in T_\text{v}$. □

Processor networks contain processors that communicate via directed connections between their ports. Each processor has its own type. The input ports are connected to at most one output port. With each pair of interconnected processors we associate a communication delay. For simplicity we assume that all processors and connections operate on the same clock frequency. However, this is not a fundamental restriction in our architecture model.

Definition 2.4 (Processor Network). The 4-tuple $(P_\text{n}, t_\text{n}, C_\text{n}, d_\text{n})$ defines a *processor network*, where

- P_n denotes a finite set of *processors*,
- $t_\text{n}(v) \in T_\text{v}$ denotes the *processor type* for all $v \in P_\text{n}$,
- $C_\text{n} \subseteq Y \times X$ denotes a finite set of *interprocessor connections*, where $Y = \{(v,m) \mid v \in P_\text{n} \wedge m \in O_\text{v}(t_\text{n}(v))\}$ denotes the set of output ports and $X = \{(v,m) \mid v \in P_\text{n} \wedge m \in I_\text{v}(t_\text{n}(v))\}$ denotes the set of input ports, such that for all $(y_1,x_1),(y_2,x_2) \in I$ it holds that if $x_1 = x_2$ then $y_1 = y_2$, and
- $d_\text{n}(v_1,v_2) \in \mathbb{N}$ denotes the *communication delay* for all $v_1, v_2 \in P_\text{n}$ for which there exists a connection $((v_1,m_1),(v_2,m_2)) \in C_\text{n}$. □

For notational convenience we represent architecture instances as *processing element networks*. This allows us to explicitly denote the processing elements and the connections between them, without considering the input and output ports of the processors. In addition, we introduce short-hand notation for often used properties of a processing element network.

Definition 2.5 (Processing Element Network). Let (P_n, t_n, C_n, d_n) be a processor network. Then the 9-tuple $(M, p_a, P, m_a, n_a, t_a, s_a, C, d_a)$ defines a *processing element network*, where

- $M = \{(v,i) \mid v \in P_n \wedge i \in M_v(t_n(v))\}$ denotes the set of *program memories*,
- $p_a(m) = p_v(t_n(v))$ denotes the *program memory size* for all $m = (v,i) \in M$,
- $P = \{(v,e) \mid v \in P_n \wedge e \in P_v(t_n(v))\}$ denotes the set of *processing elements*,
- $m_a(p) = (v, m_v(t_n(v), e))$ denotes the *program memory* for all $p = (v,e) \in P$,
- $n_a(p) = n_v(t_n(v))$ denotes the *silo size* for all $p = (v,e) \in P$,
- $t_a(p) = t_v(t_n(v), e)$ denotes the *processing element type* for all $p = (v,e) \in P$,
- $s_a(p) = s_p(t_a(p))$ denotes the *memory size* for all $p = (v,e) \in P$,
- $C = Y \cup X$ denotes the set of *communication channels*, where $Y = \{(((v,e_1), n_1),((v,e_2),n_2)) \mid (v,e_1),(v,e_2) \in P \wedge ((e_1,n_1),(e_2,n_2)) \in C_v(t_n(v))\}$ denotes the set of intraprocessor channels and $X = \{(((v_1,e_1),n_1),((v_2,e_2),n_2)) \mid (v_1,e_1),(v_2,e_2) \in P \wedge ((e_1,n_1),((v_1,m_1),(v_2,m_2)),(n_2,e_2)) \in C_v(t_n(v_1)) \times I \times C_v(t_n(v_2))\}$ denotes the set of interprocessor channels, and
- $d_a(c)$ denotes the *communication delay* for all $c = (((v_1,e_1),n_1), ((v_2,e_2),n_2)) \in C$ such that either $d_a(c) = d_v(t_n(v_1)) = d_v(t_n(v_2))$ if c is an intraprocessor channel, or $d_a(c) = d_v(t_n(v_1)) + d_n(v_1,v_2) + d_v(t_n(v_2))$ if c is an interprocessor channel.

For notational convenience we abbreviate a processing element network to (P,C) and we omit the subscripts of the functions p_a, m_a, n_a, t_a, s_a, and d_a. □

This definition completes our architecture model. In the remainder of this thesis we represent an architecture instance as a processing element network under the assumption that there exists an associated processor network.

2.2 Signal Flow Graphs

We consider video algorithms that are represented as signal flow graphs. They contain *operations* that communicate through *precedences* and *arrays*. The operations represent the basic functionality that is provided by the processing elements. They either transform input values into output values, or access arrays to read or write values. There are two types of precedences namely *data* precedences and *no-value*

precedences. Data precedences model *communication* between two operations. No-value precedences model *synchronization* between two operations.

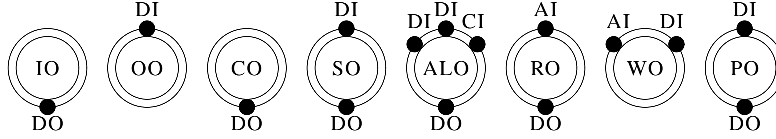


Figure 2.4. Graphical representation of the operation types. The double circles denote the operations, the closed circles above them denote the input terminals, and the closed circles below them denote the output terminals. The texts denote the types of the operations and the terminals.

We distinguish between several operation types in order to formulate type constraints for the mapping of operations onto processing elements. They are graphically depicted in Figure 2.4. To model the input and output of signal flow graphs, we introduce input operations (IO) that produce input values and output operations (OO) that consume output values. Furthermore, we have operations that transform input values into output values. To this category belong constant operations (CO) that produce constant values and shift operations (SO) that produce arithmetically and logically shifted values of the consumed input value. These operations can be executed on arithmetic and logic elements as well as buffer elements. In addition, this category contains arithmetic and logic operations (ALO) that model the instruction set of the arithmetic and logic elements. Next, we have operations that access arrays to read or write values. Here, we distinguish between read operations (RO) and write operations (WO). Read operations require an array index as input and produce a retrieved array value as output. Write operations require an array index and a value as input and store this value in the designated location of the array; they produce no output. Finally, we introduce pass operations (PO) that do not contribute to the functionality of a signal flow graph, but that are used during the mapping process for the communication and storage of values. Note that signal flow graphs do not contain operations that control switch matrices and silos. Formally, an operation type set is defined as follows.

Definition 2.6 (Operation Type Set). The tuple $(T_\text{o}, I_\text{o}, O_\text{o}, t_\text{o})$ defines a *operation type set*, where

- $T_\text{o} = \{\text{IO}, \text{OO}, \text{ALO}, \text{CO}, \text{SO}, \text{PO}, \text{RO}, \text{WO}\}$ denotes the set of *operation types*,
- $I_\text{o}(t)$ denotes a finite set of *input terminals* for all $t \in T_\text{o}$,
- $O_\text{o}(t)$ denotes a finite set of *output terminals* for all $t \in T_\text{o}$, and
- $t_\text{o}(t,n) \in T_\text{t}$ denotes the *terminal type* for all $t \in T_\text{o}$ and $n \in I_\text{o}(t) \cup O_\text{o}(t)$. □

Due to the repetitive nature of video signal processing, signal flow graph are typically repeated on successive input values. For this reason, the operations are periodic which means that they are invocated repeatedly at regular intervals in time. We adopt the periodic model of Korst [1992] who assumes that the operations have one dimension of repetition, rather than Verhaegh [1995] who has developed a multi-dimensional periodic model. The reason for this is that the mapping of multiple dimensions of invocation repetition onto one dimension of cyclostatic execution usually results in large programs. Since the program memories of our architecture can only contain one cycle of thirty-two instructions, the multi-dimensional periodic model does not offer significant advantages. We do allow that signal flow graphs are multi-rate which means that the operations can have different periods so that the number of invocations is different for various parts of a signal flow graph. We now formally define the notion of signal flow graph. See the text below for further explanation.

Definition 2.7 (Signal Flow Graph). The 8-tuple $(A, s_f, O, t_f, a_f, p_f, R, S)$ defines a *signal flow graph*, where

- A denotes a finite set of *arrays*,
- $s_f(a) \in \mathbb{Z}_+$ denotes the *array size* for all $a \in A$,
- O denotes a finite set of *operations*,
- $t_f(o) \in T_o$ denotes the *operation type* for all $o \in O$,
- $a_f(o) \in A$ denotes an *array* for all $o \in O$ for which $t_f(o) \in \{\text{RO}, \text{WO}\}$,
- $p_f(o) \in \mathbb{Z}_+$ denotes the *period* for all $o \in O$,
- $R \subseteq Y \times X \times (\mathbb{Z}_+ \times \mathbb{Z} \times \mathbb{Z})$ denotes a finite set of *data precedences*, where $Y = \{(o,n) \mid o \in O \wedge n \in O_o(t_f(o))\}$ denotes the set of output terminals and $X = \{(o,n) \mid o \in O \wedge n \in I_o(t_f(o))\}$ denotes the set of input terminals, such that for all $((o,n),(o',n'),(p,b,b')) \in R$ it holds that $p(o)|p$ and $p(o')|p$, and such that for all $r_1, r_2 \in R$ where $r_1 \neq r_2$, $r_1 = ((o_1,n_1),(o',n'),(p_1,b_1,b'_1))$, and $r_2 = ((o_2,n_2),(o',n'),(p_2,b_2,b'_2))$ it holds that
$$b'_1 \not\equiv b'_2 \pmod{\gcd(\frac{p_1}{p(o')}, \frac{p_2}{p(o')})}, \text{ and}$$
- $S \subseteq O \times O \times (\mathbb{Z}_+ \times \mathbb{Z} \times \mathbb{Z})$ denotes a finite set of *no-value precedences*, such that for all $(o,o',(p,b,b')) \in S$ it holds that $p(o)|p$ and $p(o')|p$.

For notational convenience we abbreviate a signal flow graph to (O,R) and we omit the subscripts of the functions s_f, t_f, a_f, and p_f. □

The arrays of a signal flow graph have a finite positive size and an implicitly associated address space that is consecutively numbered starting from zero. The array indices are generated at run-time as specified in the signal flow graph. Hence,

they may depend on run-time data. Arrays can have an initial filling that is specified at compile-time. Programmers often use this to implement look-up tables.

The operations of a signal flow graph are typed which among others determines the set of input and output terminals. The read and write operations access the arrays. Each read operation and each write operation operates on one designated array. In addition, the operations are periodic which indicates their average frequency. The period of an operation is given relatively compared to the clock period of the processors. Consequently, an operation o with period $p(o)$ occupies a fraction of a processing element that is equal to $1/p(o)$. We denote the kth invocation of an operation o as $o[k]$, for all integers k. Similarly, we denote the kth invocation of terminal n of operation o as $(o,n)[k]$, again for all integers k. Figure 2.5 shows a graphical representation of a periodic operation and its invocations.

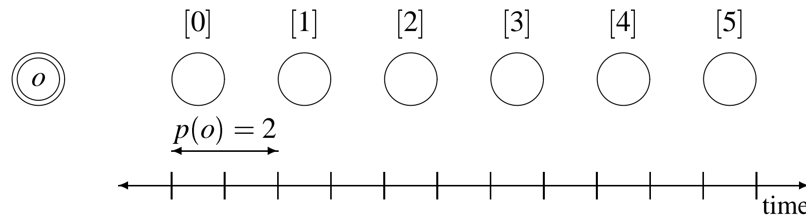


Figure 2.5. Graphical representation of a periodic operation. The double circles denote a periodic operation and the single circles denote its first six invocations. The average period between two successive operation invocations equals two.

The precedences of a signal flow graph relate invocations of a source operation o to invocations of a destination operation o'. The precedences contain triples (p,b,b') which are called *labels*. Here, the positive integer p denotes the *period* of the precedence which indicates the average frequency of the invocations. The periods of the precedences are also given relatively compared to the clock period of the processors. Hence, the invocations of a precedence with period p and a source operation with period $p(o)$ coincide with a period of $p/p(o)$. Similarly, the invocations of a precedence with period p and a destination operation with period $p(o')$ coincide with a period of $p/p(o')$. A necessary condition is that the periods of the operations divide the period of the precedence. The non-negative integers b and b' denote the *offsets* of the precedence that relate the first invocation of the precedence to the bth invocation of the source operation o and the b'th invocation of the destination operation o'. For notational convenience we denote the label of a precedence r sometimes by $(p(r),b(r),b'(r))$. Similarly, we denote the input and output terminals of a data precedence sometimes by $(o(r),n(r))$ and $(o'(r),n'(r))$, respectively. Figure 2.6 shows a graphical representation of a periodic precedence and its invocations.

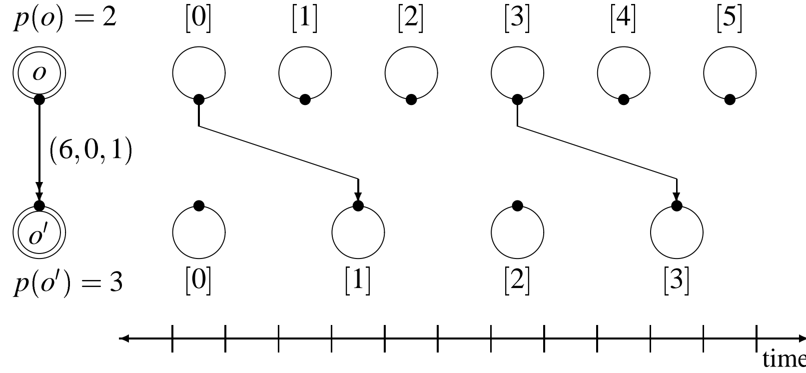


Figure 2.6. Graphical representation of a periodic precedence. The arcs with double arrows denotes a periodic precedence and the arcs with single arrows denote its first two invocations. The average period between two successive precedence invocations equals six.

Offsets are not unique in the sense that different offsets can denote the same precedence relation. For instance, if we increase the value of offset b with a multiple of $p/p(o)$ and the value of offset b' with the same multiple of $p/p(o')$, then we obtain the same precedence relation. To avoid the introduction of a canonical form we define the following equivalence relation.

Definition 2.8 (Precedence Equivalence). Let r_1 and r_2 be two precedences from source operation o to destination operation o', where $r_1 = (y,x,(p,b_1,b'_1))$ and $r_2 = (y,x,(p,b_2,b'_2))$. Then r_1 and r_2 are said to be *equivalent* if and only if

$$b_1 \equiv b_2 (\bmod \frac{p}{p(o)}) \wedge b'_1 \equiv b'_2 (\bmod \frac{p}{p(o')})$$

and

$$(b_1 - b_2)p(o) = (b'_1 - b'_2)p(o').$$

This is denoted by $r_1 = r_2$. The precedence $r_1 = (y,x,(p_1,b_1,b'_1))$ is said to *be included* by the precedence $r_2 = (y,x,(p_2,b_2,b'_2))$ if and only if $p = kp_1 = p_2$, for some positive integer k, rather than $p = p_1 = p_2$, in addition to the conditions mentioned above. This is denoted by $r_1 \subseteq r_2$. □

The first condition for equivalence is that at least one of the invocations of each of the precedences coincides at the source operation and at the destination operation. The second condition for equivalence is that the precedences periodically relate the same invocations of the source operation to the same invocations of the destination operation. The third condition for equivalence is that the precedences have the same period such that all invocations coincide. If the periods of the precedences are divisible, then only part of the invocations coincide which reduces the equivalence

relation to an inclusion relation.

A data precedence $((o,n),(o',n'),(p,b,b'))$ indicates that the output of the $(kp/p(o)+b)^{\text{th}}$ invocation of terminal (o,n) is input of the $(kp/p(o')+b')^{\text{th}}$ invocation of terminal (o',n'), for all integers k. It specifies a periodic sample stream in which the samples are ordered according to the invocations of the data precedence. The ordering is the same during the production of outputs and the consumption of inputs. Hence, data precedences represent first-in-first-out communication schemes. Other communication schemes such as last-in-first-out must be implemented with arrays. An additional necessary condition for the set of data precedences is that two different data precedences that denote different output values cannot be input at the same input terminal invocation.

A no-value precedence $(o,o',(p,b,b'))$ indicates that the $(kp/p(o)+b)^{\text{th}}$ invocation of operation o is completed before the $(kp/p(o')+b')^{\text{th}}$ invocation of operation o' is completed, again for all integers k. One often uses no-value precedences to synchronize the accesses on arrays. Because the array indices are computed at run-time as specified in the signal flow graph, it is the responsibility of the programmer of the signal flow graph to order the invocations of the accesses with no-value precedences in order to ensure that the array indices are valid. As an example we mention that write and read operations that periodically access the same location require synchronization, since a write access must precede a corresponding read access and, reversely, this read access must precede the next write access that overwrites the old array value.

Finally, we mention that the period of an entire signal flow graph is equal to the least common multiple of the periods of the operations and the periods of the precedences. If we divide that period by the period of an operation then we obtain the number of invocations of that operation during one period of the signal flow graph. Similarly, if we divide that period by the period of a precedence then we obtain the number of invocations of that precedence during one period of the signal flow graph. Note that signal flow graphs by definition satisfy the *balance equations* which state that during one period of a signal flow graph the number of productions must equal the number of consumptions for each precedence. The balance equations were introduced by Lee [1991] on a model of computation called data flow process networks. The model of signal flow graphs is comparable to the model of cyclostatic data flow; see Bilsen et al. [1994] and Parks et al. [1996].

2.3 Mappings

To simplify the formulation of a mapping of a signal flow graph onto a processing element network, we introduce two transformations which transform a given signal flow graph into a functionally equivalent signal flow graph. The purpose of these

transformations is to add pass operations to a signal flow graph which are subsequently mapped onto various types of processing elements in order to enable the communication and storage of values.

The purpose of the first transformation is to add pass operations to a signal flow graph in order to make the values that are communicated in specific production and consumption relations explicitly visible in the set of operations. We call the addition of pass operations to a signal flow graph *expansion* and the removal of pass operation from a signal flow graph *reduction*. Figure 2.7 shows an example of the transformation.

Definition 2.9 (Expansion/Reduction). Let k be a positive integer, let r be a precedence from operation o to operation o', let r_i be precedences from operation o' to operations o_i, and let r_i' be precedences from operation o to operations o_i, for all $i \in \mathbb{N}_k$, where $t(o') = \textsc{po}$, $r = ((o,n),(o',0),(p(o'),b,0))$, $r_i = ((o',0),(o_i,n_i),(p(o'),0,b_i'))$, and $r_i' = ((o,n),(o_i,n_i),(p(o'),b,b_i'))$. Then the set of precedences $\{r, r_0, \ldots, r_{k-1}\}$ is said to *expand* the set of precedences $\{r_0', \ldots, r_{k-1}'\}$, and the set of precedences $\{r_0', \ldots, r_{k-1}'\}$ is said to *reduce* the set of precedences $\{r, r_0, \ldots, r_{k-1}\}$. □

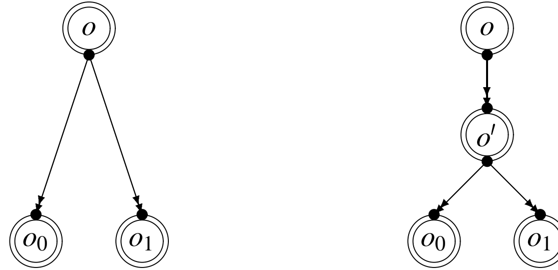


Figure 2.7. Example of expansion and reduction. The periods of the operations and precedences are equal to one. The precedences on the right expand the precedences on the left. The precedences on the left reduce the precedences on the right.

The purpose of the second transformation is to make the period of specific production and consumption patterns explicitly visible in the set of data precedences in such a way that all data precedences that denote a specific output value have the same period. The reason for this is to prevent the expansion of signal flow graphs with multiple pass operations that operate on the same value. Following the terminology used in signal processing theory, we call the increase of the period of precedences *decimation* and the decrease of the period of precedences *interpolation*. To formulate this transformation we use the precedence inclusion relation. Figure 2.8 shows an example of the transformation.

Definition 2.10 (Decimation/Interpolation). Let k be a positive integer, let r be a precedence from operation o to operation o', and let $r_i \subseteq r$ be precedences for all $i \in \mathbb{N}_k$, where $r = (y,x,(p,b,b'))$ and $r_i = (y,x,(kp,b+ip/p(o),b'+ip/p(o')))$. Then the set of precedences $\{r_0,\ldots,r_{k-1}\}$ is said to *decimate* the precedence r, and the precedence r is said to *interpolate* the set of precedences $\{r_0,\ldots,r_{k-1}\}$. □

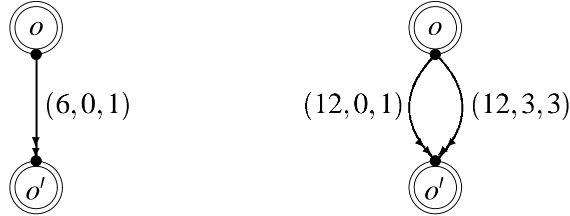


Figure 2.8. Example of decimation and interpolation. Operation o has period $p(o) = 2$ and operation o' has period $p(o') = 3$. The two precedences on the right decimate the precedence on the left. The precedence on the left interpolates the two precedences on the right.

The above-mentioned transformations change the structure of a signal flow graph, but they do not change its function. A programmer specifies a function without adding pass operations to a signal flow graph. During mapping we apply the transformations to change the structure which allows us to simplify the representation of a mapping of a signal flow graph onto a processing element network. The size of the representation of the signal flow graph increases if we apply decimation and expansion. It decreases if we apply interpolation and reduction. We now have the following inclusion relation on signal flow graphs.

Definition 2.11 (Signal Flow Graph Inclusion). Let (O,R) and (O',R') be two signal flow graphs. Then (O,R) is said to *be included* by (O',R') if and only if (O,R) can be transformed into (O',R') via decimation and expansion. This is denoted as $(O,R) \subseteq (O',R')$. Note that if (O,R) is included by (O',R') then (O',R') can be transformed into (O,R) via interpolation and reduction. □

Next, we formalize the mapping of a signal flow graph onto a processing element network. See the text below for an explanation of the symbols.

Definition 2.12 (Mapping). Let (O,R) be a signal flow graph and let (P,C) be a processing element network. Then the tuple $(\delta,\sigma,\alpha,\lambda)$ defines a *mapping*, where

- $\delta : O \to \mathbb{N}$ denotes a *delay assignment*,
- $\sigma : O \to \mathbb{Z}$ denotes a *time assignment*,
- $\alpha : O \to P$ denotes a *processing element assignment*, and
- $\lambda : R \to C$ denotes a *channel assignment*. □

A channel assignment maps each data precedence onto a communication channel. We assume that all the invocations of a data precedence are communicated along the same channel. Note that the channel assignment also specifies the processing element terminals through which the values are communicated.

A processing element assignment maps each operation onto a processing element. We assume that all the invocations of an operation are executed on the same processing element. Note that a processing element assignment does not specify a mapping of operation terminals onto processing element terminals. This is done by a channel assignment which allows different mappings of operation terminals onto processing element terminals for different invocations of an operation.

A time assignment maps each operation onto a time slot in which the first invocation of the operation is scheduled for execution. We assume strictly periodic execution of the operations, i.e., invocation k of operation o is scheduled for execution at time $\sigma(o) + kp(o)$. If the first invocation of operation o is scheduled at time $\sigma(o)$ for execution at processing element $\alpha(o)$, then the corresponding input of terminal n is required at time $\sigma(o) - d_{\mathrm{p}}(t(\alpha(o)), m)$, where m denotes the processing element terminal onto which operation terminal n is mapped.

A delay assignment maps each operation onto a time interval during which the outputs of the operation are stored in a compact silo before they arrive at an output terminal. More precisely, if the first invocation of operation o is scheduled at time $\sigma(o)$ for execution at processing element $\alpha(o)$ with a delay of $\delta(o)$, then the corresponding output of terminal n is available at time $\sigma(o) - d_{\mathrm{p}}(t(\alpha(o)), m) + \delta(o)p(o)$, where m denotes the processing element terminal onto which operation terminal n is mapped.

Note that we could have modeled the delay assignment as a mapping from data precedences, rather than operations, onto time intervals. In that case, the outgoing data precedences of an operation can have different delays. However, our architecture does not support the mapping of multiple data precedences with different delays onto a single instruction. Hence, each delay requires one instruction. We choose to make these instructions explicit by modeling them as operations in order to simplify the formulation of the constraints.

2.4 Constraints

The type constraints state that the certain types of operations and operation terminals must be mapped onto certain types of processing elements and processing element terminals. Furthermore the type constraints state that the delay of an operation is positive if and only if it is a pass operation that is mapped onto a dual ported memory element. The reason for this is twofold. First the other processing element types do not have instructions to implement positive delays. Second the

dual ported memory elements implement positive delays by means of an offset between the read and write address when a pass operation is translated into a pair of read and write operations. If this offset is zero, then the read and write operations access the same location but the old value is read before the new value is written. Hence, they cannot implement a zero delay on pass operations.

Definition 2.13 (Type Constraints). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\delta,\sigma,\alpha,\lambda)$ be a mapping. Then the *processing element type constraints* specify that for all $o \in O$ it holds that

$$t(\alpha(o)) \in \begin{cases} \{\text{ALE}\} & \text{if } t(o) \in \{\text{ALO}\} \\ \{\text{ALE},\text{BE}\} & \text{if } t(o) \in \{\text{CO},\text{SO}\} \\ \{\text{ALE},\text{BE},\text{OE}\} & \text{if } t(o) \in \{\text{PO}\} \wedge \delta(o) = 0 \\ \{\text{ME2}\} & \text{if } t(o) \in \{\text{PO}\} \wedge \delta(o) > 0 \\ \{\text{ME1},\text{ME2}\} & \text{if } t(o) \in \{\text{RO},\text{WO}\} \\ \{\text{IN}\} & \text{if } t(o) \in \{\text{IO}\} \\ \{\text{OUT}\} & \text{if } t(o) \in \{\text{OO}\}, \end{cases}$$

and the *terminal type constraints* specify that for all $r = ((o,n),(o',n'),(p,b,b')) \in R$ with $\lambda(r) = ((\alpha(o),m),(\alpha(o'),m'))$ it holds that $t_{\text{p}}(t(\alpha(o)),m) = t_{\text{o}}(t(o),n)$ and

$$t_{\text{p}}(t(\alpha(o')),m') \in \begin{cases} \{\text{CI}\} & \text{if } t_{\text{o}}(t(o'),n') \in \{\text{CI}\} \\ \{\text{DI}\} & \text{if } t_{\text{o}}(t(o'),n') \in \{\text{DI}\} \\ \{\text{RI},\text{AI}\} & \text{if } t_{\text{o}}(t(o'),n') \in \{\text{RI}\} \\ \{\text{WI},\text{AI}\} & \text{if } t_{\text{o}}(t(o'),n') \in \{\text{WI}\}. \end{cases}$$ □

Arithmetic and logic elements, buffer elements, and output elements can execute pass operations because their instruction sets contain an instruction that transfers the value of an input terminal to an output terminal without modification. Dual ported memory elements can execute pass operations because during code generation they are translated into pairs of read and write operations that execute simultaneously. It is not necessary to generate addresses for these read and write operations in the signal flow graph, because this is handled by the compact silo feature of the dual ported memory elements; see page 13. Since this hardware support is not available on single ported memory elements, we do not allow the mapping of pass operations on single ported memory elements.

The array constraints state that read and write operations that access the same array must be mapped onto the same processing element. This array is mapped onto the corresponding random access memory of the processing element.

Definition 2.14 (Array Constraints). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\delta,\sigma,\alpha,\lambda)$ be a mapping. Then the *array*

constraints specify that for all $o, o' \in O$ it holds that if $a(o) = a(o')$ then

$$\alpha(o) = \alpha(o').$$ □

The storage constraints state that the storage requirements for arrays and delays that are mapped onto a processing element cannot exceed the size of the corresponding random access memory. For notational convenience we introduce abbreviations for the subset $O_P = \{o \in O \mid \alpha(o) \in P\}$ of operations and the subset $A_P = \{a(o) \in A \mid o \in O_P\}$ of arrays that are mapped onto a set P of processing elements as specified by a given processing element assignment α.

Definition 2.15 (Storage Constraints). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\delta,\sigma,\alpha,\lambda)$ be a mapping. Then the *storage constraints* specify that for all $p \in P$ it holds that

$$\sum_{a \in A_{\{p\}}} s(a) + N\Big(\sum_{o \in O_{\{p\}}} \delta(o)\Big) \leq s(p),$$

where the function N, which is defined by

$$N(x) = \begin{cases} 0 & \text{if } x = 0 \\ 2^{\lceil {}^2\log(1+x) \rceil} & \text{if } x \neq 0, \end{cases}$$

denotes the number of memory locations that are required for the delays. □

The periodicity constraints state that the length of a program cannot exceed the size of the designated program memory. The length of a program is equal to the least common multiple of the periods of the operations and data precedences that are controlled by the program. This length determines the period between the successive executions of a program.

Definition 2.16 (Periodicity Constraints). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\delta,\sigma,\alpha,\lambda)$ be a mapping. Then the *periodicity constraints* specify that for all $m \in M$ it holds that

$$\text{lcm}\{p_O(m), p_R(m)\} \leq p(m).$$

where $p_O(m) = \text{lcm}\{p(o) \mid o \in O \wedge m(\alpha(o)) = m\}$ and $p_R(m) = \text{lcm}\{p(r) \mid r = ((o,n),(o',n'),(p,b,b')) \in R \wedge m(\alpha(o')) = m\}$. □

The connectivity constraints state that operations that are related by a data precedence must be mapped onto processing elements that are connected by a channel. Furthermore, the data precedence must be mapped onto one of the channels that connect the processing elements.

Definition 2.17 (Connectivity Constraints). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\delta,\sigma,\alpha,\lambda)$ be a mapping. Then the *connectivity constraints* specify that for all $r = ((o,n),(o',n'),(p,b,b')) \in R$ there

exist an output terminal $m \in O_\mathrm{p}(t(\alpha(o)))$ and an input terminal $m' \in I_\mathrm{p}(t(\alpha(o')))$ such that it holds that

$$\lambda(r) = ((\alpha(o),m),(\alpha(o'),m')). \qquad \square$$

The precedence constraints state that the producer and consumer relations must be valid, i.e., production must take place before consumption and consumption must take place before the next production in the same location. The producer and consumer relations through arrays may depend on run-time data. For this reason, it is the responsibility of the programmer of the signal flow graph to constrain the set of possible time assignments with no-value precedences in order to ensure that the array indices are valid. Note that dual ported memory elements can execute read and write accesses on the same location simultaneously, but that the old value is read before the new value is written. The producer and consumer relations specified by data precedences do not depend on run-time data. These values are stored in silos or compact silos which have specific addressing schemes that allow us to formulate additional data precedence constraints.

In order to formalize the precedence constraints we introduce some additional notation. With each data precedence $r = ((o,n),(o',n'),(p,b,b'))$ we associate a storage time in a silo $\chi(r) - \psi(r)$ that is equal to the time of departure from the silo, denoted as

$$\chi(r) = \sigma(o') + b'p(o') - d_\mathrm{p}(t(\alpha(o')),m'),$$

minus the time of arrival at the silo, denoted as

$$\psi(r) = \sigma(o) + (b + \delta(o))p(o) - d_\mathrm{p}(t(\alpha(o)),m) + d_\mathrm{a}(\lambda(r)),$$

where $\lambda(r) = ((\alpha(o),m),(\alpha(o'),m'))$. If there is no silo, then the storage time must equal zero. Otherwise the storage time must be in the time interval $(0,\ldots,n(\alpha(o')))$ not including the boundaries because one cannot access a location of a silo for reading and writing at the same time. We introduce similar notation for the no-value precedences. With each no-value precedence $s = (o,o',(p,b,b'))$ we associate a storage time in an array $\chi(s) - \psi(s)$, that is equal to the departure time from the array, denoted as

$$\chi(s) = \sigma(o') + b'p(o'),$$

minus an arrival time at the array, denoted as

$$\psi(s) = \sigma(o) + bp(o).$$

The departure and arrival times are given for the precedence invocations with number zero.

Definition 2.18 (Precedence Constraints). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\delta,\sigma,\alpha,\lambda)$ be a mapping. Then the

data precedence constraints specify that for all $r = ((o,n),(o',n'),(p,b,b')) \in R$ with $t(\alpha(o')) = \text{OUT}$ it holds that

$$\psi(r) = \chi(r),$$

and for all other $r = ((o,n),(o',n'),(p,b,b')) \in R$ it holds that

$$0 < \chi(r) - \psi(r) < n(\alpha(o')),$$

and the *no-value precedence constraints* specify that for all $s = (o,o',(p,b,b')) \in S$ with $t(o) = \text{RO}$, $t(o') = \text{WO}$, and $t(\alpha(o)) = t(\alpha(o')) = \text{ME2}$ it holds that

$$\psi(s) \leq \chi(s),$$

and for all other $s = (o,o',(p,b,b')) \in S$ it holds that

$$\psi(s) < \chi(s). \qquad \square$$

We rewrite the precedence constraint as linear constraints in the time assignment. To this end we introduce some additional notation. With each predecence a, which can be either a data precedence $a = ((o,n),(o',n'),(p,b,b'))$ or a no-value precedence $a = (o,o',(p,b,b'))$, we associate an arrival delay $\omega(a) = \psi(a) - \sigma(o)$ and a departure delay $\omega'(a) = \sigma(o') - \chi(a)$. Furthermore, we represent the interval of potential storage times between data arrival and data departure as a set. For each data precedences $a = ((o,n),(o',n'),(p,b,b'))$ satisfying $t(\alpha(o')) = \text{OUT}$ and for each no-value precedence $a = (o,o',(p,b,b'))$ satisfying $t(o) = \text{RO}$, $t(o') = \text{WO}$, and $t(\alpha(o)) = t(\alpha(o')) = \text{ME2}$ this set is equal to $W(a) = \{0\}$. For all other precedences a this set is equal to $W(a) = \{1, n(\alpha(o')) - 1\}$. With these notions we define lower and upper bounds between the completion times of two successive operations as follows. For any precedence a the lower bound equals

$$\underline{\omega}(a) = \omega(a) + \omega'(a) + \min W(a),$$

and the upper bound equals

$$\overline{\omega}(a) = \omega(a) + \omega'(a) + \max W(a).$$

We now reformulate the precedence constraints as follows.

Definition 2.19 (Precedence Constraints Reformulated). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\delta,\sigma,\alpha,\lambda)$ be a mapping. Then the *data precedence constraints* specify that for all $r = ((o,n),(o',n'),(p,b,b')) \in R$ it holds that

$$\underline{\omega}(r) \leq \sigma(o') - \sigma(o) \leq \overline{\omega}(r),$$

and the *no-value precedence constraints* specify that for all $s = (o,o',(p,b,b')) \in S$ it holds that

$$\underline{\omega}(s) \leq \sigma(o') - \sigma(o). \qquad \square$$

The computation constraints state that two different operations cannot occupy the same processing element at the same time, unless it involves a read and write operation that occupy a dual ported memory element.

Definition 2.20 (Computation Constraints). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\delta,\sigma,\alpha,\lambda)$ be a mapping. Then the *computation constraints* specify that for all $o,o' \in O$ with $o \neq o'$ and $\alpha(o) = \alpha(o')$ it holds that

$$\sigma(o) \not\equiv \sigma(o') \pmod{\gcd(p(o),p(o'))},$$

unless $t(o) = \textsc{ro}$, $t(o') = \textsc{wo}$, and $t(\alpha(o)) = t(\alpha(o')) = \textsc{me2}$. □

The communication constraints state that two different data values cannot occupy the same silo location at the same time. For this reason, two different data values that are stored in the same silo cannot have the same arrival time.

Definition 2.21 (Communication Constraints). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\delta,\sigma,\alpha,\lambda)$ be a mapping. Then the *communication constraints* specify that for all data precedences $r_1,r_2 \in R$ where $r_1 = ((o_1,n_1),(o'_1,n'_1),(p_1,b_1,b'_1))$ and $r_2 = ((o_2,n_2),(o'_2,n'_2),(p_2,b_2,b'_2))$ with $\lambda(r_1) = ((\alpha(o_1),m_1),(\alpha(o'_1),m'_1))$ and $\lambda(r_2) = ((\alpha(o_2),m_2),(\alpha(o'_2),m'_2))$ it holds that if $(\alpha(o_1),m_1) \neq (\alpha(o_2),m_2)$ and $(\alpha(o'_1),m'_1) = (\alpha(o'_2),m'_2)$ then

$$\psi(r_1) \not\equiv \psi(r_2) \pmod{\gcd(p_1,p_2)}.$$ □

A mapping is called *feasible* if it satisfies the type, array, storage, periodicity, connectivity, precedence, computation, and communication constraints.

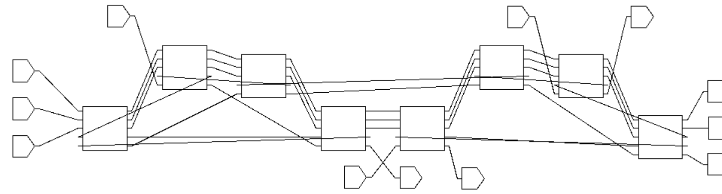
2.5 Mapping Problem

We can now formally state the mapping problem as follows.

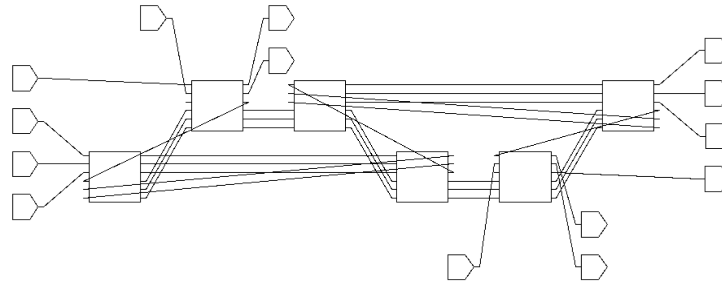
Definition 2.22 (Mapping Problem). Let (O,R) be a signal flow graph and let (P,C) be a processing element network. Find a signal flow graph (O',R') such that $(O,R) \subseteq (O',R')$ and find a feasible mapping $(\delta,\sigma,\alpha,\lambda)$ of (O',R') onto (P,C), i.e., one that satisfies the type, array, storage, periodicity, connectivity, precedence, computation, and communication constraints, if they exist. □

2.6 Problem Instances

In this section we present a set of industrially relevant problem instances. The set of processor networks contains networks of first and second generation video signal processors that have been designed especially for the development and prototyping of video applications. The first processor network called VSP1FLEX [Tregnago



(a) The VSP1FLEX processor network.



(b) The VSP2FLEX processor network.

Figure 2.9. Graphical representation of the VSP1FLEX and VSP2FLEXprocessor networks in the VSP programming environment. Squares represent video signal processors with inputs on the left and outputs on the right, pentagons represent input and output processors, and lines represent interconnections.

et al., 1992] is shown in Figure 2.9a. It contains eight first generation video signal processors. The second processor network called VSP2FLEX [Riddersma et al., 1996] is shown in Figure 2.9b. It contain six second generation video signal processors. The last processor network called VSP2TEST [La Hei et al., 1996] consists of a socket for a single second generation video signal processor. This processor network has been developed to test new processors after manufacturing.

The signal flow graphs are consumer applications for camera and television systems and professional applications for medical systems. The signal flow graph listed in Table 2.3 have been developed on the VSP1FLEX processor network. They implement television functionality such as color space conversion, contour enhancement, gamma correction, histogram modification, and aspect ratio conversion. We map these signal flow graph also on the VSP2TEST processor network because a single second generation processor has the same processing capacity compared to eight first generation processors. The signal flow graph listed in Table 2.4 have been developed on the VSP2FLEX processor network. They imple-

Table 2.3. Signal flow graphs that are to be mapped onto the VSP1FLEX and VSP2TEST processor networks. From left to right the columns indicate the instance name, the number of arrays, the number of operations, the number of data precedences, the number of no-value precedences, and the period of the signal flow graph for both processor networks.

SFG	$\|A\|$	$\|O\|$	$\|R\|$	$\|S\|$	Period
YUVTORGB	0	37	76	0	2 (4)
HORCOMPR	4	66	185	0	16 (32)
IJNTEMA1	5	140	307	32	16 (32)
CORMACK2	6	127	265	0	16 (32)
CONTOUR1	7	63	171	7	16 (32)
MONZA2	8	69	156	10	16 (32)
VDP	9	129	360	16	16 (32)
GAMMA	10	212	405	39	16 (32)
HISTMOD2	11	177	355	39	16 (32)
PANORAMA	12	86	239	8	16 (32)
VIDIWALL	13	141	398	20	16 (32)
IJNTEMA2	14	249	529	18	16 (32)
CORMACK1	15	186	444	78	48 (96)
MONZA1	34	236	512	68	16 (32)
MWTV	48	376	680	40	16 (32)

Table 2.4. Signal flow graphs that are to be mapped onto the VSP2FLEX processor network. From left to right the columns indicate the instance name, the number of arrays, the number of operations, the number of data precedences, the number of no-value precedences, and the period of the signal flow graph.

SFG	$\|A\|$	$\|O\|$	$\|R\|$	$\|S\|$	Period
CONTRAST	3	277	501	0	32
FDXD1	3	136	414	2	32
FDXD2	5	134	424	0	32
MAT30UP	8	139	300	8	24
HSRC	11	312	762	144	32
VSRC	12	285	926	28	32

ment functionality for camera, medical, and television systems such as contrast enhancement, x-ray image improvement, camera image improvement, and horizontal and vertical sample rate conversion. During the mapping of the signal flow graphs onto the processor networks the number of operations and the number of data precedences changes as a result of the signal flow graph transformations. The other characteristics do not change during mapping.

2.7 Summary

In this chapter we have presented a model to describe instances of the mapping problem. To this end we have formalized the notions of processor network and signal flow graph. Subsequently, we have formalized the decisions that are required to map a signal flow graph onto a processor network. Finally, we have formalized the constraints that indicate which combinations of decisions are feasible. To illustrate the nature of typical problem instances we have presented a set of industrially relevant processor networks and signal flow graphs.

3

Complexity Analysis

Due to the discrete nature of the mapping problem one can formulate it as a *combinatorial decision* or *combinatorial optimization* problem which allows the investigation of its computational complexity using the theory of NP-completeness. In Section 3.1 we present a short introduction to this theory. In Section 3.2 up to 3.8 we investigate the computational complexity of several relaxations of the mapping problem in order to determine which constraints are easy and which constraints are hard to satisfy. We use the results of these investigations in the next chapter to decompose the mapping problem such that we can handle the subproblems more effectively than the complete mapping problem. Finally, in Section 3.9 we summarize the contents of this chapter.

3.1 Computational Complexity

The theory of NP-completeness considers the difference between easy and hard problems. Cook [1971], Karp [1972], and Levin [1973] have been the first to formalize the theory, which is based on a computing model that is known as the *Turing machine*; see Aho et al. [1974]. For an overview of the theory we refer to Garey and Johnson [1979]. For uniformity, the theory is designed to be applied to decision problems.

