UNIVERSITÉ DE RENNES 1

Extraction d'instructions spécialisées pour ASIP

Projet CAIRN IRISA/INRIA

Travaux de Kevin Martin, Antoine Floc'h, Christophe Wolinski, François Charot,

(Présentés par Steven Derrien)

Principe

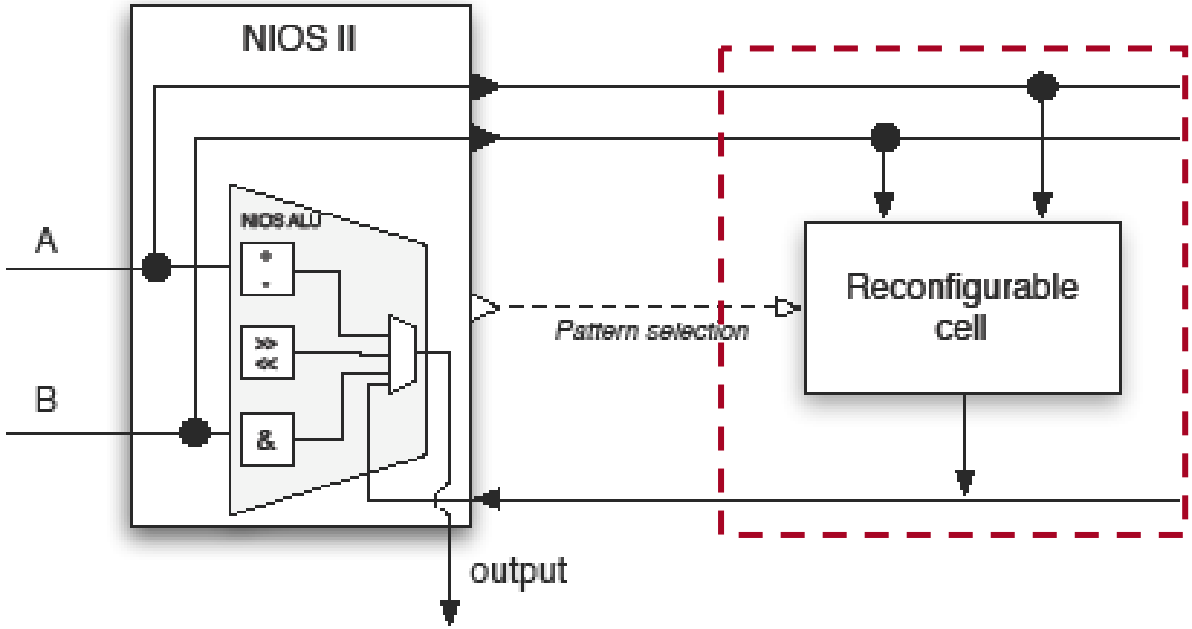
- Etendre le jeu d'instruction d'un processeur avec des instr. exécutant des motifs de traitements complexes.
- Permet d'obtenir des gains en performance relativement significatifs (entre +30% et x3) à moindre coût.

