Compiler Technology for CGRAs

Bjorn De Sutter

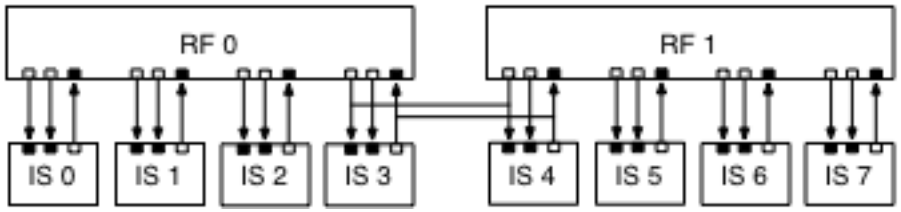
Ghent University

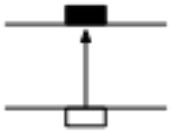
Overview

- What makes compilation for CGRAs different?
- What CGRA code generation techniques exist?
- What is not automated?

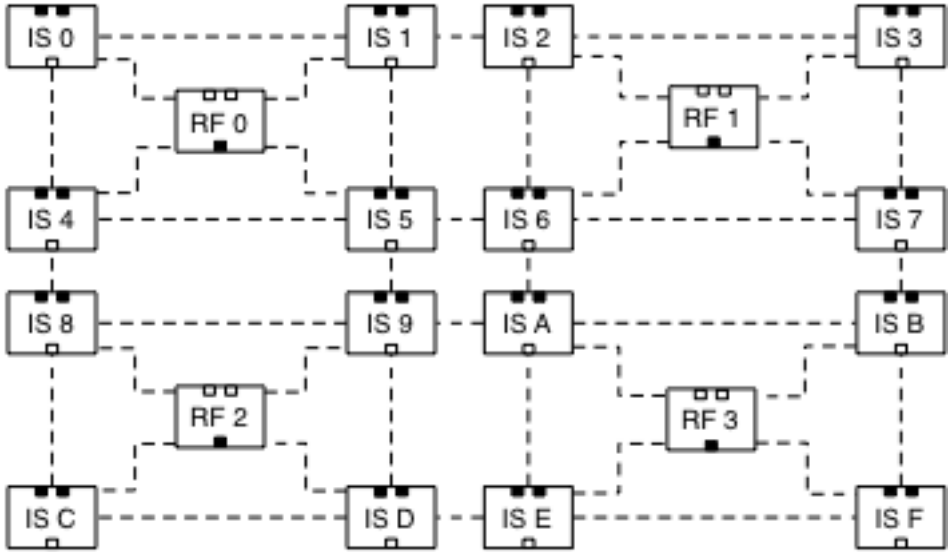
CGRA vs VLIW

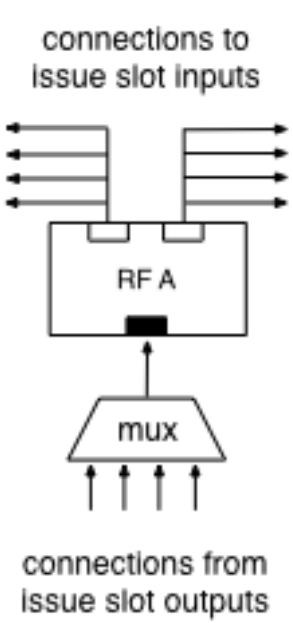
VLIW

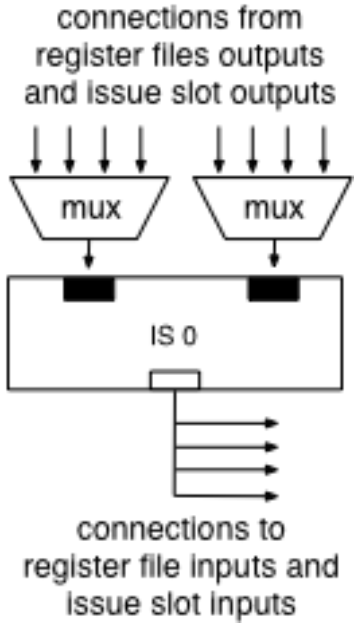




CGRA

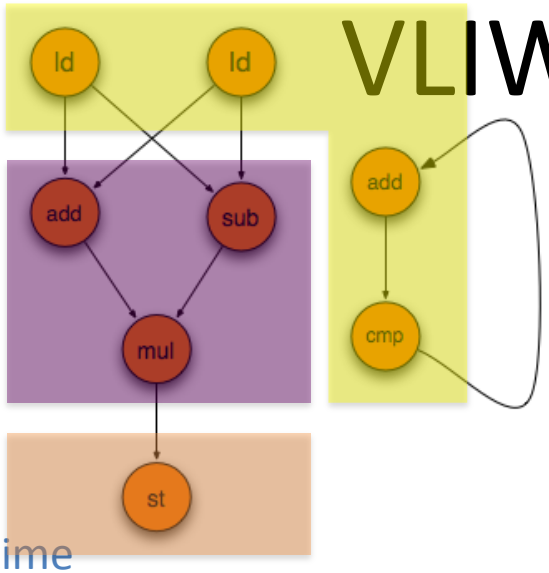


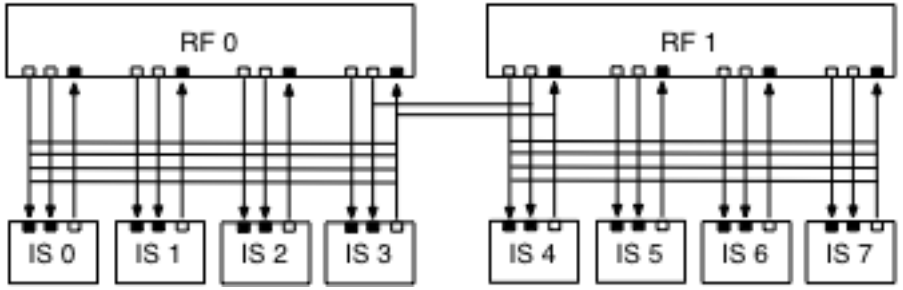




UNIVERSITEIT GENT

VLIW code scheduling





time

cycle 0

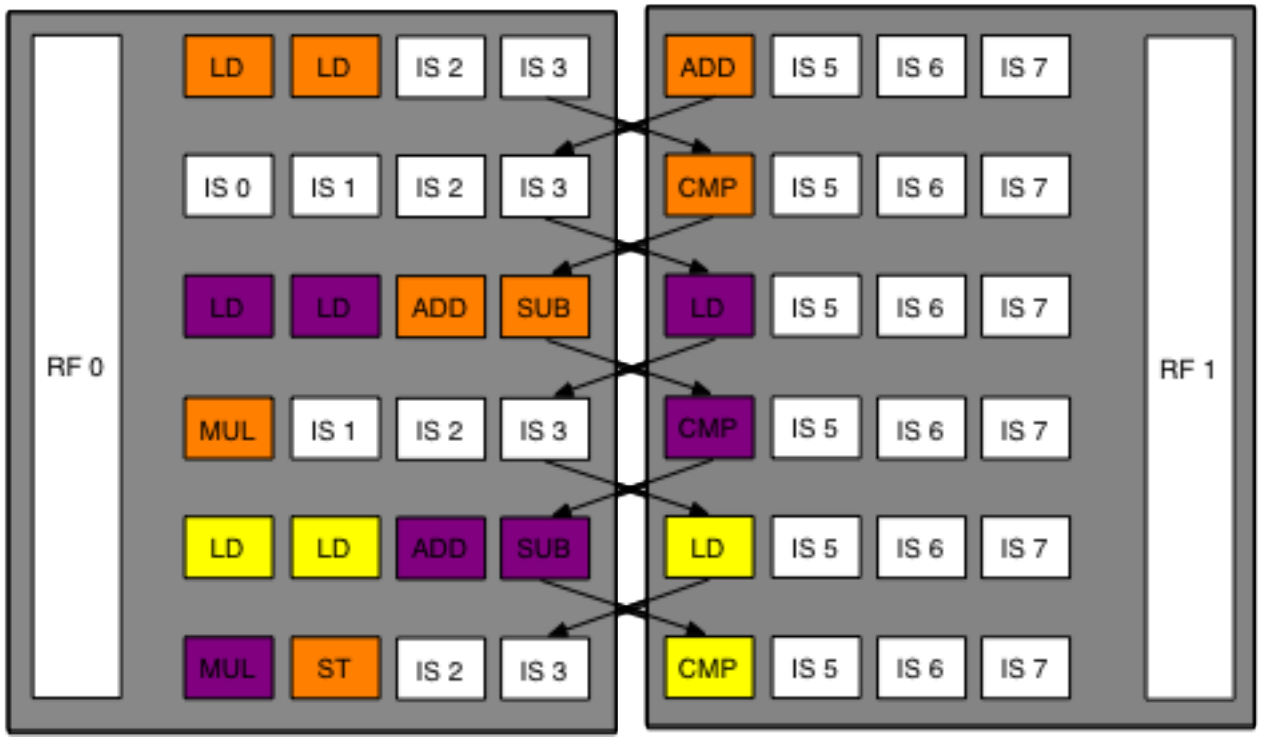
cycle 1

cycle 2

cycle 3

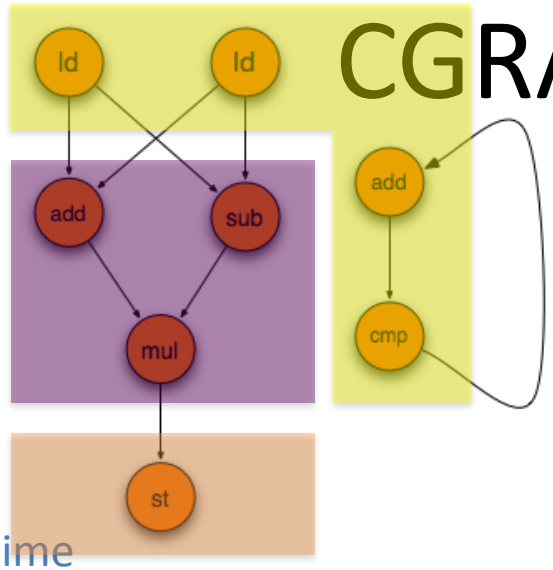
cycle 4

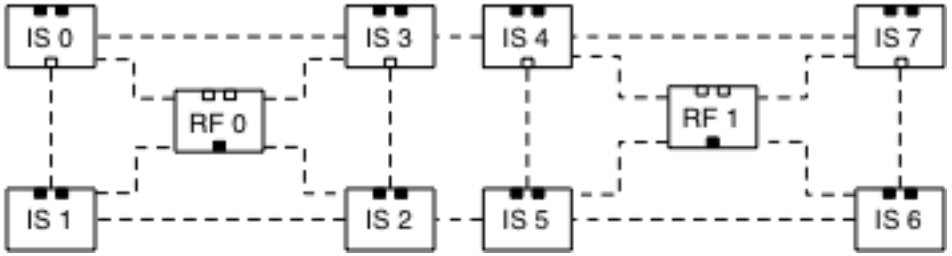
cycle 5



UNIVERSITEIT GENT

CGRA code scheduling





time

cycle 0

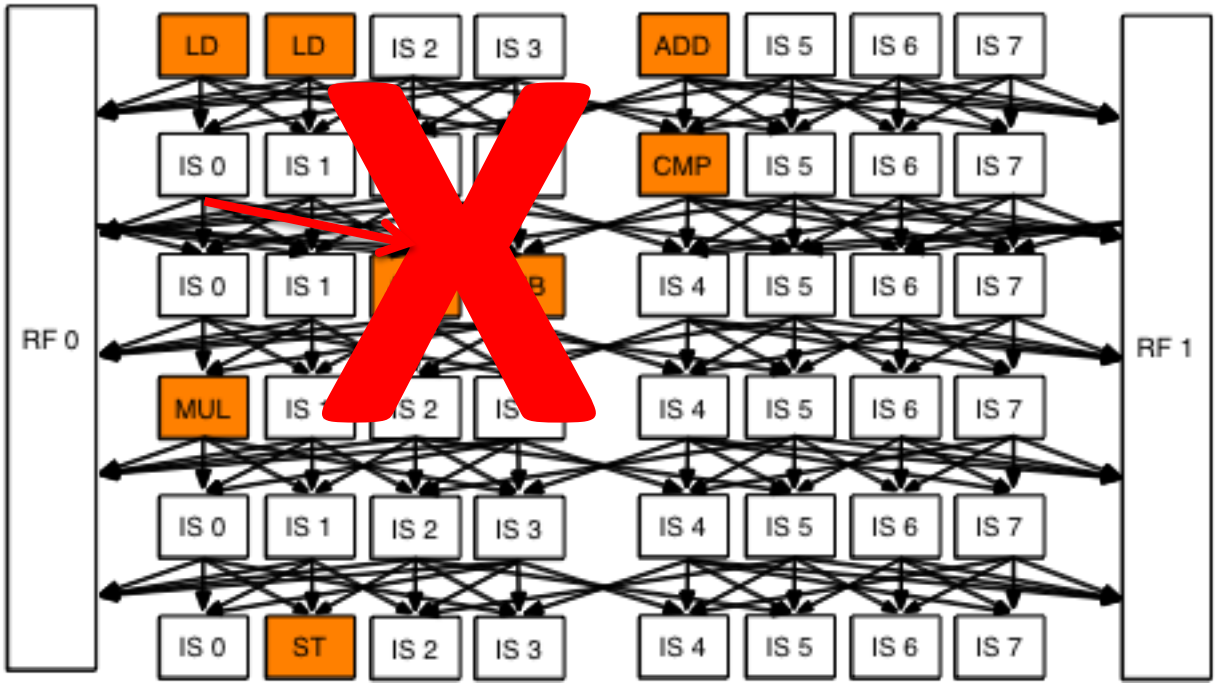
cycle 1

cycle 2

cycle 3

cycle 4

cycle 5



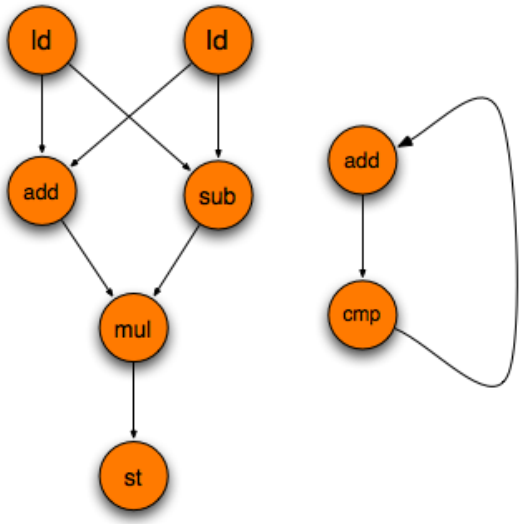
Overview

- What makes compilation for CGRAs different
- What CGRA code generation techniques exist?
- What is not automated?

