



12th EUROMICRO CONFERENCE on DIGITAL SYSTEM DESIGN
Architectures, Methods and Tools

Patras, Greece. 27-29 August 2009



IRISA

UNE UNITÉ DE RECHERCHE À LA POINTE DES SCIENCES
ET DES TECHNOLOGIES DE L'INFORMATION
ET DE LA COMMUNICATION



Architecture-Driven Synthesis of Reconfigurable Cells

C. Wolinski

*University of Rennes 1
IRISA, INRIA France*



K. Kuchcinski

*Dept. of Computer Science
Lund University, Sweden*



LUNDS
UNIVERSITET

E. Raffin

*Thomson R&D
Rennes, France*



F. Charot

*INRIA
Rennes, France*



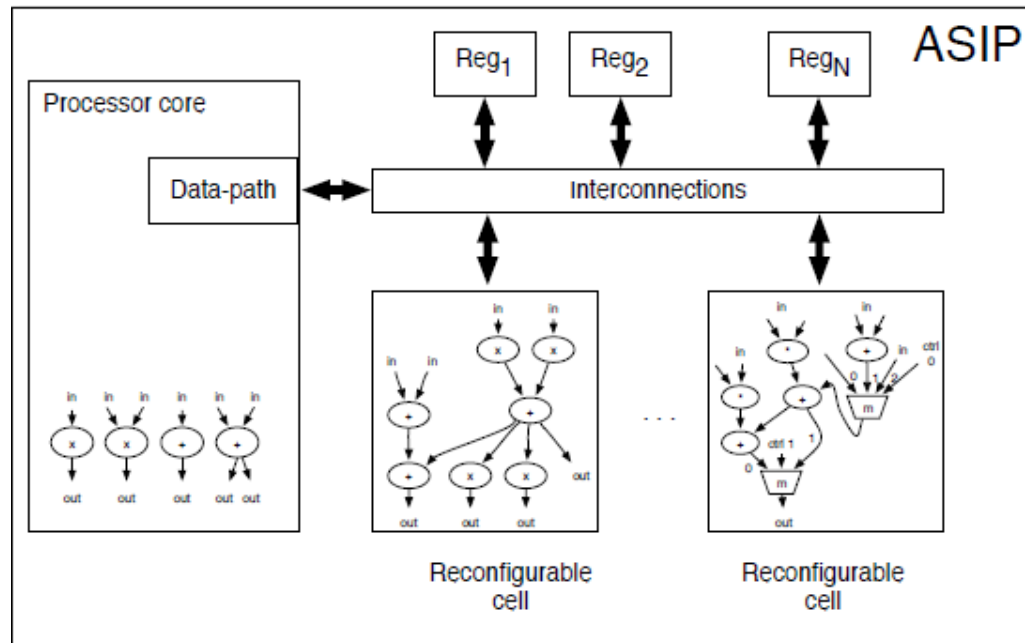
Outline

- Context and motivations
- Introduction to the pattern merging problem
- Related work
- Our contribution
- Introduction to Constraint Programming
- Pattern merging algorithm overview
- Constraint models
- Results



General Context

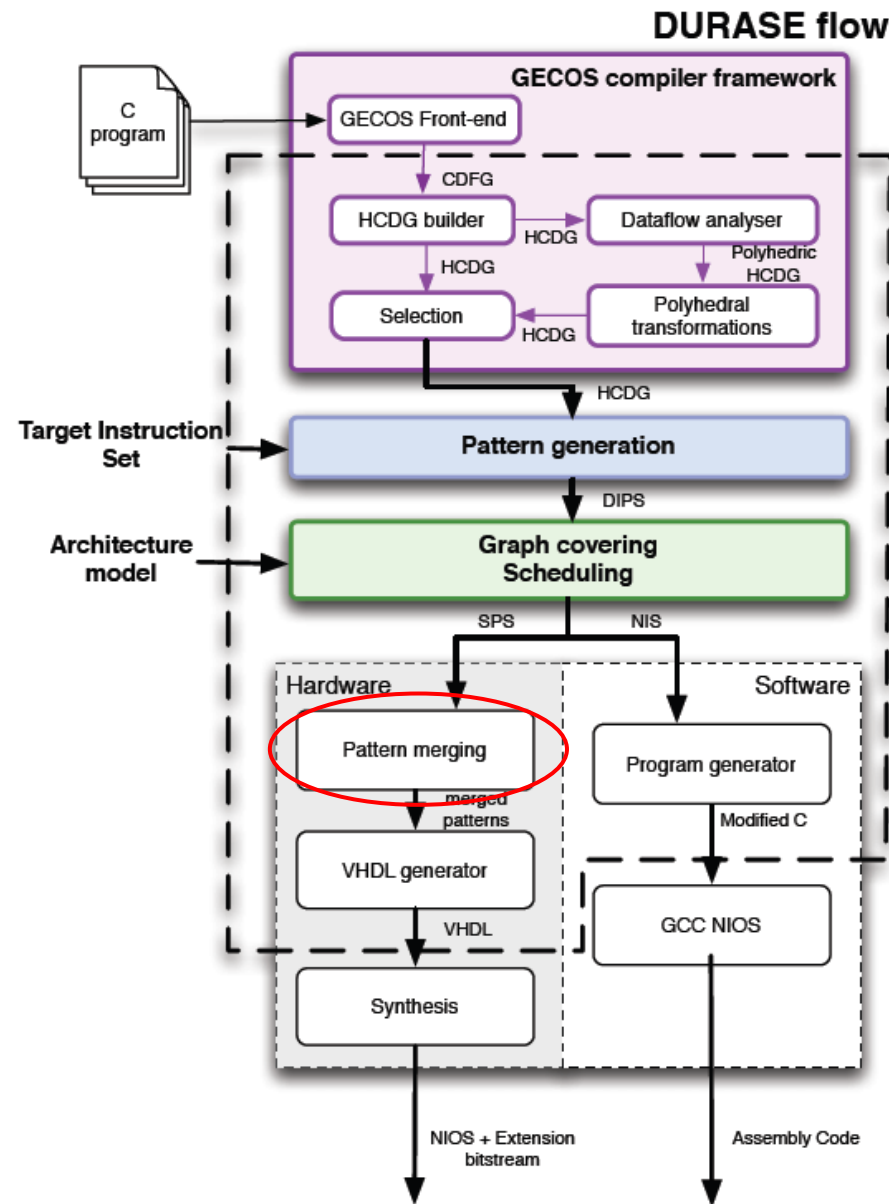
- What: speed-up multimedia applications
- How: Application Specific Instruction-set Processor
- Processor extension = system level reconfigurable cells



Context

- DURASE:
Generic Environment for
Design and
Utilization of
Reconfigurable
Application-
Specific Processors
Extensions

This design flow is based on
“Constraint-driven methodology”

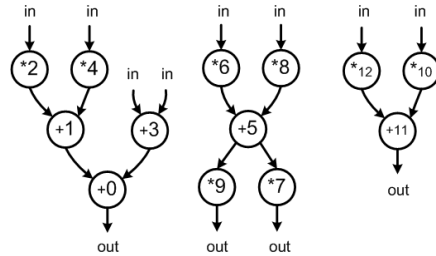


Reference : « DURASE : Generic Environment for Design and Utilization of Reconfigurable Application-Specific Processors Extensions » DATE09.

