Efficient Handling of Arrays in Dataflow Process Networks

Daniel Baudisch, Jens Brandt, Klaus Schneider

Embedded Systems Group
Department of Computer Science
University of Kaiserslautern, Germany

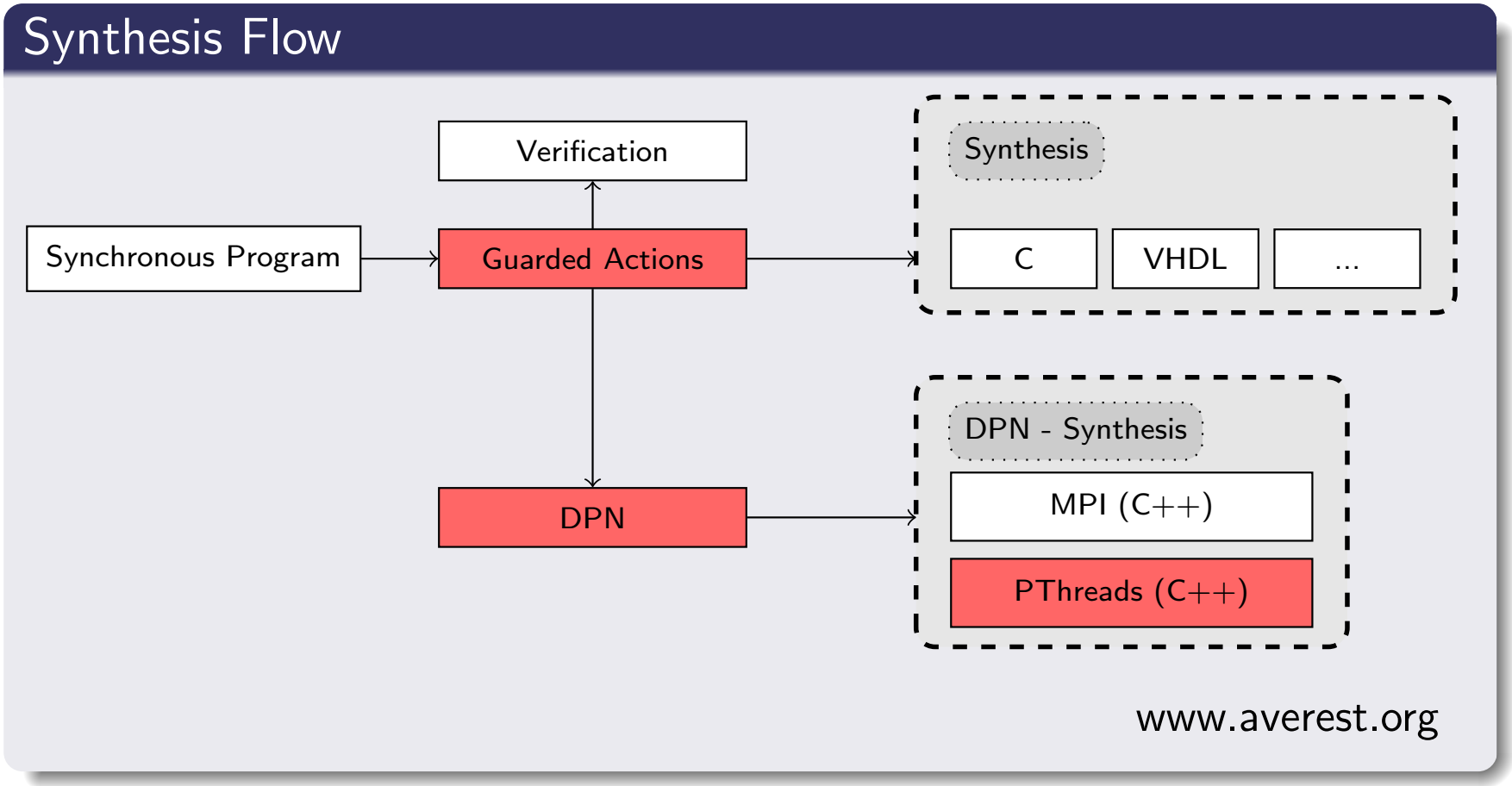
Outline

Outline

Motivation

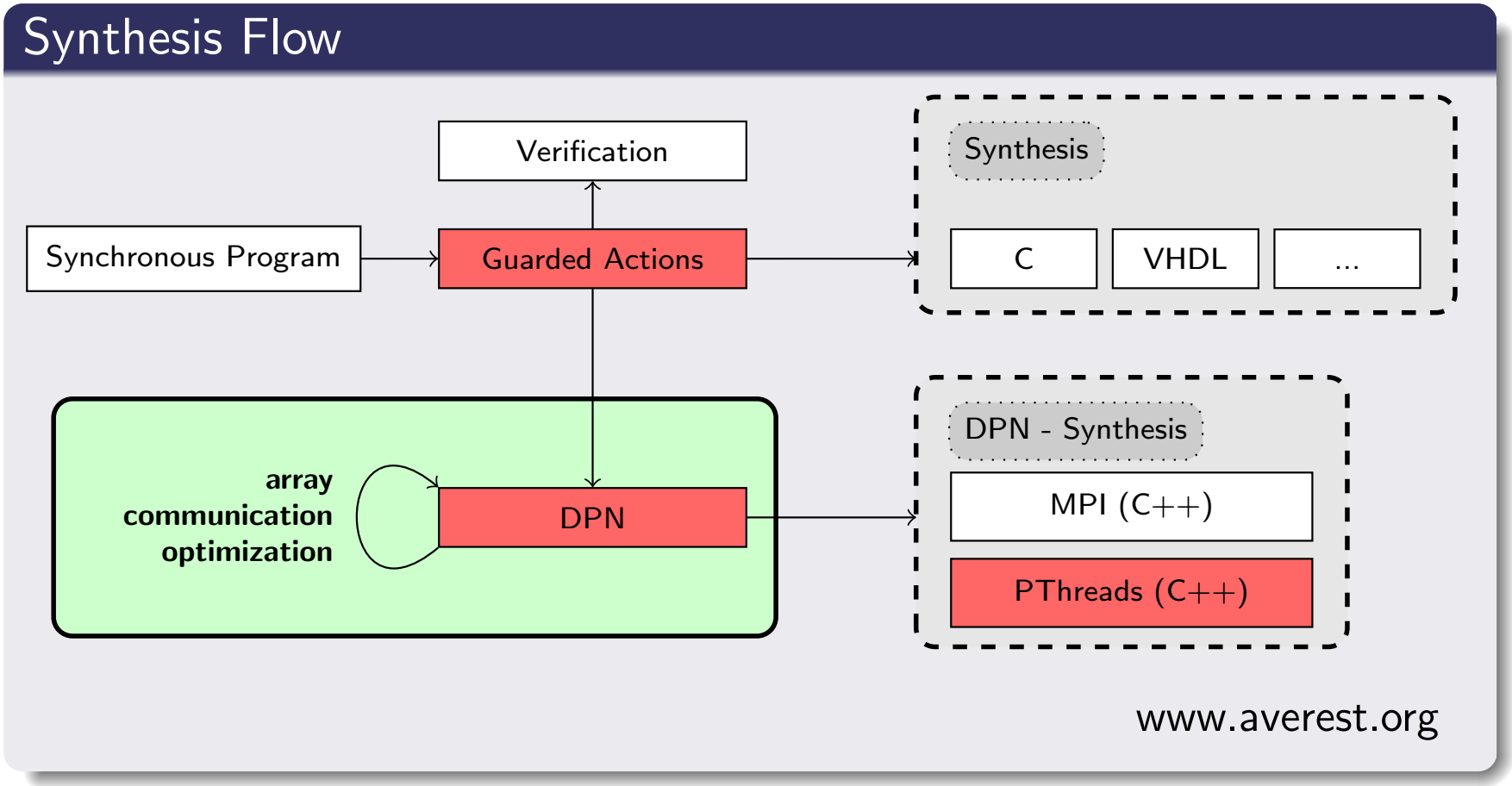
- model-based development of applications
  in particular: embedded systems
- synchronous languages, e. g. Esterel, Lustre, Quartz
  - can be used for embedded systems
  - hide communication latencies
  - execution of instructions in perfect synchrony
- synthesis more difficult
  (especially for heterogenous/distributed systems)

Synthesis Flow



- here: from DPN to DPN (array communication optimization)

Synthesis Flow



- here: from DPN to DPN (array communication optimization)

Guarded Actions (GA)

System (Example)

Interface:

Inputs: i, d, c

Output: o

Locals: $a[N]$, $j0$, j, p

Guarded Actions:

$p \Rightarrow \text{next}(p) = \text{True}$

$c \wedge p \Rightarrow a[j] = i$

$p \Rightarrow \text{next}(j) = (j+1)\%N$

$p \Rightarrow j0 = (j-d)\%N$

$p \Rightarrow o = a[j0]$

Guarded Actions (GA)

System (Example)

Interface:

Inputs: i, d, c

Output: o

Locals: $a[N]$, $j0$, j, p

Guarded Actions:

