Area and Reconfiguration Time Minimization of the Communication Network in Regular 2D Reconfigurable Architectures

Christophe Wolinski
Univ. of Rennes I/IRISA
France

Krzysztof Kuchcinski
Lund University
Sweden

Jürgen Teich, Frank Hannig
Univ. of Erlangen-Nuremberg
Germany*

Abstract

In this paper, we introduce a constraint programming*-based approach for the optimization of area and of reconfiguration time for communication networks for a class of regular 2D reconfigurable processor array architectures. For a given set of different algorithms the execution of which is supposed to be switched upon request at run-time, we provide static solutions for the optimal routing of data between processors. Here, we support also multi-casting data transfers for the first time. The routing found by our method minimizes the area or the reconfiguration time of the communication network, when switching between the execution of these algorithms. In fact, when switching, the communication network reconfiguration can be executed in just a few clock cycles. Moreover the communication network area can be minimized significantly (62% in average).*

1 Introduction

In this paper, we focus on the problems of the static optimization of area and reconfiguration time for communication network of regular 2D reconfigurable processor array architectures. To solve these problems (a) jointly and (b) not for a single, but for a whole *set of algorithms*, a unique constraint programming approach has been applied.

Previously we have introduced an abstract model for minimization of the number of multiplexers [12]. This model is limited and covers only unicasting data transfers. In this paper, we propose a new optimized formulation that makes it possible to support multicasting data transfers. Moreover, we define new cost functions that make the minimization of other communication network parameters possible, such as area as well as parallel and sequential reconfiguration time.

The correctness of our approach is illustrated by applying our methodology to a concrete architecture, namely *weakly programmable processor array* (WPPA) [7]. This

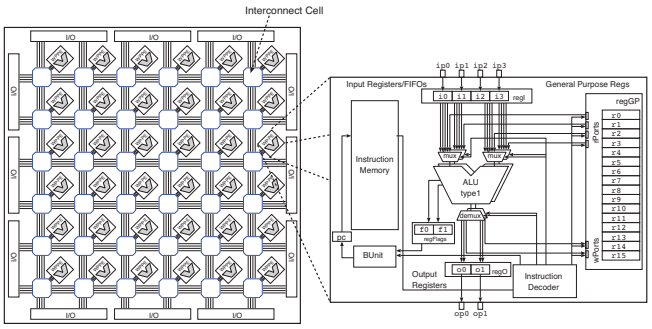

Figure 1. Example of a WPPA. All parameters such as number and type of processor elements and their interconnect structure can be defined at synthesis-time according to domain-specific needs.

architecture belongs to a class of computer architectures that consist of an array of processing elements with reconfigurable interconnections and limited programming possibilities, see Fig. 1. However, we would like to emphasize that our approach is not limited to WPPAs and can be applied to any other regular 2D reconfigurable architecture. To our knowledge there is no other published similar solution.

Related work. Routing of communication requests in reconfigurable networks is a topic of huge relevance in the area of billion transistor SoCs. Here, two different directions can be distinguished: The first aims at establishing dynamically connections between hardware components by switching wires. This area is called *circuit-switched routing* and our approach presented here also belongs to this class. Especially for fine-grained reconfigurable hardware systems (e.g., FPGA) concepts such as reconfigurable multiple buses [1] have been recently studied. In [6], a template-based approach is presented where it is possible to set a fixed routing path between modules by attaching these templates through dynamic partial reconfiguration.

In case of dynamically changing communication requests, the other main stream is based on message passing *networks-on-a-chip* (NoC), see for instance [2, 5]. Here, components send messages (packets) which are routed

*This work has been supported in part by the German Science Foundation (DFG) in project under contract TE 163/13-1 and TE 163/13-2.