A combinatorial decision problem consists of a set of problem instances and a decision function that assigns ‘yes’ or ‘no’ to each instance. If the answer to a given instance is ‘yes’, then the instance is called a ‘yes’ instance. Similarly, if the answer to a given instance is ‘no’, then the instance is called a ‘no’ instance. The problem is to determine whether a given problem instance is a ‘yes’ instance.

Usually, an instance of a combinatorial decision problem is given in terms of a set of variables, a set of domains that specifies the values that may be assigned to the variables, and a set of constraints that specifies which combinations of domain values are feasible. The problem is to find a solution, i.e., an assignment to the variables that satisfies the constraints.

An instance of a combinatorial optimization problem consists of a finite or countably infinite set of solutions and a cost function that assigns a cost to each solution. The problem is to find an optimum solution, which can be either a solution with minimum cost in case of a minimization problem, or a solution with maximum cost in case of a maximization problem.

In order to reason about the computational complexity of combinatorial problems and algorithms we introduce the notions of *instance size* and *time complexity function*. The size of a combinatorial decision or optimization problem instance is defined as the number of symbols required to represent the instance in a compact way. The time complexity function of a given algorithm defines for each possible instance size the maximum amount of time needed for solving instances of that size.

An algorithm is called *polynomial-time* if and only if there exists a polynomial function that bounds the time complexity function of the algorithm. Otherwise, the algorithm is called a *superpolynomial-time* or *exponential*. A problem is called solvable in polynomial time if and only if there exists a polynomial algorithm for the problem. Informally, problems that are solvable in polynomial time are called easy, whereas problems that are not solvable in polynomial time are called hard.

The class $\mathcal{P}$ is the set of decision problems that are solvable in polynomial time. The class $\mathcal{NP}$ is the set of decision problems for which each ‘yes’ instance has a *concise certificate*, which is an amount of data that is polynomial in the instance size such that it is possible to verify in polynomial time that an instance is a ‘yes’ or ‘no’ instance. In addition there exists the class $\mathcal{NPC}$ of NP-complete problems which are the hardest ones in $\mathcal{NP}$. To relate the computational complexity of two decision problems we use the concept of *reducibility*.

Definition 3.1 (Polynomial-Time Reduction). A problem $\pi \in \mathcal{NP}$ is *polynomially reducible* to another problem $\pi' \in \mathcal{NP}$ if and only if there exists a polynomial-time algorithm that maps instances i of problem π onto instances i' of problem π', such that i is a ‘yes’ instance for π if and only if i' is a ‘yes’ instance for π'. □

A problem is in $\mathcal{NPC}$ if and only if it is in $\mathcal{NP}$ and each problem in $\mathcal{NP}$ is polynomially reducible to it. To prove that a problem is NP-complete, it suffices to show that it is in $\mathcal{NP}$ and that some NP-complete problem is polynomially reducible to it.

Note that if one NP-complete problem can be solved in polynomial time, then all NP-complete problems can be solved in polynomial time, which means that $\mathcal{P} = \mathcal{NP}$. However, it is generally believed that $\mathcal{P} \neq \mathcal{NP}$ because nobody has succeeded in finding a polynomial-time algorithm for an NP-complete problem, despite many efforts since the development of the theory.

If a problem is NP-complete, then it cannot be solved in polynomial time, unless $\mathcal{P} = \mathcal{NP}$. However, it is possible that the problem can be solved in *pseudo-polynomial time* by an algorithm with a time complexity function that is bounded by the size of the instance and the magnitude of the largest integer that occurs in the compact representation of the instance. Problems are called *number problems* if this magnitude is not bounded by a polynomial in the instance size. So if an NP-complete problem is not a number problem, then it cannot be solved in pseudo-polynomial time, unless $\mathcal{P} = \mathcal{NP}$. An NP-complete problem is called *NP-complete in the strong sense* if it has an NP-complete subproblem that contains the instances for which the magnitude of the largest integer is bounded by a polynomial in the instance size. Otherwise, it is called *NP-complete in the ordinary sense*.

The computational complexity of combinatorial decision problems and their optimization variants are often closely related. This is for instance the case if the optimization variant can be formulated as a sequence of decision variants and the length of the sequence is bounded by a polynomial in the size of the instance and the size of the largest integer that occurs in the compact representation of the instance. In addition to decision and optimization variants we also consider a feasibility variant of a combinatorial problem in which the problem is to construct a feasible solution if the instance is a 'yes' instance. If a combinatorial decision problem is NP-complete, then the feasibility and the optimization variant are called NP-hard.

In order to illustrate the differences between decision, feasibility, and optimization variants we formulate different variants of the mapping problem. In the decision variant the question is whether there exists a feasible mapping for a given signal flow graph and a given processing element network. In the feasibility variant the question is to construct a feasible mapping for a given signal flow graph and a given processing element network, if such a mapping exists. Depending on the optimization criterion, the question in an optimization variant might be to determine the minimum number of processing elements that is required to map a given signal flow graph. In this thesis we consider the feasibility variant of the mapping problem; see Definition 2.22.

3.2 Type Constraints

To satisfy the type constraints of the mapping problem we must assign the operations to processing elements and delays in such a way that the function of each operation is supported by the processing element on which it is executed. In addition we must assign each data precedence to a communication channel in such a way that the terminal types of the operations and processing elements match. We show that the problem of constructing a delay, processing element, and channel assignment that satisfy the type constraints is solvable in polynomial time.

Theorem 3.1. *The problem of constructing a delay, processing element, and channel assignment that satisfy the type constraints is solvable in polynomial time.*

Proof. To construct a processing element assignment α we check for each operation whether there exists a processing element with a feasible type. This can be done in at most $|O| \cdot |P|$ steps. To construct a channel assignment λ we check for each data precedence whether there exists a channel with feasible terminal types. This can be done in at most $|R| \cdot |C|$ steps. If there is an operation for which such a processing element does not exist or if there is a data precedence for which such a channel does not exist, then the instance is infeasible. To construct a delay assignment δ either we define $\delta(o) = 1$ if $t(o) = \text{PO}$ and $t(\alpha(o)) = \text{ME2}$, or we define $\delta(o) = 0$ otherwise. Hence, the assignments can be constructed in polynomial time, if they exist. □

3.3 Array Constraints

To satisfy the array constraints we must assign the operations to processing elements in such a way that two operations that operate on the same array are assigned to the same processing element. We show that the problem of constructing a processing element assignment that satisfies the array constraints is solvable in polynomial time.

Theorem 3.2. *The problem of constructing a processing element assignment that satisfies the array constraints is solvable in polynomial time.*

Proof. To construct a processing element assignment α that satisfies the array constraints we assign all operations to the same processing element. If the set of processing elements is empty, then the instance is infeasible. Hence, the processing element assignment can be constructed in polynomial time, if it exists. □

3.4 Storage Constraints

To satisfy the storage constraints we must assign the arrays and delays to memories in such a way that the storage requirements do not exceed the storage capacities.

We show that the problem of constructing a mapping that satisfies the storage constraints is NP-hard using a polynomial-time reduction from bin packing.

Definition 3.2 (Bin Packing). Let U be a set of items, let $s(u) \in \mathbb{Z}_+$ denote the size of each $u \in U$, let V be a set of bins, and let B denote the size of the bins. Find a packing $f : U \rightarrow V$ such that for all $v \in V$ it holds that $\sum_{u \in U, f(u)=v} s(u) \leq B$, if one exists. □

Theorem 3.3. *The problem of constructing a processing element assignment that satisfies the storage constraints is NP-hard in the strong sense.*

Proof. For a given instance and a given solution we can verify in polynomial time whether the storage constraints are satisfied. Hence, the problem is in $\mathcal{NP}$. To prove that the problem is NP-hard we use a reduction from bin packing which is known to be NP-hard in the strong sense [Garey and Johnson, 1979].

Given an arbitrary instance of bin packing as formulated in Definition 3.2, we construct a corresponding instance of the mapping problem as follows. For each item $u \in U$ we define an array $a_u \in A$ with size $s(a_u) = s(u)$ that contains one operation $o_u \in O$. In addition we define for each bin $v \in V$ a memory element $p_v \in P$ with size $s(p_v) = B$. There are no precedences.

Suppose we have a solution f to the given instance of the bin packing problem. Then we assign each operation o_u to processing element p_v, i.e., we construct a processing element assignment α by defining $\alpha(o_u) = p_v$ if and only if $f(u) = v$. The constructed processing element assignment satisfies the storage constraints because f satisfies the bin packing constraints and the sizes of the arrays equal the sizes of the items.

Suppose we have a solution α to the constructed instance of the mapping problem. Then we assign each item u to bin v, i.e., we construct a packing f by defining $f(u) = v$ if and only if $\alpha(o_u) = p_v$. The constructed packing f satisfies the bin packing constraints because α satisfies the storage constraints and the sizes of the items equal the sizes of the arrays. □

Bin packing with a fixed bin size is solvable in polynomial time by exhaustive search [Garey and Johnson, 1979]. The proof of Theorem 3.3 implies that the problem of finding a processing element assignment that satisfies the array and storage constraints can be transformed into a bin packing problem if the memory size is fixed. Hence the problem of finding a processing element assignment that satisfies the array and storage constraints is solvable in polynomial time for fixed memory sizes by exhaustive search. However this is only practical for small memory sizes.

Corollary 3.1. *The problem of constructing a processing element assignment that satisfies the array and storage constraints is solvable in polynomial time for fixed memory sizes.* □

3.5 Computation and Communication Constraints

To satisfy the computation and communication constraints we must schedule strictly periodic invocations of operations and precedences over time on a set of processing elements and communication channels. Korst [1992] has studied this area of periodic scheduling extensively. He considers two cases in which either the resource assignment or the time assignment is given. These cases are called strictly periodic constrained time assignment and strictly periodic constrained resource assignment, respectively.

3.5.1 Strictly Periodic Constrained Time Assignment.

First we formulate the problem of finding a time assignment satisfying the computation constraints under the assumption that we are given a processing element assignment.

Definition 3.3 (Strictly Periodic Constrained Time Assignment). Let O be a set of strictly periodic operations, let each operation $o \in O$ have a period $p(o) \in \mathbb{Z}_+$, let P be a set of processing elements, and let $\alpha : O \to P$ be a processing element assignment. Find a time assignment $\sigma : O \to \mathbb{Z}$ such that for all $o, o' \in O$ it holds that if $o \neq o'$ and $\alpha(o) = \alpha(o')$ then $\sigma(o) \not\equiv \sigma(o') \pmod{\gcd(p(o), p(o'))}$, if one exists. □

Korst [1992] has shown that the strictly periodic constrained time assignment problem is NP-hard in the strong sense if the operations can occupy the processing elements for arbitrary times. Definition 3.3 captures the instances resulting from the mapping problem in which the occupation times are always equal to one. For this reason, the above-mentioned result on the complexity of the problem does not apply. However, Korst [1992] has also shown that the special cases in which the periods and the occupation times are divisible sequences are solvable in polynomial time. These results are based on bin packing with divisible item sizes which is known to be solvable in polynomial time [Coffman et al., 1987].

Theorem 3.4. *The problem of constructing a time assignment that satisfies the computation constraints if the periods form a divisible sequence is solvable in polynomial time.*

Proof. See Korst [1992], Theorem 4.7, page 82-85. □

Similarly we have the problem of finding a time assignment satisfying the communication constraints under the assumption that we are given a channel assignment. This problem is also easy if the periods form a divisible sequence.

Definition 3.4 (Strictly Periodic Channel Constrained Time Assignment). Let R be a set of strictly periodic data precedences, let each data precedence $r \in R$ have

a period $p(r) \in \mathbb{Z}_+$, let C be a set of communication channels, and let $\lambda : R \to C$ be a channel assignment. Find a data arrival time assignment $\psi : R \to \mathbb{Z}$ such that for all data precedences $r_1, r_2 \in R$ with $\lambda(r_1) = ((p_1, m_1), (p'_1, m'_1))$ and $\lambda(r_2) = ((p_2, m_2), (p'_2, m'_2))$ it holds that if $(p_1, m_1) \neq (p_2, m_2)$ and $(p'_1, m'_1) = (p'_2, m'_2)$ then $\psi(r_1) \not\equiv \psi(r_2) \pmod{\gcd(p(r_1), p(r_2))}$, if one exists. □

Theorem 3.5. *The problem of constructing a time assignment that satisfies the communication constraints if the periods form a divisible sequence is solvable in polynomial time.*

Proof. Similar to the proof of Theorem 3.4. □

These results have two corollaries which formulate necessary and sufficient conditions for the existence of time assignments that satisfy the computation and the communication constraints. We use these corollaries to reduce the search space of the mapping problem. The first corollary concerning the computation constraints bounds the number of operations that can be executed on the same processing element. The second corollary concerning the communication constraints bounds the number of samples that can be consumed by an input terminal of a processing element. In this corollary the number $q(o)$ represents the smallest number of invocations of operation o that is required to repeat the communication behavior between operation o and its consumers. The number is zero if operation o has no consumers.

Corollary 3.2. *Let (O, R) be a signal flow graph, let (P, C) be a processing element network, and let $(\delta, \sigma, \alpha, \lambda)$ be a mapping. Then there exists a time assignment that satisfies the computation constraints only if for all $p \in P$ it holds that*

$$\max\{\sum_{\substack{o \in O, \alpha(o) = p \\ \{t(o), t(p)\} \neq \{\mathrm{RO}, \mathrm{ME2}\}}} \frac{1}{p(o)}, \sum_{\substack{o \in O, \alpha(o) = p \\ \{t(o), t(p)\} \neq \{\mathrm{WO}, \mathrm{ME2}\}}} \frac{1}{p(o)}\} \leq 1.$$

If the periods form a divisible sequence then these conditions are sufficient. □

Corollary 3.3. *Let (O, R) be a signal flow graph, let (P, C) be a processing element network, and let $(\delta, \sigma, \alpha, \lambda)$ be a mapping. Then there exists a time assignment that satisfies the communication constraints only if for all $((p, m), (p', m')) \in C$ it holds that*

$$\sum_{\substack{r = ((o, n), (o', n'), (p, b, b')) \in R \\ \lambda(r) = ((p'', m''), (p', m'))}} \frac{|\{i \in \mathbb{N}_{q(o)} \mid b \equiv i \pmod{\frac{p}{p(o)}}\}|}{p(o) q(o)} \leq 1,$$

where $q(o) = \mathrm{lcm}\{\frac{p}{p(o)} \mid ((o, n), (o', n'), (p, b, b')) \in R\}$. If the periods form a divisible sequence then these conditions are sufficient. □

3.5.2 Strictly Periodic Constrained Resource Assignment.

Next we formulate the problem of finding a processing element assignment satisfying the the computation constraints under the assumption that we are given a time assignment. Korst [1992] calls this problem the strictly periodic constrained processor assignment problem. Similarly we have the problem of finding a channel assignment satisfying the communication constraints under the assumption that we are given a time assignment.

Definition 3.5 (Strictly Periodic Constrained Processor Assignment). Let O be a set of strictly periodic operations, let each operation $o \in O$ have a period $p(o) \in \mathbb{Z}_+$, let $\sigma : O \to \mathbb{Z}$ be a time assignment, and let P be a set of processing elements. Find a processing element assignment $\alpha : O \to P$ such that for all $o, o' \in O$ it holds that if $o \neq o'$ and $\sigma(o) \equiv \sigma(o') \pmod{\gcd(p(o), p(o'))}$ then $\alpha(o) \neq \alpha(o')$. □

Definition 3.6 (Strictly Periodic Constrained Channel Assignment). Let R be a set of strictly periodic data precedences, let each data precedence $r \in R$ have a period $p(r) \in \mathbb{Z}_+$, let $\psi : R \to \mathbb{Z}$ be a data arrival time assignment, and let C be a set of communication channels. Find a channel assignment $\lambda : R \to C$ such that for all data precedences $r_1, r_2 \in R$ with $\lambda(r_1) = ((p_1, m_1), (p'_1, m'_1))$ and $\lambda(r_2) = ((p_2, m_2), (p'_2, m'_2))$ it holds that if $(p_1, m_1) \neq (p_2, m_2)$ and $\psi(r_1) \equiv \psi(r_2) \pmod{\gcd(p(r_1), p(r_2))}$ then $\lambda(r_1) \neq \lambda(r_2)$. □

Korst [1992] shows that the strictly periodic constrained processor assignment problem is NP-hard in the strong sense using a reduction from graph coloring. The same result applies to the strictly periodic constrained channel assignment problem.

Theorem 3.6. *For a given time assignment the problem of constructing a processing element assignment that satisfies the computation constraints is NP-hard in the strong sense.*

Proof. See Korst [1992], Theorem 3.10, page 66-67. □

Theorem 3.7. *For a given data arrival time assignment the problem of constructing a channel assignment that satisfies the communication constraints is NP-hard in the strong sense.*

Proof. Similar to the proof of Theorem 3.6. □

The special cases in which the periods of the operations and the precedences form a divisible sequence are solvable in polynomial time. In these cases the required number of processing elements is equal to the largest number of operation invocations that must be completed simultaneously. Similarly, the required number

of input ports is equal to the largest number of samples that must be consumed simultaneously.

We summarize these results in two corollaries which define necessary and sufficient conditions for the existence of a processing element assignment that satisfies the computation constraints and for the existence of a channel assignment that satisfies the communication constraints. We use these corollaries to reduce the search space of the mapping problem. The first corollary bounds the number of operations that can be completed simultaneously by a given set of processing elements. The second corollary bounds the number of samples that can be consumed simultaneously by a given set of input terminals.

Corollary 3.4. *Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\delta,\sigma,\alpha,\lambda)$ be a mapping. Then there exists a processing element assignment that satisfies the computation constraints only if for all $i \in \mathbb{Z}$ it holds that*

$$|\{o \in O \mid \sigma(o) \equiv i \ (\mathrm{mod}\, p(o)) \wedge \{t(o), t(\alpha(o))\} \neq \{\textsc{ro}, \textsc{me2}\}\}| \leq |P|,$$

and

$$|\{o \in O \mid \sigma(o) \equiv i \ (\mathrm{mod}\, p(o)) \wedge \{t(o), t(\alpha(o))\} \neq \{\textsc{wo}, \textsc{me2}\}\}| \leq |P|.$$

If the periods form a divisible sequence then these conditions are sufficient. □

Corollary 3.5. *Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\delta,\sigma,\alpha,\lambda)$ be a mapping. Then there exists a channel assignment that satisfies the communication constraints only if for all $i \in \mathbb{Z}$ it holds that*

$$|\{(o(r), n(r)) \mid r \in R \wedge \psi(r) \equiv i \ (\mathrm{mod}\ p(r))\}| \leq |\{(p', m') \mid ((p,m),(p',m')) \in C\}|.$$

If the periods form a divisible sequence then these conditions are sufficient. □

3.6 Periodicity Constraints

The reduction from graph coloring to strictly periodic constrained processor assignment [Korst, 1992] does not consider the periodicity constraints. Instead the period of a program follows by construction from the resulting processing element assignment. Korst [1992] has shown that, under the assumption that different invocations of one operation may be assigned to different processing elements, the problem of constructing a strictly periodic constrained processor assignment with a given period p can be modeled as the problem of coloring circular arcs that are positioned on a circle with p segments such that overlapping arcs receive different colors. Hence, in that case the problem of finding a processing element assignment with period p that satisfies the computation constraints can be formulated as the problem of coloring a circular-arc graph.

Definition 3.7 (Circular-Arc Graph). A graph (V,E) is said to be a *circular-arc graph* if and only if it can be associated with a circle that is divided into p segments that are numbered clockwise as $1,...,p$, in such a way that each vertex $v \in V$ can be associated with a circular arc $a = [l,r]$ on the circle that stretches clockwise from segment l up to and including segment r, where $l,r \in \{1,\ldots,p\}$, such that $\{v,v'\} \in E$ if and only if the corresponding arcs $[l,r]$ and $[l',r']$ overlap. □

In order to construct a circular-arc graph for a given signal flow graph, we associate each of the invocations of one operation within one period p with one vertex in the circular-arc graph. Since the number of invocations per operation is at most p, the number of vertices in the circular-arc graph is bounded by $p|O|$. Garey et al. [1980] have shown that coloring circular-arcs graphs with a minimum number of colors is NP-hard. Furthermore, they have shown that coloring circular-arcs graphs with a given number k of colors can be solved in $\mathcal{O}(nk!k\log k)$ time, where $n \leq 2|V| \leq 2p|O|$. Thus, for fixed k the problem is solvable in polynomial time. However, this is only practical for small numbers k. For a video signal processor with twelve arithmetic and logic elements we have that $k!k\log k \approx 6 \times 10^9$.

3.7 Precedence Constraints

To analyze the precedence constraints we again resort to the work of Korst [1992] who has shown that the problem of scheduling strictly periodic operations under precedence constraints is solvable in polynomial time. The result is based on the observation that all precedence constraints can be represented as linear constraints on the time assignment, i.e., for all $r = ((o,n),(o',n'),(p,b,b'))$ we have

$$\underline{\omega}(r) \leq \sigma(o') - \sigma(o) \leq \overline{\omega}(r),$$

and for all $s = (o,o',(p,b,b'))$ we have

$$\underline{\omega}(s) \leq \sigma(o') - \sigma(o).$$

These equations are closely related to the so-called Bellman equations; see Lawler [1976]. Based on this analogy we formulate the problem of finding a mapping that satisfies the precedence constraints as a longest path problem.

Theorem 3.8. *The problem of constructing a time assignment that satisfies the precedence constraints is solvable in polynomial time.*

Proof. Let for an arbitrary pair of operations o_i and o_j the longest path from o_i to o_j be given by the sequence of precedences $a_1,\ldots,a_x$ provided that such a path exists. Then this path contains at most $|O|-1$ precedences. For each operation o_k on this path, the longest path from o_i to o_k is necessarily a subsequence of the path $a_1,\ldots,a_x$.

Under the assumption that all completion times initially equal zero, we now replace for all precedences a the value of the completion time of the producing operation o' by $\max\{\sigma(o'), \sigma(o) + \underline{\omega}(a)\}$. As a result, we find the minimum completion time that satisfies the lower bound for the next operation on each path. In addition, we replace for all data precedences a the value of the completion time of the consuming operation o by $\max\{\sigma(o), \sigma(o') - \overline{\omega}(a)\}$. As a result, we find the minimum completion time that satisfies the upper bound for the previous operation on each path. After $|O| - 1$ of these iterations, operation o_j has obtained a minimum completion time provided that such a completion time exists.

If such a completion time does not exist, then there is no longest path from operation o_i to operation o_j but o_j can be reached from o_i. In that case the precedence constraints are infeasible, i.e., there does not exist a time assignment that satisfies the precedence constraints. This can have two reasons. The first reason is that there is a cycle in the signal flow graph for which the sum of the weights $\underline{\omega}(a)$ is positive and there exists some precedence a for which holds $\sigma(o') < \sigma(o) + \underline{\omega}(a)$ after $|O| - 1$ iterations. The second reason is that there is a cycle via the data precedences in the signal flow graph for which the sum of the weights $\overline{\omega}(a)$ is negative and there exists some data precedence a for which holds $\sigma(o) < \sigma(o') - \overline{\omega}(a)$ after $|O| - 1$ iterations. □

We now obtain the following corollary that defines necessary and sufficient conditions for the existence of a time assignment that satisfies the precedence constraints.

Corollary 3.6. *Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\delta, \sigma, \alpha, \lambda)$ be a mapping. Then there exists a time assignment that satisfies the precedence constraints if and only if for all cycles $L \subseteq R \cup S$ it holds that*

$$\sum_{a \in L} \underline{\omega}(a) \leq 0,$$

and for all cycles $L \subseteq R$ it holds that

$$\sum_{a \in L} \overline{\omega}(a) \geq 0.$$

□

If no time assignment exists that satisfies the precedence constraints, we must either decrease the sum of the lower bounds, or increase the sum of the upper bounds for the cycles that are of concern. A possible way to achieve this is to reduce or expand the signal flow graph by removing or adding pass operations in these cycles. However, the problem of determining where to remove or add these pass operations in such a way that the computation, connectivity, and precedence constraints are satisfied is NP-complete in the strong sense, even if the periods form a divisible sequence. We demonstrate this using a polynomial-time reduction from three-satisfiability.

Definition 3.8 (Three-Satisfiability). Let U be a set of variables and let C be a set of clauses such that each clause $c \in C$ is a disjunction of three literals which can be positive, i.e., of the form u, or negative, i.e., of the form $\overline{u}$, where $u \in U$. Find a function $f : U \rightarrow \mathbb{B}$ that satisfies each clause $c \in C$, if one exists. □

Theorem 3.9. *The problem of expanding a signal flow graph and constructing a processing element, channel, and time assignment that satisfy the computation, connectivity, and precedence constraints is NP-hard in the strong sense, even if the periods form a divisible sequence.*

Proof. For a given instance and a given solution we can verify in polynomial time whether the computation, connectivity, and precedence constraints are satisfied. Hence, the problem is in $\mathcal{NP}$. To prove that the problem is NP-hard we use the following polynomial time reduction from three-satisfiability as formulated in Definition 3.8 which is known to be NP-hard in the strong sense [Cook, 1971].

Given an arbitrary instance of three-satisfiability, we construct a corresponding instance of the problem as follows. First we define a set of clause operations O_C as follows. For each clause $c \in C$ we define two arithmetic and logic operations $o_c, o'_c \in O_C$ with period one. In order to constrain each time assignment σ such that $\sigma(o_c) = \sigma(o'_c)$, we define for each clause $c \in C$ two clause no-value precedences $s_c, s'_c \in S_C$. Precedence s_c is directed from o_c to o'_c and precedence s'_c is directed from o'_c to o_c. Both precedences have period one and offsets zero. Furthermore, we define for each clause $c \in C$ two clause data precedences $r_c, r'_c \in R_C$. Again precedence r_c is directed from o_c to o'_c and precedence r'_c is directed from o'_c to o_c. Both precedences have period two, first offset zero, and second offset one plus the size of a silo. We have chosen the offsets such that these data precedences violate the precedence constraints. Therefore we have to expand the signal flow graph with pass operations. Next we define a set of literal operations O_L as follows. For each literal $l \in c$ we define an arithmetic and logic operation $o_{c_l} \in O_L$ with period one. Furthermore, we define for each literal l in each clause $c \in C$ two literal data precedences $r_{c_l}, r'_{c_l} \in R_L$. Again both precedences have period two, first offset zero, and second offset one plus the size of a silo. If $l = u$ for some variable $u \in U$ then r_{c_l} is directed from o_c to o_{c_l} and r'_{c_l} is directed from o_{c_l} to o'_c. Otherwise, r_{c_l} is directed from o_{c_l} to o_c and r'_{c_l} is directed from o'_c to o_{c_l}. Once again note that the corresponding data precedence constraints cannot be satisfied without expanding the signal flow graph with pass operations. The entire construction is graphically represented in Figure 3.1.

Subsequently, we index the clauses $C = \{c_0, ..., c_{m-1}\}$. In order to constrain each time assignment σ such that $\sigma(o_{c_i}) = \sigma(o_{c_j})$ for all $c_i, c_j \in C$, we define m no-value precedences $s_i \in S$ with period one and offsets zero. The precedences are directed from o_i to $o_{(i+1) \bmod m}$ for all $0 \leq i < m$.

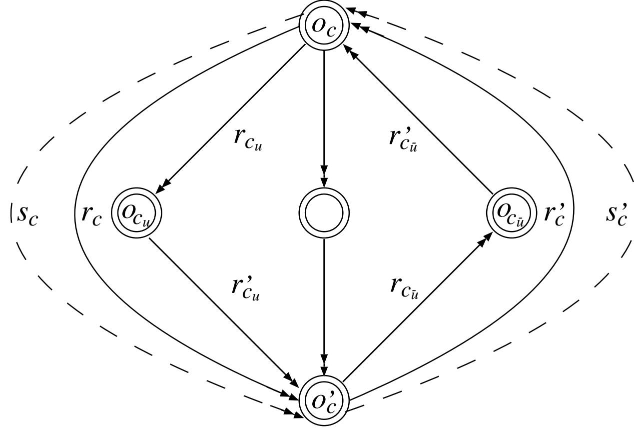


Figure 3.1. Transformation from clause into signal flow graph.

Finally, we split the set of literal operations O_L into $|U|$ sets $O_L^u = \{o_{c_l} \in O_L \mid l = u \in U \vee l = \overline{u} \in U\}$. We index the operations in these sets such that $O_L^u = \{o_{u0},\ldots,o_{u(n_u-1)}\}$. In order to constrain each time assignment σ such that $\sigma(o) = \sigma(o')$ for all $o,o' \in O_L^u$, we define n_u no-value precedences $s_{ui} \in S$ with period one and offsets zero. The precedences are directed from o_{ui} to $o_{u((i+1) \bmod n_u)}$ for all $1 \leq i < n_u$.

We complete the instance with definitions for the set of arrays and the processing element graph. We define the set of arrays $A = \emptyset$ and we define the processing element graph via the corresponding processor graph (V,I). With each $c_i \in C$ we associate a $v_i \in V$ such that $t_{\mathrm{v}}(v_i) = \mathrm{VSP1}$. For each processor v_i we introduce four interconnections $((v_i, j),(v_i, j)) \in I$ for all $1 \leq j \leq 4$ with a communication delay of zero. Note that each video signal processor $v_i \in V$ contains three arithmetic and logic elements and four output elements that connect four processor outputs with four processor inputs. Hence, the computation and connectivity constraints allow four pass operations to be added in each clause.

Suppose the function f is a satisfying truth assignment. Then we can construct a signal flow graph expansion and a mapping that satisfy the computation, connectivity, and precedence constraints as follows. We define for each clause $c \in C$ the operation set $O_c = \{o_{c_l} \mid l \in c\} \cup \{o_c, o'_c\}$ and the precedence set $R_c = \{r_{c_u} \mid u \in c \wedge f(u)\} \cup \{r_{c_{\overline{u}}} \mid \overline{u} \in c \wedge \neg f(u)\} \cup \{r_c, r'_c\}$. Note that $|R_c| \leq |O_c| = 5$. Then we split the precedence set R_c into $R_c^1,\ldots,R_c^5$ by defining $R_c^i = \{((o,n),(o',n'),(p,b,b')) \in R_c \mid o = o_i\}$ for all $o_1,\ldots,o_5 \in O_c$. Note that the set R_c^i is empty if and only if $o_i = o_{c_l}$ for some satisfied literal operation, i.e., f satisfies l. Because the function f is a satisfying truth assignment at least one of the sets R_c^i is empty. For each non-empty sets R_c^i we introduce a pass operation o'_{c_l} with period two and we expand

the signal flow graph accordingly. As a result, each operation o_{c_l} has either a new predecessor o'_{c_l} if the truth assignment f satisfies the literal l, or a new successor o'_{c_l} if the truth assignment f does not satisfy the literal l. Next we define the mapping. To this end we map all operations and precedences in clause c_i on the resources of processor v_i which satisfies the computation and connectivity constraints. This defines the delay, processing element, and channel assignment. For all clause operations $o_c, o'_c \in O_C$ we choose $\sigma(o_c) = \sigma(o'_c) = 0$. For all literal operations $o_{c_l} \in O_L$ we choose $\sigma(o_{c_l})$ equal to one plus the silo size if the truth assignment f does not satisfy literal l, and we choose $\sigma(o_{c_l})$ twice as large if the truth assignment f satisfies literal l. For each pass operation o resulting from set R^i_c we choose $\sigma(o)$ and $\sigma(o_i)$ equal to one plus the silo size. It is straightforwardly checked that this time assignment satisfies all precedence constraints.

Suppose we have a signal flow graph expansion and a mapping that satisfy the computation and connectivity constraints. Then we can construct a satisfying truth assignment f as follows. Since the precedence constraints for the precedences r_c, r'_c, s_c, s'_c, r_{c_l}, and r'_{c_l} are satisfied, each operation o_{c_l} must have at least one corresponding pass operation o'_{c_l}. We construct a truth assignment $f : U \to \mathbb{B}$ as follows. If operation o'_{c_l} is a predecessor of operation o_{c_l} then we define $f(u) = \text{TRUE}$, otherwise we define $f(u) = \text{FALSE}$. The satisfaction of the precedence constraints for no-value precedences $s_{u0}, \ldots, s_{u(n_u-1)}$ for all $u \in U$ ensures that the function f is well-defined. The computation and connectivity constraints ensure that we introduce at most four pass operations in each clause, which guarantees that each clause has a satisfying literal.

The magnitude of the largest integer in the constructed instance of the problem is polynomial in the size of the instance, which means that problem is NP-complete in the strong sense. □

3.8 Connectivity Constraints

To satisfy the connectivity constraints we must assign the operations that are related by a data precedence to processing elements that are related by a channel. In order to implement the communication from one processor to another processor, we must expand a signal flow graph with pass operations and assign these operations to output elements. The problem of expanding a signal flow graph and constructing a processing element and time assignment that satisfy the connectivity, computation, and communication constraints is NP-hard in the strong sense, even if the periods form a divisible sequence. We demonstrate this using a polynomial-time reduction from three-satisfiability in a way that is similar to the proof of Karp [1975] who shows that disjoint connecting paths is NP-complete. We owe the proof to De Fluiter [1993].

Theorem 3.10. *The problem of expanding a signal flow graph and constructing a time, processing element, and channel assignment that satisfy the connectivity, computation, and communication constraints is NP-hard in the strong sense, even if the periods form a divisible sequence.*

Proof. For a given instance and a given solution we can verify in polynomial time whether the connectivity, computation, and communication constraints are satisfied. Hence, the problem is in $\mathcal{NP}$. To prove that problem is NP-hard we use the following polynomial time reduction from three-satisfiability which is known to be NP-hard [Cook, 1971].

Given an instance of three-satisfiability as formulated in Definition 3.8, we construct an instance of the mapping problem as follows. For each clause $c \in C$ we construct a processor v_c and a processor v'_c, both of type VSP1, a processor s_c of type INPUT, and processors t_c and t'_c of type OUTPUT. Furthermore, we construct four connections between the processors, i.e., from s_c to v_c, from v_c to t'_c, from v'_c to t_c, and from v'_c to v_c. For each literal l, we denote the set of clauses that contain

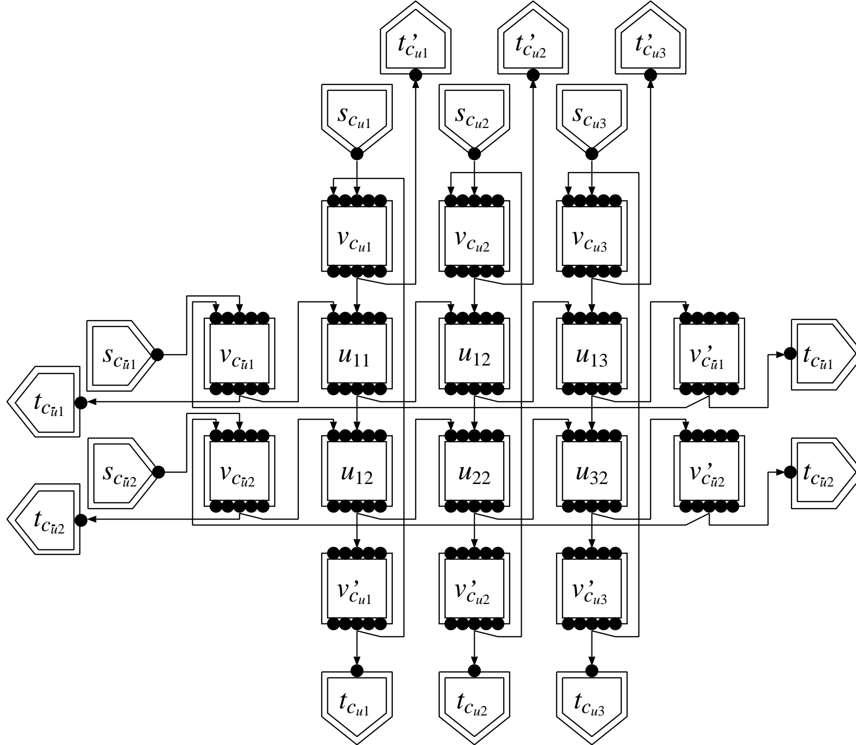


Figure 3.2. An example of the processor subnetwork N_u with $n_u = 3$ and $n_{\overline{u}} = 2$.

l by $C_l = \{c_{l1}, \ldots, c_{ln_l}\}$. Note that different sets C_l do not have to be disjoint. For each $u \in U$, for each $1 \leq i \leq n_u$, and for each $1 \leq j \leq n_{\overline{u}}$, we construct a processor

u_{ij} of type VSP1 and we construct six connections, i.e., from $v_{c_{ui}}$ to u_{i1}, from $v_{c_{\bar{u}j}}$ to u_{1j}, from $u_{in_{\bar{u}}}$ to $v'_{c_{ui}}$, from $u_{n_u j}$ to $v'_{c_{\bar{u}j}}$, from u_{ij} to $u_{i(j+1)}$, and from u_{ij} to $u_{(i+1)j}$, in such a way that each input port has at most one incoming connection. The number of input ports is sufficient, since each processor v_c has at most three incoming connections, and the processors t_c and t'_c have exactly one incoming connection. In Figure 3.2 a part of the processor network, called N_u, is drawn. In this part, all processors and interconnections that have to do with u are drawn. Note that the processors u_{ij} are only used in N_u, but all other processors are present in two other subnetworks $N_{u'}$ and $N_{u''}$.

We now construct signal flow graph (O,R). It consists of $|C|$ subparts (O_i,R_i). For each $1 \leq i \leq |C|$, we construct an operation $s_i \in O_i$ of type IO, two operations $t_i, t'_i \in O_i$ of type OO, and four operations $v_{i1}, v_{i2}, v_{i3}, v_{i4} \in O_i$ of type ALO. All operations have period one. Furthermore for each $1 \leq i \leq |C|$ we construct seven data precedences with periods of one and offsets of zero in the set R_i, i.e., from s_i to v_{i1}, from v_{i1} to t'_i, from v_{i1} to v_{i2}, from v_{i2} to v_{i3}, from v_{i3} to v_{i4}, from v_{i4} to v_{i1}, and from v_{i4} to t_i. An example of a part (O_i,R_i) is given in the left-hand side of Figure 3.3. The set of no-value precedences and the set of arrays are empty.

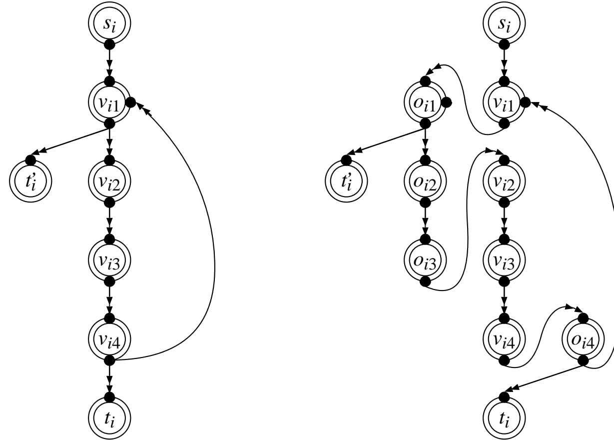


Figure 3.3. The picture of the left shows part (O_i,R_i) of the constructed signal flow graph. The picture on the right shows the part (O'_i,R'_i) of the constructed signal flow graph after transformation.

Since all precedences have period one, the paths that must be constructed have to be disjoint. Note that the number of input processors is equal to the number of input operations and each input operation has period one, so each input operation must be assigned to an individual input processor. The same holds for output operations. Furthermore note that for each i, the operations v_{i1}, v_{i2}, v_{i3}, and v_{i4} cannot all be assigned to the same processor since each processor contains only three

arithmetic and logic elements. This means that if we have a processing element assignment α that maps operation s_i onto processor s_c for some $c \in C$, then it is not possible that all v_{ij} are assigned to v_c. This means that the output element of this processor must be used for one of the following data streams: from s_i to v_{i1}, from v_{i1} to v_{i2} and t'_i, from v_{i2} to v_{i3}, or from v_{i3} to v_{i4}. Since some output operation must be assigned to t_c, the only possibility is that the data stream from v_{i1} to v_{i2} and t'_i uses the output element of v_c. So operation v_{ij} is assigned to processor v_c, the operations v_{ij} for $j = 2,3,4$ are assigned to other processors, and operation t'_i is assigned to processor t'_c. Since we have a data precedence from v_{i4} to v_{i1} there must be a path from the processor of operation v_{i4} to v_c. Such a path can only exist via v'_c, which means that operation t_i is assigned to processor t_c since v'_c has only one output element that can be used. Hence, to satisfy the connectivity constraints it must hold that if operation s_i is assigned to processor s_c, then operation t_i is assigned to processor t_c and operation t'_i is assigned to processor t'_c.

Suppose we have a satisfying truth assignment f then we construct a signal flow graph expansion, a processing element assignment, and a channel assignment as follows. Assign each s_i to an s_c such that there is one input operation assigned to each input processor. Furthermore, if s_i is assigned to s_c then t_i is assigned to t_c, t'_i is assigned to t'_c, v_{i1} is assigned to v_c, and v_{i2}, v_{i3}, and v_{i4} are assigned to v'_c. For each $c \in C$, choose one literal in c that is true. If this literal is a variable $u \in U$ then construct the 'vertical' path from that s_i to t_i in N_u such that s_i is assigned to s_c, i.e., create pass operations $o_{i1}, \ldots, o_{i(n_{\overline{u}}+2)} \in O'_i$ and map them onto the output elements of the processors as follows. Operation o_{i1} goes to $v_c = v_{c_k}$ for some k, operation $o_{i(n_{\overline{u}}+2)}$ goes to v'_c, and the other pass operations go to $u_{k(l-1)}$ for all $2 \leq l \leq n_{\overline{u}} + 1$. Furthermore we construct a set R'_i of data precedences with periods of one and offsets of zero from o_{ik} to $o_{i(k+1)}$ for all $1 \leq k \leq n_{\overline{u}}$, from s_i to v_{i1}, from v_{i1} to o_{i1}, from o_{i1} to t'_i, from $o_{i(n_{\overline{u}}+1)}$ to v_{i2}, from v_{i2} to v_{i3}, from v_{i3} to v_{i4}, from v_{i4} to $o_{i(n_{\overline{u}}+2)}$, from $o_{i(n_{\overline{u}}+2)}$ to $t_i, 0$, and from $o_{i(n_{\overline{u}}+2)}$ to v_{i1}. If the literal is the complement of a variable u, then construct the 'horizontal' path from that s_i to t_i in N_u such that s_i is assigned to s_c, in a similar way. An example of (O'_i, R'_i) is given in the right-hand side of Figure 3.3. The paths can always be constructed in this way, since if $f(u)$ holds for $u \in U$, then there are only 'vertical' paths in N_u constructed, and if $f(u)$ not holds, then there are only 'horizontal' paths constructed in N_u. Hence we have constructed a signal flow graph expansion, a processing element assignment, and a channel assignment that satisfy the connectivity constraints and, by Corollaries 3.2 and 3.3, for which there exists a time assignment that satisfies the computation and communication constraints.

