Design of Processor Accelerators with Constraints

Christophe Wolinski[1], Krzysztof Kuchcinski[2],
Kevin Martin[3], Erwan Raffin[3,4] and François Charot[3]

[1]Rennes University I/IRISA, France

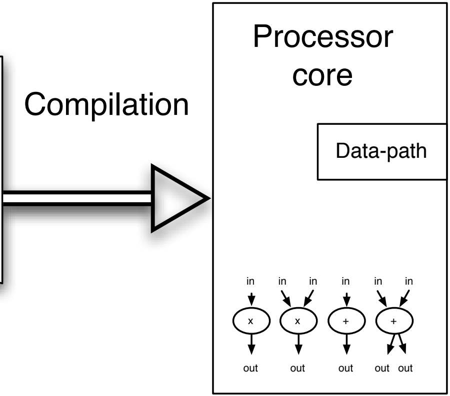
[2]Dept. of Computer Science, Lund University, Sweden

[3]INRIA, Centre Rennes-Bretagne Atlantique, France

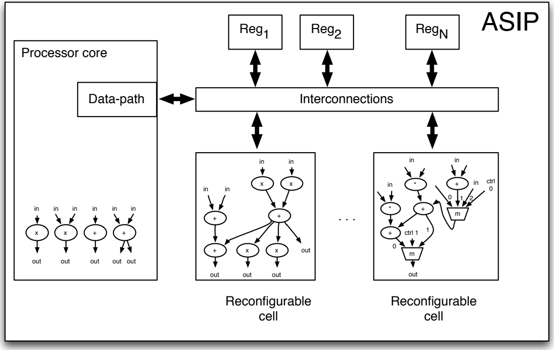
[4]Thomson R&D, Rennes, France

Problem Definition

```
/* Sample C code */
void fir(const int x[], const  int h[], int y[]) {
  int i, j, sum;
  for (j = 0; j < 100; j=j+1) {
    sum = 0;
    for (i = 0; i < 8; i=i+1)
      sum += x[i + j] * h[i];
      sum = sum >> 15;
      y[j] = sum;
  }
}
```



ASIP



- application specific instructions
- reconfigurable units
- better performance and lower power
- etc.

Main Problems

- Identification of computational patterns for instructions
- Selection of a subset of instructions for implementation
- Sequential or parallel execution scenarios
- Pattern merging to build reconfigurable cell

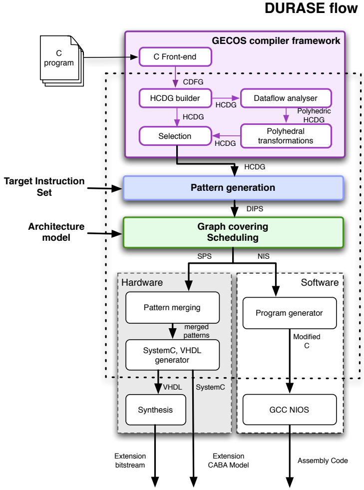
Main Problems

- Identification of computational patterns for instructions
- Selection of a subset of instructions for implementation
- Sequential or parallel execution scenarios
- Pattern merging to build reconfigurable cell

Goal:

Get speed-up of an application with minimal hardware cost.

Our Design Flow



CP-based methods:

- Pattern Generation
- Graph covering and Scheduling
- Pattern merging

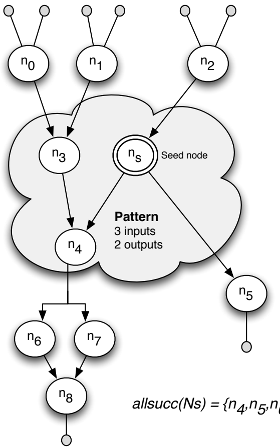
CP Solution

- `JaCoP.graph` constraints
  - (Sub-)graph isomorphism constraints
  - Clique constraints
  - Simple path
  - etc.
- Other standard constraints (number of inputs/outputs, critical path, etc.)