$p \Rightarrow \text{next}(p) = \text{True}$

$c \wedge p \Rightarrow a[j] = i$

$p \Rightarrow \text{next}(j) = (j+1)\%N$

$p \Rightarrow j0 = (j-d)\%N$

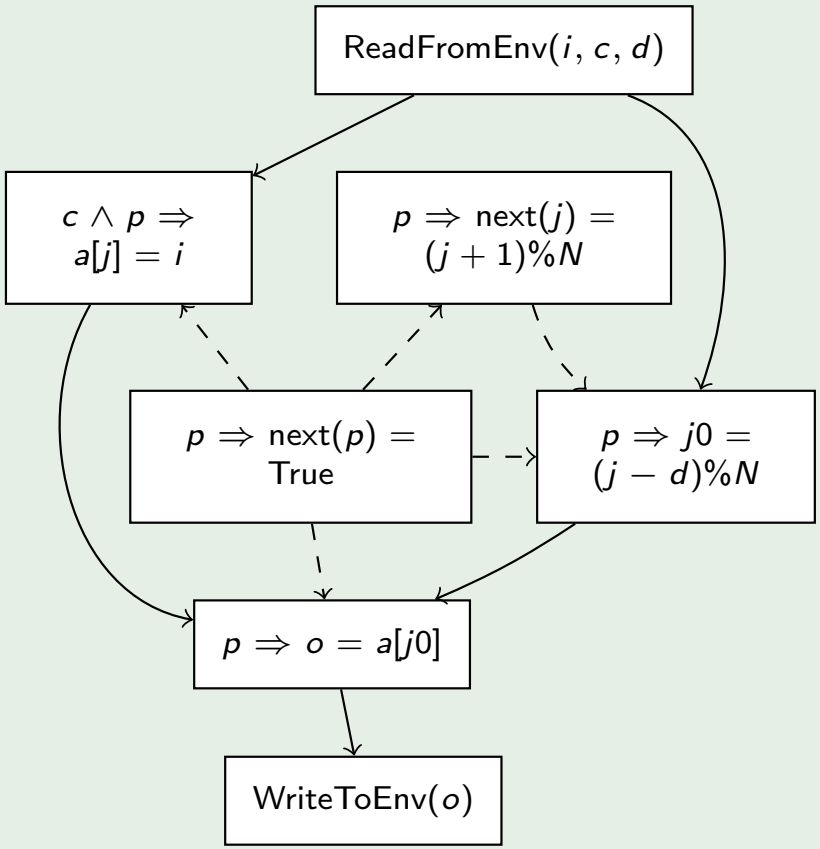
$p \Rightarrow o = a[j0]$

Generic execution

```
while(True)
    read system inputs
    execute GA in data-flow order
        update state of system
        generate outputs of system
    write system outputs
endwhile
```

Guarded Actions (GA)

Dependency Graph

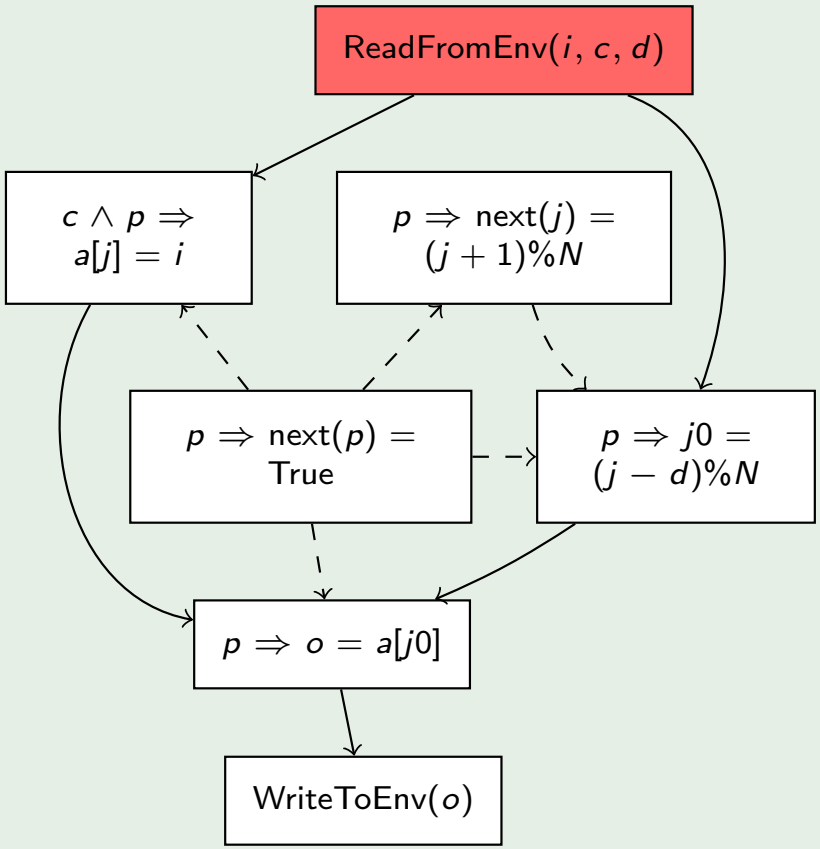


Generic execution

```
while(True)
    read system inputs
    execute GA in data-flow order
        update state of system
        generate outputs of system
    write system outputs
endwhile
```