Suppose there are a signal flow graph expansion (O', R'), a time assignment σ, a processing element assignment α, and a channel assignment λ, then we construct a satisfying truth assignment f as follows. For each $u \in U$, if there is a $c \in C$ with

$u \in c$ such that operation s_i is mapped onto processor s_c, operation t_i is mapped onto processor t_c, and there is a path from s_i to t_i via operations that are mapped to processors in N_u, then $f(u)$ holds. If there is a $c \in C$ with $\overline{u} \in c$ such that operation s_i is mapped onto processor s_c, operation t_i is mapped onto processor t_c, and there is a path from s_i to t_i via operations that are mapped to processors in N_u, then $f(u)$ does not hold. Otherwise, $f(u)$ can be chosen arbitrarily. In each part N_u, with $u \in U$, of the processor network there can either be a 'vertical' path from s_i to t_i for some $c \in C_u$, or there can be a 'horizontal' path from s_i to t_i for some $c' \in C_{\overline{u}}$, which means that f is defined properly. Furthermore, there must be a path from each s_i to t_i which means that there must be one literal that is true in each clause. This means that we have constructed a feasible truth assignment.

We have proved that the problem of finding a time, processing element, and channel assignment that satisfy the connectivity, computation, and communication constraints is NP-complete if the periods of the operations and precedences are equal to one. Since these periods are the only integer constants the problem is NP-complete in the strong sense. $\square$

3.9 Summary

The type, array, connectivity, and precedence constraints are solvable in polynomial time. The computation, communication, storage, and periodicity constraints are not solvable in polynomial time. The computation and communication constraints are solvable in polynomial time if the periods form a divisible sequence using bin packing techniques with divisible item sizes. For fixed size memories the storage constraints are solvable in polynomial time using exhaustive search, but this is only practical for small memory sizes. Similarly for fixed size programs the periodicity constraints are solvable in polynomial time using circular-arc graph coloring, but again this is only practical for small program sizes. The combination of the computation and communication constraints with divisible periods and the connectivity constraints is not solvable in polynomial time. The combination of the computation constraints with divisible periods, the connectivity constraints, and the precedence constraints is also not solvable in polynomial time.

4

Problem Decomposition

In the previous chapter we have proved that the mapping problem is NP-hard which means that no algorithm solves the problem in polynomial time unless $\mathcal{P} = \mathcal{NP}$. We propose a decomposition of the mapping problem into three subproblems, which we call the delay management problem, the partitioning problem, and the scheduling problem, because we believe that the mapping problem is too complex to handle in its entirety. The goal of this approach is to introduce a separation of concerns in the mapping process in such a way that we can handle the combination of the subproblems more effectively than the complete mapping problem.

To present the decomposition strategy we generalize the model presented in Chapter 2 by considering a set of mappings rather than a single mapping. The purpose of the generalization is to formalize the decomposition strategy which stepwisely reduces the set of mappings by removing mappings that are inconsistent with the decisions made so far. The generalization enables us to describe the current search state as a set of mappings. Furthermore, the generalization enables us to formulate necessary conditions to determine whether a mapping is inconsistent. These conditions are relaxations of the mapping constraints such that they do not remove feasible solutions from the search space. Finally, the generalization provides a means to determine in which order the mapping decisions have to be taken resulting in the decomposition of the mapping problem into three subproblems.

The outline of this chapter is as follows. In Section 4.1 we generalize a single mapping to a set of mappings in order to reason on the problem decomposition. In Section 4.2 we introduce relaxations of the mapping constraints, thereby introducing necessary conditions for feasibility that reduce the search space. In Section 4.3 we present the decomposition strategy. In Sections 4.4, 4.5, and 4.6 we formulate the delay management problem, the partitioning problem, and the scheduling problem, respectively. In Section 4.7 we present the results of the constraint relaxations on the set of industrially relevant problem instances. Finally, in Section 4.8 we summarize the contents of this chapter.

4.1 Mapping Sets

In order to formulate necessary conditions for feasibility which reduce the search space, we generalize a single mapping to a set of mappings. We formally define such a set, which contains different mappings of a signal flow graph onto a processing element network, as follows.

Definition 4.1 (Mapping Set). Let (O,R) be a signal flow graph and let (P,C) be a processing element network. Then the tuple $(\Delta,\Sigma,\mathrm{A},\Lambda)$ defines a *mapping set*, where

- $\Delta \subseteq O \to \mathbb{N}$ denotes a *delay assignment set*,
- $\Sigma \subseteq O \to \mathbb{Z}$ denotes a *time assignment set*,
- $\mathrm{A} \subseteq O \to P$ denotes a *processing element assignment set*, and
- $\Lambda \subseteq R \to C$ denotes a *channel assignment set*. □

The notation for a set of delay assignments $\Delta \subseteq O \to \mathbb{N}$ is a generalization of the notation for a single delay assignment $\delta \in O \to \mathbb{N}$. The latter is an alternative notation for $\delta : O \to \mathbb{N}$ in which a delay assignment is represented as a set of pairs $\{(o,\delta(o)) \mid o \in O \wedge \delta(o) \in \mathbb{N}\}$. For each operation o we denote the domain values of the delay assignment, i.e., the set of candidate delays, by $\Delta(o) = \{\delta(o) \mid \delta \in \Delta\}$, which is a compact representation of the delay assignment set. We adopt the same notation for the candidate time, processing element, and channel sets. For each set of operations O we denote the set of candidate delay sets by $\Delta(O) = \{\Delta(o) \mid o \in O\}$. Again we adopt the same notation for the sets of candidate time, processing element, and channel sets. The largest mapping set for a given instance of the mapping problem is the mapping set $(\Delta,\Sigma,\mathrm{A},\Lambda)$ with $\Delta(o) = \mathbb{N}$, $\Sigma(o) = \mathbb{Z}$, $\mathrm{A}(o) = P$, and $\Lambda(r) = C$ for all $o \in O$ and for all $r \in R$. An important issue in the solution strategy is the reduction of the search space by removing mappings from the mapping set that are inconsistent with the decisions made so far. This is illustrated in Figure 4.1 that shows how a reduction of the time assignment set

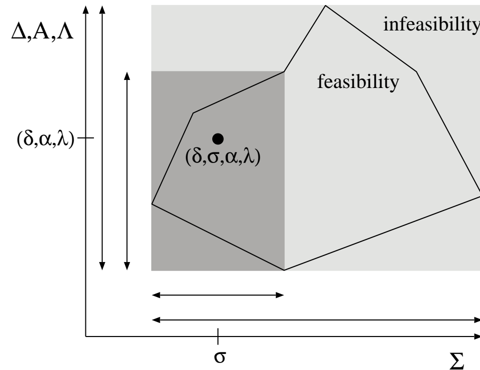


Figure 4.1. Reduction of the mapping set by removing inconsistent mappings.

can reduce the mapping set from the light-shaded area to the dark-shaded area. In the next section we introduce relaxations of the mapping constraints to prevent the need for exhaustive checking of the original mapping constraints for each mapping in the mapping set. These relaxations can be checked more efficiently, although they do not exclude all infeasible solutions from the search space.

4.2 Constraint Relaxations

The purpose of formulating relaxations of the mapping constraint is to define conditions that are necessary to satisfy the mapping constraints. Mappings that do not satisfy the relaxations can be eliminated from the search space without affecting the solution space. The constraint relaxations are the result of the complexity analysis of the previous chapter. This analysis has identified easy problems or easy special cases of hard problems which have lead to the necessary conditions formulated in Corollaries 3.2 up to 3.6. These corollaries are the motivation for the relaxations of the precedence, computation, and communication constraints.

4.2.1 Precedence Constraints

The relaxation of the precedence constraints is motivated by Corollary 3.6 which defines necessary conditions for the bounds that constrain the completion times of two adjacent operations. We generalize these bounds for a single mapping to bounds for a mapping set in order to constrain the set of completion times of two adjacent operations. To this end we introduce a set of candidate silo departure times $\mathrm{X}(r) = \{\chi(r) \mid \sigma \in \Sigma \wedge \alpha \in \mathrm{A} \wedge \lambda \in \Lambda\}$, and a set of candidate silo arrival times $\Psi(r) = \{\psi(r) \mid \delta \in \Delta \wedge \sigma \in \Sigma \wedge \alpha \in \mathrm{A} \wedge \lambda \in \Lambda\}$, for a given data precedence r. For a given no-value precedence s we introduce a set of candidate array departure times $\mathrm{X}(s) = \{\chi(s) \mid \sigma \in \Sigma\}$, and a set of candidate array arrival times $\Psi(s) =$

$\{\psi(s) \mid \sigma \in \Sigma\}$. Next we generalize the arrival delay $\omega(a)$ to a set of arrival delays $\Omega(a) = \{\omega(a) \mid \delta \in \Delta \wedge \alpha \in \mathrm{A} \wedge \lambda \in \Lambda\}$, and the departure delay $\omega'(a)$ into a set of departure delays $\Omega'(a) = \{\omega'(a) \mid \alpha \in \mathrm{A} \wedge \lambda \in \Lambda\}$, for all data precedences $a = ((o,n),(o',n'),(p,b,b'))$ and no-value precedences $a = (o,o',(p,b,b'))$. Note that the arrival and departure delays are independent of the time assignments. From the set of arrival and departure delays we derive a lower bound $\min\underline{\Omega}(a)$ and an upper bound $\min\overline{\Omega}(a)$ by using minimum arrival and departure delays, i.e.,

$$\min\underline{\Omega}(a) = \min\Omega(a) + \min\Omega'(a) + \min W(a) \text{ and}$$

$$\min\overline{\Omega}(a) = \min\Omega(a) + \min\Omega'(a) + \max W(a).$$

Similarly we derive a lower bound $\max\underline{\Omega}(a)$ and an upper bound $\max\overline{\Omega}(a)$ by using maximum arrival and departure delays, i.e.,

$$\max\underline{\Omega}(a) = \max\Omega(a) + \max\Omega'(a) + \min W(a) \text{ and}$$

$$\max\overline{\Omega}(a) = \max\Omega(a) + \max\Omega'(a) + \max W(a).$$

To satisfy the precedence constraints it must hold that the difference between the minimum completion times of two adjacent operations is larger than or equal to the smallest lower bound and that it is smaller than or equal to the smallest upper bound. In addition it must hold that the difference between the maximum completion times of two adjacent operations is larger than or equal to the largest lower bound and that it is smaller than or equal to the largest upper bound.

Definition 4.2 (Precedence Constraint Relaxation). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set. Then the *data precedence constraint relaxation* specifies that for all $r = ((o,n),(o',n'),(p,b,b')) \in R$ it holds that

$$\min\underline{\Omega}(r) \leq \min\Sigma(o') - \min\Sigma(o) \leq \min\overline{\Omega}(r) \text{ and}$$

$$\max\underline{\Omega}(r) \leq \max\Sigma(o') - \max\Sigma(o) \leq \max\overline{\Omega}(r),$$

and the *no-value precedence constraint relaxation* specifies that for all $s = (o,o',(p,b,b')) \in S$ it holds that

$$\min\underline{\Omega}(s) \leq \min\Sigma(o') - \min\Sigma(o) \text{ and}$$

$$\max\underline{\Omega}(s) \leq \max\Sigma(o') - \max\Sigma(o). \qquad \square$$

The relaxation lies in the fact that the lower and upper bounds may result from the individual assignments of different mappings whereas the original precedence constraints apply to a single mapping. The relaxation is an exact approximation if the sets $\Omega(a)$ and $\Omega'(a)$ of arrival and departure delays are singleton sets. To formulate the generalization of Corollary 3.6 we define $\underline{\Omega}(a) = \max\underline{\Omega}(a)$ as the largest lower bound and $\underline{\Omega}(a) = \min\overline{\Omega}(a)$ as the smallest upper bound.

Corollary 4.1. *Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set. Then there exists a time assignment set that satisfies the precedence constraint relaxation if and only if for all cycles $L \subseteq R \cup S$ it holds that*

$$\sum_{a \in L} \underline{\Omega}(a) \leq 0,$$

and for all cycles $L \subseteq R$ it holds that

$$\sum_{a \in L} \overline{\Omega}(a) \geq 0.$$

□

4.2.2 Computation Constraints

We derive the relaxation of the computation constraints from Corollaries 3.2 and 3.4 which define necessary conditions to satisfy the computation constraints. The relaxation bounds the sum of the frequencies of a set of periodic operations by the capacity of the processing elements on which the operations are executed. The relaxation is an exact approximation if the periods of the operations form a divisible sequence.

Definition 4.3 (Computation Constraint Relaxation). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set. Then the *computation constraint relaxation* specifies that for all $\mathrm{A}(o),\mathrm{A}(o') \in \mathrm{A}(O)$ it holds that

$$\mathrm{A}(o) \cap \mathrm{A}(o') \neq \emptyset \Rightarrow (\mathrm{A}(o) \subseteq \mathrm{A}(o') \vee \mathrm{A}(o') \subseteq \mathrm{A}(o)),$$

and that for all $\mathrm{A}(o) \in \mathrm{A}(O)$ it holds that

$$R_{\mathrm{A}}(O_{\mathrm{A}(o)}) \leq C_{\mathrm{A}}(\mathrm{A}(o)).$$

The left-hand side $R_{\mathrm{A}}(O_{\mathrm{A}(o)})$ defines the *average computation requirement*

$$R_{\mathrm{A}}(O_{\mathrm{A}(o)}) = \max\{ \sum_{\substack{o' \in O_{\mathrm{A}(o)} \\ \{t(o'),T(\mathrm{A}(o))\} \neq \{\mathrm{RO},\{\mathrm{ME2}\}\}}} \frac{1}{p(o')}, \sum_{\substack{o' \in O_{\mathrm{A}(o)} \\ \{t(o'),T(\mathrm{A}(o))\} \neq \{\mathrm{WO},\{\mathrm{ME2}\}\}}} \frac{1}{p(o')} \}$$

of operation set $O_{\mathrm{A}(o)} = \{o' \in O \mid \mathrm{A}(o') \subseteq \mathrm{A}(o)\}$ which contains the operations that are mapped onto the processing element set $\mathrm{A}(o)$. The set $T(\mathrm{A}(o)) = \{t(\alpha(o)) \mid \alpha \in \mathrm{A}\}$ denotes the set of processing element types in $\mathrm{A}(o)$. The right-hand side $C_{\mathrm{A}}(\mathrm{A}(o))$ defines the *computation capacity*

$$C_{\mathrm{A}}(\mathrm{A}(o)) = |\mathrm{A}(o)|$$

of processing element set $\mathrm{A}(o)$. □

4.2.3 Communication Constraints

Similarly we derive the relaxation of the communication constraints from Corollaries 3.3 and 3.5 which define necessary conditions to satisfy the communication constraints. The relaxation bounds the number of samples that can be consumed by the input terminals of the processing elements on which the consuming operations are executed. The relaxation is an exact approximation if the periods of the precedences form a divisible sequence.

Definition 4.4 (Communication Constraint Relaxation). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set. Then the *communication constraint relaxation* specifies that for all $\Lambda(r),\Lambda(r') \in \Lambda(R)$ it holds that

$$I(\Lambda(r)) \cap I(\Lambda(r')) \neq \emptyset \Rightarrow (I(\Lambda(r)) \subseteq I(\Lambda(r')) \vee I(\Lambda(r')) \subseteq I(\Lambda(r))),$$

and that for all $\Lambda(r) \in \Lambda(R)$ it holds that

$$R_\Lambda(R_{\Lambda(r)}) \leq C_\Lambda(\Lambda(r)).$$

The left-hand side $R_\Lambda(R_{\Lambda(r)})$ defines the *average communication requirement*

$$R_\Lambda(R_{\Lambda(r)}) = \sum_{((o,n),(o',n'),(p,b,b')) \in R_{\Lambda(r)}} \frac{|\{i \in \mathbb{N}_{q(o)} \mid b \equiv i \pmod{\frac{p}{p(o)}}\}}{p(o)q(o)}$$

of data precedence set $R_{\Lambda(r)} = \{r' \in R \mid I(\Lambda(r')) \subseteq I(\Lambda(r))\}$ which contains the data precedences that are mapped onto the set $I(\Lambda(r)) = \{(p',m') \mid ((p,m),(p',m')) \in \Lambda(r)\}$ of input terminals. The right-hand side $C_\Lambda(\Lambda(r))$ defines the *communication capacity*

$$C_\Lambda(\Lambda(r)) = |I(\Lambda(r))|.$$

of channel set $\Lambda(r)$. □

4.2.4 Storage Constraints

Next we formulate a necessary condition to satisfy the storage constraints. To this end we approximate the number of memory locations that are required for the delay assignment values with a lower bound because the exact number of required locations depends on the assignment of the individual operations to the individual processing elements.

Definition 4.5 (Storage Constraint Relaxation). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set. Then the *storage constraint relaxation* specifies that for all $\mathrm{A}(o),\mathrm{A}(o') \in \mathrm{A}(O)$ it holds that

$$\mathrm{A}(o) \cap \mathrm{A}(o') \neq \emptyset \Rightarrow (\mathrm{A}(o) \subseteq \mathrm{A}(o') \vee \mathrm{A}(o') \subseteq \mathrm{A}(o)),$$

and that for all $\mathrm{A}(o) \in \mathrm{A}(O)$ it holds that

$$R_\Delta(O_{\mathrm{A}(o)}) \leq C_\Delta(\mathrm{A}(o)).$$

The left-hand side $R_\Delta(O_{\mathrm{A}(o)})$ defines the *storage requirement*

$$R_\Delta(O_{\mathrm{A}(o)}) = \sum_{a \in A_{\mathrm{A}(o)}} s(a) + \sum_{o' \in O_{\mathrm{A}(o)}} \min \Delta(o')$$

of operation set $O_{\mathrm{A}(o)} = \{o' \in O \mid \mathrm{A}(o') \subseteq \mathrm{A}(o)\}$ which contains the operations that are mapped onto the processing element set $\mathrm{A}(o)$. The right-hand side $C_\Delta(\mathrm{A}(o))$ defines the *storage capacity*

$$C_\Delta(\mathrm{A}(o)) = \sum_{p \in \mathrm{A}(o)} s(p)$$

of the processing element set $\mathrm{A}(o)$. □

4.2.5 Periodicity Constraints

Subsequently we formulate a necessary condition to satisfy the periodicity constraints. To this end we approximate the period that is required to execute a set of strictly periodic operations and precedences by the maximum period in the given set. Note that this approximation is exact if the periods of the operations and precedences form a divisible sequence. The maximum period may not exceed the maximum program capacity that is available in the set of processing elements on which the operations are executed. Formally the relaxation of the periodicity constraints is defined as follows.

Definition 4.6 (Periodicity Constraint Relaxation). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping. Then the *periodicity constraint relaxation* specifies that for all $\mathrm{A}(o),\mathrm{A}(o') \in \mathrm{A}(O)$ it holds that

$$\mathrm{A}(o) \cap \mathrm{A}(o') \neq \emptyset \Rightarrow (\mathrm{A}(o) \subseteq \mathrm{A}(o') \vee \mathrm{A}(o') \subseteq \mathrm{A}(o)),$$

and that for all $\mathrm{A}(o) \in \mathrm{A}(O)$ it holds that

$$R_{\mathrm{A},\Lambda}(O_{\mathrm{A}(o)}) \leq C_{\mathrm{A},\Lambda}(\mathrm{A}(o)).$$

The left-hand side $R_{\mathrm{A},\Lambda}(O_{\mathrm{A}(o)})$ defines the *periodicity requirement*

$$R_{\mathrm{A},\Lambda}(O_{\mathrm{A}(o)}) = \max\{\max\{p(o') \mid o' \in O_{\mathrm{A}(o)}\}, \max\{p(r) \mid r \in R \wedge \alpha(o') \in \mathrm{A}(o)\}\}$$

of operation set $O_{\mathrm{A}(o)} = \{o' \in O \mid \mathrm{A}(o') \subseteq \mathrm{A}(o)\}$ which contains the operations that are mapped onto the processing element set $\mathrm{A}(o)$. The right-hand side $C_{\mathrm{A},\Lambda}(\mathrm{A}(o))$ defines the *periodicity capacity*

$$C_{\mathrm{A},\Lambda}(\mathrm{A}(o)) = \max\{p(m(\alpha(o))) \mid \alpha \in \mathrm{A}\}$$

of processing element element set $\mathrm{A}(o)$. □

4.2.6 Array Constraints

A necessary condition to satisfy the array constraints is that two sets of candidate processing elements that correspond with two operations of the same array are not disjoint, because in that case none of the candidate processing element assignments maps both operations to the same processing element. For this reason we introduce the following relaxation of the array constraints.

Definition 4.7 (Array Constraint Relaxation). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set. Then the *array constraint relaxation* specifies that for all $o,o' \in O$ it holds that if $a(o) = a(o')$ then

$$\mathrm{A}(o) \cap \mathrm{A}(o') \neq \emptyset. \qquad \square$$

4.2.7 Connectivity Constraints

A necessary condition to apply the precedence constraint relaxation is that the set of candidate silo arrival times $\Psi(r)$ is not empty. To this end there has to be at least one channel assignment $\lambda \in \Lambda$ for which it holds that $\lambda(r) \in C$ for all data precedences r. For this reason we introduce the following relaxation of the connectivity constraints.

Definition 4.8 (Connectivity Constraint Relaxation). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set. Then the *connectivity constraint relaxation* specifies that for all $r = ((o,n),(o',n'),(p,b,b')) \in R$ there exists a set of output terminals $M \subseteq \bigcup_{\alpha(o)\in\mathrm{A}(o)} O_{\mathrm{p}}(t(\alpha(o)))$ such that it holds that

$$O(\Lambda(r)) = (\mathrm{A}(o) \times M),$$

or a set of input terminals $M' \subseteq \bigcup_{\alpha(o')\in\mathrm{A}(o')} I_{\mathrm{p}}(t(\alpha(o')))$ such that it holds that

$$I(\Lambda(r)) = (\mathrm{A}(o') \times M'),$$

where $O(\Lambda(r)) = \{(p,m) \mid ((p,m),(p',m')) \in \Lambda(r)\}$ is a set of output terminals and $I(\Lambda(r)) = \{(p',m') \mid ((p,m),(p',m')) \in \Lambda(r)\}$ is a set of input terminals. $\square$

4.2.8 Type Constraints

Furthermore we have to decide which operations are executed on dual ported memory elements to determine which computation constraint applies to which operations. To formalize this requirement we introduce the following relaxation of the type constraints.

Definition 4.9 (Type Constraint Relaxation). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set.

Then the *processing element type constraint relaxation* specifies that for all $o \in O$ it holds that

$$T(\mathrm{A}(o)) \in \begin{cases} \mathcal{P}(\{\textsc{ale}\}) & \text{if } t(o) \in \{\textsc{alo}\} \\ \mathcal{P}(\{\textsc{ale},\textsc{be}\}) & \text{if } t(o) \in \{\textsc{co},\textsc{so}\} \\ \mathcal{P}(\{\textsc{me1}\}) \cup \mathcal{P}(\{\textsc{me2}\}) & \text{if } t(o) \in \{\textsc{ro},\textsc{wo}\} \\ \mathcal{P}(\{\textsc{ale},\textsc{be},\textsc{oe}\}) & \text{if } t(o) \in \{\textsc{po}\} \wedge \max\Delta(o) = 0 \\ \mathcal{P}(\{\textsc{me2}\}) & \text{if } t(o) \in \{\textsc{po}\} \wedge \min\Delta(o) > 0 \\ \mathcal{P}(\{\textsc{in}\}) & \text{if } t(o) \in \{\textsc{io}\} \\ \mathcal{P}(\{\textsc{out}\}) & \text{if } t(o) \in \{\textsc{oo}\}, \end{cases}$$

and the *terminal type constraint relaxation* specifies that for all $r = ((o,n),(o',n'),(p,b,b')) \in R$ with $\Lambda(r) = (\mathrm{A}(o) \times M) \times (\mathrm{A}(o') \times M')$ it holds that $T'(T(\mathrm{A}(o)),M) = \{t_\mathrm{o}(t(o),n)\}$ and

$$T'(T(\mathrm{A}(o')),M') \in \begin{cases} \mathcal{P}(\{\textsc{ci}\}) & \text{if } t_\mathrm{o}(t(o'),n') \in \{\textsc{ci}\} \\ \mathcal{P}(\{\textsc{di}\}) & \text{if } t_\mathrm{o}(t(o'),n') \in \{\textsc{di}\} \\ \mathcal{P}(\{\textsc{ri},\textsc{ai}\}) & \text{if } t_\mathrm{o}(t(o'),n') \in \{\textsc{ri}\} \\ \mathcal{P}(\{\textsc{wi},\textsc{ai}\}) & \text{if } t_\mathrm{o}(t(o'),n') \in \{\textsc{wi}\}, \end{cases}$$

where $T'(T(\mathrm{A}(o)),M) = \{t_\mathrm{p}(t,m) \mid t \in T(\mathrm{A}(o)) \wedge m \in M\}$. □

4.3 Decomposition Strategy

The largest mapping set for a given instance of the mapping problem is the mapping set $(\Delta,\Sigma,\mathrm{A},\Lambda)$ with $\Delta(o) = \mathbb{N}$, $\Sigma(o) = \mathbb{Z}$, $\mathrm{A}(o) = P$, and $\Lambda(r) = C$ for all $o \in O$ and for all $r \in R$. We reduce this mapping set by removing the delays from $\Delta(o)$, the processing elements from $\mathrm{A}(o)$, and the communication channels from $\Lambda(r)$ that violate the relaxation of the type constraints. If the resulting mapping set contains an empty assignment set then the given instance of the mapping problem is clearly infeasible. Otherwise it satisfies not only the relaxation of the type constraints but also the relaxations of the array and connectivity constraints. Furthermore, it holds that the given instance of the mapping problem is infeasible if the given signal flow graph cannot be reduced and the mapping set violates the relaxations of the computation, communication, storage, or periodicity constraints. It also holds that the given instance of the mapping problem is infeasible if the given signal flow graph cannot be reduced and the mapping set violates a lower bound of one of the relaxations of the precedence constraints, i.e., for some cycle $L \subseteq R \cup S$ it holds that $\sum_{a\in L} \max \underline{\Omega}(a) > 0$. In order to check whether some maximum lower bound violates this condition, the maximum completion times $\max\Sigma(o)$ can be computed with the longest path algorithm that has been described in the proof of Theorem 3.8. To this end we substitute the lower bound $\max\underline{\Omega}(a)$ and the upper bound $\max\overline{\Omega}(a)$ for the lower bound $\underline{\omega}(a)$ and the upper bound $\overline{\omega}(a)$, respectively.

The same algorithm computes minimum completion times $\min \Sigma(o)$ if we substitute the lower bound $\min \underline{\Omega}(a)$ and the upper bound $\min \overline{\Omega}(a)$ for the lower bound $\underline{\omega}(a)$ and the upper bound $\overline{\omega}(a)$, respectively. If the mapping set violates an upper bound of one of the relaxations of the precedence constraints, i.e., for some cycle $L \subseteq R$ it holds that $\sum_{a \in L} \min \overline{\Omega}(a) < 0$, then we have to increase the sum of the upper bounds in the cycle. This might be done by expanding the signal flow graph with pass operations in order to increase the number of precedences in the cycle, and hence, the number of terms in the sum. Alternatively, we might increase the upper bounds $\min \overline{\Omega}(r)$ and $\max \overline{\Omega}(r)$ for one or more precedences in the cycle. The minimum upper bound is equal to the maximum silo storage time plus $\min\{\omega(r) \mid \delta \in \Delta \wedge \alpha \in \mathrm{A} \wedge \lambda \in \Lambda\}$ plus $\min\{\omega'(r) \mid \alpha \in \mathrm{A} \wedge \lambda \in \Lambda\}$, where the arrival delay $\omega(r) = \psi(r) - \sigma(o)$ is equal to

$$\omega(r) = (b + \delta(o))p(o) - d_{\mathrm{p}}(t(\alpha(o)), m) + d_{\mathrm{a}}(\lambda(r)),$$

and the departure delay $\omega'(r) = \sigma(o') - \chi(r)$ is equal to

$$\omega'(r) = d_{\mathrm{p}}(t(\alpha(o')), m') - b' p(o'),$$

where $\lambda(r) = ((\alpha(o), m), (\alpha(o'), m'))$. In order to increase the minimum upper bound we have to increase the delay assignment value $\delta(o)$, because the processing element assignment and the channel assignment have little effect on the computation delays $d_{\mathrm{p}}(t(\alpha(o)), m)$ and $d_{\mathrm{p}}(t(\alpha(o')), m')$, and the communication delay $d_{\mathrm{a}}(\lambda(r))$, respectively. The same holds for the maximum upper bound.

Once the mapping set satisfies the relaxations of the mapping constraints, we strengthen the relaxations of the mapping constraints until they are exact approximations of the original mapping constraints. For the precedence constraints this means that we reduce the sets $\Omega(r)$ and $\Omega'(r)$ of arrival and departure delays to singleton sets $\{\omega(r)\}$ and $\{\omega'(r)\}$, respectively. The arrival delay $\omega(r)$ is fixed if the delay assignment value $\delta(o)$, the processing element type $t(\alpha(o))$, and the communication delay $d_{\mathrm{a}}(\lambda(r))$ are fixed. The departure delay $\omega'(r)$ is fixed if the processing element type $t(\alpha(o'))$ is fixed. The channel assignment does not affect the computation delay provided that the relaxation of the type constraints is satisfied because the computation delay only depends on the terminal type rather than on the terminal instance. Hence, to approximate the precedence constraints exactly, we have to fix the delay assignment such that $|\Delta| = 1$, we have to fix the assignment of operations to processing element types such that the type constraints are satisfied, and we have to fix the communication delays of the data precedences. In order to determine the latter it suffices to assign the operations to processors in such a way that the connectivity constraints are satisfied, because the intraprocessor communication between processing elements is implemented by switch matrices which ensure full connectivity and constant communication delays. In general it is not

possible to assign the operations to processors in such a way that the connectivity constraints and the relaxations of the computation and storage constraints are satisfied without expanding the signal flow graph with pass operations that implement the interprocessor communication. The reason for this is that the computation and storage resources, i.e., arithmetic and logic elements, memory elements, and buffer elements, of different processors are interconnected indirectly via communication resources, i.e., output elements, rather than that they are interconnected directly. The additional pass operations have to be mapped onto output elements in order to implement the interprocessor communication.

Once the mapping set satisfies the type, connectivity, and precedence constraints, we have to find a mapping in the mapping set that satisfies the computation, communication, storage, periodicity, and array constraints. This means that we have to find an appropriate time assignment $\sigma \in \Sigma$, processing element assignment $\alpha \in A$, and channel assignment $\lambda \in \Lambda$. Note that the signal flow graph expansions that are required to satisfy the connectivity and the precedence constraints increase the computation, communication, and storage requirements, and thus make it more difficult to satisfy the corresponding constraints.

Based on the above-mentioned argumentation, we propose a decomposition of the mapping problem into three subproblems that are called the delay management problem, the partitioning problem, and the scheduling problem. The primary goal of delay management is to handle the precedence constraints. More precisely, for a given signal flow graph, processing element network, and mapping set the problem is to find an expansion of the signal flow graph and a mapping set that satisfies the relaxation of the precedence constraints, and the relaxations of the type, periodicity, array, storage, computation, and communication constraints. The primary goal of partitioning is to handle the connectivity constraints. More precisely, for a given signal flow graph, processing element network, and mapping set the problem is to find an expansion of the signal flow graph and a mapping set that satisfies the relaxation of the connectivity constraints, and the relaxations of the type, periodicity, array, storage, computation, and communication constraints. The primary goal of scheduling is to handle the computation and communication constraints. More precisely, for a given signal flow graph, processing element network, and mapping set that satisfy the relaxation of the mapping constraints the problem is to find a mapping in the mapping set that satisfies the mapping constraints.

We handle the mapping problem by consecutively handling the delay management problem, the partitioning problem, and the scheduling problem. The motivation for this order is as follows. The signal flow graph expansion during delay management increases the computation, communication, and storage requirements in order to satisfy the precedence constraints. Delay management precedes partitioning because partitioning partitions the resource requirements over the processors,

thereby taking the additional resource requirements into account. A disadvantage of this decision is that we have to approximate the computation and the communication delays because these are only known after partitioning. However, in practice these delays are limited to a few clock cycles. The signal flow graph expansion during partitioning increases the computation and communication requirements in order to satisfy the connectivity constraints. Partitioning precedes scheduling because scheduling schedules the resource requirements onto the processing elements, thereby taking the additional resource requirements for the output elements into account. In practice it may be required to repeat delay management between partitioning and scheduling, because partitioning may expand the signal flow graph in such a way that the resulting computation and communication delays violate the precedence constraints. We expect that in most of these cases the additionally required storage delay is small such that it can be implemented on the processor on which it is required, thereby preventing the partitioning of the delay over multiple processors.

4.4 Delay Management Problem

The input of delay management is a signal flow graph, a processing element network, and a mapping set that satisfies the relaxations of the type, array, connectivity, computation, communication, periodicity, and storage constraints. The output of delay management is an expanded signal flow graph and a mapping set that satisfies the relaxation of the precedence constraints as well as the relaxations of the type, array, connectivity, computation, communication, periodicity, and storage constraints.

The most important decision variable in the delay management problem is the delay assignment, because its value can be used to control the arrival delays $\omega(r)$ of the data precedences. In order to approximate the precedence constraints exactly, the set $\Omega(r)$ has to be a singleton set $\{\omega(r)\}$. For this reason we impose the additional constraint in the delay management problem that a delay assignment set is a singleton set, i.e., $|\Delta| = 1$.

Furthermore we have to approximate the communication delays because we do not have a channel assignment yet. We assume that the unknown communication delays equal zero because in practice they are limited to a few clock cycles. This assumption guarantees that the approximation of the lower bound $\underline{\omega}(r)$ is indeed a lower bound of $\underline{\omega}(r)$. As a result the lower bound approximation does not remove feasible solutions to the mapping problem from the search space. However, the approximation of the upper bound $\overline{\omega}(r)$ is not necessarily an upper bound of $\overline{\omega}(r)$. As a result the upper bound approximation may lead to an underestimation of the sum of the upper bounds in a cycle. This may remove feasible solutions from the search

space because the value of the delay assignment or the expansion of the signal flow graph, and thus the computation, communication, and storage requirements, may become larger than necessary. The approximation of the upper bound is acceptable because of the fact that the resource utilization is very high if the search space does not contain other solutions. In that case it is unlikely that we can solve the given instance of the mapping problem anyway.

Definition 4.10 (Delay Management Problem). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set of (O,R) onto (P,C) that satisfies the relaxations of the type, array, connectivity, computation, communication, periodicity, and storage constraints. Find a signal flow graph (O',R') such that $(O,R) \subseteq (O',R')$ and find a mapping set $(\Delta',\Sigma',\mathrm{A}',\Lambda')$ of (O',R') onto (P,C) such that $|\Delta'| = 1$ and the relaxations of the mapping constraints are satisfied, if they exist. □

Theorem 4.1. *The delay management problem is NP-complete in the strong sense.*

Proof. The proof immediately follows from Theorem 3.9 because the problem of expanding a signal flow graph and constructing a processing element, channel, and time assignment that satisfy the computation, connectivity, and precedence constraints is a special case of the delay management problem if the periods form a divisible sequence. □

4.5 Partitioning Problem

The input of partitioning is a signal flow graph, a processing element network, and a mapping set that satisfies the relaxations of the type, array, connectivity, computation, communication, periodicity, and storage constraints. The output of partitioning is an expanded signal flow graph and a mapping set that satisfies the connectivity constraints and the relaxations of the type, array, computation, communication, periodicity, and storage constraints. We do not consider precedence constraints during partitioning because they are the subject of delay management.

The most important decision variable in the partitioning problem is the processing element assignment, because its value can be used to satisfy the connectivity constraints. We are interested in the interprocessor connectivity only, because the intraprocessor connectivity is not constrained as a result of the switch matrices of the processors. For this reason we impose the additional constraint in the partitioning problem that a processing element assignment set assigns operations to processing elements of the same processor, i.e., $|V(\mathrm{A}(o))| = 1$, where $V(P) = \{v \mid (v,e) \in P\}$ for all processing element sets P. In order to approximate the precedence constraints exactly, the sets $\Omega(r)$ and $\Omega'(r)$ have to be singleton sets. For this reason we impose the additional constraint in the partitioning problem

that a processing element assignment set assigns operations to processing elements of the same type, i.e., $|T(\mathrm{A}(o))| = 1$, where $T(P) = \{t(p) \mid p \in P\}$ for all processing element sets P. As a result of these additional constraints the computation and communication delays are known after partitioning.

Definition 4.11 (Partitioning Problem). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set of (O,R) onto (P,C) that satisfies the relaxations of the type, array, connectivity, computation, communication, periodicity, and storage constraints. Find a signal flow graph (O',R') such that $(O,R) \subseteq (O',R')$ and find a mapping set $(\Delta',\Sigma',\mathrm{A}',\Lambda')$ of (O',R') onto (P,C) such that $|V(\mathrm{A}'(o))| = |T(\mathrm{A}'(o))| = 1$, for all $o \in O'$, the connectivity constraints, and the relaxations of the type, array, computation, communication, periodicity, and storage constraints are satisfied, if they exist. □

Theorem 4.2. *The partitioning problem is NP-complete in the strong sense.*

Proof. The proof immediately follows from Theorem 3.10 because the problem of expanding a signal flow graph and constructing a time, processing element, and channel assignment that satisfy the connectivity, computation, and communication constraints is a special case of the partitioning problem if the periods form a divisible sequence. □

4.6 Scheduling Problem

The input of scheduling is a signal flow graph, a processing element network, and a mapping set for which it holds that $|\Delta| = |V(\mathrm{A}(o))| = |T(\mathrm{A}(o))| = 1$, for all operations o, and that satisfies the connectivity constraints and the relaxations of the mapping constraints. The output of scheduling is a mapping that satisfies the mapping constraints.

Definition 4.12 (Scheduling Problem). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set such that $|\Delta| = |V(\mathrm{A}(o))| = |T(\mathrm{A}(o))| = 1$, for all $o \in O$, the connectivity constraints, and the relaxation of the mapping constraints are satisfied. Find a mapping $(\delta,\sigma,\alpha,\lambda) \in (\Delta,\Sigma,\mathrm{A},\Lambda)$ that satisfies the mapping constraints, if one exists. □

Theorem 4.3. *The scheduling problem is NP-complete in the strong sense.*

Proof. The proof immediately follows from Theorem 3.6 since the problem of constructing a processing element assignment satisfying the computation constraints for a given time assignment is a special case of the scheduling problem. □

4.7 Results

In this section we present the results of the constraint relaxations on the set of problem instances presented in Chapter 2. The results include the resource requirements of the signal flow graphs and the resource capacities of the processor networks from which we obtain the degree of utilization of the resources. The resource constraints that we consider are the relaxations of the computation and storage constraints presented in Definitions 4.3 and 4.5.

Table 4.1. Processor networks and their characteristics. From left to right the columns list the computation capacity C_A per processing element type and the storage capacity C_Δ.

Processor Network	C_A ALE	ME1	ME2	BE	OE	IN	OUT	C_Δ
VSP1FLEX	24	16	0	0	40	6	6	8192
VSP2TEST	12	0	4	6	6	6	6	8192
VSP2FLEX	72	0	24	36	36	8	8	49152

Table 4.1 lists the computation and storage capacities of the three processor networks that occur in the problem instances. Note that the clock frequency of the second generation processors is twice as high as the clock frequency of the first generation processors. For this reason, the twelve arithmetic and logic elements of the single second generation processors in the VSP2TEST processor network provide the same computation capacity as the twenty-four arithmetic and logic elements of the eight first generation processors in the VSP1FLEX processor network. The dual ported memory elements of the second generation processor do not entirely provide this factor of eight over the single ported memory elements because the address port has been split into a read address port and a write address port. This means that the VSP2TEST processor network can execute eight read operations and eight write operations whereas the VSP1FLEX processor network can execute any combination of sixteen read or write operations in same amount of time.

Table 4.2 lists the computation and storage requirements of the signal flow graphs that are to be mapped onto the VSP1FLEX processor network. The computation requirements are listed according to the combinations of the processing element types that have been specified in the relaxation of the type constraints. During the mapping of signal flow graphs onto first generation processors only the computation requirement of the arithmetic and logic elements and the output elements changes because of the signal flow graph transformations.

Table 4.3 lists the computation and storage requirements of the signal flow graphs that are to be mapped onto the VSP2TEST processor network. The period of the operations when mapping onto second generation processors is twice as

Table 4.2. Signal flow graphs that are to be mapped onto the VSP1FLEX processor network. From left to right the columns list the computation requirement R_A per combination of processing element types and the storage requirement R_Δ.

Signal Flow Graph	R_A					R_Δ
	ALE	ALE OE	ME1	IN	OUT	
YUVTORGB	17.00	17.00	0.00	1.00	0.50	0
HORCOMPR	5.00	5.00	2.00	3.00	3.00	2048
IJNTEMA1	11.19	11.19	3.00	2.56	5.00	1580
CORMACK2	8.81	8.81	1.44	3.00	4.00	2564
CONTOUR1	17.44	17.44	3.13	2.00	2.00	3024
MONZA2	19.63	19.63	4.00	3.00	3.00	3440
VDP	16.25	16.25	4.88	3.00	3.50	3568
GAMMA	12.06	12.06	3.50	2.00	3.00	1264
HISTMOD2	9.00	9.00	3.44	1.50	3.00	957
PANORAMA	12.06	12.06	8.00	1.00	2.00	6144
VIDIWALL	18.69	18.69	5.56	3.50	6.00	4696
IJNTEMA2	20.56	20.56	3.13	3.00	3.00	2076
CORMACK1	21.81	21.81	11.50	3.00	4.00	6432
MONZA1	17.69	17.69	5.00	0.31	3.00	4411
MWTV	17.94	17.94	4.75	0.06	3.00	3424

large compared to mapping onto first generation processors in order to compensate for the double clock frequency. Note that the instance that maps signal flow graph CORMACK1 onto processor network VSP2TEST is infeasible because the computation capacity of the dual ported memory elements is insufficient. During the mapping of signal flow graphs onto second generation processors the computation requirement of the arithmetic and logic elements, the buffer elements, the output elements, and the dual ported memory elements as well as the storage requirement changes because of the signal flow graph transformations.

Table 4.4 lists the computation and storage requirements of the signal flow graphs that are to be mapped onto the VSP2FLEX processor network.

4.8 Summary

In this chapter we have decomposed the mapping problem into the delay management problem, the partitioning problem, and the scheduling problem. The objective of the delay management problem is to add operations to a signal flow graph that store the operands in memories thereby taking the combination of precedence, computation, and connectivity constraints into account. The objective of partitioning is to add operations to a signal flow graph that communicate operands along the

Table 4.3. Signal flow graphs that are to be mapped onto the VSP2TEST processor network. From left to right the columns list the computation requirement R_A per combination of processing element types and the storage requirement R_Δ.