CP Solution

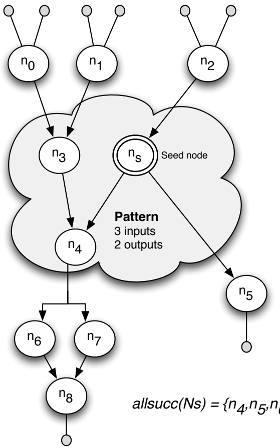
- `JaCoP.graph` constraints
  - (Sub-)graph isomorphism constraints
  - Clique constraints
  - Simple path
  - etc.
- Other standard constraints (number of inputs/outputs, critical path, etc.)
- Methods based on constraints
  - Pattern generation- purely based on constraint
  - Match identification
  - Pattern selection and scheduling
  - Pattern merging

Pattern Generation



$allsucc(Ns) = \{n_4, n_5, n_6, n_7, n_8\}$

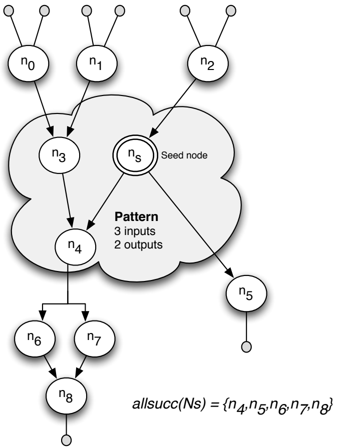
Pattern Generation



allsucc(Ns) = {n4,n5,n6,n7,n8}

$$\forall n \in N_p \wedge n \neq n_s \quad \exists path(P_{n_s}, n, n_s)$$

Pattern Generation

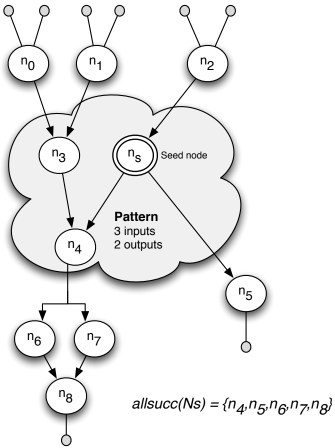


$$\forall n \in N_p \wedge n \neq n_s \quad \exists path(P_{n_s}, n, n_s)$$

$$\forall n \in (N - (allsucc(n_s) \cup n_s)) : n_{sel} = 1 \Rightarrow \sum_{m \in succ(n)} m_{sel} \geq 1$$

$$\forall n \in (N - (allsucc(n_s) \cup n_s)) : \sum_{m \in succ(n)} m_{sel} = 0 \Rightarrow n_{sel} = 0$$

Pattern Generation



$$\forall n \in N_p \wedge n \neq n_s \quad \exists path(P_{n_s}, n, n_s)$$

$$\forall n \in (N - (allsucc(n_s) \cup n_s)) : n_{sel} = 1 \Rightarrow \sum_{m \in succ(n)} m_{sel} \geq 1$$

$$\forall n \in (N - (allsucc(n_s) \cup n_s)) : \sum_{m \in succ(n)} m_{sel} = 0 \Rightarrow n_{sel} = 0$$

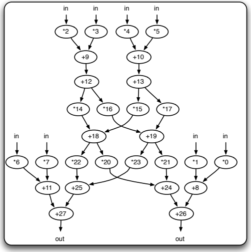
$$\forall n \in allsucc(n_s) : n_{sel} = 1 \Rightarrow \sum_{m \in (pred(n) \cap (allsucc(n_s) \cup n_s))} m_{sel} \geq 1$$

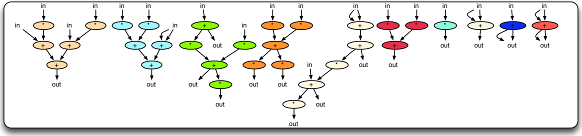
$$\forall n \in allsucc(n_s) : \sum_{m \in (pred(n) \cap (allsucc(n_s) \cup n_s))} m_{sel} = 0 \Rightarrow n_{sel} = 0$$

Pattern Generation (cont'd)

