Hardware Generation from Partitioned Regular Algorithms with Resource Constraints

Marcus Bednara

Jürgen Teich

Fachgebiet Datentechnik
Computer Engineering Lab

Universität Paderborn
University of Paderborn

Overview

Part 1: **Scheduling of processor arrays under resource constraints**

Part 2: **Generating hardware for processor arrays**

Mapping of Regular Algorithms (1)

$$c_{ij} = \sum_{k=1}^{N} a_{ik} b_{kj}$$

$\Rightarrow$

```
for (i=1; i<=N; i++)
{
   for (k=1; k<=N; k++)
   {
      for (j=1; j<=N; j++)
      {
         c[i,j]=c[i,j]+a[i,k]*b[k,j];
      }
   }
}
```

Problem

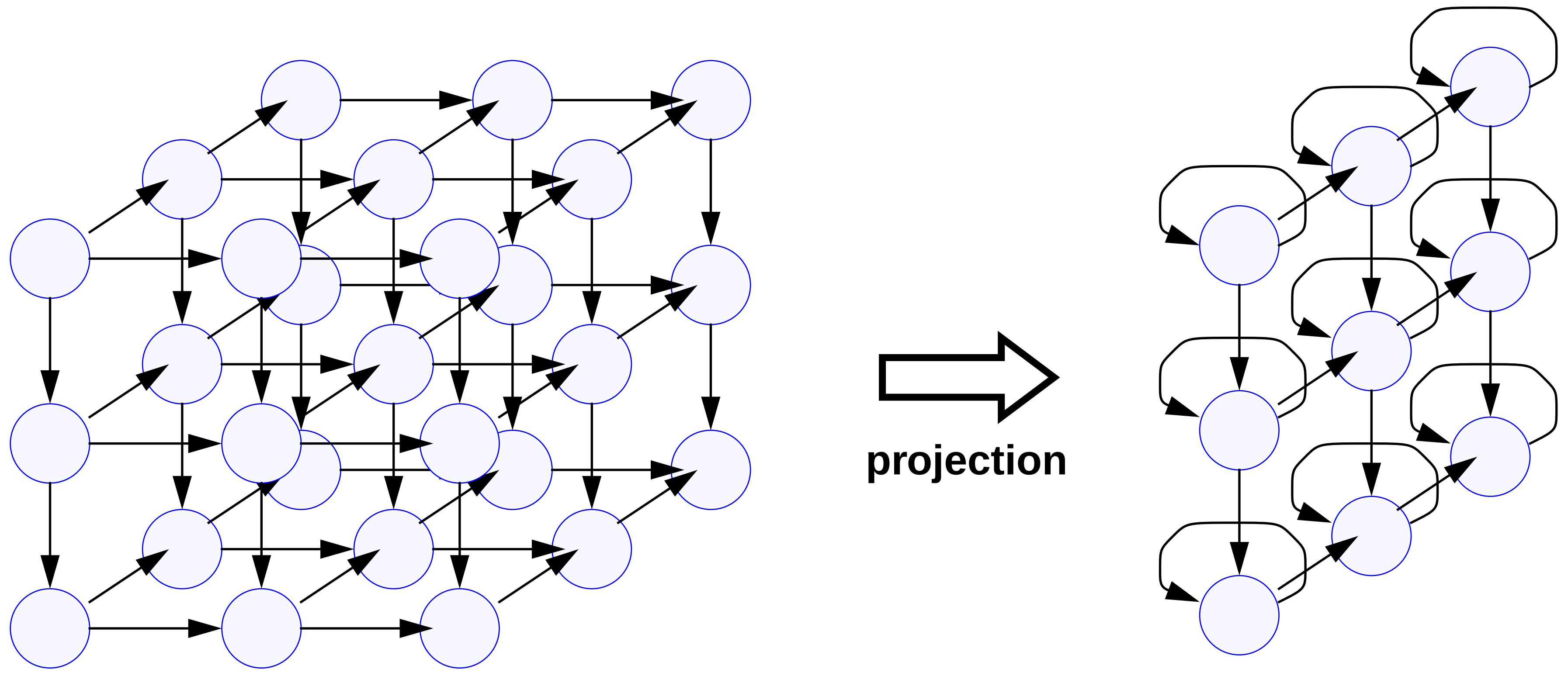
Sequential **loop program**

Mapping of Regular Algorithms (2)

```
PAR i,j,k:
      i>0 AND i<=N AND j>0 AND j<=N AND 0<k AND k<=N
DO
      c[i,j,k]=c[i,j,k-1]+a[i,0,k]*b[0,j,k];
OD
```

