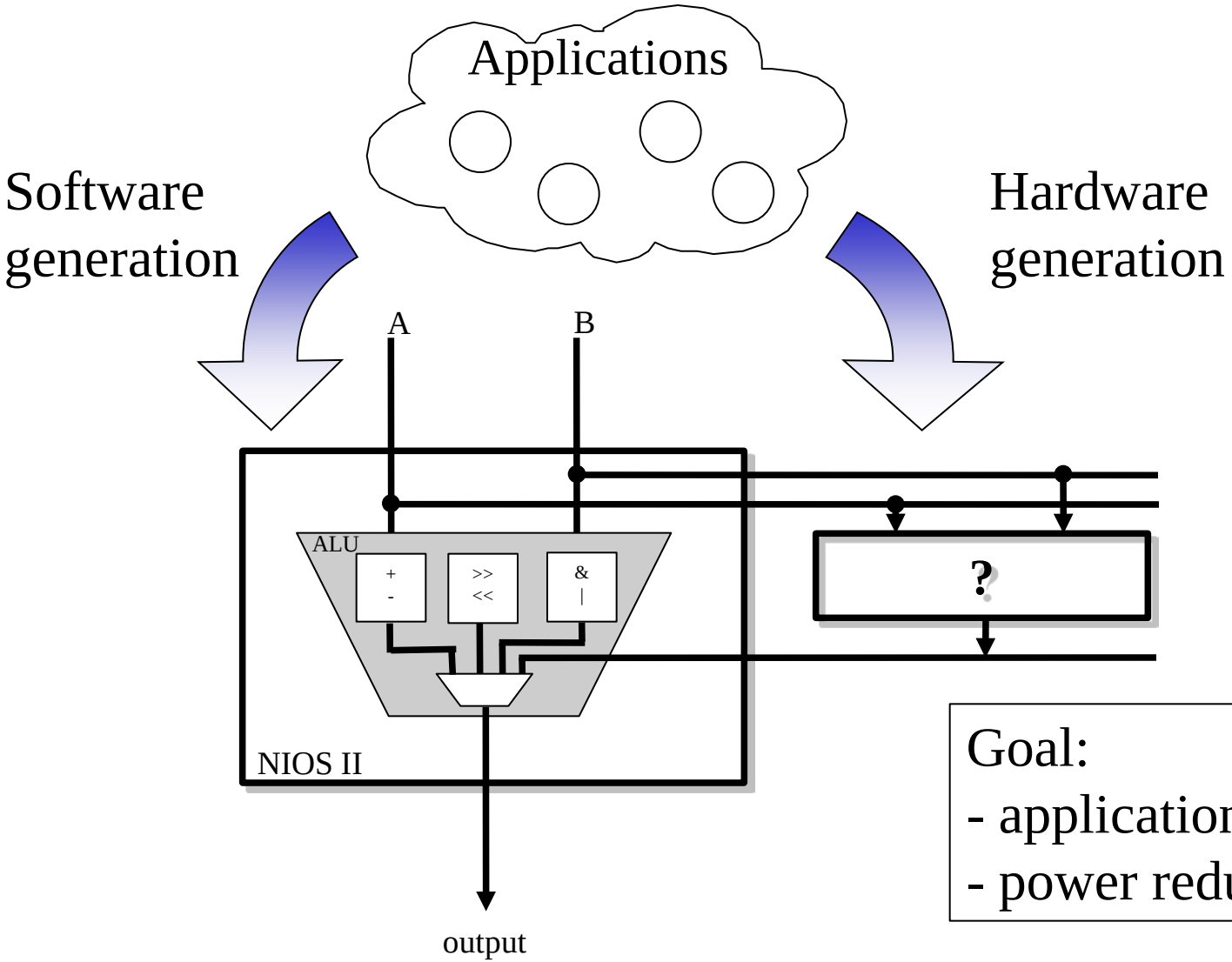
Sélection automatique d'instructions et ordonnancement d'applications basés sur la programmation par contraintes

Kevin Martin[1] , Christophe Wolinski[1,2] , Krzysztof Kuchcinski[3] , Antoine Floch[1] , and François Charot[1]

[1] INRIA Rennes - Bretagne Atlantique, France
[2] University of Rennes I, Irisa, France
[3] Dept. of Computer Science, Lund University, Sweden

- Introduction
- Constraint programming
- Pattern generation
- Pattern selection and scheduling
- Instruction reordering
- Results
- Conclusion