978-1-4244-1961-6/08/$25.00 ©2008 IEEE.

through router nodes to their destinations. In the context of reconfigurable FPGA designs, these capabilities have also been studied. For example in [3], a 2D NoC concept called DyNoC (Dynamic Network-on-a-Chip) that can be dynamically reconfigured at run-time is presented. The concept applies modified XY-routing in a mesh-like NoC that can handle also obstacles given by placed modules on the FPGA. Unfortunately, the cost of NoC solutions can be very high. Also, the delay of communications can be substantial, i.e., in case of congestions or, in case of multi-hop routing. Finally, also memory elements must be provided in router nodes to store data packets temporarily. For cycle-based reconfigurable coarse-grained architectures such as WPPAs that we are considering in this paper, a routing network would be much too slow as we demand the ability to switch communications on a cycle-base here. Also, the connections themselves should be delay-free. Therefore, circuit routing is the only viable solution here.

A communication-conscious mapping approach for WPPAs, based on integer linear programming, is presented in [10]. But, this approach considers only the static mapping of a single application. In [9], the authors present mapping heuristics to merge datapaths and to share interconnection structures in reconfigurable architectures. The minimization of the interconnection network's size leads also to reduced reconfiguration times. Whereas the optimization goals are similar to ours, we present an exact and resource constraint method where a limited number of channels between the processor elements can be considered.

Routing has been defined before using constraint satisfiability encodings. In [11], authors encode FPGA detailed routing problems using SAT definition. They can find a routing or prove that a particular global routing does not have a detailed routing for a given number of tracks per channel. Unlike our approach, their formulation cannot be used for optimization of particular features of routing.

Organization. Section 2 introduces WPPAs, shows how an application can be executed on it and presents a formula for parallel reconfiguration time overhead calculation. Section 3, is devoted to a small example introducing the problem of routing data dependencies for a given algorithm. The optimization problem is discussed in Section 4. Finally, a case study is presented (Section 5) with promising optimization results.

2 Weakly programmable processor arrays

A WPPA architecture consists of an array of *weakly programmable processor elements (WPPEs)* each having a VLIW (*Very Long Instruction Word*) structure, see Fig. 1 (right). The parameters of each WPPE can be customized at synthesis-time with respect to the number and types of functional units such as adders, subtractors, multipliers, shifters, and modules for logical operations. Furthermore, special

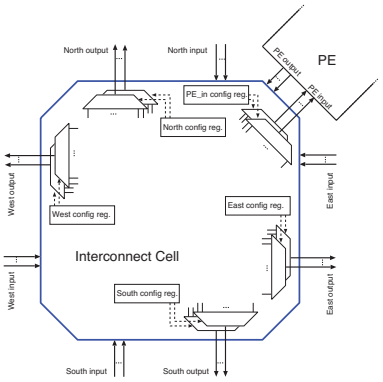

Figure 2. Multiplexer architecture of an interconnect cell (refinement of Fig. 1 (left)).

storage elements at the inputs of each WPPE have been proposed to store incoming data [7]. The instruction set of a single WPPE is minimized also according to domain-specific computational needs.

In order to allow to model a vast set of different interconnect topologies, dynamically reconfigurable interconnect structures have been investigated by the definition of a switchable *interconnect cell* structure a WPPE is connected to. In our example in Fig. 1 (left), the interconnect cell form a regular 2D-mesh topology.

A set of VLIW programs and an *interconnect configuration* together form a so-called *setup*. The *global setup memory* contains several processor array setups, one for each algorithm which can be processed by the array. Since an on-chip setup memory consumes logic resources, it has to be as small as possible.

A given WPPA is characterized not only by the number and types of processing elements and their internal structure but also by the interconnection capabilities. These are given in terms of (a) number of channels in each direction (east, west, north, south), see Fig. 2, and (b) number of ports of a processing element.

Algorithm reconfiguration on WPPAs is initiated and requiring two steps: *Program reconfiguration* which is beyond the scope of this paper and *interconnect reconfiguration*. Interconnect reconfiguration is possible through overwriting configuration registers located in each interconnect cell as shown in Fig. 2. Each connection to other PEs via so-called *channels* may be changed dynamically by loading a new value to each of these configuration registers. Note, if there are several multiplexers in one direction, the select signals are concatenated in one reconfiguration register. This enables to reconfigure all channels in the same direction within one cycle.