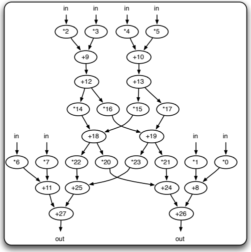
```
DIPS ← ∅
for each n_s ∈ N
  TPS ← ∅
  CPS ← FindAllPatterns(G, n_s)
  for each p ∈ CPS
      if ∀pattern ∈ TPS : p ≢ pattern
          TPS ← TPS ∪ {p},
          NMP_p ← | FindAllMatches(G, p) |
  NMP_{n_s} ← | FindAllMatches(G, n_s) |
  for each p ∈ TPS
      if coef · NMP_n ≤ NMP_p
          DIPS ← DIPS ∪ {p}
return DIPS
```

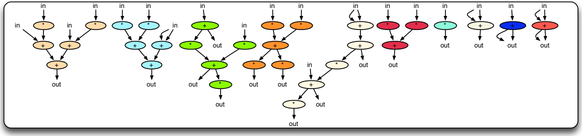
Pattern Selection





Pattern Selection

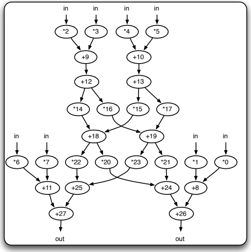


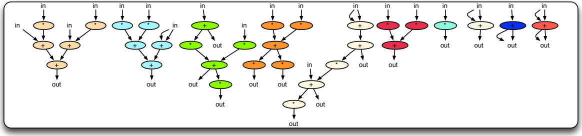


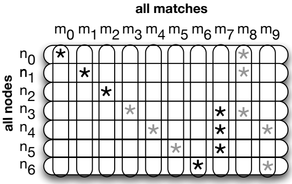
// *Inputs: G=(N,E)-- application graph,*
//   *DIPS-- Definitively Identified Pattern Set*
// M_p-- *set of matches for pattern* p,
// M-- *set of all matches,*
// $matches_n$-- *set of matches that could cover the node* n,

$M \leftarrow \emptyset$
for each $p \in DIPS$
  $M_p \leftarrow$ FindAllMatches(G, p)
  $M \leftarrow M \cup M_p$
for each $m \in M$
  for each $n \in m$
    $matches_n \leftarrow matches_n \cup \{m\}$

Pattern Selection







// Inputs: G=(N,E)-- application graph,
// DIPS-- Definitively Identified Pattern Set
// M_p*-- set of matches for pattern* p,
// M*-- set of all matches,*
// $matches_n$*-- set of matches that could cover the node* n,

$M \leftarrow \emptyset$
for each $p \in DIPS$
  $M_p \leftarrow$ FindAllMatches(G, p)
  $M \leftarrow M \cup M_p$
for each $m \in M$
  for each $n \in m$
    $matches_n \leftarrow matches_n \cup \{m\}$

Pattern Selection and Scheduling

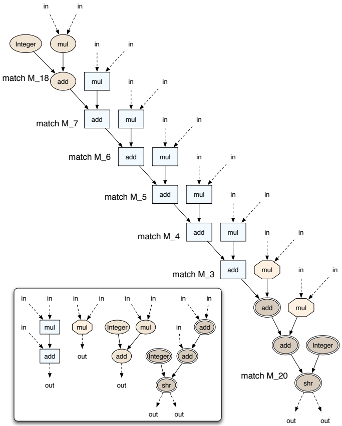
- Match selection- optimize execution time

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

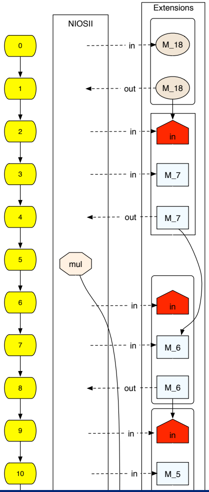
- Match delay defined by constraints

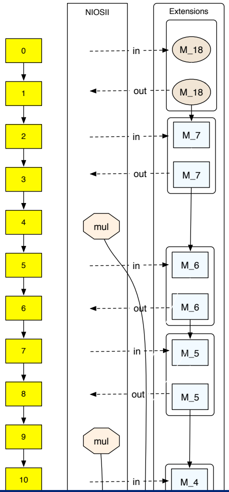
$$m_{delay} = \delta_{in_m} + \delta_m + \delta_{out_m}$$