Guarded Actions (GA)

Dependency Graph

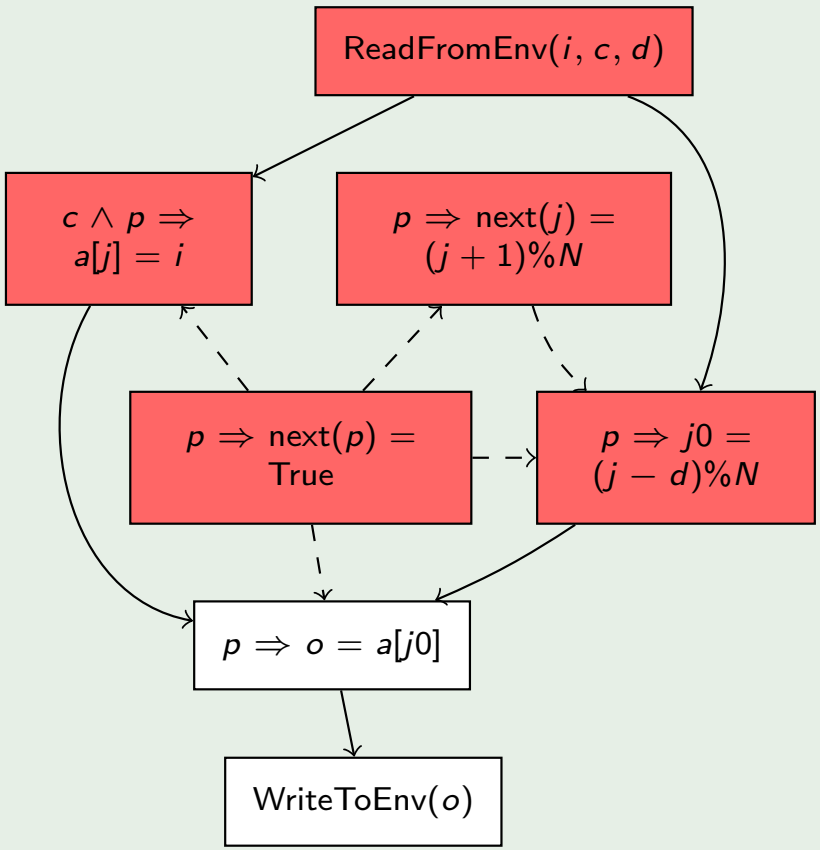


Generic execution

```
while(True)
    read system inputs
    execute GA in data-flow order
        update state of system
        generate outputs of system
    write system outputs
endwhile
```

Guarded Actions (GA)