Parallel single assignment code

Mapping of Regular Algorithms (3)



Data Flow Graph (DFG)
(localized)

Processor Array

Projection and Scheduling

Projection: **Which operations are processed on the same processing element (PE) ?**

$\longrightarrow$ projection vector u

Scheduling: **Which operations are processed at the same time step (isotemporal hyperplane) ?**

$\longrightarrow$ schedule vector λ

Condition: $\lambda \times u \neq 0$

Resource Constraints

Resources within the processing elements cause several constraints for scheduling:

- **the number of resources (ALUs, memory, multipliers)**
  - **resource sharing is required**

- **the latency of PE operations**
  - **latency of a PE must be taken into account when scheduling the array**

Scheduling

Scheduling problems:

Array scheduling:

- **When do the DFG node operations start ?**

Node scheduling:

- **When do the PE internal operations start ?**

Array schedule and PE schedules may overlap !

Resource Constraints (1)

Example:

$$y_{i,j} = z_{i,j} \times u_{i,j} + a_{i,j} \times (u_{i,j} - c)$$

$$z_{i,j} = y_{i-1,j}$$

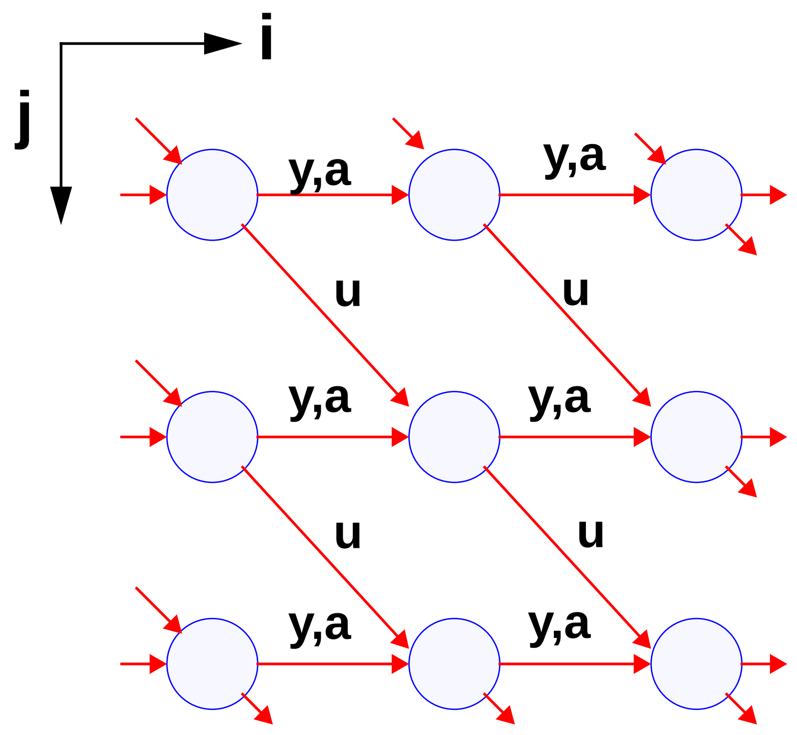
$$u_{i,j} = u_{i-1,j-1} \qquad (0 \le i, j < N)$$

$$a_{i,j} = a_{i-1,j}$$

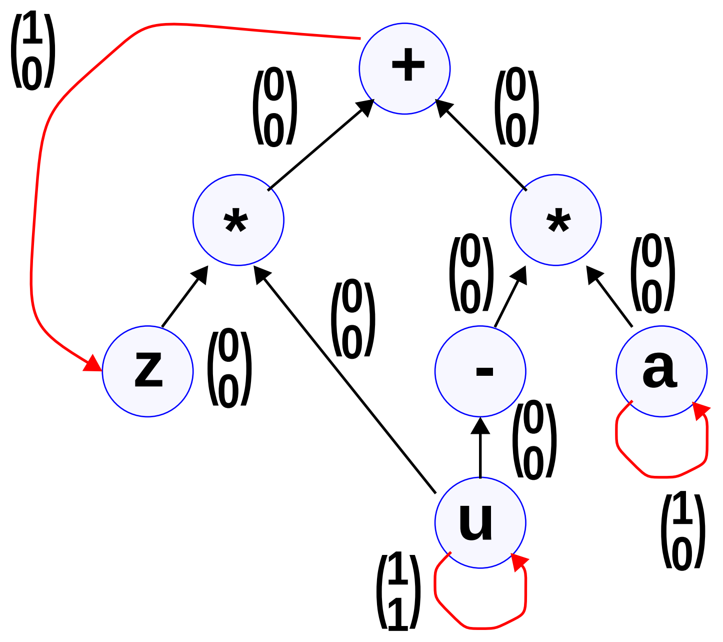
In this example, each operation is mapped to a dedicated resource:

