

COMMUNICATION-CONSCIOUS MAPPING OF REGULAR NESTED LOOP PROGRAMS ONTO MASSIVELY PARALLEL PROCESSOR ARRAYS

Sebastian Siegel and Renate Merker
Institute of Circuits and Systems
Dresden University of Technology
01062 Dresden, Germany
email: {siegel, merker}@iee1.et.tu-dresden.de

Frank Hannig and Jürgen Teich
Department of Computer Science 12
University of Erlangen-Nuremberg
91058 Erlangen, Germany
email: {hannig, teich}@cs.fau.de

ABSTRACT

Methods for an efficient mapping of algorithms to parallel architectures are of utmost importance because many state-of-the-art embedded digital systems deploy parallelism to increase their computational power. This paper deals with the mapping of loop programs onto processor arrays implemented in an FPGA or available as (reconfigurable) coarse-grained processor architectures.

Most existing work is closely related to approaches from the DSP domain and is not able to exploit the full parallelism of a given algorithm and the computational potential of a typical 2-dimensional array. In contrast, we present a mapping methodology which incorporates many important parameters of the target architecture in one approach. These are: number of processing elements, resources of the data path and memory within a processing element, and interconnection within the processor array. Based on these parameters, we formulate an optimization problem whose solution specifies an efficient mapping of an algorithm to the target architecture. We can optimize for speed of the algorithm and/or hardware cost caused by the communication and computation resources of the architecture.

KEY WORDS

Processor Array, Interconnection, Data Path, Hardware Mapping, Integer Linear Programming.

1 Introduction

The desire for more mobility and the enthusiasm for ubiquitous electronic gadgets on the one hand side and the unbowed progress in semiconductor industry on the other hand are driving forces in the market of embedded digital systems. The growing number of stream-based applications is eager for computational power. Examples for such systems include handhelds for digital audio and video broadcasting, next generation game and entertainment consoles with high-definition television (HDTV) support and high-capacity storage media like HD DVD or Blu-ray Disc, or sophisticated applications in medical image processing and radar technology with real-time requirements. In order to handle the different requirements on performance, power consumption, and cost, increasingly more and more unconventional processors are developed which consist of an

array of processors. Unfortunately, currently there barely exist mapping tools which are able to focus the hardware complexity of such devices. Hence, parallelization techniques and compilers will be of utmost importance in order to map computationally intensive algorithms efficiently to these processor arrays.

In this context, our paper deals with the specific problem of mapping a certain class of regular nested loop programs onto a dedicated processor array with resource constraints. This work can be classified to the area of loop parallelization in the polytope model [1].

The contributions of our paper can be summarized as follows:

1. Exploitation of the resources of the data path for a minimal computation time of the algorithm (Section 4.1).
2. Efficient routing within the processor array with minimal communication cost caused by data dependencies (Section 4.2).
3. Combined approach for 1. and 2. with an optimal schedule and allocation for all operations of the algorithm (Section 4.3).

The novelty of our work is characterized by a model which considers the resources of the target architecture for communication and computation purposes in a combined approach.

The rest of this paper is organized as follows. In Section 2, we give an overview of related work and the architectures we are dealing with. Afterwards, in Section 3, we specify the problem that we solve in Section 4. Finally, in Section 5, we draw some conclusions.

2 Related Work and Background

Much research effort was spent on the field of coarse-grained architectures and resulted in academic system prototypes as well as commercially available systems. A detailed overview of some academic coarse-grained architectures can be found for example in [4]. Well-known commercial architectures include the D-Fabrix [5], the DRP from NEC [6], the PACT XPP [7], Bresca from Silicon