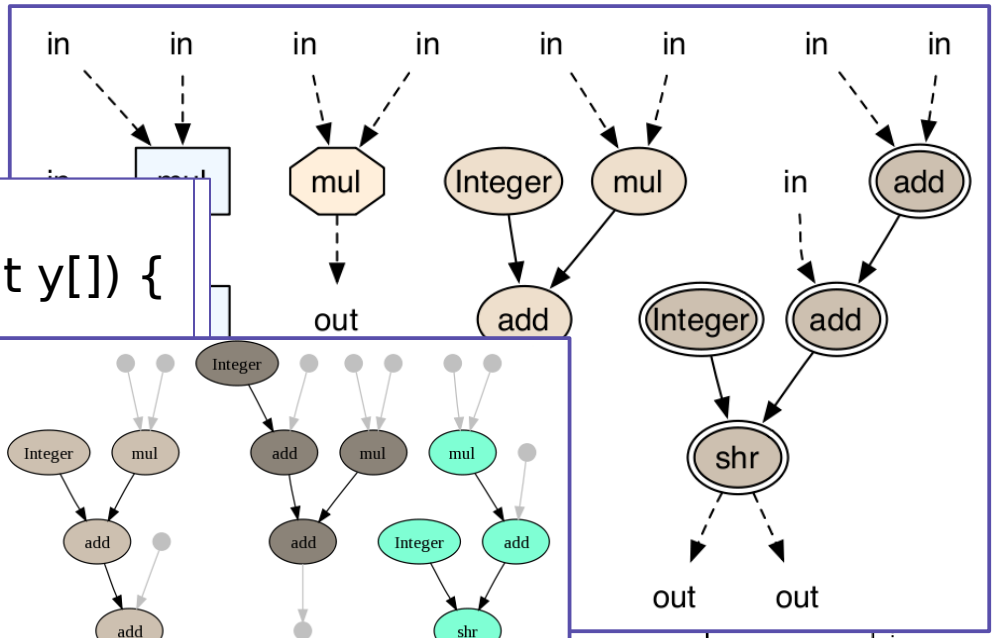
Automatic Instruction-Set Extension

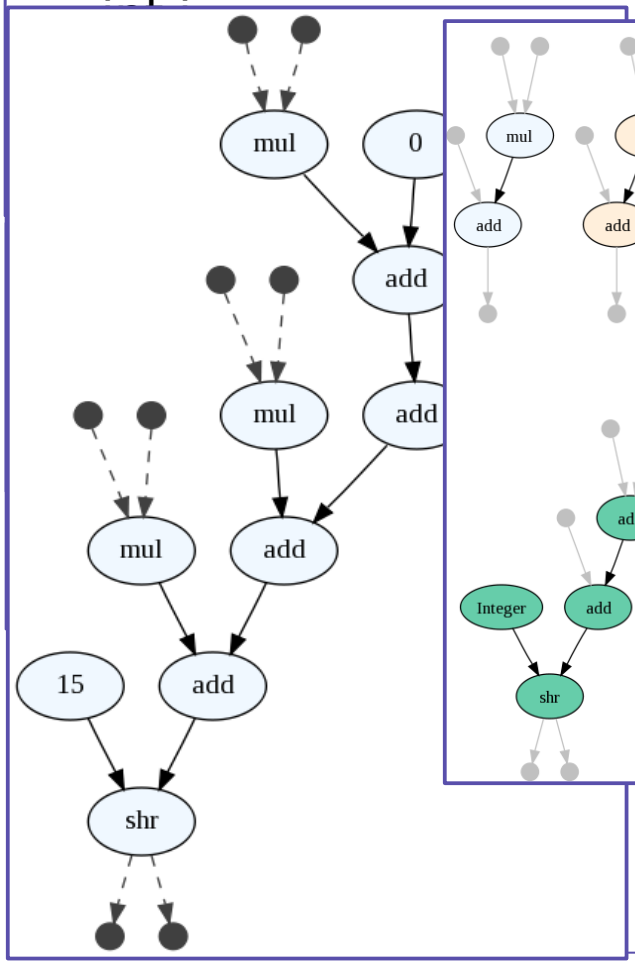


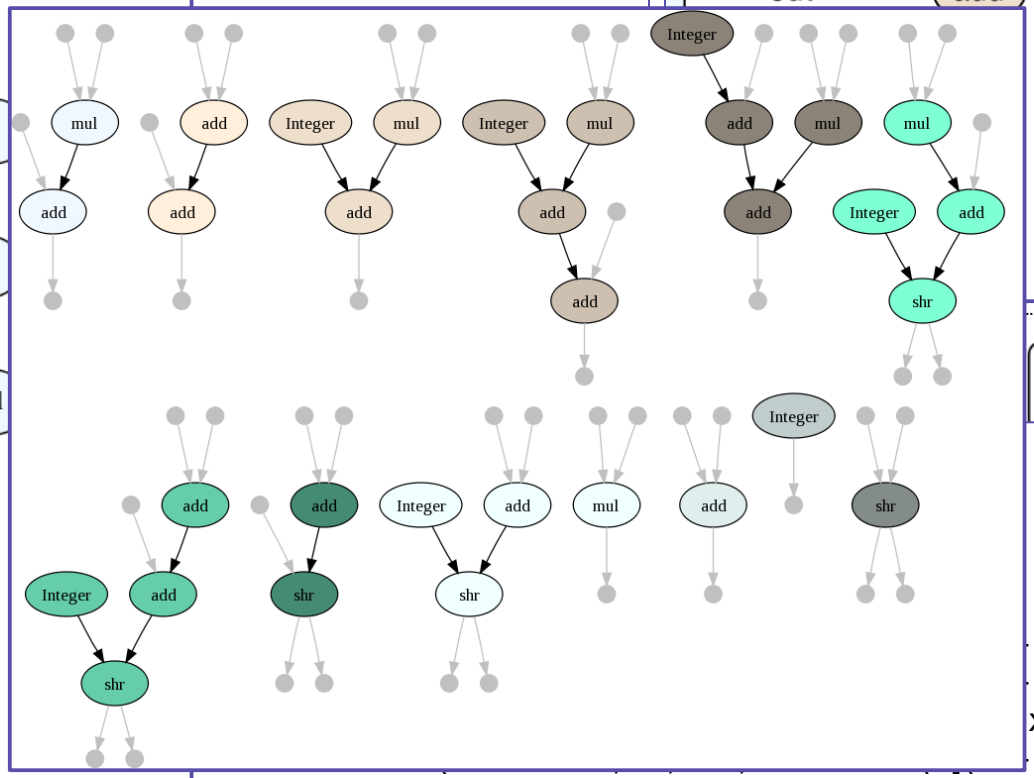
Goal:
- application speed-up
- power reduction

DURASE flow

```
/* Transformed C code */
void fir(const int x[], const int h[], int y[]) {
```








```
j)]), "r"(h[0]));
j)]), "r"(h[1]));
x[(3 + j)]), "r"(h[3]));
j)]), "r"(h[2]));
  asm volatile ("custom 4, %0, %1, %2" : "=r"(sum) : "r"(tmp1), "r"(tmp2));
  y[j] = sum;
}
```

- Introduction
- Constraint programming
- Pattern generation
- Pattern selection and scheduling
- Instruction reordering
- Results
- Conclusion

CP(FD) vs (M)LP

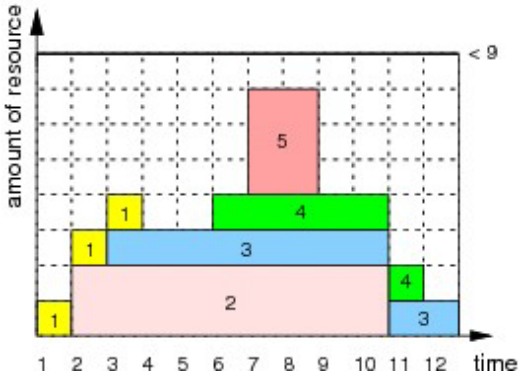
	CSP	ILP
Variables	discret	continue and discret
Constraints	linear and non-linear global constraints	linear
Modeling	flexible, problem structure preserved	linearized model
Solving	local consistency Depth-first search	linear relaxation (simplex) Branch and bound
Search	possible heuristics	standard methods