Required resources:	*MUL1, MUL2, ADD, SUB*	
Latencies:	*MUL1, MUL2:*	*4T*
	ADD, SUB:	*T*

DFG and RDG



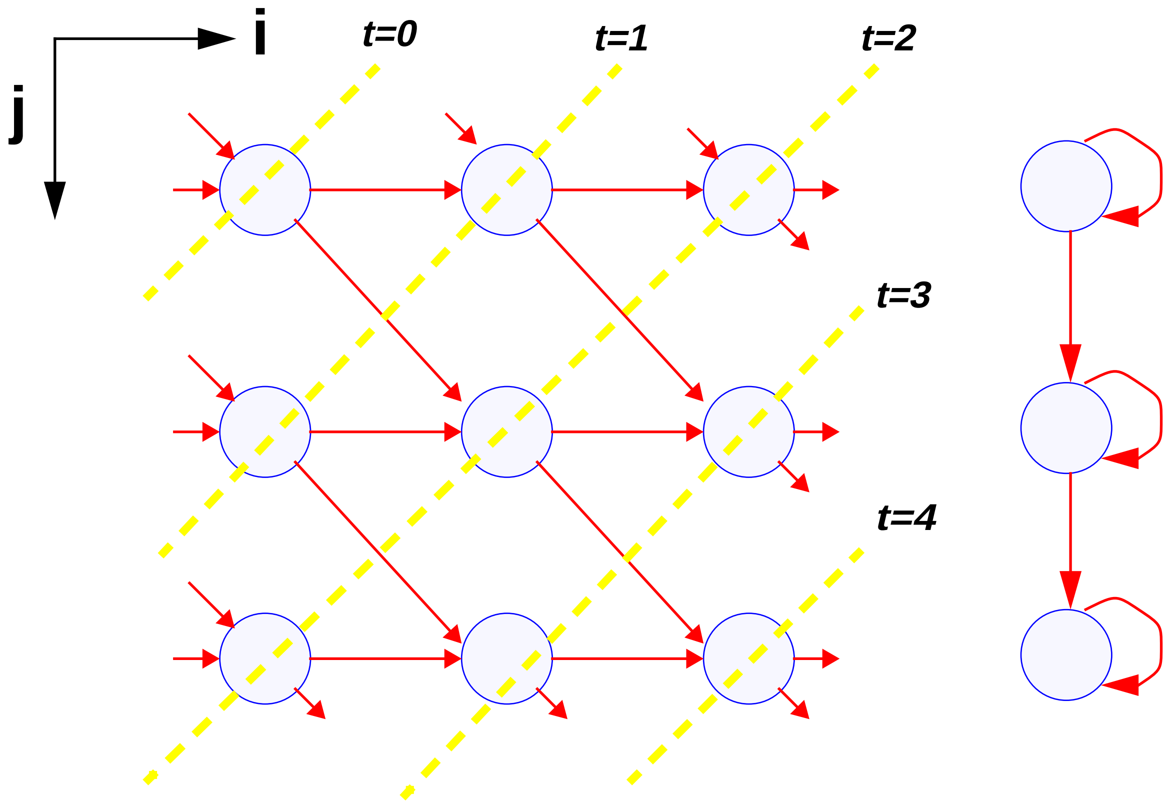
Data Flow Graph
(condensed)

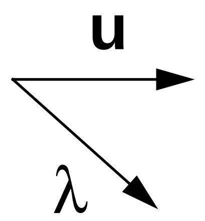


Reduced Dependence
Graph (RDG)

Projection and Scheduling

Projection vector $u = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ Schedule vector λ