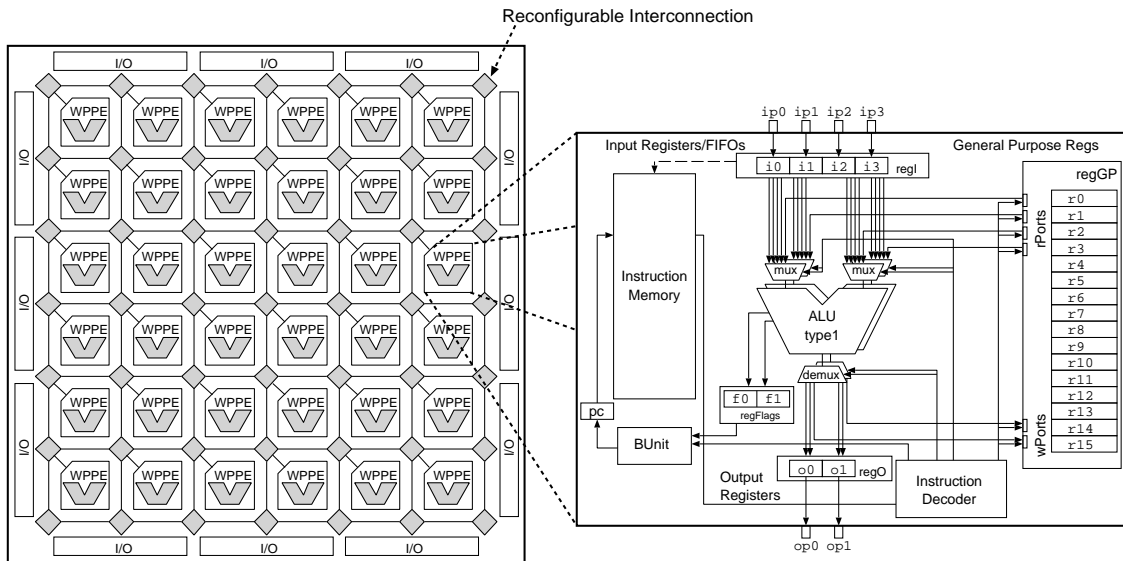


Figure 1. Example of a 6×6 weakly-programmable processor array (WPPA) with reconfigurable interconnection [2, 3].

Hive (Philips) [8], or QuickSilver Technology’s Adaptive Computing Machine (ACM) [9]. Other concepts combine reconfigurability with *weak-programmability* as depicted in Fig. 1. Such a so-called *weakly-programmable processor arrays* (WPPA) consist of an array of $\vartheta_1 \times \vartheta_2$ processing elements (PE) that contain several processing units with only very few memory and regular interconnect structures. In order to efficiently implement a certain algorithm, each PE may implement only a certain function range. The PEs are called weakly-programmable because the control overhead of each PE is optimized and kept small. The massively parallelism might be expressed by different types of parallelism: (1) several parallel working weakly-programmable processing elements (WPPEs), (2) functional and software pipelining, and (3) multiple functional units within one WPPE.

Only few research work is published which deals with the compilation to coarse-grained reconfigurable architectures. The authors in [10] describe a compiler framework to analyze SA-C programs, perform optimizations, and automatically map the application onto the MorphoSys architecture [11], a row-parallel or column-parallel SIMD (Single Instruction-stream Multiple Data-Stream) architecture. This approach is limited since the order of the synthesis is predefined by the loop order and no data dependencies between iterations are allowed. Another approach for mapping loops onto coarse-grained reconfigurable architectures is presented by Dutt et al. in [12]. Outstanding in their compilation flow is the target architecture, the DRAA, a generic reconfigurable architecture template which can represent a wide range of coarse-grained reconfigurable arrays. The mapping technique itself is based on loop pipelining and partitioning of the program tree into clusters which can be placed on a line of the array.

Within the scope of processor array design, the interconnection between the processing elements is taken into account in [13]. The data path design is regarded in detail in [14]. Comparable models for the usage of registers are presented in [15, 16] for a single processor machine.

3 Problem Definition

In preparation of the problem definition, first, we introduce the algorithm class which we consider: *systems of uniform recurrence equations (SURE)*. Afterwards, we briefly describe the parameterized partitioning method to map an algorithm to a WPPA. Finally, we formulate the problem that we solve in this paper.

Definition 1 (SURE) *A system of uniform recurrence equations consists of J statements S_1 to S_J of the form*

$$S_j : y_j[\mathbf{i}] := f_j(\dots, y_i[\mathbf{i} - \mathbf{d}_{j,i}], \dots), \forall \mathbf{i} \in \mathcal{I}_j, i, j \in \mathbb{N},$$

where \mathbf{i} denotes an iteration in the iteration space \mathcal{I}_j of statement S_j , \mathcal{I}_j is a convex polytope and f_j denotes a single-valued function. A variable y_i that is computed by statement S_i is a dependent variable if it is input to some (other) statement S_j . Vector $\mathbf{d}_{j,i}$ denotes the corresponding uniform data dependency.