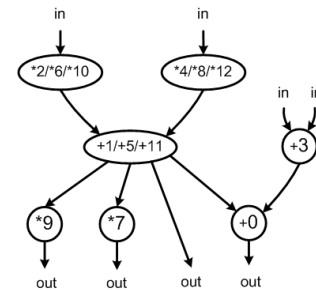
Context

Auto Regression Filter application

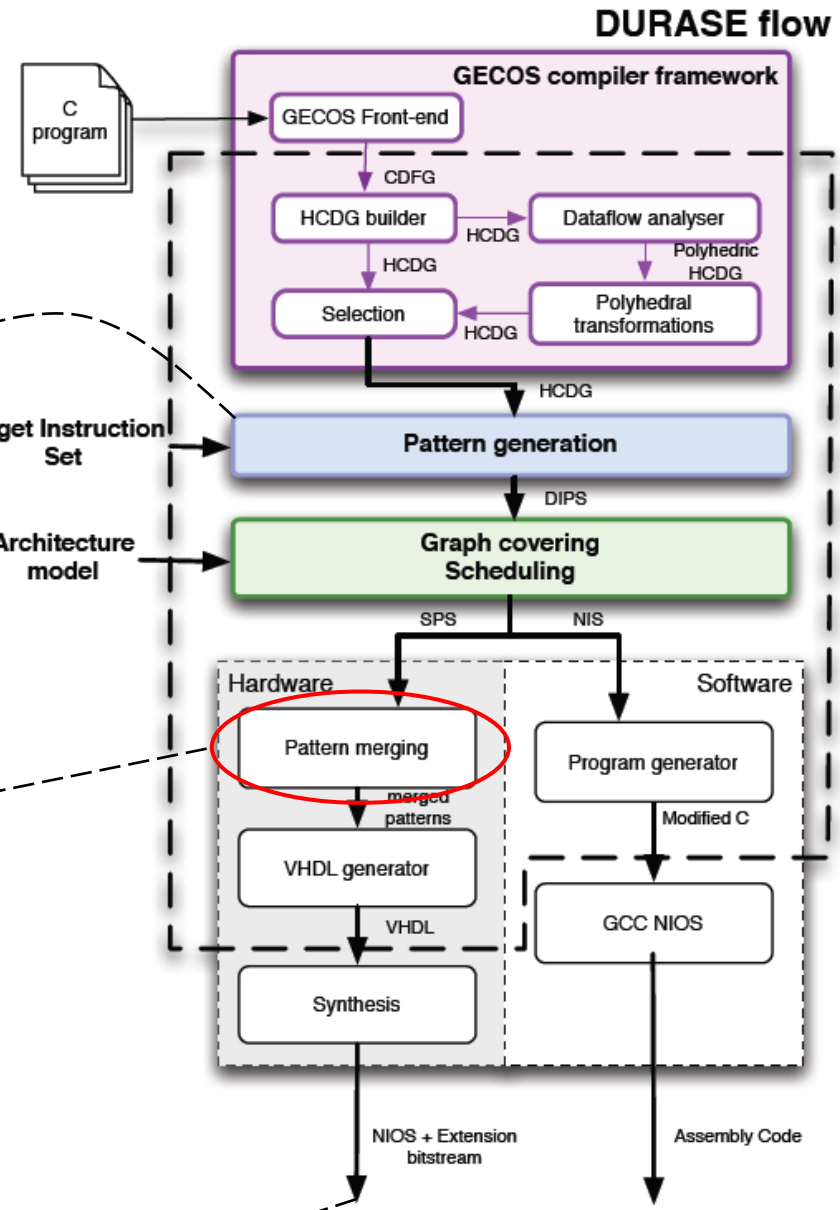
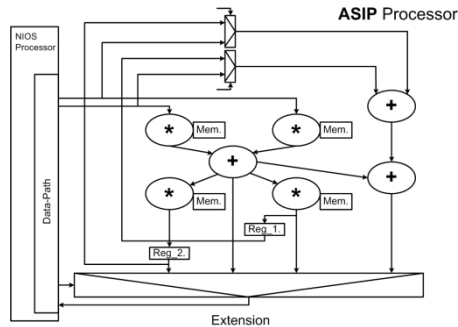
Computational patterns



Merged pattern



Reconfigurable Cell



Motivations

Synthesis of a system level reconfigurable cell in the DURASE context under technological constraint:

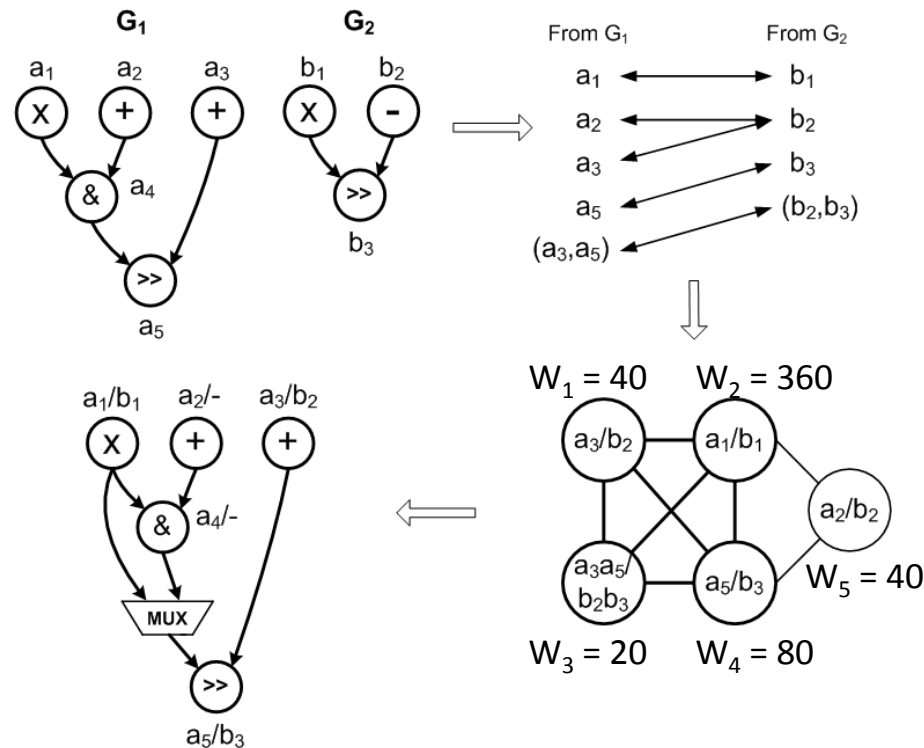
- Minimize the area of the reconfigurable cell.
- Add the possibility to ensure a certain frequency
=> Not increase the Critical Path of the extended processor core.



Related Work

- N. Moreano (2005)

- Merge iteratively two patterns by:
 - Making node and edge matches and Compatibility Graph (CG)
 - Reconstructing the temporary merged pattern
- At each iteration:
 - Solution = Maximum Weighted Clique in the CG in polynomial time
 - The weight = Area reduction of the merged pattern



- ⇒ Moreano's heuristic is among the best available suboptimal algorithm available for DataPath Merging.
- ⇒ It solves the global problem in polynomial time.

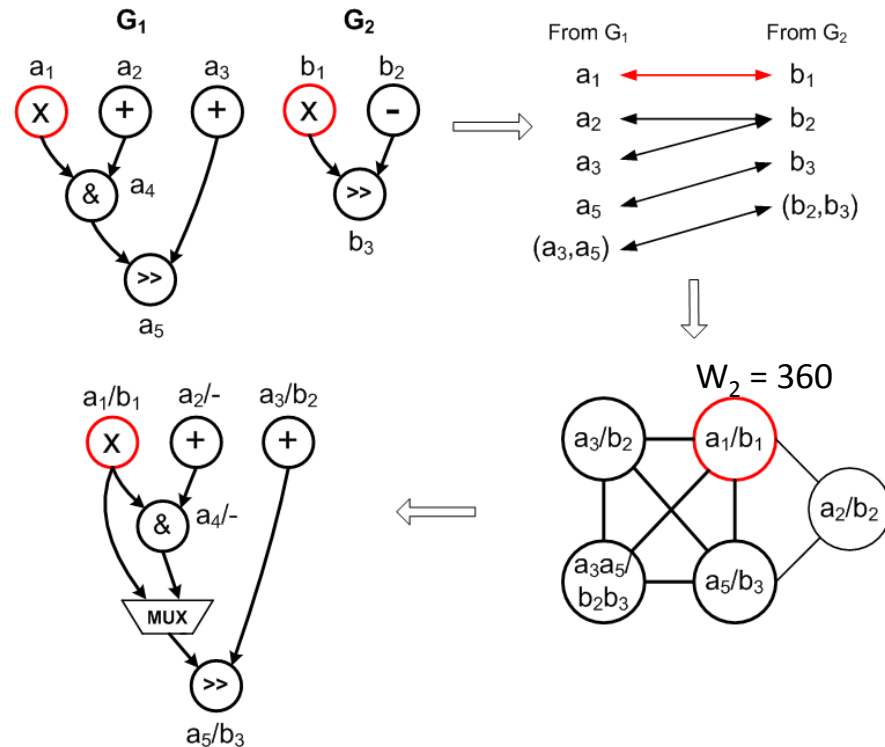


Related Work

- N. Moreano (2005)