The corresponding time overhead for configuration of the interconnect in one *multicast domain* is given as fol-

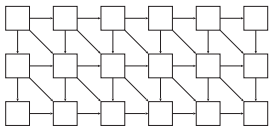

Figure 3. Processor array implementation of a FIR filter with $L = 6$ taps.

lows:

$$T_{INT_{cfg}} = o + t_{mux_N} + t_{mux_E} + t_{mux_S} + t_{mux_W} + t_{mux_{PE_{in}}} \quad (1)$$

where

$$t_{mux_x} = \begin{cases} 1 & \text{if \#multiplexers in direction } x \geq 1 \\ 0 & \text{else} \end{cases}$$

The variable o is in our case a constant which reflects the overhead for the setup of a *configuration automaton*. In a WPPA framework implemented in FPGAs, an experimental running example requires a setup time of o=4 cycles.

3 Algorithm class and mapping

Starting point of the design flow for mapping algorithms onto WPPAs is the class of so called *dynamic piecewise regular algorithms* (DPRAs) [4]. This class of algorithms describes loop nests containing uniform data dependencies by a set of recurrence equations.

The following example of an FIR filter $y_{out}[n] = \sum_{m=0}^{L-1} a[m] \times u[n-m]$ with L taps is used to illustrate the mapping of regular algorithms onto a programmable array architecture. After embedded all coefficients $a[m]$ and filter inputs $u[n-m]$ into a common two-dimensional space, we obtain the description:

```
par (n>=0 and n<=T-1 and m>=0 and m<=L-1)
  a[n,m] = a[n-1,m]           if (n > 0);
  u[n,m] = u[n-1,m-1]         if (n > 0 and m > 0);
  x[n,m] = a[n,m] * u[n,m];
  y[n,m] = y[n,m-1] + x[n,m] if (m > 0);
  y_out[n] = y[n,m]           if (m == L-1);
```

The application of space-time transformation [4] leads to a 2D processor array structure[1] as shown in Fig. 3.

Now, for configuration of the communication interconnect, we have to lock at the algorithms' *data dependencies*, e.g., the dependency between variable a and a as vector $(1,0)^T$, between u and u as vector $(1,1)^T$, and finally between y and y as vector $(0,1)^T$.

For a given algorithm A_i, we can group the set of data dependencies like in the FIR-algorithm above into a set $\{\vec{d}_{i,1}, \cdots, \vec{d}_{i,D(i)}\}$ of two-dimensional vectors in the following. Note that for each algorithm A_i to be executed at runtime, there may be a different set of data dependencies.

On the physical processor array, these data dependencies result in *connection dependencies* as can be seen in Fig. 3.

[1]The depicted array implementation of the FIR filter is able to process three input samples in parallel.

Now, in case the target WPPA does not support these vectors directly as interconnect channels, they must be routed over disjoint *routing paths* of channels. The corresponding optimization problem to find such a routing configuration for several algorithms simultaneously, is the subject of Section 4.

4 Optimization of communication overhead

Now, we present our framework for statically minimizing the interconnect configurations for a set of time-multiplexed algorithms such that the reconfiguration overhead in terms of required multiplexers is minimized as a secondary goal. We will see that this problem involves solving routing problems. We model the problem using constraint programming over finite domains formulation [8].

Modeling of all connection dependencies $\vec{d}_{i,j}$ for each algorithm A_i $(1 \leq i \leq K)$ is done using a set of 2D arrays of cells of size $N \times M$. N and M denote the maximum horizontal and vertical part of the Manhattan distance of all connection dependencies. Therefore, the optimization problem is independent of the processor array's size. Each cell $Cell_{n,m}$ is identified by its (n,m) coordinates in 2D grid $(0 \leq n \leq N-1, 0 \leq m \leq M-1)$. Each array represents implementation of a single connection dependency for a given algorithm. For example, implementation of two connection dependencies $\{(1,0)^T, (1,1)^T\}$ is depicted in Fig. 4 for $N = M = 2$.