The number of instances of statement S_j is given by the number of points (iterations) of iteration space \mathcal{I}_j . The convex hull $\mathcal{I} = \text{conv}(\bigcup_j \mathcal{I}_j) \subset \mathbb{Z}^n$ describes the common iteration space \mathcal{I} of the algorithm. In Algorithm 1, we show the edge detection algorithm (EDA) in the notation of a SURE as an example which we use to demonstrate our methods throughout this paper.

Algorithm 1: Edge detection algorithm (EDA)

$S_1 :$	$p \begin{bmatrix} x \\ y \end{bmatrix} = p_i \begin{bmatrix} x \\ y \end{bmatrix},$	$(x, y) \in \mathcal{I}_1$
$S_2 :$	$q \begin{bmatrix} x \\ y \end{bmatrix} = 2 \cdot p \begin{bmatrix} x-1 \\ y-1 \end{bmatrix},$	$(x, y) \in \mathcal{I}_2$
$S_3 :$	$h_1 \begin{bmatrix} x \\ y \end{bmatrix} = p \begin{bmatrix} x-1 \\ y-2 \end{bmatrix} + p \begin{bmatrix} x-1 \\ y-1 \end{bmatrix},$	$(x, y) \in \mathcal{I}_3$
$S_4 :$	$h_2 \begin{bmatrix} x \\ y \end{bmatrix} = h_1 \begin{bmatrix} x \\ y \end{bmatrix} + q \begin{bmatrix} x \\ y \end{bmatrix},$	$(x, y) \in \mathcal{I}_4 = \mathcal{I}_3$
$S_5 :$	$v_1 \begin{bmatrix} x \\ y \end{bmatrix} = p \begin{bmatrix} x-2 \\ y-1 \end{bmatrix} + p \begin{bmatrix} x-1 \\ y-1 \end{bmatrix},$	$(x, y) \in \mathcal{I}_5$
$S_6 :$	$v_2 \begin{bmatrix} x \\ y \end{bmatrix} = v_1 \begin{bmatrix} x \\ y \end{bmatrix} + q \begin{bmatrix} x \\ y \end{bmatrix},$	$(x, y) \in \mathcal{I}_6 = \mathcal{I}_5$
$S_7 :$	$h_3 \begin{bmatrix} x \\ y \end{bmatrix} = h_2 \begin{bmatrix} x-2 \\ y-1 \end{bmatrix} - h_2 \begin{bmatrix} x-1 \\ y-1 \end{bmatrix},$	$(x, y) \in \mathcal{I}_7$
$S_8 :$	$h_4 \begin{bmatrix} x \\ y \end{bmatrix} = h_3 \begin{bmatrix} x \\ y \end{bmatrix} ,$	$(x, y) \in \mathcal{I}_8 = \mathcal{I}_7$
$S_9 :$	$v_3 \begin{bmatrix} x \\ y \end{bmatrix} = v_2 \begin{bmatrix} x-1 \\ y-2 \end{bmatrix} - v_2 \begin{bmatrix} x-1 \\ y-1 \end{bmatrix},$	$(x, y) \in \mathcal{I}_9 = \mathcal{I}_7$
$S_{10} :$	$v_4 \begin{bmatrix} x \\ y \end{bmatrix} = v_3 \begin{bmatrix} x \\ y \end{bmatrix} ,$	$(x, y) \in \mathcal{I}_{10} = \mathcal{I}_7$
$S_{11} :$	$s \begin{bmatrix} x \\ y \end{bmatrix} = h_4 \begin{bmatrix} x \\ y \end{bmatrix} + v_4 \begin{bmatrix} x \\ y \end{bmatrix},$	$(x, y) \in \mathcal{I}_{11} = \mathcal{I}_7$
$S_{12} :$	$p_o \begin{bmatrix} x-2 \\ y-2 \end{bmatrix} = \min(255, s \begin{bmatrix} x \\ y \end{bmatrix}),$	$(x, y) \in \mathcal{I}_{12} = \mathcal{I}_7$

with

$$\begin{aligned} \mathcal{I}_1 &= \{(x, y) \in \mathbb{Z}^2 \mid 0 \leq x \leq N-1, 0 \leq y \leq M-1\}, & \mathcal{I}_2 &= \{(x, y) \in \mathbb{Z}^2 \mid 1 \leq x \leq N, 1 \leq y \leq M\}, \\ \mathcal{I}_3 &= \{(x, y) \in \mathbb{Z}^2 \mid 1 \leq x \leq N, 2 \leq y \leq M-1\}, & \mathcal{I}_5 &= \{(x, y) \in \mathbb{Z}^2 \mid 2 \leq x \leq N-1, 1 \leq y \leq M\}, \\ \mathcal{I}_7 &= \{(x, y) \in \mathbb{Z}^2 \mid 3 \leq x \leq N, 3 \leq y \leq M\} & \text{and } \mathcal{I} &= \{(x, y) \in \mathbb{Z}^2 \mid 0 \leq x \leq N, 0 \leq y \leq M\} \end{aligned}$$