CGRA scheduling = graph mapping

data dependence graph

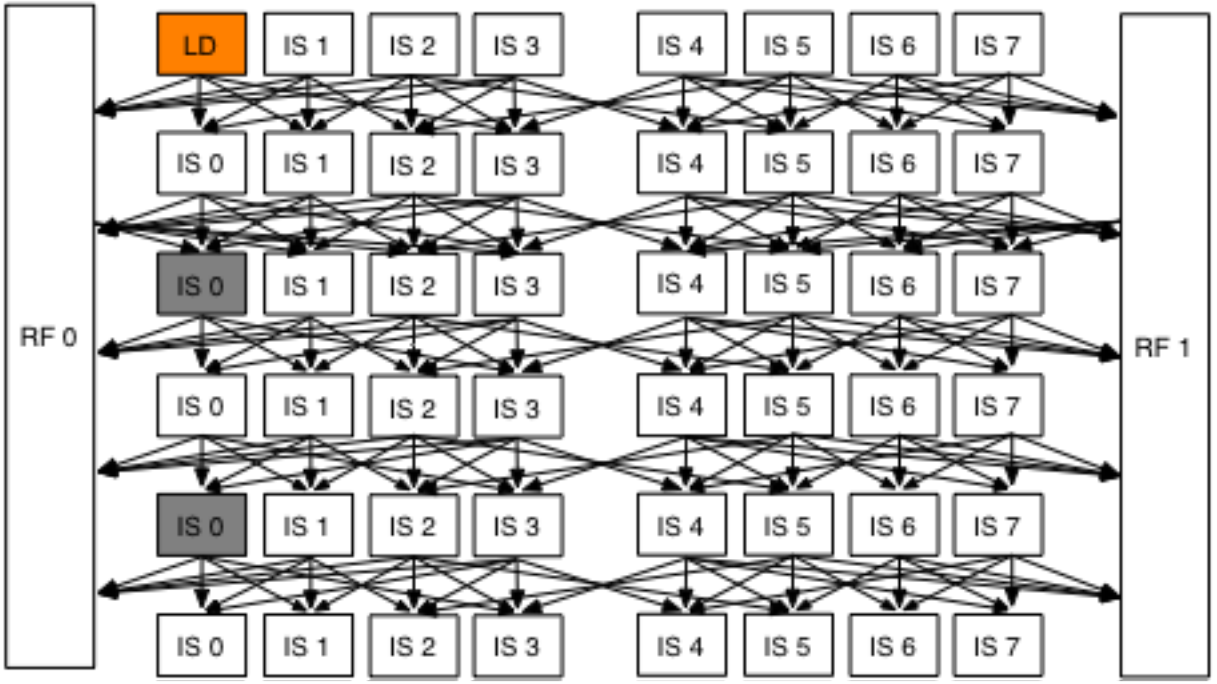
models code



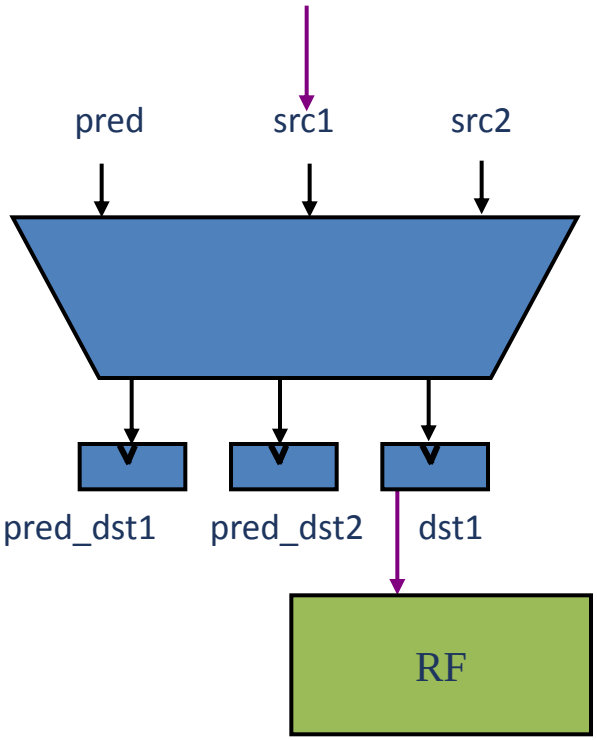
modulo routing resource graph

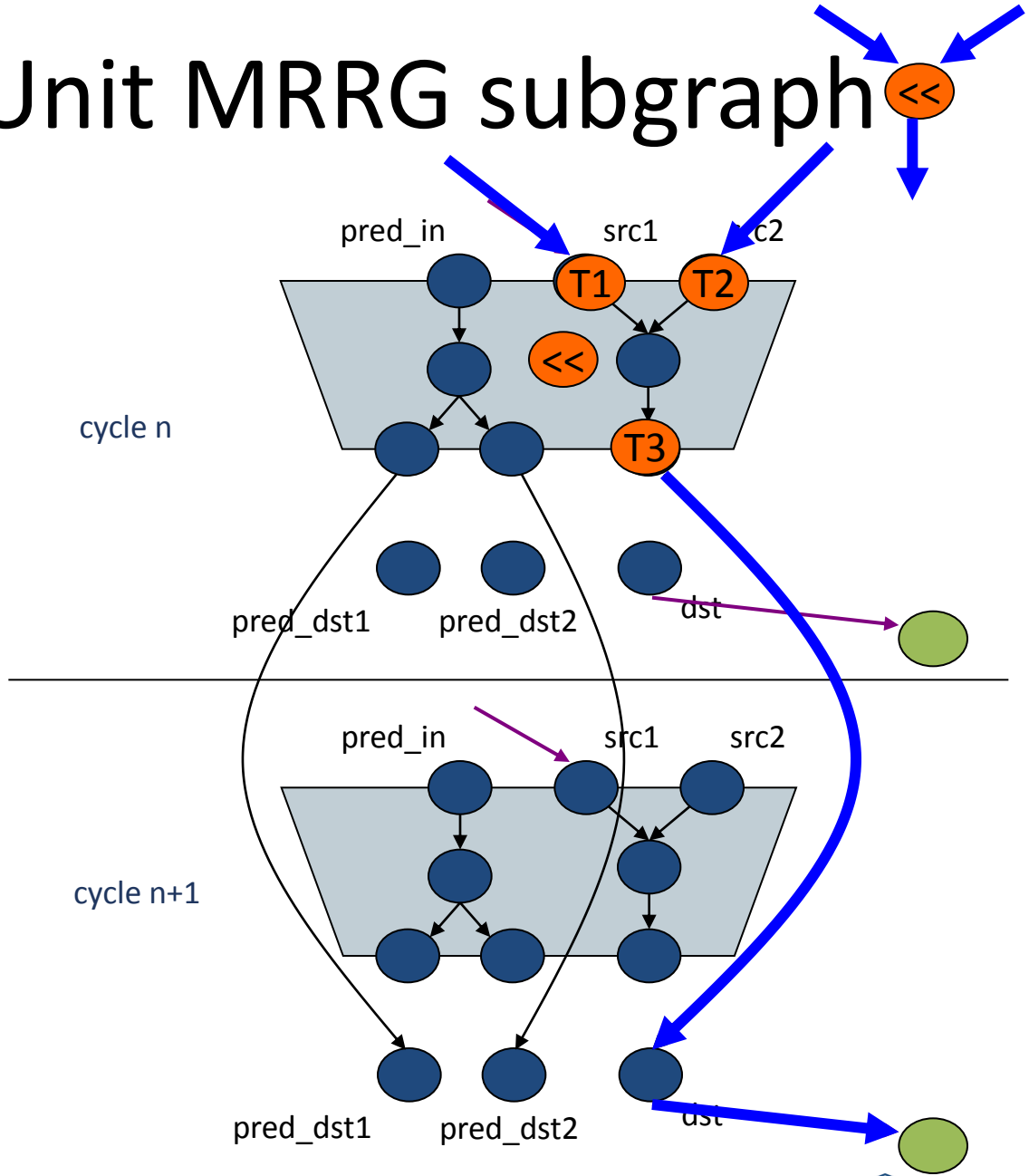
models the whole architecture

at some initiation interval

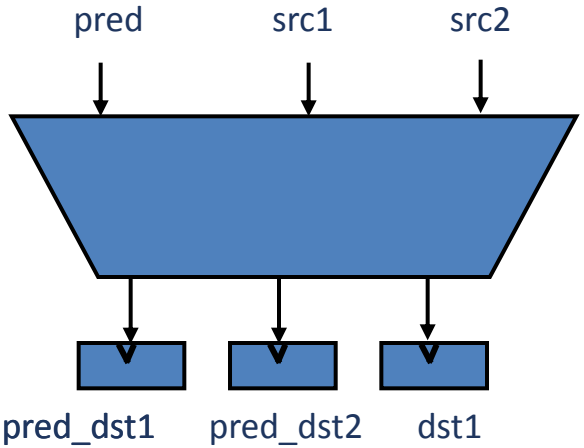


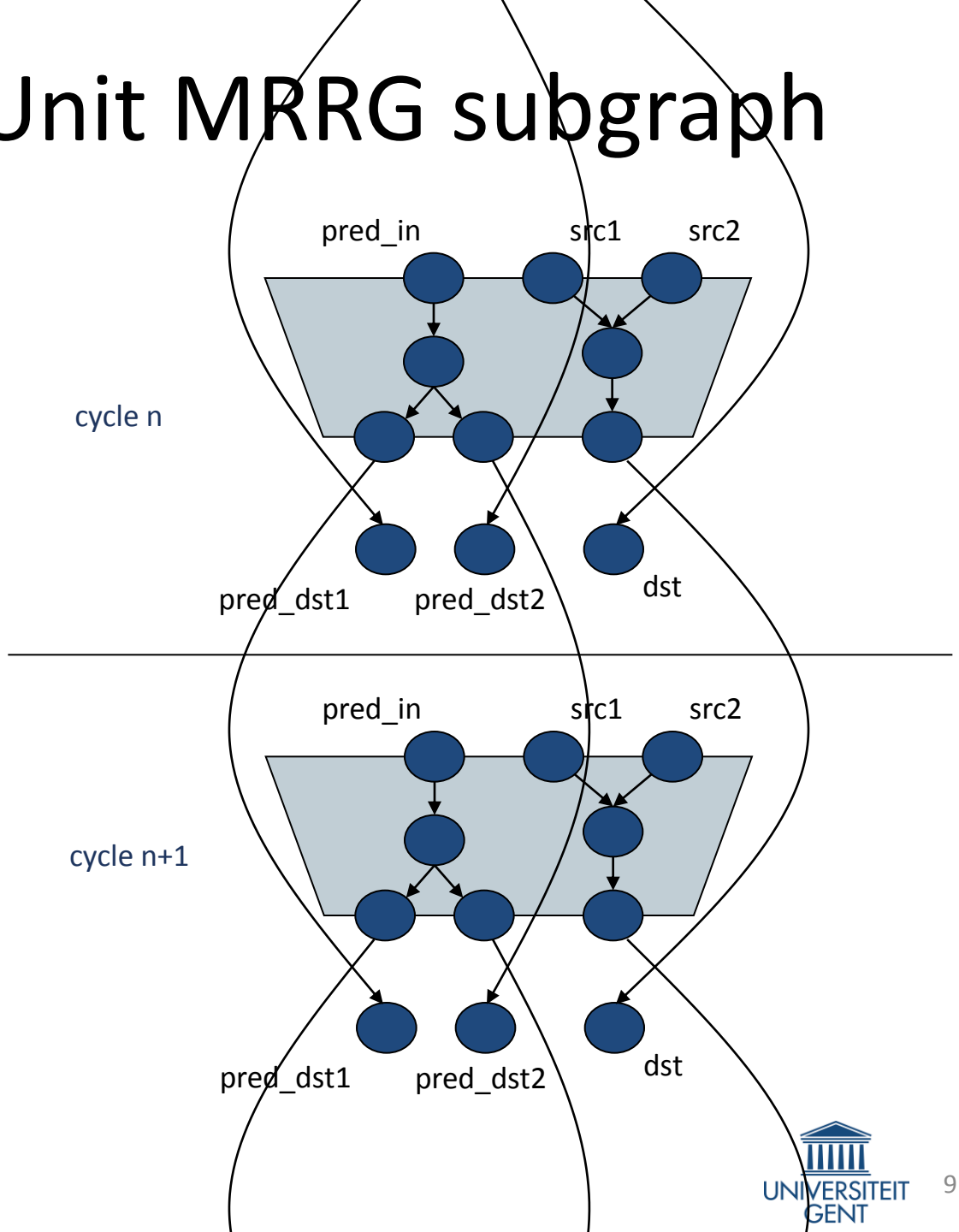
Functional Unit MRRG subgraph



