



# Modeling of Interconnection Networks in Massively Parallel Processor Architectures

Alexey Kupriyanov, Frank Hannig, Dmitrij Kissler, and  
Jürgen Teich,  
Julien Lallet\*, Olivier Sentieys\*, and Sébastien Pillement\*

Department of Computer Science 12  
Hardware-Software-Co-Design  
University of Erlangen-Nuremberg  
Am Weichselgarten 3  
D-91058 Erlangen, Germany

**Co-Design-Report 05-2006**

August 15, 2006

---

\*IRISA/R2D2, University of Rennes, France

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Modeling of Interconnection Networks</b>	<b>4</b>
2.1	Modeling of Interconnection Network using Interconnect-Wrappers	5
2.2	Modeling of Interconnection Network using DyRIBoxes . . . . .	8
2.2.1	The DyRIBox formal description based on multiplexer formal description. . . . .	8
2.3	Comparison of Interconnect-Wrapper and DyRIBox Concepts . . .	10
<b>3</b>	<b>Modeling of Interconnect Structures within MAML</b>	<b>14</b>
3.0.1	MAML description of DyRIBox and DyRIBox domain. . .	14
<b>4</b>	<b>Case-Study</b>	<b>17</b>
<b>5</b>	<b>Conclusions and Future Work</b>	<b>18</b>
	<b>References</b>	<b>19</b>

In this report, we present a new concept for modeling of interconnection networks in the field of massively parallel processor embedded architectures. The main focus of the report is on two interconnection concepts, namely, *interconnect-wrapper* and *DyRIBox* definitions of reconfigurable interconnection networks. We compare both interconnection concepts against each other and formally prove their equality. Both concepts allow to model many different reconfigurable inter-processor networks efficiently. Furthermore, we point out how to define the interconnect using an architecture description language for massively parallel processor architectures called *MAML*. Finally, we demonstrate the pertinence of our approach by modeling and evaluation of different reconfigurable interconnect topologies.

## 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. These application specific systems have to fulfil different performance, cost, and power requirements. In addition, changing standards or add-ons as unique selling point of a product demand for more flexible solutions. Thus, generic highly parameterizable architecture templates in terms of IP-cores (intellectual property) have become more and more important when building such so-called Systems-on-a-Chip (SoC).

In the domain of application specific instruction set processors (ASIP), there exist some holistic design methodologies which consider the simultaneous development of architectures, simulators, and compilers. Central link between the different design aspects are Architecture Description Languages (ADL). In the following we only refer a few of the most known ADLs. At the ACES laboratory of the University of California, Irvine, for example, the architecture description language EXPRESSION [6] has been developed. From an EXPRESSION description of an architecture, the retargetable compiler EXPRESS and the cycle-accurate simulator SIMPRESS can be automatically generated. The Trimaran system [16] has been designed to generate efficient VLIW code. It is based on a fixed basic architecture (HPL-PD) being parameterizable in the number of registers, the number of functional units, and operation latencies. Parameters of the machine are specified in the description language HMDES. The ADL LISA [14] is the basis for a retargetable compiled approach aiming at the generation of fast simulators for microprocessors even with complex pipeline structures. Finally, we refer to the language MAML which has been developed in the BUILDABONG project [5]. MAML is used for the efficient architecture/compiler co-generation of ASIPs and VLIW processor architectures.

Beside the classical usage of DSPs for dataflow dominant digital signal processing several architectures based on massively parallel processor arrays are emerging. Many academic coarse-grained reconfigurable arrays have been developed [7] and, since a while, more and more commercial ones are being developed like the D-Fabrix [3], the DRP from NEC [13], the PACT XPP [2], or Bresca from Silicon Hive (Philips) [15]. All of the above described ADLs only target at the design of ASIPs and VLIW processors. Only very few approaches are known that consider also coarse-grained processor arrays. For instance, DRAA [11] is a generic reconfigurable architecture template which can represent a wide range of coarse-grained reconfigurable arrays or the Adres [12] reconfigurable architecture template which is described by an architecture description in XML. Furthermore in [12, 1], the authors use their ADL in order to explore different interconnect topologies. In this context the contributions of our technical report can be summarized as follows:

1. We introduce two generic reconfigurable interconnect methodologies for parallel processor architectures (Section 2),
2. By graph and formal languages theory we proof the equivalence of the two models (Section 2.3),
3. We extend our ADL MAML in order to model the two interconnect concepts (Section 3),
4. We perform a case-study for different well-known interconnect topologies and evaluate the cost for the flexibility being able to switch from one topology to another by dynamic reconfiguration (Section 4).

## 2 Modeling of Interconnection Networks

Since design time and cost are critical aspects during the design of processor architectures it is important to provide efficient modeling techniques in order to evaluate architecture prototypes without actually designing them. In the scope of the methodology presented here, we are looking for a flexible reconfigurable interconnect architecture in order to find out trade-offs between different interconnect topologies for a given set of applications.

Generally, a massively parallel processor architecture is defined by an array of processing elements  $P = \{ p_{ij} \}$ ,  $i = [1, M]$ ,  $j = [1, N]$  and its interconnect as shown in Fig. 1 (a). Usually, a centralized approach is a classical and convenient way to model the interconnection of the processing elements (PE). In this case, the processors are connected to one or more central switch-nodes, which provide the reconfigurable interconnect. Whereas, a decentralized approach may be more reasonable in case of homogeneous architectures. In the following, we present a modeling concept for