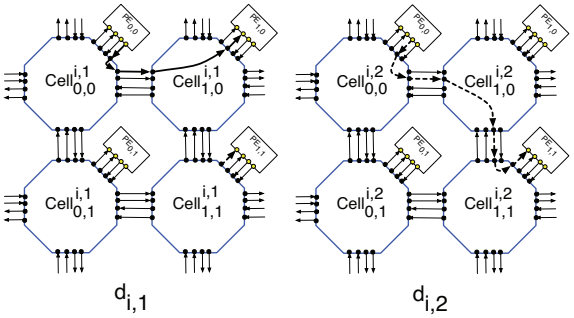

Figure 4. Example of implementation of connection dependencies $\vec{d}_{i,j} = \{(1,0)^T, (1,1)^T\}$.

It can be noted that connections between different PEs can be routed using different resources. In principle we need to make a number of decisions that can be grouped into two classes.

- decisions on the selection of the path from source PE to destination PE that passes different cells, and
- decisions on the selection of different connections in the channels between cells.

Both decisions influence the number and size of multiplexers that need to be included to reconfigure static connections between cells when reconfiguring from an algorithm into another one. They need to be considered simultaneously. For this purpose we have defined a constraint programming model.

Before defining the model we need to point out that all connections need to be implemented in a single cell since all

the cells in the architecture execute the same program and they use the same channel connections to transfer the data (so called *modulo routing* [12]). This means that we can reduce the optimization problem to a single cell that contains all routed connection for all connection dependencies for given algorithms. Normally, the implemented connections for one algorithm cannot use the same connection channels. A cell that implements all connections from Fig. 4 is depicted in Fig. 5. Bold lines represent the implementation of connection dependency $(1,0)^T$ while doted lines represent connection dependency $(1,1)^T$,

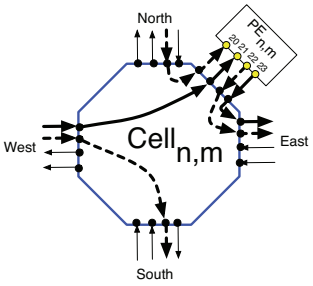

Figure 5. The cell connections for implementation of $\vec{d_{i,j}} = \{(1,0)^T, (1,1)^T\}$.

Our communication minimization problem is split into two steps to reduce the complexity of our method. In the first step, all possible paths between interconnect cells and PEs in the system corresponding to all $\vec{d_{i,j}}$ $1 \leq j \leq D(i)$ and $1 \leq i \leq K$ are found using a CP formulation. In the second step, another CP formulation is used for multiplexer area and reconfiguration time minimization. We will see later that the area and reconfiguration time are expressed in terms of size and number of multiplexers. It encodes all paths using model variables and assigns connection dependencies to connections of channels implementing identified paths.

All paths corresponding to a given dependency $\vec{d_{i,j}}$ are found by applying a `SimplePath` constraint to the *Simplified System Graph* (*SSG*). The *SSG* graph is defined as a directed graph. Vertices are cells and processing elements while edges represent inter-cell channels between processing elements and cells.

The constraint that finds all simple paths in a graph takes as a parameter a graph, a source and a destination vertex (i.e. PEs in our case). This constraint can be combined with other constraints to generate paths of a limited length that is useful in practice.

The identified paths need to be encoded in our model as variables. This is achieved by a special constraint (`ExtensionalSupport` in our case) that defines a relation between model variables using a table of values. For this purpose we use $0/1$ variables $Ch^{i,j}_{(n,m)(n',m')}$ that define for algorithm i and dependency j whether a directional channel from cell (n,m) to cell (n',m') is used (value 1) or not (value 0). Table 1 presents variables that are set to one for different paths for connection dependencies $(1,0)^T$ and $(1,1)^T$ from Fig. 4. All other variables in each row are equal zero.

In our model we also maintain explicitly input/output

Table 1. Encoding of simple paths for implementation of connection dependencies.