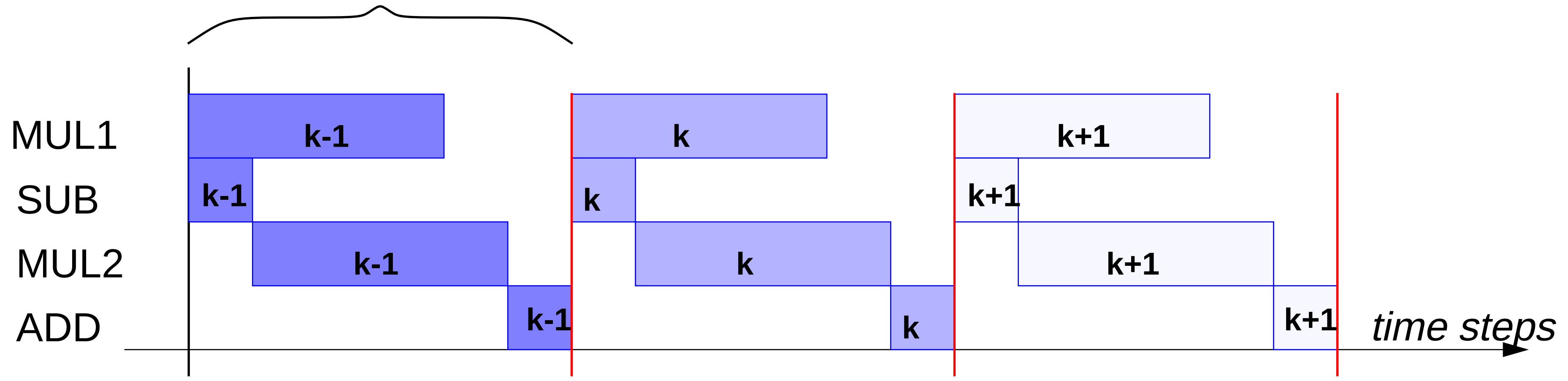


$$P = \lambda \times u$$

Node Schedule (1)

Node internal schedule: $\Gamma = (0, 0, 1, 5)^T$

$$\textit{interval period}\ P = \lambda u = 6 \Rightarrow \lambda = (6, 6)$$

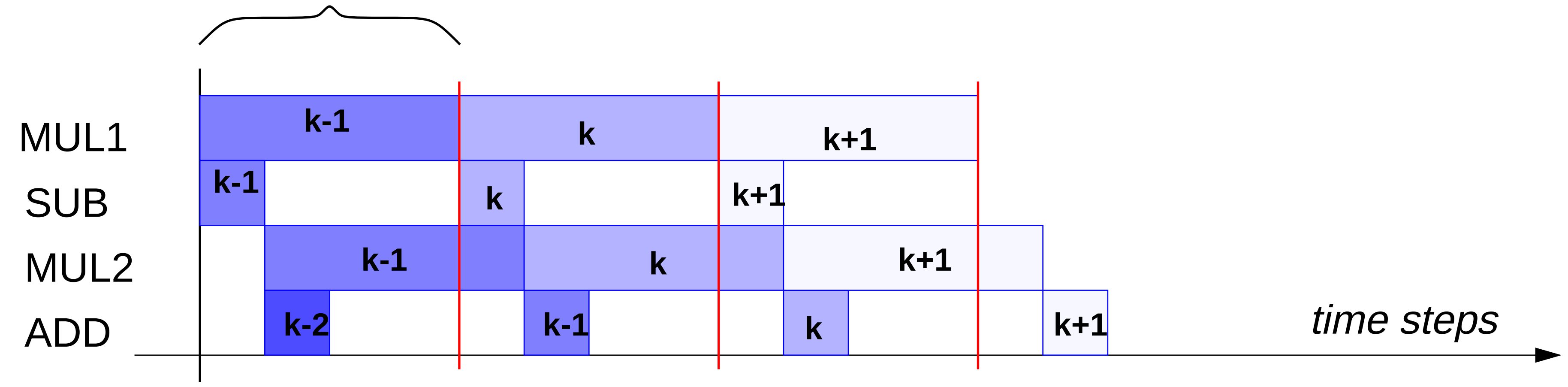


Simple PE internal schedule without overlapping

Node Schedule (2)