- Merge iteratively two patterns by:

- Making node and edge matches and Compatibility Graph (CG)
 - Reconstructing the temporary merged pattern



- At each iteration:

- Solution = Maximum Weighted Clique in the CG in polynomial time
 - The weight = Area reduction of the merged pattern

- ⇒ Moreano's heuristic is among the best available suboptimal algorithm available for DataPath Merging.
- ⇒ It solves the global problem in polynomial time.



Related Work

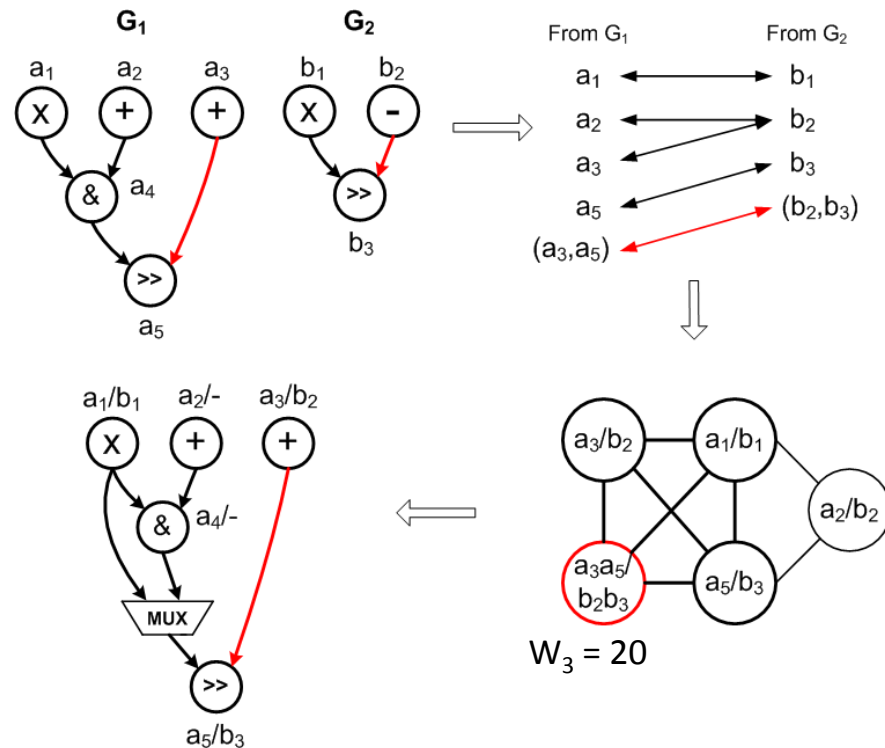
- N. Moreano (2005)

- Merge iteratively two patterns by:

- Making node and edge matches and Compatibility Graph (CG)
 - Reconstructing the temporary merged pattern

- At each iteration:

- Solution = Maximum Weighted Clique in the CG in polynomial time
 - The weight = Area reduction of the merged pattern



⇒ Moreano's heuristic is among the best available suboptimal algorithm available for DataPath Merging.

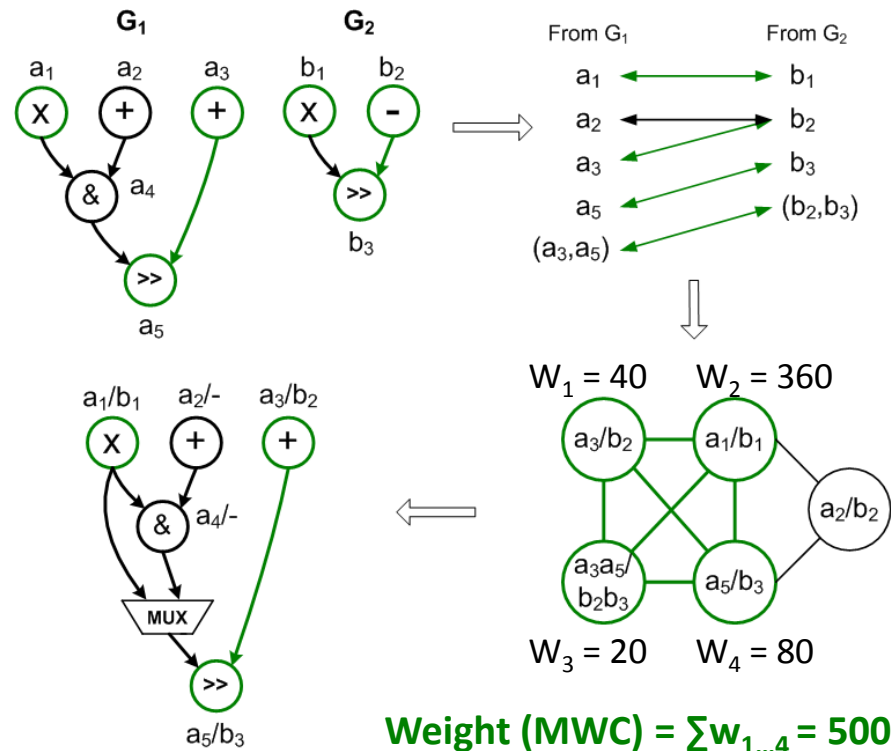
⇒ It solves the global problem in polynomial time.



Related Work

- N. Moreano (2005)

- Merge iteratively two patterns by:
 - Making node and edge matches and Compatibility Graph (CG)
 - Reconstructing the temporary merged pattern
- At each iteration:
 - Solution = Maximum Weighted Clique in the CG in polynomial time
 - The weight = Area reduction of the merged pattern

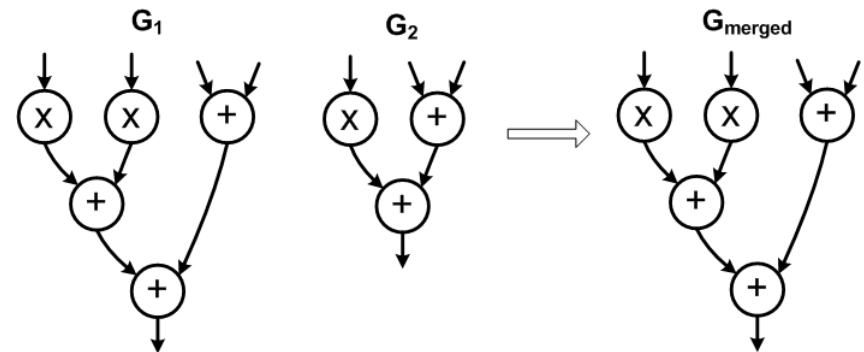


- ⇒ Moreano's heuristic is among the best available suboptimal algorithm available for DataPath Merging.
- ⇒ It solves the global problem in polynomial time.