$\vec{d_{i,1}} = (1,0)^T$	path 1	$Ch^{i,j}_{PE(0,0)(0,0)}, Ch^{i,j}_{(0,0)(1,0)}, Ch^{i,j}_{(1,0)PE(1,0)}$
	path 2	$Ch^{i,j}_{PE(0,0)(0,0)}, Ch^{i,j}_{(0,0)(0,1)}, Ch^{i,j}_{(0,1)(1,1)}, Ch^{i,j}_{(1,1)(1,0)}, Ch^{i,j}_{(1,0)PE(1,0)}$
$\vec{d_{i,2}} = (1,1)^T$	path 1	$Ch^{i,j}_{PE(0,0)(0,0)}, Ch^{i,j}_{(0,0)(1,0)}, Ch^{i,j}_{(1,0)(1,1)}, Ch^{i,j}_{(1,1)PE(1,1)}$
	path 2	$Ch^{i,j}_{PE(0,0)(0,0)}, Ch^{i,j}_{(0,0)(0,1)}, Ch^{i,j}_{(0,1)(1,1)}, Ch^{i,j}_{(1,1)PE(1,1)}$

pairs of channels that define internal *cell connectivity*. For example, pair $(Ch^{i,2}_{(0,0)(1,0)}, Ch^{i,2}_{(1,0)(1,1)})$ defines internal cell connection from "West" to "South" indicated as doted line in Fig. 5.

In this way we provide opportunity to select one of the paths for implementing a given connection dependency. For a selected path a number of specific channel connections need to be determined. They implement communication between cells and a cell and a PE. In our example depicted in Fig. 5, each channel has two connections and communication dependencies can use any of the connections but normally they cannot share a channel connection. It is only possible for multicast communications that will be discussed later.

The channel selection is implemented using a *channel occupation table*. It is in turn implemented in our model using `Diff2` constraint that assures that any pair of rectangles specified on a list of rectangles do not overlap. The idea is depicted in Fig. 6. In this formulation each rectangle represents a channel connection and the constraint assures that two connections will not use the same connection in the channel. A rectangle is specified using its origin (x,y) and lengths in both directions l_x and l_y, i.e. using list $[x,y,l_x,l_y]$. All channels connecting two cells or a cell and PE in a given direction are collected in a list of rectangles. For example, in direction "South" from Fig. 5, variables $Ch^{i,j}_{(0,0)(0,1)}$ and $Ch^{i,j}_{(1,0)(1,1)}$ for all $1 \leq j \leq D(i)$ are used for selection of connections. For our example it is defined using constraints (2). Note that we use a special feature of `Diff2` constraint that considers rectangles with length zero as non-existing. This makes it possible to consider only selected connections and assure that they do not use the same connection channel at the same time.

$$\begin{aligned}\texttt{Diff2}([&1, Out^{i,1}_{(0,0)(0,1)}, Ch^{i,1}_{(0,0)(0,1)}, 1], \\ &[1, Out^{i,1}_{(1,0)(1,1)}, Ch^{i,1}_{(1,0)(1,1)}, 1], \\ &[1, Out^{i,2}_{(0,0)(0,1)}, Ch^{i,2}_{(0,0)(0,1)}, 1], \\ &[1, Out^{i,2}_{(1,0)(1,1)}, Ch^{i,2}_{(1,0)(1,1)}, 1]),\end{aligned} \tag{2}$$

For multicast communication this condition is relaxed. In addition we also enforce that the same connection is used for multicast communications in the same direction (constraint 3).

$$\forall_{1 \leq j,j' \leq D(i),\ j \neq j'}\ Out^{i,j}_{(n,m)(n',m')} = Out^{i,j'}_{(n,m)(n',m')} \tag{3}$$

We introduce vectors Tab^i_{Dir} to collect all inputs that are connected to a given cell output connection for a given al-

gorithm i and connection direction Dir. These vectors are defined for direction south, west, north, east and PE. The information for these vectors is gathered using constraint (4) that is formulated for all internal cell connections. For the example of connection dependencies presented in Table 1 only one internal cell connection exists from "West" to "South" and therefore only one constraint has to be formulated for this output direction (4). Vectors Tab^{i}_{Dir} are later used for formulation of cost functions. It can be noted that the number of model variables in the model is radically reduced comparing to our previous model [12].

