

Structures Newgen pour l'intégration de PAF dans PIPS

Françoise Lamour
Arnauld Leservot
Alexis Platonoff

DMA/PCP, CEA CEL-V

October 20, 2021

Introduction

Les structures de données suivantes sont utilisées par les phases qui reproduisent les calculs mis en œuvre dans le paralléliseur PAF : le *graphe du flôt de données* (DFG, “Data Flow Graph”), la *base de temps* (BDT), la *fonction de placement* (PLC) et les appels au logiciel de résolution de systèmes paramétriques en nombres entiers (PIP).

Après une description rapide des structures de données importées et externes, nous présentons successivement les notions de DFG, de BDT, de PLC, de Quast, de Lisp_expression et de Static_control.

1 Structures de données importées

Import reference from "ri.newgen"

Le domaine **reference** est utilisé pour représenter une référence à un élément de tableau. Les variables scalaires étant représentées par des tableaux de dimension 0, les références à des scalaires sont aussi prises en compte. Elles contiennent une liste vide d'expressions d'indices.

Import predicate from "ri.newgen"

Le domaine **predicate** définit une relation entre valeurs de variables scalaires entières.

Import expression from "ri.newgen"

Le domaine **expression** permet de stocker les expressions sous deux formes. La première est une description de l'expression telle qu'elle apparaît dans le texte

source du programme et la deuxième est une description compilée des expressions linéaires sous forme de *Pvecteur*.

```
Import loop from "ri.newgen"
```

Le domaine *loop* permet de représenter les boucles du type *DO Fortran* ou *FOR Pascal*.

```
Import statement from "ri.newgen"
```

Le domaine *statement* permet de repérer les instructions d'un module.

```
Import entity from "ri.newgen"
```

Tout objet ayant un nom dans un programme Fortran est représenté par une *entity*. Un tel objet peut être un module, une variable, un common, un opérateur, une constante, un label, etc.

```
Import sccflags from "dg.newgen"
```

Le domaine *sccflags* est utilisé par l'algorithme de calcul des composantes fortement connexes d'un graphe.

2 Structures de données externes

External Pvecteur

Le domaine *Pvecteur* est utilisé pour représenter les expressions linéaires telles que $3I+2$ ou des contraintes linéaires telles que $3I + J \leq 2$ ou $3I == J$.

Un *Pvecteur* est une suite de monômes, un monôme étant un couple (coefficient, variable). Le coefficient d'un tel couple est un entier, positif ou négatif. La variable est une entité, sauf dans le cas du terme constant qui est représenté par la variable prédéfinie de nom *TCST*.

La structure de données *Pvecteur* est importée de la bibliothèque (C3) d'algèbre linéaire en nombres entiers du *CRI*.

3 Graphe du flôt de données

```
Dfg_vertex_label = statement:int x exec_domain:predicate x sccflags
```

Le domaine *dfg_vertex_label* définit la structure des informations qui seront attachées à un nœud du *DFG*. Le sous-domaine *statement* spécifie l'instruction sur laquelle porte la dépendance ; cet entier correspond au champ *ordering* du domaine *statement*. Le sous-domaine *exec_domain* est le domaine d'exécution de cette instruction ; c'est un prédicats, qui correspond aux bornes des boucles englobantes ainsi qu'aux tests englobants. Le sous-domaine *sccflags* contient diverses informations utiles pour le calcul des composantes fortement connexes. La structure de graphe elle-même (domaine *graph*) est défini dans le fichier *graph.f.tex*.

`Dfg_arc_label = dataflows:dataflow*`

Le domaine `dfg_arc_label` définit la structure des informations qui seront attachées à un arc du DFG, i.e. le flôt de données passant par cet arc. La structure de graphe elle-même (domaine `graph`) est défini dans le fichier `graph.f.tex`.

`Dataflow = reference x transformation:expression* x governing_pred:predicate x communication`

Le domaine `dataflow` permet de décrire les informations nécessaires à la description du flôt d'une donnée. Le sous-domaine `reference` donne la référence sur laquelle porte la dépendance. Le sous-domaine `transformation` donne dans l'ordre la liste d'expressions qui donnent la valeur des indices des boucles englobantes de l'instruction source en fonction des indices des boucles englobantes de l'instruction destination. Le sous-domaine `governing_pred` donne le prédicat de contrôle de la dépendance. Enfin, le sous-domaine `communication` spécifie le type de communication que ce flôt engendre.

`Communication = broadcast:predicate x reduction:predicate x shift:predicate`

Le domaine `communication` permet de spécifier le type de communication qui peut être engendrée par le flôt auquel elle est associée. Le sous-domaine `broadcast` correspond à une diffusion. Le sous-domaine `reduction` correspond à une récurrence. Le sous-domaine `shift` correspond à une translation.

La direction d'une communication est caractérisée par un Pvecteur qui est exprimée en fonction des indices des boucles englobantes. Chaque type peut avoir plusieurs directions, ainsi, nous utilisons la structure `predicate` pour les spécifier. Dans ce `predicate`, le `Psystème` correspondant ne sera constitué que d'équations, chacune représentant la direction de communication.

