

Compilation from Matlab to Process Networks

Edwin Rypkema¹, Ed Deprettere¹,
and Bart Kienhuis²

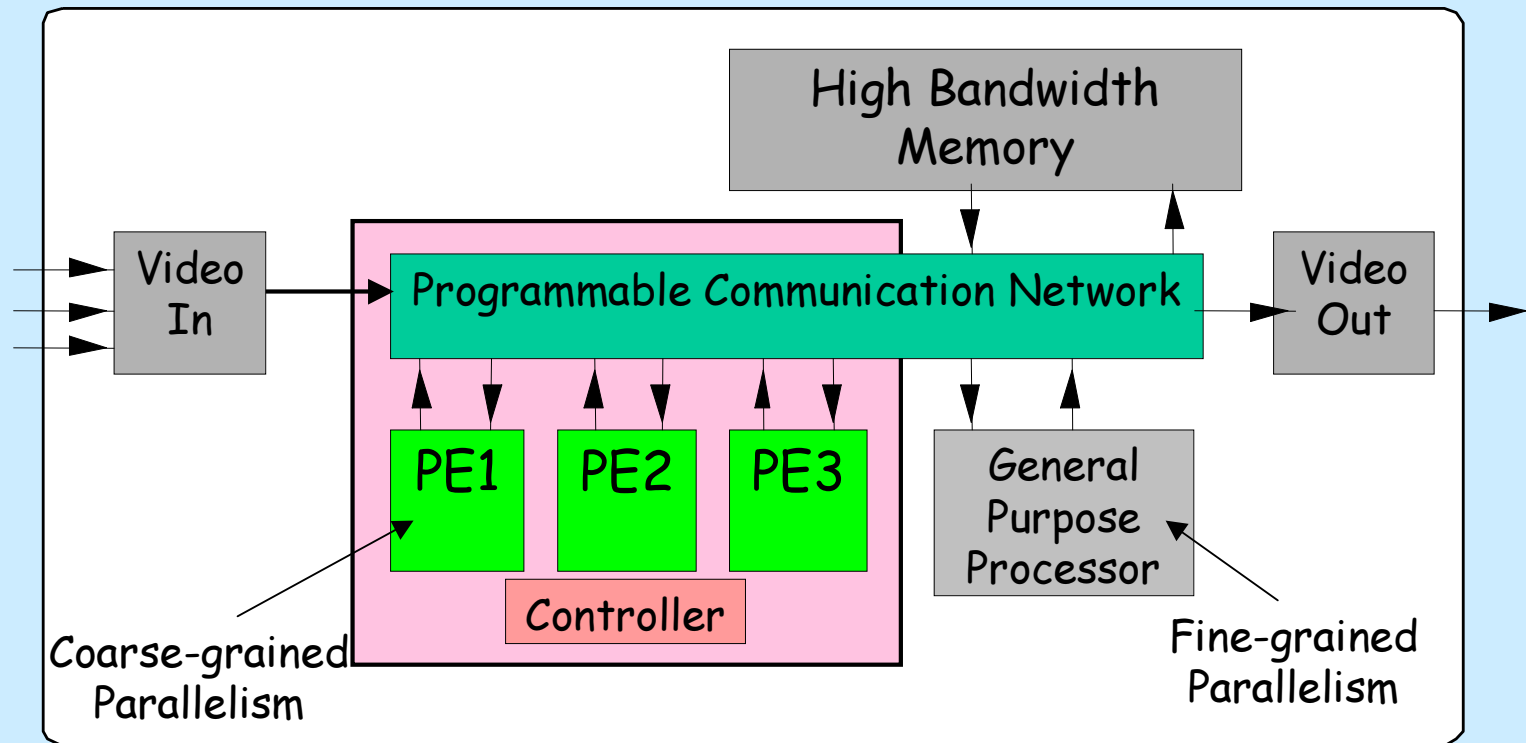
1: TU Delft, The Netherlands

2: UC Berkeley, USA

by Bart Kienhuis

At the Compiler and Architecture Support
for Embedded Systems Workshop,
Washington October 1-3 1999

Novel High-performance architectures



These architectures exploit parallelism

- Instruction level parallelism
- Coarse-grained parallelism

By courtesy
of Philips

What is missing

- Compiler that could extract the available parallelism in existing algorithms
 - Mobile Communication/Video/Radar
 - Imperative description language (C/Matlab)
 - Belong typical to the class of Nested Loop Programs
- A description in a format that can be mapped either onto
 - Microprocessor
 - Coprocessors

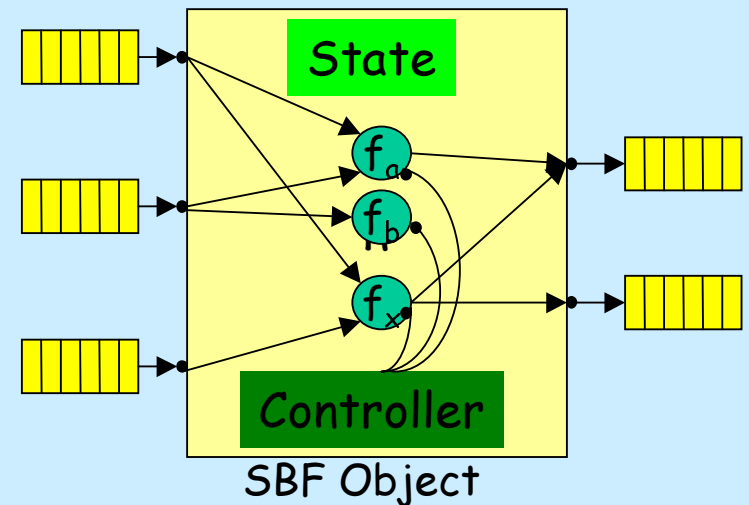
Process Network

- Particular Model of Computation (Kahn'74)
- Network of concurrent executing processes
- Communicate over unbounded FIFOs
- Each process executes a sequence of statements
- Performing some operation, a blocking read or a non-blocking write