Constraint programming

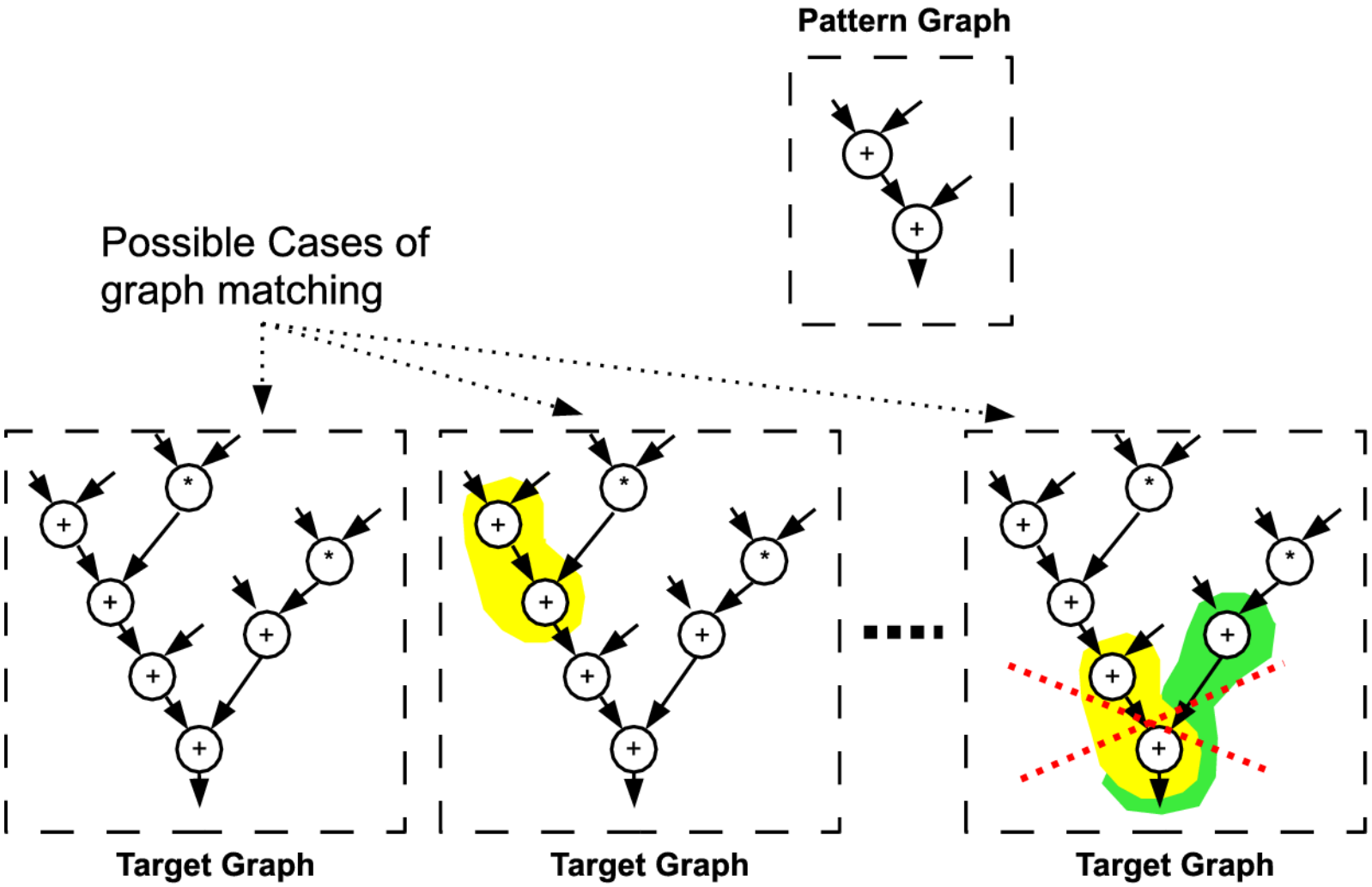
Formally, a Constraints Satisfaction Problem (CSP) is defined by a 3-tuple (V;D;C)
Where :

- V = x1; x2; ... ; x_n is a finite set of finite domain variables (FDV's)
- D = D1; D2; ... ; D_n is finite set of domains
- C is a set of constraints restricting the values that the variables can simultaneously take. In practice, the constraints are defined by equations, inequalities and combinatorial constraints.

Example of a combinatorial constraint : the cumulative constraint



Constraint : Graph Match



- Introduction
- Constraint programming
- Pattern generation
- Pattern selection and scheduling
- Instruction reordering
- Results
- Conclusion

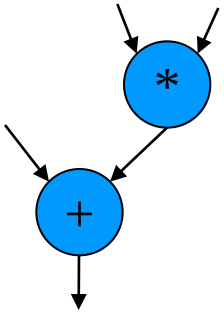
Pattern Generation

- What is a pattern ?
- How to generate it ?

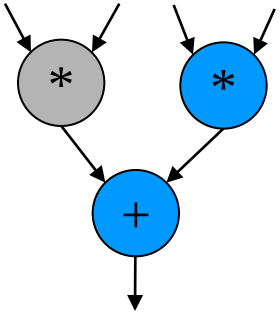
What is a pattern ?

- A connected graph

Pattern p



A match m of pattern p in graph G



How to generate patterns ?

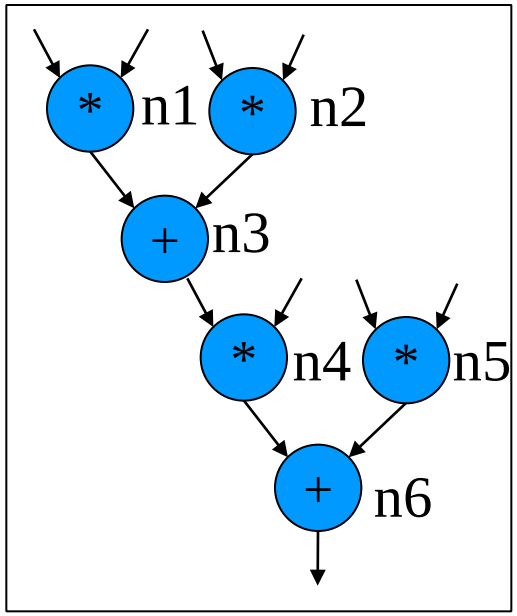
- Subgraph enumeration
- Generation of computational patterns under architectural and technological constraints
  - Number of inputs
  - Number of outputs
  - Critical path
  - Number of ressource
  - ...
- Problem of (sub-)graph matching

[Ref] K. Martin, C. Wolinski, K. Kuchcinski, F. Charot et A. Floc'h : *Constraint-driven Identification of Application Specific Instructions in the DURASE system*. SAMOS-IX, 2009.