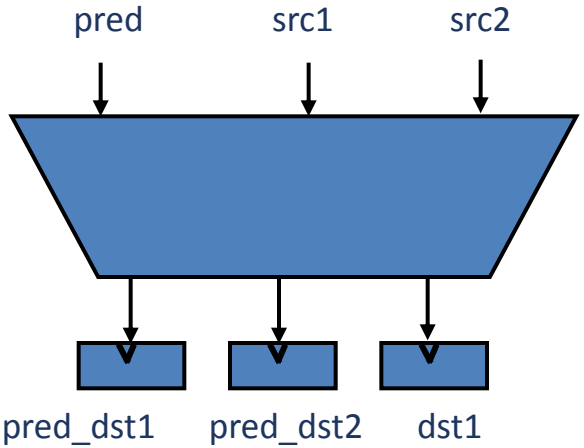


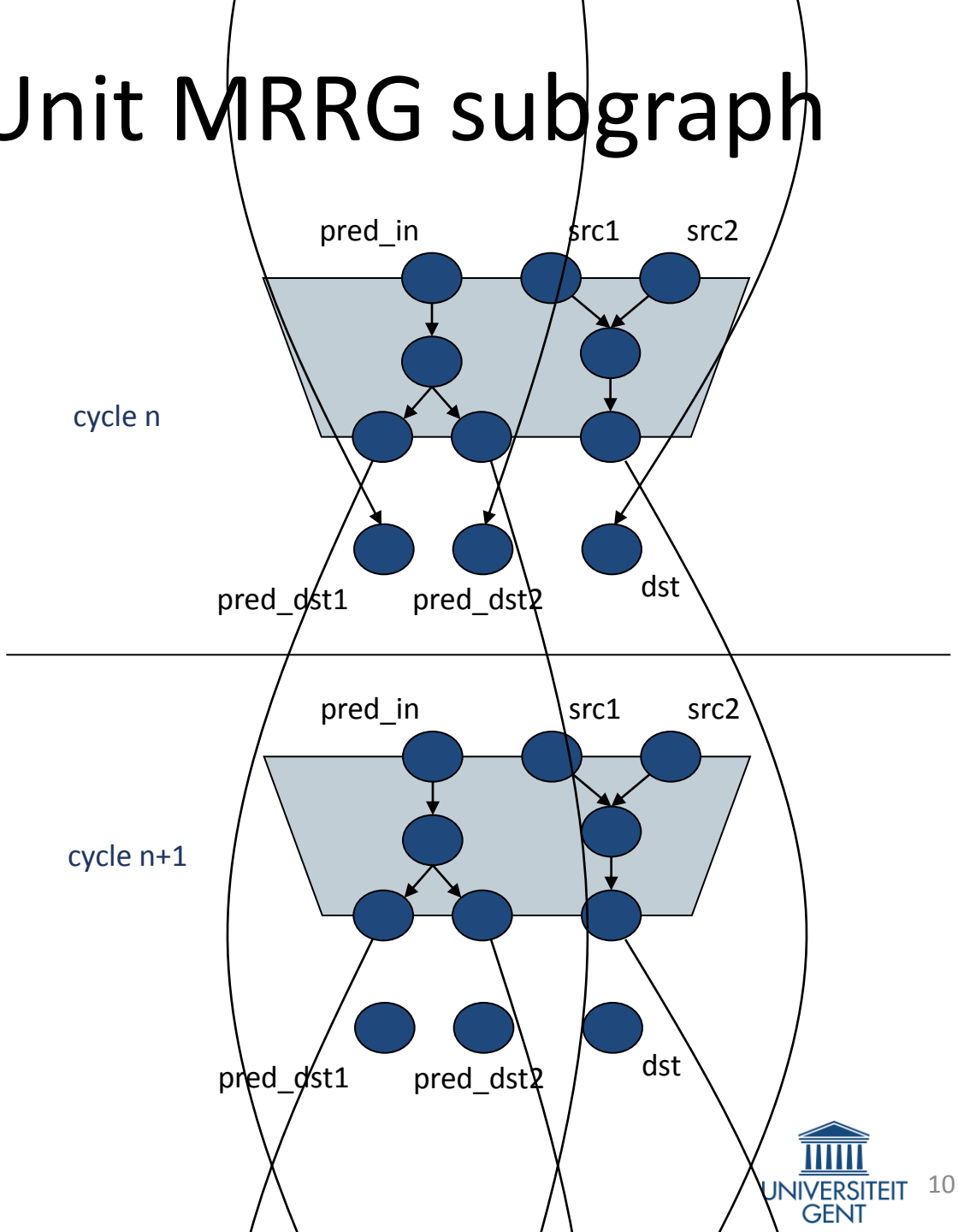
Functional Unit MRRG subgraph





Functional Unit MRRG subgraph

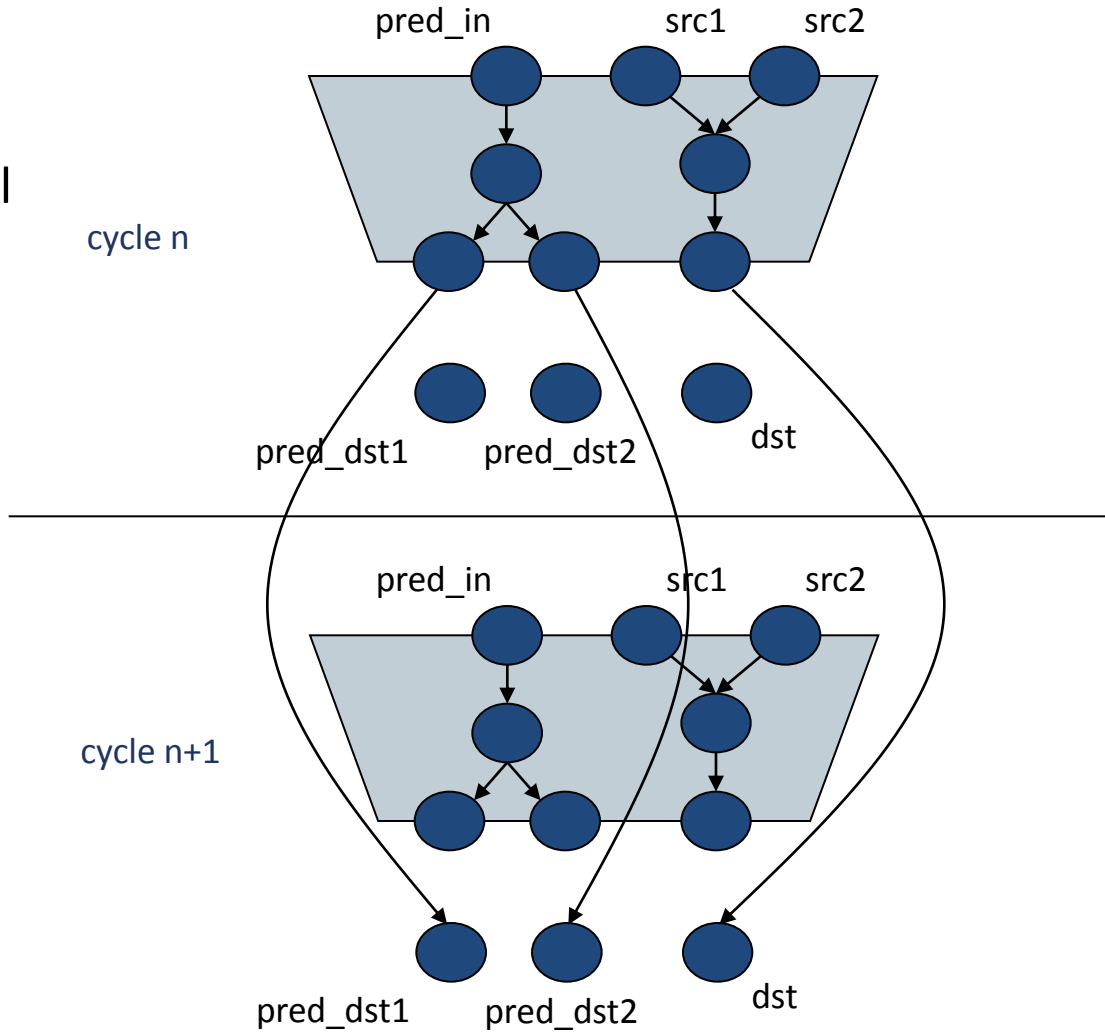




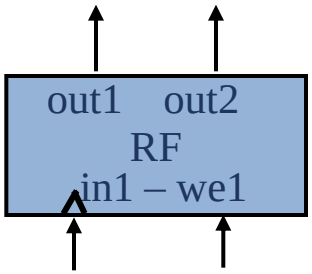
Functional Unit MRRG subgraph

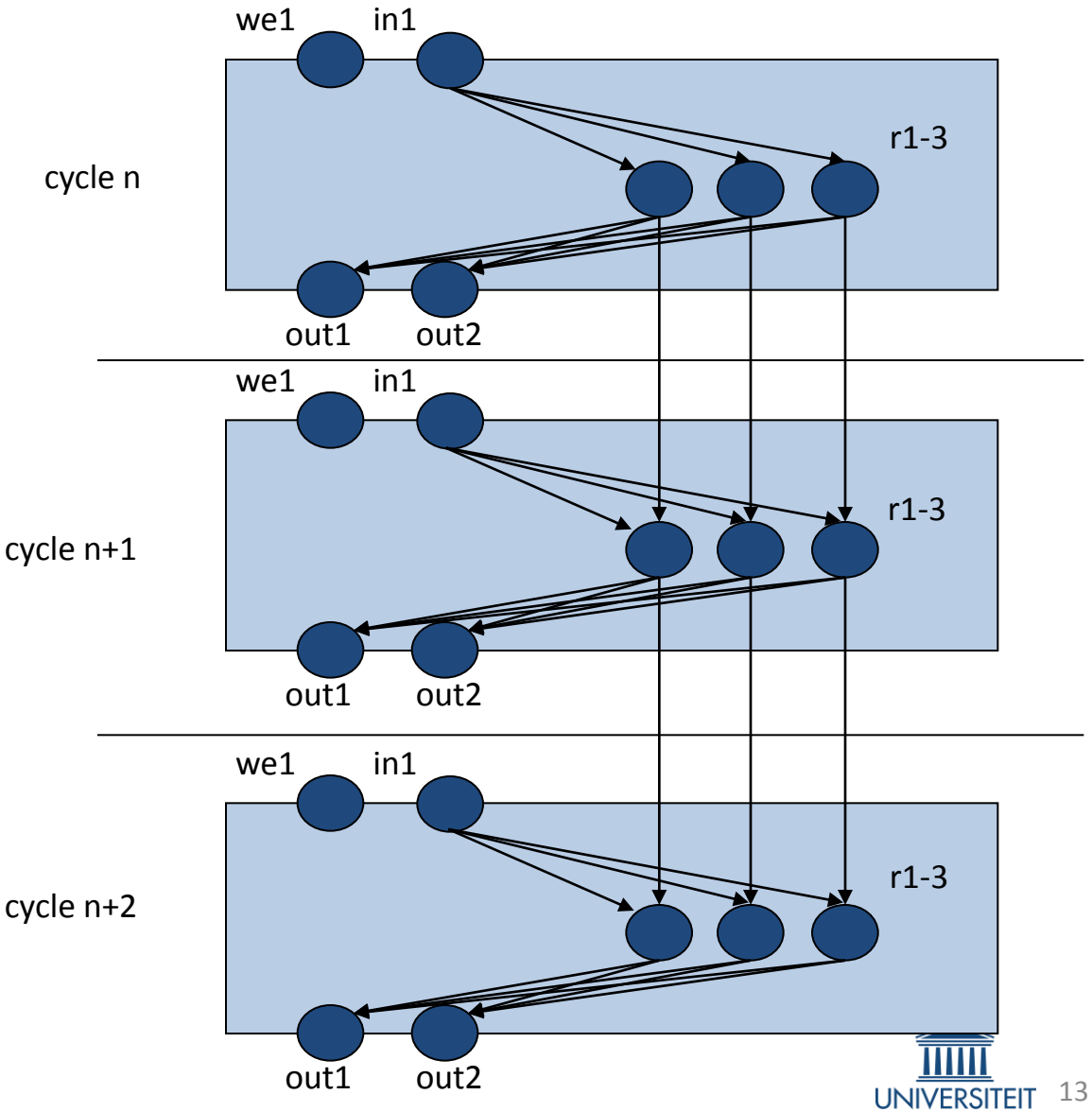
- appropriate connections model latency
- internal nodes and edges model data passing and additional routing freedom