In order to map an algorithm to a WPPA, one needs to: 1) determine an order of execution for all iterations (schedule), 2) assign a WPPE to each iteration where the computational tasks of that iteration take place (allocation). By application of a partitioning method, we obtain mappings that are characterized by high regularity which is desirable in the design of WPPAs [2, 3]. In this paper, we consider *locally parallel globally sequential (LPGS) partitioning* [17]. Other scheduling schemes can be regarded similarly.

LPGS partitioning is characterized by a tiling step which separates the iteration space into two-dimensional partitions (tiles) of size $\vartheta_1 \times \vartheta_2$, i. e., each partition can be mapped directly to the WPPA (allocation). Fig. 2 shows an excerpt of the tiled iteration space where the partitions are of size 4×4 . A (globally) sequential schedule is responsible for a temporal ordering of these partitions, where the start time of each partition is a multiple of the *iteration interval* λ which is defined as follows.

Definition 2 (Iteration Interval λ) *The iteration interval λ denotes the number of time steps between the beginning of two successive iterations. The time steps of an iteration interval are given by the set $\{0, 1, \dots, \lambda-1\}$.*

In this paper, we emphasize the characterization of the inner structure of partitions. Hence, we have to specify the time behavior inside a partition. The scheduling function $t_j^{\text{rel}}(\kappa)$ determines the *inner schedule*, i. e., the relative start time of statement S_j of an iteration at position $\kappa \in \mathbb{Z}^2$ ($0 \leq \kappa_i < \vartheta_i$, $i \in \{1, 2\}$) within an arbitrary partition. This start time t_j^{rel} is relative with respect to the start time of the corresponding partition which is determined by the globally sequential schedule.

$$t_j^{\text{rel}}(\kappa) = \tau^{\text{offs}} \cdot \kappa + b_j, \quad \tau^{\text{offs}} \in \mathbb{Z}^{1 \times 2}, b_j \in \mathbb{Z} \quad (1)$$

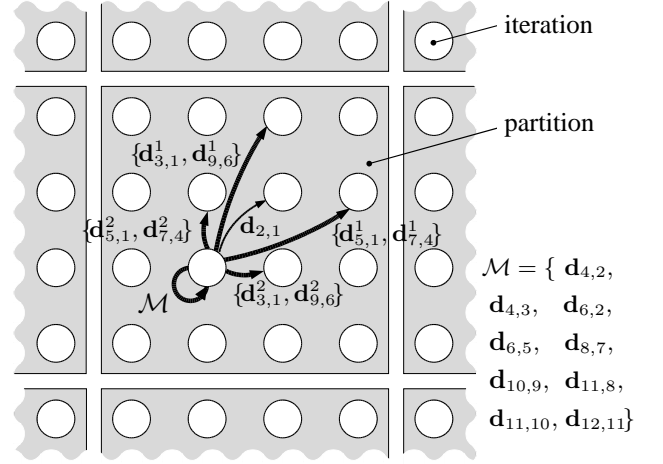


Figure 2. Extract of the tiled iteration space with partitions of size 4×4 ; all 18 dependencies of the EDA emerging from an iteration at the relative position $\kappa = (1 \ 1)^T$.

Term b_j is responsible for a temporal ordering of the statements S_j ($1 \leq j \leq J$) with respect to the time period given by the iteration interval λ . We call vector τ^{offs} *scheduling offset* which allows to shift the relative start time of an iteration at position κ within a partition. For example, if $\tau^{\text{offs}} = (0 \ 1)$, all WPPEs within a row run synchronously, but the schedule of the WPPEs from two adjacent rows differs by one time step.

The EDA (see Algorithm 1) comes with 18 data dependencies, as depicted in Fig. 2. Note that the optional upper index is used to distinguish between data dependencies that have the same source and sink. For example, vectors $\mathbf{d}_{7,4}^1$ and $\mathbf{d}_{7,4}^2$ represent the two different data dependencies between statements S_4 (source) and S_7 (sink). Each data dependency causes a data transfer of an instance of a dependent variable from the source WPPE to the sink WPPE. *Channels* between the WPPEs are used for the transfer and *registers* within the WPPEs are used for a temporary storage of a variable instance.