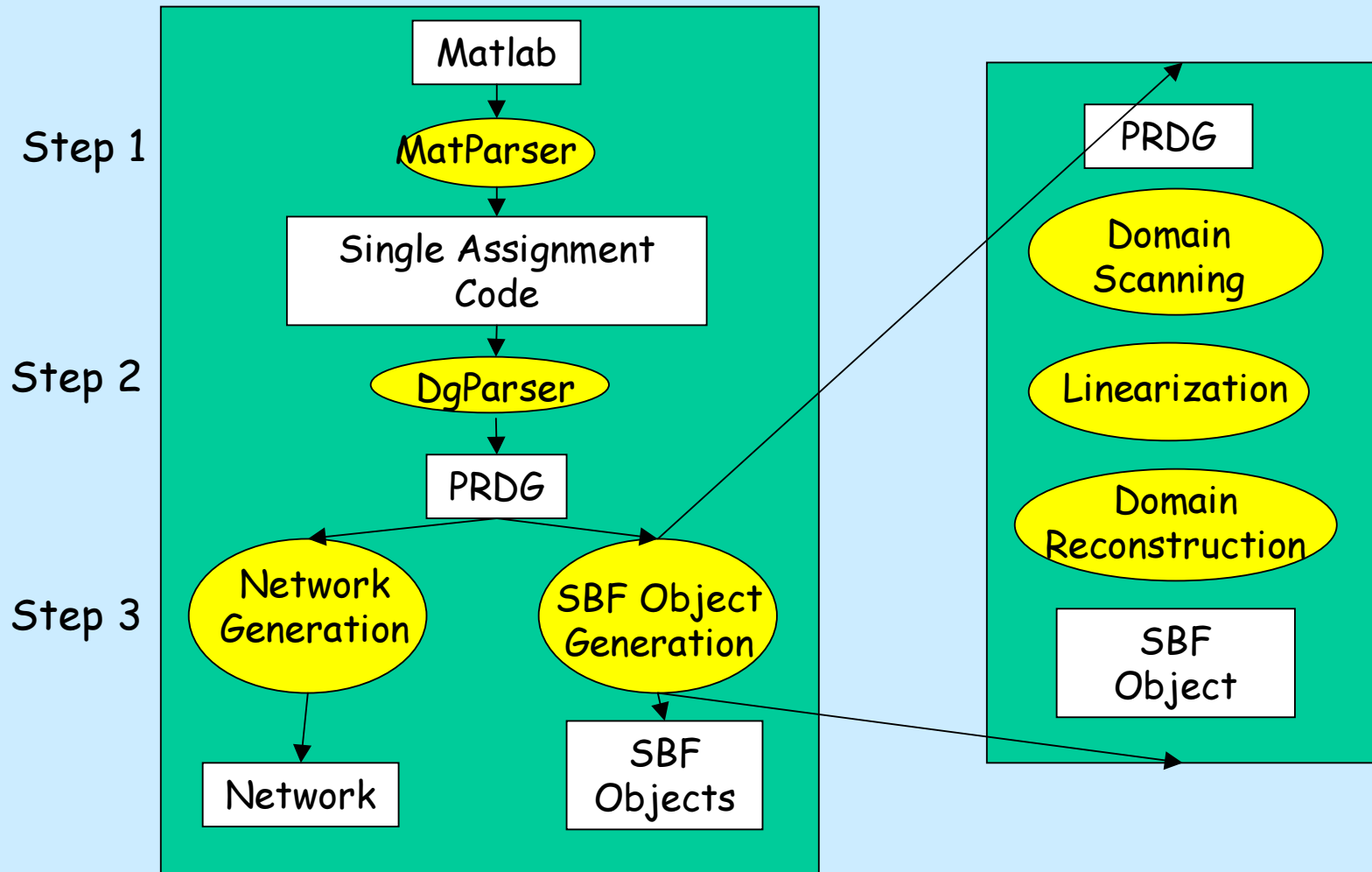
- Characteristics of Process Networks
 - Deterministic Execution
 - Able to express fine grained to coarse grained parallelism

Problem Statement

- Derive in an automated way a Process Network description from an algorithm described in an imperative language, in particular *Matlab*.
- Model the Processes according to the SBF Model of Computation
 - Controller
 - State
 - Set of Functions



Our Solution Approach



Step 1: MatParser

```

for i= 1 : 1 : N,
  for j= 1 : 1 : M,
    [a(i+j)] = funcA(a(i+j));
  end
end

```

```

for i=1 : 1 : N,
  for j=1 : 1 : M,

```

```

    if i-2 >= 0,
      if M-j-1 >= 0,
        [ in0 ] = ipd(a_1(i-1,j+1));
      else %% if -M+j >= 0;
        [ in0 ] = ipd(a(i+j));
      end