- more src inputs are possible

- all variations use same basics
- adapting the subgraph implements differences
- without requiring changes to mapping algorithm itself
- **inherently retargetable**



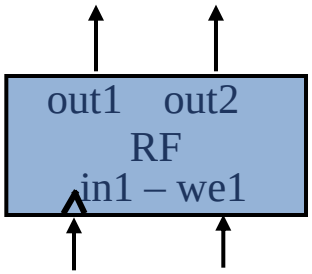
Register File MRRG subgraph

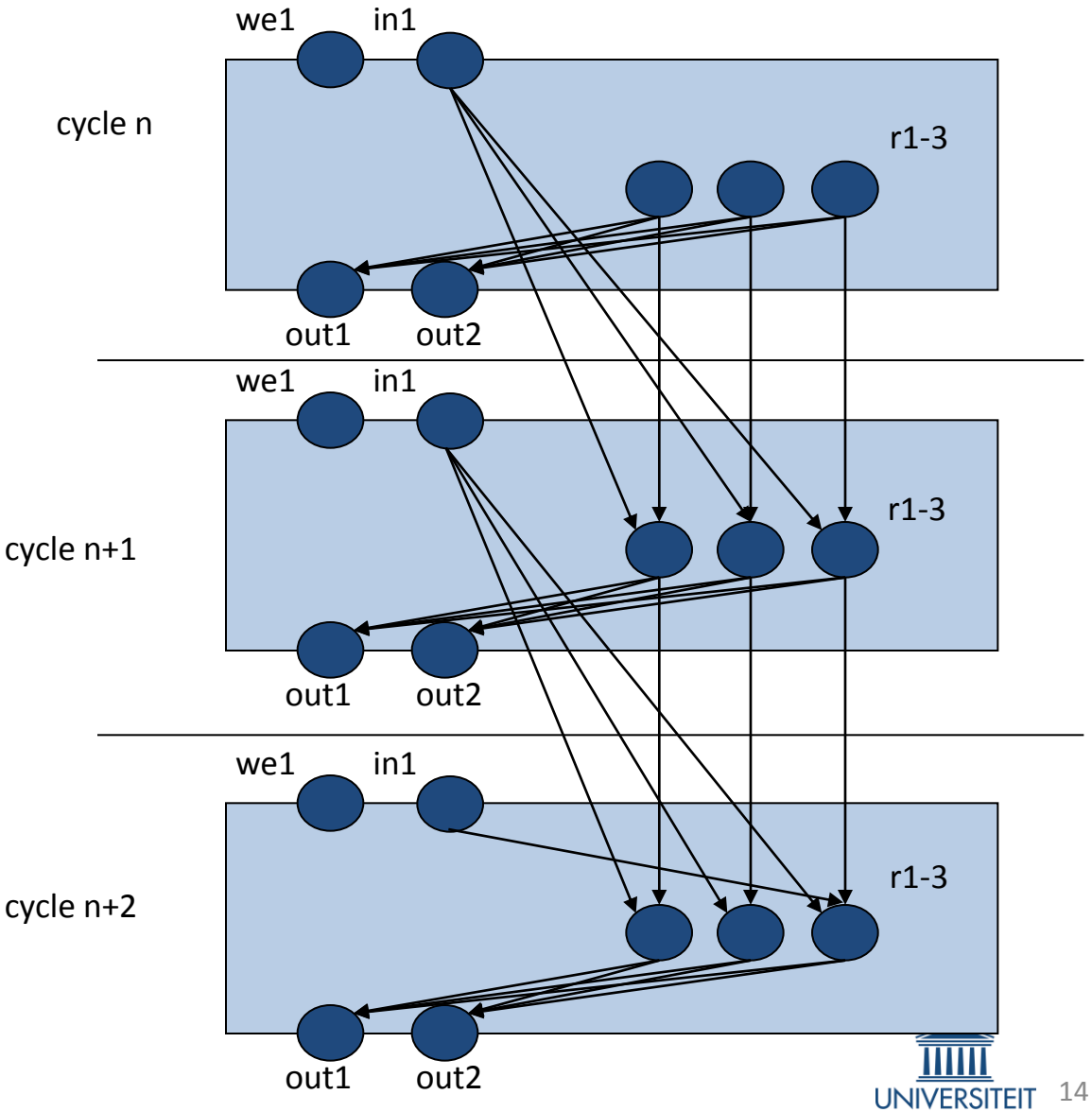




- model different delays:
  - delay 0 (forwarding)