Problèmes

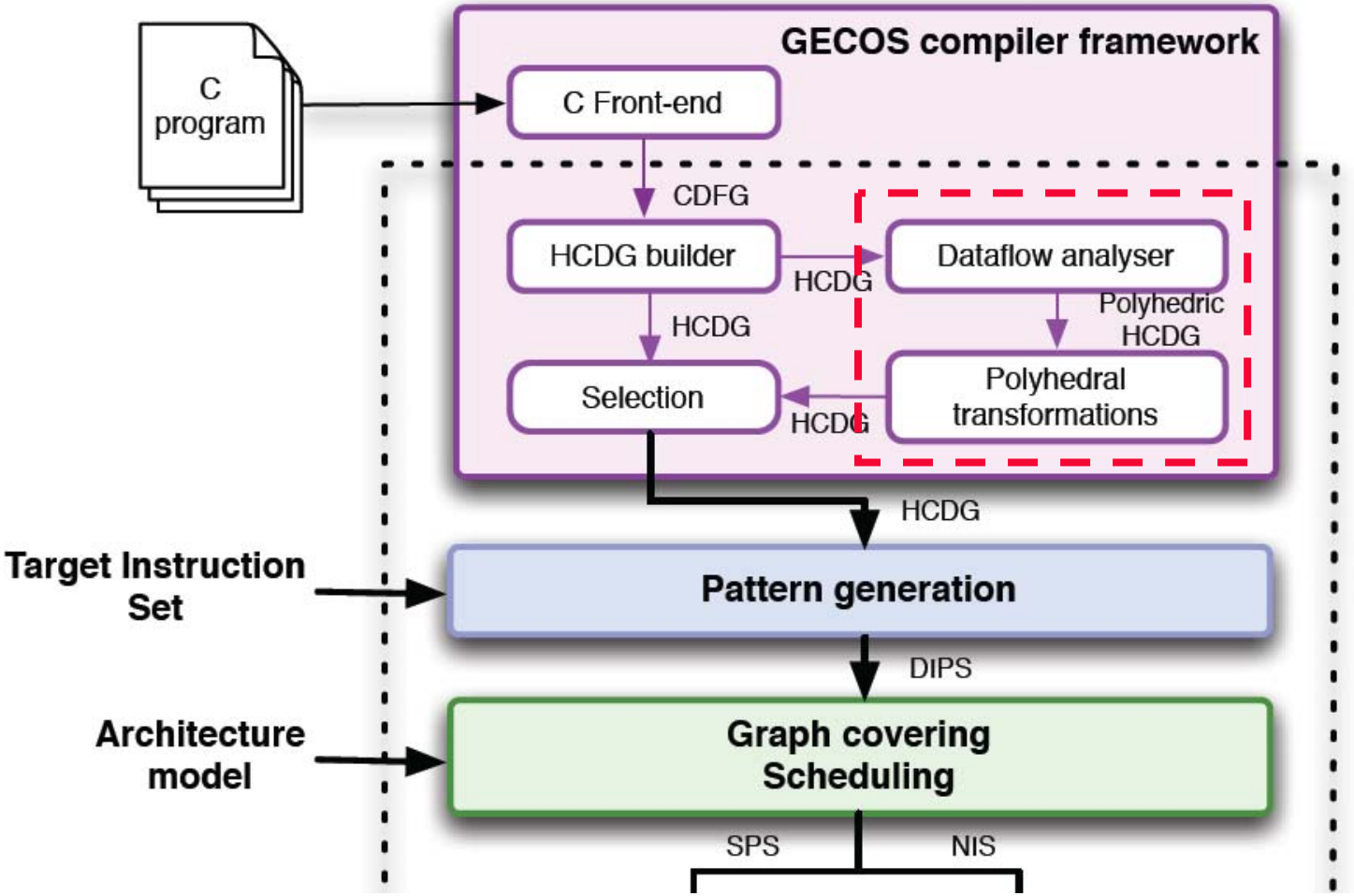
- Identifier les motifs de calculs intéressant dans le prog.
- Intégrer ces nouvelles instructions au flot de compilation.
- Générer la microarchitecture du processeur « étendu ».

Les custom instructions du NIOS II

- Plusieurs types d'instructions spécialisées
  - Combinatoire ou multi-cycle, l'exécution pipelinée n'est pas directement supportée par la microarchitecture.
  - Accès à la file de registre (2 lectures/1 écriture par cycle).
  - Jusqu'à 256 opcodes disponibles.