Related Work

- M. Corazao (2004)
 - HLS for data-path-intensive ASIC design
 - Performance optimization using template mapping
 - Key of their algorithm:
 - Notion of “*bypassability*” (example: Node “+1” in the figure)

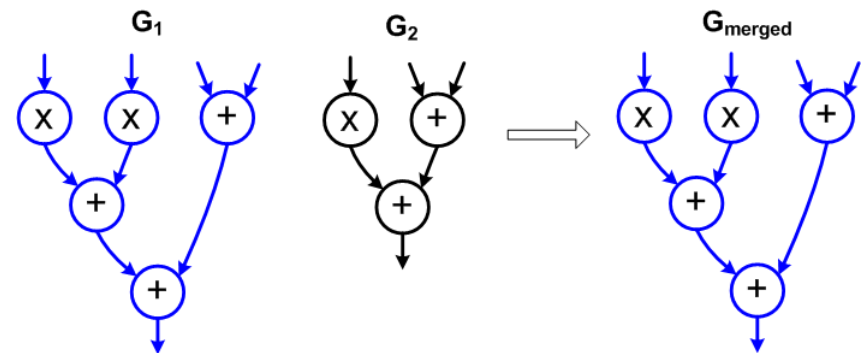


⇒ Interesting method to reduce the number of Mux in the merged pattern.



Related Work

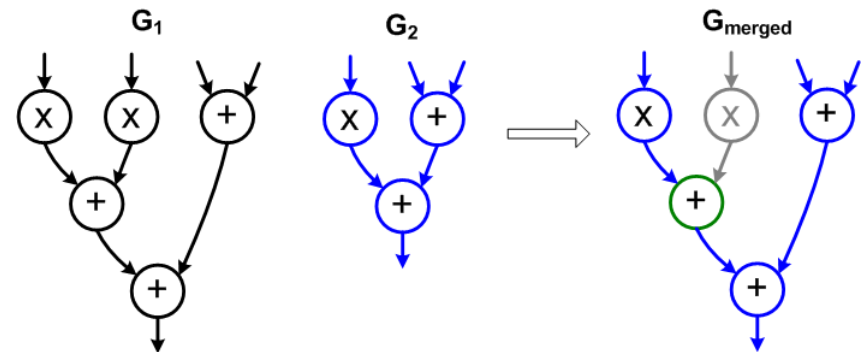
- M. Corazao (2004)
 - HLS for data-path-intensive ASIC design
 - Performance optimization using template mapping
 - Key of their algorithm:
 - Notion of “*bypassability*” (example: Node “+1” in the figure)



⇒ Interesting method to reduce the number of Mux in the merged pattern

Related Work

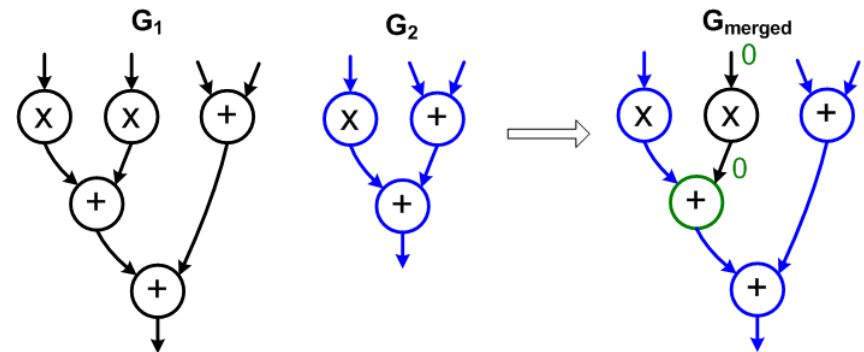
- M. Corazao (2004)
 - HLS for data-path-intensive ASIC design
 - Performance optimization using template mapping
 - Key of their algorithm:
 - Notion of “*bypassability*” (example: Node “+1” in the figure)



⇒ Interesting method to reduce the number of Mux in the merged pattern

Related Work

- M. Corazao (2004)
 - HLS for data-path-intensive ASIC design
 - Performance optimization using template mapping
 - Key of their algorithm:
 - Notion of “*bypassability*” (example: Node “+1” in the figure)



⇒ Interesting method to reduce the number of Mux in the merged pattern

Our Contribution

- Make patterns merging with/without increasing the critical path under different conditions as:
 - Maximum number of MUX on the critical path
 - Maximum number of bypassed nodes on a path
(generalization of the bypassability notion)
- Combine the constraints modeling these conditions and solve the entire problem while minimizing the area of the merged pattern.



Our Contribution

- Make patterns merging with/without increasing the critical path under different conditions as:
 - Maximum number of MUX on the critical path
 - Maximum number of bypassed nodes on a path
(generalization of the bypassability notion)
- Combine the constraints modeling these conditions and solve the entire problem while minimizing the area of the merged pattern.
- The problem has been modelised using constraint programming and solved by JaCoP solver.



JaCoP web site: <http://jacop.osolpro.com>



Constraint programming

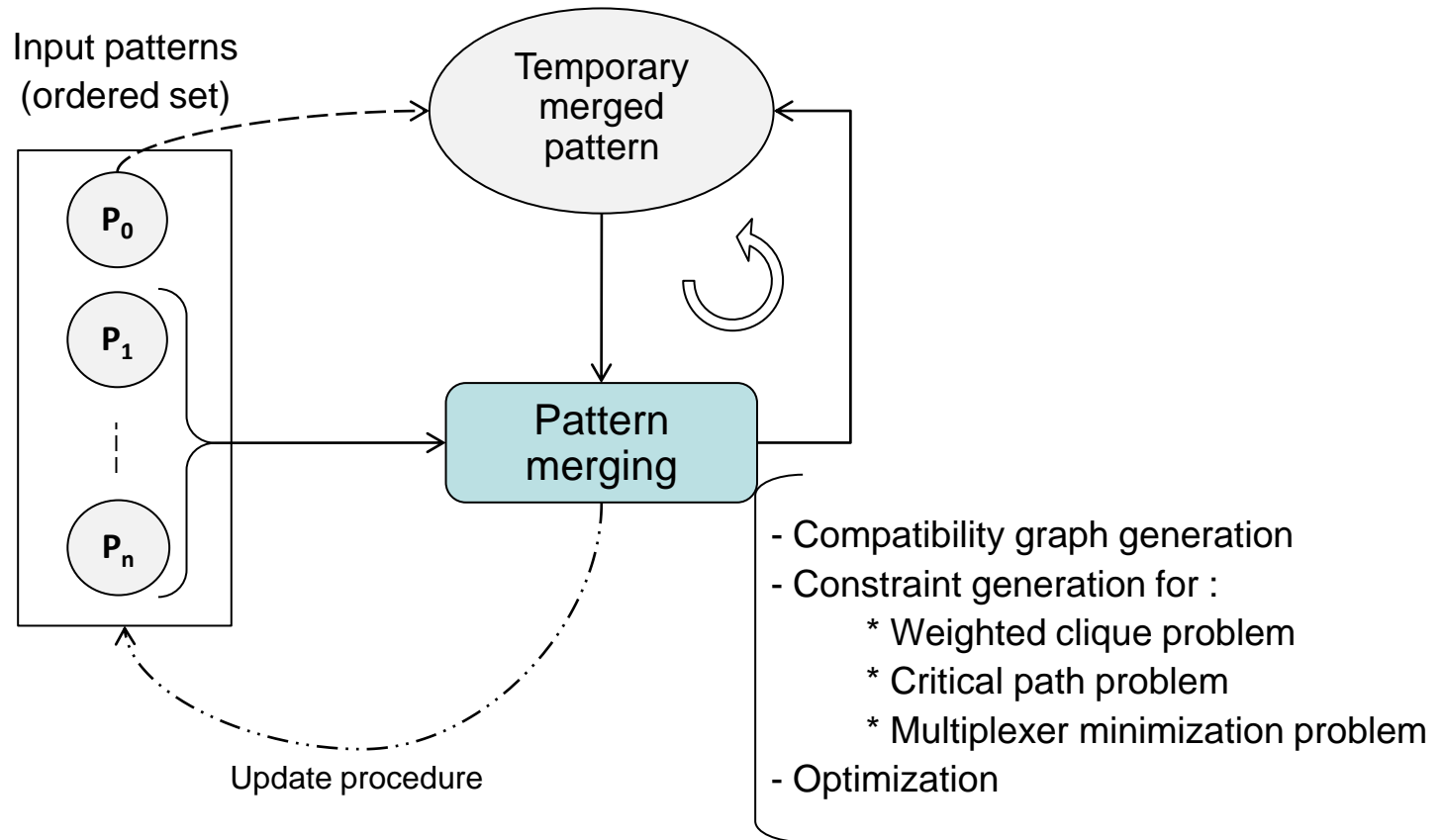
- Formally, a Constraint Satisfaction Problem (CSP) is defined by a 3-tuple (**V**;**D**;**C**)

Where:

- **V** = $X_1; X_2; \dots; X_n$ is a finite set of finite domain variables (FDV's),
- **D** = $D_1; D_2; \dots; D_n$ is a 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.