Signal Flow Graph	R_A ALE	ALE BE	ALE BE, OE	ME2	IN	OUT	R_Δ
YUVTORGB	8.50	8.50	8.50	0.00	0.50	0.25	0
HORCOMPR	2.50	2.50	2.50	0.50	1.50	1.50	2048
IJNTEMA1	5.53	5.59	5.59	0.91	1.28	2.50	1580
CORMACK2	4.41	4.41	4.41	0.72	1.50	2.00	2564
CONTOUR1	8.72	8.72	8.72	0.78	1.00	1.00	3024
MONZA2	9.81	9.81	9.81	1.38	1.50	1.50	3440
VDP	8.13	8.13	8.13	1.44	1.50	1.75	3568
GAMMA	6.03	6.03	6.03	1.00	1.00	1.50	1264
HISTMOD2	4.50	4.50	4.50	0.94	0.75	1.50	957
PANORAMA	6.03	6.03	6.03	2.25	0.50	1.00	6144
VIDIWALL	9.34	9.34	9.34	1.47	1.75	3.00	4696
IJNTEMA2	10.28	10.28	10.28	1.09	1.50	1.50	2076
CORMACK1	10.91	10.91	10.91	4.25	1.50	2.00	6432
MONZA1	8.84	8.84	8.84	1.25	0.16	1.50	4411
MWTV	8.97	8.97	8.97	1.75	0.03	1.50	3424

Table 4.4. Signal flow graphs that are to be mapped onto the VSP2FLEX processor network. From left to right the columns list the computation requirement R_A per combination of processing element types and the storage requirement R_Δ.

Signal Flow Graph	R_A ALE	ALE BE	ALE BE, OE	ME2	IN	OUT	R_Δ
CONTRAST	40.06	42.56	42.56	3.00	1.13	1.00	12768
FDXD1	26.09	26.22	26.22	1.12	4.00	7.00	549
FDXD2	30.75	30.75	30.75	5.16	2.56	3.50	8796
MAT30UP	30.63	30.63	30.63	1.38	0.50	1.50	2773
HSRC	34.90	38.19	38.91	6.06	3.00	2.00	13694
VSRC	32.56	34.53	34.53	5.25	5.00	6.00	9937

communication channels thereby taking the combination of computation, communication, and connectivity constraints into account. The objective of the scheduling problem is to schedule the operations of a signal flow graph on the processing elements thereby taking the mapping constraints into account. In order to formally define these problems we have generalized the notion of mapping to the notion of

mapping set and we have relaxed the mapping constraints. These relaxations provide necessary conditions for feasibility such that mappings that are inconsistent with the decisions made so far can be removed from the mapping set.

5

Delay Management

In this chapter we present a solution strategy for the delay management problem. The solution strategy is based on a decomposition of the delay management problem into a delay minimization problem and a delay assignment problem. The objective of the delay minimization problem is to minimize the additional memory requirements that are needed to satisfy the precedence constraints. The objective of the delay assignment problem is to partition these memory requirements into computation requirements and storage requirements.

The outline of the chapter is as follows. In Section 5.1 we decompose the delay management problem into the delay minimization problem and the delay assignment problem. In Sections 5.2 and 5.3 we present solution approaches to handle these subproblems. In Section 5.4 we present the results of the delay management approach on the set of problem instances. Finally, in Section 5.5 we summarize the contents of this chapter.

5.1 Problem Decomposition

In Chapter 4 we have formulated the delay management problem as follows.

Definition 5.1 (Delay Management Problem). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set of

(O,R) onto (P,C) that satisfies the relaxations of the type, array, connectivity, computation, communication, periodicity, and storage constraints. Find a signal flow graph (O',R') such that $(O,R) \subseteq (O',R')$ and find a mapping set $(\Delta',\Sigma',\mathrm{A}',\Lambda')$ of (O',R') onto (P,C) such that $|\Delta'| = 1$ and the relaxations of the mapping constraints are satisfied, if they exist. □

The objective of the delay management problem is to satisfy the relaxation of the precedence constraints, i.e., for all data precedences $r = ((o,n),(o',n'),(p,b,b'))$ it must hold that

$$\min \underline{\Omega}(r) \leq \min \Sigma(o') - \min \Sigma(o) \leq \min \overline{\Omega}(r) \text{ and}$$

$$\max \underline{\Omega}(r) \leq \max \Sigma(o') - \max \Sigma(o) \leq \max \overline{\Omega}(r),$$

and for all no-value precedences $s = (o,o',(p,b,b'))$ it must hold that

$$\min \underline{\Omega}(s) \leq \min \Sigma(o') - \min \Sigma(o) \text{ and}$$

$$\max \underline{\Omega}(s) \leq \max \Sigma(o') - \max \Sigma(o).$$

To satisfy the relaxation of the precedence constraints it suffices to find lower bounds $\underline{\Omega}(a) = \max \underline{\Omega}(a)$ and upper bounds $\overline{\Omega}(a) = \min \overline{\Omega}(a)$ for all precedences a that satisfy Corollary 4.1, i.e., for all cycles $L \subseteq R \cup S$ it must hold that

$$\sum_{a \in L} \underline{\Omega}(a),$$

and for all cycles $L \subseteq R$ it must hold that

$$\sum_{a \in L} \overline{\Omega}(a).$$

The corresponding completion times can be computed with a longest path algorithm as we have outlined in the proof of Theorem 3.8.

To compute the lower and upper bounds we decompose the delay management problem into a delay minimization problem and a delay assignment problem. The objective of the delay minimization problem is to compute a time assignment that satisfies the lower bounds and minimizes the memory requirements to satisfy the upper bounds. We estimate the memory requirements by the number of memory locations that is needed to store the samples thereby assuming that duplication of samples is avoided as much as possible. The objective of the delay assignment problem is to determine whether these samples are stored in silos or in random access memories of dual ported memory elements.

5.1.1 Delay Minimization problem

To formulate the delay minimization problem we express the number of memory locations that is needed to satisfy the the relaxation of the precedence constraints as a cost function of the time assignment. If a time assignment violates the upper

bound $\overline{\Omega}(r)$ of a data precedence r then we require an additional storage time of $\sigma(o') - \sigma(o) - \overline{\Omega}(r)$ clock cycles. We divide this number by the period of the precedence to estimate the number of memory locations. We do not store duplicates of samples. Therefore we consider only the maximum storage time for each produced sample, i.e., if a production has multiple consumptions then we consider only the latest consumption. In the subsequent delay assignment step we ensure that earlier consumptions have access to the sample. Formally the delay minimization is defined as follows.

Definition 5.2 (Delay Minimization Problem). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set of (O,R) onto (P,C) that satisfies the relaxations of the type, array, connectivity, computation, communication, periodicity, and storage constraints. Find a time assignment $\sigma \in \Sigma$ such that for all precedences $a \in R \cup S$ it holds that

$$\underline{\Omega}(r) \leq \sigma(o') - \sigma(o),$$

and such that

$$f(\sigma) = \sum_{o \in O} \sum_{i=0}^{q(o)-1} \max\{\frac{\max\{0, \sigma(o') - \sigma(o) - \overline{\Omega}(r)\}}{p(o)q(o)} \mid r \in R((o,n)[i])\}$$

is minimal, where the number $q(o)$ is defined for all $o \in O$ by

$$q(o) = \mathrm{lcm}\{\frac{p}{p(o)} \mid ((o,n),(o',n'),(p,b,b')) \in R\},$$

and the set $R((o,n)[i])$ is defined for all $o \in O$ and for all $i \in \mathbb{Z}$ by

$$R((o,n)[i]) = \{((o,n),(o',n'),(p,b,b')) \in R \mid b \equiv i \pmod{\frac{p}{p(o)}}\},$$

if one exists. □

The number $q(o)$ is the smallest number of invocations of operation o that is required to repeat the communication behavior between operation o and its consumers. It is defined to be zero if the domain of the least common multiple is empty. The set $R((o,n)[i])$ is the set of data precedences that consume the value of the ith invocation of output terminal (o,n). A data precedence contributes to the value of the cost function if and only if it violates its upper bound. The contribution is equal to the difference of the difference of the completion times and the upper bound divided by the period of production. We count only the largest contribution whenever different data precedences consume the same production to avoid duplication of the same sample. We have defined the cost function in such a way that its value equals zero if and only if the solution to the delay minimization problem is a solution to the delay management problem.

Theorem 5.1. *An instance of the delay minimization problem has a solution with zero cost if and only if it contains a solution to the delay management problem.*

Proof. Let $(\Delta, \Sigma, \mathrm{A}, \Lambda)$ be an instance of the delay minimization problem and let $\sigma \in \Sigma$ be a solution to the instance. Then for all precedences a it holds that $\underline{\Omega}(r) \leq \sigma(o') - \sigma(o)$. The solution has zero cost if and only if it holds that $\sigma(o') - \sigma(o) \leq \overline{\Omega}(r)$ for all operations $o \in O$, for all integers $i \in \mathbb{Z}$, and for all data precedences $r \in R((o,n)[i])$. In other words, the solution has zero cost if and only if $(\Delta, \{\sigma\}, \mathrm{A}, \Lambda)$ is a solution to the delay management problem. □

5.1.2 Delay Assignment Problem

If an instance of the delay minimization problem does not have a solution with zero cost, then there exists some infeasible upper bound. To solve the delay management problem we have to change the set of upper bounds by expanding the signal flow graph or increasing the delay assignment value.

Theorem 5.2. *Let $(\Delta, \Sigma, \mathrm{A}, \Lambda)$ be an instance of the delay minimization problem and let $\sigma \in \Sigma$ be an optimal solution with nonzero cost. Then there exist an expanded mapping set $(\Delta', \Sigma', \mathrm{A}', \Lambda')$ and a solution $\sigma' \in \Sigma'$ such that $f(\sigma') < f(\sigma)$.*

Proof. Let $(\Delta, \Sigma, \mathrm{A}, \Lambda)$ be an instance of the delay minimization problem and let $\sigma \in \Sigma$ be an optimal solution with nonzero cost. Then there exist an operation $o \in O$, an integer $0 \leq i < q(o)$, and data precedences $r = ((o,n),(o',n'),(p,b,b')) \in R((o,n)[i])$ satisfying $\sigma(o') - \sigma(o) > \overline{\Omega}(r)$.

Now we expand the signal flow graph as follows. We decimate these data precedences such that their period becomes equal to $p(o)q(o)$ by replacing each data precedence r with $k = p(o)q(o)/p$ new data precedences $r_{i'} = ((o,n),(o',n'),(p(o)q(o), b + i'p/p(o), b' + i'p/p(o')))$ where $0 \leq i' < k$. This transformation does not violate the constraints and does not change the value of the cost function. Since $p(o)|p$ and thus $k \leq q(o)$, it holds that only the data precedences r_0 consume the given invocation i, i.e., $r_0 \in R((o,n)[i])$. We expand the signal flow graph by adding a pass operation o_i with period $p(o_i) = p(o)q(o)$ and replacing the set of data precedences containing all r_0's with data precedences $r'_0 = ((o_i,0),(o',n'),(p(o)q(o),0,b'))$ and one additional data precedence $r' = ((o,n),(o_i,0),(p(o)q(o),b,0))$.

Finally, we adapt the mapping set to the new pass operation o_i. To satisfy the lower bound of the precedence constraints, $\sigma(o_i)$ must satisfy $\sigma(o) + \underline{\Omega}(r') \leq \sigma(o_i) \leq \sigma(o') - \underline{\Omega}(r'_0)$ for all $r'_0 \in R((o_i,0)[0])$, which is feasible if and only if $\delta(o_i)$ is chosen such that $\underline{\Omega}(r') + \underline{\Omega}(r'_0) \leq \sigma(o') - \sigma(o)$. Since we have that $\sigma(o') - \sigma(o) > \overline{\Omega}(r)$, it suffices to satisfy the constraint $\underline{\Omega}(r') + \underline{\Omega}(r'_0) < \overline{\Omega}(r)$ by choosing $\delta(o_i) = 0$. Using that $\overline{\Omega}(r) < \overline{\Omega}(r') + \overline{\Omega}(r'_0)$ we have that $\sigma(o') - \sigma(o) - \overline{\Omega}(r) > \sigma(o') - \sigma(o_i) - \overline{\Omega}(r'_0) + \sigma(o_i) - \sigma(o) - \overline{\Omega}(r')$, which completes the proof. □

A potential problem of the above-mentioned expansion is that the expansion can violate the relaxation of other constraints than the precedence constraints. The objective of the delay assignment problem is to expand an instance of the delay minimization problem in such a way that the expansion does not violate the constraint relaxations.

Definition 5.3 (Delay Assignment Problem). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set of (O,R) onto (P,C) that satisfies the relaxations of the type, array, connectivity, computation, communication, periodicity, and storage constraints, and let $\sigma \in \Sigma$ be a time assignment such that for all precedences $a \in R \cup S$ it holds that

$$\underline{\Omega}(a) \leq \sigma(o') - \sigma(o).$$

Find a signal flow graph (O',R') such that $(O,R) \subseteq (O',R')$ and find a mapping set $(\Delta',\Sigma',\mathrm{A}',\Lambda')$ of (O',R') onto (P,C) such that $|\Delta| = 1$ and the relaxations of the precedence, type, array, connectivity, computation, communication, periodicity, and storage are satisfied, if they exist. □

5.2 Delay Minimization

In this section we present an approach to solve the delay minimization problem. We show that the delay minimization problem is a special case of the dual of the minimum cost flow problem. From the literature we know that the primal of the minimum cost flow problem is a special case of integer linear programming which is solvable in polynomial time. We furthermore know that primal/dual linear programming pairs have equal cost at optimality. Consequently, we can solve the delay minimization problem in polynomial time.

5.2.1 Minimum Cost Flow

Linear programming problems are usually denoted in three forms called general, canonical, and standard form. In general form we are given an $m \times n$ matrix A with rows $\boldsymbol{a_i}$, a set of row indices M corresponding to equality constraints, and a set of row indices $\bar{M}$ corresponding to inequality constraints. Furthermore we have a vector $\boldsymbol{x} \in \mathbb{R}^n$, a set of column indices N corresponding to constrained variables, and a set of column indices $\bar{N}$ corresponding to unconstrained variables. Finally we have a vector $\boldsymbol{b} \in \mathbb{R}^m$ and a vector $\boldsymbol{c} \in \mathbb{R}^n$. An instance of a linear programming

problem in general form is then defined in matrix notation as

$$\begin{array}{lll} \text{minimize} & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} & \\ \text{subject to} & \boldsymbol{a_i}^{\mathrm{T}}\boldsymbol{x} = b_i & \text{for all } i \in M \\ & \boldsymbol{a_i}^{\mathrm{T}}\boldsymbol{x} \geq b_i & \text{for all } i \in \bar{M} \\ & x_j \geq 0 & \text{for all } j \in N \\ & x_j \neq 0 & \text{for all } j \in \bar{N}. \end{array}$$

In canonical form we eliminate all equality constraints and unconstrained variables from the general form. The equality constraints $\boldsymbol{a_i}^{\mathrm{T}}\boldsymbol{x} = b_i$ are replaced by two inequality constraints $\boldsymbol{a_i}^{\mathrm{T}}\boldsymbol{x} \geq b_i$ and $-\boldsymbol{a_i}^{\mathrm{T}}\boldsymbol{x} \geq -b_i$. The unconstrained variables x_j are replaced by two constrained variables x_j^+ and x_j^- such that $x_j = x_j^+ - x_j^-$, $x_j^+ \geq 0$, and $x_j^- \geq 0$. An instance of a linear programming problem in canonical form is then defined in matrix notation as

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}. \end{array}$$

In standard form we eliminate all inequality constraints and unconstrained variables from the general form. The inequality constraints $\boldsymbol{a_i}^{\mathrm{T}}\boldsymbol{x} \geq b_i$ are replaced by equality constraints $\boldsymbol{a_i}^{\mathrm{T}}\boldsymbol{x} - s_i = b_i$. The unconstrained variables are replaced in the same way as before. An instance of a linear programming problem in standard form is then defined in matrix notation as

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}. \end{array}$$

If we restrict the solution space to integer solutions, then we speak of integer linear programming problems.

Network flow problems are special linear programming problems. Here we adopt the network flow model presented by Ahuja et al. [1989]. They represent a network by a directed graph. The flow in the network is running along the arcs. Each node has a supply or demand. The incoming flow minus the outgoing flow of each node must meet this supply or demand. Each arc has a lower and upper bound. The amount of flow along each arc must be between this lower and upper bound. Furthermore, each arc has a cost per unit of flow. We are interested in the minimum cost flow problem in which the objective is to minimize the total flow cost. We may assume without loss of generality that the lower bounds on the arc flows equal zero and the arc costs per unit of flow are nonnegative. This results in

the following definition of the minimum cost flow problem.

Definition 5.4 (Minimum Cost Flow Problem). Let $G = (N,A)$ be a directed graph. With each arc $(i,j) \in A$ we associate a cost $c_{ij} \in \mathbb{R}$ and an upper bound $u_{ij} \in \mathbb{R}$ satisfying $c_{ij} \geq 0$ and $u_{ij} > 0$. With each node $i \in N$ we associate an integer $b_i \in \mathbb{Z}$ representing its supply or demand. Node i is a *supply* node if $b_i > 0$, node i is a *demand* node if $b_i < 0$, and node i is a *transshipment* node if $b_i = 0$. Find a *flow* $x_{ij} \in \mathbb{Z}$ for each arc $(i,j) \in A$ such that the *mass balance constraints* are satisfied, i.e., for all $i \in N$ it holds that

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = b_i,$$

the *flow bound constraints* are satisfied, i.e., for all $(i,j) \in A$ it holds that

$$0 \leq x_{ij} \leq u_{ij},$$

and the flow cost

$$\sum_{(i,j)\in A} c_{ij} x_{ij}$$

is minimal. □

Here the minimum cost flow problem is formulated as an integer linear programming problem in general form. If we represent the set of arcs A as incidence matrix $\boldsymbol{A}$, then we can transform this to an integer linear programming problem in standard form according to

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{subject to} & \begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{s} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{u} \end{pmatrix} \\ & \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{s} \end{pmatrix} \geq \boldsymbol{0}. \end{array}$$

Note that the matrix $\boldsymbol{A}$ is totally unimodular, i.e., all of its square submatrices have determinants equal to -1, 0 , or 1. Consequently, the minimum cost flow problem is solvable in polynomial time which is a well-known result.

In order to find the dual of the minimum cost flow problem we write the integer linear programming formulation in matrix/vector notation as

$$\begin{array}{ll} \begin{pmatrix} \boldsymbol{c}^{\mathrm{T}} & \boldsymbol{0} \end{pmatrix} & \\ \begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{I} \end{pmatrix} & \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{u} \end{pmatrix} \end{array}.$$

Transposition of the matrix/vector notation yields the dual of the minimum cost flow problem in which the equalities have been replaced by inequalities and the

cost function has been negated, i.e.,

$$\begin{pmatrix} \boldsymbol{c} \\ \boldsymbol{0} \end{pmatrix} \quad \begin{pmatrix} \boldsymbol{A}^{\mathrm{T}} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix}$$
$$\begin{pmatrix} \boldsymbol{b}^{\mathrm{T}} & \boldsymbol{u}^{\mathrm{T}} \end{pmatrix}$$

With the equality constraints $\boldsymbol{Ax} = \boldsymbol{b}$ we associate a dual vector $\boldsymbol{\pi} \in \mathbb{Z}^m$ and with the equality constraints $\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{u}$ we associate a vector $\boldsymbol{\delta} \in \mathbb{Z}^n$. The resulting integer linear programming formulation is

$$\begin{aligned} \text{Maximize} \quad & \boldsymbol{b}^{\mathrm{T}}\boldsymbol{\pi} + \boldsymbol{u}^{\mathrm{T}}\boldsymbol{\delta} \\ \text{subject to} \quad & \boldsymbol{c} \geq \boldsymbol{A}^{\mathrm{T}}\boldsymbol{\pi} + \boldsymbol{\delta} \\ & \boldsymbol{0} \geq \boldsymbol{\delta}. \end{aligned}$$

If we negate the vector $\boldsymbol{\delta}$ and use the fact that the matrix $\boldsymbol{A}$ is an incidence matrix we obtain the dual of the minimum cost flow problem as formulated by Ahuja et al. [1989].

Definition 5.5 (Dual of the Minimum Cost Flow Problem). Let $G = (N, A)$ be a directed graph. With each arc $(i, j) \in A$ we associate a cost c_{ij} and an upper bound u_{ij} satisfying $c_{ij} \geq 0$ and $u_{ij} > 0$. With each node $i \in N$ we associate an integer $b_i \in \mathbb{Z}$ representing its supply or demand. Find a dual variable π_i for the mass balance constraint of each node $i \in N$ and a dual variable δ_{ij} for the flow bound constraint of each arc $(i, j) \in A$ such that for all $(i, j) \in A$ it holds that

$$\pi_i - \pi_j - \delta_{ij} \leq c_{ij},$$

$$\delta_{ij} \geq 0,$$

and

$$\sum_{i \in N} b_i \pi_i - \sum_{(i,j) \in A} u_{ij} \delta_{ij},$$

is maximal. □

From the literature we know that if a linear programming problem has an optimal solution then its dual also has an optimal solution and that their optimal costs are equal [Papadimitriou and Steiglitz, 1982]. Hence, the dual of the minimum cost flow problem is solvable in polynomial time

5.2.2 Delay Minimization Reformulated

To formulate the delay minimization problem as an instance of the dual of the minimum cost flow problem we remove the nonlinearity in the cost function. To this end we introduce additional decision variables, denoted by $\sigma(o, i)$, for all operations $o \in O$ and for all integers $0 \leq i < q(o)$.

Definition 5.6 (Network Flow Variant of the Delay Minimization Problem). Let $(\Delta, \Sigma, \mathrm{A}, \Lambda)$ be a mapping set of signal flow graph (O,R) onto processing element network (P,C). Find a time assignment $\sigma \in \Sigma$ and additional decision variables $\sigma(o,i)$ for all operations $o \in O$ and for all integers $0 \le i < q(o)$ such that for all precedences $a \in R \cup S$ it holds that

$$\underline{\Omega}(a) \le \sigma(o') - \sigma(o),$$

and for all output terminals (o,n), for all integers $0 \le i < q(o)$, and for all precedences $r \in R((o,n)[i])$ it holds that

$$\sigma(o) \le \sigma(o,i) \wedge \sigma(o') - \sigma(o,i) \le \overline{\Omega}(r),$$

and

$$\sum_{o \in O} \sum_{i=0}^{q(o)-1} \frac{\sigma(o,i) - \sigma(o)}{p(o)q(o)}$$

is minimal, if they exist. □

Theorem 5.3. *An instance of the delay minimization problem has a solution if and only if its network flow variant has a solution and their optimal costs are equal.*

Proof. Suppose we have given an optimal solution to an instance of the delay minimization problem. Then we define additional decision variables

$$\sigma(o,i) = \sigma(o) + \max\{\max\{0, \sigma(o') - \sigma(o) - \overline{\Omega}(r)\} \mid r \in R((o,n)[i])\},$$

which satisfies the constraints

$$\sigma(o) \le \sigma(o,i) \wedge \sigma(o') - \sigma(o,i) \le \overline{\Omega}(r).$$

The values of the decision variables are minimal because any smaller values would violate the constraint $\sigma(o') - \sigma(o,i) \le \overline{\Omega}(r)$.

Now suppose we have given an optimal solution to an instance of the network flow variant of the delay minimization problem. Then the satisfied constraints

$$\sigma(o) \le \sigma(o,i) \wedge \sigma(o') - \sigma(o,i) \le \overline{\Omega}(r)$$

can be written as

$$\sigma(o,i) - \sigma(o) \ge \max\{0, \sigma(o') - \sigma(o) - \overline{\Omega}(r)\},$$

for all data precedences $r \in R((o,n)[i])$. The assumption

$$\sigma(o,i) - \sigma(o) > \max\{\max\{0, \sigma(o') - \sigma(o) - \overline{\Omega}(r)\} \mid r \in R((o,n)[i])\}$$

implies that

$$\sigma(o) \le (\sigma(o,i) - 1) \wedge \sigma(o') - (\sigma(o,i) - 1) \le \overline{\Omega}(r).$$

Subsequently we can lower the values of the decision variables $\sigma(o,i)$ without violating the constraints. This is however in contradiction with the fact that we

have given an optimal solution. Hence, our assumption is invalid such that it must hold that

$$\sigma(o,i) - \sigma(o) = \max\{\max\{0, \sigma(o') - \sigma(o) - \overline{\Omega}(r)\} \mid r \in R((o,n)[i])\},$$

which completes the proof. □

Theorem 5.4. *The delay minimization problem is a special case of the dual of the minimum cost flow problem.*

Proof. We rewrite a given instance of the delay minimization problem to its network flow variant. Subsequently we rewrite the cost function of instance of the network flow variant to

$$\sum_{o \in O} \frac{\sigma(o)}{p(o)} - \sum_{o \in O} \sum_{i=0}^{q(o)-1} \frac{\sigma(o,i)}{p(o)q(o)},$$

thereby negating the cost function which transforms the minimization problem into a maximization problem. This allows us to map the instance to an instance of the dual of the minimum cost flow problem with cost function

$$\sum_{i \in N} b_i \pi_i - \sum_{(i,j) \in A} u_{ij} \delta_{ij}$$

as follows. For the supply or demand b_o of each node $o \in O$ and $b_{o,i}$ of each additional node (o,i), where $0 \leq i < q(o)$, we define

$$b_o = \frac{\operatorname{lcm}\{p(o')q(o') \mid o' \in O\}}{p(o)} \wedge b_{o,i} = -\frac{\operatorname{lcm}\{p(o')q(o') \mid o' \in O\}}{p(o)q(o)}.$$

For the dual variables of the mass balance constraints π_o of each node $o \in O$ and $\pi_{o,i}$ of each additional node (o,i), where $0 \leq i < q(o)$, we define

$$\pi_o = \sigma(o) \wedge \pi_{o,i} = \sigma(o,i).$$

Furthermore, we define $\boldsymbol{\delta} = \mathbf{0}$ for the dual variables of the flow bound constraints. Finally, we define arcs with cost c_a for all precedences $a \in R \cup S$ as $c_a = -\underline{\Omega}(a)$, for the constraints $\sigma(o) \leq \sigma(o,i)$ we define $c_a = 0$, and for the constraints $\sigma(o') - \sigma(o,i) \leq \overline{\Omega}(r)$ we define $c_a = \overline{\Omega}(r)$, which completes the proof. □

Corollary 5.1. *The delay minimization problem is solvable in polynomial time.* □

5.2.3 Successive Shortest Path

Ahuja et al. [1989] present a polynomial-time algorithm to solve the dual of the minimum cost flow problem called *right-hand-side scaling* that is based on successive shortest paths. To present the algorithm we adopt some terminology concerning residual networks. The residual network $G(x)$ corresponding to a flow x is defined as follows. For each arc $a \in A$ we define an additional arc $\overline{a} \in A$. With these arcs we associate costs c_a and $c_{\overline{a}} = -c_a$, and residual capacities $r_a = u_a - x_a$

and $r_{\overline{a}} = x_a$. For each node $i \in N$ we define the excess e_i as

$$e_i = b_i + \sum_{\overline{a}=(j,i)\in A} x_{\overline{a}} - \sum_{a=(i,j)\in A} x_a,$$

and for each arc $a = (i, j) \in A$ we define nonnegative reduced cost $\overline{c}_a$ as

$$\overline{c}_a = c_a - \pi_i + \pi_j.$$

The right-hand-side scaling algorithm successively computes shortest paths in the residual network with respect to the reduced cost. For this purpose we have adopted Dijkstra's algorithm. Dijkstra's algorithm can be applied $n = |N|$ times to compute shortest path between all pairs of nodes in order to implement an all pair shortest path algorithm.

The right-hand-side scaling algorithm maintains a flow x that satisfies the nonnegativity and the capacity constraints but violates the supply and demand constraints of the nodes. At each iteration, the algorithm selects a node k with a sufficiently large supply and a node l with a sufficiently large demand and augments the flow from k to l along a directed shortest path in the residual network. The algorithm terminates when the supply and demand constraints are met. The algorithm computes a shortest path maximally $n \log U$ times where U is an upper bound for the maximum supply of a node. By substituting the delay minimization variables for the network flow variables we obtain an algorithm to solve the delay minimization problem, where $U = \mathrm{lcm}\{p(o)q(o) \mid o \in O\}$. With some clever techniques, see Ahuja et al. [1989], the right-hand-side scaling algorithm solves the minimum cost flow problem in $\mathcal{O}(m \log n(m + m \log n))$ time with $m = |A|$, which is currently the best known strongly polynomial-time algorithm.

5.3 Delay Assignment

The objective of the delay assignment is to expand the signal flow graph or to increase the delay assignment value such that the upper bounds in the relaxation of the precedences constraints are met. To minimize the resource requirements we expand the signal flow graph with delay lines.

5.3.1 Delay Lines

The purpose of a delay line is to implement the storage of a sample between its time of production and the time of its latest consumption with a minimum of resources. For each intermediate consumption we have to duplicate the sample. The duplicate of the sample is needed as input for the intermediate consumer. The sample itself is needed as input for subsequent consumers. We schedule the duplication of the sample as late as possible to store the duplicate as short as possible in order to minimize its memory requirement.

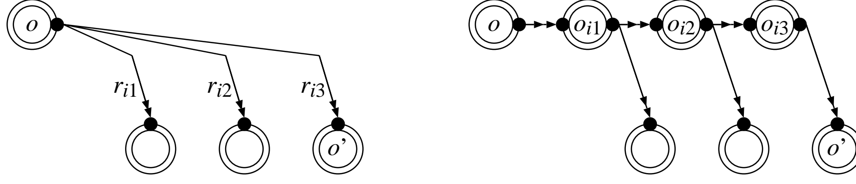


Figure 5.1. Example of a delay line that has been created by three consecutive expansions to implement the storage of samples in a way that allows intermediate consumptions. The periods of the operations and precedences are all equal.

Each data precedence relates a periodic production to a periodic consumption. The minimum storage time between production and consumption is equal to $\delta(r) = \max\{0, \sigma(o') - \sigma(o) - \overline{\Omega}(r)\}$ for each data precedence $r \in R$. We index the data precedences $R((o,n)[i]) = \{r_{i1}, \ldots, r_{im_i}\}$ that periodically consume the production $(o,n)[i]$ in such a way that $\delta(r_{i(j-1)}) \leq \delta(r_{ij})$ for all $1 < j \leq m_i$. The maximum storage time that has been computed during delay minimization is equal to $\delta(r_{im_i})$. We partition the maximum storage time into a sequence of storage times $\delta_1((o,n)[i]), \ldots, \delta_{m_i}((o,n)[i])$ with $\delta_1((o,n)[i]) = \delta(r_1)$ and $\delta_j((o,n)[i]) = \delta(r_j) - \delta(r_{j-1})$ for all $1 < j \leq m_i$. The sample $(o,n)[i]$ has to be duplicated for consumption by precedence r_{ij} when it has been stored for $\delta_j((o,n)[i])$ clock cycles. For notational convenience we denote the maximum storage time by $\delta((o,n)[i]) = \sum_{j=1}^{m_i} \delta_j((o,n)[i]) = \delta(r_{im_i})$. We implement these storage times by inserting a *delay line* which is a sequence of pass operations $o_{i1},\ldots,o_{im_i}$ as illustrated in Figure 5.1 for $m_i = 3$. Each operation o_{ij} has one incoming precedence $r'_{i(j-1)}$ from its predecessor $o_{i(j-1)}$ and one outgoing precedence r'_{ij} to its successor $o_{i(j+1)}$.

5.3.2 Resource Constraints

Each operation o_{ij} in a delay line increases the computation requirement because it needs to be executed every $p(o_{ij})$ clock cycles. Furthermore it increases the storage requirement because its associated delay $\delta(o_{ij})$ needs to be stored in a dual ported memory element. The operations of a delay line also increase the communication requirement. This increase is equal to the increase of the computation requirement because the number and period of the added precedences is equal to the number and period of the added operations. If we assume that each output terminal of each processing element is connected to at least one communication channel, then the communication workload is feasible if the computation workload is feasible. This is due to the fact that the communication requirement between two operation sets is equal to the number of consumed productions rather than the number of consumptions, i.e., different consumptions of the same production contribute only once.

To satisfy the relaxation of the connectivity constraints we have to assign the operations of a delay line to processing elements that we can reach from the producing processing element and from which we can reach the consuming processing element. Therefore we assign the operations of a delay line to processing element sets on the paths from producer to consumer, i.e., to processing element sets of the set $\mathrm{A}((o,n)[i])$ that is defined by

$$\mathrm{A}((o,n)[i]) = \{V \cap \bigcap_{\substack{r \in R((o,n)[i]) \\ \sigma(o') - \sigma(o) > \overline{\Omega}(r)}} \bigcup_{\substack{(p_1,\ldots,p_k) \in C(\mathrm{A}(o),\mathrm{A}(o')) \\ 1<j\leq k}} \{p_j\} \mid V \in \mathrm{A}(O)\},$$

thereby taking the relaxation of the connectivity constraints into account. Here $C(\mathrm{A}(o),\mathrm{A}(o'))$ is the set of all processing element disjoint paths from $\mathrm{A}(o)$ to $\mathrm{A}(o')$, i.e.,

$$C(\mathrm{A}(o),\mathrm{A}(o')) = \{(p_1,\ldots,p_k) \in \mathrm{A}(o) \times P^* \times \mathrm{A}(o') \mid ((p_1,n_1),\ldots,(p_k,n_k)) \in \hat{C}\},$$

and $\hat{C}$ is the set of all processing element disjoint paths given by

$$\hat{C} = \{((p_1,n_1),\ldots,(p_k,n_k)) \mid \forall_{1\leq i<k}((p_i,n_i),(p_{i+1},n_{i+1})) \in C \wedge \forall_{1\leq i<j\leq k} p_i \neq p_j\}.$$

Note that if we assign the operations of a delay line to suitable processing element sets of the set $\mathrm{A}((o,n)[i])$ and the substitute data precedences obeyed the relaxation of the connectivity constraints, then the substituted data precedences also obey the relaxation of the connectivity constraints. This is due to the fact that the switch matrices create a processing element network in which it is always possible to take an indirect route from producer to consumer. More precisely, if it holds that $((p,m),(p',m')) \in C$ then there exists a (p'',m'') such that it holds that $((p,m),(p'',m'')) \in C$ and $((p'',m''),(p',m')) \in C$. This property holds under the assumption that no input processor is directly connected to an output processor. As a result of this property the operations o_{ij} of a delay line can be assigned to the processing element set $\mathrm{A}(o_{ij})$ for which it holds that $p'' \in \mathrm{A}(o_{ij})$ without violating the relaxation of the connectivity constraints.

5.3.3 Precedence Constraints

We rewrite the relaxation of the precedence constraints concerning an incoming precedence $r'_{i(j-1)}$ and an outgoing precedence r'_{ij} of a delay line operation o_{ij} into

$$\sigma(o_{i(j-1)}) + \underline{\Omega}(r'_{i(j-1)}) \leq \sigma(o_{ij}) \leq \sigma(o_{i(j+1)}) - \underline{\Omega}(r'_{ij}),$$

in combination with

$$\sigma(o_{i(j-1)}) + \overline{\Omega}(r'_{i(j-1)}) \geq \sigma(o_{ij}) \geq \sigma(o_{i(j+1)}) - \overline{\Omega}(r'_{ij}).$$

Now a necessary and sufficient condition for the existence of $\sigma(o_{ij})$ is

$$\underline{\Omega}(r'_{i(j-1)}) + \underline{\Omega}(r'_{ij}) \leq \sigma(o_{i(j+1)}) - \sigma(o_{i(j-1)}) \leq \overline{\Omega}(r'_{i(j-1)}) + \overline{\Omega}(r'_{ij}).$$

This means that a delay line satisfies the relaxation of the precedence constraints if and only if for all $r_{im} = ((o,n),(o',n'),(p,b,b')) \in R((o,n)[i])$ it holds that

$$\sum_{j=0}^{m} \underline{\Omega}(r'_{ij}) \leq \sigma(o') - \sigma(o) \leq \sum_{j=0}^{m} \overline{\Omega}(r'_{ij}).$$

To satisfy these precedence constraints the decision variables δ, α, and λ can be chosen such that it holds that $\sum_{j=0}^{m} \overline{\Omega}(r'_{ij}) = \sum_{j=1}^{m} \delta_j((o,n)[i]) = \delta(r_{im})$ for all $r_{im} \in R((o,n)[i])$.

To decouple the precedence constraints from the computation and storage constraints we approximate the differences $\underline{\Omega}(r'_{ij})$ and $\overline{\Omega}(r'_{ij})$ between the completion times of two successive operations by $\underline{\Omega}((o,n)[i]) + \delta(o_{ij})p(o_{ij})$ and $\overline{\Omega}((o,n)[i]) + \delta(o_{ij})p(o_{ij})$, respectively, where

$$\underline{\Omega}((o,n)[i]) = \min\{\underline{\Omega}(r_{ij}) \mid r_{ij} \in R((o,n)[i])\} - \delta(o)p(o) \text{ and}$$

$$\overline{\Omega}((o,n)[i]) = \max\{\overline{\Omega}(r_{ij}) \mid r_{ij} \in R((o,n)[i])\} - \delta(o)p(o).$$

We now have that

$$\sum_{j=0}^{m} \underline{\Omega}(r'_{ij}) \approx (m+1)\underline{\Omega}((o,n)[i]) + \sum_{j=0}^{m} \delta(o_{ij})p(o_{ij}) \text{ and}$$

$$\sum_{j=0}^{m} \overline{\Omega}(r'_{ij}) \approx (m+1)\overline{\Omega}((o,n)[i]) + \sum_{j=0}^{m} \delta(o_{ij})p(o_{ij}).$$

Furthermore, we have that

$$\sigma(o') - \sigma(o) \approx \delta(r_{im}) + \overline{\Omega}((o,n)[i]) + \delta(o)p(o).$$

Substitution of these approximations into the relaxation of the precedence constraints gives

$$(m+1)\underline{\Omega}((o,n)[i]) - \overline{\Omega}((o,n)[i]) \leq \delta(r_{im}) - \sum_{j=1}^{m} \delta(o_{ij})p(o_{ij}) \leq m\overline{\Omega}((o,n)[i]),$$

which we can strengthen to

$$m\underline{\Omega}((o,n)[i]) \leq \delta(r_{im}) - \sum_{j=1}^{m} \delta(o_{ij})p(o_{ij}) \leq m\overline{\Omega}((o,n)[i]),$$

because it holds that $\underline{\Omega}((o,n)[i]) \leq \overline{\Omega}((o,n)[i])$. To obtain minimal resource requirements we choose the smallest feasible values for the delay assignment $\delta(o_{ij})$ such that it holds that $\sum_{j=1}^{m} \delta(o_{ij})p(o_{ij}) = \delta(r_{im}) - m\overline{\Omega}((o,n)[i])$. The resulting delay assignment values are $\delta(o_{ij}) = \max\{0, \lceil(\delta(r_{ij}) - j\overline{\Omega}((o,n)[i]))/p(o_{ij})\rceil\}$ for all $r_{ij} \in R((o,n)[i])$.

5.3.4 Processing Element Assignment

We formulate the problem of assigning operations to processing element sets for a given delay line and a given delay assignment in such a way that the computation and storage requirements are met as a bin packing problem. Each operation o_{ij} of a given delay line corresponds to an item that has two sizes $1/p(o_{ij})$ and $\delta(o_{ij})$. Each processing element set $\mathrm{A}(o_{ij}) \in \mathrm{A}((o,n)[i])$ corresponds to a bin that has two capacities $C_{\mathrm{A}}(\mathrm{A}(o_{ij}))$ and $C_{\Delta}(\mathrm{A}(o_{ij}))$. The processing elements already have an initial workload which corresponds with initial bin fillings $R_{\mathrm{A}}(\mathrm{A}(o_{ij}))$ and $R_{\Delta}(\mathrm{A}(o_{ij}))$ of the two capacities. The initial bin filling can be seen as the result of a partial assignment of items to bins. The problem is to assign each operation o_{ij} to a processing element set $\mathrm{A}(o_{ij})$ such that for all $V \in \mathrm{A}((o,n)[i])$ the relaxations of the computation and storage constraints are satisfied.

There are many heuristics known from the literature to handle bin packing problem such as those for instance proposed by Coffman et al. [1997]. Here we adopt the *first fit decreasing* algorithm which is known to be optimal if the item sizes form a divisible sequence. The first step of the algorithm is to sort the items in decreasing order. We do this in decreasing order of the delay assignment values $\delta(o_{ij})$ because all operations of a delay line have the same period. The second step is to assign the sorted operations iteratively to the first processing element set in which they fit and which satisfies the relaxation of the connectivity constraints.

5.3.5 Delay Line Refinement

A potential problem of the delay lines is that they may impose large computation or storage requirements on the dual ported memory elements because the delay assignment values are often nonzero. To avoid this potential problem we allow the possibility to implement each storage time $\delta_j((o,n)[i])$ with a sequence of pass operations $o_{ij1},\ldots,o_{ijm_{ij}}$. This enables us to decrease the value of the delay assignment at the expense of more pass operations thereby shifting the workload from dual ported memory elements to other processing elements. We reformulate the precedence constraint approximation now as

$$m_{ij}\underline{\Omega}((o,n)[i]) \leq \delta_j((o,n)[i]) - \sum_{k=1}^{m_{ij}} \delta(o_{ijk})p(o_{ijk}) \leq m_{ij}\overline{\Omega}((o,n)[i])$$

for all $0 \leq i < q(o)$ and for all $1 \leq j \leq m$. The variables m_{ij} and $\delta(o_{ijk})$ are the unknowns in these constraints. The variable m_{ij} determines the number of pass operations that is used to implement storage time $\delta_j((o,n)[i])$. The variables $\delta(o_{ijk})$ determine the delays of these pass operations.

To obtain minimal resource requirements we again choose the smallest feasible values for the delay assignment satisfying the equation $\sum_{k=1}^{m_{ij}} \delta(o_{ijk})p(o_{ijk}) = \delta_j((o,n)[i]) - m_{ij}\overline{\Omega}((o,n)[i])$. Furthermore we know that each additional opera-

tion o_{ijk} increases the computation requirement with $p(o_{ijk})^{-1}/C_{\mathrm{A}}(\mathrm{A}(o_{ijk}))$ and the storage requirement with $\delta(o_{ijk})/C_{\Delta}(\mathrm{A}(o_{ijk}))$. We adopt the heuristic to assign the total delay to a single pass operation rather than to partition the total delay over a number of pass operations because the latter increases the computation workload and does not decrease the storage workload. So we choose $\delta(o_{ij1}) = \max\{0, \lceil(\delta_j((o,n)[i]) - m_{ij}\overline{\Omega}((o,n)[i]))/(p(o)q(o))\rceil\}$ and $\delta(o_{ijk}) = 0$ for all $1 < j \leq m_{ij}$. We bound the value of the integer m_{ij} by $\lceil\delta_j((o,n)[i])/\overline{\Omega}((o,n)[i])\rceil$ because larger values increase the computation workload but do not decrease the storage workload. It is easily shown that the values in this range satisfy the precedence constraint approximation.