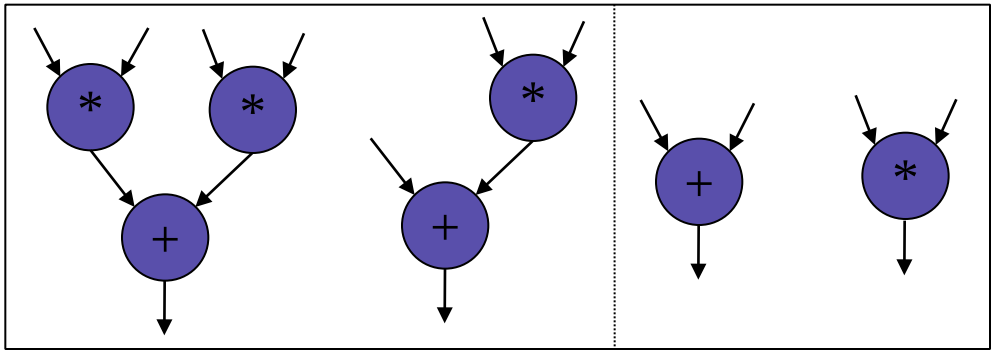
- Introduction
- Constraint programming
- Pattern generation
- Pattern selection and scheduling
- Instruction reordering
- Results
- Conclusion

Pattern selection and scheduling – problem

Subject graph



Candidates patterns

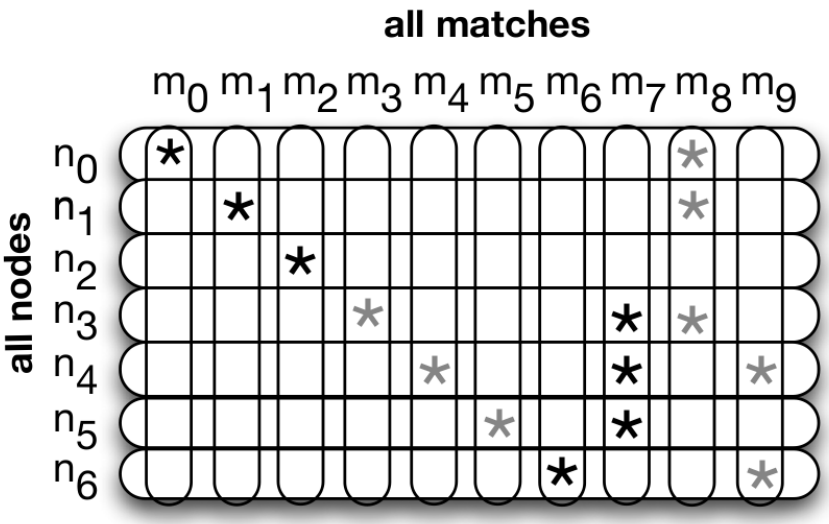


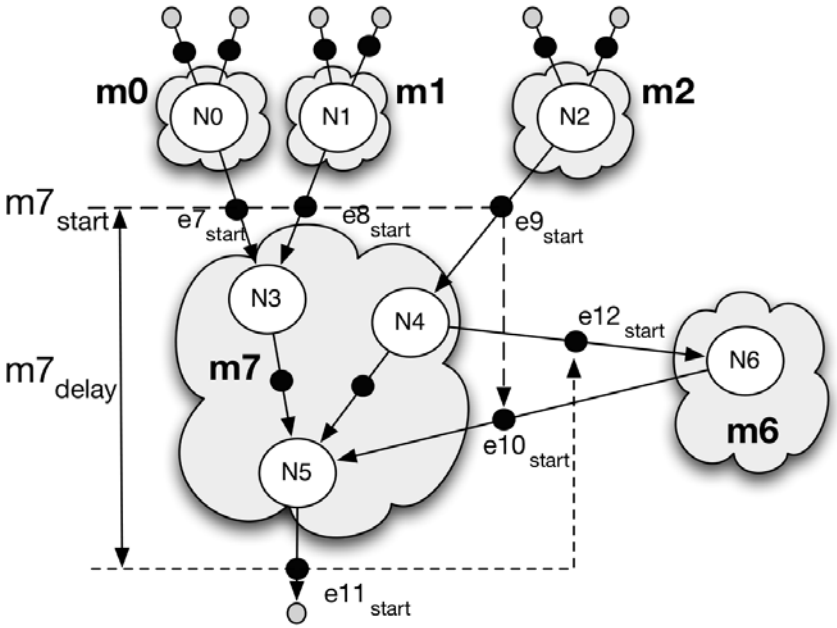
P1 P2 P3 P4

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
n1	X		X				X					
n2	X			X				X				
n3	X		X	X							X	
n4		X			X				X			
n5		X				X				X		
n6		X			X	X						X

P1 P2 P3 P4

Pattern selection and graph covering





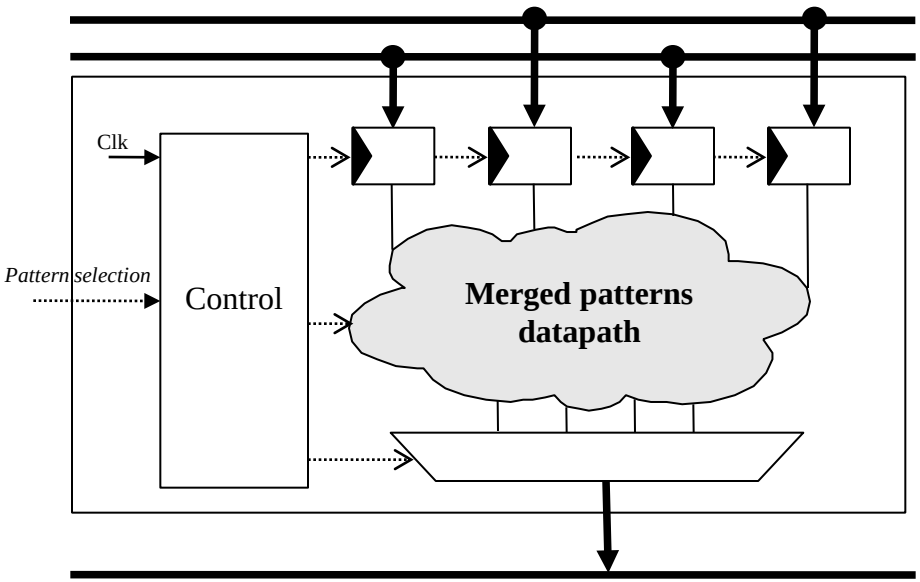
$$\forall n \in N : \sum_{m \in matches_m} m_{sel} = 1$$