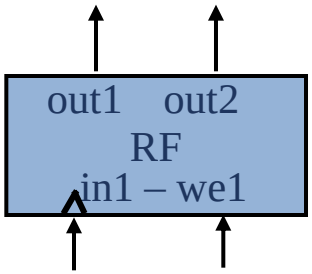
Register File MRRG subgraph

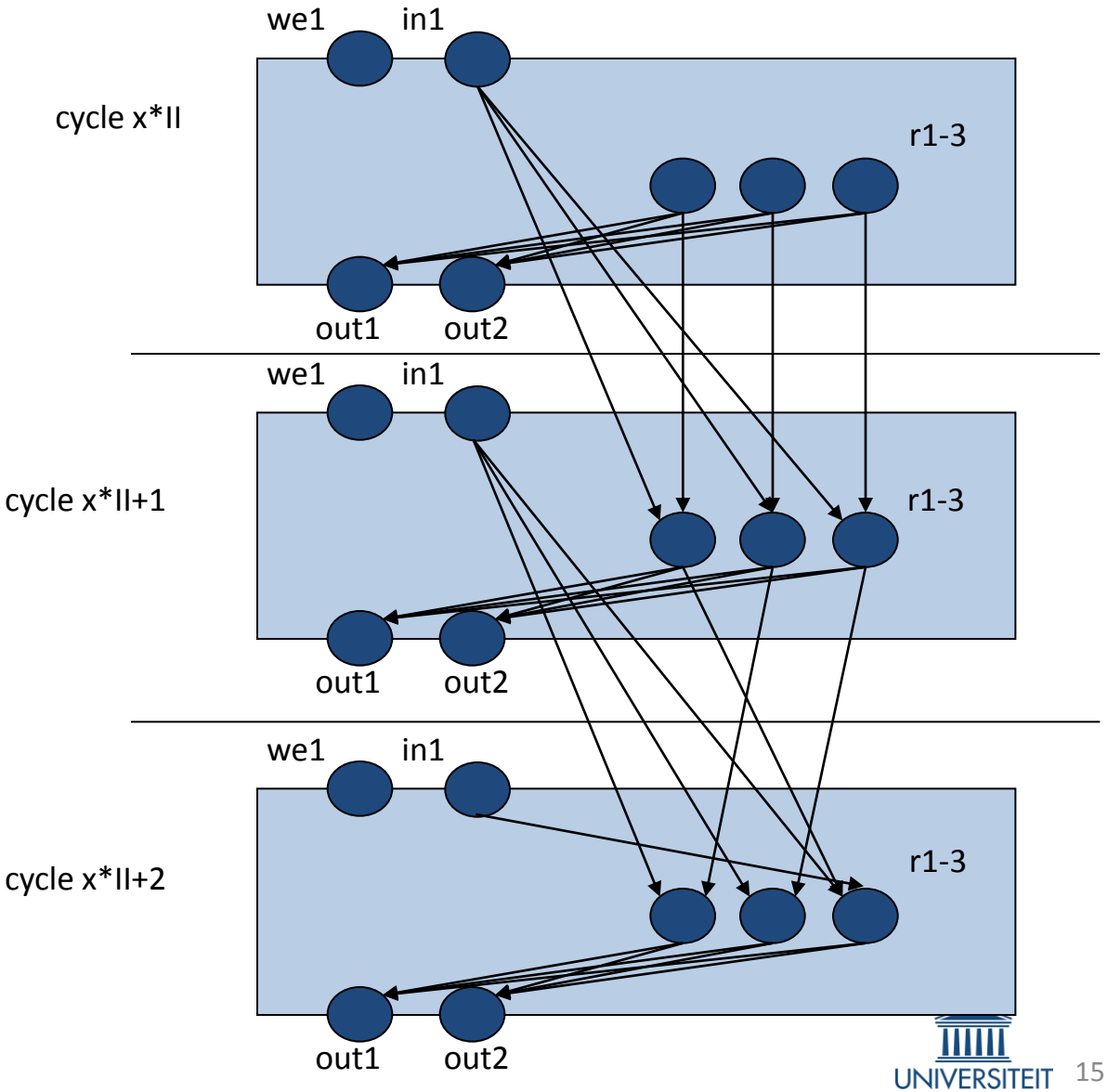




- model different delays:
  - delay 1
- model rotation
  - non-rotating

Register File MRRG subgraph

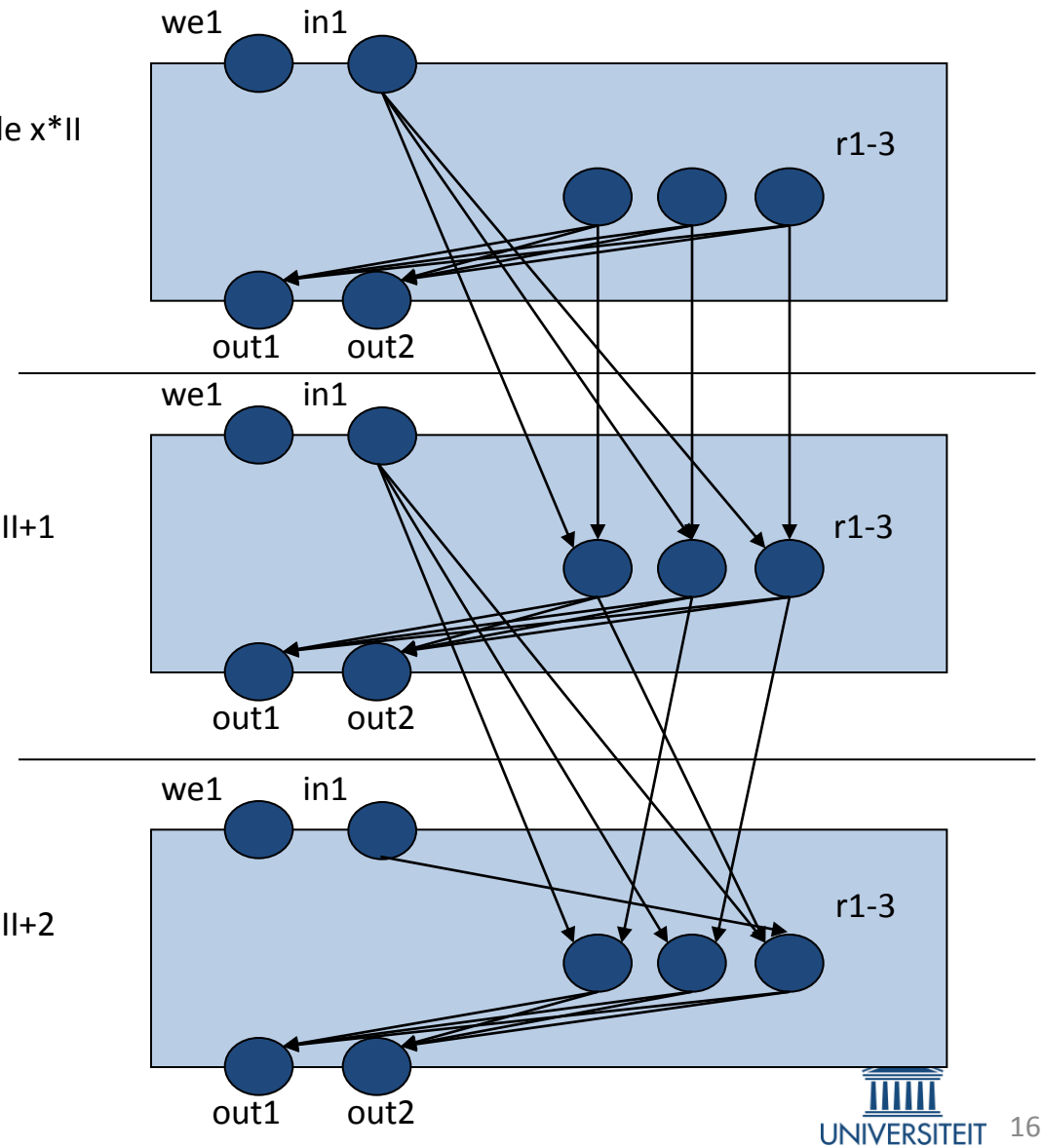




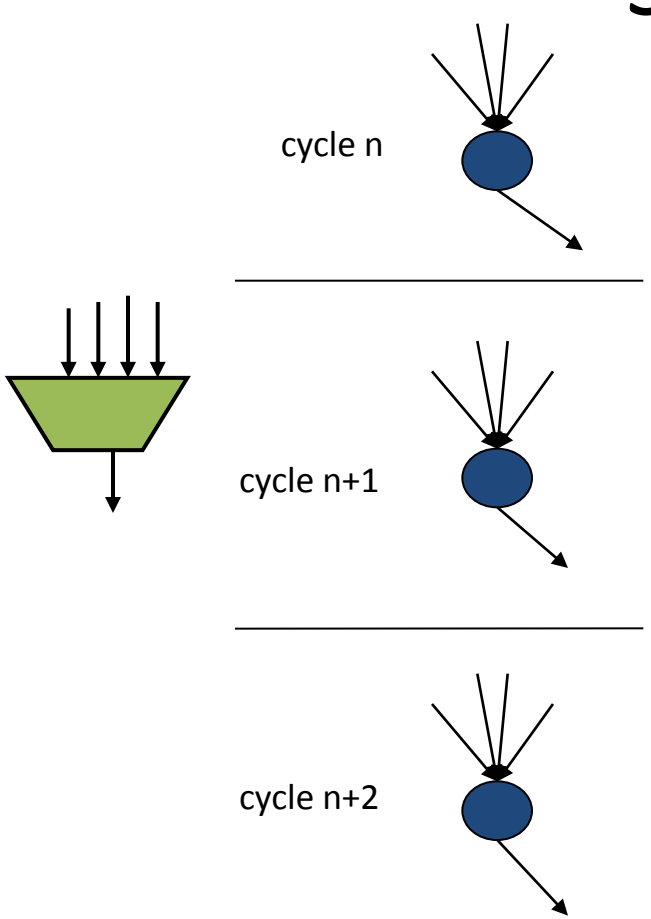
- model different delays:
  - delay 1
- model rotation
  - rotating

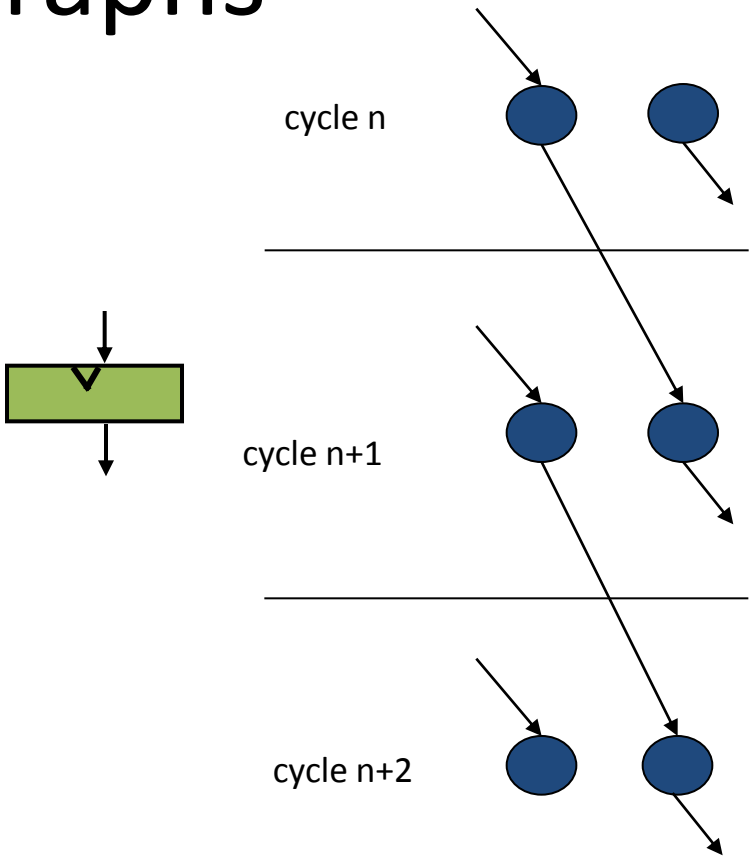
Register File MRRG subgraph

- appropriate connections model latency
- edges model non-rotating vs. rotating RFs
- no explicit register allocation performed
- when a dependency is mapped onto a net through a RF, the value is allocated in the corresponding register
- if not, the value is not allocated to a RF