Scheduling Example- FIR filter

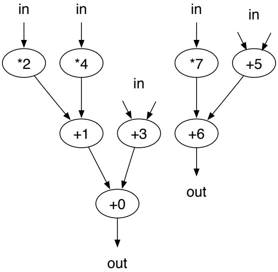


Scheduling Example (cont’d)

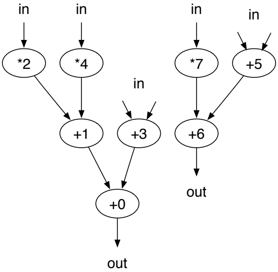


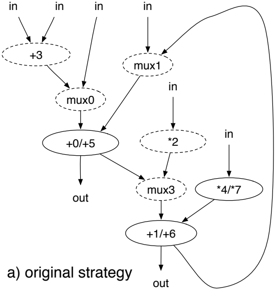


Pattern Merging

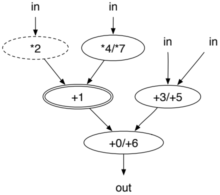


Pattern Merging



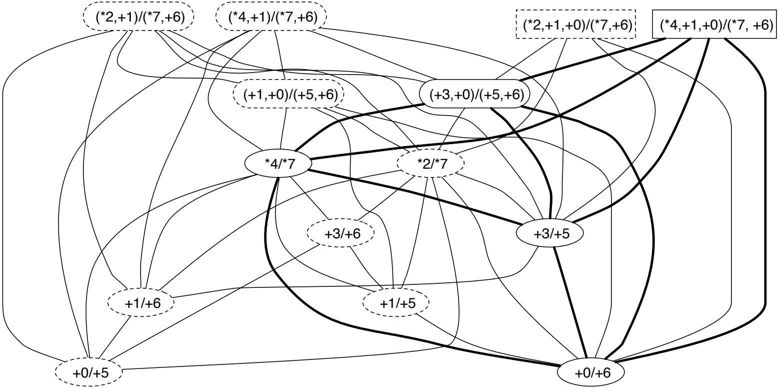


a) original strategy



b) our method

Pattern Merging- compatibility graph



Pattern Merging- additional constraints

- Critical path constraints

$$Delay_u = Latency(u) \cdot sel_u$$
$$\forall (u,v) \in E : Start_u + Delay_u \leq Start_v$$
$$\forall u \in Out : Start_u + Delay_u \leq CPL$$

- Number of multiplexers on critical path

Results

Results obtained for MediaBench and MiBench benchmark sets compiled for NIOS target processor with DURASE system.

			2 in / 1 out										**4 in / 2 out**									
					model A				**model B**						**model A**				**model B**			
Benchmarks	Nodes	cycles	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

Results (cont'd)

Rijndael and GSM encoders for patterns with 7 nodes limit.

Application	|V|	identified	selected	coverage	speed-up Seq.	speed-up Par.
Part of Rijndael encryption encoder	106	10	6	75%	1.9	2.9
Part of GSM encoder	604	11	7	66%	2.1	3.4

Conclusions

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

Further Reading

- Ch. Wolinski and K. Kuchcinski.
  Automatic selection of application-specific reconfigurable processor extensions.
  In *Proc. Design Automation and Test in Europe*, Munich, Germany, March 10-14, 2008.

- Ch. Wolinski, K. Kuchcinski, K. Martin, E. Raffin, and F. Charot.
  How constrains programming can help you in the generation of optimized application specific reconfigurable processor extensions.
  In *Proc. of The Intl. Conference on Engineering of Reconfigurable Systems and Algorithms*, Las Vegas, USA, (Invited paper), July 13-16, 2009.

- K. Martin, Ch. Wolinski, K. Kuchcinski, A. Floch, and F. Charot.
  Constraint-driven identification of application specific instructions in the *DURASE* system.
  In *SAMOS IX: International Workshop on Systems, Architectures, Modeling and Simulation*, Samos, Greece, July 20-23, 2009.