We now compute a set of potential delay lines for each storage time $\delta_j((o,n)[i])$ by computing delay assignment values for all integers m_{ij} in the range from 1 up to $\lceil\delta((o,n)[i])/\overline{\Omega}((o,n)[i])\rceil$. From this set of solutions we select a delay line $o_{ij1},\ldots,o_{ijm_{ij}}$ such that the maximum workload of the processing element sets $\mathrm{A}(o_{ij1}),\ldots,\mathrm{A}(o_{ijm_{ij}})$ is minimal. The first fit decreasing algorithm that we proposed to handle the processing element assignment is no longer suited because it does not strive to meet this objective. The algorithm is designed to minimize the number of bins by filling the bins as much as possible. Our objective is to minimize the maximum filling of the available bins. To meet this objective we propose to use another well-known heuristic bin packing algorithm called *worst fit decreasing*. Again the first step is to sort the items according to decreasing size. The second step is to assign the sorted items iteratively to the emptiest bin. We define emptiest as having the lowest workload, i.e., we assign each operation o_{ijk} to a bin $\mathrm{A}(o_{ijk})$ in such a way that the resulting workload

$$\max\left\{\frac{R_{\mathrm{A}}(O_{\mathrm{A}(o_{ijk})})}{C_{\mathrm{A}}(\mathrm{A}(o_{ijk}))}, \frac{R_{\Delta}(O_{\mathrm{A}(o_{ijk})})}{C_{\Delta}(\mathrm{A}(o_{ijk}))}\right\}$$

is minimal. The resource constraints are infeasible if this workload exceeds one.

5.3.6 Successive Delay Minimization

An important issue in the decomposition of the delay management problem is the fact that the delay assignment step is performed under the assumption that the time assignment is optimal with respect to the cost function of the delay minimization step. The following theorem shows that this assumption can be violated by an expansion of a signal flow graph with a delay line.

Theorem 5.5. *Let* $(\Delta,\Sigma,\mathrm{A},\Lambda)$ *be an instance of the delay minimization problem, let* $\sigma \in \Sigma$ *be an optimal solution with nonzero cost, let* $o \in O$ *be an operation, let* $0 \leq i < q(o)$ *be an integer, and let* $r_{ij} \in R((o,n)[i])$ *be a data precedence such that* $\delta_j((o,n)[i]) > 0$*. Furthermore, let* $(\Delta',\Sigma',\mathrm{A}',\Lambda')$ *be the mapping set of signal flow graph expansion* (O',R') *onto processing element network* (P,C) *after de-*

lay line expansion on $\delta_j((o,n)[i])$, and let $\sigma' \in \Sigma'$ be a time assignment such that $f(\sigma') < f(\sigma)$ and $\sigma'(o) = \sigma(o)$ for all $o \in O$. Then σ' may be a nonoptimal solution to the corresponding instance of the delay minimization problem.

Proof. Suppose we have a mapping set $(\mathrm{A},\Lambda,\Delta,\Sigma)$ of signal flow graph (O,R) onto processing element graph (P,C) with $O = \{o,o',o''\}$ and $R = \{r,r',r''\}$, where $r = ((o,0),(o',0),(1,b,0))$, $r' = ((o',0),(o'',0),(1,0,b'))$ and $r'' = (o'',0),(o,0),(1,c,c'))$. Furthermore, suppose we have an optimal solution $\sigma \in \Sigma$ to the corresponding instance of the delay minimization problem such that $\delta_1((o,0)[0]) > 0$, $\delta_1((o',0)[0]) > 0$, and $\delta_1((o'',0)[0]) = 0$. Then the cost of the time assignment σ is equal to $f(\sigma) = \delta_1((o,0)[0]) + \delta_1((o',0)[0])$. Delay line expansion on precedence r has two results. First the resulting time assignment σ' has cost $f(\sigma') = \delta_1((o',0)[0])$ because of the fact that $\sigma'(o) = \sigma(o)$ for all $o \in O$. Second the signal flow graph expansion contains data precedences r_{01k} satisfying $\sum_{k=1}^{m_{01}} \underline{\Omega}(r_{01k}) \leq \sigma(o') - \sigma(o) \leq \sum_{k=1}^{m_{01}} \overline{\Omega}(r_{01k})$. We now have that it is possible to increase the value of $\sigma(o')$ and to decrease the value of $f(\sigma') = \delta_1((o',0)[0])$ unless $\sigma(o') - \sigma(o)$ is equal to the upper bound $\sum_{k=1}^{m_{01}} \overline{\Omega}(r_{01k})$, □

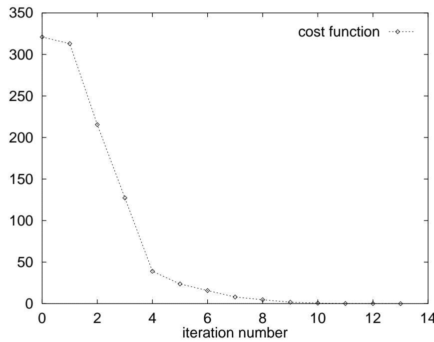
The violation of the above-mentioned assumption increases the resource requirements more than necessary. To prevent this we perform an additional delay minimization step after a delay line expansion. Each delay line expansion stores the output of an invocation i of terminal (o,n), with $o \in O$ and $0 \leq i < q(o)$, as input for the consumers of the data precedences $\mathrm{R}((o,n)[i])$. To avoid fragmentation of the total storage requirements into many small delays we handle the invocations according to decreasing storage requirements.

5.4 Results

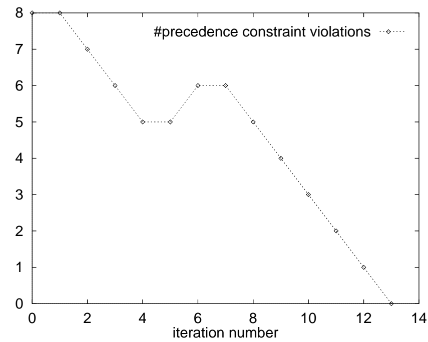
In this section we illustrate the operation of the proposed approach. Next we present the results of the approach on the set of industrially relevant problem instances presented in Chapter 2. Finally we discuss these results.

Figure 5.2 shows the operation of the proposed delay management approach. The value of the cost function that is minimized in the delay minimization step indeed decreases after each delay assignment step. However, the number of precedence constraint violations may increase in order to minimize the value of the cost function. For the instances shown in Figure 5.2 it holds that the number of iterations is larger than the initial number of precedence constraint violations. This shows that distribution of the total delay requirements over the signal flow graph can change after each delay assignment step.

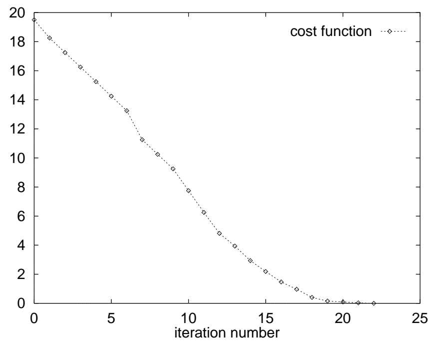
Table 5.1 shows the results of the presented delay management approach on the signal flow graphs that are mapped onto the VSP1FLEX processor network. Note that the positive value of the cost function shows that the relaxation of the precedence



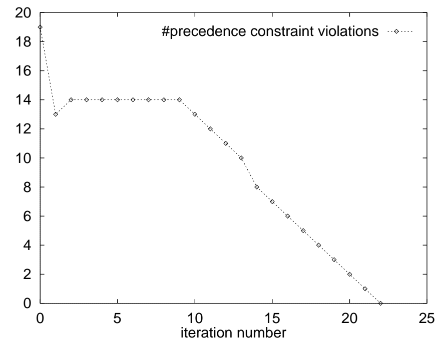
(a) CONTRAST.



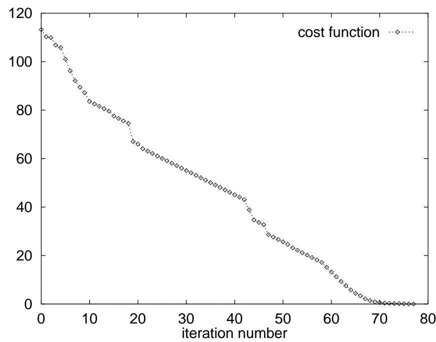
(b) CONTRAST.



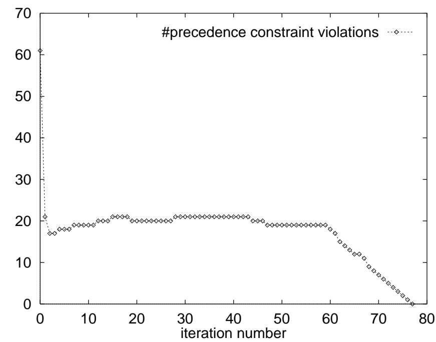
(c) HSRC.



(d) HSRC.



(e) VSRC.



(f) VSRC.

Figure 5.2. Delay management results for the signal flow graphs CONTRAST, HSRC, and VSRC that are mapped onto the VSP2FLEX processor network. The cost function $f(\sigma)$ and the number of precedence constraint violations are shown as a function of the number of iterations.

Table 5.1. Delay management results of the signal flow graphs that are mapped onto the VSP1FLEX processor network. From left to right the columns list the number of operations and data precedences, the computation requirement R_A and utilization $R_A/C_A \times 100\%$ per processing element type set, the storage requirement R_Δ and utilization $R_\Delta/C_\Delta \times 100\%$, the value of the cost function $f(\sigma)$, and the number of precedence constraint violations #.

SFG	$\|O\|$	$\|R\|$	R_A				R_Δ		$f(\sigma)$	#
			ALE		ALE, OE					
YUVTORGB	41	80	17.00	71	19.00	30	0	0	24.00	3
HORCOMPR	86	205	5.00	21	8.00	13	2048	25	1.94	17
IJNTEMA1	176	343	11.19	47	16.19	25	1580	19	3.69	31
CORMACK2	159	297	8.81	37	11.81	18	2564	31	2.63	28
CONTOUR1	83	191	17.44	73	19.44	30	3024	37	25.44	17
MONZA2	83	170	19.63	82	24.63	38	3440	42	31.00	7
VDP	165	396	16.25	68	18.44	29	3568	44	6.50	31
GAMMA	261	454	12.06	50	16.00	25	1264	15	74.25	15
HISTMOD2	198	376	9.00	38	12.06	19	957	12	3.00	19
PANORAMA	92	251	12.06	50	16.19	25	6144	75	5.94	2
VIDIWALL	167	424	18.69	78	24.25	38	4696	57	4.00	21
IJNTEMA2	277	557	20.56	86	24.31	38	2076	25	38.19	15
CORMACK1	202	460	21.81	91	22.81	36	6432	79	0.94	15
MONZA1	248	524	17.69	74	19.94	31	4411	54	2.13	9
MWTV	399	712	17.94	75	19.94	31	3424	42	9.56	33

constraints is violated initially in all problem instances. This indicates the need for delay management. Table 5.2 shows the results of the presented delay management approach on the signal flow graphs that are mapped onto the VSP2TEST processor network. Again the positive value of the cost function shows that the relaxation of the precedence constraints is violated initially in all problem instances. Furthermore, the values of the cost function in Tables 5.1 and 5.2 differ because the periods of the operations in the signal flow graph differ. Table 5.3 shows the results of the presented delay management approach on the signal flow graphs that are mapped onto the VSP2FLEX processor network. Once again the relaxation of the precedence constraints is violated initially in all problem instances.

Table 5.3 shows that the first delay minimization step of signal flow graph CONTRAST indicates that at least 321 memory locations are required to satisfy the precedence constraints. From the increase of 3.69 in computation requirements resulting in 118 silo locations and the increase of 278 in storage requirements compared to Table 4.4 we conclude that the delay management step has allocated a total of 396 memory locations to satisfy the precedence constraints. To this end 14

Table 5.2. Delay management results of the signal flow graphs that are mapped onto the VSP2TEST processor network. From left to right the columns list the number of operations and data precedences, the computation requirement R_A and utilization $R_A/C_A \times 100\%$ per processing element type set, the storage requirement R_Δ and utilization $R_\Delta/C_\Delta \times 100\%$, the value of the cost function $f(\sigma)$, and the number of precedence constraint violations #.

SFG	$\|O\|$	$\|R\|$	R_A						R_Δ		$f(\sigma)$	#
			ALE		ALE BE, OE		ME2					
YUVTORGB	41	80	8.50	71	9.50	40	0.00	0	0	0	11.50	3
HORCOMPR	86	205	2.50	21	3.56	15	0.50	13	2048	25	2.13	20
IJNTEMA1	184	358	5.53	46	7.84	33	0.91	23	1580	19	4.06	37
CORMACK2	163	301	4.41	37	5.97	25	0.72	18	2564	31	3.09	36
CONTOUR1	82	191	8.72	73	9.69	40	0.78	20	3024	37	14.41	18
MONZA2	79	166	9.81	82	11.56	48	1.38	34	3440	42	16.19	8
VDP	191	422	8.13	68	9.22	38	1.44	36	3568	44	4.78	33
GAMMA	243	436	6.03	50	7.13	30	1.09	27	1334	16	83.56	24
HISTMOD2	211	389	4.50	38	6.00	25	0.94	23	957	12	10.53	30
PANORAMA	97	256	6.03	50	8.25	34	2.25	56	6144	75	3.75	8
VIDIWALL	168	425	9.34	78	11.28	47	1.47	37	4696	57	4.41	27
IJNTEMA2	291	580	10.28	86	12.13	51	1.16	29	2084	25	30.06	27
MONZA1	261	537	8.84	74	10.19	42	1.25	31	4411	54	8.25	23
MWTV	399	712	8.97	75	9.97	42	1.75	44	3424	42	5.66	36

Table 5.3. Delay management results of the signal flow graphs that are mapped onto the VSP2FLEX processor network. From left to right the columns list the number of operations and data precedences, the computation requirement R_A and utilization $R_A/C_A \times 100\%$ per processing element type set, the storage requirement R_Δ and utilization $R_\Delta/C_\Delta \times 100\%$, the value of the cost function $f(\sigma)$, and the number of precedence constraint violations #.

SFG	$\|O\|$	$\|R\|$	R_A						R_Δ		$f(\sigma)$	#
			ALE		ALE BE, OE		ME2					
CONTRAST	291	515	40.06	56	43.75	31	4.25	18	13046	27	321.00	8
FDXD1	181	532	26.09	36	29.34	20	2.28	10	608	1	116.81	22
FDXD2	158	448	30.75	43	35.19	24	5.22	22	8800	18	114.78	15
MAT30UP	148	309	30.63	43	31.88	22	2.38	10	2795	6	15.54	5
HSRC	343	793	34.91	48	41.47	29	6.06	25	13694	28	19.50	19
VSRC	416	1057	32.56	45	43.09	30	7.03	29	10047	20	113.22	61

pass operations have been added to the signal flow graph; see Table 2.3. The same comparison for signal flow graph HSRC gives a delay minimization estimate of 20 locations versus a delay management allocation of 210 locations thereby extending the signal flow graph with 31 pass operations. For signal flow graph VSRC the delay minimization estimate is 113 locations and the delay management allocation is 450 locations using 131 additional pass operations. From these numbers we conclude that we are not able to minimize the number of memory locations. The main reason for this is the specific addressing scheme for writing data into a silo that cyclically writes all available locations.

Further inspection of the delay management results shows that the utilization of the different resources is well balanced. The difference between the average utilization of the arithmetic and logic element, buffer elements, and output elements, the average utilization of the memory elements, and the average utilization of the storage capacity decreases in many problem instances. From this we conclude that the delay assignment strategy effectively balances the workload over the resources.

5.5 Summary

In this chapter we have presented an approach to handle the delay management problem. The approach decomposes the delay management problem into the delay minimization problem and the delay assignment problem. The objective of the delay minimization problem is to compute a time assignment that minimizes the lifetimes of the operands. The objective of the delay assignment problem is to allocate room for each operand in one or more memories. We have shown that the delay minimization problem is a special case of the dual of the minimum cost flow problem which can be solved in polynomial time by network flow techniques. We have proposed a bin packing technique to handle the delay assignment problem. The results indicate that the presented approach effectively handles our instances of the delay management problem and that it effectively balances the workload over the resources.

6

Partitioning

In this chapter we present a solution strategy to handle the partitioning problem. Lengauer [1990] has given an overview of techniques that can be applied to partitioning problems. The solution strategy is based on hierarchical bipartitioning which decomposes the single multi-way partitioning problem into multiple two-way partitioning problems. Such an approach reduces the set of feasible solutions but it effectively handles the problem of routing the flow of data through the processor network.

The outline of this chapter is as follows. In Section 6.1 we decompose the partitioning problem into a processor assignment problem and a type assignment problem, thereby formulating the processor assignment problem as a sequence of two-way partitioning problems. In Sections 6.2 and 6.3 we present solution approaches to handle these subproblems. In Section 6.4 we present the results of the partitioning approach on the set of problem instances. Finally, in Section 6.5 we summarize the contents of this chapter.

6.1 Problem Decomposition

The goal of the partitioning problem is to find a signal flow graph expansion and a processing element assignment set that maps each pair of operations that is related by a data precedence onto a pair of processing elements that is interconnected

by a communication channel in such a way that the computation, storage, and communication constraints are satisfied.

Definition 6.1 (Partitioning Problem). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set of (O,R) onto (P,C) that satisfies the relaxations of the type, array, computation, communication, periodicity, and storage constraints. Find a signal flow graph (O',R') such that $(O,R) \subseteq (O',R')$ and find a mapping set $(\Delta',\Sigma',\mathrm{A}',\Lambda')$ of (O',R') onto (P,C) such that $|V(\mathrm{A}'(o))| = |T(\mathrm{A}'(o))| = 1$, for all $o \in O'$, the connectivity constraints, and the relaxations of the type, array, computation, communication, periodicity, and storage constraints are satisfied, if they exist. □

We decompose this multi-way partitioning problem into multiple two-way partitioning problems because in a two-way partition, unlike in a multi-way partition, the processing element assignment also specifies the channel assignment. This is due to the fact that the communication between the two subsets of a two-way partition can have only one route, whereas the communication between two subsets of a multi-way partition can have multiple routes via a third set of the partition.

Figure 6.1 shows an example of a processing element assignment set which maps a signal flow graph onto a network of three processors. We assume that operation o is mapped onto processor v_1 and that operation o' is mapped onto processor v_2. Without an expansion of the signal flow graph, data precedence r_1 has to be mapped onto interconnection i_1. With an expansion of the signal flow graph, data precedence r_1 can be replaced by two data precedences r_2 and r_3. Such an expansion introduces an additional pass operation o'' between operations o and o' which can be mapped onto processor v_2. The data precedences r_2 and r_3 can then be mapped onto the interconnections i_2 and i_3, respectively.

To route the flow of data we propose a solution strategy that tries to find a solution to the partitioning problem by constructing a sequence of two-way partitions of both the processor network and the signal flow graph. Each two-way partition in the processor network defines a cut of the interconnections and thus a connectivity constraint. The sequence of two-way partitions is chosen such that each interconnection is cut exactly once. Therefore, it represents the entire set of connectivity constraints. Formally, we represent a sequence of two-way partitions as a sequence of multi-way partitions as follows.

Definition 6.2 (Two-Way Partitioning Scheme). Let (P,C) be a processing element network. Then a sequence $\mathcal{P}_1,\ldots,\mathcal{P}_n$ is called a *two-way partitioning scheme* if and only if $\mathcal{P}_i$ is a partition of P into i subsets for all $1 \le i \le n$, $|\mathcal{P}_i \setminus \mathcal{P}_{i+1}| = 1$ and $|\mathcal{P}_{i+1} \setminus \mathcal{P}_i| = 2$ for all $1 \le i < n$, and $\mathcal{P}_n = \{\{(v,e) \in P \mid v = v'\} \mid (v',e') \in P\}$. □

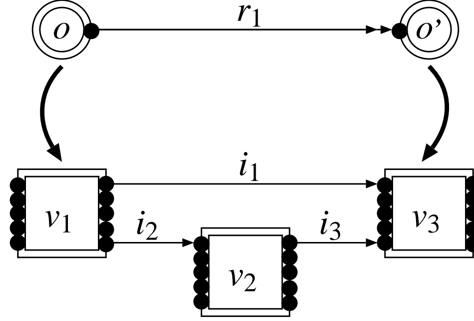


Figure 6.1. Example of a processing element assignment set that maps operation o onto processor v_1 and operation o' onto processor v_3. The communication between operations o and o' specified by data precedence r_1 can be routed via interconnection i_1 or via interconnections i_2 and i_3.

Each consecutive pair $(\mathcal{P}_i, \mathcal{P}_{i+1})$ in a two-way partitioning scheme encodes a two-way partition $\mathcal{P}_{i+1} \setminus \mathcal{P}_i$ of the processing element set $\mathcal{P}_i \setminus \mathcal{P}_{i+1}$. A potential two-way partitioning scheme in the example of Figure 6.1 is the sequence $\{P_{\{v_1,v_2,v_3\}}\}$, $\{P_{\{v_1\}}, P_{\{v_2,v_3\}}\}$, $\{P_{\{v_1\}}, P_{\{v_2\}}, P_{\{v_3\}}\}$, where $P_V = \{(v,e) \in P \mid v \in V\}$. Each two-way partition in the signal flow graph partitions the operations that are executed on the processing elements in the set $\mathcal{P}_i \setminus \mathcal{P}_{i+1}$ over the subsets in the two-way partition $\mathcal{P}_{i+1} \setminus \mathcal{P}_i$.

In order to model the stepwise refinement of the connectivity constraints we introduce an abstraction of processing element networks which extends the set of communication channels for a given partition of the processing elements.

Definition 6.3 (Processing Element Network Abstraction). Let (P,C) be a processing element network and let $\mathcal{P}$ be a partition of P. Then the pair $(\mathcal{P}, \mathcal{C})$ is called a *processing element network abstraction* where

$$\mathcal{C} = \{(p,p') \in P^2 \mid ((p,m),(p',m')) \in C \vee \exists_{\hat{P} \in \mathcal{P}} (p,p') \in \hat{P}^2\}. \qquad \square$$

A two-way partitioning scheme has an associated sequence of processing element network abstractions $(\mathcal{P}_1, \mathcal{C}_1), \ldots, (\mathcal{P}_n, \mathcal{C}_n)$. The last set $\mathcal{C}_n$ of communication channels contains only real communication channels that are in the set C. The other sets $\mathcal{C}_i$, $1 \le i < n$, of communication channels contains also fictive communication channels that are not in the set C. Our partitioning approach handles the communication and connectivity constraints by stepwisely removing the fictive communication channels. A mapping set $(\Delta, \Sigma, \mathrm{A}, \Lambda)$ of signal flow graph (O,R) onto processing element network (P,C) is called a *partition* of (O,R) onto a processing element network abstraction $(\mathcal{P}, \mathcal{C})$ if the processing element assignment set A

partitions the processing elements according to the partition $\mathcal{P}$ and obeys the computation and storage constraints as well as the communication constraints imposed by the communication channels of the set $\mathcal{C}$. Formally we define a partition as follows.

Definition 6.4 (Partition). Let $(\Delta, \Sigma, \mathrm{A}, \Lambda)$ be a mapping set of signal flow graph (O, R) onto processing element network (P, C) and let $(\mathcal{P}, \mathcal{C})$ be a processing element network abstraction of (P, C). Then the mapping set $(\mathrm{A}, \Lambda, \Delta, \Sigma)$ is called a *partition* of (O, R) onto $(\mathcal{P}, \mathcal{C})$ if and only if for all $o \in O$ there exists a $\hat{P} \in \mathcal{P}$ such that it holds that

$$\mathrm{A}(o) \subseteq \hat{P}.$$

A partition $(\mathrm{A}, \Lambda, \Delta, \Sigma)$ is called *feasible* if and only if it satisfies the relaxations of the type, array, computation, and storage constraints, and for all $r = ((o,n),(o',n'),(p,b,b')) \in R$ it holds that

$$\mathrm{A}(o) \times \mathrm{A}(o') \subseteq \mathcal{C},$$

and for all $\hat{P}, \hat{P}' \in \mathcal{P}$ it holds that

$$R_\Lambda((O_{\hat{P}}, O_{\hat{P}'})) \leq C_\Lambda((\hat{P}, \hat{P}')),$$

where $(O_{\hat{P}}, O_{\hat{P}'}) = \{r \in R \mid \mathrm{A}(o) \subseteq \hat{P} \wedge \mathrm{A}(o') \subseteq \hat{P}'\}$ represents the set of data precedences from operations in $O_{\hat{P}}$ to operations in $O_{\hat{P}'}$, and $(\hat{P}, \hat{P}') = \{((p,m),(p',m')) \in C \mid p \in \hat{P} \wedge p' \in \hat{P}'\}$ represents the set of communications channels from processing elements in $\hat{P}$ to processing elements in $\hat{P}'$. $\Box$

We decompose the partitioning problem into a processor assignment problem and a type assignment problem. The processor assignment problem decomposes the multi-way partitioning problem into a sequence of two-way partitioning problems. The resulting processing element assignment set ensures that $|V(\mathrm{A}(o))| = 1$ for all $o \in O$ because of the definition of $(\mathcal{P}_n, \mathcal{C}_n)$. The goal of the subsequent type assignment problem is to further partition the operations onto the different processing element types in order to ensure that $|T(\mathrm{A}(o))| = 1$ for all $o \in O$.

Definition 6.5 (Processor Assignment Problem). Let $(\Delta, \Sigma, \mathrm{A}, \Lambda)$ be a mapping set of signal flow graph (O, R) onto processing element network (P, C). Find a two-way partitioning scheme $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and feasible partitions $(\Delta_i, \Sigma_i, \mathrm{A}_i, \Lambda_i)$ of signal flow graph expansions (O_i, R_i) onto processing element network abstractions $(\mathcal{P}_i, \mathcal{C}_i)$ for all $1 \leq i \leq n$, if they exist. $\Box$

Definition 6.6 (Type Assignment Problem). Let $(\Delta, \Sigma, \mathrm{A}, \Lambda)$ be a partition of a signal flow graph (O, R) onto a processing element network abstraction $(\mathcal{P}, \mathcal{C})$ such that $|V(\mathrm{A}(o))| = 1$ for all $o \in O$. Find a partition $(\Delta', \Sigma', \mathrm{A}', \Lambda')$ of (O, R) onto $(\mathcal{P}, \mathcal{C})$ such that $|V(\mathrm{A}'(o))| = |T(\mathrm{A}'(o))| = 1$ for all $o \in O$, if one exists. $\Box$

6.2 Processor Assignment

An important issue in our partitioning approach is the construction of a two-way partitioning scheme, since this scheme determines the order in which the communication and connectivity constraints are handled.

6.2.1 Two-Way Partitioning Schemes

In each partitioning step we consider the communication and connectivity constraints due to one cut in the processor network under the assumption that the remaining communication and connectivity constraints can be handled in a subsequent step. If we arrive at the individual processors, then all communication and connectivity constraints are feasible because the processing elements of a single processor are fully interconnected by a switch matrix. Our goal is to implement as much of the communication as possible on these switch matrices because of their large communication bandwidth. Furthermore, we aim at two-way partitioning schemes in which the communication bandwidth between the sets of the two-way partitions increases with the consecutive partitioning steps in order to avoid communication bottlenecks in subsequent steps. We adopt the heuristic proposed by Jansen [1994] to meet these objectives.

We iteratively construct the two-way partitioning scheme in a bottom-up way as follows. Initially, we have the individual processors of the processor network which form partition $\mathcal{P}_n$. For each pair of processors, we define a ratio between the communication capacities and the computation capacities as follows.

Definition 6.7 (Capacity Ratio). Let (P,C) be a processing element graph and let $\mathcal{P}$ be a partition of P. Then the function $K : \mathcal{P} \times \mathcal{P} \to \mathbf{Q}$ defines the *capacity ratio* for all $\hat{P}, \hat{P}' \in \mathcal{P}$ by

$$K(\hat{P},\hat{P}') = \frac{(C_\Lambda(\hat{P},\hat{P}')+1)(C_\Lambda(\hat{P}',\hat{P})+1)}{C_\Lambda(\hat{P})+C_\Lambda(\hat{P}')}. \qquad \square$$

Subsequently, we group a pair of processors having the highest capacity ratio. This pair defines a two-way partition. Next, we consider this group to be a single node in the processor network from which we derive partition $\mathcal{P}_{n-1}$, and we repeat the above-mentioned procedure until the whole processor network is one node which represents partition $\mathcal{P}_1$.

6.2.2 Two-Way Partitioning Problem

We extend each two-way partition $\mathcal{P}_{i+1} \setminus \mathcal{P}_i$ encoded by a two-way partitioning scheme with adjacent producing processing elements to model the communication constraints of each partitioning step. The resulting two-way partition is called a *bipartite* processing element set.

Definition 6.8 (Bipartite Processing Element Set). Let (P,C) be a processing element network, let $\mathcal{P}_1,\ldots,\mathcal{P}_n$ be a two-way partitioning scheme, and let i be an integer such that $1 \leq i < n$. Then a pair $(P',\bar{P}')$ with $P' = \{p \in P \mid \exists_{p' \in P_i}(p,p') \in \mathcal{C}_{i+1}\}$, $\bar{P}' = \{p \in P \mid \exists_{p' \in \bar{P}_i}(p,p') \in \mathcal{C}_{i+1}\}$, and $\{P_i,\bar{P}_i\} = \mathcal{P}_{i+1} \setminus \mathcal{P}_i$, is called a *bipartite processing element set.* $\square$

A two-way partition of the operations that are executed on the processing elements of a bipartite processing element set is called a bipartite operation set. A bipartite operation set must satisfy a number of constraints in order to obtain a feasible partition. The problem of finding such a bipartite operation set is called the two-way partitioning problem. Once we have a feasible bipartite operation set, we update the processing element assignment set and, if necessary, insert additional pass operations to satisfy the connectivity constraint $\mathrm{A}(o) \times \mathrm{A}(o') \subseteq \mathcal{C}_{i+1}$.

Definition 6.9 (Bipartite Operation Set). Let (P,C) be a processing element network, let $\mathcal{P}_1,\ldots,\mathcal{P}_n$ be a two-way partitioning scheme, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a partition of a signal flow graph (O,R) onto a processing element network abstraction $(\mathcal{P}_i,\mathcal{C}_i)$ with $1 \leq i < n$, and let $(P',\bar{P}')$ be a bipartite processing element set. Then a pair $(O',\bar{O}')$ with $O',\bar{O}' \subseteq O$ such that $O' \cup \bar{O}' = \{o \in O \mid \mathrm{A}(o) \subseteq P' \cup \bar{P}'\}$ and $O' \cap \bar{O}' = \emptyset$ is called a *bipartite operation set* if and only if both $(\hat{O},\hat{P}) \in \{(O',P'),(\bar{O}',\bar{P}')\}$ satisfy the array constraints, i.e., for all $o,o' \in O$ satisfying $a(o) = a(o')$ it holds that $o \in \hat{O}$ if and only if $o' \in \hat{O}$, the storage constraints, i.e., it holds that

$$R_\Delta(\hat{O}) \leq C_\Delta(\hat{P}),$$

the computation constraints, i.e., it holds that

$$R_\mathrm{A}(\hat{O}) \leq C_\mathrm{A}(\hat{P}),$$

and the communication constraints, i.e., for all $V \in \mathcal{P}_i \cap \mathcal{P}_{i+1}$ it holds that

$$R_\Lambda((O_V,\hat{O})) \leq C_\Lambda((V,\hat{P})) \wedge R_\Lambda((\hat{O},O_V)) \leq C_\Lambda((\hat{P},V)).$$

A bipartite operation set is called *feasible* if and only if

$$R_\Lambda((O',\bar{O}')) \leq C_\Lambda((P',\bar{P}')) \wedge R_\Lambda((\bar{O}',O')) \leq C_\Lambda((\bar{P}',P')). \qquad \square$$

To illustrate the concept of bipartite operation sets we resort to the example of Figure 6.1. Assuming that we are given the partitioning scheme $\{P_{\{v_1,v_2,v_3\}}\}$, $\{P_{\{v_1\}},P_{\{v_2,v_3\}}\}$, $\{P_{\{v_1\}},P_{\{v_2\}},P_{\{v_3\}}\}$ we define two bipartite operation sets. We define the first bipartite operation set as $(\{o\},\{o'\})$ which assigns operation o to processor v_1 and operation o' to processors v_2 and v_3. We assume that operation o cannot be executed on an output element. In that case we have to expand the signal flow graph with a pass operation o_1 that can be executed on the first output element p_1 or fifth output element p_5 of processor v_1. These output elements are included

in the second bipartite processing element set $(P_{\{v_2\}} \cup \{p_5\}, P_{\{v_3\}} \cup \{p_1\})$, i.e., the first output element is added to $P_{\{v_3\}}$ and the fifth output element is added to $P_{\{v_2\}}$. If operation o' is executed on processor v_2, then the second bipartite operation set has to be defined as $(\{o_1, o'\}, \emptyset)$ in which case operation o_1 is executed on output element p_5. Alternatively, if operation o' is executed on processor v_3, then the second bipartite operation set can be defined either as $(\emptyset, \{o_1, o'\})$ in which case operation o_1 is executed on output element p_1, or as $(\{o_1\}, \{o'\})$ in which case operation o_1 is executed on output element p_5. In the latter case we have to expand the signal flow graph with a second pass operation o_2 that is executed on the first output element of processor v_2.

Definition 6.10 (Two-Way Partitioning Problem). Let (P,C) be a processing element network, let $\mathcal{P}_1, \ldots, \mathcal{P}_n$ be a two-way partitioning scheme, let i be an integer such that $1 \leq i < n$, let $(P', \bar{P}')$ be a bipartite processing element set, let (O,R) be a signal flow graph, and let $(\Delta, \Sigma, \mathrm{A}, \Lambda)$ be a partition of (O,R) onto $(\mathcal{P}_i, \mathcal{C}_i)$. Find a feasible bipartite operation set $(O', \bar{O}')$, if one exists. □

Theorem 6.1. *The two-way partitioning problem is NP-hard in the strong sense.*

Proof. For a given instance of the bipartitioning problem we can verify in polynomial time whether the bipartite operation set $(O', \bar{O}')$ is feasible. Hence the two-way partitioning problem is in $\mathcal{NP}$.

To prove that the two-way partitioning problem is NP-hard we use the following reduction from the minimum cut into bounded sets problem which is proved to be NP-hard in the strong sense by Garey et al. [1976]. Minimum cut into bounded sets can be defined as follows. Given are a graph (V,E), vertices $s,t \in V$, positive integer $B \leq |V|$, and positive integer K. Find a partition of V into disjoint subsets V_1 and V_2 such that $s \in V_1$, $t \in V_2$, $|V_1| \leq B$, $|V_2| \leq B$, and the number of edges with endpoints in V_1 and V_2 is no more than K.

Given an arbitrary instance of minimum cut into bounded sets as defined above, we construct a corresponding instance of the two-way partitioning problem such that a cut exists if and only if a bipartite operation set exists. To this end we define an operation o_v for all $v \in V$ and two precedences $r_{(v,v')}$ and $r_{(v',v)}$ for all edges $\{v, v'\} \in E$. The periods of the operations and precedences equal one. The operation o_s has type IO, the operation o_t has type OO, and the other operations have type ALO. Furthermore, we define a processing element p_v for all $v \in V$, and a bipartite processing element set $(P', \bar{P}')$ with $|P'| = |\bar{P}'| = B$, such that $t(p_s) = $ IN, $t(p_t) = $ OUT, and the other processing elements have type ALE. The sets P' and $\bar{P}'$ are chosen such that $p_s \in P'$ and $p_t \in \bar{P}'$. There are K connections from P to $\bar{P}'$, and K connections vice versa.

Suppose V_1 and V_2 define a feasible minimum cut. Then V_1 and V_2 define a bipartite operation set in which the operations of V_1 are executed on processing

elements of P', and the operations of V_2 are executed on processing elements of $\bar{P}'$. The computation constraints are feasible because operation o_s is in V_1, operation o_t is in V_2, and $|P'| = |\bar{P}'| = B$. The communication constraints are feasible, because the communication requirement from P' to $\bar{P}'$, or vice versa, is no more than K.

Suppose $(O', \bar{O}')$ is a feasible bipartite operation set. Then $V_1 = O'$ and $V_2 = \bar{O}'$ define a minimum cut. We have $s \in V_1$, $t \in V_2$, $|V_1| \leq B$, and $|V_2| \leq B$ because the computation constraints are feasible. Furthermore, the number of edges with endpoints in V_1 and V_2 is no more than K because the communication constraints are feasible. □

6.2.3 Local Search

We cannot expect to find a polynomial-time algorithm that solves the two-way partitioning problem, because it is NP-hard. The proof of this result shows the relation with graph partitioning problems for which Kernighan and Lin [1970] have developed an approximation algorithm that is called *variable-depth search*. Variable-depth search is a local search algorithm with a sophisticated neighborhood structure.

Local search algorithms are general approximation algorithms that are based on stepwise improvement of the value of a cost function by exploring neighborhoods. They start of with a given initial solution, which is often randomly chosen. Subsequently, they search the neighborhood for a better solution, i.e., a solution with lower cost. If such a solution is found, it becomes the current solution. Local search algorithms terminate if the neighborhood of the current solution does not contain better solutions. The main advantage of local search algorithms is that they are generally applicable and flexible since the specification of a search space, a cost function, and a neighborhood suffices, and a combinatorial optimization problem by definition inhibits the concepts of search space and cost function. For an overview of local search, the reader is referred to Aarts and Lenstra [1997]. Formally, a local search problem is defined as follows.

Definition 6.11 (Local Search Problem). A *local search problem* is a 3-tuple $(\mathcal{S}, f, \mathcal{N})$, where the *search space* $\mathcal{S}$ denotes the set of candidate solutions, the *cost function* f is a mapping defined as $f : \mathcal{S} \to \mathrm{I\!R}$, and the *neighborhood structure* $\mathcal{N}$ is a mapping defined as $\mathcal{N} : \mathcal{S} \to \mathcal{P}(\mathcal{S})$, which defines for each solution $i \in \mathcal{S}$ a set $\mathcal{N}(i) \subseteq \mathcal{S}$. The set $\mathcal{N}(i)$ is called the *neighborhood* of solution i, and each $j \in \mathcal{N}(i)$ is called a *neighbor* of i. The problem is to find a start solution $s \in \mathcal{S}$ and a sequence of neighbors $\mathcal{N}^0(s), \mathcal{N}^1(s), \mathcal{N}^2(s), \ldots, \mathcal{N}^n(s)$, where $\mathcal{N}^0(s) = s$ and $\mathcal{N}^n(s) = i^*$, that leads to a *globally* optimum solution $i^* \in \mathcal{S}$ such that $f(i^*) \leq f(i)$, for all $i \in \mathcal{S}$. A solution $\hat{\imath} \in \mathcal{S}$ is said to be a *locally* optimum solution with respect to $\mathcal{N}$ if $f(\hat{\imath}) \leq f(i)$, for all $i \in \mathcal{N}(\hat{\imath})$. □

Generally speaking, the search space $\mathcal{S}$ is not given explicitly. Usually, one resorts to the use of a compact representation from which each element in $\mathcal{S}$ can be computed using a polynomial-time algorithm.

Local search algorithms terminate in a local minimum unless the neighborhood is *exact*, i.e., each local optimum $\hat{\imath} \in \mathcal{S}$ is also a global optimum. However, exact neighborhoods are usually impractical since they lead to a complete enumeration of the search space. Therefore, one resorts to non-exact neighborhoods resulting in locally optimum solutions. There is generally no guarantee to the quality of a local optimum which often depends on the initial solution. A well-known neighborhood structure is the exchange neighborhood that performs very well for graph partitioning in combination with the technique called variable-depth search.

Definition 6.12 (Exchange). Let (A,B) be a bipartite set, and let $X \subseteq A$ and $Y \subseteq B$ be two subsets. Then, the *exchange* of X and Y results in the bipartite set $(A,B)_{(X,Y)}$ which is defined by $(A,B)_{(X,Y)} = ((A \setminus X) \cup Y, (B \setminus Y) \cup X)$. □

To limit the computation time one often uses a two-exchange neighborhood which bounds the cardinality of each of the exchanged subsets by one. The variable-depth search technique replaces a single two-exchange by a well-chosen sequence of two-exchanges using the gain of an exchange to guide the search. The length of such sequences may vary. The basic idea behind the variable-depth search exchange is to allow unfavorable two-exchanges in the sequence to eventually obtain a favorable k-exchange without exhaustive search of the k-exchange neighborhood.

The variable-depth search algorithm proceeds as follows. Each iteration starts with a given initial solution as current solution. The neighborhood of the current solution is searched for the best neighbor. This neighbor is accepted as current solution. When a neighbor is accepted, the exchanged operations are marked and cannot be exchanged again within the same iteration. Consequently, the length of the iterations is bounded since the size of the subsets of the bipartite set is bounded. After an iteration has been completed, a bipartite set with minimum cost is selected from the generated sequence of solutions. The algorithm terminates when the cost of the initial solution of the iteration has not improved. Otherwise, a new iteration is started using the best solution from the previous iteration as initial solution.

To analyze the complexity of local search problems, Johnson et al. [1988] have introduced the class $\mathcal{PLS}$ of polynomial-time local search problems which defines the class of local search algorithms for which local optimality can be verified in polynomial time. A combinatorial local search problem is in $\mathcal{PLS}$ if and only if for each instance there exist polynomial-time algorithms to determine a start solution, to determine the value of the cost function, and to determine local optimality or a neighbor with better cost. Furthermore, there exists the class $\mathcal{PLSC}$ of PLS-

complete problems which are the hardest ones in $\mathcal{PLS}$. The assumption is that for these problems instances exist that require superpolynomial time to find a locally optimal solution. In order to prove that a problem is PLS-complete one must show that it is in $\mathcal{PLS}$ and that there exists a PLS-reduction from a known PLS-complete problem. The notion of PLS-reducibility is defined in such a way that a PLS-reduction does not only map instances of one problem onto those of another problem, as is the case with reductions within the classical NP-completeness complexity class $\mathcal{NPC}$, but also maps the neighborhood structure of one problem onto that of another problem such that the topology of both structures with respect to local optimality is the same.