$$ExecutionTime = \sum_{m \in M} m_{sel} \cdot m_{delay}$$

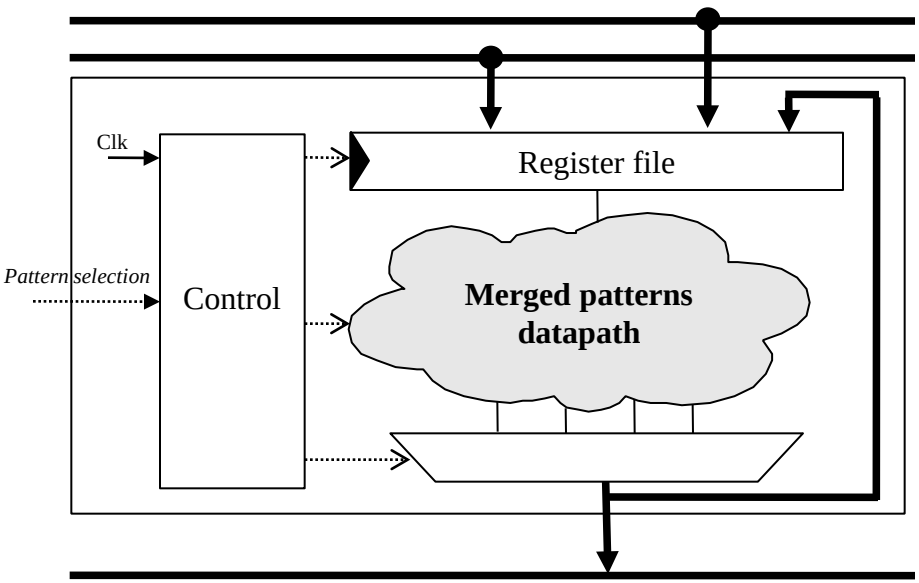
$$m_{sel} = 1 \Rightarrow \forall_{e \in m_{in}} : e_{start} = m_{start} \wedge \forall_{e \in m_{out}} : m_{start} + m_{delay} \leq e_{start}$$

Pattern selection and scheduling : 2 architecture models

Model A

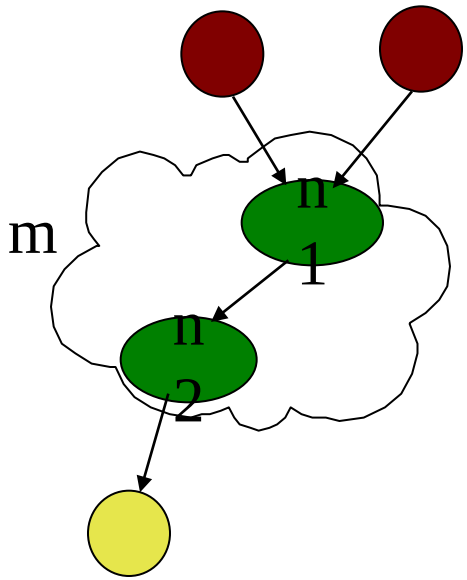


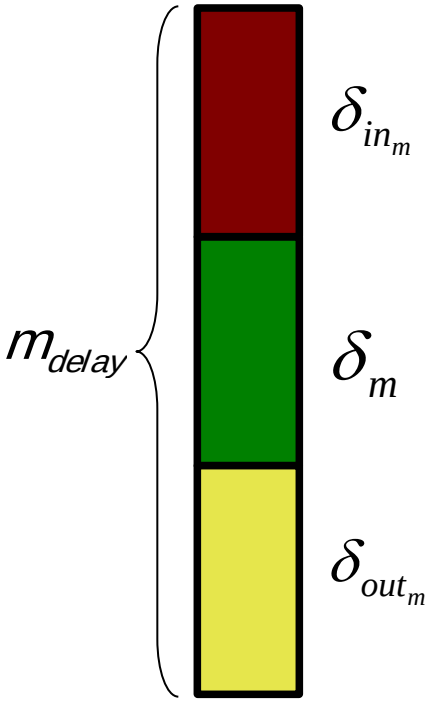
Model B



Pattern selection and scheduling – match execution time modeling

$$m_{delay} = \delta_{in_m} + \delta_m + \delta_{out_m}$$



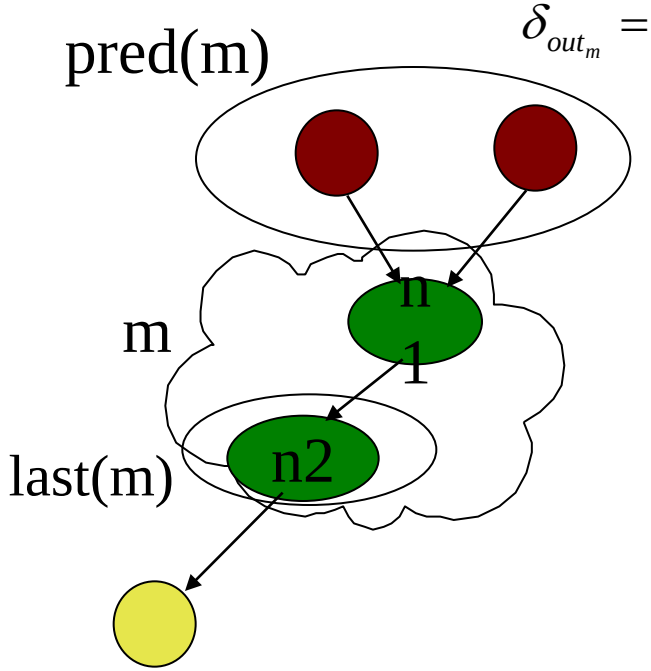


Pattern selection and scheduling – architecture model A

$$m_{delay} = \delta_{in_m} + \delta_m + \delta_{out_m}$$

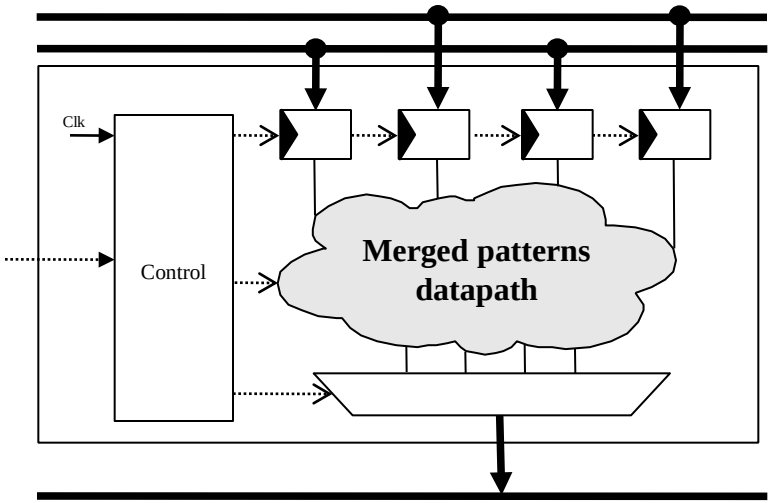
$$\delta_{in_m} = \left\lceil \frac{|pred(m)|}{in\_PerCycle} \right\rceil - 1$$

$$\delta_{out_m} = \left\lceil \frac{|last(m)|}{out\_PerCycle} \right\rceil - 1$$



Example for a Nios2 :

|pred(m)| = 2
in_PerCycle = 2
|last(m)| = 1
out_PerCycle = 1
$\delta_m = 1$
$\Rightarrow m_{delay} = 1$