$$Tab^{i}_{South}[Out^{i,2}_{(1,0)(1,1)}] = In^{i,2}_{((0,0)1,0)} \Leftrightarrow Ch^{i,2}_{(0,0),(1,0)} = 1 \wedge Ch^{i,2}_{(1,0),(1,1)} = 1 \quad (4)$$

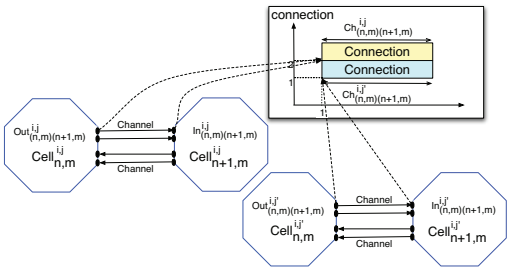

Figure 6. The channel connection selection for two connection dependencies $\vec{d}_{i,j}$ and $\vec{d}_{i,j'}$.

The model defines all communications for a single algorithm. They are then combined into a single model that contains all algorithms with their connections dependencies. In this model we define a number of cost function to reduce the communication overhead related to reconfiguration. To define these cost function the tables Tab^{i} defined in (4) are used. The main cost function defines a condition that specifies when a multiplexer is needed. A multiplexer needs to be included in a WPPE if there exist two paths implementing connection dependencies for two disjoint algorithms A_i and $A_{i'}$, and there exist an output connection that has inputs from two different connections. This condition is defined using our tables depicted in Fig. 7. In this table each vector Tab^{i}_{Dir} (column in the array) defines input connection numbers connected to given output connections in the algorithm i. Each row, on the other hand defines, for each output connection, input connection numbers for all algorithms. Therefore a number of different connection numbers (in the row) defines number of inputs to this particular output and a multiplexer with this number of inputs. In this case we do not consider zeros since numbering starts from one. Two variables $MultSize_{Dir,t}$ and $MultExist_{Dir,t}$ are associated to each row. The first one defines the multiplexer's size and the second defines whether the multiplexer is needed or not. In our experiments, we consider two optimization objectives; the multiplexers' area and the reconfiguration time overhead. The related cost functions are specified using variables $MultSize_{Dir,t}$ and $MultExist_{Dir,t}$ whose values are calculated directly from table Tab_{Dir}.

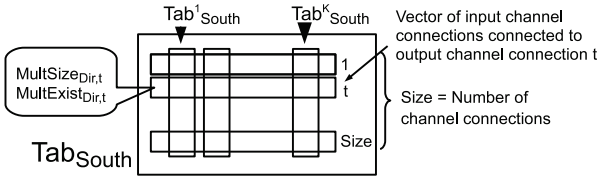

Figure 7. The table structure used for computation of cost functions.

The reconfiguration time overhead is defined according to Eq. (1) and expressed below with constraints (5).

$$\forall_{DIR} \sum_{t} MultExist_{Dir,t} > 0 \Leftrightarrow t\_mux_{Dir} \quad (5)$$

$$RecTimeOverheadCostFunction = \sum_{Dir} t\_mux_{Dir}$$

Above constraints use *reified* constraint, i.e., constraint $Cond \Leftrightarrow B$ that reflects satisfiability of condition $Cond$ into a 0/1 variable B.

Area overhead is defined below as weighted sum of different types of multiplexers, i.e., two input, three input, etc.

$$List = \{MultSize_{South,1}, \ldots, MultSize_{PE,K}\} \quad (6)$$

$$\forall_{i \in \{2,\ldots,K\}} \texttt{Count}(i, List, MuxSize_i)$$

$$AreaOverheadCostFunction = \sum_{i=2}^{K} (i-1) \cdot MuxSize_i$$

Constraint `Count(K, List, Var)`, used in the above formulation, assures that number of elements of `List` with value `K` equals `Var`.