Dependency Graph

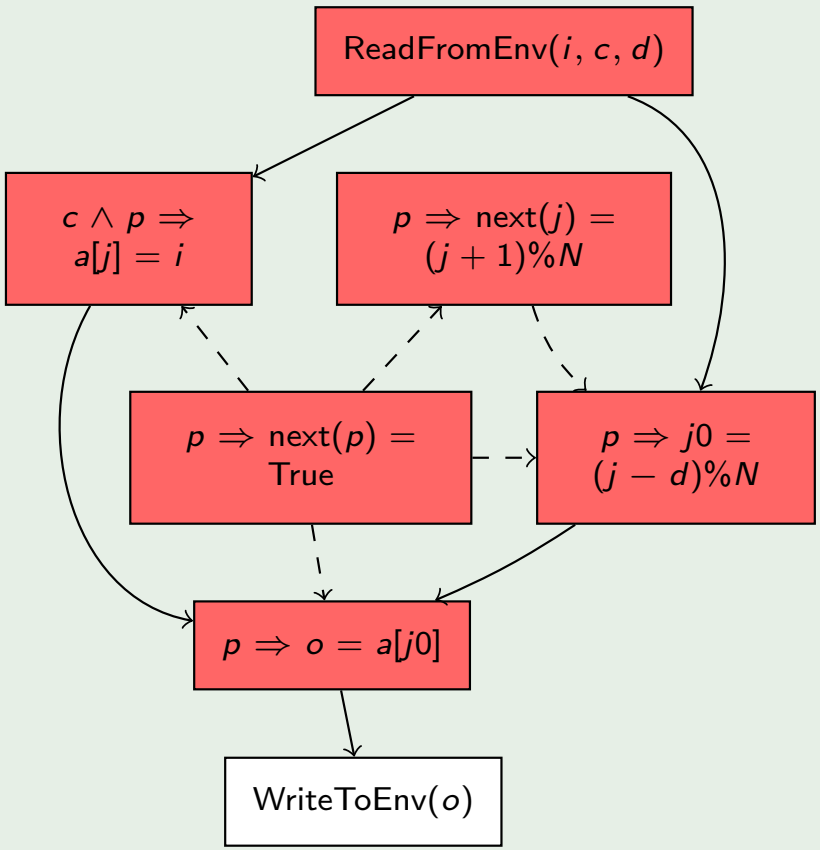


Generic execution

```
while(True)
    read system inputs
    execute GA in data-flow order
        update state of system
        generate outputs of system
    write system outputs
endwhile
```

Guarded Actions (GA)

Dependency Graph

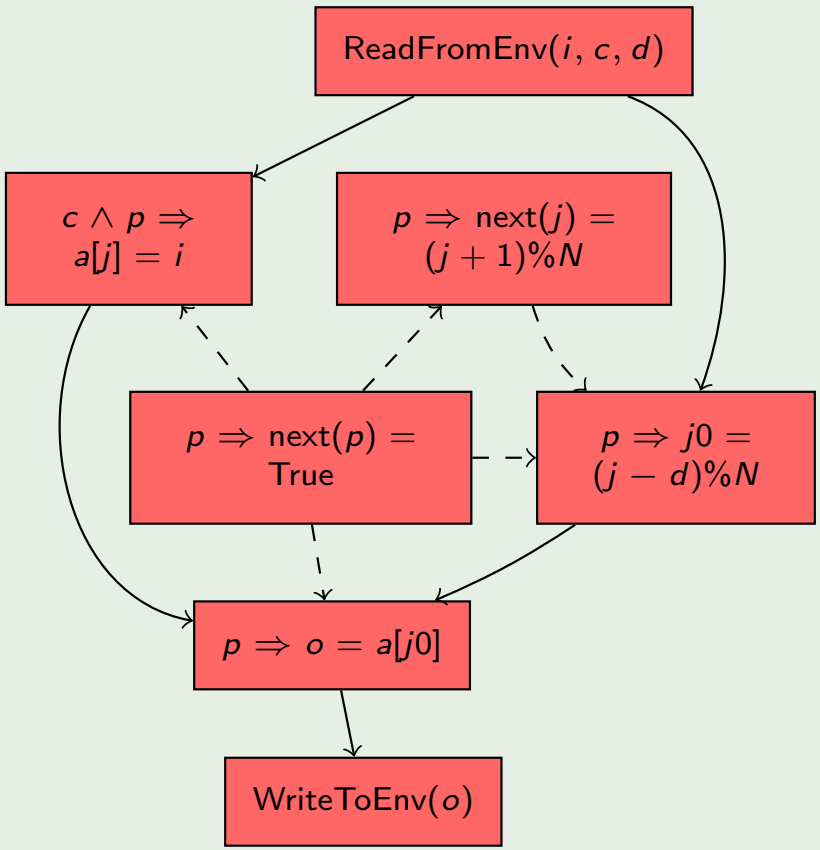


Generic execution

```
while(True)
    read system inputs
    execute GA in data-flow order
        update state of system
        generate outputs of system
    write system outputs
endwhile
```

Guarded Actions (GA)