```

```

    end
    else %% if -i+1 >= 0;
      [ in0 ] = ipd(a(i+j));
    end

```

```

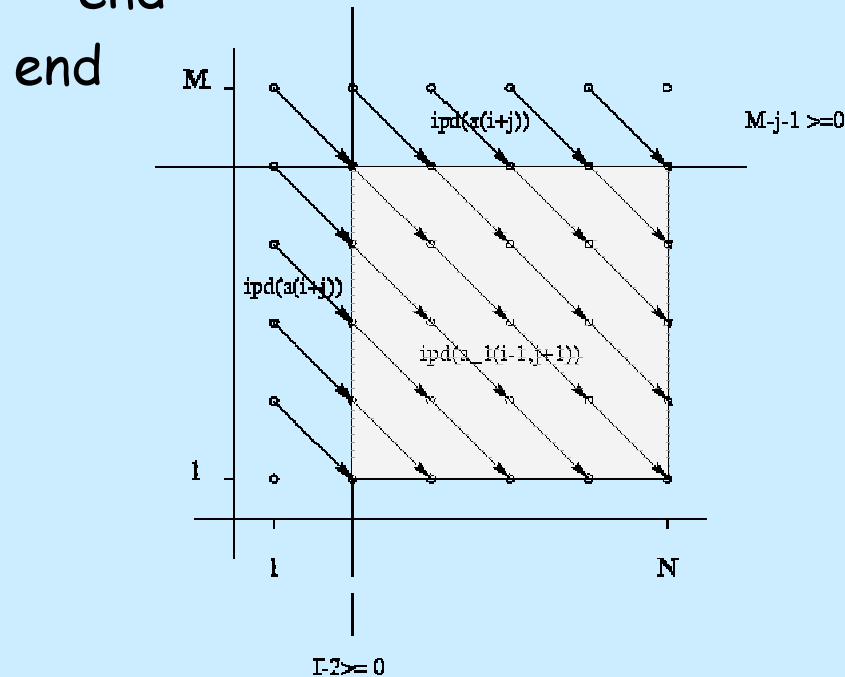
    [ out0 ] = funcA( in0 );
    [a_1( i ,j )] = opd(out0);