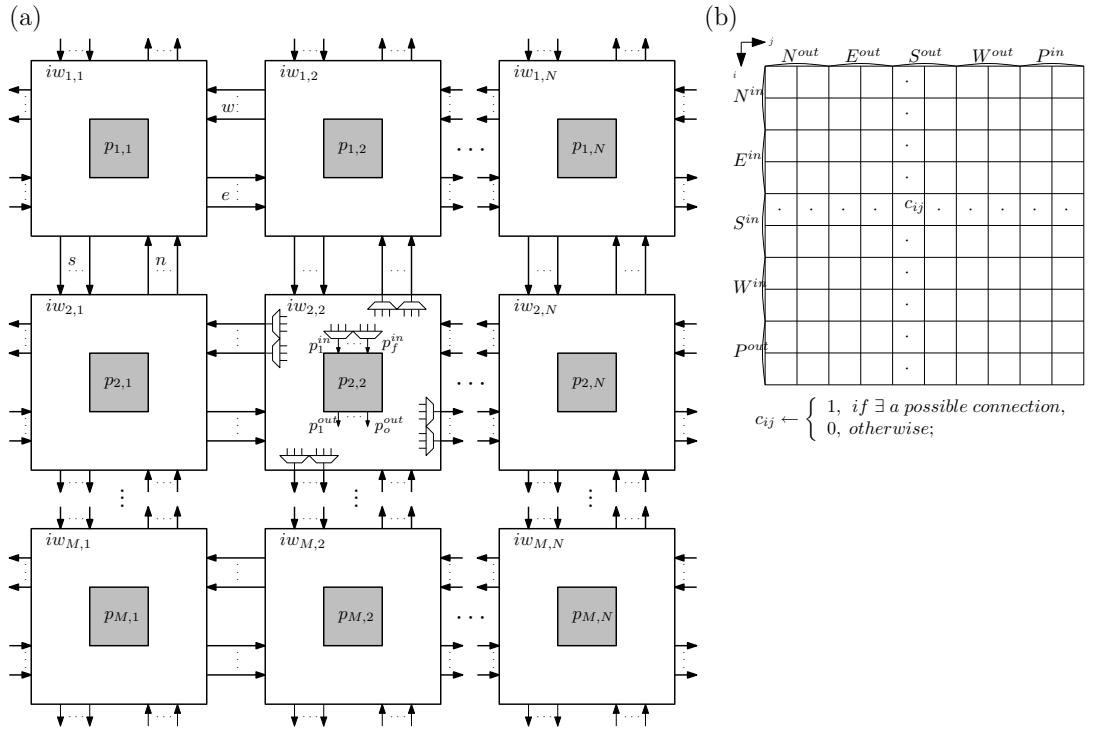


Figure 1: In (a), a structure of parallel processor architecture with interconnect modeled by the use of interconnect-wrappers (IW). In (b), a configuration of an IW - interconnect adjacency matrix.

distributed reconfigurable interconnect, compare it to the classical approach, and formally prove the ability of the decentralized approach to model any arbitrary dynamically reconfigurable interconnect topology. But first, we introduce a decentralized concept for modeling reconfigurable interconnection based on so-called *interconnect-wrappers* and a classical centralized interconnection approach called *DyRIBox*.

## 2.1 Modeling of Interconnection Network using Interconnect-Wrappers

In order to model and specify a reconfigurable interconnect topology, a PE *interconnect-wrapper* (IW) [10, 9] concept is introduced. An interconnect-wrapper describes the ingoing and outgoing signal ports of a processing element. Each interconnect wrapper has a constant number of inputs and outputs on each of its sides which are connected to the inputs and outputs of neighbor IW instances. An arbitrary massively parallel processor array can be defined as a tuple  $A = (P, IW)$  with an array of PEs  $P = \parallel p_{ij} \parallel$  and an array of interconnect-wrappers  $IW = \parallel iw_{ij} \parallel$ ,  $i = [1, M]$ ,  $j = [1, N]$ . Each processing element is represented by the sets of ingoing and outgoing ports  $p = (p^{in}, p^{out})$ , where  $p^{in} = \{p_1^{in}, \dots, p_f^{in}\}$ ,  $p^{out} = \{p_1^{out}, \dots, p_o^{out}\}$ . An interconnect wrapper is represented as a rectangle around a PE and consists of the

input and output ports on the northern, eastern, southern, and western side of it as shown in Fig. 1 (a). However, we require the input ports and the output ports on the opposite sides of an IW (i.e., northern inputs and southern outputs) to have equal bitwidths. Also, the number of them must be the same. Introduction of this condition proves the correct interconnection between neighbor IW instances. The condition can be completely satisfied by the introduction of *directed interconnect channels*. Each directed interconnect channel represents a pair of one input and one output port on the opposite sides of the interconnect wrapper with a certain common bitwidth. The direction of the channel is determined by the position of the output port. For example, if we consider a pair of *northern input* and *southern output* IW ports, then the direction of corresponding interconnect channel is southward. The interconnection between all interconnect-wrappers of the processor array is correct if and only if each IW has equivalently directed interconnect channels. An interconnect-wrapper is defined by a quintuple  $iw = (CS, CN, CE, CW, C)$  with southward channels  $CS = \{cs_1, \dots, cs_s\}$ , northward channels  $CN = \{cn_1, \dots, cn_n\}$ , eastward channels  $CE = \{ce_1, \dots, ce_e\}$ , and westward channels  $CW = \{cw_1, \dots, cw_w\}$ . The configuration of an interconnect-wrapper is defined by  $C = \parallel c_{ij} \parallel$ .  $C$  is the so-called *interconnect adjacency matrix* (IAM) (see Fig. 1 (b)). By the *configuration of an IW*, the definition of the mapping of the possible connections between the ports of an interconnect wrapper and a processor element is meant. Therefore, the particular ports of an IW should be considered instead of interconnect channels (the pair of ports). The rows of IAM represent the input ports of an IW, except the last few rows (dependent on the number of the PEs output ports) which represent the output ports of the PE. The columns represent the output ports of an IW, except the last few columns (dependent on the number of the PEs input ports) which represent the input ports of the PE. The matrix contains the values  $c_{ij}$  which are equal to "1" if there exists a possible connection between input and output ports and equal to "0" otherwise. The last rows and columns of IAM represent the port mapping between PE and IW ports. The positions of input PE ports are interchanged with the positions of the output PE ports in the IAM. This allows to avoid the configuration of such incorrect connections as a connection between IW input and PE output or a connection between PE input and IW output. If many input signals are allowed to drive a single output signal a *multiplexer* (see Definition 2.1) with appropriate number of input signals is generated.

**Definition 2.1 (Multiplexer)** A multiplexer  $f$  is a boolean function where  $f : \mathbb{B}^{n+m} \rightarrow \mathbb{B}$  with inputs  $I_f = \{x_0 \dots x_{n-1}\}$  and control signals  $S_f = \{s_0 \dots s_{m-1}\}$  (with  $m = \lceil \log_2(n) \rceil$  and  $n$  inputs):  $f(x_0 \dots x_{n-1}, s_0 \dots s_{m-1}) = x_{u(s_0 \dots s_{m-1})}$ , where  $u(s_0 \dots s_{m-1})$  is the interpretation of the positive binary number  $s_0 \dots s_{m-1}$ .

The inputs of a multiplexer are connected to the corresponding IW input signals and the output to the corresponding IW output signal. The control signals for such gener-

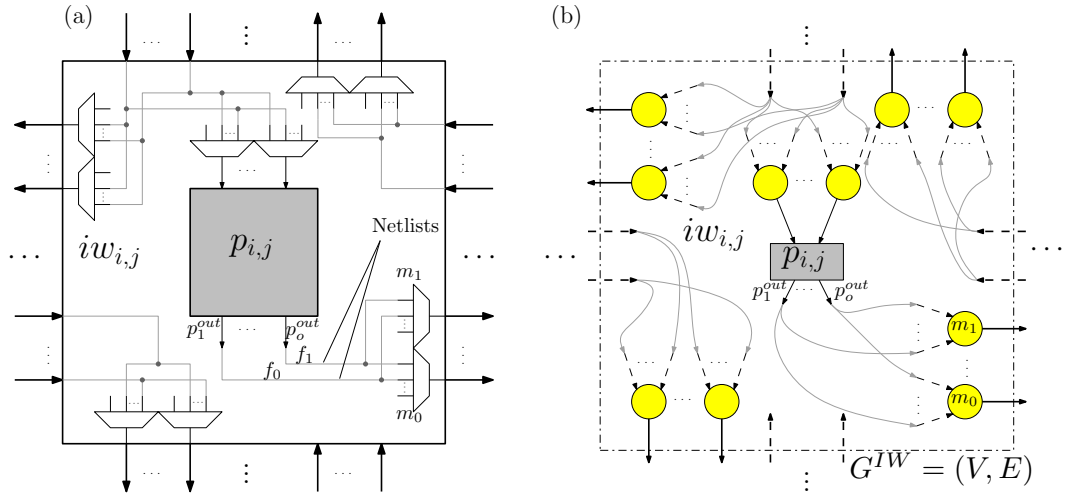


Figure 2: Representation of parallel processor architecture netlist. In (a), a netlist of IW  $iw_{i,j}$ , and in (b), its netgraph representation.

ated multiplexers are stored in configuration registers and can therefore be changed dynamically. By changing the values of the configuration registers in an interconnect-wrapper component, different interconnect topologies can be implemented and changed dynamically at run-time. Configuration registers and reconfiguration mechanisms are not in the scope of this paper.