Definition 6.13 (Polynomial-Time Local Search Reduction). A problem $\pi \in \mathcal{PLS}$ is *PLS-reducible* to another problem $\pi' \in \mathcal{PLS}$ if and only if there exists a polynomial-time algorithm that maps solutions i of instances x of π onto solutions i' of instances x' of π' such that i is a local optimum for x if and only if i' is a local optimum for x'. □

6.2.4 Two-Way Partitioning

To handle the two-way partitioning problem, it is transformed into an optimization problem in which computation and storage constraints are treated as 'hard' constraints that must be satisfied by all solutions in the search space, whereas the communication constraints are treated as 'soft' constraints by taking them into account as a cost measure. This choice is based on the fact that it is easy to calculate solutions that satisfy the computation and storage constraints, but it is hard to take the communication constraints into account. The reason for this is that the computation requirements of an operation are fixed, while the communication requirements of an operation depend on the processing element assignment of its neighboring operations. The cost function is chosen such that low-cost solutions have a high probability of being feasible with respect to the communication constraints.

More specifically, for a given instance of the two-way partitioning problem we define the search space as the set of all bipartite operation sets. To explore this space we adopt a two-exchange neighborhood, which means that, except for arrays, we do not consider exchanges of sets containing more than one operation. We define the value of the cost function with a given bipartite operation set as a weighted sum of the external communication requirements between the two subsets and the internal communication requirements of both subsets to communicate the results of pass operations with zero delay. The reason to include the external communication requirements is to increase the probability that low-cost solutions are feasible. The reason to include the internal communication requirements is

to increase the probability that pass operations with zero delay that have been inserted during delay management are executed on output elements. Formally, the local search variant of the two-way partitioning problem is defined as follows.

Definition 6.14 (Local Search Variant of the Two-Way Partitioning Problem). Let (P,C) be a processing element network, let $\mathcal{P}_1,\dots,\mathcal{P}_n$ be a two-way partitioning scheme, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a partition of a signal flow graph (O,R) onto a processing element network abstraction $(\mathcal{P}_i,\mathcal{C}_i)$ with $1 \leq i < n$, and let $(P',\bar{P}')$ be a bipartite processing element set. Then, a *local search variant of the two-way partitioning problem* is a 3-tuple $(\mathcal{S},f,\mathcal{N})$, where the search space $\mathcal{S}$ represents the set of bipartite operation sets, i.e.,

$$\mathcal{S} = \{(O',\bar{O}') \mid (O',\bar{O}') \text{ is a bipartite operation set}\},$$

the cost function $f:\mathcal{S}\to\mathbf{Q}$ is defined by $f = f_{\mathrm{e}}+f_{\mathrm{i}}$, where $f_{\mathrm{e}}:\mathcal{S}\to\mathbf{Q}$ is an external communication cost function defined by

$$f_{\mathrm{e}}((O',\bar{O}')) = \alpha_{\mathrm{e}}R_{\Lambda}((O',\bar{O}')) + \beta_{\mathrm{e}}R_{\Lambda}((\bar{O}',O')),$$

and $f_{\mathrm{i}}:\mathcal{S}\to\mathbf{Q}$ is an internal communication cost function defined by

$$f_{\mathrm{i}}((O',\bar{O}')) = \alpha_{\mathrm{i}}R_{\Lambda}((O'_{\mathrm{PO}},O')) + \beta_{\mathrm{i}}R_{\Lambda}((\bar{O}'_{\mathrm{PO}},\bar{O}')),$$

where $O_{\mathrm{PO}} = \{o\in O \mid t(o) = \textsc{po} \wedge \delta(o) = 0\}$, and the neighborhood structure $\mathcal{N}:\mathcal{S}\to\mathcal{P}(\mathcal{S})$ is defined by

$$\mathcal{N}(O',\bar{O}') = \left\{ (O',\bar{O}')_{(X,Y)} \in \mathcal{S} \,\middle|\, \begin{array}{l} X\subseteq O' \wedge (|X|\leq 1 \vee |A(X)|=1)\wedge \\ Y\subseteq \bar{O}' \wedge (|Y|\leq 1 \vee |A(Y)|=1) \wedge X\cup Y \neq \emptyset \end{array} \right\},$$

with $A(X) = \{a(o)\in A \mid o\in X \wedge t(o)\in\{\textsc{ro},\textsc{wo}\}\}$ denoting the set of arrays that are accessed by operations in X. The weights of the cost functions are equal to the inverses of the corresponding external and internal communication capacities, i.e.,

$$\alpha_{\mathrm{e}} = \frac{1}{C_{\Lambda}((P',\bar{P}'))} \wedge \beta_{\mathrm{e}} = \frac{1}{C_{\Lambda}((\bar{P}',P'))} \wedge \alpha_{\mathrm{i}} = \frac{1}{C_{\Lambda}((P'_{\mathrm{PO}},P'))} \wedge \beta_{\mathrm{e}} = \frac{1}{C_{\Lambda}((\bar{P}_{\mathrm{PO}},\bar{P}'))},$$

where $P_{\mathrm{PO}} = \{p\in P \mid t(p)\in\{\textsc{ale},\textsc{be},\textsc{oe}\}\}$. The weights are equal to ∞ if the capacities equal zero. □

We use a reduction from the uniform graph partitioning problem to show that the local search variant of the two-way partitioning problem with the variable-depth search neighborhood is PLS-complete. The uniform graph partitioning problem can be defined as follows.

Definition 6.15 (Uniform Graph Partitioning). Let (V,E) be an undirected graph with $|V| = 2n$ vertices and cost $c(e)$ for all edges $e\in E$. Find a partition $V = A\cup B$ with $|A| = |B|$ such that $\sum_{\{a,b\}\in E, a\in A, b\in B} c(\{a,b\})$ is minimal, if one exists. □

Theorem 6.2. *The local search variant of the two-way partitioning problem is PLS-complete.*

Proof. In order to prove that the local search variant of the two-way partitioning problem is in $\mathcal{PLS}$, we have to show the existence of three polynomially computable functions. First, we have to compute an initial bipartite operation set. To this end, we formulate the problem as a bin packing problem with two bins representing two processing element sets which we solve by exhaustive search. The items and the bins have multi-dimensional sizes. Each dimension i corresponds to a resource type for which we have a capacity c_i. The vector $(c_1, \ldots, c_n)$ represents the multi-dimensional bin capacity. For each non-read and non-write operation and for each array we define an item vector consisting of zeros, except in the dimensions that correspond to the frequency, the delay, and the array size. Since the period and delay of the operations, and the size of the arrays are bounded, the number of different vectors is bounded and the problem is solvable in polynomial time. Second, the value of the cost function is clearly polynomially computable. Third, the number of neighboring solutions is bounded by $|O|^2 + |O|$, where O denotes the set of operations. Since the value of the cost function is polynomially computable in each neighbor, either a neighbor with better cost can be determined or local optimality can be detected in polynomial time. Hence, we conclude that the local search version of the two-way partitioning problem is in $\mathcal{PLS}$.

Furthermore, we prove that uniform graph partitioning with the variable-depth search neighborhood, which is known to be PLS-complete; see Johnson et al. [1988] or Schäffer and Yannakakis [1991], is a special case of the local search variant of the two-way partitioning problem. For simplicity we assume that both mechanisms to select one neighbor out of a set of neighbors which have equal cost are equivalent and deterministic. Let (V,E) be an undirected graph with $2n$ vertices, $n \in \mathbb{N}$, and let $c : E \to \mathbb{N}$ define the cost on each edge. The cost of a partition is computed by adding the cost of the edges of the cut. We define

$$m = \max\{ \sum_{v' \in V, v' \neq v} c(\{v, v'\}) \mid v \in V \},$$

and we construct an undirected graph (V,E') by replacing each edge $e \in E$ with $c(e)$ new edges. From this graph we construct a signal flow graph (O,R), where $O = V$, $t(o) = \text{ALO}$, and $p(o) = 1$ for all $o \in O$. The set of data precedences is constructed by replacing each edge $e \in E'$ with two data precedences which have opposite directions and period m. The numbers b and b' have to be chosen such that the signal flow graph is well-defined, and for all $r_1, r_2 \in R$, where $r_1 = ((o,n),(o'_1,n'_1),(p_1,b_1,b'_1))$ and $r_2 = ((o,n),(o'_2,n'_2),(p_2,b_2,b'_2))$, it holds that

$$b_1 \equiv b_2 \pmod{\gcd(\frac{p_1}{p(o)}, \frac{p_2}{p(o)})} \Rightarrow r_1 = r_2,$$

which is always possible because of the definition of m. We assume that signal flow graph (O,R) has to be partitioned onto an equally sized partition of a processor network consisting of $2n$ processors. Each processor contains precisely one arithmetic and logic element. For the weights of the cost function we choose $\alpha_{\mathrm{e}} = \beta_{\mathrm{e}} = 1$ and $\alpha_{\mathrm{i}} = \beta_{\mathrm{i}} = 0$. Clearly the solution spaces of the instances of uniform graph partitioning and the local search instance of the two-way partitioning problem are isomorphic and both cost functions are linearly related by factor $2m$, which completes the proof. □

An important issue in local search is the incremental update of the value of the cost function in order to evaluate the cost function for neighboring solutions. We demonstrate that our cost function can be computed incrementally by analyzing the effect of an exchange of one operation between the elements of a bipartite operation set on the value of the cost function. To this end we distinguish between four types of communication, namely external output, external input, internal output, and internal input; see Figure 6.2. With these types we associate four communica-

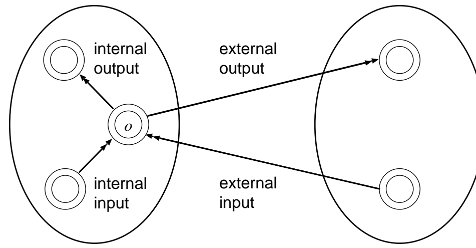


Figure 6.2. Classification of communication into external output, external input, internal output, and internal input.

tion costs. The cost of output is described by the communication requirements of Definition 4.4. To describe the cost of input we introduce a similar definition.

Definition 6.16 (Communication Requirements). Let R be a set of data precedences. Then the *output requirement* $R_\Lambda(R)$ is defined by

$$R_\Lambda(R) = \sum_{((o,n),(o',n'),(p,b,b'))\in R} \frac{|\{i \in \mathbb{N}_{q(o)} \mid b \equiv i \pmod{\frac{p}{p(o)}}\}}{p(o)q(o)},$$

and the *input requirement* $R'_\Lambda(R)$ is defined by

$$R'_\Lambda(R) = \sum_{((o,n),(o',n'),(p,b,b'))\in R} \frac{|\{i \in \mathbb{N}_{q(o')} \mid b' \equiv i \pmod{\frac{p}{p(o')}}\}}{p(o')q(o')}.$$ □

Using these communication requirements we incrementally update the cost after the move of an operation as shown in the following theorem.

Theorem 6.3. *Let $(O',\bar{O}')$ be a bipartite operation set. Then the change of the cost due to a move of an operation $o \in O'$ is given by $f((O',\bar{O}')_{(\{o\},\emptyset)}) - f(O',\bar{O}')$ and equals*

$$\alpha_e(R'_\Lambda((\{o\},O')) - R_\Lambda((\{o\},\bar{O}'))) + \beta_e(R_\Lambda((\{o\},O')) - R'_\Lambda((\{o\},\bar{O}'))) -$$
$$\alpha_i(R_\Lambda((\{o\}_{PO},O')) + R'_\Lambda((\{o\},O'_{PO}))) + \beta_i(R'_\Lambda((\{o\},\bar{O}'_{PO})) + R_\Lambda((\{o\}_{PO},\bar{O}'))).$$

The change of the cost due to a move of an operation $o \in \bar{O}'$ is given by $f((O',\bar{O}')_{(\emptyset,\{o\})}) - f(O',\bar{O}')$ and equals

$$\beta_e(R'_\Lambda((O',\{o\})) - R_\Lambda((\bar{O}',\{o\}))) + \alpha_e(R_\Lambda((O',\{o\})) - R'_\Lambda((\bar{O}',\{o\}))) -$$
$$\beta_i(R_\Lambda((O',\{o\}_{PO})) + R'_\Lambda((O'_{PO},\{o\}))) + \alpha_i(R'_\Lambda((\bar{O}'_{PO},\{o\})) + R_\Lambda((\bar{O}',\{o\}_{PO}))).$$

Proof. Suppose we move an operation $o \in O'$. Then the internal data flow becomes external which increases the cost with $\alpha_e R'_\Lambda((\{o\},O')) + \beta_e R_\Lambda((\{o\},O'))$ and decreases the cost with $\alpha_i(R_\Lambda((\{o\}_{PO},O')) + R'_\Lambda((\{o\},O'_{PO})))$. At the same time, the external data flow becomes internal which decreases the cost with $\alpha_e R_\Lambda((\{o\},\bar{O}')) + \beta_e R'_\Lambda((\{o\},\bar{O}'))$ and increases the cost with $\beta_i(R'_\Lambda((\{o\},\bar{O}'_{PO})) + R_\Lambda((\{o\}_{PO},\bar{O}')))$. The proof of a move of an operation $o \in \bar{O}'$ is similar. □

An arbitrary k-exchange can be written as a sequence of moves, so the total change of the communication cost of a k-exchange can be computed via the cost changes that correspond to these moves.

6.2.5 Load Balancing

A potential drawback of the proposed partitioning approach is that partitions with minimum cost may be infeasible. We illustrate this in the following theorem.

Theorem 6.4. *Bipartite operation sets with minimum cost may be infeasible.*

Proof. Consider the processor network and the signal flow graph of Figure 6.3. Each processor has three arithmetic and logic elements and all operations and data precedences have period one. Hence, the weights of the external communication cost function are $\alpha_e = \frac{1}{2}$ and $\beta_e = 1$. The weights of the internal communication cost function are irrelevant since there are no pass operations. The solution that is indicated by the matching of the fill patterns of the operations and processors has cost $2\alpha_e + \beta_e = 2$ and is feasible. The solution that is indicated by the dashed line has cost $3\alpha_e = 1\frac{1}{2}$. The latter solution is infeasible, although it has minimum cost, because the communication capacity from the video signal processor on the left to the video signal processor on the right is exceeded. □

In addition there may be an imbalance between the workloads of both subsets in the resulting two-way partition which makes the subsequent two-way partition-

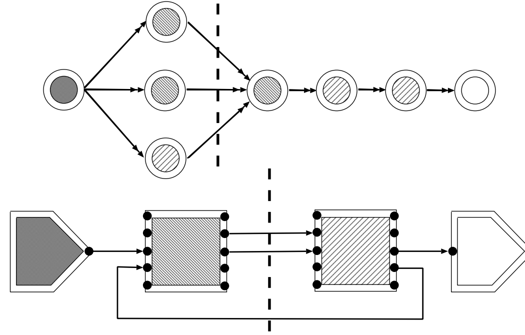


Figure 6.3. Two two-way partitions of a signal flow graph and a two-way partition of a processor network. The feasible solution indicated by the matching of the fill patterns of the operations and processors has higher cost than the infeasible solution indicated by the dashed line.

ing steps more difficult. To handle these potential problems we use the fact that local search algorithms usually generate many feasible solutions during the optimization process. From this set of feasible solutions we may select any suitable solution rather than the locally optimum solution. A potential selection criterion is to minimize the maximum utilization of any resource type as we did during delay management in order to balance the workload, i.e., to select the bipartite operation set $(O', \bar{O}')$ mapped onto bipartite processing element set $(P', \bar{P}')$ such that

$$\begin{aligned} g((O',\bar{O}')) = & \sum_{V \in \Lambda(O)} \max\left\{\frac{R_\Lambda(O_V,O')}{C_\Lambda(V,P')}, \frac{R_\Lambda(O_V,\bar{O}')}{C_\Lambda(V,\bar{P}')}\right\} + \\ & \sum_{V \in \Lambda(O)} \max\left\{\frac{R_\Lambda(O',O_V)}{C_\Lambda(P',V)}, \frac{R_\Lambda(\bar{O}',O_V)}{C_\Lambda(\bar{P}',V)}\right\} + \\ & \sum_{T \in T(\Lambda(O))} \max\left\{\frac{R_\Lambda(O'_T)}{C_\Lambda(P'_T)}, \frac{R_\Lambda(\bar{O}'_T)}{C_\Lambda(\bar{P}'_T)}\right\} + \max\left\{\frac{R_\Delta(O')}{C_\Delta(P')}, \frac{R_\Delta(\bar{O}')}{C_\Delta(\bar{P}')}\right\} \end{aligned}$$

is minimal. Note that these balance cost are not part of the cost function used in the local search algorithm because they do not guide the search to feasible solutions. Furthermore, an exchange may have a large effect on the balance cost because of the normalization of the requirements with the capacities. The disadvantage of the above-mentioned selection criterion is that it does not consider the communication workload between the two subsets. For this reason we propose to use a weighted sum of both the communication cost function and the balance cost function, i.e., to select the bipartite operation set $(O', \bar{O}')$ such that

$$h((O',\bar{O}')) = \varepsilon f((O',\bar{O}')) + (1-\varepsilon) g((O',\bar{O}'))$$

is minimal, where the value of ε is chosen such that $0 \leq \varepsilon \leq 1$. We use a default ε value of 0.5, but the value can be changed in order to balance the computation workload at the expense of higher communication workloads and vice versa.

6.3 Type Assignment

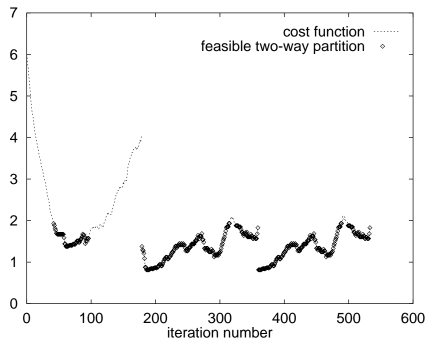
The relaxed type constraints allow the mapping of operations onto multiple processing element types. The goal of the type assignment problem is to assign each operation to one processing element type thereby obeying the type constraints. The motivation for this goal is to fix the time shapes of the operations such that the access times of the operation terminals are known if the completion times of the operations are known. In the processor assignment step the signal flow graph has been expanded with pass operations which are to be executed on output elements in order to satisfy the communication constraints. To handle the type assignment problem for the other processing element types we again resort to bin packing.

Each of the candidate processing element types corresponds to a bin. The bin capacity corresponds to the number of available processing elements of that type. Each operation can be assigned to at least one but possibly more bins. For instance, operations that can be executed on buffer elements can also be executed on arithmetic and logic elements. The order in which the operations are assigned depends on the number of candidate processing element types, i.e., operations are assigned later if the number of candidate processing element types is larger. To minimize the maximum workload of the processing element types we apply the worst-fit heuristic which assigns each operation to the currently emptiest bin. The combination of the ordering and the heuristic ensures that the arithmetic and logic operations are assigned to arithmetic and logic elements. Subsequently the constant, shift, and remaining pass operations with zero delay are assigned to buffer elements until their workload equals that of the arithmetic and logic elements. Thereafter they are equally distributed among both processing element types.

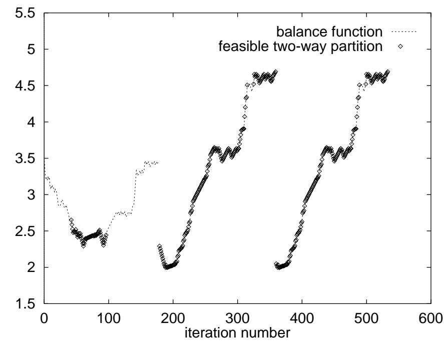
6.4 Results

In this section we illustrate the operation of the proposed approach. Next we present the results of the approach on the set of industrially relevant problem instances presented in Chapter 2. Finally we discuss these results.

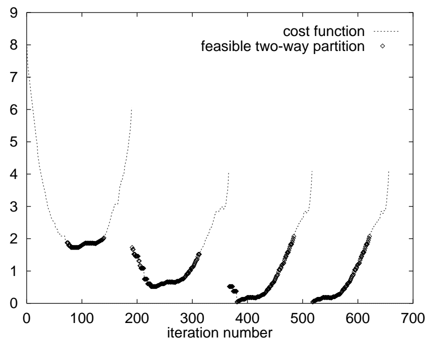
Figure 6.4 shows the behavior of the variable-depth search algorithm during the first two-way partitioning of the signal flow graphs CONTRAST, HSRC, and VSRC. The figure shows that the value of the cost function and the feasibility of the generated solutions are strongly related which indicates the effectiveness of the partitioning strategy. It also shows that many feasible solutions are generated from which a balanced solution can be selected. Figure 6.4b shows that the balance cost are



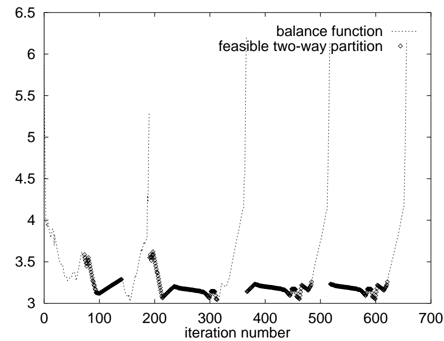
(a) CONTRAST.



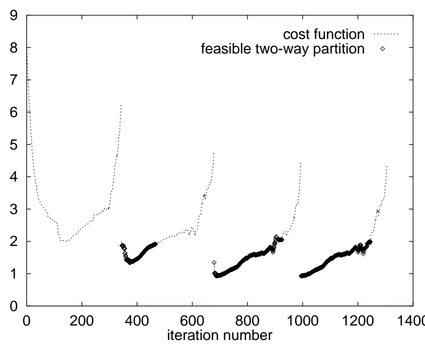
(b) CONTRAST.



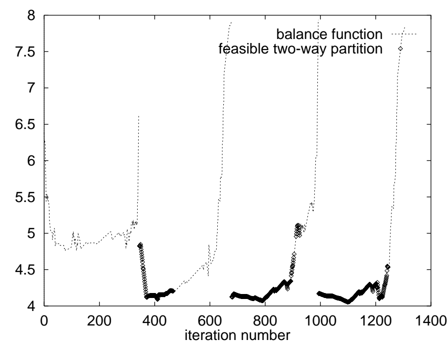
(c) HSRC.



(d) HSRC.



(e) VSRC.



(f) VSRC.

Figure 6.4. Two-way partitioning results for the signal flow graphs CONTRAST, HSRC, and VSRC that are mapped onto the VSP2FLEX processor network. The cost function and the balance function are shown as a function of the number of iterations.

Table 6.1. Partitioning results of signal flow graphs that are mapped onto the VSP1FLEX processor network. From left to right the columns list the number of operations and data precedences, the computation requirement R_A and utilization $R_A/C_A \times 100\%$ per processing element type set, and the storage requirement R_Δ and utilization $R_\Delta/C_\Delta \times 100\%$.

SFG	$\|O\|$	$\|R\|$	R_A				R_Δ	
			ALE		ALE, OE			
YUVTORGB	73	112	17.00	71	35.00	55	0	0
HORCOMPR	154	273	5.00	21	12.88	20	2048	25
IJNTEMA1	257	426	11.19	47	26.19	41	1580	19
CORMACK2	230	368	8.81	37	19.00	30	2564	31
CONTOUR1	131	240	17.44	73	33.31	52	3024	37
MONZA2	131	224	19.63	82	39.13	61	3440	42
VDP	282	516	16.25	68	37.13	58	3568	44
GAMMA	364	557	12.06	50	30.23	47	1264	15
HISTMOD2	328	506	9.00	38	24.69	39	957	12
PANORAMA	160	319	12.06	50	34.38	54	6144	75
VIDIWALL	270	548	18.69	78	43.56	68	4696	57
IJNTEMA2	485	795	20.56	86	41.81	65	2076	25
CORMACK1	334	601	21.81	91	47.15	74	6432	79
MONZA1	380	656	17.69	74	33.31	52	4411	54
MWTV	600	913	17.94	75	32.50	51	3424	42

not suited to guide the search to feasible solutions since there are many infeasible solutions that have lower balance cost than many feasible solutions.

Table 6.1 lists the results of partitioning of the signal flow graphs onto the VSP1FLEX processor network. The relative large increase of computation requirements shows that this processor network has a large communication overhead which is to be expected since each processor contains only three arithmetic and logic units. For signal flow graph PANORAMA the computation requirement of the arithmetic and logic element, the buffer element, and the output elements has increased from 16.19, see Table 5.1, to 34.38. From these number we conclude that 45% of the capacity of the output elements is required. To this end 63 pass operations have been added to the signal flow graph. Table 6.2 lists the results of partitioning of the signal flow graphs onto the VSP2TEST processor network. The communication overhead in this processor network is much smaller than that of the VSP1FLEX processor network. For signal flow graph PANORAMA the computation requirement of the arithmetic and logic element, the buffer element, and the output elements has increased from 8.25, see Table 5.3, to 8.75 using only 1 pass operation. Table 6.3 lists the results of partitioning of the signal flow graphs onto

Table 6.2. Partitioning results of signal flow graphs that are mapped onto the VSP2TEST processor network. From left to right the columns list the number of operations and data precedences, the computation requirement R_{A} and utilization $R_{\mathrm{A}}/C_{\mathrm{A}} \times 100\%$ per processing element type set, and the storage requirement R_{Δ} and utilization $R_{\Delta}/C_{\Delta} \times 100\%$.

SFG	$\lvert O \rvert$	$\lvert R \rvert$	R_{A} ALE		R_{A} ALE BE, OE		R_{A} ME2		R_{Δ}	
YUVTORGB	42	81	8.50	71	9.75	41	0.00	0	0	0
HORCOMPR	89	208	2.50	21	4.09	17	0.50	13	2048	25
IJNTEMA1	189	363	5.53	46	8.50	35	0.91	23	1580	19
CORMACK2	168	306	4.41	37	6.72	28	0.72	18	2564	31
CONTOUR1	86	194	8.72	73	10.22	43	0.78	20	3024	37
MONZA2	85	175	9.81	82	12.81	53	1.38	35	3440	42
VDP	195	426	8.13	68	10.06	42	1.44	36	3568	44
GAMMA	253	446	6.03	50	8.19	20	1.09	27	1333	16
HISTMOD2	214	392	4.50	38	6.56	27	0.94	24	957	12
PANORAMA	98	257	6.03	50	8.75	36	2.25	56	6144	75
VIDIWALL	175	432	9.34	78	12.81	54	1.47	37	4696	57
IJNTEMA2	309	598	10.28	86	13.19	55	1.16	29	2084	25
MONZA1	265	541	8.84	74	11.09	46	1.25	31	4411	54
MWTV	418	731	8.97	75	10.81	45	1.75	44	3424	42

Table 6.3. Partitioning results of signal flow graphs that are mapped onto the VSP2FLEX processor network. From left to right the columns list the number of operations and data precedences, the computation requirement R_{A} and utilization $R_{\mathrm{A}}/C_{\mathrm{A}} \times 100\%$ per processing element type set, and the storage requirement R_{Δ} and utilization $R_{\Delta}/C_{\Delta} \times 100\%$.

SFG	$\lvert O \rvert$	$\lvert R \rvert$	R_{A} ALE		R_{A} ALE BE, OE		R_{A} ME2		R_{Δ}	
CONTRAST	355	583	40.06	56	55.28	38	4.25	18	13046	27
FDXD1	230	581	26.09	36	43.63	30	2.28	10	608	1
FDXD2	264	572	30.75	43	49.41	34	5.22	22	8800	18
MAT30UP	258	454	30.63	43	43.08	30	2.38	10	2795	6
HSRC	398	848	34.90	48	52.16	36	6.06	25	13694	28
VSRC	558	1202	32.56	45	59.19	41	7.03	29	10047	20

the VSP2FLEX processor network. For signal flow graph CONTRAST the computation requirement of the arithmetic and logic element, the buffer elements, and the output elements the computation requirement increases from 43.75 to 55.28. From this we conclude that 33% of the capacity of the output elements is required.

Table 6.4. Partitioning results of signal flow graphs that are mapped onto the VSP2FLEX processor network. From left to right the columns list the minimum, average, and maximum computation and storage utilization of the processors.

(a) Two-way partition selection with $\varepsilon = 0.5$.

SFG	$R_A/C_A \times 100\%$									$R_\Delta/C_\Delta \times 100\%$		
	ALE			OE			ME2					
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
CONTRAST	4	57	92	21	33	44	13	18	25	3	33	56
FDXD1	0	37	79	27	45	69	0	10	30	0	1	6
FDXD2	15	45	75	27	43	59	13	22	27	0	18	50
MAT30UP	11	43	83	10	32	51	0	10	23	0	6	16
HSRC	0	50	100	0	35	63	0	25	63	0	28	85
VSRC	29	50	67	40	57	81	1	29	76	0	21	63

(b) Two-way partition selection with $\varepsilon = 0$.

SFG	$R_A/C_A \times 100\%$									$R_\Delta/C_\Delta \times 100\%$		
	ALE			OE			ME2					
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
CONTRAST	13	57	100	23	37	56	0	18	44	0	41	119
FDXD1	2	37	96	29	44	60	0	10	26	0	1	6
FDXD2	0	45	98	0	31	52	0	22	75	0	19	50
MAT30UP	0	43	94	13	23	39	0	10	25	0	6	19
HSRC	0	50	100	0	40	67	0	25	81	0	28	82
VSRC	6	50	94	27	39	50	0	29	97	0	21	74

(c) Two-way partition selection with $\varepsilon = 1$.

SFG	$R_A/C_A \times 100\%$									$R_\Delta/C_\Delta \times 100\%$		
	ALE			OE			ME2					
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
CONTRAST	47	57	66	25	40	62	13	18	25	2	33	56
FDXD1	9	37	83	43	64	90	0	10	30	0	1	6
FDXD2	19	45	75	50	65	86	13	22	39	2	18	52
MAT30UP	33	44	50	14	44	65	0	10	17	0	6	14
HSRC	29	53	98	58	69	91	0	25	63	0	28	85
VSRC	34	51	71	56	70	91	2	29	78	0	21	63

Table 6.4 lists the minimum, average, and maximum resource utilization of the processors after partitioning of the signal flow graphs onto the VSP2FLEX processor network with different values of the two-way partition selection parameter ε. This parameter controls the selection of a two-way partition after termination of the local search algorithm. Low values correspond to minimizing the communication along

the channels in the processor network. High values correspond to balancing the computation over the processors in the network. The table shows that the parameter can indeed be used to make this tradeoff in the partitioning step.

6.5 Summary

In this chapter we have presented an approach to handle the partitioning problem. This approach decomposes the single multi-way partitioning problem into a sequence of two-way partitioning problems. The advantage of this approach is that it stepwisely refines the assignment of operations to processors and the routing of the flow of data through the processor network. We have developed a local search heuristic to handle the two-way partitioning problem. The results indicate that the presented approach is able to produce balanced partitions in which the degree of utilization of the resources is evenly distributed among the processors in the network.

7

Scheduling

In this chapter we present a solution approach to handle the scheduling problem. The approach is based on a decomposition of the scheduling problem into a time assignment problem, a processing element assignment problem, and a channel assignment problem. The objective of the time assignment problem is to determine when the operations are executed. The objective of the processing element problem is to determine on which processing elements the operations are executed. The objective of the channel assignment problem is to determine along which channels the samples are communicated.

The outline of this chapter is as follows. In Section 7.1 we decompose the scheduling problem into the time assignment problem, processing element assignment problem, and channel assignment problem. In Sections 7.2, 7.3, and 7.4 we present solution approaches to handle these subproblems. In Section 7.5 we present the results of the scheduling approach on the set of problem instances. Finally, in Section 7.6 we summarize the contents of this chapter.

7.1 Problem Decomposition

In Chapter 4 we have formulated the scheduling problem as follows.

Definition 7.1 (Scheduling). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set such that $|\Delta| =$

$|V(\mathrm{A}(o))| = |T(\mathrm{A}(o))| = 1$ for all $o \in O$ and the relaxation of the mapping constraints is satisfied. Find a mapping $(\delta, \sigma, \alpha, \lambda) \in (\Delta, \Sigma, \mathrm{A}, \Lambda)$ such that the mapping constraints are satisfied, if one exists. □

The objective of the scheduling problem is to construct feasible time, processing element, and channel assignments assuming that a feasible delay assignment has already been constructed in the delay management step.

We propose a decomposition strategy that first constructs a time assignment, then constructs a processing element assignment, and finally constructs a channel assignment. This decomposition is motivated by the fact that the problem of constructing a processing element assignment for a given time assignment can be modeled as a graph coloring problem.

Definition 7.2 (Graph Coloring Problem). Given are a graph (V,E) and a positive integer K. Find a function $f : V \to \mathbb{N}_K$ such that $f(v) \neq f(v')$ for all $\{v, v'\} \in E$, if one exists.

Each operation corresponds to a vertex and each computation constraint concerning two operations corresponds to an edge between the two corresponding vertices. Furthermore, each processing element corresponds to a color. There exists a feasible processing element assignment if and only if there exists a feasible graph coloring. Also the problem of constructing a channel element assignment for given time and processing element assignments can be modeled as a graph coloring problem. Graph coloring is a well-known problem in the literature. We refer to Gibbons [1989] for an overview on the subject. Karp [1972] has shown that the problem is NP-complete in the strong sense for three or more colors. The literature presents many heuristics to handle graph coloring problems. These heuristics are often tuned to the characteristics of the graph instances. For video signal processor scheduling Essink et al. [1991a], Essink et al. [1991b], and Van Dongen [1991] have reported good results with a recoloring technique based on the *kempe-chain argument* [Kempe, 1879] [Gibbons, 1989]. Korst [1992] reports good results for circular-arc graph coloring with *sequential coloring* techniques [Welsh and Powell, 1967].

Another motivation for the above-mentioned decomposition is that in the scheduling problem the computation and communication constraints depend on the time assignment as well as on the processing element assignment and the channel assignment. The precedence constraints on the contrary only depend on the time assignment and not on the processing element assignment and the channel assignment. Hence, an approach which constructs a time assignment for a given processing element assignment, as is suggested in Theorem 3.4, is more difficult than an approach which constructs a processing element assignment for a given time

assignment as is suggested in Theorem 3.6. This is due to the fact that in the former approach the computation, communication, and precedence constraints need to be considered at the same time, whereas in the latter approach the computation and communication constraints can be considered under the assumption that the precedence constraints have already been satisfied. In the former approach this assumption is not realistic because the computation, communication, and precedence constraints depend on the time assignment. For these reasons, we decompose the scheduling problem into three subproblem which are called the time assignment problem, the processing element assignment problem, and the channel assignment problem. These subproblems are formally defined as follows.

Definition 7.3 (Time Assignment Problem). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set such that $|\Delta| = |V(\mathrm{A}(o))| = |T(\mathrm{A}(o))| = 1$ for all $o \in O$, the connectivity constraints, and the relaxation of the mapping constraints are satisfied. Find a time assignment $\sigma \in \Sigma$ such that the precedence constraints are satisfied, if one exists. □

Definition 7.4 (Processing Element Assignment Problem). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set such that $|\Delta| = |V(\mathrm{A}(o))| = |T(\mathrm{A}(o))| = 1$ for all $o \in O$, the connectivity constraints, and the relaxation of the mapping constraints are satisfied, and let $\sigma \in \Sigma$ be a time assignment such that the precedence constraints are satisfied. Find a processing element assignment $\alpha \in \mathrm{A}$, if one exists, such that the computation constraints are satisfied, i.e., for all $o,o' \in O$ it holds that if $o \neq o'$ and $\sigma(o) \equiv \sigma(o') (\bmod \gcd(p(o),p(o')))$ then $\alpha(o) \neq \alpha(o')$, the periodicity constraints are satisfied, and the storage constraints are satisfied. □

Definition 7.5 (Channel Assignment Problem). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set such that $|\Delta| = |V(\mathrm{A}(o))| = |T(\mathrm{A}(o))| = 1$ for all $o \in O$, the connectivity constraints, and the relaxation of the mapping constraints are satisfied, let $\sigma \in \Sigma$ be a time assignment such that the precedence constraints are satisfied, and let $\alpha \in \mathrm{A}$ be processing element assignment such that the periodicity, storage constraints, and computation constraints are satisfied. Find a channel assignment $\lambda \in \Lambda$, if one exists, such that the communication constraints are satisfied, i.e., for all data precedences $r,r' \in R$ it holds that if $r \neq r'$ and $\psi(r) \equiv \psi(r') (\bmod \gcd(p(r),p(r')))$ then $\lambda(r) \neq \lambda(r')$. □

A possible approach to the scheduling problem is to handle these subproblems consecutively. The problem of this approach is that the time assignment has to be

chosen such that there exists a feasible processing element assignment. Furthermore, the processing element assignment has to be chosen such that there exists a feasible channel assignment. A more fine-grained approach which overcomes this problem is to iteratively consider one additional operation or precedence while handling the subproblems consecutively. In this fine-grained approach we assign a completion time to one operation. Subsequently we try to find processing element and channel assignments given this partial time assignment. If this succeeds then we assign a completion time to the next operation, otherwise we assign another completion time to the current operation. We formally describe this approach by a general technique that is known as constraint satisfaction. To this end we adopt the formal description of Nuijten [1994].

7.1.1 Constraint Satisfaction

An instance of a constraint satisfaction problem [Montanari, 1974] involves a set of variables, a domain for each variable specifying the values that may be assigned to that variable, and a set of constraints on the variables. The constraints define which combinations of domain values are allowed. The problem is to assign values to variables such that all constraints are simultaneously satisfied. An instance of a constraint satisfaction problem is formalized as follows.

Definition 7.6 (Constraint Satisfaction Problem). An instance of a *constraint satisfaction problem* is a triple $(\mathcal{X},\mathcal{D},\mathcal{C})$, where $\mathcal{X} = \{x_1,...,x_p\}$ denotes a set of *variables*, $\mathcal{D} = \{\mathcal{D}(x_1),\ldots,\mathcal{D}(x_p)\}$ denotes a set of *domains*, such that $\mathcal{D}(x_i)$ gives the domain of each variable x_i, and $\mathcal{C} = \{c_1,...,c_q\}$ denotes a set of *constraints* on $\mathcal{D}$ such that $c_i : \mathcal{D}(x_1) \times ... \times \mathcal{D}(x_p) \to \mathbb{B}$ for all $1 \leq i \leq q$ gives a set of feasible domain values. The problem is to find an assignment $\boldsymbol{x} \in \mathcal{D}(x_1) \times ... \times \mathcal{D}(x_p)$ that satisfies $c_i(\boldsymbol{x})$ for all $1 \leq i \leq q$. Such an assignment is called a *solution*. The set of all assignments is denoted by $\mathcal{A}_{\mathcal{X},\mathcal{D}}$. □

To formulate the scheduling problem as a constraint satisfaction problem we define the set of variables, domains, and constraints as follows. The set $\mathcal{X}$ of variables contains the decision variables that we have defined in the scheduling problem, i.e., the elements of the sets $\{\delta(o) \mid o \in O\}$, $\{\sigma(o) \mid o \in O\}$, $\{\alpha(o) \mid o \in O\}$, and $\{\lambda(r) \mid r \in R\}$. The set $\mathcal{D}$ of domains contains the possible values for the set of variables, i.e., variable $\delta(o)$ has domain $\Delta(o) = \{\delta(o) \mid \delta \in \Delta\}$, variable $\sigma(o)$ has domain $\Sigma(o) = \{\sigma(o) \mid \sigma \in \Sigma\}$, variable $\alpha(o)$ has domain $\mathrm{A}(o) = \{\alpha(o) \mid \alpha \in \mathrm{A}\}$, and variable $\lambda(r)$ has domain $\Lambda(r) = \{\lambda(r) \mid \lambda \in \Lambda\}$. The set $\mathcal{C}$ of constraints contains the mapping constraints.

Often search trees are used in approaches to handle constraint satisfaction problems. *Search trees* are trees with *search states* as nodes. Going from one node to another is done by taking a *decision* and modifying the current domains. Decisions

are modeled by adding constraints. Search states record the decisions that have been taken and the current domain of each variable that contains the values of each variable that are still under consideration. Formally, a search state is defined as follows.

Definition 7.7 (Search State). A *search state* of an instance $(\mathcal{X},\mathcal{D},\mathcal{C})$ of a constraint satisfaction problem is a pair (Γ,E), where $\Gamma \subseteq \mathcal{C}$ is a set of *added constraints* and $\mathrm{E}(x) \subseteq \mathcal{D}(x)$ gives the *current domain* of x for each $x \in \mathcal{X}$. □

Initially, i.e., in the root of a search tree, the search state is $(\emptyset,\mathcal{D})$, representing that no constraints are added and the current domains are equal to $\mathcal{D}(x)$ for all variables $x \in \mathcal{X}$. In a tree search algorithm, each decision corresponds to the selection of one value for a certain variable. Selecting a value $v \in \mathrm{E}(x)$ for a variable x can be seen as adding the constraint $x = v$. At any node in the search tree a limited number of decisions can be made, defining the edges of the tree. This number equals the total number of elements in the current domains of the variables for which no value is selected yet. Each search state represents a set of solutions that satisfy both the constraints in $\mathcal{C}$ and the added constraints, and that satisfy that each variable $x \in \mathcal{X}$ has a value from $\mathrm{E}(x)$. The solution set of a search state is defined implicitly as follows.

Definition 7.8 (Solution Set). Let $(\mathcal{X},\mathcal{D},\mathcal{C})$ be an instance of the constraint satisfaction problem, and let $\Theta = (\Gamma,\mathrm{E})$ be a search state of $(\mathcal{X},\mathcal{D},\mathcal{C})$. The *solution set* of Θ is the set $\mathcal{S}(\Theta) = \{\boldsymbol{x} \in \mathcal{A}_{\mathcal{X},\mathcal{D}} \mid \forall_{c\in\mathcal{C}}\, c(\boldsymbol{x}) \wedge \forall_{c\in\Gamma}\, c(\boldsymbol{x})\}$. □

Each time a constraint is added, some values of variables may become *inconsistent*. Let $\Theta = (\Gamma,\mathrm{E})$ be a search state. A value $v \in \mathrm{E}(x)$ for a variable x is called inconsistent if no solution in $\mathcal{S}(\Theta)$ exists that includes the assignment of v to x. Then this value can be removed from the current domain of x without losing any solutions. This process of removing inconsistent values is usually called *consistency checking*. The resulting search state $\Theta' = (\Gamma,\mathrm{E}')$ is called *solution equivalent* to Θ. So, consistency checking transforms one search state into a solution equivalent one. The following definition defines this property formally.

Definition 7.9 (Solution Equivalence). Let $(\mathcal{X},\mathcal{D},\mathcal{C})$ be an instance of the constraint satisfaction problem. A *search state* $\Theta = (\Gamma,\mathrm{E})$ of $(\mathcal{X},\mathcal{D},\mathcal{C})$ is called *solution equivalent* to a search state $\Theta' = (\Gamma,\mathrm{E}')$ if and only if $\mathcal{S}(\Theta) = \mathcal{S}(\Theta')$. □

If a leaf of the search tree is reached, i.e., $|\mathrm{E}(x)| = 1$ for all $x \in \mathcal{X}$, a solution is found and the instance is solved. If a search state is reached where the current domain of any variable is empty, i.e., there exists an $x \in \mathcal{X}$ such that $\mathrm{E}(x) = \emptyset$, we say that a *dead end* occurs.