```

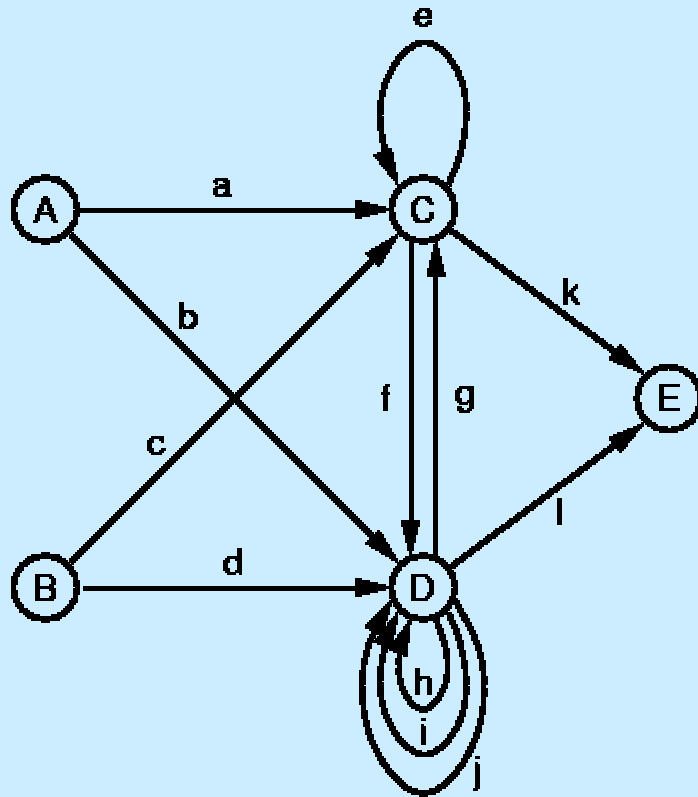
```

  end
end

```



Step 2: DgParser



$$G = \{V, E\}$$

$$V = \{v_1, v_2, \dots, v_{|V|}\}$$

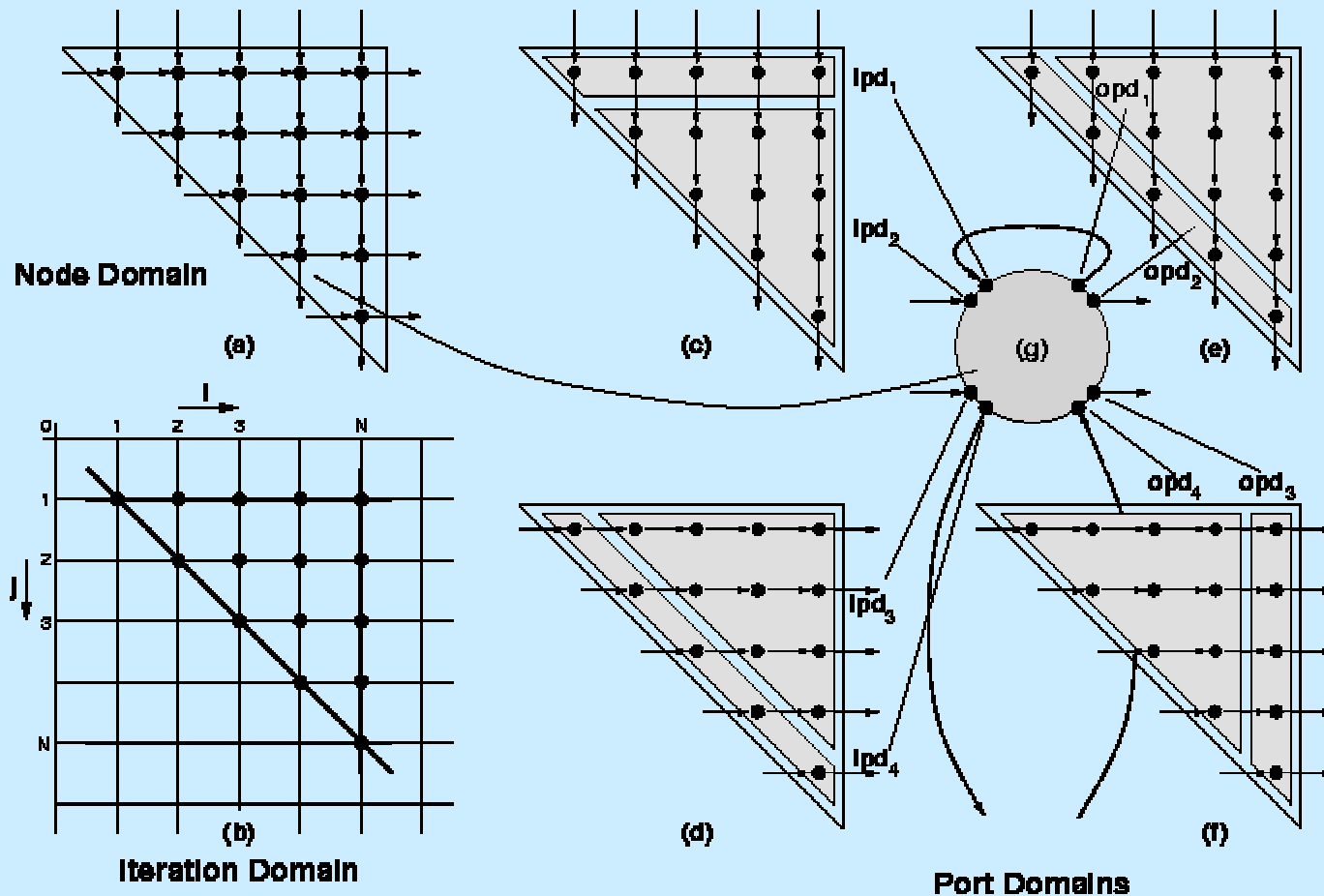
$$E = \{e_1, e_2, \dots, e_{|E|}\}$$

$$\mathbf{P} = \{\mathbf{x} \in \mathbf{Z}^n \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$$

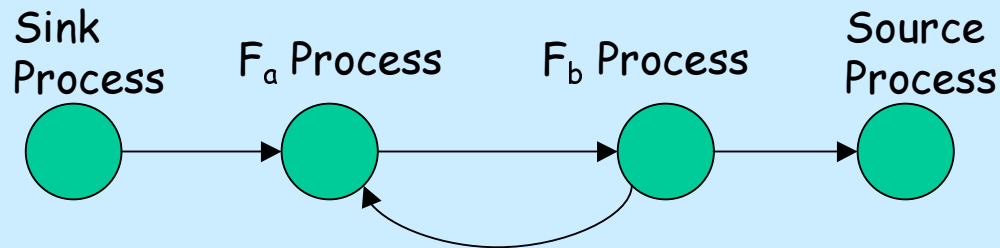
$$\mathbf{I} = \{\mathbf{I} \mid \mathbf{I} = \mathbf{L}\mathbf{x} + \mathbf{m} \wedge \mathbf{x} \in \mathbf{P}\}$$