In order to be able to compare the interconnect-wrapper concept to others, a parallel processor architecture with IW-based interconnect is represented as a directed graph  $G^{IW}(V, E)$ . Initially, the processor architecture is given by a *netlist* (see, Definition 2.2).

**Definition 2.2 (Netlist)** A netlist  $N = (V, F)$  is a set  $V$  of logic elements and a set  $F$  of nets interconnecting the elements  $v \in V$  with each other. It defines a circuit in terms of basic logic elements. A unidirectional<sup>1</sup> net  $f \in F$ , which interconnects  $n+m$  elements will be represented through  $f = (\{v_1, \dots, v_n\}, \{u_1, \dots, u_m\})$ , where  $v_1, \dots, v_n \in V$  are source nodes and  $u_1, \dots, u_m \in V$  are target nodes of net  $f$ .

In Fig. 2 (a), a netlist of interconnect-wrapper  $iw_{i,j}$  is shown. For example, net  $f_0$  is given by  $f_0 = (\{p_1^{out}\}, \{m_0, m_1\})$ . Nodes named  $p_i^{out}$  denote output ports of processing elements whereas nodes named  $m_i$  denote the multiplexer elements.

A netlist can be seen as a hypergraph, where the elements and nets are vertices and edges, respectively. This hypergraph can be transformed into a graph by the introduction of a *netgraph* concept.

**Definition 2.3 (Netgraph)** A netgraph  $G^{IW} = (V, E)$ ,  $E \subseteq V \times V$  is a directed graph containing two disjoint sets of vertices  $V = V_p \cup V_m$ , representing the processing elements or processor output ports  $V_p$  and multiplexer elements  $V_m$  of a

<sup>1</sup>A bidirectional net can be modeled by two unidirectional nets.

given netlist  $N = (V, F)$ . Netlist interconnections are represented by directed edges  $e = (v_1, v_2) \in E$ .

In Fig. 2, an IW netlist and its netgraph  $G^{IW} = (V, E)$  are shown. The subset of multiplexer elements  $V_m = \{m_0, m_1, \dots\}$  is shown as circles and the subset of processing elements  $V_p = \{p_{i,j}\}$  is represented by rectangles. In order to transform a given netlist  $N$  into a netgraph  $G$  all elements of the netlist must be analyzed first and, according to the element's type (processing element or multiplexer), they are either included in the subset  $V_p$  or in the subset  $V_m$ . All of the nets are represented as directed edges  $e \in E$  of a netgraph  $G^{IW}$ . In case when a net contains a  $n : m$  connection, it is transformed into  $n \times m$  directed edges of the netgraph, (in Fig. 2 (a), for  $f_0$  the case of a  $1 : 2$  connection is represented which is transformed into edges  $(p_1^{out}, m_0)$  and  $(p_1^{out}, m_1)$  of the graph  $G^{IW}$  in Fig. 2 (b)).

## 2.2 Modeling of Interconnection Network using DyRIBoxes

The second methodology to model reconfigurable parallel processor networks is used to permit the centralized description of dynamically reconfigurable processors. In this purpose, the interconnections have to be flexible (to be able to modify the connections "on the fly"), parameterizable in reconfiguration time and size. In this section, we give a brief description of the "Dynamically Reconfigurable Interconnections **Box**" (DyRIBox) concept which assumes all these constraints.

The basic elements of a DyRIBox are multiplexers (see Definition 2.1) connected in a way that any input can be connected to the desired output.

### 2.2.1 The DyRIBox formal description based on multiplexer formal description.

A DyRIBox is a one level switching network of multiplexers. A DyRIBox  $\mathcal{D}$  (see Fig. 3 (a)) can be described as an oriented acyclic graph  $\mathcal{D} = (V, E)$ , where  $V$  is a number of nodes and  $E \subset V \times V$  is a number of oriented edges (see Fig. 3 (b)). The set of nodes  $V$  contains a set of inputs  $\mathcal{I}(\mathcal{D})$ , a set of outputs  $\mathcal{O}(\mathcal{D})$ , and a set of multiplexers  $\mathcal{M}(\mathcal{D})$ :

- $deg^-(v) = 0, deg^+(v) > 0$ , when  $v \in \mathcal{I}(\mathcal{D})$
- $deg^-(v) = 1, deg^+(v) = 0$ , when  $v \in \mathcal{O}(\mathcal{D})$
- $deg^-(v) = |\mathcal{I}(\mathcal{D})|, deg^+(v) = 1$ , when  $v \in \mathcal{M}(\mathcal{D})$ ,

where  $deg^-$  and  $deg^+$  are input and output degrees of a node, respectively.

Each edge  $e \in E$  which comes in a multiplexer node  $v \in \mathcal{M}(\mathcal{D})$  is an input  $i_v(e) \in \{0, \dots, deg^-(v) - 1\}$ . A configuration of a DyRIBox is the following representation

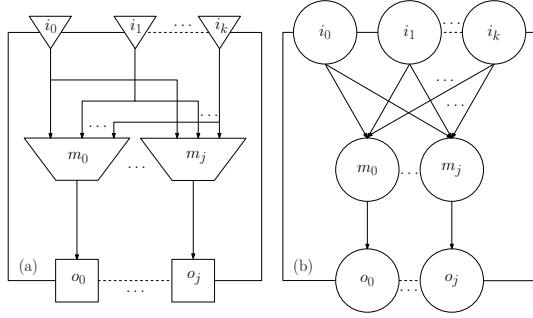


Figure 3: DyRIBox with multiplexer and node view.

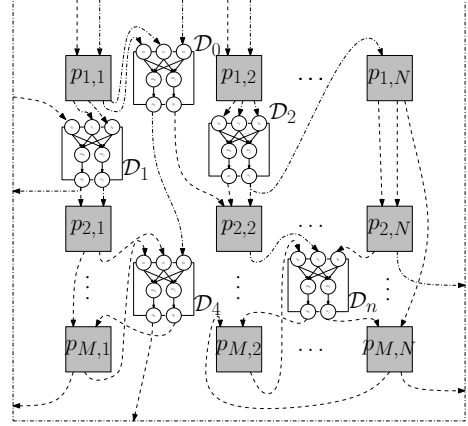


Figure 4: Example of a heterogeneous architecture using DyRIBoxes for interconnect modeling.

$\chi : M(\mathcal{D}) \rightarrow \mathbb{N}$ , with for all multiplexer nodes  $v \in M(\mathcal{D}) : 0 \leq \chi(v) \leq \text{deg}^-(v) - 1$  and  $\chi(v)$  is a configuration for  $v$ .