4 Base de temps

`Bdt = schedules:schedule*`

Le domaine `bdt` définit la base de temps pour l'ensemble des instructions du programme. Elle se compose tout simplement d'une liste de fonctions de temps, chacune associée à une instruction et un prédicat.

`Schedule = statement:int x predicate x dims:expression*`

Le domaine `schedule` est utilisé pour décrire la fonction de temps associée à une instruction et un prédicat. C'est en fait une liste de fonctions, car la base de temps peut être multidimensionnelle. Le sous-domaine `statement` correspond au champ `ordering` du `statement` auquel correspond cette base de temps. Le sous-domaine `predicate` donne le prédicat d'existence de cette fonction. Enfin, le sous-domaine `dims` donne les différentes expressions de la fonction multidimensionnelle.

5 Fonction de placement

`Plc = placements:placement*`

Le domaine `plc` définit la fonction de placement pour l'ensemble des instructions du programme. Elle se compose tout simplement d'une liste de placements, chacun associé à une instruction.

`Placement = statement:int x dims:expression*`

Le domaine `placement` est utilisé pour décrire le placement associée à une instruction. C'est en fait une liste d'expressions, car la fonction de placement peut être multidimensionnelle. Le sous-domaine `statement` correspond au champ `ordering` du `statement` auquel correspond ce placement. Enfin, le sous-domaine `dims` donne les différentes expressions de la fonction multidimensionnelle.

6 Représentation des quasts

`Quast = quast_value x newparms:var_val*`

Le domaine `quast` est utilisé pour contenir les informations renvoyées par PIP. Ces informations sont l'expression (dépendant de certaines conditions) de la valeur de chaque variable du système en fonction des paramètres de structure. Le sous-domaine `quast_value` donne la valeur du `quast`.

Le sous-domaine `newparms` donne la liste des nouveaux paramètres de structure introduit par le `quast`. Pour chaque nouveau paramètre, l'information est composée d'une variable et d'une valeur. La variable est une entité représentant le nouveau paramètre de structure et la valeur est l'expression de sa valeur en fonction des autres paramètres de structure. La création de nouveau paramètre de structure est nécessaire lorsqu'apparaissent dans les expressions des divisions par des constantes, comme par exemple : $n/2$.

`Quast_value = quast_leaf + conditional`

Le domaine `quast_value` contient la valeur du `quast` dans une structure directement inspirée de la structure lisp "quast" utilisée dans Paf. Cette valeur est soit une feuille `quast_leaf`, soit un `conditional`.

`Conditional = predicate x true_quast:quast x false_quast:quast`

Le domaine `conditional` est utilisé pour contenir les informations d'un `quast` qui dépendent d'un prédicat. Si le `predicate` est vrai, alors l'information utilisée est celle contenue dans `true_quast`, sinon l'information utilisée est celle contenue dans `false_quast`.

`Quast_leaf = solution:expression* x leaf_label`

Le domaine `quast_leaf` contient une solution rendu par PIP et le champ `leaf_label` utilisé lors du calcul d'une source dans le data flow graph. `Solution`

est une liste d'expressions, dont chacune donne la valeur de la solution trouvée par PIP pour la variable de même rang dans la liste des variables du système (l'ordre de ces deux listes est donc très important).

```
Leaf_label = statement:int x depth:int
```

Le domaine `leaf_label` est utilisé lors du calcul d'une source : il informe sur l'origine de cette source (`statement`) ainsi que sur la profondeur de la dépendance (`depth`). Le quast rendu par PIP n'utilise pas ce champ ; il est dans ce cas mis à `leaf_label_undefined`.

```
Var_val = variable:entity x value:expression
```

Le domaine `var_val` est utilisé pour spécifier l'association d'une variable et de sa valeur exprimée en fonction d'autres variables.

7 Représentation des expressions Lisp

```
Lisp_expression = operation:string x args:expression*
```

Le domaine `lisp_expression` permet de décrire les informations contenues dans une expression numérique en notation Lisp. Une telle expression est composée d'un opérateur et d'une liste d'arguments (chaque argument étant une `expression`). Cette structure est utilisée pour les codes réalisants le transfert des structures de PAF depuis les fichiers de sortie écrits en notation `Le_Lisp` vers les structures `newgen` définies plus haut.

8 Représentation du contrôle statique

```
Static_control = yes:bool x params:entity* x loops:loop* x tests:expression*
```

Le domaine `static_control` définit la structure de l'information qui est associée à une instruction (*i.e.* un `statement`). Le sous-domaine `yes` indique si ce `statement` se trouve dans une zone à contrôle statique ; si ce n'est pas le cas, les trois sous-domaines suivant sont vides. Le sous-domaine `params` donne la liste des paramètres de structure de la zone à contrôle statique dans laquelle se trouve ce `statement`. Le sous-domaine `loops` donne la liste des boucles englobantes de ce `statement`. Le sous-domaine `tests` donne la liste des conditions trouvées dans les tests englobants de ce `statement`.