Polyhedral Reduced Dependence Graph

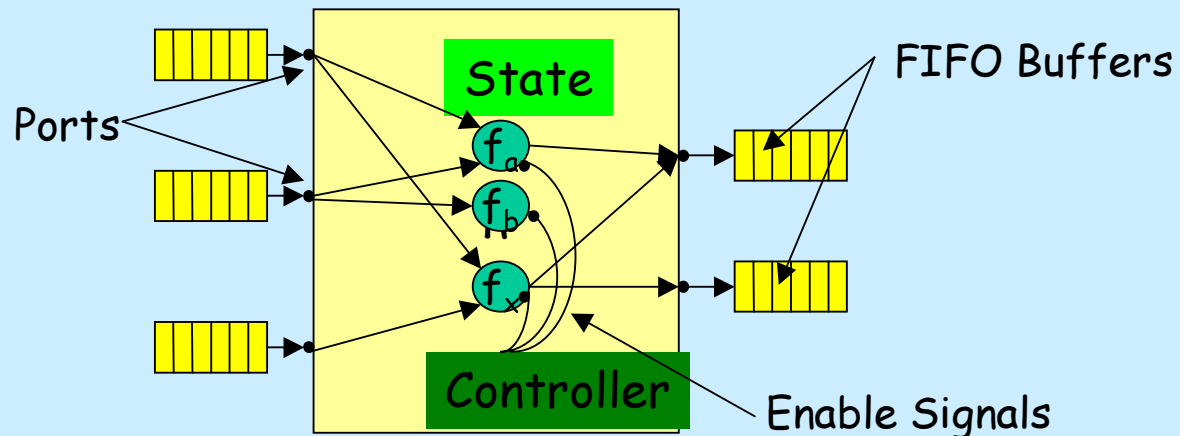
Example of Node and Port Domains



Step 3: SBF Object and Network Generation



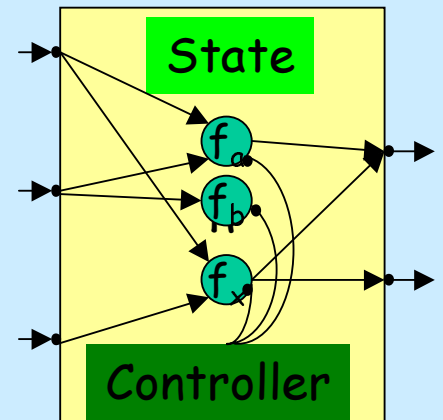
SBF Network (Parallel)



SBF Object (Sequential)

Elements of an SBF Object

- Function Repertoire $F = \{f_1, f_2, \dots, f_{|F|}\}$
- Binding Function $\mu: C \rightarrow F$
- Transition Function $\omega: C \rightarrow C$
- Sequence of invocations
 - Stream processing

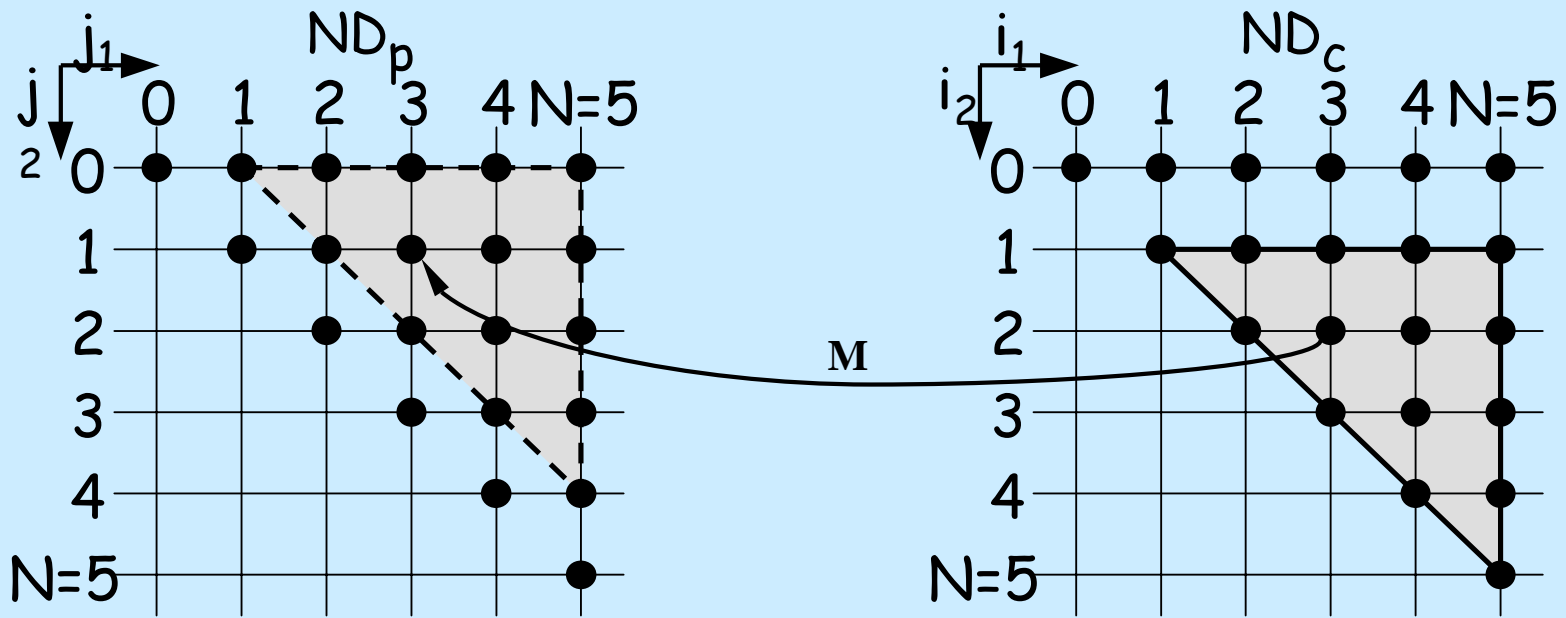
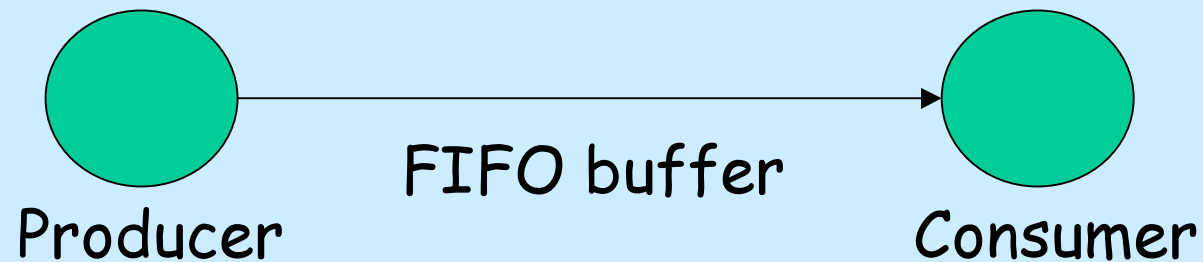


$$f_{init} \xrightarrow{\mu(\omega(c))} f_a \xrightarrow{\mu(\omega(c))} f_b \xrightarrow{\mu(\omega(c))} \dots f_x \xrightarrow{\mu(\omega(c))} \dots$$