A route taken across a DyRIBox  $\mathcal{D} = (V, E)$  is described as an oriented route  $R = (v_0, e_0 \dots v_r, e_r, v_{r+1})$  in the graph  $(V, E)$  with  $v_i \in V$  and  $e_i \in E$  where  $v_0 \in I(\mathcal{D}), v_{r+1} \in O(\mathcal{D})$  and for  $i = \{1 \dots r\}$  is  $v_i \in M(\mathcal{D})$ . The length of  $R$  is define as  $l(R) \stackrel{\text{def}}{=} r$ . The source of the route  $R$  is defined as  $S(R) = v_0$  and the destination as  $D(R) = v_{r+1}$ . The route  $R$  is defined on a DyRIBox as soon as a  $\chi$  exist with:  $\forall i = \{1 \dots r\} \mathcal{S}(\chi(v_i)) = v_{i-1}$  and  $v_{r+1} = \text{succ}(v_r)$ . The predecessor of  $v_i$  through the input of  $\chi(v_i)$  is  $v_{i-1}$ . The configuration  $K_R$  from the route  $R$  is defined as all configurations from the DyRIBox  $\mathcal{D}$ . This provides  $R$  to be configurable on  $\mathcal{D}$ . It is also possible to configure two routes  $R_1$  and  $R_2$  on one DyRIBox  $\mathcal{D}$  using two separate multiplexer nodes with  $R_1 = (v_0, e_0 \dots v_r, e_r, v_{r+1})$  and  $R_2 = (v'_0, e'_0 \dots v'_s, e'_s, v'_{s+1})$ .  $R_1$  and  $R_2$  are compatible when the two nodes are completely disconnected:  $v_i \neq v'_j$  ( $\forall i \in \{1 \dots r\}$  and  $\forall j \in \{1 \dots s\}$ ).

A massively parallel processor array can be described as an oriented acyclic graph  $\mathcal{A} = (\mathcal{D}, \mathcal{P}, E)$ , where  $E$  is the set of oriented edges,  $\mathcal{D}$  is the set of DyRIBoxes used, and  $\mathcal{P}$  is the set of PEs with  $\mathcal{D} = \|\mathcal{D}_n\|$  ( $n \in \mathbb{N}^+$ ) and  $\mathcal{P} = \|\mathcal{P}_{ij}\|$  ( $\forall i \in \{1 \dots M\}$  and  $\forall j \in \{1 \dots N\}$ ). Considering DyRIBox inputs  $\mathcal{I}(\mathcal{D})$ , DyRIBox outputs  $\mathcal{O}(\mathcal{D})$ , PE inputs  $\mathcal{I}(\mathcal{P})$ , and PE outputs  $\mathcal{O}(\mathcal{P})$ , all the following connections are feasible (in Fig. 4):  $e = (v \in \mathcal{O}(\mathcal{D}), v \in \mathcal{I}(\mathcal{D}))$ ,  $e = (v \in \mathcal{O}(\mathcal{D}), v \in \mathcal{I}(\mathcal{P}))$ ,  $e = (v \in \mathcal{O}(\mathcal{P}), v \in \mathcal{I}(\mathcal{D}))$ , and  $e = (v \in \mathcal{O}(\mathcal{P}), v \in \mathcal{I}(\mathcal{P}))$  with  $e \in E$ .

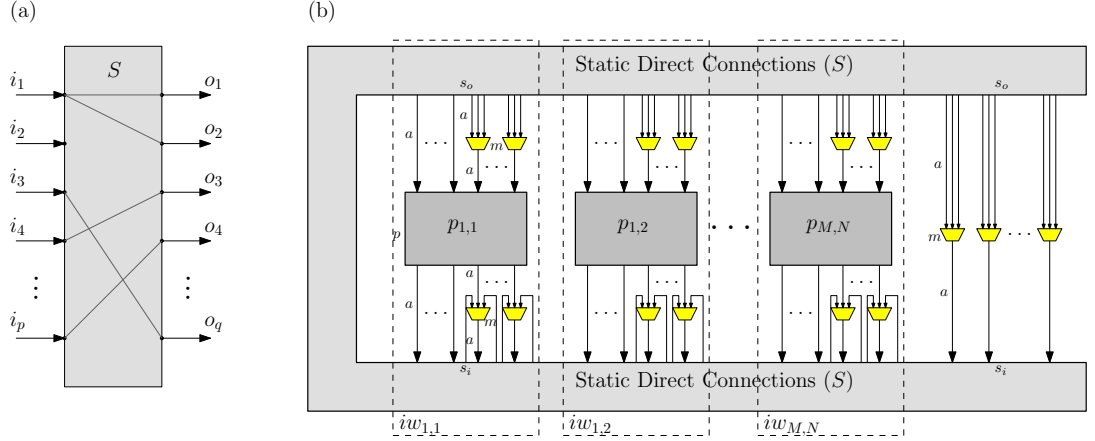


Figure 5: Parallel processor architecture general case representation. In (a), an example of a static direct connector is shown, in (b) the general case of interconnect-wrapper interconnection is presented.

### 2.3 Comparison of Interconnect-Wrapper and DyRIBox Concepts

In the following, we compare both interconnection concepts. In fact, the DyRIBox concept allows for modeling of all possible interconnect topologies ranging from linear arrays, meshes, multistage hierarchical networks, butterflies, trees, etc. A special case of a DyRIBox network is a full crossbar, when the number of DyRIBox inputs and outputs is equal. In this section, we prove that interconnect-wrapper and DyRIBox interconnect definitions are equivalent.

In order to generalize the parallel processor architecture with configurable interconnect and to separate the dynamically reconfigurable interconnection from static connections we introduce an interconnect object called *static direct connector* (SDC).

**Definition 2.4 (Static Direct Connector)** A static direct connector (SDC)  $S = (S^{in}, S^{out})$  is an interconnection black-box or a tuple with an infinite set of input ports  $S^{in} = \{i_1, \dots, i_p\}$  and infinite set of output ports  $S^{out} = \{o_1, \dots, o_q\}$ , where  $p \rightarrow \infty$ ,  $q \rightarrow \infty$ . The input ports are statically connected to the output ports inside of SDC.

An SDC provides a formal description of a static, i.e. not configurable, interconnection network. A graphical representation of an SDC is shown in Fig. 5 (a).