Dependency Graph

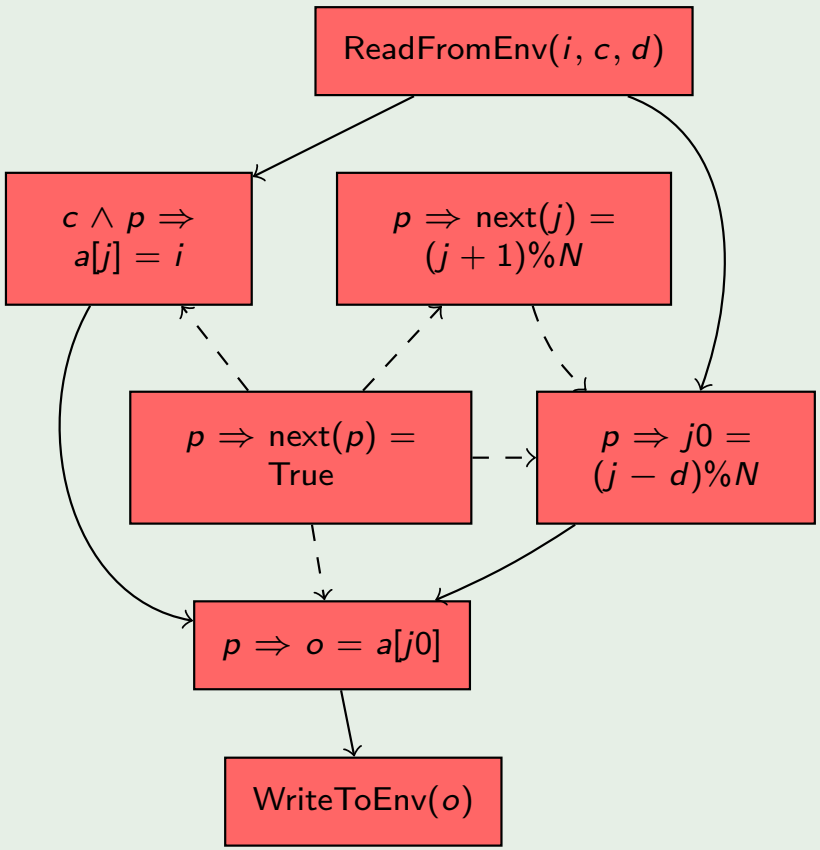


Generic execution

```
while(True)
    read system inputs
    execute GA in data-flow order
        update state of system
        generate outputs of system
    write system outputs
endwhile
```

Guarded Actions (GA)

DPN

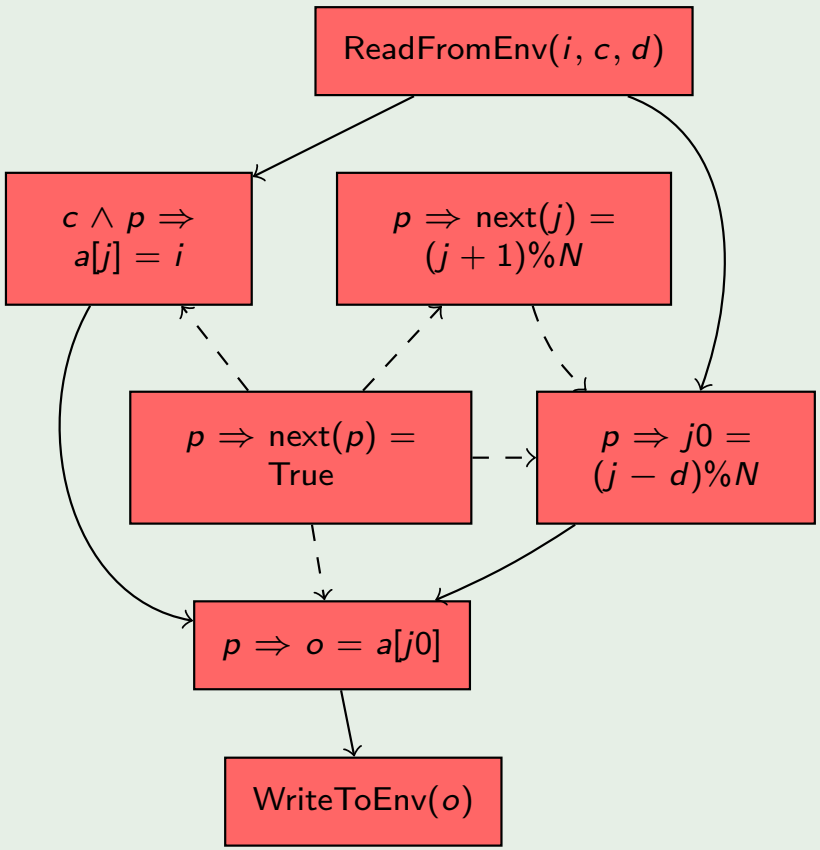


Notes

- DPN is gained by replacing dependencies by FIFO-buffers ⇒ "Theory of Latency Insenstive Design" (McMillan et. al)
- writer must be uniquely determined: group nodes writing to the same variable
- grouping of nodes / partitioning increase computation effort per node

Guarded Actions (GA)