Pattern selection and scheduling – architecture model B

$$m_{delay} = \delta_{in_m} + \delta_m + \delta_{out_m}$$

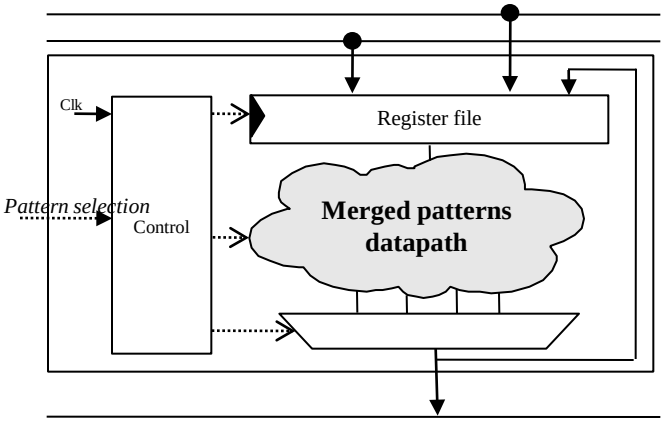
$$\delta_{in_m} = \left\lceil \frac{IN}{in\_PerCycle} \right\rceil - 1$$

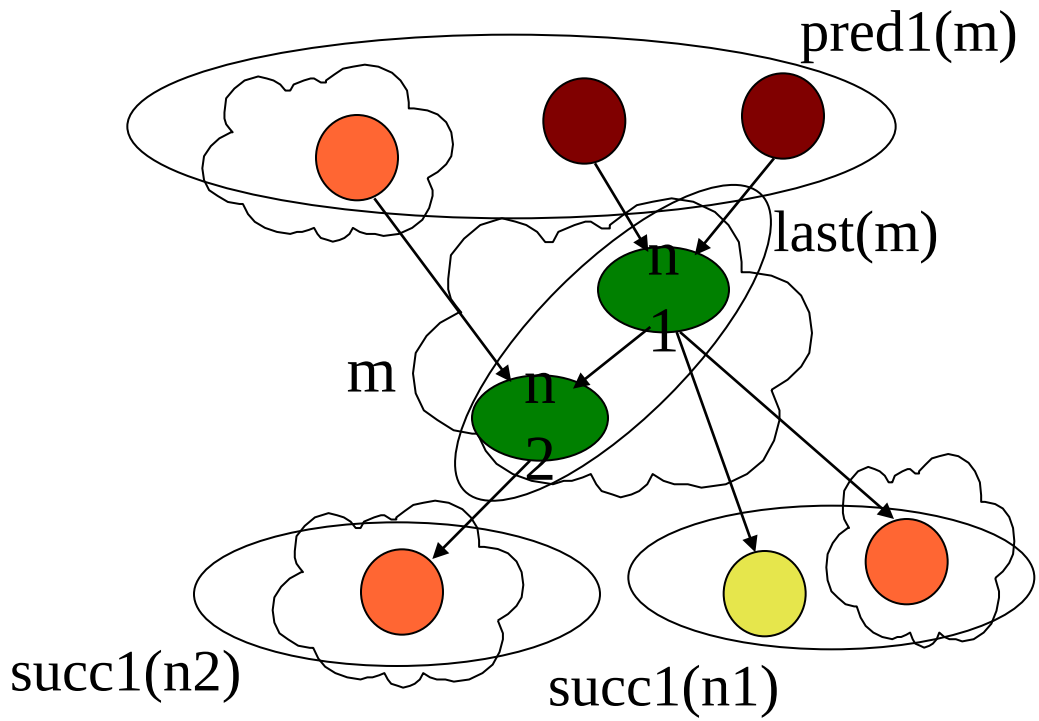
$$IN = \sum_{n \in pred1(m)} n_{sel}$$

$$\delta_{out_m} = \left\lceil \frac{OUT}{out\_PerCycle} \right\rceil - 1$$

$$\forall n \in last(m): \sum_{m \in succ1(n)} m_{sel} > 0 \Leftrightarrow B_n = 1$$