As shown in Section 2.1, a generic massively parallel processor architecture with IW-based interconnect structure can be represented as a netgraph  $G^{IW} = (V, E)$  (see Definition 2.3). Due to Definition 2.4, all edges  $E$  of this directed graph can be represented as an SDC. Therefore, a generic parallel processor architecture with IW-based interconnect topology can be considered as shown in Fig. 5 (b). There are only few types of routes or signal paths through the interconnect-wrappers  $\{iw_{1,1}, \dots, iw_{M,N}\}$  possible. All of them are shown in Fig. 5 (b): (i), a direct route through the process-

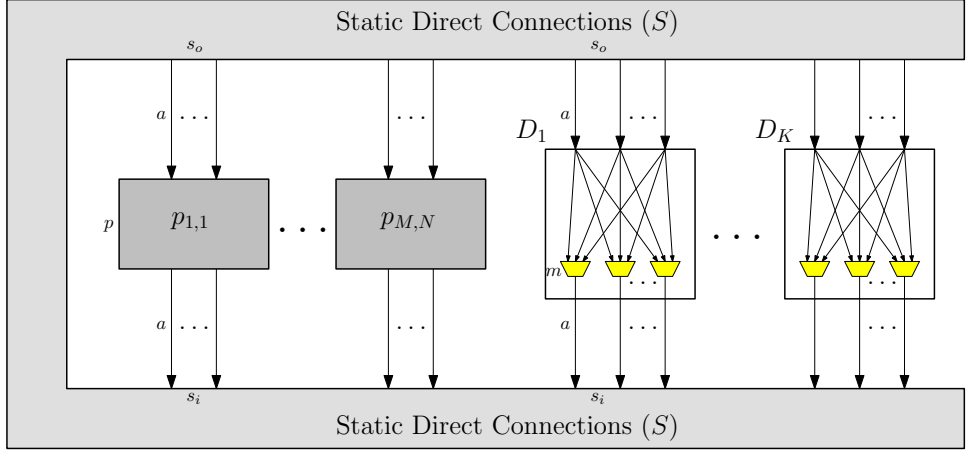


Figure 6: A general case of DyRIBox Interconnection.

ing element inside of the IW, (ii), a route through the multiplexer directly connected to one of the PE inputs (this is the case, when more than one line are connected to the input of the PE), (iii), a route through the PE inside of the IW, where one of the PE outputs is directly connected to the multiplexer (this is the case, when more than one line are connected to one of the IW output ports), (iv), a route through the multiplexer directly connected to one of the PE inputs, where one of the PE outputs is directly connected to another multiplexer, and (v), a direct route through the multiplexer inside of the IW not passing the processing element through (this is the case, when IW input ports are directly connected to IW output port; this route is transformed into direct connection without passing through the multiplexer, when only one IW input port is involved).

Section 2.2 shows that a generic massively parallel processor architecture with DyRIBox-based interconnect structure can be represented as a netgraph  $\mathcal{D} = (V, E)$ . Therefore, in the same manner as for the interconnect-wrapper-based concept, an architecture with DyRIBox definition can be represented with an SDC as shown in Fig. 6. The following signal paths are shown in this case: (i), a direct route through the processing element, (ii), a direct route through the set of multiplexers which compound a DyRIBox (in case of only one DyRIBox input and output, this route is transformed into a direct connection). In order to describe the generalized interconnect concepts formally, we make the following assumptions: Let  $s_i$ ,  $s_o$  be the input and output ports of the static direct connector, respectively. Let  $a$  be a direct connection, let  $p$  be a route through the PE, and let  $m$  be a route through the multiplexer.

Let  $G^{IW}$  and  $G^{DB}$  be formal grammars [8]. Now, the definitions of IW and DyRIBox interconnect can be considered as two formal languages  $L^{IW}(G^{IW})$  and  $L^{DB}(G^{DB})$ , respectively [8]. The definition of the formal languages is given in Fig. 7. The grammar  $G^{IW}$  for language  $L^{IW}$  is defined by the finite set  $\{S, A, M\}$  of nonterminal symbols, by the alphabet of terminal symbols  $\{s_i, s_o, a, p, m\}$ , and by the set

$$\begin{array}{ll}
\begin{array}{l}
G^{IW} = ( \quad L^{IW}(G^{IW}) \\
P^{IW} = \{ \quad \{S, A, M\}, \{s_i, s_o, a, p, m\}, P^{IW}, S) \\
\quad S \rightarrow s_o A s_i S, \\
\quad S \rightarrow s_o A p A s_i S, \\
\quad A \rightarrow a, \\
\quad A \rightarrow a M a, \\
\quad M \rightarrow m, \\
\quad S \rightarrow \varepsilon \}
\end{array}
&
\begin{array}{l}
\quad L^{DB}(G^{DB}) \\
G^{DB} = ( \quad \{D, E, F\}, \{s_i, s_o, a, p, m\}, P^{DB}, D) \\
P^{DB} = \{ \quad D \rightarrow s_o E s_i D, \\
\quad E \rightarrow a, \\
\quad E \rightarrow a F a, \\
\quad F \rightarrow m, \\
\quad F \rightarrow p, \\
\quad D \rightarrow \varepsilon \}
\end{array}
\end{array}$$

Figure 7: Representation of general case interconnect as formal language. In (a), a definition of formal language  $L^{IW}(G^{IW})$  for IW is shown, in (b), a definition of formal language  $L^{DB}(G^{DB})$  for DyRIBox is presented.

$P^{IW}$  of production rules with start symbol  $S$ . The nonterminals  $S$ ,  $A$ , and  $M$  represent the possible signal paths through the interconnect-wrapper. The grammar  $G^{DB}$  for language  $L^{DB}$  is defined by the finite set  $\{D, E, F\}$  of nonterminal symbols, by the alphabet of terminal symbols  $\{s_i, s_o, a, p, m\}$ , and by the set  $P^{DB}$  of production rules with start symbol  $D$ . The nonterminals  $D$ ,  $E$ , and  $F$  represent the possible signal paths through the DiRIBox. An  $\varepsilon$  in both grammars denotes the empty string, i.e., the string of length 0 such that  $\varepsilon \notin L^{IW} \wedge \varepsilon \notin L^{DB}$ . The grammars  $G^{IW}$  and  $G^{DB}$  are context-free grammars, as the left hand sides of a their production rules are formed by only a single non-terminal symbol. Moreover, both of the grammars have the same alphabet of terminal symbols  $\{s_i, s_o, a, p, m\}$ . Therefore, we will prove the equivalence of modeling power of both concepts by proving language equivalence.

**Theorem 2.1** *The PE interconnection networks based on IW are equivalent to the PE interconnection networks based on DyRIBox:  $L^{IW}(G^{IW}) \equiv L^{DB}(G^{DB})$ .*

**Proof 2.1** *Two formal languages are equivalent if their grammars are equivalent. The equivalence of the grammars must be shown in both directions  $L^{DB}(G^{DB}) \subseteq L^{IW}(G^{IW}) \wedge L^{IW}(G^{IW}) \subseteq L^{DB}(G^{DB})$  in order to prove the equivalence of the languages.*

*First, the relation  $L^{DB}(G^{DB}) \subseteq L^{IW}(G^{IW})$  will be shown. Obviously, the following production rules are similar:  $M \rightarrow m \sim F \rightarrow m$ ,  $A \rightarrow a \sim E \rightarrow a$ ,  $A \rightarrow a M a \sim E \rightarrow a F a$ ,  $S \rightarrow \varepsilon \sim D \rightarrow \varepsilon$ , and  $S \rightarrow s_o A s_i S \sim D \rightarrow s_o E s_i D$ . The similarity of production rules  $S \rightarrow s_o A p A s_i S$  and  $F \rightarrow p$  remains questionable. Consider the production rule  $S \rightarrow s_o A p A s_i S$ . After applying of a rule  $A \rightarrow a M a$  on it, the derivation  $S \rightarrow s_o a M a p a M a s_i S$  occurs. Let's introduce the equivalent transformation  $T^{IW} : a \simeq a s_i s_o a$ . This transformation is equivalent because it physically means just a splitting of a direct connection  $a$  into two direct connections which are connected through the input ( $s_i$ ) and output ( $s_o$ ) ports of an SDC (see Fig. 8). Applying the transformation  $T^{IW}$  on the terminals  $a$  staying next to the terminal symbol  $p$ , the following production rule is obtained:  $S' = T^{IW}(S \rightarrow$*

