RenPar'6, L. Bougé (réd.),
ENS Lyon, France, 6–10 juin 1994.

Calcul de Régions de Tableaux Exactes

Béatrice APVRILLE, apvrille@cri.ensmp.fr

Centre de Recherche en Informatique, École Nationale Supérieure des Mines de Paris
35, rue Saint Honoré, 77305 FONTAINEBLEAU Cedex, FRANCE

Résumé

Un nouvel algorithme de calcul des régions de tableaux exactes est présenté. Il s'appuie sur une analyse préalable des effets des instructions et des procédures sur les valeurs des variables scalaires entières. Il a été implémenté dans le cadre du projet PIPS, *où les régions sont également utilisées pour le calcul des dépendances, et constitue une première étape vers l'implémentation d'un algorithme de privatisation de tableau. Blume et Eigenmann ont en effet récemment montré qu'une parallélisation automatique efficace nécessitait la détection des sections de tableaux privatisables.*

1. Introduction

Blume et Eigenmann [3] ont récemment mené une étude sur les performances des paralléliseurs et les améliorations à leur apporter. Ils ont notamment montré la nécessité de détecter les ensembles d'éléments de tableaux locaux à une boucle, et donc privatisables. Pour cela, il est nécessaire de connaître avec précision les éléments de tableaux qui sont écrits à l'intérieur du corps de la boucle.

Dans le projet PIPS [1], le concept de *régions* [13] a été choisi pour représenter les références aux éléments des tableaux. Une région est un ensemble d'éléments de tableaux décrit par des équations et inéquations affines formant un polyèdre convexe. Deux autres notions ont été introduites pour traiter le problème précédent et celui des effets des procédures sur les tableaux : (1) l'*action* réalisée sur les éléments de la région (`READ` (`R`) pour une utilisation ou `WRITE` (`W`) pour une définition) ; (2) l'*approximation* de la région (`MUST` si tous les éléments de la région sont effectivement concernés par l'action, pour tous les chemins du graphe de contrôle et pour toutes les valeurs possibles des variables intervenant dans les expressions des indices ; ou `MAY` si les éléments de la région sont seulement potentiellement atteints). Par exemple,

1. Paralléliseur Interprocédural de Programmes Scientifiques ([7, 8])

```
K = FOO()
DO I = 1,N
   DO J = 1,N
      WORK(J,K) = J + K
   ENDDO
   CALL INC1(K)
   DO J = 1,N
      WORK(J,K) = J * J - K * K
      A(I) = A(I) + WORK(J,K) + WORK(J,K-1)
   ENDDO
ENDDO

SUBROUTINE INC1(I)
I = I + 1
END
```

FIG. 1 - Exemple d'application

la région :

```
<A(PHI1,PHI2)-W-MUST-{1<=PHI1,PHI1<=3,PHI1==PHI2}>
```

où `PHI1` et `PHI2` représentent respectivement la première et la deuxième dimension de `A`, correspond à une référence en écriture aux éléments `A(1,1)`, `A(2,2)` et `A(3,3)`.

Dans l'exemple de la figure 1, les éléments du tableau `WORK` sont référencés à l'aide de la variable `K`, dont la valeur ne s'exprime pas linéairement en fonction des valeurs des autres variables, et qui est modifiée dans le corps de la boucle `I`. Il est nécessaire de connaître cette transformation de la valeur de `K` pour calculer avec précision l'ensemble des éléments du tableau `WORK` définis dans une itération de la boucle `I`, et pouvoir conserver l'approximation `MUST` de la région correspondante. Cette approximation indique que l'information portée par la région est exacte, et pourra donc être utilisée pour traiter le problème de la privatisation de tableaux. Pour préserver ces approximations `MUST`, un nouvel algorithme de calcul des régions d'une séquence linéaire d'instructions complexes est introduit. Il s'appuie sur une analyse préalable des effets des instructions et des

procédures sur les valeurs des variables scalaires entières.

Cet article est organisé de la façon suivante : la section 2 décrit l'algorithme de calcul des régions d'une séquence linéaire, après une présentation des caractéristiques de PIPS nécessaires à sa compréhension ; les résultats obtenus avec l'exemple précédent y sont également présentés ; enfin, la section 3 donne une comparaison avec d'autres travaux.

2. Calcul des régions

Le calcul des régions comporte deux phases : (1) le calcul intraprocédural, qui s'appuie sur le graphe de contrôle hiérarchisé du corps de la procédure pour calculer un résumé de ses effets sur les éléments de tableaux ; (2) la propagation interprocédurale, qui élimine des régions du corps de la procédure les variables locales, et masque les régions correspondant à des variables désactivées en sortie de procédure ; le résumé obtenu est ensuite utilisé au niveau des sites d'appel après traduction des paramètres formels en paramètres réels.

Cette section s'intéresse à la première phase, et plus précisément au calcul des régions d'une séquence linéaire d'instructions complexes. La deuxième phase a déjà été décrite dans [7].

2.1. Pré-requis

Le calcul des régions repose sur deux types d'informations calculées dans des phases précédentes du processus de parallélisation de PIPS : les *transformeurs* et les *préconditions* [7].