DPN



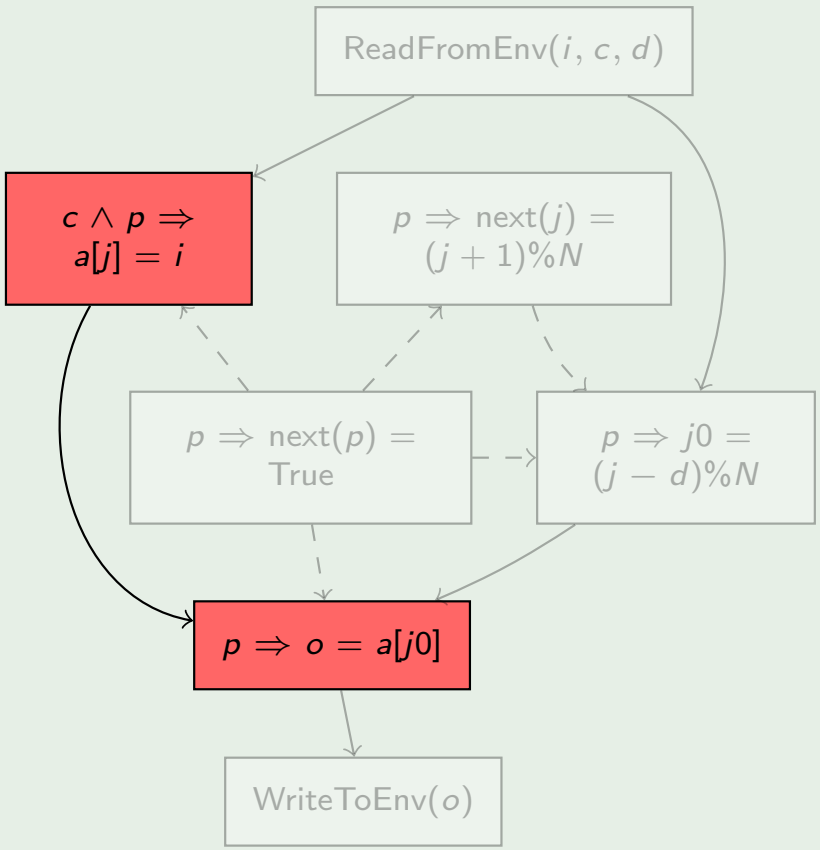
Synthesis (PThreads)

- create thread for each node
- each thread:

```
while(True)
    read input buffers
    execute GA of node
    write output buffers
endwhile
```

Guarded Actions (GA)

DPN



Issue

- transport of arrays expensive
- here:
  only few elements are written in each step
  (recognized by variable array indicies)
  ⇒ only changes should be commited

Outline

Related Work

- Gupta et. al.: analysis of structured programs with data parallelism
  CFG to generate 'Section Communication Descriptors' and 'Availability Section Descriptors'
- Arvind: demand-driven optimization
  Compute data only if it is required. Reduces computations, keeps communication.
- CAL (OpenDF): arrays only as local states
  Communication is left to programmer.
- *more related work in paper*

Basic Idea

- create local copy a^R of array a in each node R reading a
- determine tuple(s) containing update information in writer:
  - guard $valid_a$ (condition whether update has to be done)
  - index idx_a (address of element that is overwritten)
  - value rhs_a
- communicate update information instead of modified array to readers
- *detailed pseudo-algorithm in the paper*

Example

Origin

$$c \wedge p \Rightarrow a[j] = i$$

$$p \Rightarrow o = a[j0]$$

Modified

$$c \wedge p \Rightarrow valid_a = true$$
$$valid_a \Rightarrow idx_a = j$$
$$valid_a \Rightarrow rhs_a = i$$
$$valid_a \Rightarrow a[idx_a] = rhs_a$$