5 Experimental results

To validate our approach for area and reconfiguration time overhead minimization, we have carried out experiments using six algorithms A_i with different connection dependencies, as presented in Table 2. Each experiment used different combinations of these algorithms to evaluate reconfiguration overhead. The algorithms represent both existing algorithms as well as synthetic benchmarks. Algorithm 1 represents the connection dependencies of a matrix-matrix multiplication algorithm and Algorithm 2 is a FIR filter algorithm. They are two frequently used digital signal processing algorithms. The connection dependencies of A_3 stem from a Sobel image filtering example. Algorithms A_4 to A_6 represent synthetic benchmarks. All of the following experiments have been run on 2 GHz Intel Core Duo under Mac OSX operating system.

In Table 3, we present our results obtained for minimization of area overhead and sequential reconfiguration time. We also compare the obtained results with a naïve approach. All experiments are carried out for a minimal assumed number of channel connections and input/output ports needed for routing all dependencies. As can be seen, the area and sequential reconfiguration time improvement between the naïve approach and our method are rather large. The average value for area improvement is 62% and average sequential reconfiguration time improvement is 41%.

Table 2. Six algorithms and their connection dependencies.

Algorithm	$\vec{d}_{i,j}$
A_1	$\{(-1,0)^T,(0,1)^T,(0,1)^T\}$
A_2	$\{(-1,0)^T,(-1,-1)^T,(0,1)^T\}$
A_3 multicasting	$\{(0,-1)^T,(-1,-2)^T,(-1,-1)^T,(-2,-1)^T,(1,0)^T\}$
A_4	$\{(2,0)^T,(2,1)^T\}$
A_5	$\{(0,1)^T,(1,1)^T\}$
A_6	$\{(0,1)^T,(1,0)^T\}$

Table 3. Area overhead and sequential reconfiguration time of (a) naïve solution and (b) optimized solutions for different combinations of algorithms.

Algorithms	#MUX with N inputs Naïve				Optimized				Area		Improvement Area	Serial Reconf.
	2	3	4	5	2	3	4	5	Naïve	Opt.		
A_1, A_2	6				1				6	1	83.33%	50.00%
A_1, A_5	4				3				4	3	25.00%	12.50%
A_1, A_6	5				3				5	3	40.00%	22.22%
A_2, A_5	5				1				5	1	80.00%	44.44%
A_2, A_6	5				3				5	3	40.00%	22.22%
A_5, A_6	5								5	0	100.00%	55.56%
A_1, A_2, A_5	3	4			3				11	3	72.53%	53.33%
A_1, A_2, A_6	4	3			4				10	4	60.00%	42.86%
A_1, A_5, A_6	5	3			4				11	4	63.64%	46.67%
A_2, A_5, A_6	4	3			2	1			10	4	60.00%	42.86%
A_1, A_2, A_5, A_6	3	3	2		4	1			15	6	60.00%	47.37%
$A_1, A_2, A_3, A_4, A_5, A_6$	7	5	3	1		2	2	2	30	11	63.33%	55.88%

Table 4 presents the results of minimizing the parallel reconfiguration time, again comparing naïve with our optimized solutions. We specify the number of dimensions that use multiplexers. The reconfiguration time is shorter for parallel reconfiguration when optimized with our method and in average we obtain 32% improvement.

6 Conclusions and future work

In this paper, we have presented a constraint programming formulation for minimization of area as well as sequential and parallel reconfiguration time overhead for regular reconfigurable architectures. Our system makes it also possible to make a design space exploration that involves trading multiplexers against channel connections, for example. The experimental results indicate large savings of area, specially for applications that have larger number of algorithms and a large number of PEs.

The following extensions are possible for future work: First, a different cost function would make it possible to minimize, for example, a number of channel connections with a given limit on a number of multiplexers or a maximal length of routed paths for a given set of algorithms.

The search space for the considered problem is large, and for large problems, we cannot find or prove optimality of our solutions. This is partially caused by existence of many symmetrical solutions with the same cost. These could be eliminated by introduction of additional symmetry elimination constraints. We leave it for our future work.

Table 4. Parallel reconfiguration time of (a) naïve solution and (b) optimized solutions for different combinations of algorithms.