Algorithm Overview



Compatibility Graph

- Undirected graph
- Node types

Type

- Regular node
- Edge node
- **Path node**

Example

Add / Add

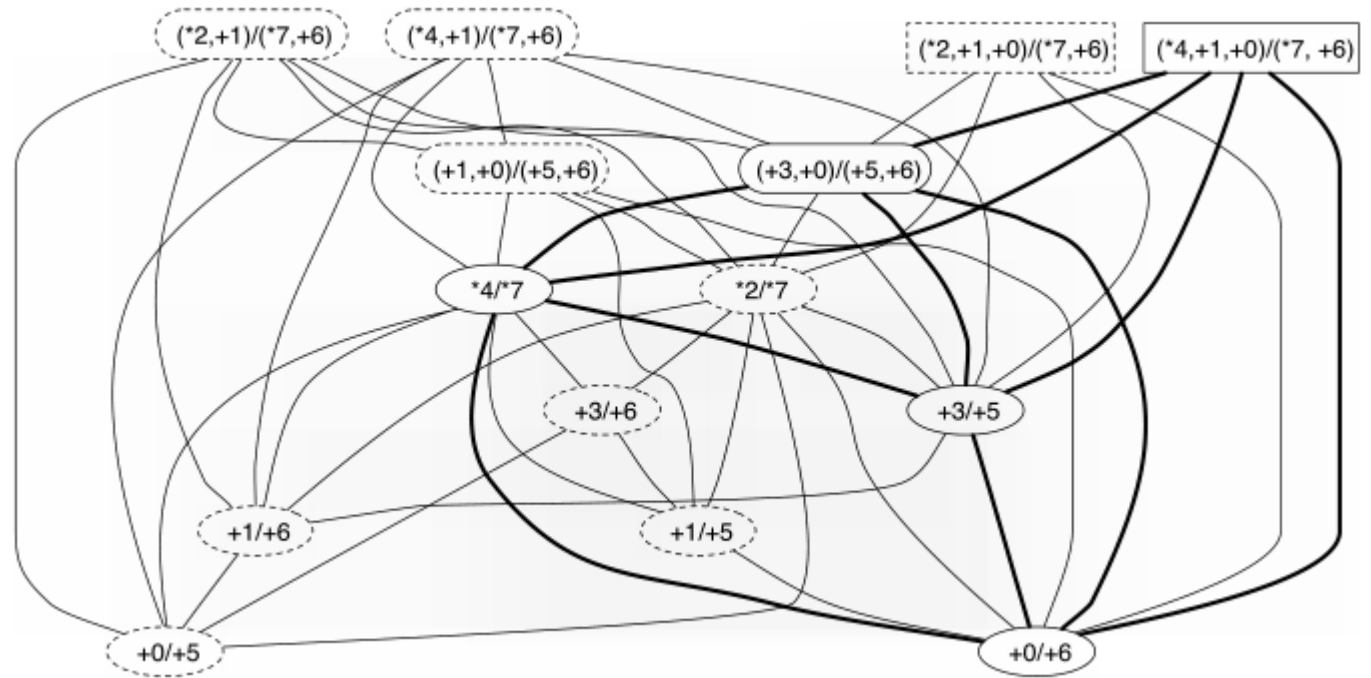
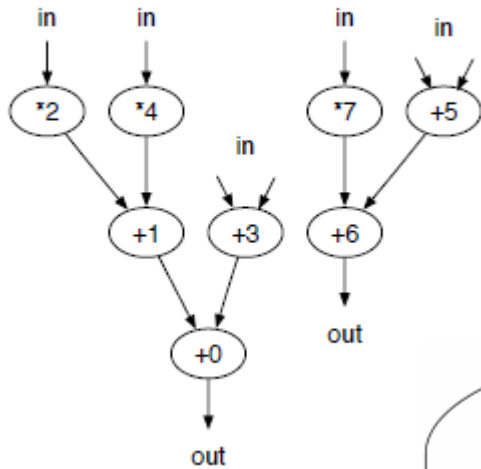
Add → Mul / Add → Mul

Add → ... → Mul / Add → Mul

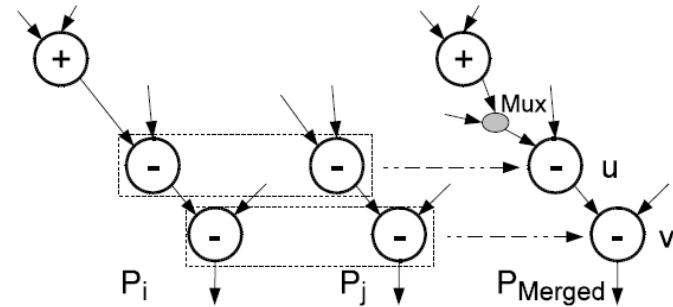
- An edge defines “*mapping compatibility*” between two nodes

	(u'_i/u'_j)	$(u'_i, v'_i)/(u'_j, v'_j)$	$(u'_i, w'_i, \dots, v'_i)/(u'_j, v'_j)$
(u_i/u_j)	$u_i \neq u'_i \wedge u_j \neq u'_j$	$(u_i = u'_i \wedge u_j = u'_j) \vee$ $(u_i = v'_i \wedge u_j = v'_j) \vee$ $(u_i \neq u'_i \wedge u_j \neq u'_j \wedge u_i \neq v'_i \wedge u_j \neq v'_j)$	$(u_i = u'_i \wedge u_j = u'_j) \vee$ $(u_i = v'_i \wedge u_j = v'_j) \vee$ $[u_i \neq u'_i \wedge u_j \neq u'_j \wedge (\forall n \in (u'_i, w'_i, \dots, v'_i) n \neq u_i)]$
$(u_i, v_i)/(u_j, v_j)$		$(u_i \neq u'_i \wedge u_j \neq u'_j) \vee$ $(v_i \neq v'_i \wedge v_j \neq v'_j)$	$[(u_i \neq u'_i \wedge u_j \neq u'_j) \vee (v_i \neq v'_i \wedge v_j \neq v'_j)] \wedge$ $(\forall n \in (u'_i, w'_i, \dots, v'_i) (n \neq u_i) \wedge n \neq v_i)$
$(u_i, w_i, \dots, v_i)/(u_j, v_j)$			$[(u_i \neq u'_i \wedge u_j \neq u'_j) \vee (v_i \neq v'_i \wedge v_j \neq v'_j)] \wedge$ $(\forall n \in (u'_i, w'_i, \dots, v'_i) (n \neq u_i) \wedge n \neq v_i) \wedge$ $(\forall n \in (u_i, w_i, \dots, v_i) (n \neq u'_i) \wedge n \neq v'_i)$

Compatibility Graph



Weights in Compatibility Graphs



- Regular node weight

$$\text{Weight } (u_i / u_j) = \text{Area}(u) - \text{Area}(\text{Mux}) \quad (1)$$