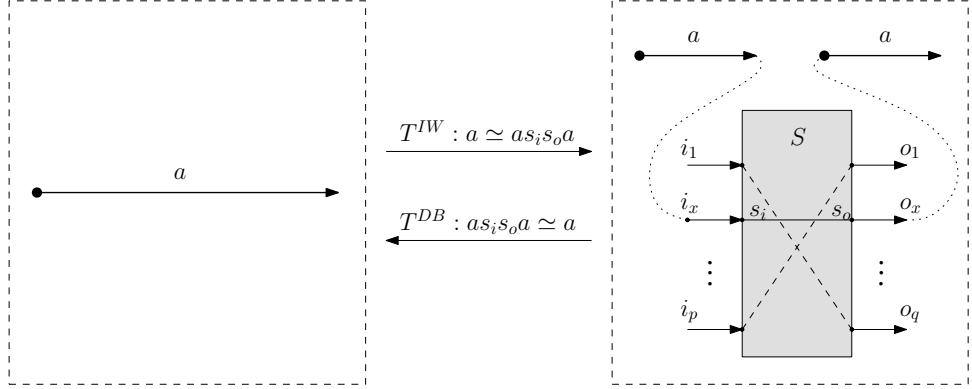


Figure 8: The equivalent transformations  $T^{IW} : a \simeq as_i s_o a$  and  $T^{DB} : as_i s_o a \simeq a$  are presented.

$s_o a M a p a M a s_i S$ ) :  $S' \rightarrow s_o a M a s_i s_o a p a s_i s_o a M a s_i S$ . The subsequence of symbols  $s_o a M a s_i$  in  $S'$  occurs twice and corresponds to the rule  $S \rightarrow s_o A s_i S$  which is a start symbol of  $G^{IW}$ . Therefore,  $L^{IW}$  will not be changed by the elimination of this subsequence. Thus,  $S'$  takes the following form:  $S' \rightarrow s_o a p a s_i S$ . We get the following set of production rules:  $\{S \rightarrow s_o A s_i S, A \rightarrow a, A \rightarrow a M a, M \rightarrow m, S \rightarrow \varepsilon, S' \rightarrow s_o a p a s_i S\}$ . After integration of  $S'$  into  $S$  the set of production rules can be rewritten:  $\{S' \rightarrow s_o A' s_i S', A' \rightarrow a, A' \rightarrow a M' a, M' \rightarrow m, M' \rightarrow p, S' \rightarrow \varepsilon\} \simeq P^{DB}$ . Thus,  $L^{DB}(G^{DB}) \subseteq L^{IW}(G^{IW})$ .

Now, the relation  $L^{IW}(G^{IW}) \subseteq L^{DB}(G^{DB})$  will be shown. Obviously, the following production rules are similar:  $F \rightarrow m \sim M \rightarrow m$ ,  $E \rightarrow a \sim A \rightarrow a$ ,  $E \rightarrow a F a \sim A \rightarrow a M a$ ,  $D \rightarrow \varepsilon \sim S \rightarrow \varepsilon$ , and  $D \rightarrow s_o E s_i D \sim S \rightarrow s_o A s_i S$ . The similarity of production rules  $F \rightarrow p$  and  $S \rightarrow s_o A p A s_i S$  remains questionable. Let's include the production rule  $D \rightarrow s_o E s_i s_o E s_i s_o E s_i D$  in  $P^{DB}$ . Language  $L^{DB}$  is not changed by this operation because  $D \rightarrow s_o E s_i s_o E s_i s_o E s_i D \equiv D \rightarrow s_o E s_i D$ . This is an obvious derivation after three times applying the recursive production rule  $D \rightarrow s_o E s_i D$ . After applying the rule  $E \rightarrow a F a$  on the new included production rule, we obtain  $D \rightarrow s_o a F a s_i s_o a F a s_i s_o a F a s_i D$ . Let's introduce the equivalent transformation  $T^{DB} : as_i s_o a \simeq a$ . This transformation is equivalent because it physically means replacing of a connection through the input ( $s_i$ ) and output ( $s_o$ ) ports of SDC with a direct connection  $a$  (see Fig. 8). Applying the transformation  $T^{DB}$  on the subsequences of terminals  $as_i s_o a$ , the following production rule is obtained:  $D' = T^{DB}(D \rightarrow s_o a F a s_i s_o a F a s_i s_o a F a s_i D) : D' \rightarrow s_o a F a F a F a s_i D$ . The subsequence of symbols  $a F a$  can be replaced by the nonterminal symbol  $E$  using the production rule  $E \rightarrow a F a$ :  $D' \rightarrow s_o E F E s_i D$ . Applying the rule  $F \rightarrow p$ ,  $D'$  can be rewritten:  $D' \rightarrow s_o E p E s_i D$ . We get the following set of production rules:  $\{D \rightarrow s_o E s_i D, E \rightarrow a, E \rightarrow a F a, F \rightarrow m, F \rightarrow p, D \rightarrow \varepsilon, D' \rightarrow s_o E p E s_i D\}$ . After integration of  $D'$  into  $D$  the set of production rules can be rewritten:  $\{D' \rightarrow s_o E' s_i D', E' \rightarrow a, E' \rightarrow a F' a, F' \rightarrow m, D' \rightarrow s_o E' p E' s_i D', D' \rightarrow \varepsilon\} \simeq P^{IW}$ .

Thus,  $L^{IW}(G^{IW}) \subseteq L^{DB}(G^{DB})$ .  $\square$

According to Theorem 2.1, the set of interconnection networks that can be expressed using the interconnect-wrapper definition (distributed interconnection concept) is equivalent to those that can be modeled using the DyRIBox definition (centralized interconnection concept).

## 3 Modeling of Interconnect Structures within MAML

In order to allow the specification of reconfigurable interconnect topologies in massively parallel processor architectures we use the *MAchine Markup Language* (MAML) [4, 10]. MAML is based on the XML notation and is used for describing architecture parameters required by possible mapping methods such as partitioning, scheduling, functional unit and register allocation. Moreover, the parameters extracted from a MAML architectural description can be used for interactive visualization and simulation of the given processor architecture. For the MAML description of interconnect-wrappers we refer to [10, 9].

### 3.0.1 MAML description of DyRIBox and DyRIBox domain.

The structure of the DyRIBOX element description is shown in Fig. 9. In MAML, a DyRIBox is represented by the `<PEInterconnectDyRIBox>` element. It contains the attribute name which specifies the name of the DyRIBox which is going to be described. It also contains the following set of subelements: `<ReconfigurationTime>`, `<DBPorts>`, and `<PElementsPorts>`. The subelement `<ReconfigurationTime>` specifies the number of cycles needed to dynamically reconfigure the complete DyRIBox. This subelement is initialized by giving a value to the `cycle` parameter. The subelement `<DBPorts>` specifies the number of ports directly connected to others DyRIBOXes. The number of `Inputs`, `Outputs` and their bitwidth are specified by the attributes. The subelement `<PElementsPorts>` specifies the number of ports directly connected to the ports of processing elements. The number of `Inputs`, `Outputs` and their bitwidth are specified by the attributes.