Deriving SBF Objects

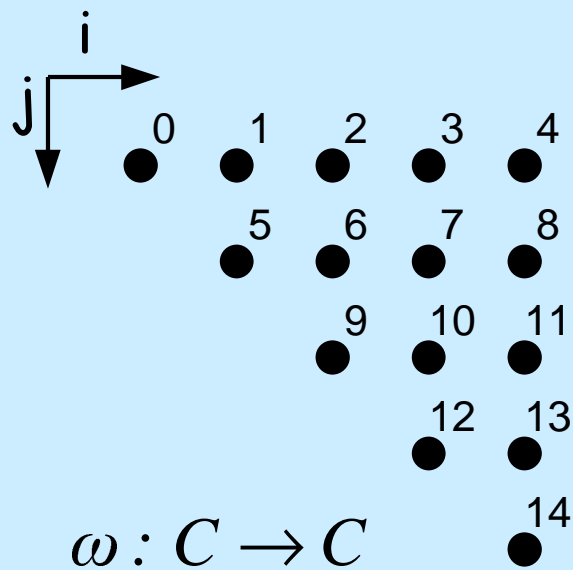
- Domain Scanning
 - Order the execution of functions in a particular order
- Linearization
 - Use 1-D communication structure (FIFO)
- Domain Reconstruction
 - A produced token needs to be consumed by the correct function

Running Example



Domain Scanning

Currently, we still follow the lexicographical ordering



for $i_1 = \dots,$

for $i_2 = \dots,$

Fourier-
Motzkin

sort order(i_1, i_2)