In summary, mainly three questions arise when mapping an algorithm to a WPPA as described above:

1. In what order and on what resource type should the statements of an iteration be executed?
2. By what scheduling offset τ^{offs} should the schedule among the WPPEs be shifted?
3. How are the data dependencies routed within the WPPA (transfer and storage)?

In the next section, we deal with a solution to these questions. For demonstration purposes, we illustrate our methods for a simple WPPA where we assume an interconnection between the WPPEs according to a grid topology (a vertical and a horizontal channel). The latency of each channel is said to be one clock cycle. Further, we assume that the data path of any WPPE consists of two similar ALUs, where each can perform any of the operations

from Algorithm 1 within one clock cycle. Note that our model can handle channels in any direction (e. g., diagonal). Further, we can deal with data paths which consist of different resources and pipelining is allowed.

4 Matching of Data Path and Communication Resources

To exploit the data path of each PE, we formulate an optimization problem (data path problem) which is described in Section 4.1. As a result, we obtain the minimal iteration interval λ_{\min} and an inner schedule. Next, in Section 4.2, we formulate an optimization problem (communication problem) with the objective to route the data dependencies of the algorithm with minimal cost for channels and registers. In Section 4.3, we deal with the combining of the data path problem and the communication problem.

4.1 Data Path Problem

The data path problem is formulated as a set of constraints for a mixed integer linear program (MILP) similar as in [18, 19]. In this MILP, the number of resources inside each processing element can be limited. Furthermore, the possibility that an operation can be mapped onto different resource types (module selection) is given, and pipelining is also possible. As a result of the MILP, we obtain:

- The minimal iteration interval λ_{\min} ,
- An according optimal inner schedule, i. e., scheduling offset τ^{offs} and relative start times b_j , see also (1),
- The selected resource type for each statement S_j .

In order to determine for each statement S_j a relative start time b_j and a resource type, we assume that the right-hand side expression of each statement is already split such that it consists only of one basic operation or assignment (this is already the case in Algorithm 1).

The basic principle of the MILP is based on the introduction of binary variables $x_{j,k,t}$, where $x_{j,k,t} = 1$ denotes that at relative time instance t statement S_j and function f_j respectively starts its operation on resource type $r_k \in V_T$. V_T denotes the set of different available resource types such as for instance an adder, multiplier, ALU, etc. By dint of the binary variables, data dependence and resource constraints are formulated (cf. [19]). Although the exact path of the transfer of an instance of a dependent variable will be specified later (communication problem), we already take the channel delays into account. Thereby, we assure that an instance of a dependent variable can be transferred on time to a WPPE where it is needed (causality).

4.2 Communication Problem

The minimal iteration interval λ_{\min} as determined according to Section 4.1 can always be achieved when unlim-

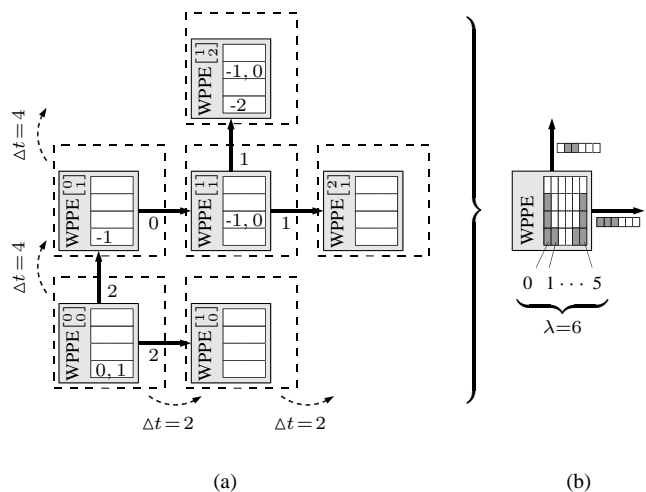


Figure 3. (a) Transfer and storage of dependent variable p , $\tau^{\text{offs}} = (2 \ 4)$, $\lambda = 6$, source: WPPE at position $(0 \ 0)^T$; (b) Usage of channels and registers mapped to one iteration interval (λ time steps) of one WPPE.

ited communication resources (channels and registers) are available. In this section, we describe how to minimize the communication resources that are required to route the data dependencies where we presume that the iteration interval λ and the scheduling offset τ^{offs} have already been determined. Our approach is also suitable for finding a routing of the data dependencies (if possible) where the communication resources of the WPPA are specified a priori.