Outil DURASE : Flot général



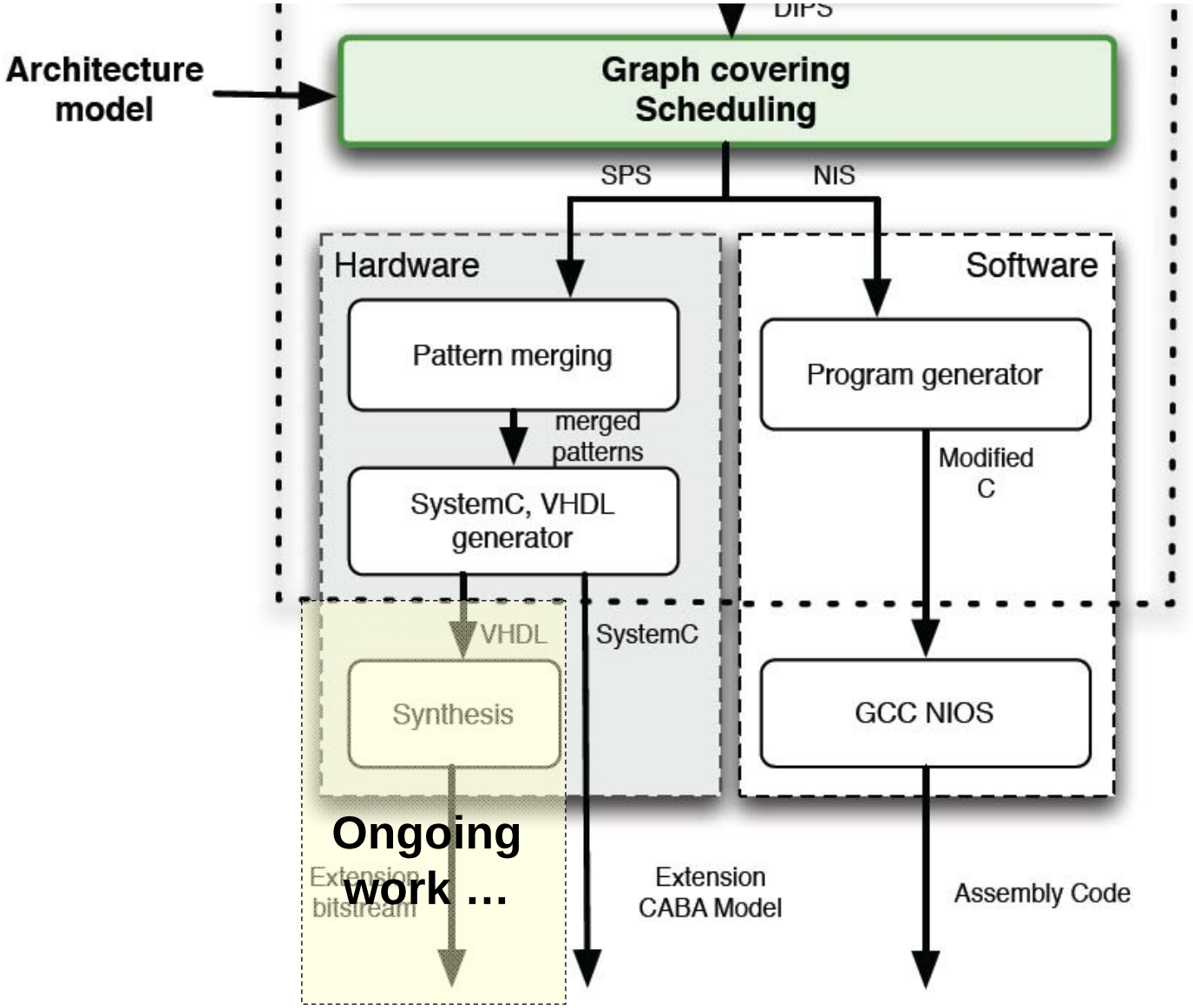
1. Pattern generation

- Recherche des motifs de calculs qui apparaissent fréquemment dans le code.
- Problème d'isomorphisme de sous-graphes (NP) combiné avec des heuristiques de filtrage de motifs.
- Critères de filtrage (non exhaustif)
  - Motif contient un accès mémoire, etc.
  - Sortie d'un nœud du motif utilisé par un nœud qui n'appartient pas au motif (contrainte de convexité).
  - Ratio nombre d'E/S par rapport aux calculs trop faible, ou taille du motif trop importante (# nœuds).
  - Chemin critique associé à la réalisation du motif trop élevé.