Definition 7.10 (Dead End). Let $(\mathcal{X},\mathcal{D},\mathcal{C})$ be an instance of the constraint satisfaction problem. A search state (Γ,E) is called a *dead end* if there exists an $x \in \mathcal{X}$ such that $\mathrm{E}(x) = \emptyset$. □

If a dead end is reached it is proved that no solution exists that satisfies all the original constraints together with the added constraints. Then the tree search algorithm has to *backtrack*, i.e., undo certain decisions and try alternatives for them. The search stops if the bottom of the tree is reached, i.e., a solution is found, or if all alternative decisions in the root of the tree have been tried without success. In the latter case, the instance is said to be *infeasible*. A general tree search approach iteratively selects a variable and assigns a value in its current domain. Subsequently, consistency checking is used to eliminate values that are inconsistent with the assignments made so far. Both variable and value selection strategies try to prevent the search from getting stuck in a dead end. If, however, the search does get stuck in a dead end, backtracking is needed to escape from it.

We apply this general approach to the scheduling problem in the following way. For the instances of the scheduling problem, the delay assignment has already has been determined, i.e., it initially holds that $|\Delta(o)| = 1$ for all $o \in O$. Hence, the value of the variable $\delta(o)$ is given for all $o \in O$. Subsequently, we select a variable of the form $\sigma(o)$ for some operation $o \in O$, and we assign this variable to some value v in its domain $\Sigma(o)$. This results in the additional constraint $\sigma(o) = v$. Next, we select the variable of the form $\alpha(o)$ for the same operation $o \in O$, and we assign this variable to some value v in its domain $\mathrm{A}(o)$. Finally, we select the variables $\lambda(r)$ for all data precedences r of the form $((o',n'),(o,n),(p,b,b'))$ which are connected to an input terminal of the same operation o, and we assign them to some values in their domains $\Lambda(r)$. This is followed by a consistency checking step. In the following sections we elaborate in more detail on the above-mentioned variable selection, value selection, consistency checking, and backtracking techniques for each of the subproblems in the scheduling problem.

7.2 Time Assignment

The objective of the time assignment problem is to find a time assignment that satisfies the precedence constraints. In addition, this time assignment has to be chosen such that there exists feasible processing element and channel assignments. To initialize the time assignment set we use the algorithm based on the Bellman equations as outlined in the proof of Theorem 3.8. This algorithm is based on the fact that we can write the precedence constraints as linear constraints as follows. For all data precedences $r = ((o,n),(o',n'),(p,b,b'))$ we have $\underline{\omega}(r) \leq \sigma(o') - \sigma(o) \leq \overline{\omega}(r)$, and for all no-value precedences $s = (o,o',(p,b,b'))$ we have $\underline{\omega}(s) \leq \sigma(o') - \sigma(o)$. A necessary condition to satisfy these constraints is that for all data precedences r it

holds that

$$\underline{\omega}(r) \le \min\Sigma(o') - \min\Sigma(o) \le \overline{\omega}(r) \wedge \underline{\omega}(r) \le \max\Sigma(o') - \max\Sigma(o) \le \overline{\omega}(r),$$

and that for all no-value precedences s it holds that

$$\underline{\omega}(s) \le \min\Sigma(o') - \min\Sigma(o) \wedge \underline{\omega}(s) \le \max\Sigma(o') - \max\Sigma(o).$$

In the scheduling problem we assume that the lower bounds $\underline{\omega}$ and the upper bounds $\overline{\omega}$ on the difference between the completion times of two adjacent operations are known. These bounds have been computed in the delay management and partitioning steps.

The computation of the minimum completion times is done as follows. Let for an arbitrary pair of operations o_i and o_j the longest path from o_i to o_j be given by the precedences $a_1,\dots,a_x$ provided that such a path exists. Then this path contains at most $|O|-1$ arcs. For each operation o_k on this path, the longest path from o_i to o_k is necessarily a subsequence of the path $a_1,\dots,a_x$. So by updating $\min\Sigma(o)$ to $\max\{\min\Sigma(o), \min\Sigma(o') - \overline{\omega}(a)\}$ and by updating $\min\Sigma(o')$ to $\max\{\min\Sigma(o'), \min\Sigma(o) + \underline{\omega}(a)\}$ in each iteration, we find a minimum completion time for the next operation on this path. After $|O|-1$ iterations operation o_j has obtained a minimum completion time. If no longest path exists from operation o_i to operation o_j but o_j can be reached from o_i, then the precedence constraints are infeasible. In that case there are two possibilities. The first possibility is that there is a cycle via the data precedences in the signal flow graph for which the sum of the weights $\overline{\omega}(a)$ is negative and there exists some data precedence a for which holds $\min\Sigma(o) < \min\Sigma(o') - \overline{\omega}(a)$ after $|O|-1$ iterations. The second possibility is that there is a cycle in the signal flow graph for which the sum of the weights $\underline{\omega}(a)$ is positive and there exists some precedence a for which holds $\min\Sigma(o') < \min\Sigma(o) + \underline{\omega}(a)$ after $|O|-1$ iterations.

In a similar way we find maximum completion times. Each iteration we find a maximum completion time for the next operation on the path $a_1,\dots,a_x$ by updating $\max\Sigma(o')$ to $\min\{\max\Sigma(o'), \max\Sigma(o) + \overline{\omega}(a)\}$ and by updating $\max\Sigma(o)$ to $\min\{\max\Sigma(o), \max\Sigma(o') - \underline{\omega}(a)\}$. After $|O|-1$ iterations operation o_j has obtained a maximum completion time. If the precedence constraints are infeasible then there are again two possibilities. The first possibility is that there is a cycle via the data precedences of the signal flow graph for which the sum of the weights $\overline{\omega}(a)$ is negative and there exists some data precedence a for which holds $\max\Sigma(o') > \max\Sigma(o) + \overline{\omega}(a)$ after $|O|-1$ iterations. The second possibility is that there is a cycle in the signal flow graph for which the sum of the weights $\underline{\omega}(a)$ is positive and there exists some precedence a for which holds $\max\Sigma(o) > \max\Sigma(o') - \underline{\omega}(a)$ after $|O|-1$ iterations.

Due to the no-value precedences it may be that the maximum completion time

of an operation is not bounded by the precedence constraints. In that case the maximum completion times of the successors of the operation may also be unbounded. For such operations we choose the maximum completion time equal to their minimum completion time plus the maximum program length minus one such that the operations can be placed anywhere in a program. We repeat the above-mentioned procedure until the maximum completion times of all operations are bounded.

7.2.1 Operation Selection

The operation selection strategy is based on the heuristic that variables that have smaller current domains are given priority over variables that have larger current domains in order to prevent that their current domains become empty. Therefore, our variable selection strategy chooses operations o that result in the selection of the decision variables $\sigma(o)$, $\alpha(o)$, and $\lambda(r)$ with $r = ((o',n'),(o,n),(p,b,b'))$ based on information about the time assignment set $\Sigma(o)$.

A first operation selection criterion is the cardinality of the time assignment set $\Sigma(o)$. The final completion time $\sigma(o)$ of an operation o must be a member of the set $\Sigma(o)$. In the extreme case, the time assignment set is a singleton set and it completely fixes the completion time. In general, operations with small time assignment sets are given a high priority.

A second operation selection criterion is the period of an operation. Operations with small periods have high computation requirements and more chance of overlap in time with other operations. In the extreme case, an operation has period one and it overlaps with all other operations. As a result, it fully occupies one processing element. In general, operation with small periods are given a high priority. This selection criterion is also applied in *rate-monotonic scheduling* [Liu and Layland, 1973].

We combine both criteria by observing that two operations overlap in time if and only if they have overlapping phases. The phase $\phi(o)$ of an operation o is defined as its time modulo its period, i.e., $\phi(o) = \sigma(o) \bmod p(o)$. Two operations o and o' have overlapping phases if and only if $\phi(o) \equiv \phi(o') \pmod{\gcd(p(o), p(o'))}$. Such two operations must be executed on different processing elements. For each operation o we define a set of phases $\Phi(o) = \{\sigma(o) \bmod p(o) \mid \sigma(o) \in \Sigma(o)\}$. The removal of inconsistent values from the time assignment set may also remove inconsistent values from the phase set. The number of consistent phases is a measure for the ability to avoid overlap with other operations. The number is never larger than the period of an operation.

We give the phase set priority over the time assignment set because the number of consistent phases never exceeds the number of consistent completion times. So if an operation has less consistent phases than another operation, then the latter cannot have less consistent completion times than the former. If two operations

have the same number of consistent phases then we give priority to the operation that has the smallest number of consistent completion times. This defines a partial lexicographical order of the operations.

Definition 7.11 (Operation Order). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping. Then the partial order $\preceq$ is defined for all $o,o' \in O$ in such a way that $o \preceq o'$ if and only if either $|\Phi(o)| < |\Phi(o')|$, or $|\Phi(o)| = |\Phi(o')|$ and $|\Sigma(o)| \leq |\Sigma(o')|$. □

The order of the operations can change at run-time because the consistency checking step cam remove inconsistent completion times from the time assignment set.

7.2.2 Time Selection

The time selection criterion is based on the heuristic that values that result in smaller domain reductions are given priority over values that result in larger domain reductions in order to prevent that the current domains become empty. Therefore our time selection strategy aims to choose a completion time such that the phase set $\Phi(o)$ and time assignment set $\Sigma(o)$ remain as large as possible for all $o \in O$.

A first time selection criterion is based on the *thickness function* introduced by Korst [1992]. This function represents the computation requirements at a particular time $i \in \mathbb{Z}$ based on the overlap of the operations that have already been given a completion time. Corollary 3.4 states a necessary condition for feasibility based on these computation requirements.

Definition 7.12 (Computation Requirements). Let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set of signal flow graph (O,R) onto processing element network (P,C). Then the *computation requirement* of a set O_V of operations with $V \in \mathrm{A}(O)$ at time $i \in \mathbb{Z}$ is denoted by

$$R^i_{\mathrm{A}}(O_V) = \max\{|\{o \in O_V \mid \sigma(o) \equiv i \,(\mathrm{mod}\, p(o)) \wedge \{t(o),T(V)\} \neq \{\textsc{wo},\{\textsc{me2}\}\}\}|,$$
$$|\{o \in O_V \mid \sigma(o) \equiv i \,(\mathrm{mod}\, p(o)) \wedge \{t(o),T(V)\} \neq \{\textsc{ro},\{\textsc{me2}\}\}\}|\}.$$

The *computation capacity* of a set $V \in \mathrm{A}(O)$ of processing elements is denoted by

$$C_{\mathrm{A}}(V) = |V|.$$

The fraction $R^i_{\mathrm{A}}(O_V)/C_{\mathrm{A}}(V)$ denotes the *computation workload* of processing element set V due to operation set O_V *at time i*. □

We adopt the heuristic to select completion times $\sigma(o)$ such that $R^{\sigma(o)}_{\mathrm{A}}(O_{\mathrm{A}(o)})$ is minimal thereby aiming to minimize the maximum computation workload over time. This criterion is also applied in *force-directed scheduling* [Paulin and Knight, 1989]. By doing this we aim to avoid the elimination of phases from the phase set in the consistency checking step.

To avoid the elimination of completion times from the time assignment set we introduce a second time selection criterion. If there are different completion times that minimize the maximum computation requirement then we select the completion time that is nearest to the median of the time assignment set $\Sigma(o)$. The reason for this is that the median generally results in a smaller reduction of the time assignment set of neighboring operations than the minimum or maximum value. This is illustrated in Figure 7.1 which shows the typical time assignment sets of two operations o and o' that are related by a data precedence $((o,n),(o',n'),(p,b,b'))$. The

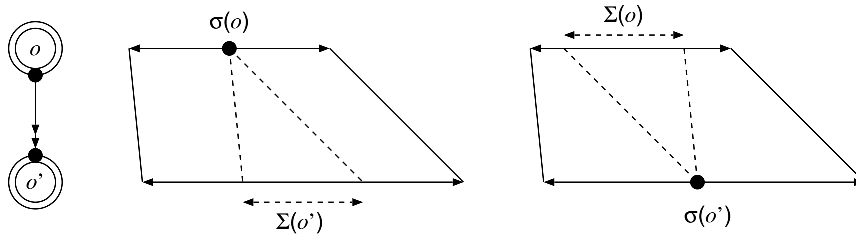


Figure 7.1. Time assignment set reduction due to precedence constraints. The picture on the left shows the impact of setting the completion time of a producer on a consumer. The picture on the right shows the impact of setting the completion time of a consumer on a producer.

selection of a completion time for producing operation o may reduce the time assignment set of consuming operation o' via the precedence constraints. Similarly, the selection of a completion time for consuming operation o' may reduce the time assignment set of producing operation o. The number of candidate consistent completion times decreases if we select completion times near the extreme values of the time assignment set, because the overlap between the time interval defined by the precedence constraints and the time assignment set of the adjacent operation decreases.

We give the computation constraints priority over the precedence constraints in order to guide the search towards a feasible processing element assignment. If different completion times result in the same computation workload we give priority to the completion time that is nearest to the median. This defines a partial lexicographical order of the completion times of operations.

Definition 7.13 (Completion Time Order). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, and let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping. Then the partial order $\leq_\sigma$ is defined for all $\sigma(o),\sigma'(o)\in\Sigma(o)$ in such a way that $\sigma(o)\preceq_\sigma\sigma'(o)$ if and only if either $R_{\mathrm{A}}^{\sigma(o)}(O_{\mathrm{A}(o)}) < R_{\mathrm{A}}^{\sigma'(o)}(O_{\mathrm{A}(o)})$, or $R_{\mathrm{A}}^{\sigma(o)}(O_{\mathrm{A}(o)}) = R_{\mathrm{A}}^{\sigma'(o)}(O_{\mathrm{A}(o)})$ and $|\mathrm{med}\Sigma(o)-\sigma(o)|\leq|\mathrm{med}\Sigma(o)-\sigma'(o)|$. □

The order of the completion times can change at run-time because the operations are given a completion time one by one.

7.2.3 Consistency Checking and Domain Reduction

After we have selected a value for the completion time $\sigma(o)$ of operation o, we remove values from the current domains that are inconsistent on account of the computation, communication, and precedence constraints.

To eliminate inconsistent values based on the computation constraints we observe that the computation requirement is not allowed to exceed the computation capacity. If we assign a completion time $\sigma(o)$ to an operation o, then the computation requirement $R_{\mathrm{A}}^{\sigma(o)}(O_{\mathrm{A}(o)})$ increases. Since it must hold that $R_{\mathrm{A}}^{\sigma(o)}(O_{\mathrm{A}(o)}) \leq C_{\mathrm{A}}(\mathrm{A}(o))$ we eliminate all completion times $\sigma(o')$ of operations $o' \in O_{\mathrm{A}(o)}$ that have not been scheduled yet and that overlap with $\sigma(o)$, if $R_{\mathrm{A}}^{\sigma(o)}(O_{\mathrm{A}(o)}) = C_{\mathrm{A}}(\mathrm{A}(o))$.

Furthermore, we know that two operations of the same array must be executed on the same processing element. So, if we assign a completion time to an operation of an array, then we can safely remove the overlapping completion times from the time assignment sets of other operations of the array that cannot overlap with the first operation on the same processing element.

Similar to the computation constraints, Corollary 3.5 states that the communication requirement is not allowed to exceed the communication capacity at time $\psi(r)$. The communication requirement and capacity at a specific moment in time are defined as follows.

Definition 7.14 (Communication Requirements). Let $(\Delta, \Sigma, \mathrm{A}, \Lambda)$ be a mapping set of signal flow graph (O,R) onto processing element network (P,C). Then the *communication requirement* of a set R_I of data precedences with $I \in \Lambda(R)$ at time $i \in \mathbb{Z}$ is denoted by

$$R_{\Lambda}^{i}(R_I) = |\{(o,n) \mid r = ((o,n),(o',n'),(p,b,b')) \in R_I \wedge \psi(r) \equiv i \pmod{p(r)}\}|.$$

The *communication capacity* of a set $I \in \Lambda(R)$ of channels is denoted by

$$C_{\Lambda}(I) = |\{(p',m') \mid ((p,m),(p',m')) \in I\}|.$$

The fraction $R_{\Lambda}^{i}(R_I)/C_{\Lambda}(I)$ denotes the *communication workload* of the channel set I due to data precedence set R_I *at time* i. □

If we assign a completion time $\sigma(o)$ to an operation o, then for all data precedences $r \in R$ the communication requirement $R_{\Lambda}^{\psi(r)}(R_{\Lambda(r)})$ at time $\psi(r)$ increases. Since it must hold that $R_{\Lambda}^{\psi(r)}(R_{\Lambda(r)}) \leq C_{\Lambda}(\Lambda(r))$ we eliminate all completion times $\sigma(o')$ of operations o' that have not been scheduled yet and that result in arrival times

$\psi(r')$ for data precedences r' for which $I(\Lambda(r')) = I(\Lambda(r))$ that overlap with $\psi(r)$, if $R_\Lambda^{\psi(r)}(R_{\Lambda(r)}) = C_\Lambda(\Lambda(r))$.

In addition we know that two different data values are not allowed to arrive at the same processing element input at the same time. So, if two data precedences are connected to two operation inputs that are mapped onto the same processing element input, then the producing operations must be scheduled such that the corresponding data arrival times do not overlap. Consequently, if we assign a completion time to an operation that results into a specific data arrival time of an array operation, then we can safely remove the completion times of other operations that lead to overlapping data arrival times of other operations in the same array.

Finally we eliminate inconsistent completion times based on the precedence constraints as follows. If we assign a completion time $\sigma(o)$ to an operation o, then we reduce the time domain $\Sigma(o)$ to a singleton set $\{\sigma(o)\}$. Subsequently, we recompute the minimum and maximum values for the completion times of operations using the longest path algorithm as outlined in the proof of Theorem 3.8. In each iteration we update the minimum completion time $\min \Sigma(o)$ of a producing operation by $\min\{\sigma(o) \in \Sigma(o) \mid \sigma(o) \geq \max\{\min \Sigma(o), \min \Sigma(o') - \overline{\omega}(a)\}\}$ and the minimum completion time $\min \Sigma(o')$ of a consuming operation by $\min\{\sigma(o') \in \Sigma(o') \mid \sigma(o') \geq \max\{\min \Sigma(o'), \min \Sigma(o) + \underline{\omega}(a)\}\}$. In each iteration we also update the maximum completion time $\max \Sigma(o)$ of a producing operation by $\max\{\sigma(o) \in \Sigma(o) \mid \sigma(o) \leq \min\{\max \Sigma(o), \max \Sigma(o') - \underline{\omega}(a)\}\}$ and the maximum completion time of a consuming operation $\max \Sigma(o')$ by $\max\{\sigma(o') \in \Sigma(o') \mid \sigma(o') \leq \min\{\max \Sigma(o'), \max \Sigma(o) + \overline{\omega}(a)\}\}$.

7.2.4 Backtracking

The search can get stuck in a dead end because we cannot satisfy the periodicity, storage, computation, or communication constraints while constructing the processing element or channel assignment for the selected completion time. Alternatively, the search can get stuck in a dead end because the time assignment set of some, yet unscheduled, operation becomes empty. In both cases we backtrack on the time selection, i.e., we select a different completion time by taking the next completion time designated by the partial order of Definition 7.13 after having removed the current infeasible completion time from the time assignment set. If the time assignment set of the current operation becomes empty, we have to backtrack on the decisions that were made for some, already scheduled, operations. One potential solution is to undo the scheduling of previous operations and to schedule the operation with the empty time assignment set earlier. The drawback of this solution is that it involves a significant amount of state saving to administrate which parts of the search space have been explored. To avoid this state saving we adopt a random restart strategy in which we restart the scheduling algorithm and we randomize the

operation selection by choosing randomly an operation out of multiple operations with the same priority in the partial ordering of Definition 7.11.

7.3 Processing Element Assignment

Next, we focus on the problem of finding a feasible processing element assignment. Formally, this problem is defined as follows.

Definition 7.15 (Processing Element Assignment Problem). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set such that $|\Delta| = |V(\mathrm{A}(o))| = |T(\mathrm{A}(o))| = 1$ for all $o \in O$, the connectivity constraints, and the relaxation of the mapping constraints are satisfied, and let $\sigma \in \Sigma$ be a time assignment such that the precedence constraints are satisfied. Find a processing element assignment $\alpha \in \mathrm{A}$, if one exists, such that the computation constraints are satisfied, i.e., for all $o,o' \in O$ it holds that if $o \neq o'$ and $\sigma(o) \equiv \sigma(o') (\bmod \gcd(p(o), p(o')))$ then $\alpha(o) \neq \alpha(o')$, the periodicity constraints are satisfied, and the storage constraints are satisfied. □

We reformulate the processing element assignment problem as a graph coloring problem. To this end we formulate an instance of the processing element assignment problem as a conflict graph. The operations of the signal graph correspond to vertices of the conflict graph except for the read and write operations because the operations of the same array have to be executed on the same processing element. For this reason, the arrays of the signal flow graph correspond to hypervertices of the conflict graph in such a way that each hypervertex contains all operations of the corresponding array. The vertices of the conflict graph are connected if and only if they contain overlapping operations that are executed on the same set of processing elements.

Definition 7.16 (Processing Element Assignment Problem Reformulated). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set such that $|\Delta| = |V(\mathrm{A}(o))| = |T(\mathrm{A}(o))| = 1$ for all $o \in O$, the connectivity constraints, and the relaxation of the mapping constraints are satisfied, and let $\sigma \in \Sigma$ be a time assignment such that the precedence constraints are satisfied. Then the graph $(V_\mathrm{A}, E_\mathrm{A})$ is called a *processing element conflict graph* where the set of vertices V_A is defined by

$$\begin{aligned} V_\mathrm{A} = &\{\{o\} \in \mathcal{P}(O) \mid t(o) \notin \{\textsc{ro}, \textsc{wo}\}\} \cup \\ &\{\{o \in O \mid t(o) \in \{\textsc{ro}, \textsc{wo}\} \wedge a(o) = a\} \mid a \in A\}, \end{aligned}$$

and the subset of edges E_A contains an edge $\{V, V'\}$ with $V \neq V'$ if and only if there exists operations $o \in V$ and $o' \in V'$ satisfying $\mathrm{A}(o) = \mathrm{A}(o')$ for which it holds that

$$\sigma(o) \equiv \sigma(o') \pmod{\gcd(p(o), p(o'))}$$

or

$$T(\mathrm{A}((o)) = T(\mathrm{A}((o')) = \{\mathrm{ME2}\} \wedge \{t(o), t(o')\} = \{\mathrm{RO}, \mathrm{WO}\}.$$

The problem is to find a function $f : V_\mathrm{A} \to P$, if one exists, such that for all $V \in V_\mathrm{A}$ it holds that

$$f(V) \in \bigcup_{o \in V} \mathrm{A}(o),$$

for all $\{V, V'\} \in E_\mathrm{A}$ it holds that

$$f(V) \neq f(V'),$$

and the processing element assignment $\alpha \in \mathrm{A}$, that is defined for all $V \in V_\mathrm{A}$ and for all $o \in V$ by $\alpha(o) = f(V)$, satisfies the periodicity and storage constraints. □

In general a processing element conflict graph is not a connected graph. During scheduling the processing element assignment set is such that for each pair of operations $o, o' \in O$ it holds that either $\mathrm{A}(o) = \mathrm{A}(o')$ or $\mathrm{A}(o) \cap \mathrm{A}(o') = \emptyset$. Hence, the set of processing element sets $\mathrm{A}(O)$ partitions the conflict graph in $|\mathrm{A}(O)|$ different components. In order to satisfy the computation constraints each vertex $V \in V_A$ of each component $V_A \subseteq V_\mathrm{A}$ has to be assigned to a processing element from the set $\bigcup_{o \in V} \mathrm{A}(o) \in \mathrm{A}(O)$ in such a way that two adjacent vertices are assigned to different processing elements.

7.3.1 Operation Selection

The complete edge set of the conflict graph can only be determined if the complete time assignment is known. However, a partial time assignment already defines a subset of the final edge set namely the edges of which both ends contain an operation that has been given a completion time. Once an operation o has been assigned to a time $\sigma(o)$ and has to be assigned to a processing element $\alpha(o)$, we extend the edge set. At any time we have that the edge set E_A contains the edge $\{V, V'\}$ if and only if there exists operations $o \in V$ and $o' \in V'$ satisfying $\mathrm{A}(o) = \mathrm{A}(o')$ for which it holds that $|\Sigma(o)| = |\Sigma(o')| = 1$. So the selection of variable $\alpha(o)$ extends the edge set with zero or more edges which are incident to the vertex V that corresponds to operation o. As a result the coloring of the conflict graph may become infeasible.

7.3.2 Processing Element Selection

If the coloring of the conflict graph becomes infeasible we have to recolor the graph. The Kempe-chain argument [Kempe, 1879] [Gibbons, 1989] concerns the recoloring of vertices in a properly colored graph to obtain a different proper coloring of the graph. Consider a vertex V which is colored $f(V) = a$. Then this vertex plus all other vertices which are reachable from this vertex via paths along vertices

that are colored a or b for some color $b \neq a$, define a subgraph $H(a,b)$. The Kempe-chain argument states that in a proper coloring of the graph the vertices of such a subgraph $H(a,b)$ which are colored a can be colored b, and those that are colored b can be colored a, in order to obtain a new proper coloring of the graph. We apply this argument whenever we add an edge $\{V,V'\}$ to the conflict graph between two vertices V and V' with the same color $f(V) = f(V') = a$. In such a situation we try to find a subgraph $H(a,b)$ starting from vertex V that does not contain vertex V'. If such a subgraph exists then we can recolor the subgraph and add the edge $\{V,V'\}$. Note that the choice to start from vertex V' does not affect the existence of such a subgraph. However, it does affect the resulting coloring of the conflict graph. The order in which the edges are added to the conflict graph does affect the existence of such a subgraph as is illustrated in the following example.

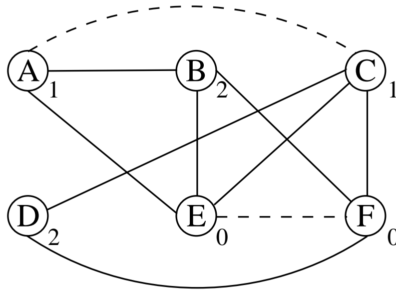


Figure 7.2. Graph coloring based on the Kempe-chain argument.

Given is the graph in Figure 7.2 in which the set of allowed colors for each vertex equals $\{0,1,2\}$. All edges except the dashed ones are in the graph. The current color of each vertex has been depicted next to the vertex. In the initial graph there were no edges and each vertex had color 0. The coloring of Figure 7.2 has been obtained by adding the edges to this initial graph, each time recoloring with Kempe-chains, in the following order: AB, CD, AE, BE, CE, BF, CF, DF. We now try to add edge EF to this graph. Vertices E and F have the same color, so we have to recolor either vertex E or vertex F. If we try to recolor vertex E with color 1, we find the subgraph with vertices E, A, C, and F. Because vertex F is also in this subgraph, vertex E cannot be recolored with color 1. The same reasoning holds if we try to color vertex F with color 1. We now try color 2. Then we find the subgraph with vertices E, B, and F. Both vertices E and F are again in the subgraph, so neither of them can be recolored with color 2. Because there are no other colors available, edge EF cannot be added to this graph with Kempe-chains recoloring. Now we try to add edge AC to this graph. Vertices A and C have the same color, so we have to recolor either vertex A or vertex C. This cannot be done with color 0, because we then find the subgraph with vertices A, E, C, and F, which contains both

vertices A and C. If we try to color vertex A with color 2, we find the subgraph with vertices A and B. Because vertex C is not in this subgraph, we can recolor vertex A with 2 and vertex B with 1. Then the edge AC can be added to this graph. However, now it is possible to add edge EF to this graph which failed earlier when edge AC was not yet in the graph. We can now recolor vertex E with color 2. This leads to the subgraph with vertices E and A which does not contain vertex F. We can then recolor vertex E with 2 and vertex A with 0. This example shows that the success of the recoloring with Kempe-chains depends on the order in which the edges are added to the graph.

The recoloring algorithm based on the Kempe-chain argument does not consider the periodicity and storage constraints. In order to model the storage constraints, we require a *capacitated* graph coloring problem. In the capacitated graph coloring problem the coloring of a vertex requires a certain amount of color and there is only a certain amount of each color available. To model the periodicity constraints, we even require that the coloring of a set of vertices requires a different amount of color than the sum of the amounts of the individual vertices. If the current coloring violates the periodicity or the storage constraints or if the recoloring procedure based on the Kempe-chain argument fails to construct a proper coloring, then we resort to a modified version of sequential coloring [Welsh and Powell, 1967] to handle the capacitated graph coloring problem. Sequential coloring usually sorts the vertices according to degree and, subsequently, colors them sequentially in order of descending degree. In our version of sequential coloring, we define a lexicographical order of the vertices in which the storage requirement of the vertices has priority over the degree of the vertices. Subsequently, we color the vertices using a first fit heuristic, i.e., we choose the first color such that the resulting periodicity and storage requirements do not exceed the available capacities.

7.3.3 Backtracking

If for some operation o we cannot find a feasible processing element assignment, then we assume that it does not exist. Subsequently we backtrack on the time assignment of operation o. The selected completion time $\sigma(o)$ always gives an infeasible solution so we remove it from the current time assignment set $\Sigma(o)$. We also remove the completion times $\sigma(o)+kp(o)$ for all integers k because they result in the same conflict graph. On the resulting time assignment set we reapply the time selection algorithm as outlined in Section 7.2.2.

7.4 Channel Assignment

Finally, we focus on the channel assignment problem. Formally, the problem is defined as follows.

Definition 7.17 (Channel Assignment Problem). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set such that $|\Delta| = |V(\mathrm{A}(o))| = |T(\mathrm{A}(o))| = 1$ for all $o \in O$, the connectivity constraints, and the relaxation of the mapping constraints are satisfied, let $\sigma \in \Sigma$ be a time assignment such that the precedence constraints are satisfied, and let $\alpha \in \mathrm{A}$ be a processing element assignment such that the computation constraints, the periodicity constraints, and the storage constraints are satisfied. Find a channel assignment $\lambda \in \Lambda$, if one exists, such that the communication constraints are satisfied, i.e., for all data precedences $r, r' \in R$ it holds that if $r \neq r'$ and $\psi(r) \equiv \psi(r') (\bmod \gcd(p(r), p(r')))$ then $\lambda(r) \neq \lambda(r')$. □

We also reformulate the channel assignment problem as a graph coloring problem. To this end we formulate an instance of the channel assignment problem as a conflict graph. Each data precedence in the signal flow graph corresponds to a vertex in the conflict graph. The vertices in the conflict graph are connected if and only if the corresponding data precedences have overlapping arrival times for different data on the same set of input terminals of the same processing element.

Definition 7.18 (Channel Assignment Problem Reformulated). Let (O,R) be a signal flow graph, let (P,C) be a processing element network, let $(\Delta,\Sigma,\mathrm{A},\Lambda)$ be a mapping set such that $|\Delta| = |V(\mathrm{A}(o))| = |T(\mathrm{A}(o))| = 1$ for all $o \in O$, the connectivity constraints, and the relaxation of the mapping constraints are satisfied, let $\sigma \in \Sigma$ be a time assignment such that the precedence constraints are satisfied, and let $\alpha \in \mathrm{A}$ be processing element assignment such that the computation constraints, the periodicity constraints, and the storage constraints are satisfied. Then the graph (V_Λ, E_Λ) is called a *channel conflict graph* where the set of vertices V_Λ is defined by

$$V_\Lambda = R,$$

and the set of edges E_Λ contains the edge $\{r_1, r_2\}$, where $r_1 = ((o_1,n_1), (o'_1,n'_1),(p_1,b_1,b'_1))$ and $r_2 = ((o_2,n_2),(o'_2,n'_2),(p_2,b_2,b'_2)) \in R$ if and only if $(o_1,n_1) \neq (o_2,n_2)$, $(\alpha(o'_1), I_\mathrm{p}(t(\alpha(o'_1)))) = (\alpha(o'_2), I_\mathrm{p}(t(\alpha(o'_2))))$, and

$$\psi(r_1) \equiv \psi(r_2) \pmod{\gcd(p_1, p_2)}.$$

The problem is to find a channel assignment $\lambda : R \to C$, if one exists, such that for all $r \in R$ it holds that

$$\lambda(r) \in \{((\alpha(o),m),(\alpha(o'),m')) \in \Lambda(r) \mid m \in O_\mathrm{p}(t(\alpha(o))) \wedge m' \in I_\mathrm{p}(t(\alpha(o')))\},$$

and for all $\{r_1, r_2\} \in E_\Lambda$ it holds that

$$\lambda(r_1) \neq \lambda(r_2).$$

□

The complete edge set of the conflict graph can only be determined if the complete time and processing element assignments are known. However, a partial time and processing element assignment already define a subset of the final edge set namely the edges of which both ends contain an operation that has been given a completion time and a processing element. Once an operation o has been assigned to a time $\sigma(o)$ and to a processing element $\alpha(o)$, we extend the edge set. At any time we have that the edge set E_Λ contains the edge $\{r_1, r_2\}$ if and only if there exist vertices V and V' in the processing element conflict graph for which it holds that $o'_1 \in V$, $o'_2 \in V'$, and $f(V) = f(V')$ and if it holds that $|\Sigma(o_1)| = |\Sigma(o_2)| = 1$. The resulting graph coloring instances are solvable in polynomial time because they have to be colored with at most two colors [Garey et al., 1976]. This is due to the fact that the channel assignment set $\Lambda(r)$ contains at most two channel assignments that obey the type constraints for the given processing element assignment α. To handle the channel assignment problem we reuse the heuristic for coloring processing element conflict graphs that is based on the Kempe-chain argument. This heuristics optimally solves the coloring of graphs with two colors.

7.4.1 Backtracking

If for some precedence r_1 we cannot find a feasible channel assignment, then it must be in conflict with some other precedence r_2. In that case there is an odd-length cycle in the channel conflict graph such that it cannot be colored with two colors. The removal of one edge $\{r_1, r_2\}$ in the cycle, by adding the corresponding edge $\{V, V'\}$ with $o'_1 \in V$ and $o'_2 \in V'$ in the processing element conflict graph, suffices to solve that particular channel conflict. The choice of which edge is to be removed, determines which edge is added to the processing element graph. Preferably, we prevent a recoloring of the processing element conflict graph by adding an edge between two vertices that already have different colors. If that is not possible, we adopt the heuristic to add an edge between the vertices of which the maximum degree is minimal in order to simplify the recoloring of the processing element conflict graph. If the new processing element conflict graph cannot be recolored then we resort to the old conflict graph and we change the time assignment of the current operation to obtain different processing element and channel conflict graphs.

7.5 Results

In this section we present the results of the proposed mapping approach on the set of industrially relevant problem instances. We applied the proposed scheduling approach on the instances that have been generated by the preceding delay management and partitioning steps. If we were not able to solve these instances, then we backtracked on the delay management and partitioning steps.

Table 7.1. Mapping results of signal flow graphs that are mapped onto the VSP1FLEX processor network. From left to right the columns list the number of operation and data precedences, the average computation and storage utilization of the processors in percentages, and whether we succeeded to map the instance with presented approach.

SFG	$\lvert O\rvert$	$\lvert R\rvert$	$R_{\mathrm{A}}/C_{\mathrm{A}}$			R_{Δ}/C_{Δ}	solved
			ALE	OE	ME1		
YUVTORGB	73	112	71	45	0	0	yes
HORCOMPR	154	273	21	20	13	25	yes
IJNTEMA1	257	426	47	38	19	19	yes
CORMACK2	232	370	37	25	9	31	yes
CONTOUR1	131	240	73	40	20	37	yes
MONZA2	123	224	82	45	25	42	yes
VDP	282	516	68	48	30	44	yes
GAMMA	366	559	59	41	22	15	yes
HISTMOD2	328	506	38	33	21	12	yes
PANORAMA	164	323	55	54	50	75	yes
VIDIWALL	284	568	82	52	35	57	yes
IJNTEMA2	485	795	86	53	20	25	no
CORMACK1	334	601	91	63	72	79	no
MONZA1	380	656	74	39	31	54	yes
MWTV	600	953	75	36	30	42	no

Table 7.1 lists the mapping results of the signal flow graphs that are mapped onto the VSP1FLEX processor network. The signal flow graphs CORMACK2, GAMMA, PANORAMA, and VIDIWALL violate an upper bound of a precedence constraint after delay management and partitioning. For this reason we have reapplied delay management on these instances which has been successful except for VIDIWALL where a relaxation of the computation constraints was violated after delay management. To solve this problem we have generated a different partition using $\varepsilon = 0.3$, after which the subsequent delay management step was successful. Furthermore, the signal flow graphs HISTMOD2, VDP, IJNTEMA2, and MWTV violate a lower bound of a precedence constraints after delay management and partitioning. In case of HISTMOD2 we have generated a different partition using $\varepsilon = 0.25$. As a result, the partitioning step inserts less pass operation in the cycle that violated the lower bound. In case of VDP this strategy does not work. Instead we have performed the initial delay management before partitioning with a smaller upper bound for the precedence constraints assuming that the silo size is 22 rather than 32. We were not able to satisfy the precedence constraints for the signal flow graphs IJNTEMA2 and MWTV with the presented solution approaches for delay management and partitioning.

Once the preconditions for scheduling are satisfied, the subsequent scheduling step is not able to find feasible solutions for the signal flow graphs MONZA2, PANORAMA, and CORMACK1. Signal flow graph MONZA2 can be scheduled with a different partition using $\varepsilon = 0$. Signal flow graph PANORAMA can be scheduled with a different partition using $\varepsilon = 0.35$ and a subsequent delay management step using silo size 31. The scheduling algorithm cannot satisfy all computation and communication constraints in signal flow graph CORMACK1 despite trying different partitions. The reason for this is that the partitioning step produces infeasible instances of the scheduling problem. This is caused by the fact the signal flow graph contains operations with periods 1, 2, 4, 12, and 16 which do not form a divisible sequence. The average computation requirement of many combinations of these operations does not correctly reflect the actual number of required processing elements. For instance, a combination of 4 operations with period 2, 7 operations with period 12, and 1 operation with period 16 has an average computation requirement of 2.64, but this combination cannot be mapped onto 3 processing elements that have a program length of 16. Due to the bound on the program length the operations with period 12 and 16 cannot be mapped onto the same processing element, which makes the combination infeasible. However, these periodicity constraints are not considered during partitioning.

Table 7.2 lists the mapping results for the signal flow graphs that are mapped onto the VSP2TEST processor network. The signal flow graphs CORMACK2, GAMMA, MONZA2, PANORAMA, and VIDIWALL violate an upper bound of a precedence constraints after delay management and partitioning. We have successfully reapplied delay management to satisfy the precedence constraints after partitioning. The resulting problem instances can be scheduled successfully

Table 7.3 lists the mapping results for the signal flow graphs that are mapped onto the VSP2FLEX processor network. The signal flow graphs CONTRAST, FDXD2, HSRC, and VSRC violate an upper bound of a precedence constraints after delay management and partitioning. We have successfully reapplied delay management to satisfy the precedence constraints after partitioning. However, the instance VSRC cannot be scheduled under the resulting precedence constraints. Furthermore, our scheduling algorithm cannot find a solution for the resulting instance FDXD2. For this reason, we have again reapplied delay management on these two signal flow graphs under the assumption the silos have a size of 28 and 31, respectively. This slightly changes the precedence constraints, after which our scheduling algorithm is able to find solutions.

The instances of the mapping problem that are listed in Table 7.3 have a degree of utilization of the arithmetic and logic elements of approximately 50%. In order to evaluate the presented mapping techniques at higher degrees of utilization we map the signal flow graphs onto smaller processor networks. To this end we reduce

Table 7.2. Mapping results of signal flow graphs that are mapped onto the VSP2TEST processor network. From left to right the columns list the number of operation and data precedences, the average computation and storage utilization of the processors in percentages, and whether we succeeded to map the instance with presented approach.

SFG	$\|O\|$	$\|R\|$	R_A/C_A				R_Δ/C_Δ	solved
			ALE	BE	OE	ME1		
YUVTORGB	42	81	71	17	4	0	0	yes
HORCOMPR	89	208	22	0	50	13	25	yes
IJNTEMA1	189	363	48	4	42	23	19	yes
CORMACK2	169	307	39	1	33	18	31	yes
CONTOUR1	86	201	73	8	17	20	37	yes
MONZA2	89	179	82	25	25	34	42	yes
VDP	195	426	68	17	29	36	44	yes
GAMMA	254	447	50	10	25	27	17	yes
HISTMOD2	214	392	42	0	25	23	12	yes
PANORAMA	99	258	53	23	17	56	75	yes
VIDIWALL	178	438	78	12	50	37	57	yes
IJNTEMA2	309	598	86	23	25	29	26	yes
MONZA1	265	541	74	12	25	31	54	yes
MWTV	418	731	75	2	25	44	42	yes

Table 7.3. Mapping results of signal flow graphs that are mapped onto the VSP2FLEX processor network. From left to right the columns list the number of operation and data precedences, the average computation and storage utilization of the processors in percentages, and whether we succeeded to map the instance with presented approach.

SFG	$\|O\|$	$\|R\|$	R_A/C_A				R_Δ/C_Δ	solved
			ALE	BE	OE	ME2		
CONTRAST	356	584	57	7	33	18	33	yes
FDXD1	230	581	37	1	45	10	1	yes
FDXD2	281	589	45	6	43	22	18	yes
MAT30UP	258	462	43	2	32	10	6	yes
HSRC	402	996	50	11	35	25	28	yes
VSRC	598	1270	50	15	57	29	21	yes

the VSP2FLEX processor network to a network of three processors and to a network of four processors. We map the signal flow graphs FDXD1, FDXD2, and MAT30UP onto the reduced network of three processors because these signal flow graphs have a utilization of less than 50%. We map the signal flow graphs CONTRAST, HSRC, VSRC that have a utilization of 50% or more onto the reduced network of four processors. The results are listed in Table 7.4. These results show that the presented

Table 7.4. Mapping results of signal flow graphs that are mapped onto two fictive processor networks. From left to right the columns list the number of operation and data precedences, the average computation and storage utilization of the processors in percentages, and whether we succeeded to map the instance with presented approach.

SFG	$\lvert O\rvert$	$\lvert R\rvert$	R_A/C_A				R_Δ/C_Δ	solved
			ALE	BE	OE	ME2		
CONTRAST	340	576	84	13	38	27	46	yes
FDXD1	217	587	72	7	62	19	3	yes
FDXD2	231	551	85	38	54	44	36	yes
MAT30UP	159	328	85	3	24	20	11	yes
HSRC	527	990	74	35	51	38	42	yes
VSRC	456	1125	69	15	52	33	30	yes

mapping techniques are effective up to resource utilizations of 75%. Furthermore, the presented mapping techniques significantly improve efficiency since each mapping presented in this thesis has been produced within several minutes. To illustrate the improvement in efficiency we mention that an experienced person requires several hours to manually map the signal flow graph CONTRAST onto the VSP2FLEX processor network.