- Edge node weight

$$\text{Weight } ((u_i, v_i) / (u_j, v_j)) = \text{Area}(\text{Mux}) \quad (2)$$

- Path node weight

$$\text{Weight } ((u_i, w_i, \dots, v_i) / (u_j, v_j)) = \text{Area}(\text{Mux}) - \text{Area}((u_i, w_i, \dots, v_i)) \quad (3)$$



Weighted Clique Model

- Compatibility Graph (CG) model

CG = (V_c, E_c) where: V_c is a set of vertices
E_c ⊆ V_c × V_c is a set of edges

- Clique model

Each node $u \in V_c$ is modeled by finite domain variable $Sel_u = \{0, 1\}$.
 $Sel_u = 1$ if the node is contained in the Maximum Weighted Clique.
 Sum is the goal function that is maximized.

$$\forall (u_c, v_c) \notin E_c : Sel_{u_c} \neq 1 \vee Sel_{v_c} \neq 1 \quad (4)$$

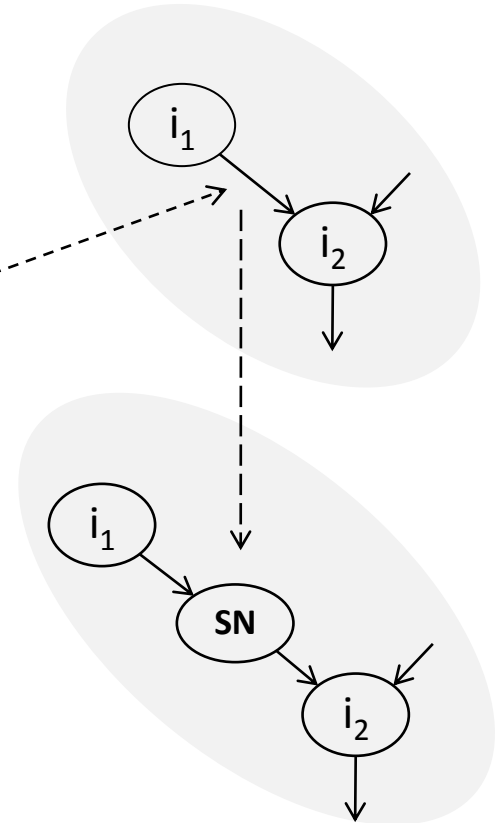
$$Sum = \sum_{u \in V_c} Sel_u \cdot weight(u) \quad (5)$$



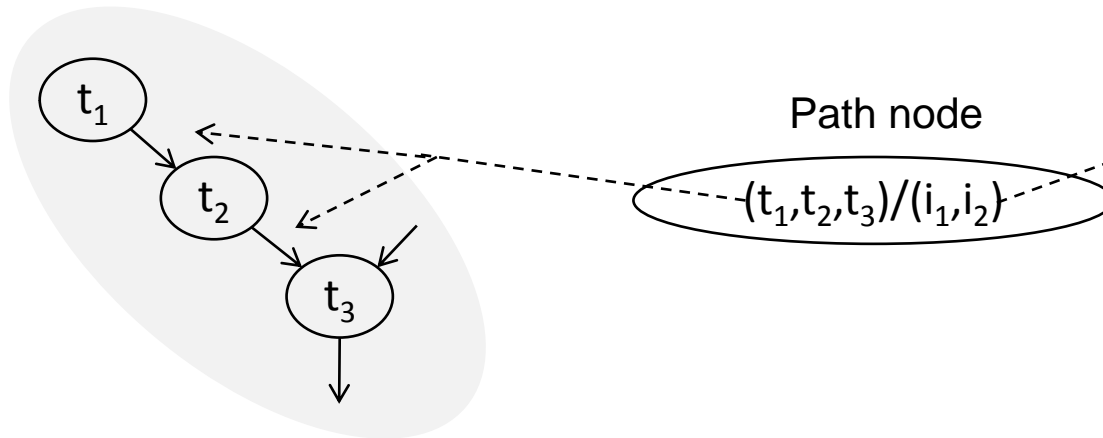
Critical Path Model

- Problematic

Original input pattern



Temporary merged pattern



– The node t_2 is bypassed in the temporary merged pattern to match with the (i_1, i_2) edge of the input pattern.

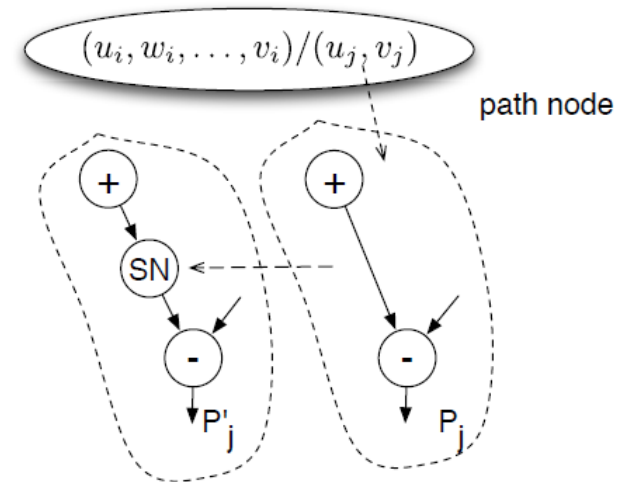
➤ The delay of the bypassed node must be taken into account if the path node is selected in the clique.

New input pattern



Critical Path Model

- Constraint modeling the critical path problem



$$CPL = \text{Max}\{\text{Latency}(P_1), \dots, \text{Latency}(P_K)\}$$

$$\forall u \in V_c, u = (u_i, w_i, \dots, v_i) / (u_j, v_j) : \quad (6)$$

$$\text{Delays}_{SN_u} = \text{Sel}_u \cdot \sum_{n \in \{u_i, w_i, \dots, v_i\}} \text{Latency}(n)$$

$$\forall (u_j, v_j) \in E'_j : \text{Start}_{u_j} + \text{Delay}_{u_j} \leq \text{Start}_{v_j} \quad (7)$$

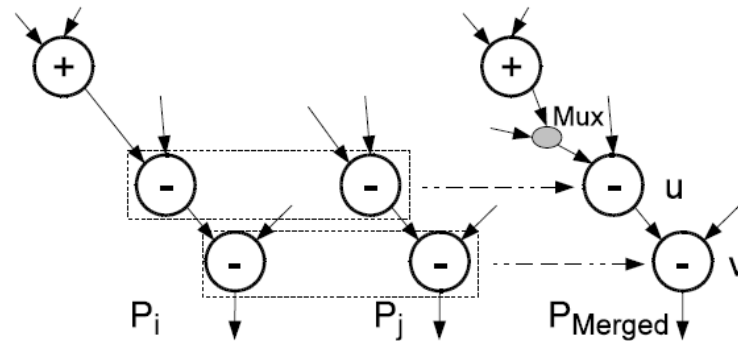
$$\forall u \in \text{ONS} : \text{Start}_u + \text{Delay}_u \leq \text{CPL} \quad (8)$$

“ONS”: Output Node Set, “CPL”: Critical Path Latency



Model with Multiplexers

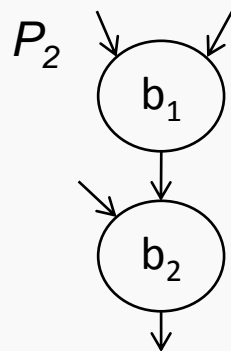
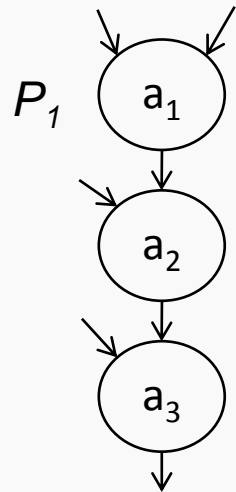
- Problematic
 - When two nodes are shared in the merged graph without sharing their inputs, multiplexers are added.
 - Objective: Modeling the condition related to the number of MUX on the critical path.