2. Graph covering/scheduling

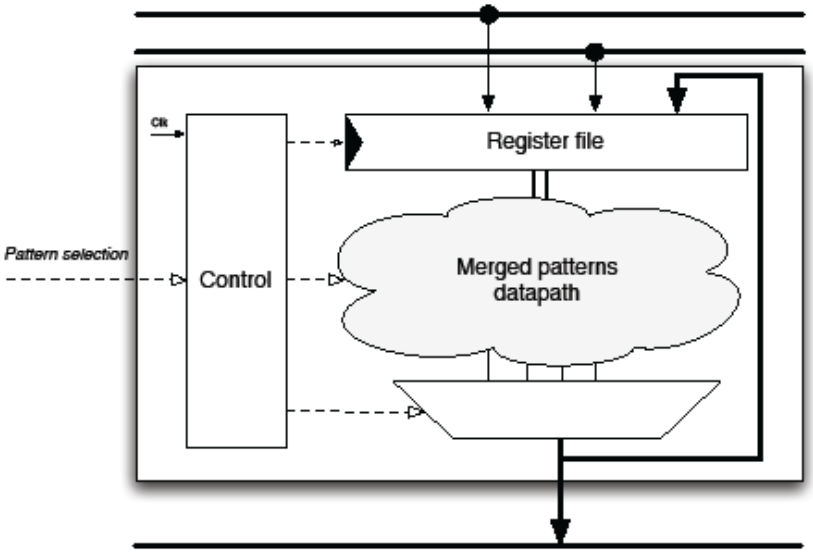
- Objectif : régénérer le code machine en exploitant les motifs de calculs extraits à l'étape précédente.
  - En intégrant les latences des motifs de calculs, ce qui peut permettre une exécution pipelinée de ces motifs.
- Problème de couverture optimale de graphe sous contraintes (utilisation d'un solveur CSP Jacop).
  - Coût = f (# motifs différents, durée d'exécution, #instructions)
  - Fonctionne correctement sur les benchmarks (mediabench)
  - Problème NP Complet, problèmes de passage à l'échelle !

Flot général



Pattern merging

- Pour éviter d'avoir autant de chemins de données qu'il y a d'I.S. on les fusionne en un seul.



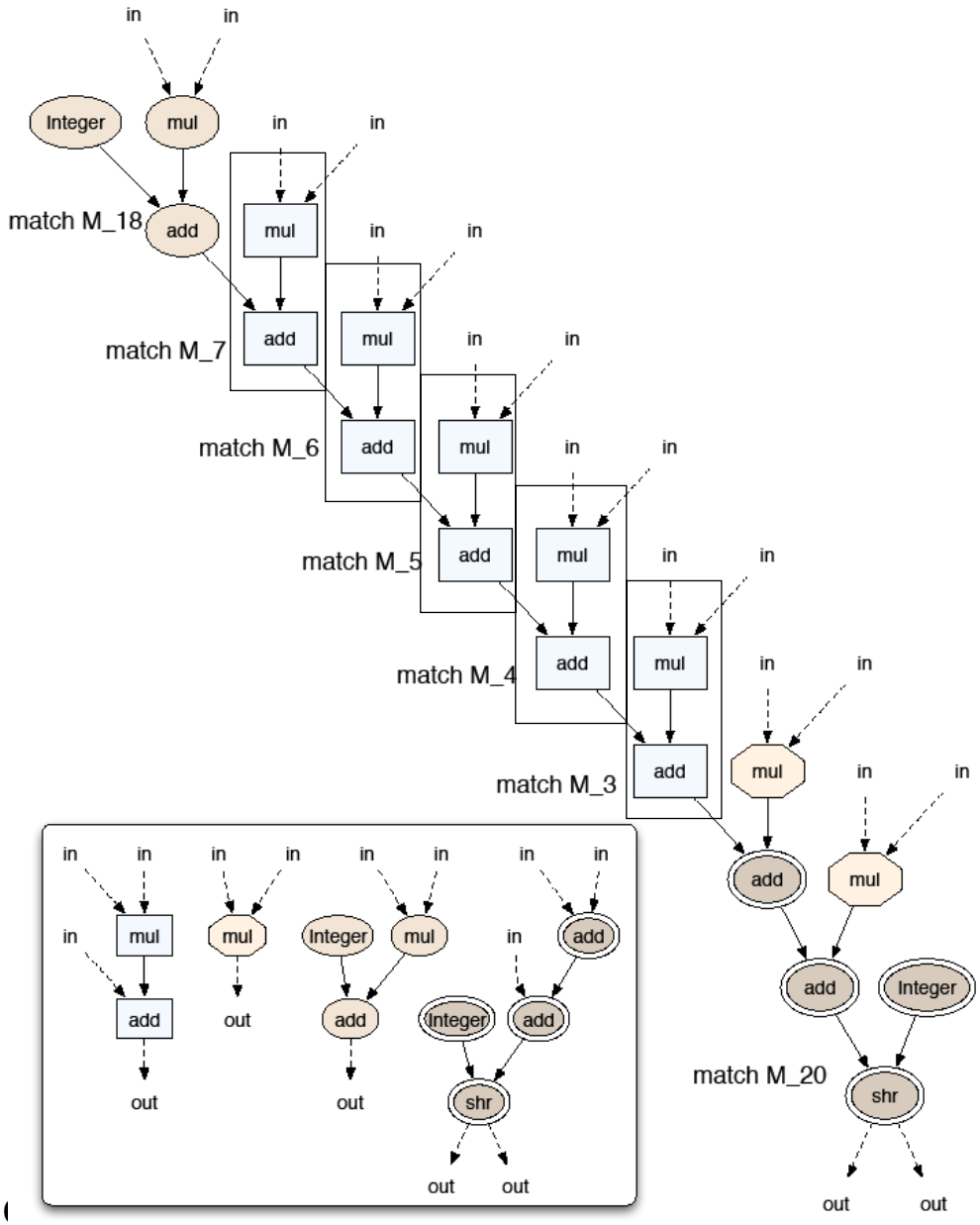
- Utilise un graphe de compatibilité qui définit les possibilités de correspondances entre deux nœuds de motifs différents
- La fusion de coût minimum entre 2 motifs revient à réaliser un recherche de clique de poids maximum (NP complet).