Node internal schedule: $\Gamma = (0, 0, 1, 1)^T$

$$\textit{interval period } P = \lambda u = 4 \Rightarrow \lambda = (4, 4)$$



PE internal schedule with funtional pipelining

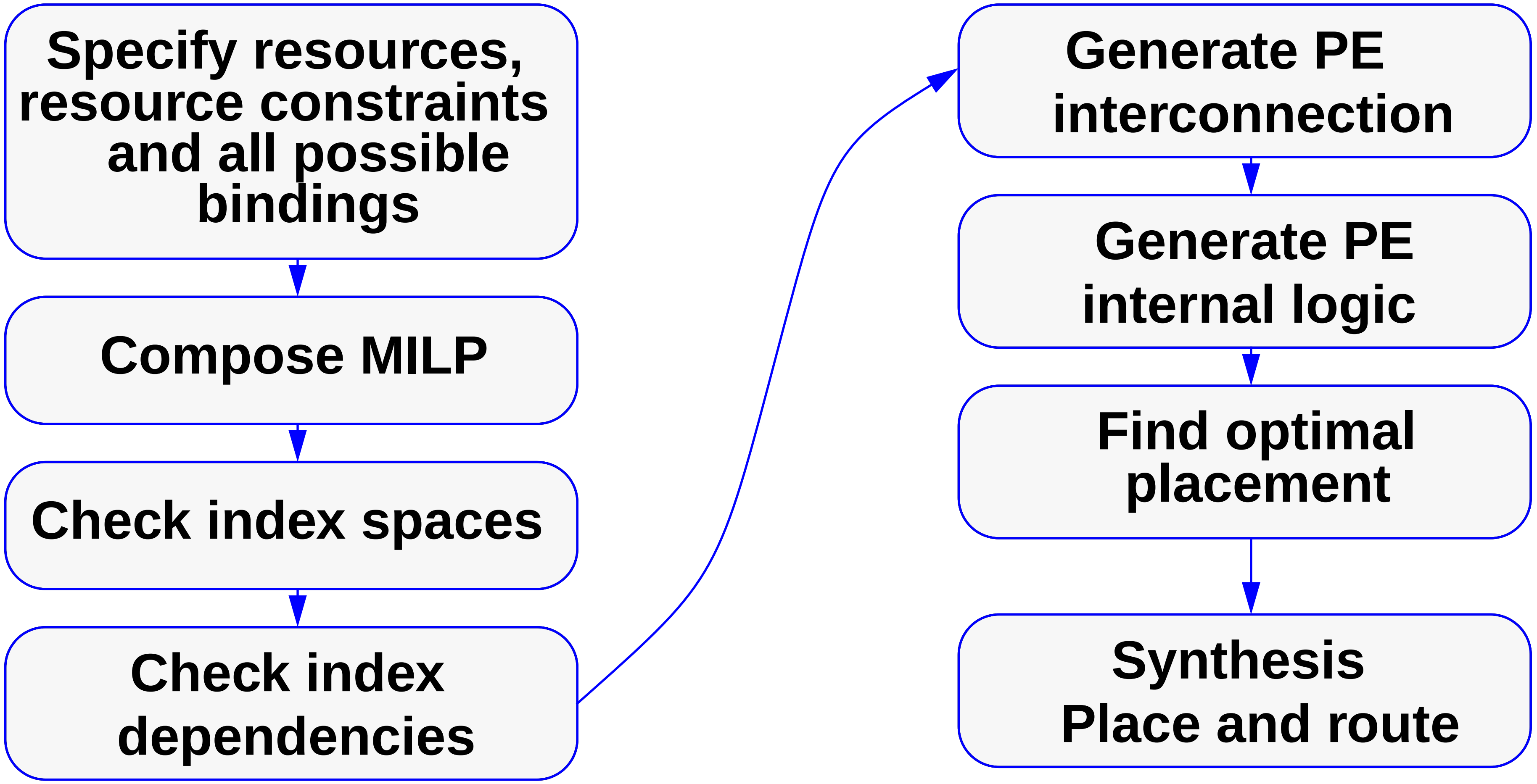
Scheduling

Goal:

Find a *latency minimal* global schedule λ that respects internal resource constraints and exploits functional pipelining.