Example of multiplexer insertion after merging patterns P_i and P_j

Model with Multiplexers

- Example (1)
- Generation of the Set of Nodes on the Critical Path



$$CPL = \text{Latency}(P_1)$$

$$SNCP_1 = \{ a_1, a_2, a_3 \}$$

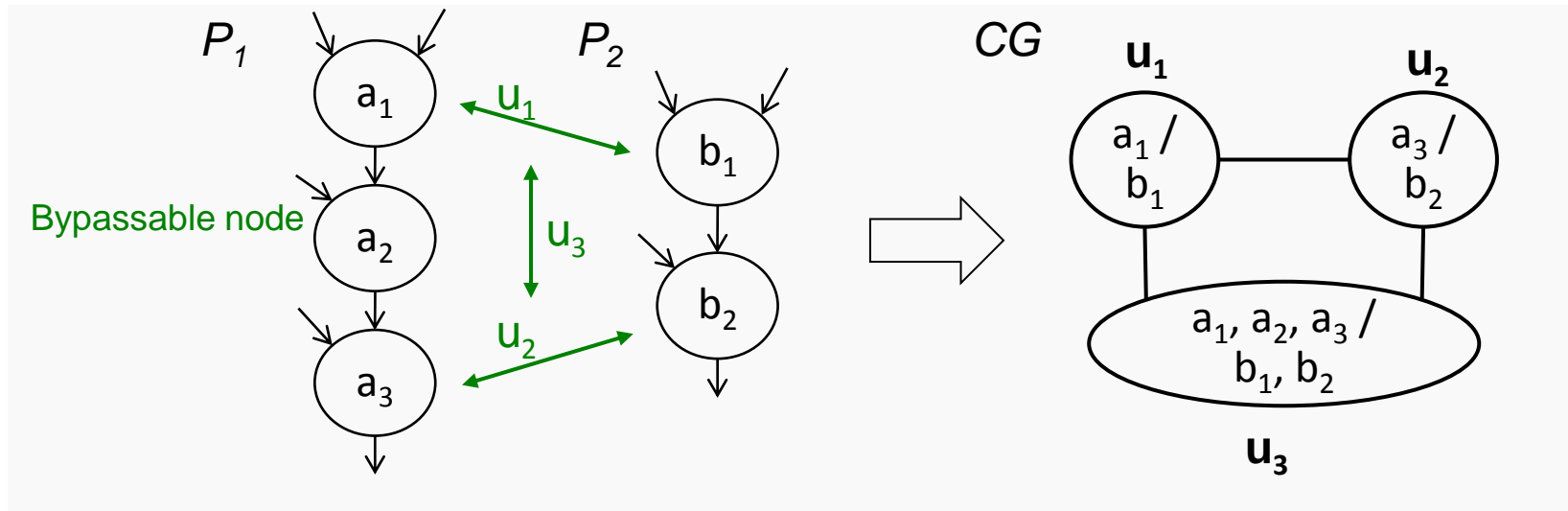
CPL: Critical Path Length

SNCP: Set of Node on Critical Path



Model with Multiplexers

- Example (2)
- Compatibility Graph generation



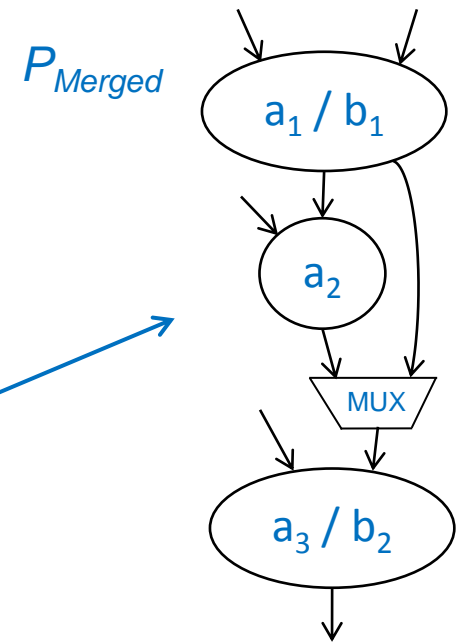
CG: Compatibility Graph



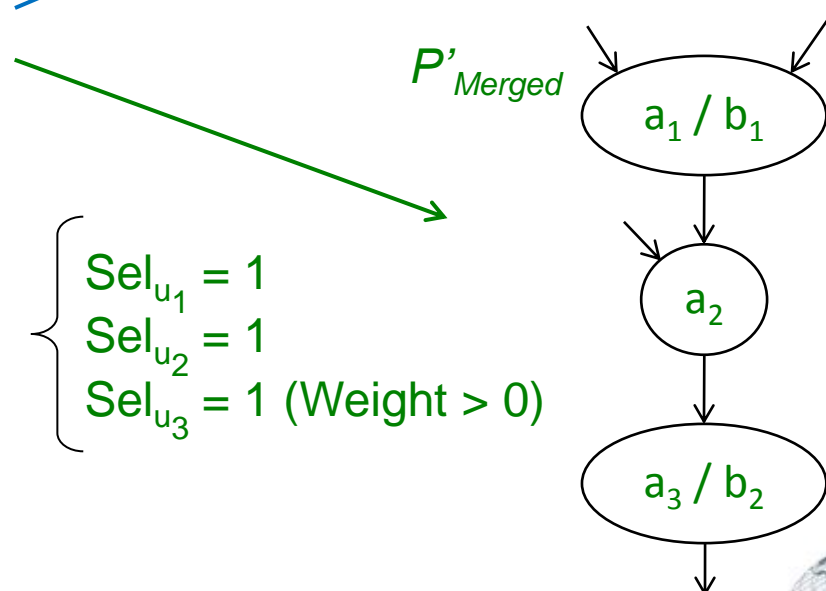
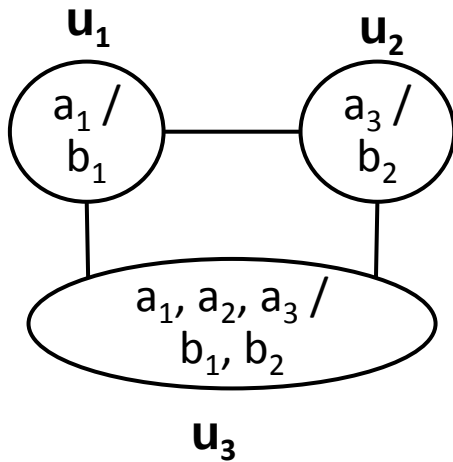
Model with Multiplexers

- Example (3)

$$\begin{cases} \text{Sel}_{u_1} = 1 \\ \text{Sel}_{u_2} = 1 \\ \text{Sel}_{u_3} = 0 \text{ (Weight < 0)} \end{cases}$$



CG



$$\begin{cases} \text{Sel}_{u_1} = 1 \\ \text{Sel}_{u_2} = 1 \\ \text{Sel}_{u_3} = 1 \text{ (Weight > 0)} \end{cases}$$



Model with Multiplexers

- Constraints modeling (1)
- Generation of the Set of Nodes on the Critical Path (SNCP).

$$CPL = \text{Max}\{\text{Latency}(P_1), \dots, \text{Latency}(P_k)\}$$

For each pattern P_i with $\text{Latency}(P_i) = CPL$ we generate $SNCP_i$ which contains all the node on the critical path P_i .



Model with Multiplexers

- Constraints modeling (2)
- Generation algorithm for SSN and SSN' sets

Algorithm result

If $SSN_{u_i} \neq \emptyset$ the node u_i is implicated in a regular node of the CG.