Algorithms	Naïve MUX in dim.	Naïve Time (cycles)	Optimized Mux in dim.	Optimized Time (cycles)	Improvement Parallel reconf. (cycles)
A_1, A_2	3	7	1	5	28.60%
A_1, A_5	3	7	1	5	28.60%
A_1, A_6	3	7	1	5	28.60%
A_2, A_5	3	7	1	5	28.60%
A_2, A_6	3	7	1	5	28.60%
A_5, A_6	3	7	0	4	42.90%
A_1, A_2, A_5	3	7	1	5	28.60%
A_1, A_2, A_6	3	7	1	5	28.60%
A_1, A_5, A_6	4	8	1	5	37.50%
A_2, A_5, A_6	4	8	1	5	37.50%
A_1, A_2, A_5, A_6	4	8	1	5	37.50%
$A_1, A_2, A_3, A_4, A_5, A_6$	5	9	2	6	33.33%

References

[1] A. Ahmadinia, C. Bobda, J. Ding, M. Majer, J. Teich, S. Fekete, and J. van der Veen. A Practical Approach for Circuit Routing on Dynamic Reconfigurable Devices. In *Proc. of the 16th Int. Workshop on Rapid System Prototyping (RSP)*, pages 84–90, Montreal, Canada, June 2005.

[2] L. Benini and G. Micheli. Network on Chips: A new SoC Paradigm. *IEEE Computer*, January 2001.

[3] C. Bobda and A. Ahmadinia. Dynamic Interconnection of Reconfigurable Modules on Reconfigurable Devices. *IEEE Design & Test*, 22(5):443–451, 2005.

[4] F. Hannig and J. Teich. Resource Constrained and Speculative Scheduling of an Algorithm Class with Run-Time Dependent Conditionals. In *Proc. of the 15th Int. Conference on Application-specific Systems, Architectures, and Processors (ASAP)*, pages 17–27, Galveston, TX, USA, Sept. 2004.

[5] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist. Network on Chip: An Architecture for Billion Transistor Era. In *Proc. of the Int. NorChip Conference*, Sept. 2000.

[6] M. Hübner, C. Schuck, M. Kühnle, and J. Becker. New 2-Dimensional Partial Dynamic Reconfiguration Techniques for Real-time Adaptive Microelectronic Circuits. In *Proc. of the Symposium on Emerging VLSI Technologies and Architectures (ISVLSI)*, page 97, Washington, DC, USA, 2006.

[7] D. Kissler, F. Hannig, A. Kupriyanov, and J. Teich. A Highly Parameterizable Parallel Processor Array Architecture. In *Proc. of the Int. Conference on Field Programmable Technology (FPT)*, pages 105–112, Bangkok, Thailand, Dec. 2006.

[8] K. Kuchcinski. Constraints-Driven Scheduling and Resource Assignment. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 8(3):355–383, July 2003.

[9] N. Moreano, G. Araujo, Z. Huang, and S. Malik. Datapath Merging and Interconnection Sharing for Reconfigurable Architectures. In *Proc. of the 15th Int. Symposium on System Synthesis (ISSS)*, pages 38–43, New York, NY, USA, 2002.

[10] S. Siegel, R. Merker, F. Hannig, and J. Teich. Communication-conscious Mapping of Regular Nested Loop Programs onto Massively Parallel Processor Arrays. In *Proc. of the 18th Int. Conference on Parallel and Distributed Computing and Systems (PDCS)*, pages 71–76, Dallas, TX, USA, Nov. 2006.

[11] M. N. Velev and P. Gao. Comparison of Boolean Satisfiability Encodings on FPGA Detailed Routing Problems. In *Proc. of the conference on Design, Automation and Test in Europe (DATE)*, pages 1268–1273, Munich, Germany, 2008.

[12] C. Wolinski, K. Kuchcinski, J. Teich, and F. Hannig. Optimization of Routing and Reconfiguration Overhead in Programmable Processor Array Architectures. In *Proc. of the 16th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), poster*, Palo Alto, CA, USA, Apr. 2008.