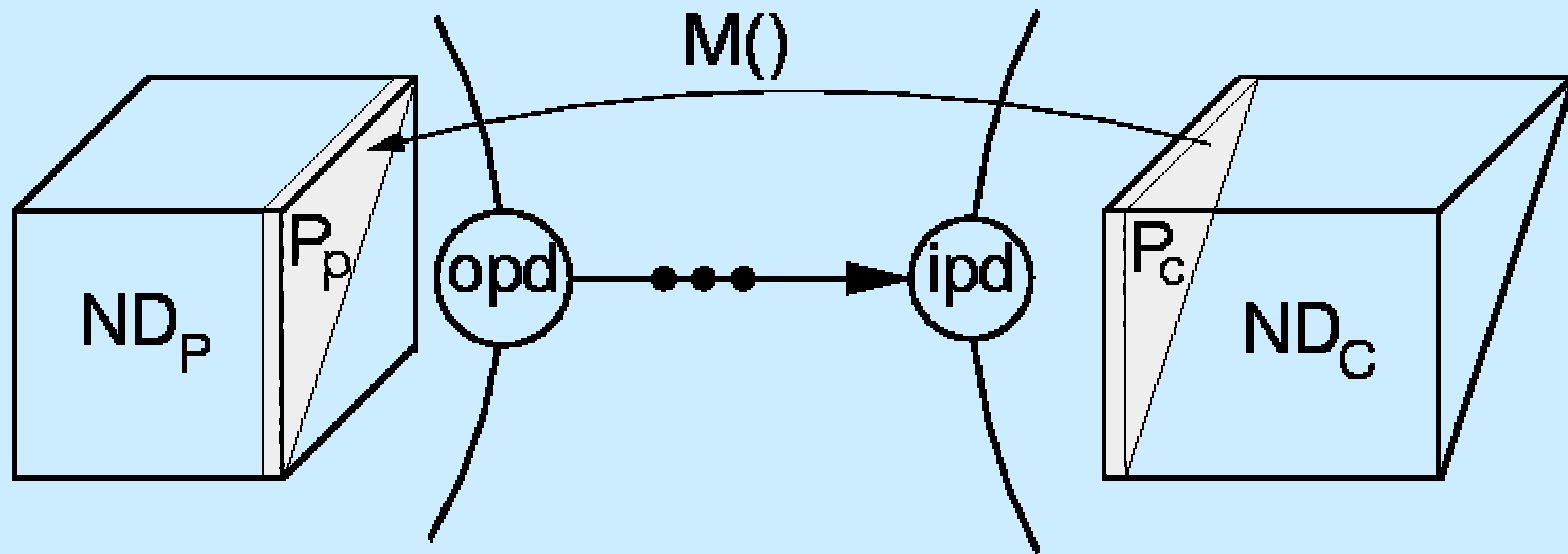
$$0 \leq i_1 \leq N$$

$$0 \leq i_2 \leq i_1$$

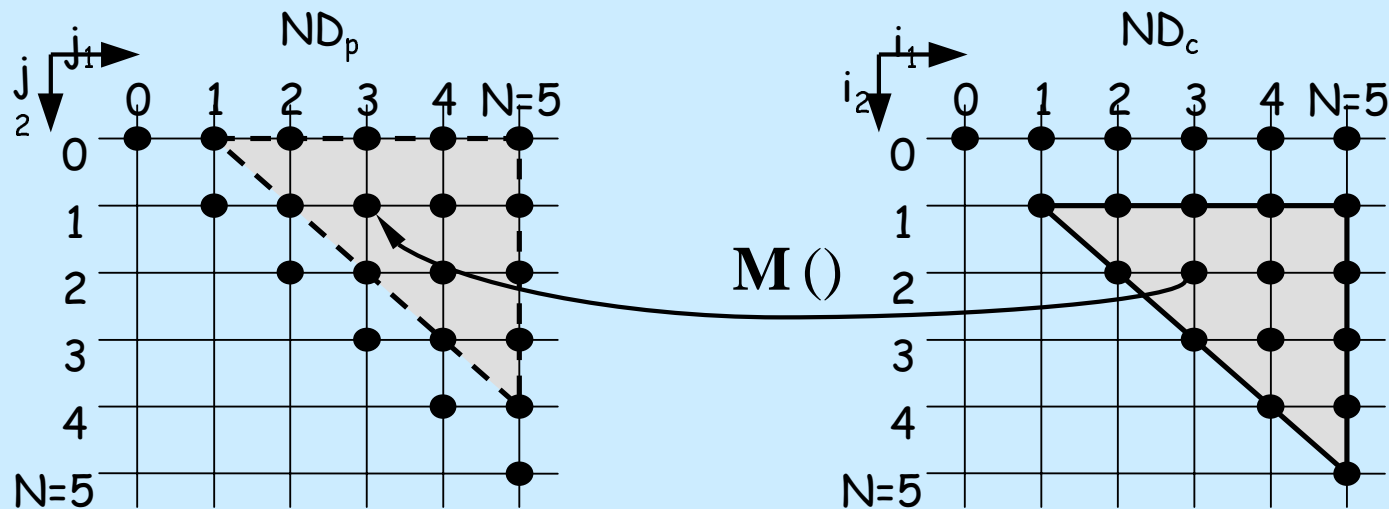
for $i_1 = 0 : 1 : N,$

for $i_2 = 0 : 1 : i_1,$

Domain Reconstruction



Domain Reconstruction



$$M(): \begin{bmatrix} j_1 \\ j_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} \Rightarrow \begin{aligned} j_1 &= i_1 \\ j_2 &= i_2 - 1 \end{aligned}$$

$$0 \leq j_2 \leq N-1$$

$$j_2 + 1 \leq j_1 \leq N$$

$$1 \leq i_2 \leq N$$

$$i_2 \leq i_1 \leq N$$

Domain Reconstruction

opd

ipd

- the port domains are given by

$$\begin{array}{ll} 0 \leq j_2 \leq N-1 & 1 \leq i_2 \leq N \\ j_2+1 \leq j_1 \leq N & i_2 \leq i_1 \leq N \end{array}$$

- since we scan the iteration domains of the node domains we already have that

$$\begin{array}{ll} 0 \leq j_2 \leq N & 0 \leq i_1 \leq N \\ j_2 \leq j_1 \leq N & 0 \leq i_2 \leq i_1 \end{array}$$

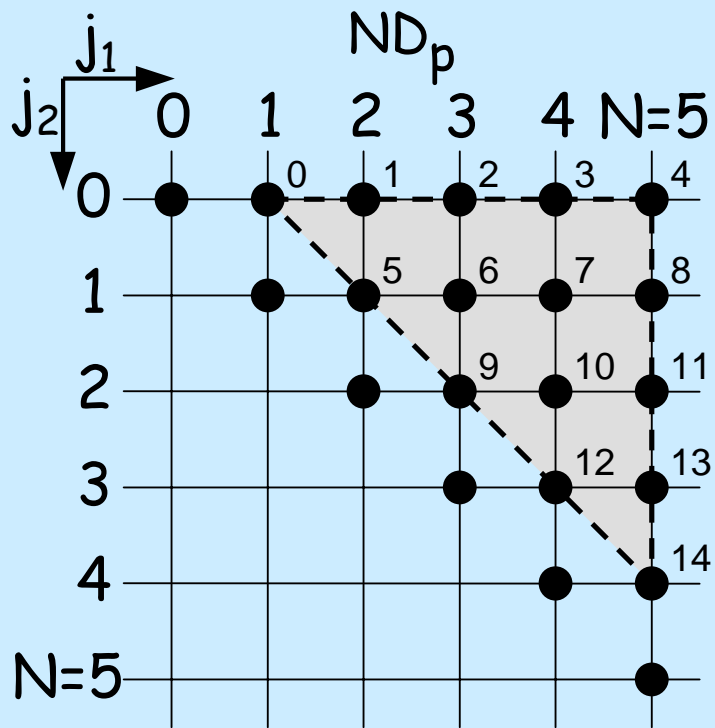
- thus, what remains to be checked at run-time

$$j_2+1 \leq j_1 \qquad 1 \leq i_2$$

Linearization

- one-dimensional memory m for every input port domain (ipd) / output port domain pair (opd).
- let t be the token produced in the opd at index point j , the address where t is stored in m is $w(j)$. We call $w()$ the **write polynomial**. We have $m(w(j)) = t$
- the **rank** of a point is the number of points that are lexicographical smaller than that point.
- $w()$ is constructed such that for the given scan order $w(j)$ is the rank of the point at index j .
- at index point i in the ipd we read a token from $m(r(i))$. $r(i)$ is the **read polynomial** and is the substitution of $j = w(i)$ in the write polynomial.