**The DyRIBox domain description** A DyRIBox domain `<DBDomain>` is typically used for describing the interconnections of heterogeneous architectures. The structure of the DyRIBOX domain description is shown in Fig. 10. The element `<DBDomain>` is followed by the attribute name which specifies the name of the DyRIBox domain which is going to be described. It also contains a set of subelements:

- `<Interconnect>`

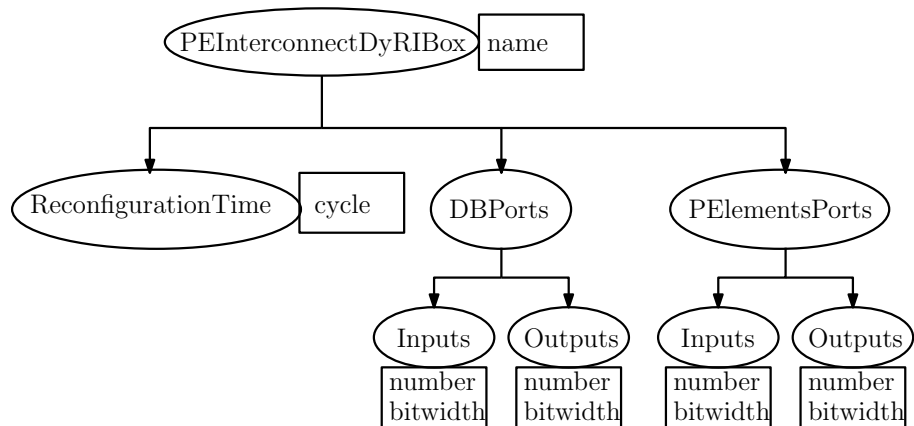


Figure 9: The `<PEInterconnectDyRIBox>` elements.

- `<ElementsPolytopeRange>`

`<Interconnect>` specifies how the interconnections are going to be described by giving a `type`. The `type` can be `manual` when all connections are explicitly described, or `mesh` or `honeycomb` or `tree` when the connections are automatically generated as a mesh form or honeycomb form or tree form. In case that `<Interconnect>` is a `manual` type, the subelement `<Instantiation>` specifies which `DyRIBox` is being used and follows the subelement `<InternalConnections>` which describe the internal interconnections of the domain ports by ports one after the other. `<ElementsPolytopeRange>` specifies a range of PEs in shape of a given polytope which belong to the domain which is going to be described. Concerning the interconnections with external resources of the `DyRIBox` domain, an element `<PortMapping>` is given at the end of the `<ProcessorArray>` description. In Fig. 12, a `DyRIBox` example using MAML is given. This example implements four processor elements and one `DyRIBox` (see Fig. 11).

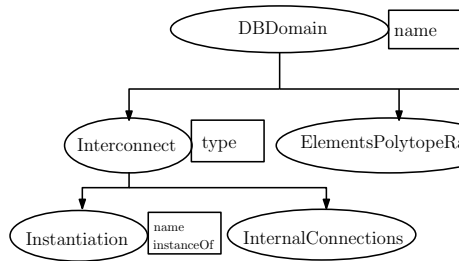


Figure 10: The <DBDomain> elements.

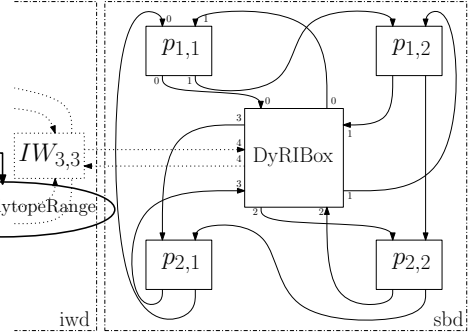


Figure 11: MAML example with the use of the DyRIBox concept.

```

<ProcessorArray name= "processor" version="1.0">
  <PElements name="PE" rows="2" columns="2"/>
  <PEInterconnectDyRIBox name="DB">
    <ReconfigurationTime cycle="4"/>
    <DBPorts>
      <Inputs number="0" bitwidth="8" />
      <Outputs number="0" bitwidth="8" />
    </DBPorts>
    <PElementsPorts>
      <Inputs number="4" bitwidth="8" />
      <Outputs number="4" bitwidth="8" />
    </PElementsPorts>
  </PEInterconnectDyRIBox>
  <DBDomain name="dbd">
    <Interconnect type="manual">
      <Instantiation name="DyRIBox" instanceOf="DyRIBox">
      <InternalConnections>
        <DyRIBox_in0 = PE[1,1]_out0/>
        <DyRIBox_in1 = PE[1,2]_out0/>
        <DyRIBox_in2 = PE[2,1]_out0/>
        <DyRIBox_in3 = PE[2,2]_out0/>
        <DyRIBox_out0 = PE[1,1]_in1/>
        <DyRIBox_out1 = PE[1,2]_in1/>
        <DyRIBox_out2 = PE[2,1]_in0/>
        <DyRIBox_out3 = PE[2,2]_in0/>
        <DyRIBox_out4 = PE[1,1]_out1/>
        <DyRIBox_out5 = PE[1,2]_out1/>
        <DyRIBox_out6 = PE[2,1]_out1/>
        <DyRIBox_out7 = PE[2,2]_out1/>
      </InternalConnections>
    </Instantiation>
  </Interconnect>
  <ElementsPolytopeRange>
    <MatrixA row = " -1 0"/>
    <MatrixA row = " 1 0"/>
    <MatrixA row = " 0 -1"/>
    <MatrixA row = " 0 1"/>
    <VectorB value = " -1"/>
    <VectorB value = " 2"/>
    <VectorB value = " -1"/>
    <VectorB value = " 2"/>
  </ElementsPolytopeRange>
  </DBDomain>
  <PortMapping>
    <dbd_DyRIBox_in4 = iwd_IW[3,3]_Eout0/>
    <dbd_DyRIBox_out4 = iwd_IW[3,3]_Eout1/>
  </PortMapping>
</ProcessorArray>

```

Figure 12: MAML description of a DyRIBox-based architecture.

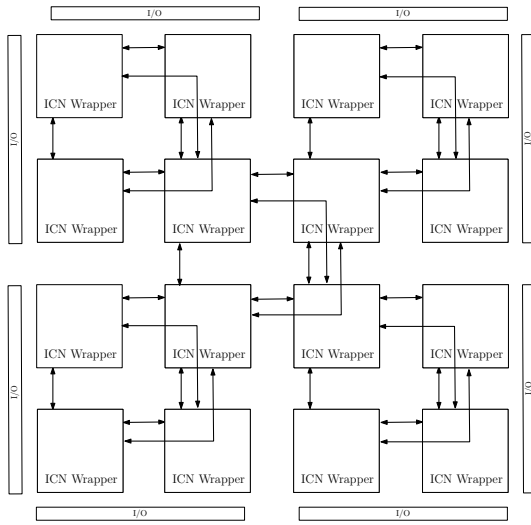


Figure 13: BFT interconnection scheme embedded in a  $4 \times 4$  array.