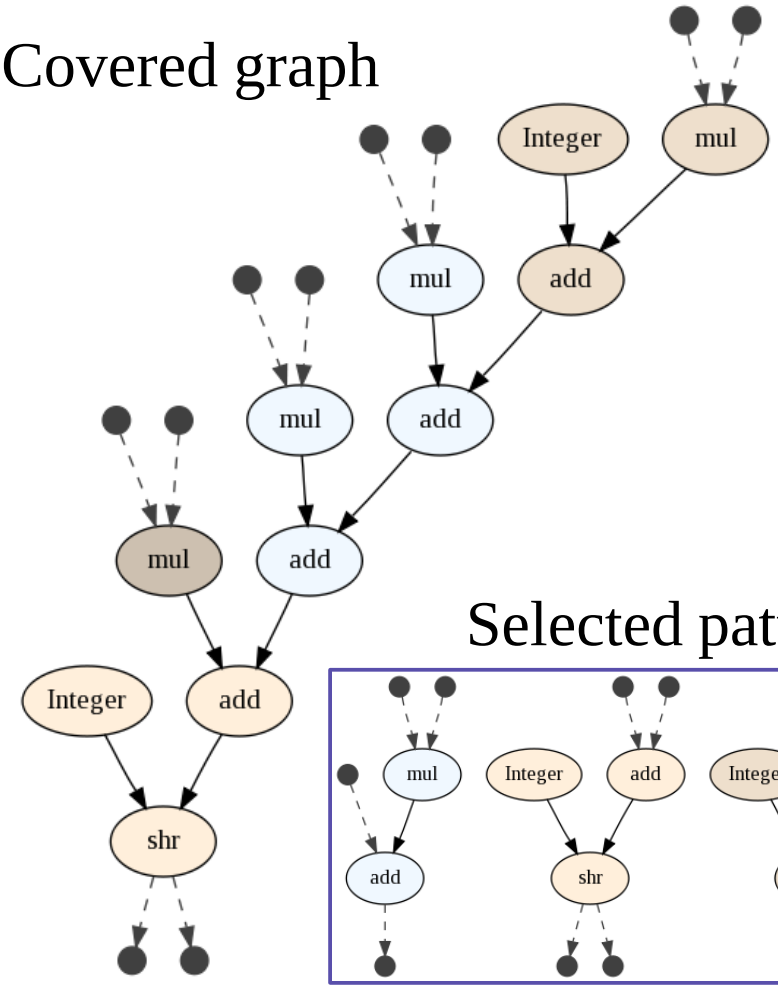
$$OUT = \sum_{n \in last(m)} B_n$$



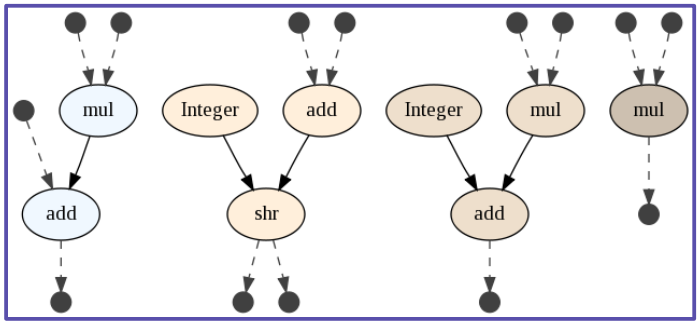


Pattern selection and scheduling – 4-tap FIR filter example

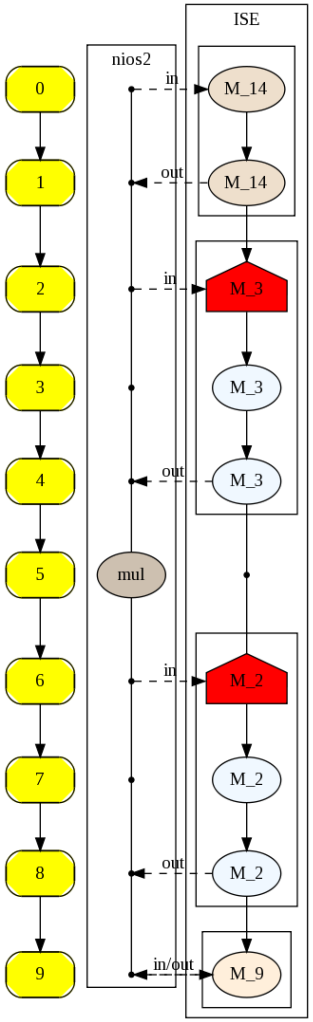
Covered graph



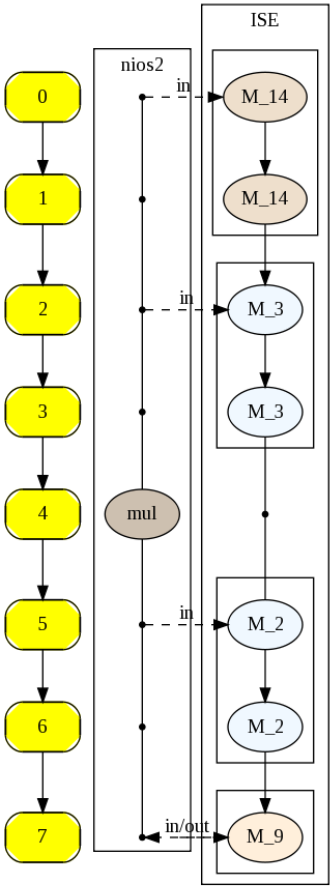
Selected patterns



Model A



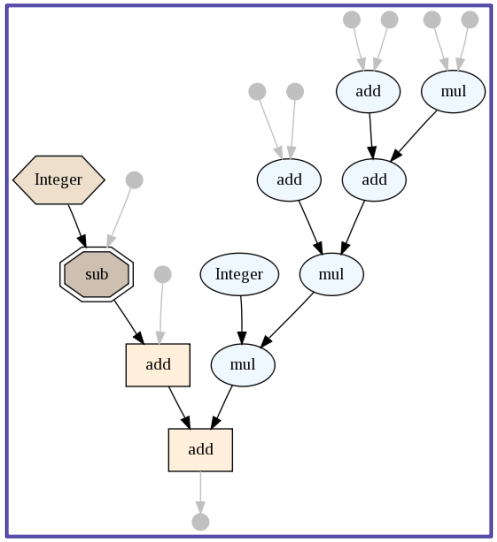
Model B



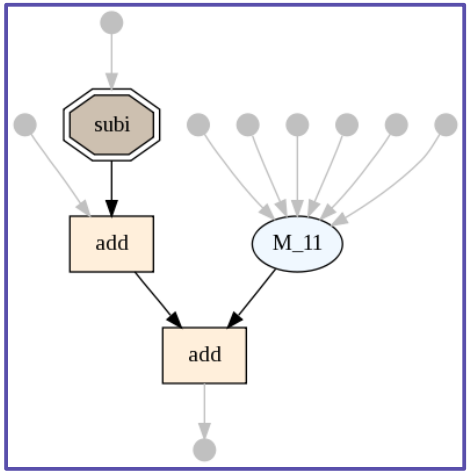
- Introduction
- Constraint programming
- Pattern generation
- Pattern selection and scheduling
- Instruction reordering
- Results
- Conclusion