Exemple : FIR

```
/* 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;
  }
}
```

```
/* Transformed C code */
void fir_n(const int x[], const int h[], int y[]) {
  int j;
  int sum;
  for (j = 0; j < 100; j = j + 1) {
     sum = x[(0 + j)] * h[0];
     sum = sum + x[(1 + j)] * h[1];
     sum = sum + x[(2 + j)] * h[2];
     sum = sum + x[(3 + j)] * h[3];
     sum = sum + x[(4 + j)] * h[4];
     sum = sum + x[(5 + j)] * h[5];
     sum = sum + x[(6 + j)] * h[6];
     sum = sum + x[(7 + j)] * h[7];
     sum = sum >> 15;
     y[j] = sum;
  }
}
```



Réuni

Exemple : FIR (suite)

- Code C régénéré incluant les macros *custom instruction*
  - Fonctionnalité offerte par GCC, pas spécifique au NIOS

```
void fir_n(int x[], int h[], int y[100]) {
    int j;
    int sum;
    for (j = 0; j < 100; j = j + 1) {
        asm volatile ("custom  2, r0, %0, %1" : : "r"(x[(0 + j)]), "r"(h[0]));
        asm volatile ("custom  3, r0, %0, %1" : : "r"(x[(1 + j)]), "r"(h[1]));
        asm volatile ("mul     %0, %1,%2" : "=r"(tmp1) : "r"(x[(6 + j)]), "r"(h[6]));
        asm volatile ("custom  3, r0, %0, %1" : : "r"(x[(2 + j)]), "r"(h[2]));
        asm volatile ("custom  3, r0, %0, %1" : : "r"(x[(3 + j)]), "r"(h[3]));
        asm volatile ("mul     %0, %1,%2" : "=r"(tmp2) : "r"(x[(7 + j)]), "r"(h[7]));
        asm volatile ("custom  3, r0, %0, %1" : : "r"(x[(4 + j)]), "r"(h[4]));
        asm volatile ("custom  3, r0, %0, %1" : : "r"(x[(5 + j)]), "r"(h[5]));
        asm volatile ("custom  4, %0, %1, %2" : "=r"(sum) : "r"(tmp1), "r"(tmp2));
        y[j] = sum;
    }
}
```

- Modèle ISSS NIOS II SocLib incluant les instructions
  - Modifications légères (décodeur + étage d'exécution)

Résultats préliminaires

- Validation basée sur la plate-forme SOCLib
  - Utilisation du modèle NIOS-II

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

Intégration dans le projet

- Dans une optique système : offrir la possibilité de générer des processeurs + instructions spécialisées.
  - Solution intermédiaire entre HLS et software
- Il faudrait alors sans doute adapter le flot au MIPS
  - Peut permettre un couplage + fort avec la microarchitecture, et ouvrir d'autre possibilités (accès mémoire, réutilisation des opérateurs de l'étage d'exécution, etc.).
- Les formats d'entrées
  - Pour l'instant un sous-ensemble de C-ANSI supporté par la représentation HCDG,
  - MAIS, la boite à outils devrait pouvoir fonctionner à partir d'une représentation de graphe flot de données.

Questions