If $SSN'_{u_j} \neq \emptyset$ the node u_j is implicated in a edge or path node of the CG as a destination node.

```
for each  $u_i \in SNCP_i$   
   $SSN_{u_i} = \emptyset, SSN'_{u_i} = \emptyset$   
  for each  $u \in V_c$   
    if  $u = u_i/u_j$   
       $SSN_{u_i} = SSN_{u_i} \cup \{Sel_u\}$   
    if  $u = (v_i, u_i)/(u_j, v_j) \wedge v_i \in SNCP_i$   
       $SSN'_{u_i} = SSN'_{u_i} \cup \{Sel_u\}$   
    if  $u = (v_i, w_i, \dots, u_i)/(u_j, v_j) \wedge v_i \in SNCP_i$   
       $SSN'_{u_i} = SSN'_{u_i} \cup \{Sel_u\}$ 
```

Don't forget: $Sel_u = \{0, 1\}$



Model with Multiplexers

- Constraints modeling (3)

➤ Definition of the FDV: For each node u_i two variables, $Start_{u_i}$ and Mux_{u_i} are defined. The latter is defined as follow:

$$Mux_{u_i} = \begin{cases} 1 & \text{if } TNM_{u_i} > 0 \\ R_1 & \text{if } TNM_{u_i} = 0 \wedge SSN_{u_i} \neq \emptyset \wedge SSN'_{u_i} = \emptyset \\ R_2 & \text{if } TNM_{u_i} = 0 \wedge SSN_{u_i} \neq \emptyset \wedge SSN'_{u_i} \neq \emptyset \\ 0 & \text{if } TNM_{u_i} = 0 \wedge SSN_{u_i} = \emptyset \end{cases} \quad (9)$$

$$R_1 \Leftrightarrow \sum_{Sel \in SSN_{u_i}} Sel > 0$$

$$R_2 = R_1 \cdot R, \text{ where } R \Leftrightarrow \sum_{Sel \in SSN'_{u_i}} Sel = 0$$

➤ Constraints definition

$$\forall u_i, v_i \in SNCP_i, (u_i, v_i) \in E_i : Start_{u_i} + Mux_{u_i} \leq Start_{v_i} \quad (10)$$

$$\forall u \in ONS' : Start_u + Mux_u \leq MNM \quad (11)$$



Results

- EPIC decoder application example from Moreano's paper

Application	Input Patterns		Selected Optimizations					Compatibility Graph				Selected Nodes in Max. Weighted Clique			Time in sec	Merged Pattern			Area Imp. in %
	nodes	edges	N	E	P	CP	NM	Nodes			Edges	Reg.	Edge	Path		node	edge	mux	
								Reg.	Edge	Path									
EPIC DECODER	12	11																	
	16	16	Yes	Yes				31	15	0	858	10	6	0	1.02	18	30	5	48%
	15	13	Yes	Yes				49	31	0	2717	15	9	0	0.85				
EPIC DECODER	12	11																	
	16	16	Yes	Yes	2			31	15	8	1151	10	6	1	3.71	18	25	3	50%
	15	13	Yes	Yes	2			49	28	12	3225	15	9	2	1.24				
EPIC DECODER	16	16																	
	12	11	Yes	Yes	2	Yes	X 2	31	15	2	1151	9	6	0	1.18	20	29	4	45%
	15	13	Yes	Yes	2	Yes	X 2	49	28	12	3225	13	8	1	2.15				

>> The system found local optimal weighted cliques <<
And proved their optimality !



Results

- Average area reduction for five different experimentations applied to the *Mediabench* test using the DURASE flow:

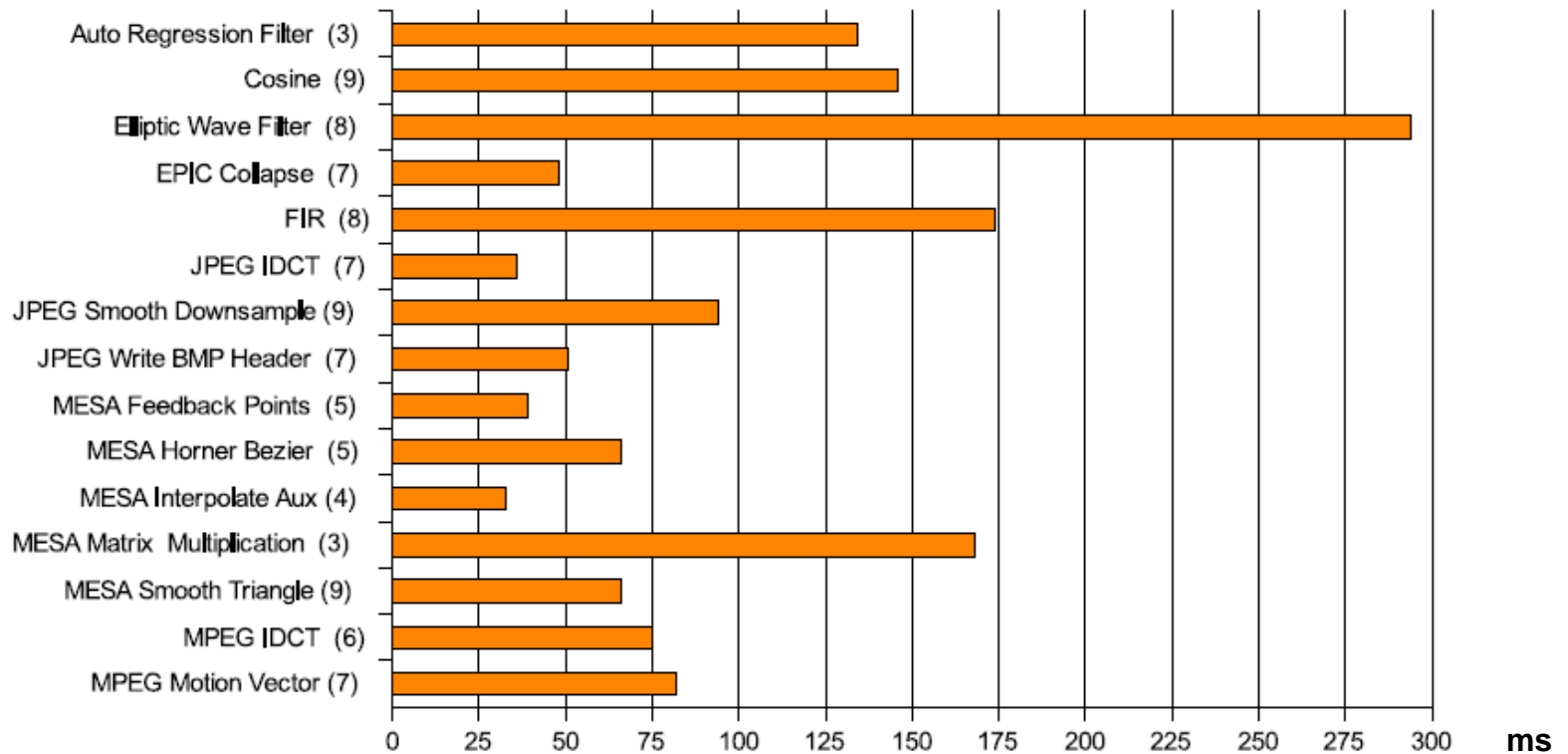
1. Node sharing only	66.5%
2. Node and Edge + 2 Bypassed Node	67.7%
3. N & E + No Mux on the Critical Path	41.1%
4. N & E + No Mux + 2 BP + CP opt.	50.5%
5. N & E + 2 Mux + 2 BP + CP opt.	66.7%

Our system gives good results under different optimization constraints but not only...



Results

- ... gives it fast !



Example : A weighted clique in a graph with 2,122 nodes and 2,116,470 edges in **2 sec**.



Questions...



Constraint Programming (Finite Domain) vs. (Mixed) Integer Programming

	CP(FD)	(M)IP
Variables	Discrete	Continues and discrete
Constraints	Linear & 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 is Software Engineering applied to Operation Research”

Jean-François Puget, ILOG



Constraint Programming (Finite Domain) vs. (Mixed) Integer Programming

	CP(FD)	(M)IP
Variables	Discrete	Continues and discrete
Constraints	Linear & 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 is Software Engineering applied to Operation Research”

Jean-François Puget, ILOG