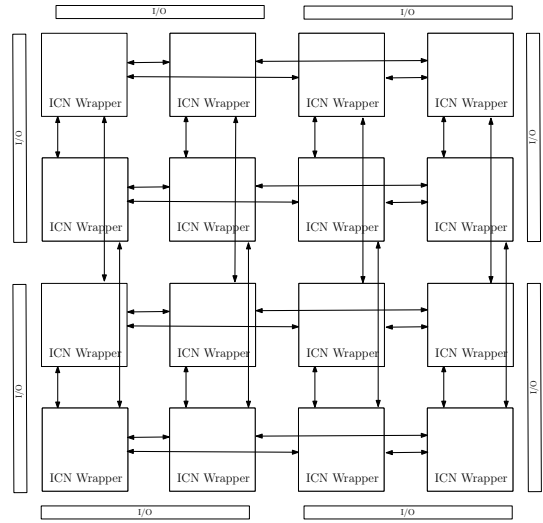


Figure 14: 4D Hypercube interconnection scheme embedded in a  $4 \times 4$  array.

## 4 Case-Study

We implemented a highly parameterizable template for the generation of parallel processor arrays in VHDL (*Very High Speed Hardware Description Language*). An example  $4 \times 4$  processor array was instantiated and tested on a *Xilinx Virtex-II Pro<sup>TM</sup> xc2vp100* FPGA. To reduce synthesis and mapping time, a very simple configuration was chosen for all processing elements in the array. Each WPPE was configured to contain two adder modules and one module for the transfer of control signals. The data path width was chosen to be 16 bit. Different interconnection schemes were implemented. The logical interconnection scheme for the *4D hypercube* topology is depicted in Fig. 14 and for the *butterfly fat tree (BFT)* topology in Fig. 13. The results are shown in Table 1. It can be seen from the synthesis results, that starting from the basic mesh interconnection scheme additional interconnection configurations can be added at relative little hardware cost of roughly 2% in the case of the *4D hypercube* topology. In the case of the 4D hypercube with the additional butterfly fat tree topology the hardware cost amounts to roughly 1%.

Table 1: Synthesis results for different interconnection networks in a  $4 \times 4$  processor array.

	<i>Mesh</i>	<i>4D Hypercube</i>	<i>4D Hypercube &amp; Mesh</i>	<i>BFT &amp; 4D Hypercube</i>
Equivalent Gates	859482	865941	880125	875412
$MUX_1^4(16)$	32	32	33	37
$MUX_1^5(16)$	0	0	0	7
Logic LUT	30287	31724	33296	32883
Route-Through LUT	302	244	167	140
Memory LUT	4768	4768	4768	4768
FPGA Resources	50%	51%	52%	52%

## 5 Conclusions and Future Work

In this report, we introduced a new concept for modeling of interconnection networks in the field of massively parallel processor embedded architectures. Two interconnection concepts, namely, *interconnect-wrapper* and *DyRIBox* definitions of reconfigurable interconnection were formally defined and compared against each other. The equivalence of distributed and centralized interconnection concepts was formally proved which also proved the ability of the *interconnect-wrappers* to efficiently model many possible interconnect topologies. Moreover, we demonstrated the pertinence of our approach by synthesis of a real  $4 \times 4$  processor array with different reconfigurable interconnect topologies in a case-study.

In the future, we would like to extend MAML in order to model different transport mechanisms like blocking FIFOs, asynchronous data-driven models, or even concepts such as routers in Networks-on-a-Chip (NoC). This would allow for architecture exploration of a vast class of processor arrays.

## References

- [1] N. Bansal, S. Gupta, N. Dutt, A. Nicolau, and R. Gupta. Network Topology Exploration of Mesh-Based Coarse-Grain Reconfigurable Architectures. In *Proceedings Design Automation and Test in Europe (DATE'2004)*, pages 474–479, Paris, France, Feb. 2004.
- [2] V. Baumgarte, G. Ehlers, F. May, A. Nüchel, M. Vorbach, and M. Weinhardt. PACT XPP – A Self-Reconfigurable Data Processing Architecture. *The Journal of Supercomputing*, 26(2):167–184, 2003.
- [3] Elixent Ltd. [www.elixent.com](http://www.elixent.com).
- [4] D. Fischer, J. Teich, M. Thies, and R. Weper. Design Space Characterization for Architecture/Compiler Co-Exploration. In *ACM SIG Proceedings International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES 2001)*, pages 108–115, Atlanta, GA, U.S.A., November 2001.
- [5] D. Fischer, J. Teich, M. Thies, and R. Weper. BUILDABONG: A Framework for Architecture/Compiler Co-Exploration for ASIPs. *Journal for Circuits, Systems, and Computers, Special Issue: Application Specific Hardware Design*, pages 353–375, 2003.
- [6] A. Halambi, P. Grun, A. Khare, V. Ganesh, N. Dutt, and A. Nicolau. EXPRESSION: A Language for Architecture Exploration through Compiler/Simulator Retargetability. In *Proceedings Design Automation and Test in Europe (DATE'1999)*, 1999.
- [7] R. Hartenstein. A Decade of Reconfigurable Computing: A Visionary Retrospective. In *Proceedings of Design, Automation and Test in Europe*, pages 642–649, Munich, Germany, Mar. 2001. IEEE Computer Society.
- [8] J. Hopcroft. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Series in Computer Science. Addison Wesley Publishing Company, older edition, Apr. 1979.
- [9] A. Kupriyanov, F. Hannig, D. Kissler, R. Schaffer, and J. Teich. MAML - An Architecture Description Language for Modeling and Simulation of Processor Array Architectures, Part I. Technical Report 03-2006, University of Erlangen-Nuremberg, Department of Computer Science, Hardware-Software-Co-Design, Mar. 2006.
- [10] A. Kupriyanov, F. Hannig, D. Kissler, J. Teich, R. Schaffer, and R. Merker. An Architecture Description Language for Massively Parallel Processor Architectures. In *GI/ITG/GMM-Workshop 2006 - Methoden und Beschreibungssprachen*

zur Modellierung und Verifikation von Schaltungen und Systemen, pages 11–20, Dresden, Germany, Feb. 2006.

- [11] J. Lee, K. Choi, and N. 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.
- [12] B. Mei, A. Lambrechts, D. Verkest, J. Mignolet, and R. Lauwereins. Architecture Exploration for a Reconfigurable Architecture Template. In *IEEE Design and Test of Computers*, pages 90–101, Mar. 2005.
- [13] M. Motomura. A Dynamically Reconfigurable Processor Architecture. In *Microprocessor Forum*, CA, 2002.
- [14] S. Pees, A. Hoffmann, and H. Meyr. Retargeting of Compiled Simulators for Digital Signal Processors Using a Machine Description Language. In *Proceedings Design Automation and Test in Europe (DATE'2000)*, Paris, March 2000.
- [15] Silicon Hive. [www.siliconhive.com](http://www.siliconhive.com).
- [16] Trimaran. <http://www.trimaran.org>.