7.6 Summary

In this chapter we have presented an approach to handle the scheduling problem. The approach decomposes the scheduling problem into the time assignment problem, the processing element assignment problem, and the channel assignment problem. The objective of the time assignment problem is to compute completion times for operations in such a way that the computation requirements are evenly distributed among the clock cycles. The objective of the processing element assignment problem is to assign the operations of which the executions overlap in time to different processing elements. The objective of the channel assignment problem is to assign data precedences that overlap in time to different communication channels. We have developed a constraint satisfaction heuristic to handle scheduling problem in which a partial time assignment, a partial processing element assignment, and a partial channel assignment are iteratively extended to total assignments. The results indicate that the scheduling approach can produce feasible solutions even at high degrees of resource utilization.

8

Conclusion

We have considered the problem of mapping signal flow graphs onto networks of programmable video signal processors. In this mapping problem we have to schedule the executions of periodic operations on processing elements, the samples of periodic data precedences on communication channels, and the lifetimes of periodic operands in memories subject to resource and time constraints. We have presented an approach to handle the mapping problem that finds feasible solutions even at high degrees of resource utilization.

In Chapter 2 we have presented a mathematical formulation of the mapping problem. The formulation consists of four decision variables which are called delay assignment, time assignment, processing element assignment, and channel assignment. The combination of these assignment is subject to eight constraints which are called type constraints, array constraints, connectivity constraints, precedence constraints, computation constraints, communication constraints, storage constraints, and periodicity constraints. We have introduced two transformations on signal flow graphs in order to be able to efficiently implement the communication and storage of operands.

In Chapter 3 we have shown that the mapping problem is hard in the formal sense. This caused by several combinations of constraints. The storage and periodicity constraints are hard, but they can be solved in polynomial time by exhaustive search if the data memory and program memory, respectively, have fixed

sizes. The computation and communication constraints are hard, but they can be solved in polynomial time if the periods of the operations and precedences, respectively, form a divisible sequence. The type, array, connectivity, and precedence constraints are easy. Signal flow graph transformation in combination with the precedence, computation, and connectivity constraints is hard even when the periods form a divisible sequence, because the computation and connectivity constraints bound the number of operations that can be added to satisfy the precedence constraints. Signal flow graph transformation in combination with the computation, communication, and connectivity constraints is hard even when the periods form a divisible sequence, because the computation and communication constraints bound the number of operations and precedences that can be added to satisfy the connectivity constraints.

In Chapter 4 we have decomposed the mapping problem into the delay management problem, the partitioning problem, and the scheduling problem. The objective of the delay management problem is to add operations to a signal flow graph that store the operands in memories subject to the combination of precedence, computation, and connectivity constraints. The objective of partitioning is to add operations to a signal flow graph that communicate operands along the communication channels subject to the combination of computation, communication, and connectivity constraints. The objective of the scheduling problem is to schedule the operations of a signal flow graph on the processing elements subject to the mapping constraints.

In Chapter 5 we have presented an approach to handle the delay management problem. The approach is based on a decomposition of the delay management problem into the delay minimization problem and the delay assignment problem. The objective of the delay minimization problem is to minimize the lifetimes of the operands. The objective of the delay assignment problem is to allocate room for each operand in one or more memories. We have shown that the delay minimization problem can be solved in polynomial time with network flow techniques because it is a special case of the dual of the minimum cost flow problem. We have developed a bin packing heuristic to handle the delay assignment problem. The results indicate that the delay management approach produces balanced solutions in which the degree of utilization is evenly distributed among the different memory types.

In Chapter 6 we have presented an approach to handle the partitioning problem. The approach is based on a decomposition of the single multi-way partitioning problem into multiple two-way partitioning problems. The advantage of this approach is that it effectively reduces the search space for the routing of communication through a processor network. The decomposition produces a binary tree of a processor network in which the root represents the entire processor network

and each node represents a subgraph of the processor network. The two children of each node split the subgraph into two disjoint subgraphs. The leafs of the tree represent the individual processors of the network. The objective of the resulting two-way partitioning problem is to recursively split a signal flow graph into two pieces such that each piece fits on the corresponding subgraph of the processor network. We have developed a local search heuristic to handle the two-way partitioning problem. The results indicate that the partitioning approach is able to produce balanced solutions in which the degree of utilization of the resources is evenly distributed among the processors in the network.

In Chapter 7 we have presented an approach to handle the scheduling problem. The approach is based on a decomposition of the scheduling problem into the time assignment problem, the processing element assignment problem, and the channel assignment problem. The objective of the time assignment problem is to determine completion times for operations in such a way that the computation requirements are evenly distributed among the clock cycles. The objective of the processing element assignment problem is to assign the operations of which the executions overlap in time to different processing elements. The objective of the channel assignment problem is to assign data precedences that overlap in time to different communication channels. We have developed a constraint satisfaction heuristic to handle the scheduling problem in which a partial time assignment, a partial processing element assignment, and a partial channel assignment are iteratively extended to total assignments. The results indicate that the scheduling approach can produce feasible solutions even at high degrees of resource utilization.

We have applied the developed mapping techniques on a set of industrially relevant video applications and processor networks. These experiments have shown that these techniques can automatically map applications containing hundreds of operations onto processor networks containing tens of processing elements. The automatic mapping techniques are effective up to resource utilizations of 75 percent. Furthermore, they increase efficiency compared to manual mapping because they reduce the time that is required to map an application onto a processor network from hours to minutes.

The added value of the research presented in this thesis is the formal model of the complete mapping problem and the use of this model in order to reason on the decomposition of the mapping problem and the definition of the subproblems. The model provides an entry point for combinatorial optimization techniques in order to handle the subproblems. Furthermore, the model can serve as a template for the modeling of other mapping problems. Finally, we mention that current research directions are moving from off-line mapping to on-line mapping techniques because video applications tend to become more dynamic which means that their execution depends on the size and the value of run-time data.

Bibliography

AARTS, E.H.L., G. ESSINK, AND E.A. DE KOCK [1996], Recursive bipartitioning of signal flow graphs for programmable video signal processors, *Proceedings European Design & Test Conference 1996*, 460–466.

AARTS, E.H.L., AND J.K. LENSTRA [1997], *Local search in combinatorial optimization*, Wiley, Chichester.

AHO, A.V., J.E. HOPCROFT, AND J.D. ULLMAN [1974], *The design and analysis of computer algorithms*, Addison-Wesley, Reading.

AHUJA, R.K., T.L. MAGNANTI, AND J.B. ORLIN [1989], Network flows, in: G.L. Nemhauser, A.H.G. Rinnooy Kan, and M.J. Todd (eds.), *Handbooks in operations research and management science; Volume 1: Optimization*, North Holland, Amsterdam, 211–369.

ARAUJO, G., AND S. MALIK [1995], Optimal code generation for embedded memory non-homogeneous register architectures, *Proceedings of the 8th International Symposium on System Synthesis*, 36–41.

ASHRAF, M., AND S.H. BOKHARI [1995], Efficient algorithms for a class of partitioning problems, *IEEE Transactions on Parallel and Distributed Systems* **6**, 170–175.

BALMER, K., N. SIMMONS, P. MOYSE, I. ROBERTSON, J. KEAY, M. HAMMES, E. OAKLAND, R. SIMPSON, G. BARR, AND D. ROSKELL [1994], A single chip multimedia video processor, *Proceedings of IEEE Custom Integrated Circuits Conference - CICC 1994*, 91–94.

BILSEN, G., M. ENGELS, R. LAUWEREINS, AND J. PEPERSTRAETE [1994], Static scheduling of multi-rate cyclo-static DSP applications, *Proceedings Workshop on VLSI Signal Processing*, 137–146.

BOKHARI, S.H. [1988], Partitioning problems in parallel, pipelined, and distributed computing, *IEEE Transactions on Computers* **37**, 48–57.

BOVE, V.M., AND J.A. WATLINGTON [1995], Cheops: a reconfigurable data-flow system for video processing, *IEEE Transaction on Circuits and Systems for Video Technology* **5**, 140–149.

BUCK, J.T. [1994], Static scheduling and code generation from dynamic dataflow graphs with integer-valued control streams, *Proceedings of the 1994 28th Asilomar Conference on Signals, Systems, and Computers*, 508–513.

CAVIGIOLI, C.D. [1987], Architecture of the ADSP-2100 digital signal processor, *Midcon 1987 Conference Record*, 412–416.

CHANG, P., D.Y CHEN, Y.F. LEE, Y. WU, AND U. BANERJEE [1997], Bidirectional scheduling: a new global code scheduling approach, *Proceedings of the 9th International Workshop on Languages and Compilers for Parallel Computing*, 220–230.

CHEN, Y., Y. HSU, AND C. KING [1994], MULTIPAR, behavioural partition for synthesising multiprocessor architectures, *IEEE Transactions in Very Large Scale Integration Systems* **2**, 140–149.

CHENG, W.K., AND Y.L. LIN [1995], A transformation-based approach for storage optimization, *Proceedings of the 32nd Design Automation Conference*, 158–163.

COFFMAN, E.G. JR. [1976], *Computer and job-shop scheduling theory*, Wiley, New York.

COFFMAN, E.G. JR., M.R. GAREY, AND D.S. JOHNSON [1987], Bin packing with divisible item sizes, *Journal of Complexity* **3**, 406–428.

COFFMAN, E.G. JR., M.R. GAREY, AND D.S. JOHNSON [1997], Approximation algorithms for bin packing: a survey, in: D.S. Hochbaum (ed.), *Approximation algorithms for NP-hard problems*, PWS Publishing, 49–93.

COOK, S.A. [1971], The complexity of theorem-proving procedures, *Proceedings 3rd Annual ACM Symposium on Theory of Computing*, 151–158.

COOK, W.J., W.H. CUNNINGHAM, W.R. PULLEYBLANK, AND A. SCHRIJVER [1997], *Combinatorial Optimization*, Wiley, New York.

DENK, T.C., AND K. PARHI [1994], Calculation of minimum number of registers in 2-D discrete wavelet transforms using lapped block processing, *Proceedings 1994 International Symposium on Circuits and Systems*, 77–80.

DEPUYDT, F., G. GOOSSENS, AND H. DE MAN [1994], Scheduling with register constraints for DSP architectures, *Integration The VLSI Journal* **18**, 95–120.

DESMET, D., AND G. GENIN [1993], ASSYNT: efficient assymbly code generation for digital signal processors starting from a data flowgraph, *Proceedings of ICASSP 1993*, 45–48.

DIJKSTRA, H., H. HOLLMANN, K. HUIZER, AND R. SLUYTER [1989], New programmable delay element, *Electronic Letters* **25**, 1019–1021.

DONGEN, R.C.A. VAN [1991], *Mapping for digital video signal processors: models and algorithms*, Master's thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.

ESSINK, G., E. AARTS, R. VAN DONGEN, P. VAN GERWEN, J. KORST, AND K. VISSERS [1991b], Architecture and programming of a VLIW style video signal processor, *Micro-24*, 181–188.

ESSINK, G., E. AARTS, R. VAN DONGEN, P. VAN GERWEN, J. KORST, AND

K. VISSERS [1991a], Scheduling in programmable video signal processors, *Proceedings of the IEEE International Conference on Computer-Aided Design*, 284–287.

FLUITER, B.L.E. DE [1993], *A complexity catalogue of high-level synthesis problems*, Master's thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.

FRENCH, S. [1982], *Sequencing and scheduling: an introduction to the mathematics of the job shop*, Ellis Horwood, Chichester.

GAREY, M.R., AND D.S. JOHNSON [1979], *Computers and intractability: a guide to the theory of NP-completeness*, W.H. Freeman and Company, New York.

GAREY, M.R., D.S. JOHNSON, G.L. MILLER, AND C.H. PAPADIMITRIOU [1980], The complexity of coloring circular arcs and chords, *SIAM Journal on Algebraic and Discrete Methods* **1**, 216–227.

GAREY, M.R., D.S. JOHNSON, AND L. STOCKMEYER [1976], Some simplified NP-complete graph problems, *Theoretical Computing Science* **1**, 237–267.

GEBOTYS, C.H. [1997], An efficient model for DSP code generation: performance, code size, estimated energy, *Proceedings of the 10th International Symposium System Synthesis*, 41–47.

GIBBONS, A. [1989], *Algorithmic graph theory*, Cambridge University Press.

GOOSSENS, G., J. RABAEY, J. VANDEWALLE, AND H. DE MAN [1990], An efficient microcode compiler for application-specific DSP-processors, *IEEE Transactions on Computer-Aided Design* **9**, 925–937.

GREWAL, G.W., AND T.C. WILSON [1997], Shake and bake: a method of mapping code to irregular DSPs, *Proceedings of the 10th International Conference on VLSI Design*, 506–508.

GUMUSKAYA, H., B. ORENCIK, T. KURUGOLLA, AND T. PALAZ [1994], Automatic scheduling of real-time digital filtering algorithms onto processors, *Proceedings of the 5th International Conference on Signal Processing Applications and Technology*, 606–611.

HEI, G.D. LA, E. RIDDERSMA, AND A.K. RIEMENS [1996], *A VSP2 chip test board*, Technical report, Philips Research NL-TN 032/96.

HU, X., S.C. BASS, AND R.G. HARBER [1994], Minimizing the number of delay buffers in the synchronization of pipelined systems, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **13**, 1441–1449.

HWANG, Y.T., AND J.S. HWANG [1997], Efficient code generation for digital signal processors with parallel and pipelined instructions, *1997 IEEE Workshop on Signal Processing Systems*, 243–252.

JANSEN, R.E.J. [1994], *Partitioning of signal flow graphs for programmable video signal processors*, Master's thesis, Department of Mathematics and

Computing Science, Eindhoven University of Technology.

JOHNSON, D.S., C.H. PAPADIMITRIOU, AND M. YANNAKAKIS [1988], How easy is local search?, *Journal of Computer and System Science* **37**, 79–100.

KALOUPTSIDIS, N. [1997], *Signal processing systems: theory and design*, Wiley, New York.

KARP, R.M. [1972], Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher (eds.), *Complexity of computer computations*, Plenum Press, New York, 85–103.

KARP, R.M. [1975], On the computational complexity of combinatorial problems, *Networks* **5**, 45–68.

KEMPE, A.B. [1879], On the geographical problems of the four colours, *American Journal of Mathematics* **2**, 193–200.

KERNIGHAN, B.W., AND S. LIN [1970], An efficient heuristic procedure for partitioning graphs, *Bell System Technical Journal* **49**, 291–307.

KLOKER, K.L. [1987], The architecture and applications of the Motorola DSP56000 digital signal processor family, *Proceedings of the 1987 International Conference on Acoustics, Speech, and Signal Processing*, 523–526.

KOCH, P., K.K. BAGCHI, AND K. HERMANSEN [1993], Implementation studies of efficient and realistic multi-signal processor solutions for DSP applications, *DSP - The enabling technology for communications. Conference Proceedings (ERA 93-0008)*, 7.3/1–10.

KOCK, E.A. DE, E.H.L. AARTS, AND G. ESSINK [1998], Real-time scheduling in video systems, *Parallel and Distributed Computing Practices* **1**, 85–98.

KOCK, E.A. DE, E.H.L. AARTS, G. ESSINK, R.E.J. JANSEN, AND J.H.M. KORST [1995], A variable-depth search algorithm for the recursive bipartitioning of signal flow graphs, *OR Spektrum* **17**, 159–172.

KORST, J.H.M. [1992], *Periodic Multiprocessor Scheduling*, Ph.D. thesis, Eindhoven University of Technology.

LANNEER, D., J. VAN PRAET, A. KIFLI, K. SCHOOFS, W. GEURTS, F. THOEN, AND G. GOOSSENS [1995], Chess: Retargetable code generation for embedded DSP processors, in: P. Marwedel and G. Goossens (eds.), *Code generation for embedded processors*, Kluwer Academic Publishers.

LAWLER, E.L. [1976], *Combinatorial optimization: networks and matroids*, Holt, Rinehart and Winston, New York.

LEE, E.A. [1991], Consistency in dataflow graphs, *IEEE Transactions on Parallel and Distributed Systems* **2**, 223–235.

LEE, E.A., AND J.C. BIER [1990], Architectures for statically scheduled dataflow, *Journal of Parallel and Distributed Computing* **10**, 333–348.

LENGAUER, T. [1990], *Combinatorial algorithms for integrated circuit layout*, Wiley, Chichester.

LEVIN, L.A. [1973], Universal sorting problems, *Problems of information transmission* **9**, 265–266.

LIU, C.L., AND J.W. LAYLAND [1973], Scheduling algorithms for multiprogramming in a hard-real-time environment, *Journal of the Association for Computing Machinery* **20**, 46–61.

MARWEDEL, P., AND G. GOOSSENS [1995], *Code generation for embedded processors*, Kluwer Academic Publishers.

MEERBERGEN, J.L. VAN, P.E.R. LIPPENS, W.F.J. VERHAEGH, AND A. VAN DER WERF [1995], Phideo: high-level synthesis for high throughput applications, *Journal of VLSI Processing* **9**, 89–104.

MESMAN, B., M. STRIK, A.H. TIMMER, J.L. VAN MEERBERGEN, AND J.A.G. JESS [1998], A constraint driven approach to loop pipelining and register binding, *Proceedings 1998 European Design Automation and Test Conference*, 377–383.

MONTANARI, U. [1974], Networks of constraints: fundamental properties and applications to picture processing, *Information Sciences* **7**, 95–132.

NEMHAUSER, G.L., AND L.A. WOLSEY [1988], *Integer and combinatorial optimization*, Wiley, New York.

NUIJTEN, W.P.M. [1994], *Time and resource constrained scheduling: a constraint satisfaction approach*, Ph.D. thesis, Eindhoven University of Technology.

PALENICHKA, R.M., AND A.Y. LUTSYK [1996], Parallel image processing by using homogeneous computing structures, *Proceedings of the 3rd International Conference of the ACPC Parallel Databases and Parallel I/O*, 233–234.

PAPADIMITRIOU, C.H., AND K. STEIGLITZ [1982], *Combinatiorial optimization: algorithms and complexity*, Prentice-Hall, New Jersey.

PARKS, T.M., J.L. PINO, AND E.A. LEE [1996], A comparison of synchronous and cyclo-static dataflow, *Proceedings of the 29th Asimolar Conference on Signal, Systems and Computers*, 204–210.

PAULIN, P.G., AND J.P. KNIGHT [1989], Force-directed scheduling for the behavioral synthesis of ASICs, *IEEE Transactions on Computer-Aided Design* **8**, 661–679.

PAULIN, P.G., C. LIEM, T.C. MAY, AND S. SUTARWALA [1995], FlexWare: A flexible firmware development environment for embedded systems, in: P. Marwedel and G. Goossens (eds.), *Code generation for embedded processors*, Kluwer Academic Publishers.

PINEDO, M. [1995], *Scheduling: theory, algorithms, and systems*, Prentice-Hall, Englewood Cliffs.

PRATT, W.K. [1991], *Digital image processing 2nd edition*, Wiley, New York.

RIDDERSMA, E., G.D. LA HEI, AND A.K. RIEMENS [1996], *The VSP2flex board*, Technical report, Philips Research NL-TN 024/96.

ROERMUND, A.H.M. VAN, P.J. SNIJDER, H. DIJKSTRA, C.G. HEMERYCK, C.M. HUIZER, J.M.P. SCHMITZ, AND R.J. SLUYTER [1989], A general-purpose programmable video processor, *IEEE Transactions on Consumer Electronics* **35**, 249–258.

SAHA, A., AND R. KRISNAMURTHY [1994], Some design issues in multi-chip FPGA implementation of DSP algorithms, *Proceedings 5th International Workshop on Rapid System Prototyping*.

SCHÄFFER, A.A., AND M. YANNAKAKIS [1991], Simple local search problems that are hard to solve, *SIAM Journal on Computing* **20**, 56–87.

SCHRIJVER, A. [1986], *Linear and integer programming*, Wiley, Chichester.

SHEU, J.P., AND T.S. CHEN [1995], Partitioning and mapping of nested loops for linear array multicomputers, *Journal of Supercomputing* **9**, 183–202.

SMEETS, M.L.G., E.H.L. AARTS, G. ESSINK, AND E.A. DE KOCK [1997], Delay management for programmable video signal processors, *Proceedings European Design & Test Conference 1997*, 126–133.

SOHIE, G.R.L. [1989], The Motorola DSP96002 IEEE floating-point digital signal processor, *22nd Asilomar Conference on Signals, Systems and Computers*, 909–913.

STEWART, R.W. [1988], Mapping signal processing algorithms to fixed architectures, *1988 International Conference on Acoustics, Speech and Signal Processing*, 2037–2040.

THEIS, J.P. [1996], *Parallel processor architectures for image processing*, Ph.D. thesis, University of Saarland.

TIMMER, A.H., AND J.A.G. JESS [1993], Execution interval analysis under resource constraints, *Proceedings of the International Conference on Computer Aided Design*, 454–459.

TREGNAGO, R., A.H.M. VAN ROERMUND, AND A.A.J. DE LANGE [1992], *A general-purpose VSP-based digital video processing board*, Technical report, Philips Research NL-TN 039/92.

VEENDRICK, H.J.M., O. POPP, G. POSTUMA, AND M. LECOUTERE [1994], A 1.5 GIPS video signal processor (VSP), *Proceedings CICC 6.2*, 95–98.

VERHAEGH, W.F.J. [1995], *Multidimensional periodic scheduling*, Ph.D. thesis, Eindhoven University of Technology.

VISSERS, K.A., G. ESSINK, P.H.J. VAN GERWEN, P.J.M. JANSSEN, O. POPP, E. RIDDERSMA, W.J.M. SMITS, AND H.J.M. VEENDRICK [1995], Architecture and programming of two generations video signal processors, *Microprocessing and Microprogramming* **41**, 373–390.

WELSH, D.J.A., AND M.B. POWELL [1967], An upper bound on the chromatic

number of a graph and its application to timetabling problems, *The Computer Journal* **10**, 85–87.

WESS, B., W. KREUZER, AND M. GOTSCHLICH [1995], Automatic generation of optimized DSP assembly code, *Proceedings of the 21st Annual Conference on IEEE Industrial Electronics* **2**, 979–984.

WILSON, T., G. GREWAL, S. HENSHALL, AND D. BANERJI [1995], An ILP-based approach to code generation, in: P. Marwedel and G. Goossens (eds.), *Code generation for embedded processors*, Kluwer Academic Publishers.

YEUNG, A., AND J. RABAEY [1992], A data-driven architecture of rapid prototyping of high throughput DSP algorithms, *Proceedings VLSI Signal Processing*, 225–234.

Symbol Index

The numbers refer to the pages of first occurrence.

General Symbols

$\mathbb{B}$	set of booleans	46
$\mathbb{Z}$	set of integers	16
$\mathbb{Z}_+$	set of positive integers	20
$\mathbb{N}$	set of non-negative integers	16
$\mathbb{N}_k$	set of k smallest non-negative integers	25

Terminal Symbols

T_t	set of terminal types	15

Processing Element Type Symbols

T_p	set of processing element types	16
$I_\mathrm{p}(t)$	set of input terminals of processing element type t	16
$O_\mathrm{p}(t)$	set of output terminals of processing element type t	16
$t_\mathrm{p}(t,n)$	type of terminal n of processing element type t	16
$d_\mathrm{p}(t,n)$	computation delay of terminal n of processing element type t	16
$s_\mathrm{p}(t)$	size of random access memory of processing element type t	16

Processor Type Symbols

T_v	set of processor types	17
$I_\mathrm{v}(t)$	set of input ports of processor type t	17
$O_\mathrm{v}(t)$	set of output ports of processor type t	17
$M_\mathrm{v}(t)$	set of program memories of processor type t	17
$p_\mathrm{v}(t)$	size of program memories of processor type t	17
$P_\mathrm{v}(t)$	set of processing elements of processor type t	17
$t_\mathrm{v}(t,e)$	type of processing element e of processor type t	17
$m_\mathrm{v}(t,e)$	program memory of processing element e of processor type t	17
$C_\mathrm{v}(t)$	set of intraprocessor connections of processor type t	17

$d_{\mathrm{v}}(t)$	communication delay of processor type t	17
$n_{\mathrm{v}}(t)$	silo size of processor type t	17

Processor Network Symbols

P_{n}	set of processors	17
$t_{\mathrm{n}}(v)$	type of processor v	17
C_{n}	set of interprocessor connections	17
$d_{\mathrm{n}}(v_1, v_2)$	communication delay between processors v_1 and v_2	17

Processing Element Network Symbols

M	set of program memories	18
$p_{\mathrm{a}}(m)\ (= p(m))$	size of program memory m	18
P	set of processing elements	18
$m_{\mathrm{a}}(p)\ (= m(p))$	program memory of processing element p	18
$n_{\mathrm{a}}(p)\ (= n(p))$	silo size of processing element p	18
$t_{\mathrm{a}}(p)\ (= t(p))$	type of processing element p	18
$s_{\mathrm{a}}(p)\ (= s(p))$	memory size of processing element p	18
C	set of processing element connections	18
$d_{\mathrm{a}}(c)\ (= d(c))$	communication delay of processing element connection c	18

Operation Type Symbols

T_{o}	set of operation types	19
$I_{\mathrm{o}}(t)$	set of input terminal of operation type t	19
$O_{\mathrm{o}}(t)$	set of output terminal of operation type t	19
$t_{\mathrm{o}}(t, n)$	type of terminal n of operation type t	19

Signal Flow Graph Symbols

A	set of arrays	20
$s_{\mathrm{f}}(a)\ (= s(a))$	size of array a	20
O	set of operations	20
$t_{\mathrm{f}}(o)\ (= t(o))$	type of operation o	20
$a_{\mathrm{f}}(o)\ (= a(o))$	array of operation o	20
$p_{\mathrm{f}}(o)\ (= p(o))$	period of operation o	20
R	set of data precedences	20
S	set of no-value precedences	20

Mapping Symbols

$\delta : O \to \mathbb{N}$	delay assignment	25
$\sigma : O \to \mathbb{Z}$	time assignment	25
$\alpha : O \to P$	processing element assignment	25
$\lambda : R \to C$	channel assignment	25
$\chi : R \cup S \to \mathbb{Z}$	data departure time	29
$\psi : R \cup S \to \mathbb{Z}$	data arrival time	29
$\underline{\omega} : R \cup S \to \mathbb{Z}$	minimum completion time difference	30
$\overline{\omega} : R \cup S \to \mathbb{Z}$	maximum completion time difference	30

Mapping Set Symbols

$\Delta \subseteq O \to \mathbb{N}$	set of delay assignments	54
$\Sigma \subseteq O \to \mathbb{Z}$	set of time assignments	54
$\mathrm{A} \subseteq O \to P$	set of processing element assignments	54
$\Lambda \subseteq R \to C$	set of channel assignments	54
$\Delta(o) \subseteq \mathbb{N}$	set of delay assignment values of operation o	54
$\Sigma(o) \subseteq \mathbb{Z}$	set of time assignment values of operation o	54
$\mathrm{A}(o) \subseteq P$	set of processing element assignment values of operation o	54
$\Lambda(r) \subseteq C$	set of channel assignment values of data precedence r	54
$\Delta(O) \subseteq \mathcal{P}(\mathbb{N})$	set of delay assignment value sets	54
$\Sigma(O) \subseteq \mathcal{P}(\mathbb{Z})$	set of time assignment value sets	54
$\mathrm{A}(O) \subseteq \mathcal{P}(P)$	set of processing element assignment value sets	54
$\Lambda(R) \subseteq \mathcal{P}(C)$	set of channel assignment value sets	54
$\mathrm{X} \subseteq R \cup S \to \mathbb{Z}$	set of data departure times	55
$\Psi \subseteq R \cup S \to \mathbb{Z}$	set of data arrival times	55
$\underline{\Omega} \subseteq R \cup S \to \mathbb{Z}$	estimated minimum completion time difference	56
$\overline{\Omega} \subseteq R \cup S \to \mathbb{Z}$	estimated maximum completion time difference	56
$R_{\Delta}(O)$	storage requirement of operation set O	59
$R_{\mathrm{A}}(O)$	computation requirement of operation set O	57
$R_{\Lambda}(R)$	communication requirement of data precedence set R	58
$R_{\mathrm{A},\Lambda}(O)$	periodicity requirement of operation set O	59
$C_{\Delta}(P)$	storage capacity of processing element set P	59

$C_{\mathrm{A}}(P)$	computation capacity of processing element set P	57
$C_{\Lambda}(C)$	communication capacity of channel set C	58
$C_{\mathrm{A},\Lambda}(P)$	periodicity capacity of processing element set P	59

Author Index

A
Aarts, E.H.L., 7, 100, 116
Aho, A.V., 35
Ahuja, R.K., 76, 78, 80, 81
Araujo, G., 9
Ashraf, M., 9

B
Bagchi, K.K., 9
Balmer, K., 8
Banerjee, U., 9
Banerji, D., 8
Barr, G., 8
Bass, S.C., 9
Bier, J.C., 8
Bilsen, G., 23
Bokhari, S.H., 9
Bove, V.M., 8
Buck, J.T., 8

C
Cavigioli, C.D., 8
Chang, P., 9
Chen, D.Y., 9
Chen, T.S., 9
Chen, Y., 9
Cheng, W.K., 9
Coffman, E.G. Jr., 9, 40, 85
Cook, S.A., 35, 46, 49
Cook, W.J., 7
Cunningham. W.H., 7

D
Denk, T.C., 9
Depuydt, F., 9
Desmet, D., 8
Dijkstra, H., 14, 15
Dongen, R.C.A. van, 7, 116

E
Engels, M., 23
Essink, G., 4, 7, 116

F
Fluiter, B.L.E. de, 48
French, S., 9

G
Garey, M.R., 8, 35, 39, 40, 44, 85, 99
Gebotys, C.H., 9
Genin, G., 8
Gerwen, P.H.J. van, 4, 7, 116
Geurts, W., 8
Gibbons, A., 116, 128
Goossens, G., 8, 9
Gotschlich, M., 8
Grewal, G.W., 8
Gumuskaya, H., 9

H
Hammes, M., 8
Harber, R.G., 9
Hei, G.D. La, 32
Hemeryck, C.G., 15
Henshall, S., 8
Hermansen, K., 9
Hollmann, H., 14
Hopcroft, J.E., 35

Hsu, Y., 9
Hu, X., 9
Huizer, C.M., 14, 15
Hwang, J.S., 8
Hwang, Y.T., 8

J

Jansen, R.E.J., 7, 97
Janssen, P.J.M., 4
Jess, J.A.G., 8
Johnson, D.S., 8, 35, 39, 40, 44, 85, 99, 101, 104

K

Kalouptsidis, N., 8
Karp, R.M., 35, 48, 116
Keay, J., 8
Kempe, A.B., 116, 128
Kernighan, B.W., 100
Kifli, A., 8
King, C., 9
Kloker, K.L., 8
Knight, J.P., 123
Koch, P., 9
Kock, E.A. de, 7
Korst, J.H.M., 7–9, 20, 40, 42–44, 116, 123
Kreuzer, W., 8
Krisnamurthy, R., 9
Kurugolla, T., 9

L

Lange, A.A.J. de, 31
Lanneer, D., 8
Lauwereins, R., 23
Lawler, E.L., 44
Layland, J.W., 122
Lecoutere, M., 15
Lee, E.A., 8, 23
Lee, Y.F., 9
Lengauer, T., 9, 93
Lenstra, J.K., 100
Levin, L.A., 35
Liem, C., 8
Lin, S., 100
Lin, Y.L., 9
Lippens, P.E.R., 4
Liu, C.L., 122
Lutsyk, A.Y., 9

M

Magnanti, T.L., 76, 78, 80, 81
Malik, S., 9
Man, H. de, 8, 9
Marwedel, P., 8
May, T.C., 8
Meerbergen, J.L. van, 4, 8
Mesman, B., 8
Miller, G.L., 44
Montanari, U., 118
Moyse, P., 8

N

Nemhauser, G.L., 7
Nuijten, W.P.M., 118

O

Oakland, E., 8
Orencik, B., 9
Orlin, J.B., 76, 78, 80, 81

P

Palaz, T., 9
Palenichka, R.M., 9
Papadimitriou, C.H., 7, 44, 78, 101, 104
Parhi, K., 9
Parks, T.M., 23
Paulin, P.G., 8, 123
Peperstraete, J., 23
Pinedo, M., 9
Pino, J.L., 23

Popp, O., 4, 15
Postuma, G., 15
Powell, M.B., 116, 130
Praet, J. van, 8
Pratt, W.K., 1
Pulleyblank, W.R., 7

R

Rabaey, J., 8
Riddersma, E., 4, 32
Riemens, A.K., 32
Robertson, I., 8
Roermund, A.H.M. van, 15, 31
Roskell, D., 8

S

Saha, A., 9
Schäffer, A.A., 104
Schmitz, J.M.P., 15
Schoofs, K., 8
Schrijver, A., 7
Sheu, J.P., 9
Simmons, N., 8
Simpson, R., 8
Sluyter, R.J., 14, 15
Smeets, M.L.G., 7
Smits, W.J.M., 4
Snijder, P.J., 15
Sohie, G.R.L., 8
Steiglitz, K., 7, 78
Stewart, R.W., 9
Stockmeyer, L., 99
Strik, M., 8
Sutarwala, S., 8

T

Theis, J.P., 8
Thoen, F., 8
Timmer, A.H., 8
Tregnago, R., 31

U

Ullman, J.D., 35

V

Vandewalle, J., 8
Veendrick, H.J.M, 4, 15
Verhaegh, W.F.J., 4, 8, 9, 20
Vissers, K.A., 4, 7, 116

W

Watlington, J.A., 8
Welsh, D.J.A., 116, 130
Werf, A. van der, 4
Wess, B., 8
Wilson, T.C., 8
Wolsey, L.A., 7
Wu, Y., 9

Y

Yannakakis, M., 101, 104
Yeung, A., 8

Subject Index

A
amplitude, 2
array, 18
array constraints, 27
  relaxation, 60

B
bin packing, 7, 39
bipartite operation set, 98
bipartite processing element set, 97

C
capacity ratio, 97
channel assignment, 25
channel assignment problem, 117, 130
circular-arc graph, 43
combinatorial decision problem, 36
combinatorial optimization problem, 36
communication constraints, 31
  relaxation, 58
communication requirement, 105, 125
compact silo, 13, 26, 27
completion time order, 124
computation constraints, 31
  relaxation, 57
computation requirement, 123
concise certificate, 36
conflict graph
  channel, 131
  processing element, 127
connectivity constraints, 28
  relaxation, 60
consistency checking, 119
constraint satisfaction, 7, 8, 118
constraint satisfaction problem, 118
constraints
  array, 27, 60
  communication, 31, 58
  computation, 31, 57
  connectivity, 28, 60
  periodicity, 28, 59
  precedence, 29, 30, 56
  storage, 28, 58
  type, 27, 60
cyclostatic, 4, 6, 10

D
dead end, 119
decimation, 24
delay assignment, 25
delay assignment problem, 75
delay line, 82
delay management problem, 65, 71
delay minimization problem, 73
dual of minimum cost flow problem, 78

E
exchange, 101
expansion, 24

F
feasibility problem, 37

first fit decreasing, 85, 86
force-directed scheduling, 123
frame, 3
frame blanking, 3

G

genetic algorithms, 8
graph coloring, 7, 116

I

instance size, 36
integer linear programming, 8, 76
integrated circuit, 2
    application-specific, 4
interpolation, 24

K

kempe-chain argument, 116, 128, 132

L

line, 3
line blanking, 3
list scheduling, 8
local search, 7
local search problem, 100
local search reduction, 102

M

mapping, 25
mapping problem, 31
mapping set, 54
minimum cost flow problem, 77

N

network
    processing element, 18
    processor, 17
network flow, 7
NP-complete, 37
NP-hard, 37

O

operation, 18
    arithmetic, 19
    constant, 19
    input, 19
    logic, 19
    output, 19
    pass, 19
    read, 19
    shift, 19
    type, 19
    write, 19
operation order, 123

P

partition, 96
partitioning problem, 66, 94
periodicity constraints, 28
    relaxation, 59
pixel, 2
polynomial-time algorithm, 36
precedence, 18
    data, 18
    equivalence, 22
    inclusion, 22
    label, 21
    no-value, 18
    offset, 21
    period, 21
precedence constraints, 29, 30
    relaxation, 56
processing element, 15
    type, 16
processing element assignment, 25
processing element assignment problem, 117, 127
processing element network
    abstraction, 95
processor, 16
    general-purpose, 3

micro, 3
signal, 3
type, 17
video, 3
processor assignment problem, 96

Q

quantization, 2

R

rate-monotonic scheduling, 122
reduction, 24, 36, 102
right-hand-side scaling, 80

S

sampling, 2
scanning, 3
scheduling problem, 66, 115
search state, 119
search tree, 118
sequential coloring, 116
signal
analog, 2
continuous, 2
digital, 2
discrete, 2
processing, 1
processor, 3
video, 2
signal flow graph, 5, 20
inclusion, 25
solution equivalence, 119
solution set, 119
storage constraints, 28
relaxation, 58
successive shortest path, 80

T

terminal, 15
terminal type, 15
thickness function, 123
three-satisfiability, 45
time assignment, 25
time assignment problem, 117
time complexity function, 36
two-way partitioning problem, 99, 103
two-way partitioning scheme, 94
type assignment problem, 96
type constraints, 27
relaxation, 60

U

uniform graph partioning, 103

V

variable-depth search, 101

Samenvatting

Dit proefschift handelt over het afbeelden van video-applicaties op programmeerbare videosignaalprocessoren. Het afbeeldingsprobleem vormt de kern van het programmeringsprobleem om een netwerk van processoren een gewenste video-applicatie te laten uitvoeren. Kenmerkend voor de processornetwerken is de hoge mate van parallellisme die nodig is vanwege de aard van het applicatiegebied. Om dit te bereiken zijn speciale processorarchitecturen noodzakelijk waardoor de complexiteit van de afbeeldingsproblematiek toeneemt. We behandelen het afbeeldingsprobleem voor een specifieke architectuur die toegesneden is op stroomgeoriënteerde berekeningen.

Het afbeeldingsprobleem laat zich modelleren als een combinatorisch beslissingsprobleem waarvan de oplossingsruimte bestaat uit alle afbeeldingen die voldoen aan beperkingen die voortkomen uit de processorarchitectuur en de tijdseisen van het applicatiegebied. We hebben aangetoond dat de verschillende combinaties van beperkingen tot gevolg hebben dat het afbeeldingsprobleem formeel lastig is. Om met de beperking om te kunnen gaan, decomponeren we het afbeeldingsprobleem in drie deelproblemen genaamd het bufferprobleem, het routeringsprobleem en het planningsprobleem zodanig dat de oplossingsruimte zo weinig mogelijk wordt aangetast. Hoewel deze deelproblemen ook formeel lastig zijn, worden er in de bestaande literatuur technieken beschreven die goed toepasbaar zijn op deze deelproblemen.

De decompositie is gebaseerd op een eigenschap van de processorarchitectuur die kenmerkend is voor stroomgeoriënteerde berekeningen. Deze eigenschap houdt in dat informatie slechts korte tijd bij een operator aanwezig is, om zodoende geheugen te besparen. Dit heeft twee consequenties. Ten eerste is het tijdsinterval waarin operaties uitgevoerd kunnen worden klein. Ten tweede zijn opeenvolgende operaties sterk in de tijd aan elkaar gerelateerd. Dit heeft tot gevolg dat het routeren van de informatiestromen tussen de operatoren een sterke invloed heeft op de absolute positie van de tijdsintervallen. Bovendien vraagt de synchronisatie van reconvergente informatiestromen om buffers hetgeen sterk van invloed is op het routeren.

Het bufferprobleem handelt over het synchroniseren van reconvergente informatiestromen. Daarbij zijn de tijdstippen van synchronisatie en de opslag van informatie tussen deze tijdstippen van belang. Dit geeft aanleiding tot

een decompositie van het bufferprobleem in een bufferminimaliseringsprobleem en een buffertoekenningsprobleem. In het minimaliseringsprobleem bepalen we tijdstippen van synchronisatie zodanig dat de totale geheugenbehoefte minimaal is. Dit probleem laat zich modelleren als een 'network flow' probleem, dat in polynomiale tijd oplosbaar is. In het toekenningsprobleem bepalen we hoe iedere informatiestroom in de beschikbare geheugentypes wordt opgeslagen. Dit probleem laat zich modelleren als een 'bin packing' probleem, waarvoor goede benaderingsalgoritmen in de literatuur bekend zijn.

Het routeringsprobleem handelt over het routeren van de informatiestromen tussen de processoren in een netwerk. Daarbij zijn de toekenning van operaties aan processoren en het transport van informatie tussen de processoren van belang. Dit geeft aanleiding tot een decompositie van het routeringsprobleem in een sequentie van tweedelingsproblemen. De sequentie is zodanig dat iedere verbinding in het processornetwerk eenmaal doorsneden wordt. Voor iedere tweedeling in het processornetwerk bepalen we een corresponderende tweedeling in de video-applicatie met behulp van een 'local search' methode. In deze aanpak garandeert de keuze van zoekruimte dat er met de capaciteit van de processoren rekening wordt gehouden. De kostenfunctie representeert de communicatiebehoefte tussen de beide elementen van een tweedeling.

Het planningsprobleem handelt over het toekennen van operaties aan operatoren en aan tijdstippen zodanig dat operaties die overlappen in tijd op verschillende operatoren uitgevoerd worden. Het toekenningsprobleem van operaties aan operatoren laat zich voor een gegeven tijdstoekenning modelleren als een graafkleuringsprobleem, waarvoor heuristieken in de literatuur bekend zijn. Dit geeft aanleiding tot een 'constraint satisfaction' aanpak waarin de tijdsintervallen van de operaties, die het resultaat zijn van het bufferen en het routeren, gereduceerd worden op grond van de tijdsrelaties tussen de operaties en het resultaat van de graafkleuring.

De ontwikkelde afbeeldingsmethode is toegepast op een twintigtal industrieel relevante video-applicaties. De resultaten laten zien dat men applicaties die enkele honderden operaties bevatten in enkele minuten automatisch af kan beelden op processornetwerken die tientallen operatoren bevatten, waarbij men hoge bezettingsgraden haalt. Het handmatig afbeelden van deze applicaties neemt vaak enkele tot tientallen uren in beslag. In combinatie met een interactieve programmeeromgeving is de ontwikkelde methode een effectief afbeeldingsgereedschap.

Curriculum Vitae

Erwin de Kock was born on May 12, 1970, in Tilburg, the Netherlands. From 1988 to 1993 he studied Computing Science at the Eindhoven University of Technology. He graduated on the subject of partitioning of signal flow graphs for networks of video signal processors. Subsequently he participated in a two-year's post-master's program on software technology at the Eindhoven University of Technology from which he graduated on the design of a programming environment for video signal processors. Since October 1995 he has been affiliated to the Philips Research Laboratories, where he has been working on methods and tools for the design and programming of digital signal processing systems. He combines this work with his interests in combinatorial optimization and software engineering.