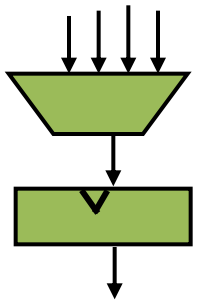


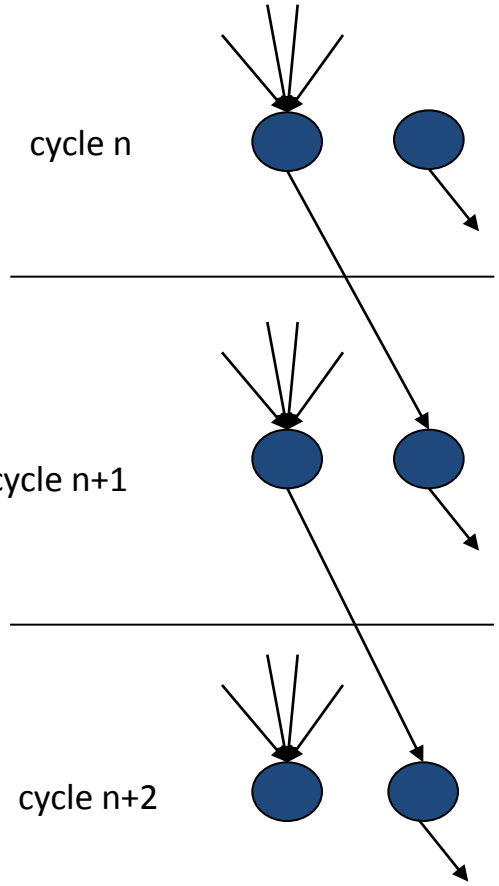
Muxes & pipelining register MRRG subgraphs

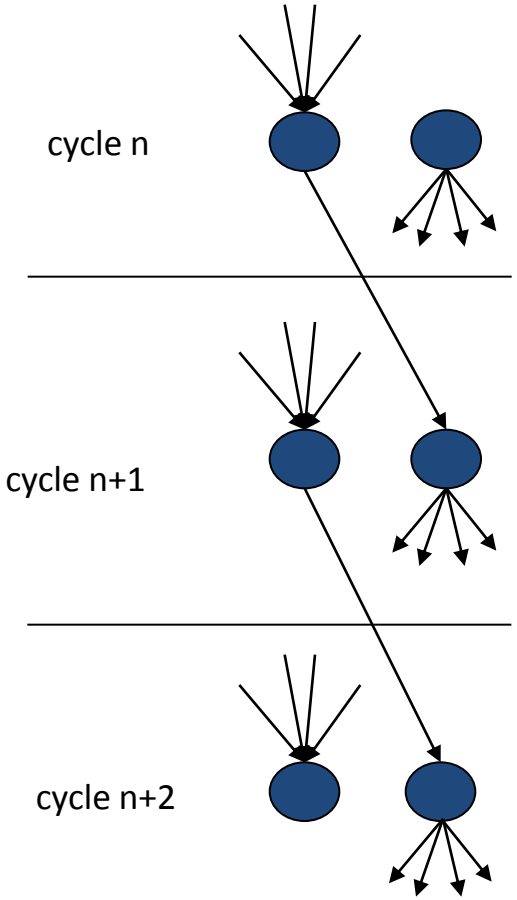




Delayed muxes & bus MRRG subgraphs



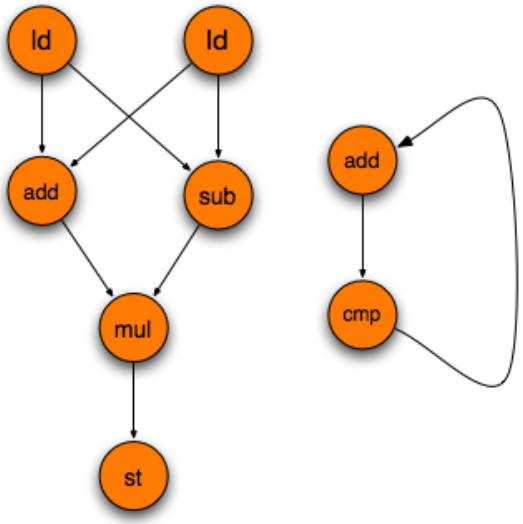




CGRA scheduling = graph mapping

data dependence graph

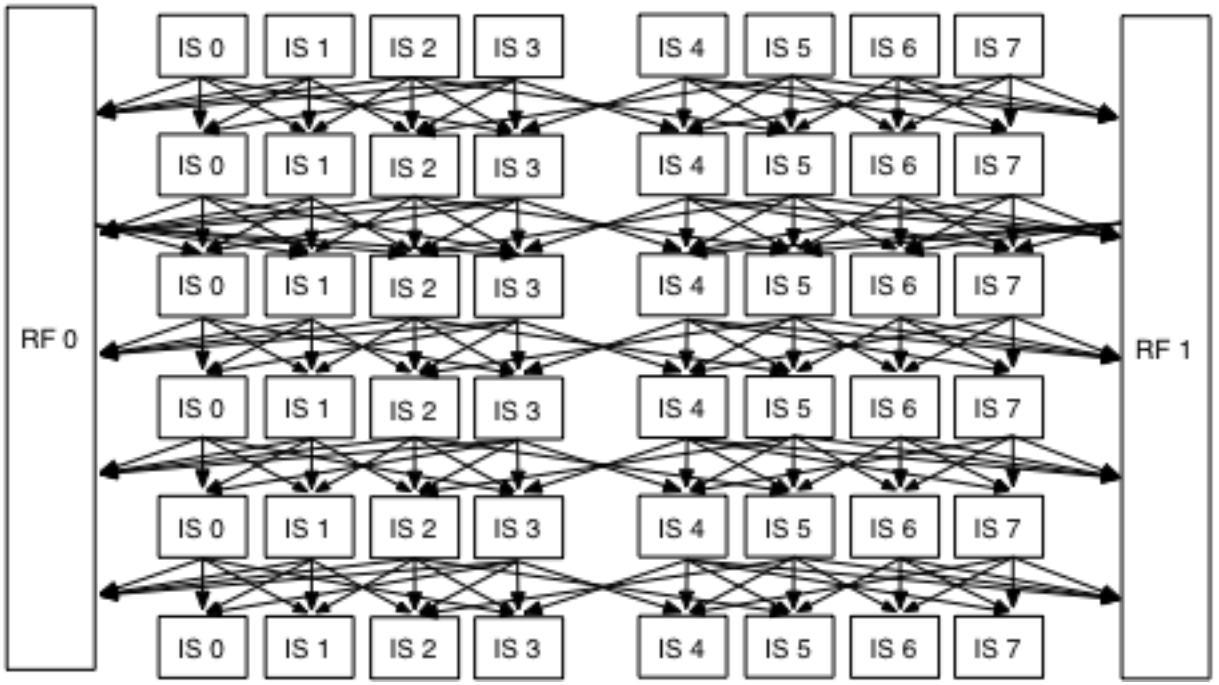
models code



modulo routing resource graph

models the whole architecture

at some initiation interval

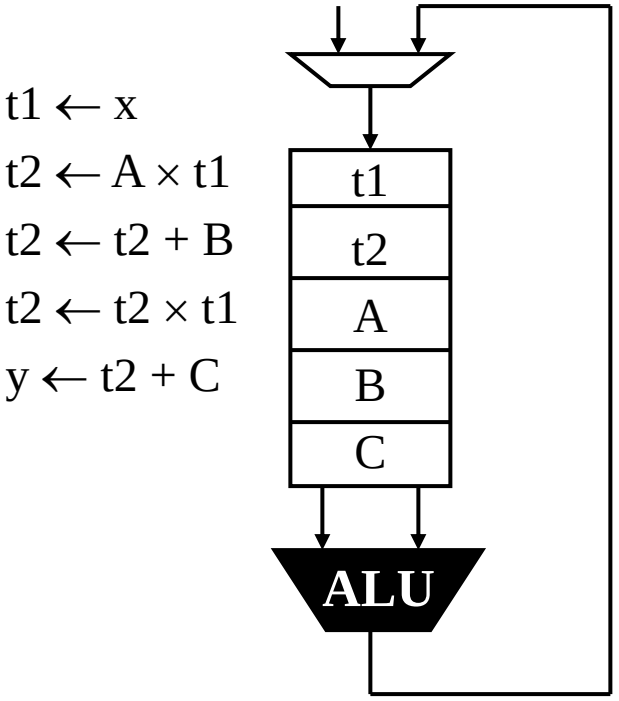


inherently retargetable

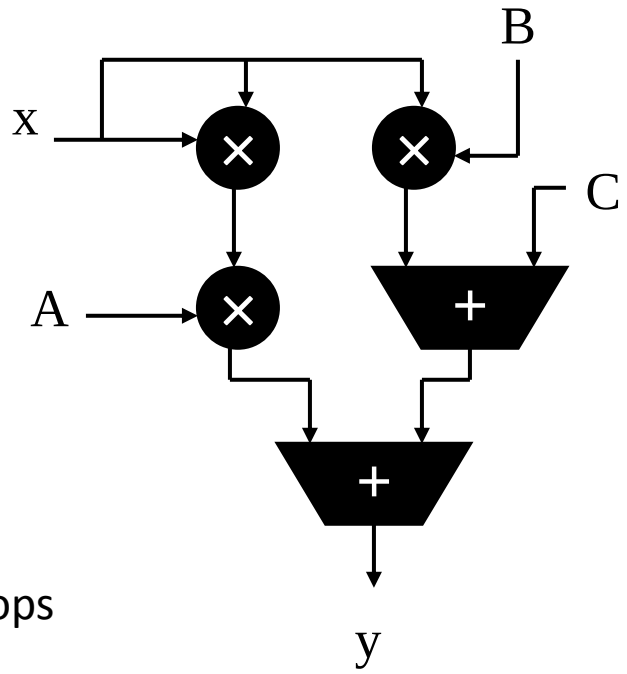
easily model heterogeneous architectures

Data dependence graph

sequential C code



parallel data dependence graph



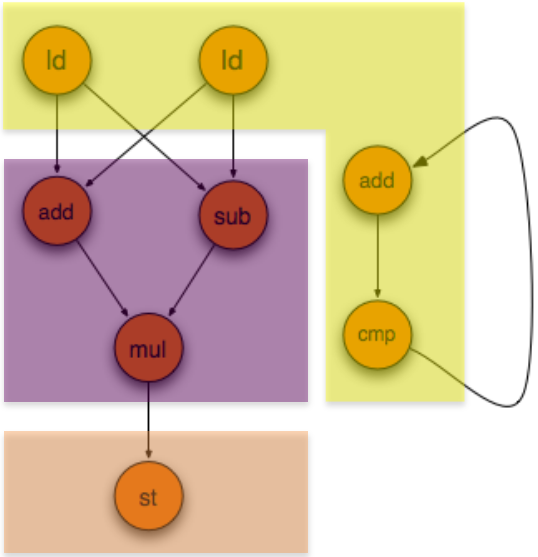
- if-conversion
- predication
- predicate speculation
- hyperblock formation
- inspector-executor loops