A routing of a data dependency consists in moving an instance of a dependent variable successively from its source WPPE to the corresponding sink WPPE. The channels are used to move a variable instance from one WPPE to another and the registers within the WPPEs are used to store a variable instance temporarily, if necessary.

An example is illustrated in Fig. 3(a), where we show the time behavior of a routing of all five data dependencies emerging from statement S_1 of the EDA. We illustrate the transfer and storage of an instance of variable p (see Algorithm 1) from WPPE $[0^0]$ (i. e., the WPPE at position (0^0) within the WPPA) to five other WPPEs according to the corresponding data dependencies $\mathbf{d}_{2,1}^1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $\mathbf{d}_{3,1}^1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$, $\mathbf{d}_{3,1}^2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $\mathbf{d}_{5,1}^1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ and $\mathbf{d}_{5,1}^2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. Each WPPE has four registers (white rectangles). The numbers denote the time steps, when a channel or a register is used. For example, at WPPE $[0^0]$ a register is used at time steps 0 and 1.

To determine the usage of the communication resources within the WPPA, it is sufficient to model the usage of the outgoing channels of one arbitrary WPPE and the registers within that WPPE for one iteration interval. We formulate this approach as an integer linear program (ILP). We refer to [17] for a detailed derivation of that optimization problem. Here, we emphasize what the model is capable of and we outline some aspects that we need to deal with for an exact model.

When the scheduling offset τ^{offs} is a non-zero vector, the time steps that need to be considered when modeling the routing of a data dependency may exceed one iteration interval. Since the schedule inside each WPPE repeats with a period of λ , it is suitable to regard the time steps modulo λ . Thereby, the channel and register usage can be determined by considering one arbitrary iteration interval only. This is illustrated in Example 1.

Example 1 Let $\tau^{\text{offs}} = (2\ 4)$ and $\lambda = 6$. We consider a communication between $\text{WPPE} \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $\text{WPPE} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ via the vertical channel (see Fig. 3(a)). If an instance of a variable leaves $\text{WPPE} \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ at time step $t = 2$, that instance is available at the next time step at $\text{WPPE} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ due to the latency of one of the channel. Because of the scheduling offset, $\text{WPPE} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ is four time steps behind in comparison to $\text{WPPE} \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. Hence, the instance arrives at time step $t = 2 + 1 - 4 = -1$ at $\text{WPPE} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. In the solution shown in Fig. 3(a), that instance is stored for one time step. We consider the time step of this register usage with modulo λ . Hence, this storage causes a register usage at time step $t = -1 \bmod 6 = 5$. Note that the dashed rectangles in Fig. 3(a) show the zones whose time behavior is shifted according to the scheduling offset in comparison to other zones. In Fig. 3(b), we show the usage of channels and registers (gray filling) from Fig. 3(a) mapped to an iteration interval of an arbitrary WPPE.

Next, we consider redundancy in the routing of data dependencies and we show that by avoiding it, communication resources can be saved. Redundancy can appear, if one instance of a variable needs to be transferred to two or more WPPEs. Again, we illustrate this effect for the five data dependencies according to the dependent variable p of the EDA. An individual routing of each data dependency would require the usage of a channel for a total of ten times. For example, any routing of data dependency $\mathbf{d}_{3,1}^1 = (\frac{1}{2})$ would cause the usage of a horizontal channel once and the usage of a vertical channel twice. In the routing of the five data dependencies shown in Fig. 3(a), the total channel usage is only five. That is because the routing of certain data dependencies benefits from the routing of other data dependencies. For example, the routing of data dependency $\mathbf{d}_{3,1}^1 = (\frac{1}{2})$ emerging from $\text{WPPE} \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ passes the variable instance via $\text{WPPE} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ (see Fig. 3(a)). Thus, a routing of data dependency $\mathbf{d}_{2,1}^1 = (\frac{1}{1})$ becomes redundant, if the variable instance is available at $\text{WPPE} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ when it is needed there. Similarly, we minimize the usage of registers by avoiding a storage of duplicates of variable instances simultaneously at the same WPPE.