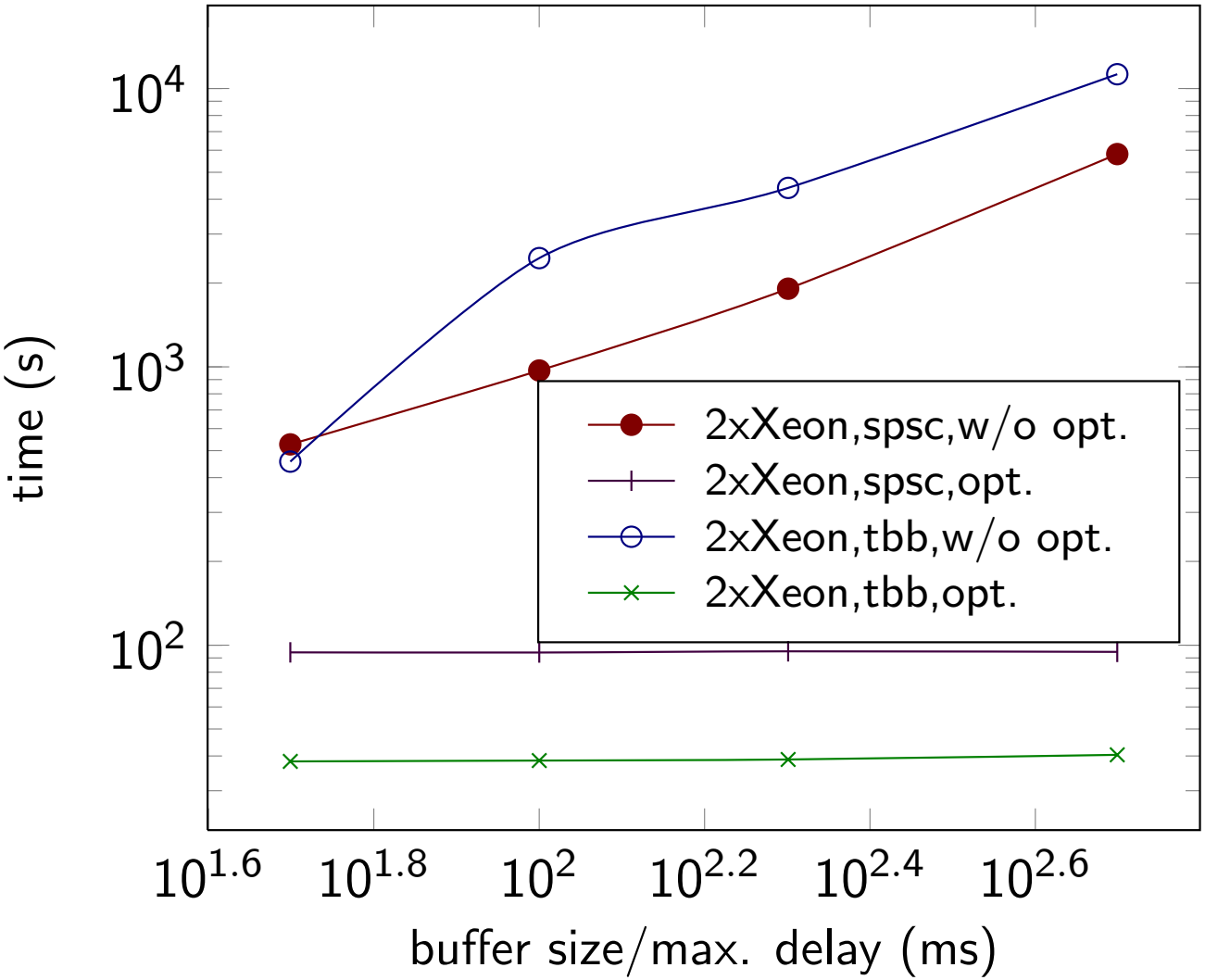
$$p \Rightarrow o = a^R[j0]$$
$$valid_a \Rightarrow a^R[idx_a] = rhs_a$$

Some Details

- take care of timing (immediate / delayed)
- index tuple for multi-dimensional array access
  $\Rightarrow$ neglect constants
- when to use communication reduction
  (size of update) * (number of potential update) $<$ size of array
- *re-use update tuple* for assignments with same index
- *writer of memory cell is uniquely determined*
  allows to parallelize operations on multi-dimensional arrays

Outline

Benchmarks - Results



Benchmarks - Results

Benchmark systems: 2x Xeon X5450 (= 8x 3.00GHz)
i5-750 (4x 2.66GHz)

Benchmark Name	Speedup on i5-750		Speedup on 2xXeon	
	custom queue	Intel TBB queue	custom queue	Intel TBB queue
Delay (max. 50ms)	53.29	19.02	5.59	11.94
Delay (max. 100ms)	98.19	39.90	10.31	63.84
Delay (max. 200ms)	189.43	66.30	20.04	113.10
Delay (max. 500ms)	343.85	208.37	61.44	278.69
MatrixMult (16x16)	1.14	1.35	1.18	1.33
MatrixMult (32x32)	1.88	1.47	1.30	3.03
MatrixMult (48x48)	1.56	1.53	1.33	1.56
Pitchshift	809.00	289.00	145.64	340.08
Landscape Rend. (H=400)	8.35	3.96	4.20	5.31
Landscape Rend. (H=800)	34.73	4.66	7.38	9.81
Landscape Rend. (H=1600)	36.41	20.26	17.97	20.05

Outline

Conclusion + Future Work

- synchronous MoC hides communication latencies
  $\Rightarrow$ nice for parallel programming
- in practice: communication costs time
- done: reduction of communication by sending update of single array elements instead of complete array
  - never slower
  - may be faster
- further issues:
  - generic reduction of communication
    $\Rightarrow$ endo- and isochronous systems to reduce overhead
    big idea: consider when a variable is changed and when it is needed

The End

Thank you for your attention!

Questions? Suggestions? Ideas?