- but limited
  - single loop exit
  - not too many predicates

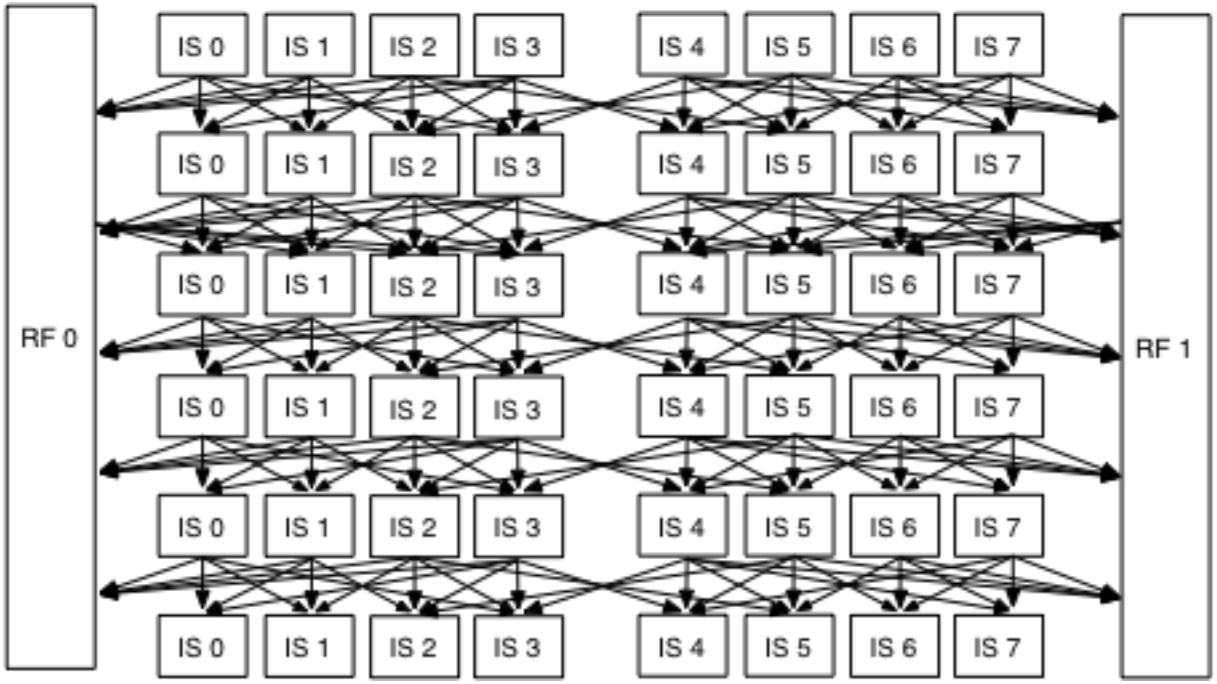
CGRA scheduling = graph mapping

data dependence graph

models code



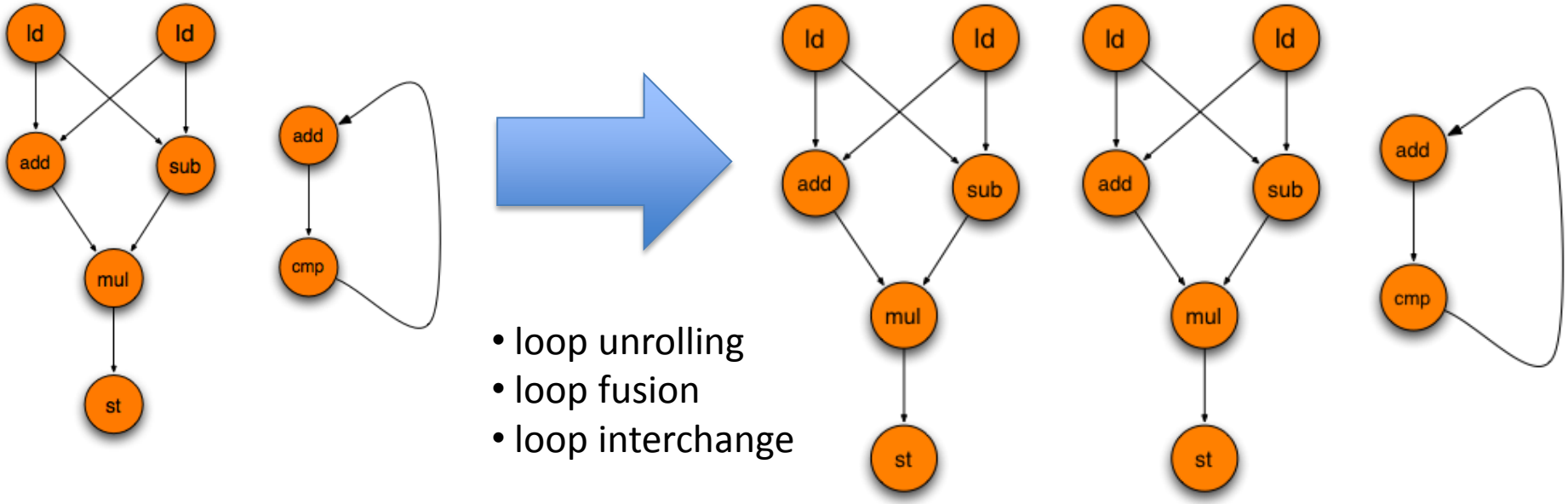
modulo routing resource graph
models the whole architecture
at some initiation interval



minimal initiation interval depends on
- length of recurrence cycle: 2
- number of resources: 8/8 = 1

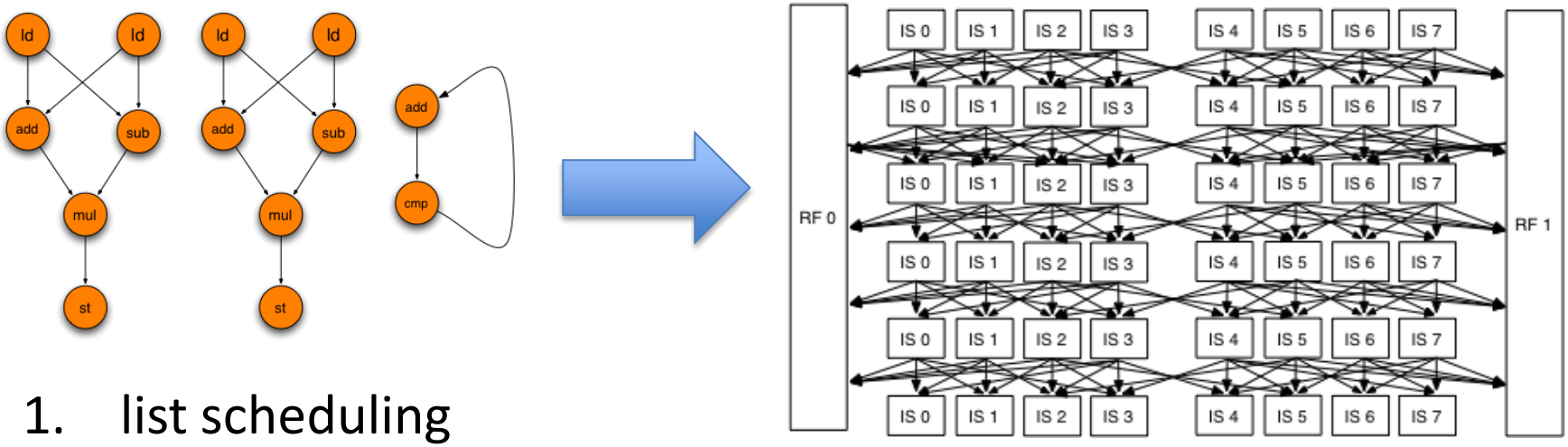
maximum utilization = 1/2

Generate ILP



- loop unrolling
- loop fusion
- loop interchange

CGRA scheduling = graph mapping

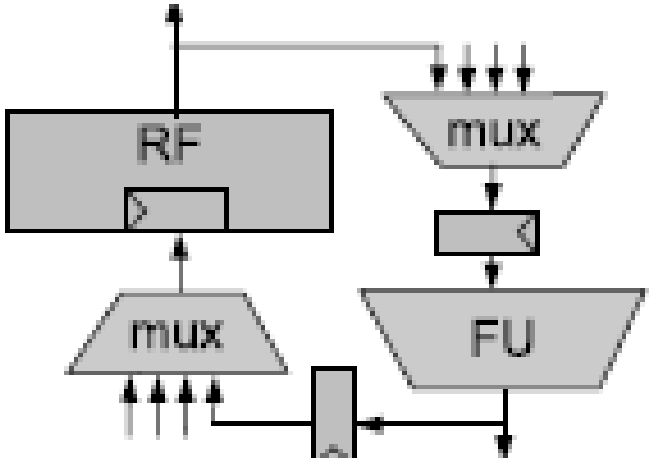


1. list scheduling
   - schedule nodes one by one
   - placement of nodes determined by router
   - resource cost models potential later need for resource
   - most constrained nodes scheduled first, almost no backtracking
2. simulated annealing
   - initial schedule overuses resources
   - try many small changes to schedule until overuse is gone
   - simpler cost function, but it includes overuse

Results multimedia processing

Application	# loops	minimal II	simulated annealing		list scheduling	
			II	time (s)	II	time (s)
3D rendering	80	290	305	44K	315	159
AAC decoder	35	123	141	35K	137	29
AMR-WB+ decoder	44	230	233	20K	235	80
eAAC+ decoder	47	204	217	25K	214	64
H.264 decoder	61	387	470	57K	456	677
mp3 decoder	8	44	45	17K	45	30
MPEG surround decoder	115	522	554	65K	557	510
overall	**390**	**1800**	**1965**	**264K**	**1959**	**1549**

Some more results WLAN



			pipelined			
Loop	#ops	ResMII	RecMII	II	IPC	Time
DemapQAM64	55	4	3	6	9.2	368.1
FFT (64point)	123	8	4	10	12.3	576.7
FFTRadix8	122	8	3	10	12.2	468.3
comp	54	4	4	5	10.8	270.6
DataShuffle	153	14	3	14	10.9	908.5
Average					**11.1**	

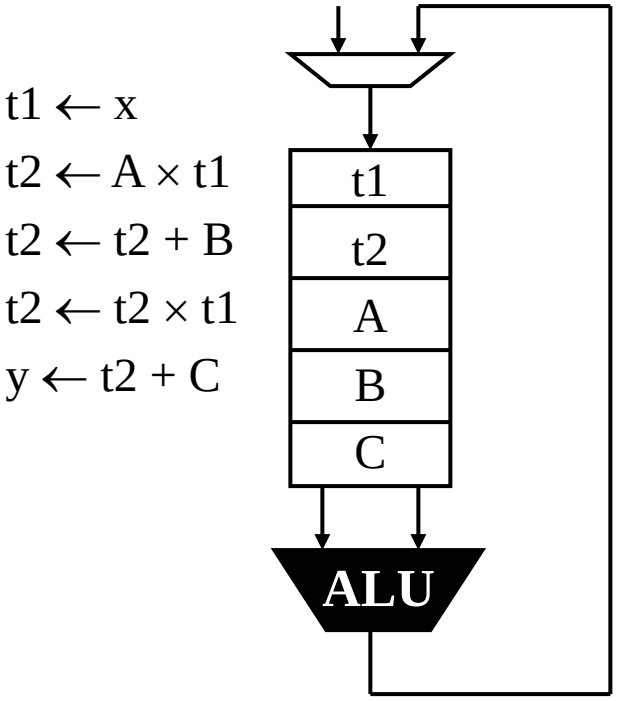
non-pipelined			
RecMII	II	IPC	Time
1	6	9.2	110.8
2	12	10.3	256.0
1	12	10.2	195.6
2	5	10.8	77.3
1	16	9.6	878.0
		10.0	