The scheduling offset τ^{offs} has a significant influence on possible solutions to the communication problem. The time between the source WPPE and the sink WPPE of a data dependency \mathbf{d} is shifted by $\tau^{\text{offs}} \cdot \mathbf{d}$ time steps. For example, if $\tau^{\text{offs}} = (2\ 4)$, there are ten additional time steps for the routing of data dependency $\mathbf{d}_{3,1}^1 = (\frac{1}{2})$ because $(2\ 4) \cdot (\frac{1}{2}) = 10$. This results in a lot of flexibility in the

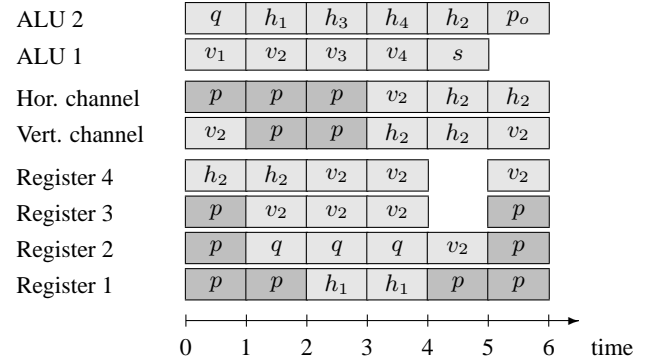


Figure 4. Gantt chart of the usage of the data path and the communication resources within an iteration interval. The variable of the EDA that is computed, transferred or stored is named.

timing behavior of the routing of the data dependency at the cost of an increased register usage.

4.3 Combining of the Data Path Problem and the Communication Problem

In the following, we deal with the combining of the data path problem and the communication problem. We begin with a formulation of the communication problem for the iteration interval λ_{\min} and the scheduling offset τ^{offs} according to a solution to the data path problem (see Section 4.1). Next, we add the constraints from the data path problem to ensure that the data path is exploited as well. A typical objective for the combined optimization problem is the minimization of the channels and registers that need to be implemented in the WPPA. Other objectives are the minimization of the registers, where the number of channels is fixed or vice versa.

As an example, we have searched a solution to the combined optimization problem of the EDA, where we presume a WPPA as described at the end of Section 3 (two ALUs in each WPPE, interconnection: grid topology). We determined the minimal iteration interval to $\lambda_{\min} = 6$. Suitable scheduling offsets were $\tau^{\text{offs}} = (1\ 4)$ or $\tau^{\text{offs}} = (4\ 1)$. In both cases, it was not possible to find a routing of the data dependencies with only one horizontal and one vertical channel. Hence, we have tried larger scheduling offsets because of the increased timing flexibility. We found a solution for $\tau^{\text{offs}} = (2\ 4)$ with a minimum of four registers at each WPPE. This solution is depicted in the Gantt chart in Fig. 4. There, we show the usage of the two ALUs, the two channels and the four registers of a WPPE for one iteration interval λ . This usage repeats periodically with λ . Note that we have emphasized the usage of channels and registers in the Gantt chart for variable p (see similarity to Fig. 3(b)).

Even by varying the scheduling offset, it may not always be possible to find a routing of the data dependen-

cies when the communication resources are limited. In that case, it may be necessary to increase the iteration interval, i. e., try $\lambda = \lambda_{\min} + 1$, $\lambda = \lambda_{\min} + 2$, and so on.

Some remarks about the optimization. We have formulated ILPs and solved them with CPLEX 8.1 on an UltraSparc III, 900 MHz. First, we have solved the data path problem which took less than a second. Next, we have solved the combined optimization problem which is more complex. To quickly obtain results, we have omitted the minimization of registers at first. The optimization was finished after 17 seconds. In that solution, five registers would have been required within each WPPE. The Gantt chart (Fig. 4) shows a solution to the overall combined optimization problem where registers are minimized as well. In our example, this complete approach saves 20 percent in register cost (four instead of five registers). But it took 97 minutes to find this optimal solution. The verification that there is no better solution was finished after an additional 400 minutes. We believe that for typical applications, the fast approach without a register optimization is suitable.

5 Conclusion and Outlook

In this paper we have presented a systematic approach to map a certain class of regular nested loop programs to dedicated processor arrays. This can be used to efficiently accelerate the execution of compute-intensive algorithms, for instance in fast digital signal, image and video processing under real-time requirements. The methods presented are integrated in our framework of a compiler for coarse-grained reconfigurable architectures. We have shown a novel approach, where we consider several architectural constraints in one model. Thereby, we find an efficient schedule and allocation for the computational tasks of the algorithm as well as a regular routing for the communication within the processor array. In the future, we would like to extend our methods to more advanced (combined) partitioning schemes which allow a balancing of memory bandwidth and local memory.