Instruction reordering

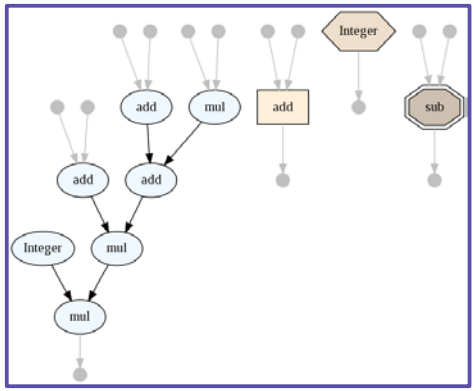
Covered graph



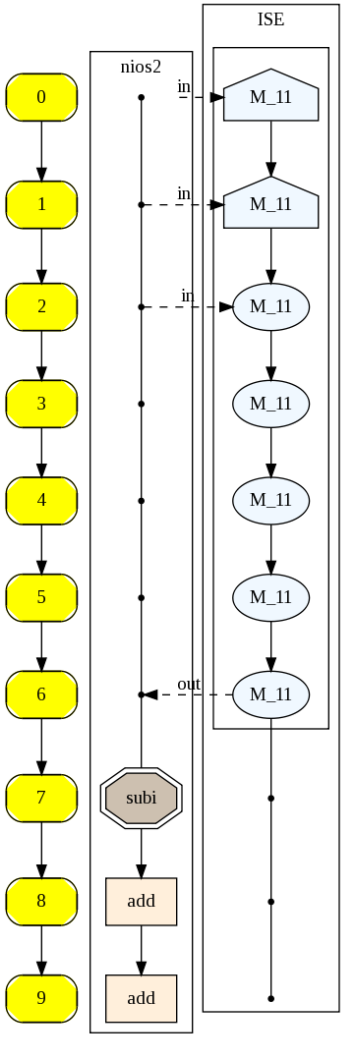
Collapsed graph



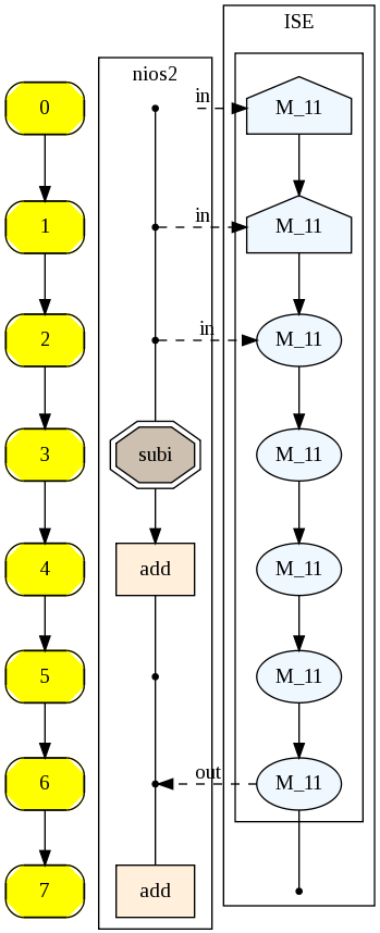
Selected patterns



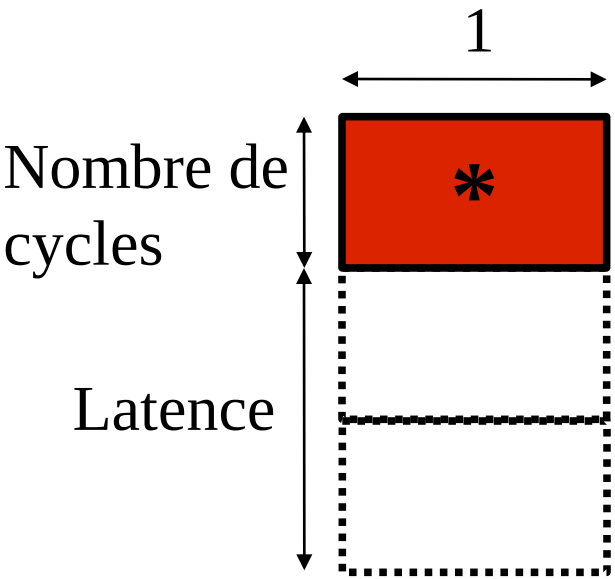
Without reordering

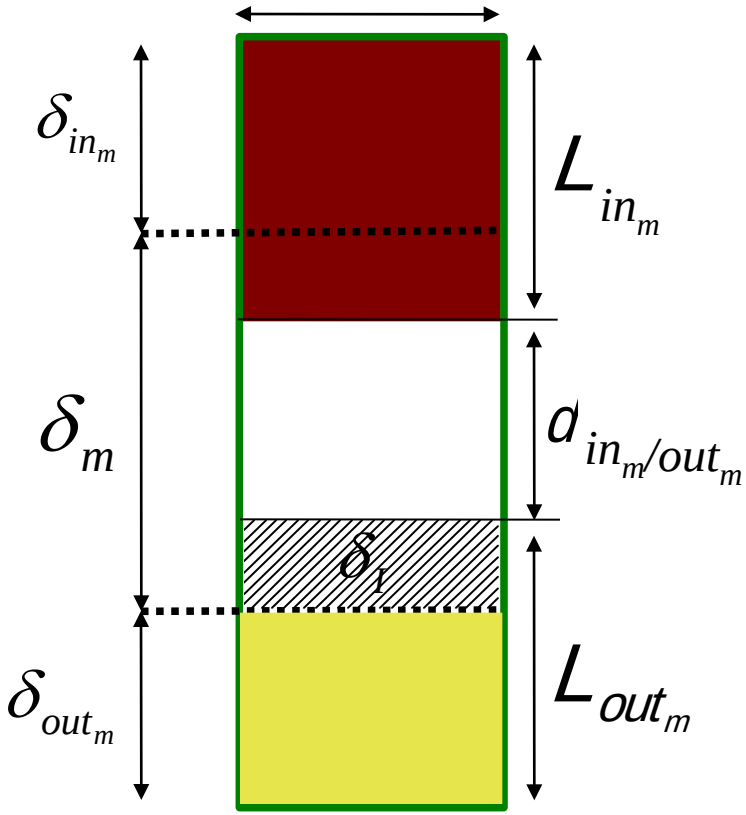


With reordering



Instruction reordering





- Introduction
- Constraint programming
- Pattern generation
- Pattern selection and scheduling
- Instruction reordering
- Results
- Conclusion

Results

Benchmarks	Nodes	cycles	2 in / 1 out										4 in / 2 out									
						model A				model B						model A				model B		
			coef	identified	selected	coverage	cycles	speedup	selected	coverage	cycles	speedup	coef	identified	selected	coverage	cycles	speedup	selected	coverage	cycles	speedup
JPEG Write BMP Header	34	34	0	6	2	82%	14	2.42	2	82%	14	2.42	0	66	2	88%	12	2.83	3	88%	12	2.83
JPEG Smooth Downsample	66	78	0	5	2	19%	68	1.14	2	19%	68	1.14	0	49	4	95%	44	1.77	4	100%	35	2.22
JPEG IDCT	250	302	0.5	28	10	76%	214	1.41	10	76%	134	2.25	0.5	254	13	83%	141	2.36	15	89%	112	2.69
EPIC Collapse	274	287	0	11	8	68%	165	1.74	8	68%	165	1.74	0	111	11	71%	156	1.83	14	71%	159	1.8
BLOWFISH encrypt	201	169	0.5	11	3	74%	90	1.87	3	74%	90	1.87	0	153	8	90%	81	2.08	7	92%	73	2.31
SHA transform	53	57	0	5	3	64%	28	2.03	3	64%	28	2.03	0	48	8	98%	22	2.59	6	95%	17	3.35
MESA invert matrix	152	334	0.5	2	2	10%	320	1.04	2	10%	320	1.04	0.5	53	9	65%	262	1.27	9	65%	243	1.37
FIR unrolled	67	131	0	3	2	9%	126	1.04	2	9%	126	1.04	1	10	2	94%	98	1.30	2	97%	67	1.95
FFT	10	18	0	0	-	-	-	-	-	-	-	-	0	12	2	60%	10	1.80	2	60%	10	1.80
Average						50%		1.5		50%		1.7				83%		2		84%		2.3

CP max = 15000
NN max = 5

- Introduction
- Constraint programming
- Pattern generation
- Pattern selection and scheduling
- Instruction reordering
- Results
- Conclusion

Conclusion

- Constraints make it possible to explore solutions that are difficult to examine using specific algorithms.
- Constraints provide flexibility to the defining of the different conditions.
- (Sub-)graph isomorphism constraints offer an easy way of defining design problems.
- Experimental results are very encouraging.

Sélection automatique d'instructions et ordonnancement d'applications basés sur la programmation par contraintes

Merci!