Overview

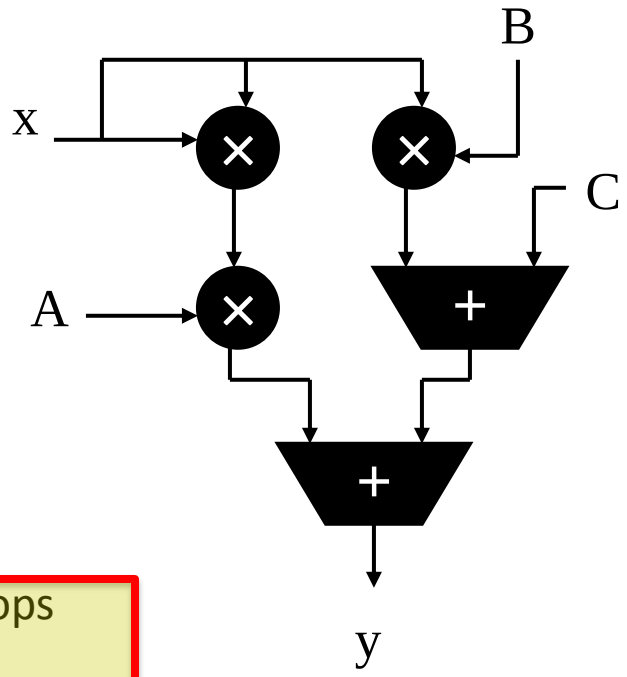
- What makes compilation for CGRAs different
- What CGRA code generation techniques exist?
- What is not automated?

Data dependence graph

sequential C code



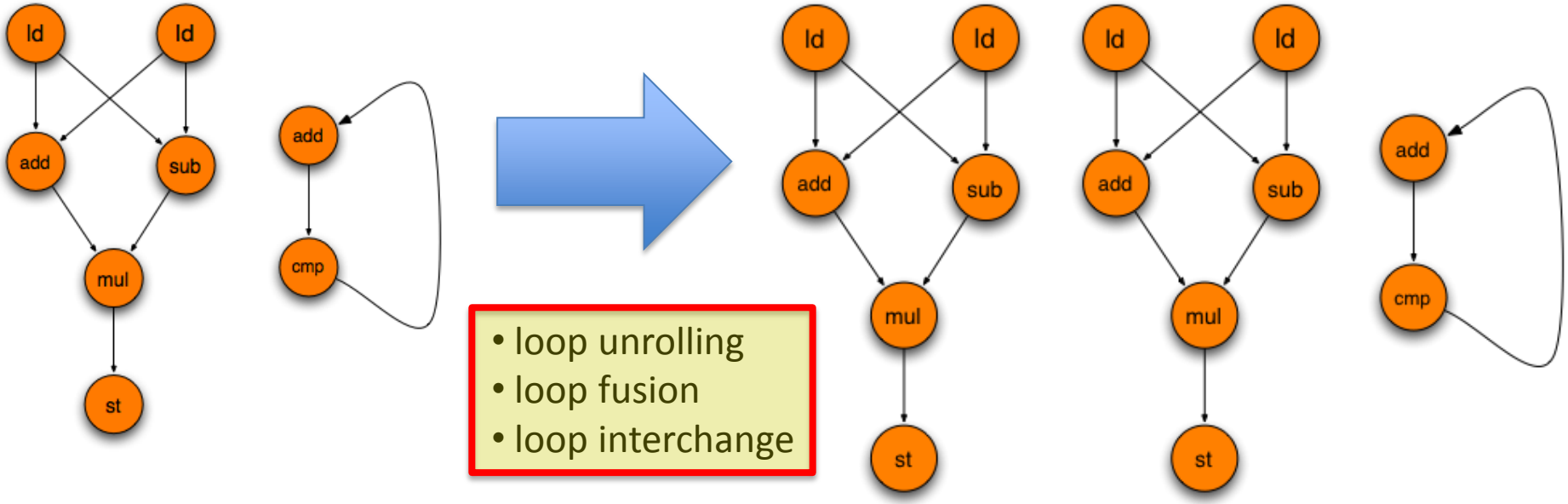
parallel data dependence graph



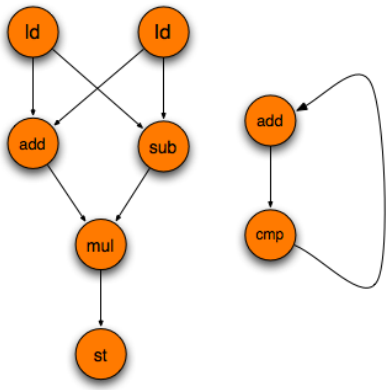
- if-conversion
- predication
- predicate speculation
- hyperblock formation
- inspector-executor loops

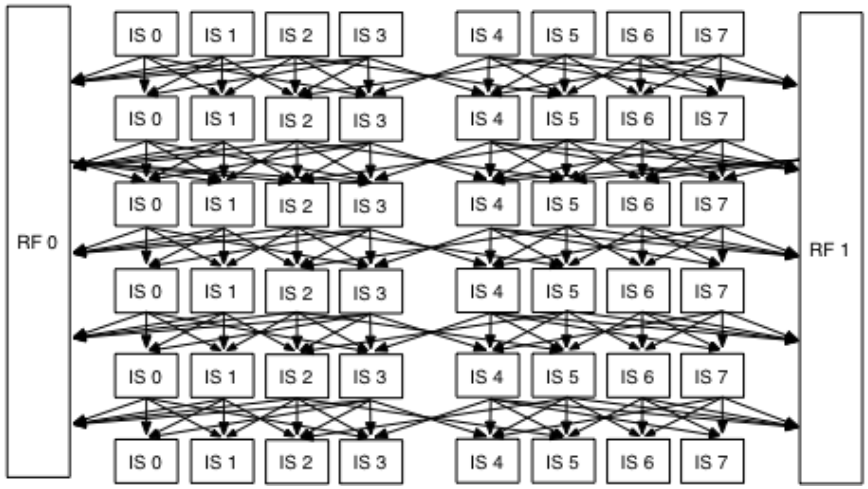
- but limited
  - single loop exit
  - not too many predicates

Generate ILP



CGRA scheduling = graph mapping





1. list scheduling
   - schedule nodes one by one
   - placement of nodes determined by router
   - resource cost models potential later need for resource
   - most constrained nodes scheduled first, almost no backtracking
2. simulated annealing
   - initial schedule overuses resources
   - try many small changes to schedule until overuse is gone
   - simpler cost function, but it includes overuse

Example: FIR filter

(a) original 15-tap FIR filter

```
const short c[15] = {-32, ..., 1216};

for (i = 0; i < nr; i++) {
  for(value = 0, j = 0; j < 15; j++)
    value += x[i+j]*c[j];
  r[i] = value;
}
```

(b) filter after loop unrolling, with hard-coded constants

```
const short c00 = -32, ..., c14 = 1216;

for (i = 0; i < nr; i++)
  r[i] = x[i+0]*c00 + x[i+1]*c01 + ... + x[i+14]*c14;
```

Example: FIR filter

(c) filter after redundant memory accesses are eliminated

```
int i, value, d0, ..., d14;
const short c00 = -32, ..., c14 = 1216;

for (i = 0; i < nr+15; i++) {
  d0 = d1;
  ... ;
  d13 = d14;
  d14 = x[i];
  value = c00 * d0 + c01 * d1 + ... + c14 * d14;
  if (i >= 14) r[i - 14 ] = value;
}
```

loop	4x4 ADRES		TI C64+	
	cycles	mem accesses	cycles	mem access
(a)	11828	6221	1054	1618
(b)	1247	3203	1638	2799
(c)	664	422	10062	416

Further reading

- **hyperblock formation – predication**
  MAHLKE , S.A. , LIN , D. C. , CHEN , W.Y. , HANK, R.E. , and BRINGMANN , R. A. Effective compiler support for predicated execution using the hyperblock. In MICRO 25: Proceedings of the 25th annual international symposium on Microarchitecture (Los Alamitos, CA, USA, 1992), IEEE Computer Society Press, pp. 45–54.

- **list scheduling for CGRAs**
  OH, T. , EGGER, B. , PARK, H. , and MAHLKE , S . Recurrence cycle aware modulo scheduling for coarse-grained reconfigurable architectures. In LCTES '09: Proceedings of the 2009 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems (New York, NY, USA, 2009), ACM, pp. 21–30.
  PARK , H. , FAN , K. , MAHLKE , S . A. , OH , T. , KIM, H. , and KIM, H. - S . Edge-centric modulo scheduling for coarse-grained reconfigurable architectures. In PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques (New York, NY, USA, 2008), ACM, pp. 166–176.

- **simulated-annealing based scheduling for CGRAs**
  MEI, B. , VERNALDE, S. , VERKEST, D. , and LAUWEREINS , R. Design methodology for a tightly coupled VLIW/reconfigurable matrix architecture: A case study. In Proc. of Design, Automation and Test in Europe (DATE) (2004), pp. 1224–1229.

Further reading

- **register allocation for CGRAs**
  DE SUTTER, B. , COENE, P. , VANDER AA, T., and ME , B. Placement-and-routing-based register allocation for coarse-grained reconfigurable arrays. In LCTES '08: Proceedings of the 2008 ACM SIGPLAN-SIGBED conference on Languages, compilers, and tools for embedded systems (New York, NY, USA, 2008), ACM, pp. 151–160.
- **memory bank aware code generation for CGRAs**
  KIM, Y. , LEE, J., SHRIVASTAVA, A., Paek, Y. Operation and data mapping for CGRAs with multi-bank memory. In LCTES'10: Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems (Stockholm, Sweden, 2010), ACM, pp. 17-26.

- **Overview of CGRA techniques and issues**
  Handbook of Signal Processing Systems. Bhattacharyya, S.S.; Deprettere, E.F.; Leupers, R.; Takala, J. (Eds.) 2010, XXXVIII,1117 p., Springer, ISBN: 978-1-4419-6344-4

- **Results on real-world applications on ADRES using simulated-annealing**
  BOUGARD, B., DE SUTTER, B., VERKEST, D., VAN DER PERRE, L. and LAUWEREINS, R. A Coarse-Grained Array Accelerator for Software-Defined Radio Baseband Processing. IEEE Micro 28(4), p. 41-50, July 2008.
  BINGFENG, M., DE SUTTER, B., VANDER AA, T., WOUTERS, M., KANSTEIN, A., DUPONT, S. Implementation of a Coarse-Grained Reconfigurable Media Processor for AVC Decoder. Journal of VLSI Signal Processing Systems 51(3) p 225-243, June 2008