Acknowledgement

This project is supported in part by the German Science Foundation (DFG) in project under contract TE 163/13-1.

References

[1] Paul Feautrier. Automatic Parallelization in the Polytope Model. Technical Report 8, Laboratoire PRiSM, Université des Versailles St-Quentin en Yvelines, 45, avenue des États-Unis, F-78035 Versailles Cedex, June 1996.

[2] Dmitriy Kissler, Frank Hannig, Alexey Kupriyanov, and Jürgen Teich. A Highly Parameterizable Parallel Processor Array Architecture. In *Proceedings of the IEEE International Conference on Field Programmable Technology (FPT)*, Bangkok, Thailand, December 2006.

[3] Dmitriy Kissler, Frank Hannig, Kupriyanov Alexey, and Jürgen Teich. A Dynamically Reconfigurable Weakly Programmable Processor Array Architecture Template. In *Proceedings of the 2nd International Workshop on Reconfigurable Communication Centric System-on-Chips*, Montpellier, France, July 2006.

[4] Reiner Hartenstein. A Decade of Reconfigurable Computing: A Visionary Retrospective. In *Proceedings of Design, Automation and Test in Europe*, pages 642–649, Munich, Germany, March 2001. IEEE Computer Society.

[5] Elixent Ltd. www.elixent.com.

[6] Masato Motomura. A Dynamically Reconfigurable Processor Architecture. In *Microprocessor Forum*, CA, 2002.

[7] V. Baumgarte, G. Ehlers, Frank May, A. Nüchel, Martin Vorbach, and Markus Weinhardt. PACT XPP – A Self-Reconfigurable Data Processing Architecture. *The Journal of Supercomputing*, 26(2):167–184, 2003.

[8] Silicon Hive. www.siliconhive.com.

[9] QuickSilver Technology. www.qstech.com.

[10] G. Venkataramani, W. Najjar, F. Kurdahi, N. Bagherzadeh, Wim Böhm, and Jeff Hammes. Automatic Compilation to a Coarse-grained Reconfigurable System-on-Chip. *ACM Trans. on Embedded Computing Systems*, 2(4):560–589, November 2003.

[11] Hartej Singh, Ming-Hau Lee, Guangming Lu, Nader Bagherzadeh, Fadi J. Kurdahi, and Eliseu M. Chaves Filho. Morphosys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications. *IEEE Transactions on Computers*, 49(5):465–481, 2000.

[12] Jong-Eun Lee, Kiyoun Choi, and Nikil Dutt. An Algorithm for Mapping Loops onto Coarse-grained Reconfigurable Architectures. In *Languages, Compilers, and Tools for Embedded Systems (LCTES'03)*, pages 183–188, San Diego, CA, June 2003. ACM Press.

[13] Dirk Fimmel and Renate Merker. Localization of Data Transfer in Processor Arrays. In *Euro-Par'99 - Parallel Processing: 5th International Euro-Par Conference*, volume 1685 of *LNCS*, pages 401–408, 1999.

[14] S. Derrien, S. Rajopadhye, and S. Sur-Kolay. Combining Instruction and Loop Level Parallelism for Array Synthesis on FPGAs. In *Int. Symposium on System Synthesis*, 2001.

[15] Christine Eisenbeis and Antoine Sawaya. Optimal Loop Parallelization under Register Constraints. Technical Report 2781, INRIA, January 1996.

[16] Jan Müller, Dirk Fimmel, and Renate Merker. Optimal Loop Scheduling with Register Constraints Using Flow Graphs. In *Proc. of the 7th Int. Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN)*, Hong Kong, 2004.

[17] Sebastian Siegel and Renate Merker. Minimum Cost for Channels and Registers in Processor Arrays by Avoiding Redundancy. In *IEEE Int. Conf. on Application-Specific Systems, Architectures, and Processors (ASAP 2006)*, pages 28–32, Steamboat Springs, CO, USA, September 2006.

[18] Lothar Thiele. Resource Constrained Scheduling of Uniform Algorithms. *Journal of VLSI Signal Processing*, 10:295–310, 1995.

[19] Jürgen Teich, Lothar Thiele, and Li Zhang. Scheduling of Partitioned Regular Algorithms on Processor Arrays with Constrained Resources. *Journal of VLSI Signal Processing*, 17(1):5–20, September 1997.