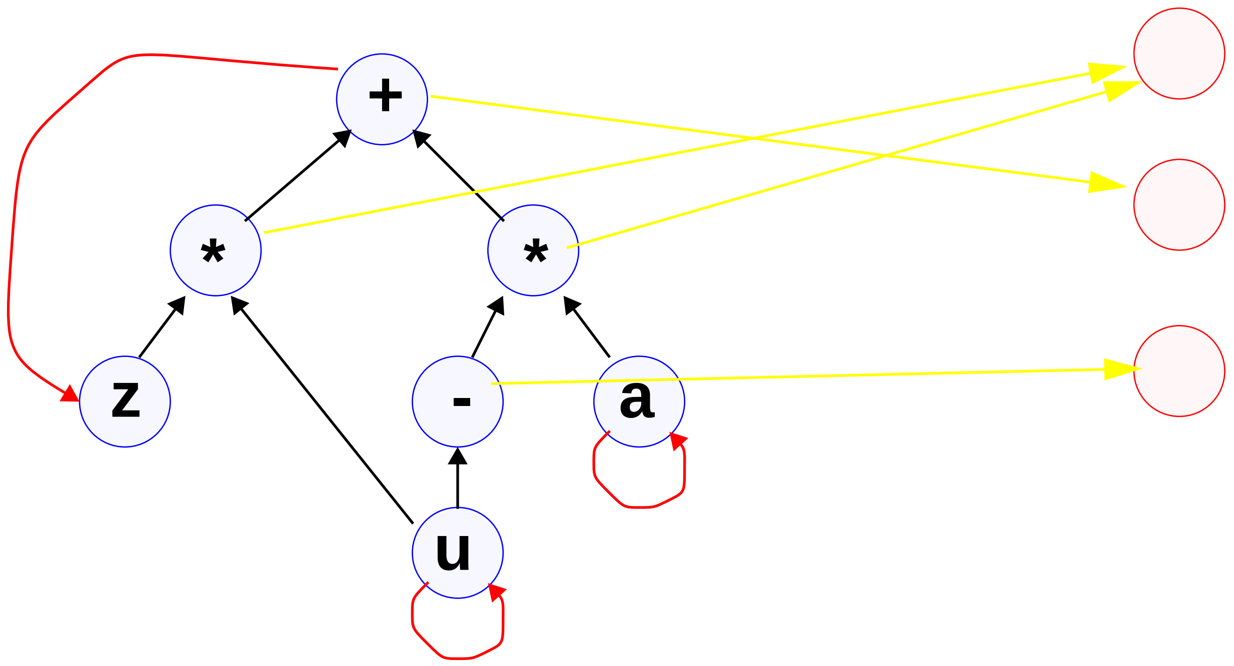
TEICH, J., L. THIELE AND L. ZHANG: SCHEDULING OF PARTITIONED REGULAR ALGORITHMS ON PROCESSOR ARRAYS WITH CONSTRAINED RESOURCES. INT. JOURNAL OF VLSI SIGNAL PROCESSING. 17(1):5 20, SEPTEMBER 1997.

Hardware Generation: Overview


Specify resources, resource constraints and all possible bindings
Compose MILP
Check index spaces
Check index dependencies
Generate PE interconnection
Generate PE internal logic
Find optimal placement
Synthesis Place and route

Resource Specification

Specification of resoures: Build a resource allocation graph



resource type 1 (MUL)

α (MUL)=2

resource type 2 (ADD)

α (ADD)=1

resource type 3 (SUB)

α (SUB)=1

Mixed Integer Linear Program

Compose an integer linear program with the following constraints:

- **The resource allocation graph**
- **The latency *delay(v,r)* of each operation *v* computed on a resource *r*.**
- **The pipeline rate *PR(r)* of each resource *r*.**
- **The index polytope** $A \times I \geq b$

Solve the MILP Problem.

MILP Solution

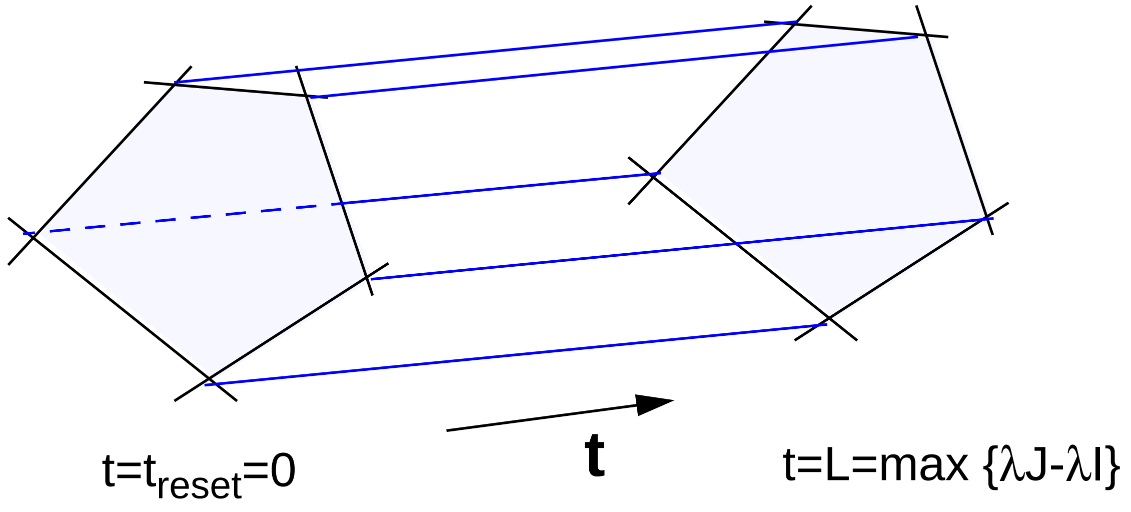
The M(I)LP solution provides the following results:

- Definite binding of each node operation ***v*** to a resource ***r***
- Schedule of all PE internal resources: $\Gamma_{v,r}$ (start time of operation ***v*** on resource ***r***)
- Projection vector ***u*** and array schedule vector λ

Check Index Space

Synchronous Computation Condition: All PE operations must be performed from the beginning (reset time) until the end.

- **The index polytope must be a prism, i.e. processor indices must not depend on time index:**



Check Index Dependencies

Check wether the design can be implemented:

- **Is** $\lambda \times u \neq 0$ **?**

  Operations on the same isotemporal hyperplane can not be computed on the same processor element !

- **Is** $\lambda \times d \geq 0$ **?**

  DFG must be causal, i.e. all dependency vectors must cross the hyperplanes in non-negative directions (semi-systolic condition) !

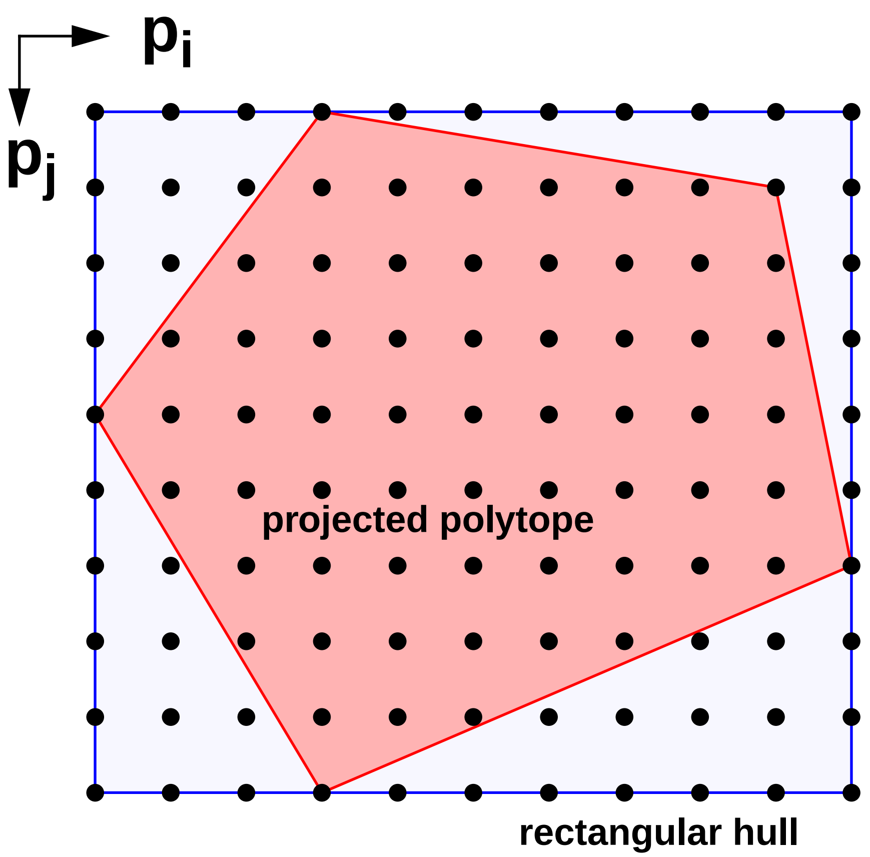
Array structure (1)

Generate the node interconnection structure:

In VHDL:

- **Loop over a rectangular hull covering the whole projected index space.**
- **For each processor index $I=(p_i, p_j)$ check, if a PE instance must be generated (*valid*-function).**
- **Define a grid of indexed signals for the PE interconnection structure.**

Array structure (2)



```
loop_x: FOR pi IN 0 TO N-1 GENERATE
    loop_y: FOR pj in 0 TO N-1 GENERATE

  condition_PE:
  IF (valid_PE (pi,pj) =1) GENERATE
-- Port maps for component PE:
    PE_MAP: pe_instance
    GENERIC MAP (...)
    PORT MAP
    (
       ...
    );

  END GENERATE loop_y;
END GENERATE loop_x;
```

PE internal logic

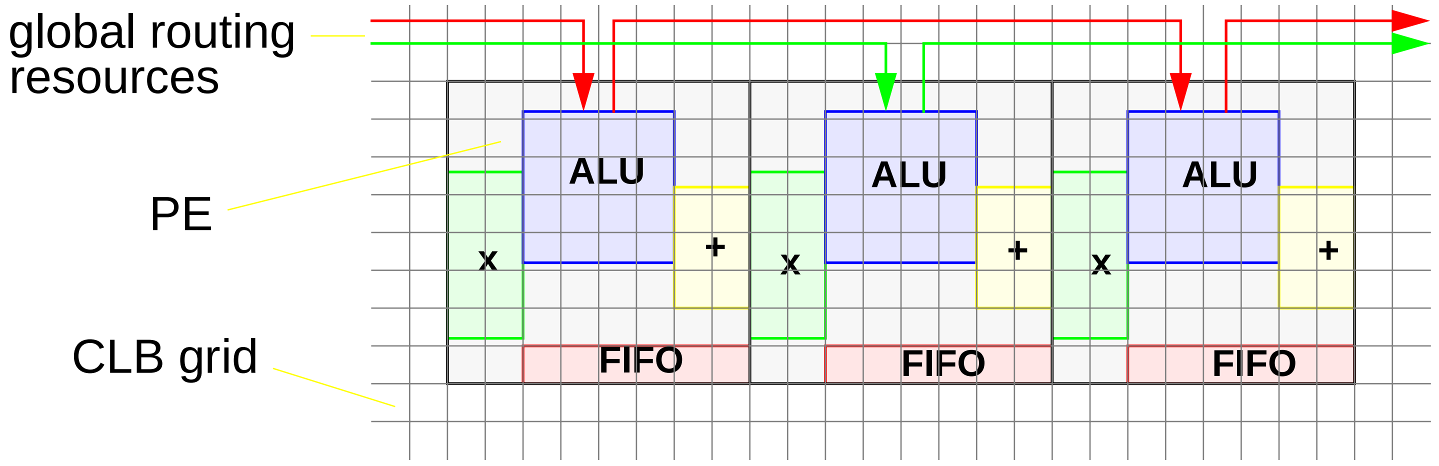
Build the internal logic of a processor element:

- **Operator logic (ALUs, multipliers...)**
- **Input multiplexer for shared resources**
- **Internal fifos used for matching the λ- and Γ- schedule**
- **Controller synthesis for controlling the multiplexers and resource enable signals**

Placement Constraints (1)

Internal placement of PE depends on:

- **connections to the neighbouring PE**
- **routing resources of the FPGA chip**
- **orientation of a PE**



Placement Constraints (2)

For commercial synthesis tools, additional placement constraints must be provided in order to keep the regular structure:

Required: Area estimation of a single array node.

- Estimation from the area requirements of library elements (fast but low accuracy)
- Estimation from a completely synthezised and placed/routed array PE (slow but accurate)
- Build a bounding box containing all logic of a single PE

Placement Constraints (3)

Array Placement.

Let Δx and Δy be the extensions of a PE-bounding box (as CLB count in each direction) and (p_i, p_j) the index coordinates of a certain PE in the projected index polytope.

The position (x,y) of the PE on the FPGA chip simply can be determined as:

$$\boxed{\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \Delta x \times p_i \\ \Delta y \times p_j \end{pmatrix}}$$

Additional borders for routing may be required.

Backends for Hardware Generation

- VHDL backend
  - VHDL code generation for array structure and PE (based on DesignWare components)
  - code generation for a VHDL test bench
  - generation of placing information

- JBits backend (for Virtex FPGA)
  - Generation of a JAVA class hierarchy
  - Complete user control over placing and FPGA resources
  - synthesis process must be implemented completely