Linearization

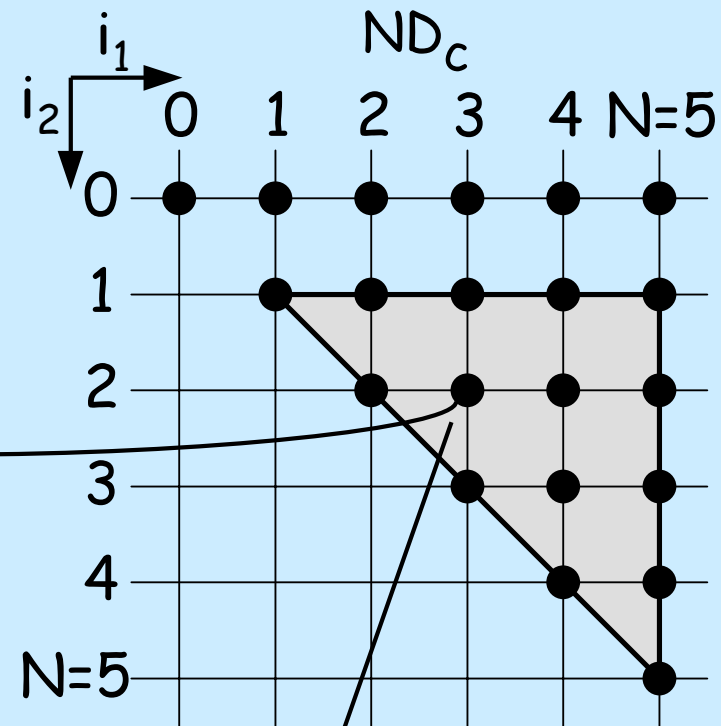
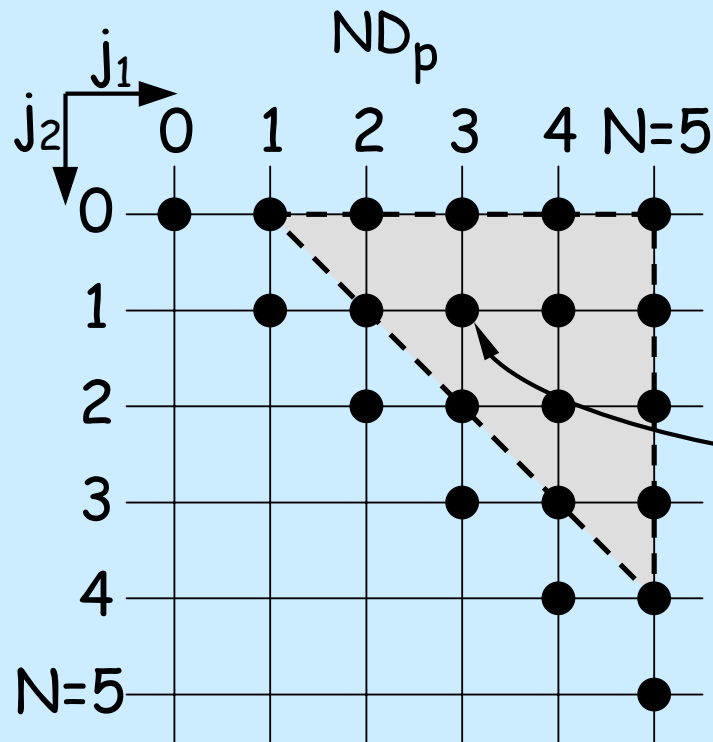


- the array position where a token is stored is equal to the rank of the node that produces the token.
- the function that ranks the nodes in the output port domain is called the write polynomial.
- ranking is a counting problem

write polynomial:

$$w(j_1, j_2) = -\frac{1}{2} j_2^2 + (N - \frac{1}{2}) j_2 + j_1 - 1$$

Linearization



M

read polynomial:

$$r(i_1, i_2) = w(M(i_1, i_2)) = w(i_1, i_2 - 1)$$

$$r(i_1, i_2) = -\frac{1}{2}i_2^2 + (N + \frac{1}{2})i_2 + i_1 - (N + 1)$$

from where must this node read a token?

Putting it all together

ND_p

```
for j2 = 0 to N
  for j1 = j2 to N

    [out] = f(...);

    if (j2+1 ≤ j1)
      m(w++) = out;
    end

  end

end
```

ND_c

```
for i1 = 0 to N
  for i2 = 0 to i1
    if (1 ≤ i2)
      in = m(r(i1, i2));
    end

    ... = f(in);

  end

end
```

In the SBF object, the for-next loop is replaced by a FSM

Conclusions

- Showed a 3-step approach to compile a Matlab algorithm into a Process Network description.
 - Polyhedral Reduced Dependence Graph
 - SBF Model of Computation
 - Elaborated on the third step, the derivation of an SBF Object.
- Showed that Process Network descriptions are of interest for new embedded architectures that exploit coarse- and fine-grained parallelism

Future Work

- The code (Java) works for simple examples.
 - MatParser
 - Panda
 - SBFsim
- Code needs to be made more generic
- Ptolemy Interface to the PN-domain to replace SBFsim

[Http://www.gigascale.org/compaan](http://www.gigascale.org/compaan)