Les transformeurs donnent les relations affines qui existent entre les valeurs des variables scalaires entières dans les états-mémoires suivant et précédant l'exécution d'une instruction ou l'appel d'une procédure. La figure 2 donne avant chaque instruction son transformeur sous la forme `T(argument) {prédicat}` ; l'argument du transformeur donne la liste des variables modifiées, et le prédicat les relations affines non triviales[2] entre ces variables. On remarquera dans la figure 2 que l'expression de `K` dans l'instruction `K = FOO(N)` n'étant pas affine, aucune équation ou inéquation n'apparaît dans le prédicat du transformeur correspondant.

Les préconditions sont des prédicats sur les variables scalaires entières, vrais avant exécution de l'instruction. L'exemple de la figure 2 donne pour chaque instruction ses préconditions, sous la forme

2. Une relation triviale est de la forme `I == I`.

```
C  T(K) {K == 3}
   K = 3
C  T(K) {}
   K = FOO(N)
C  T(K) {K == K + 1}
   CALL INC1(K)
```

```
C  P() {}
   K = 3
C  P(K) {K==3}
   K = FOO()
C  P(K) {}
   CALL INC1(K)
C  P(K) {}
```

FIG. 2 - Transformeurs et préconditions

`P(var) {...}`, qui exprime l'effet du module courant sur son état-mémoire, entre son point d'entrée et l'instruction concernée.

Les transformeurs sont calculés "bottom-up", c'est-à-dire de la fin du module courant vers le début, et les préconditions sont propagées dans le sens contraire, de telle sorte que si T_1 et P_1 sont respectivement le transformeur et la précondition associés à l'instruction S_1, P_2 la précondition associée à l'instruction S_2 suivant S_1, alors $P_2 = T_1(P_1)$.

2.2. Description de l'algorithme

Une région est exprimée en fonction des valeurs des variables scalaires entières dans l'état-mémoire précédant l'exécution de l'instruction à laquelle elle correspond. Ces valeurs sont contraintes par les préconditions. Le principe de l'algorithme est d'exprimer les régions dans l'état-mémoire précédant l'exécution de la séquence linéaire, en tenant compte des effets des instructions modifiant les valeurs des variables scalaires entières en fonction desquelles elles sont exprimées. Pour effectuer cette transformation, on utilisera les transformeurs.

Problème

Soit B, la séquence linéaire constituée des instructions $S_1, S_2, \ldots, S_n$. On cherche R_0 l'ensemble de régions associé à B.

Définition

Soient T_k et R_k, le transformeur et l'ensemble de régions associés à l' instruction S_k, $k \in [1, n]$.

On définit la transformation *inverse* de T_k, $\widetilde{T}_k^{-1}$ par :

$$\widetilde{T}_k^{-1} : R_{k+1} \rightarrow \widetilde{R}_k = proj_{\sigma_k}(intersection(R_{k+1}, T_k))$$

$\widetilde{R}_k$ est la projection sur l'état-mémoire σ_k de la région dont le prédicat est l'intersection du prédicat initial de R_{k+1} et du prédicat de T_k. Cette pro-

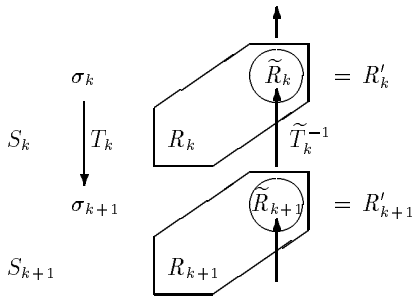

FIG. 3 - Algorithme

jection consiste en une élimination des variables modifiées par l'instruction, qui sont données par l'argument du transformeur. Si le transformeur T_k décrit exactement la transformation effectuée par S_k, alors les approximations `MUST` peuvent être conservées. Dans ce cas, la projection consiste uniquement en des substitutions exactes de variables[3].

Algorithme

Pour $n \geq 1$,

$$\begin{cases} R'_k = env\_conv(R_k, \widetilde{T}_k^{-1}(R'_{k+1})), \forall k \in [1, n-1] \\ R'_n = R_n \\ R_0 = R'_1 \end{cases}$$

R'_k est l'ensemble de régions associé à la séquence linéaire $S_k, \ldots, S_n$ et exprimé dans l'état-mémoire σ_k précédant S_k. L'opérateur *env_conv* rend une région dont le prédicat est l'enveloppe convexe des prédicats des régions initiales. Si cette enveloppe convexe est égale à l'union des prédicats, alors les approximations `MUST` sont préservées ; sinon l'approximation de la région résultante devient `MAY`.

La figure 3 donne une vision intuitive de cet algorithme. En appliquant $\widetilde{T}_k^{-1}$ à la région R'_{k+1} obtenue à l'étape précédente, on obtient $\widetilde{R}_k$ qui est exprimée dans l'état σ_k. R'_k est alors l'enveloppe convexe de $\widetilde{R}_k$ et de R_k qui est la région associée à S_k et exprimée dans l'état σ_k. On a également fait figurer le transformeur T_k qui donne une approximation de la transformation de σ_k en σ_{k+1} par S_k.

Propriété 1 *L'algorithme précédent définit bien l'enveloppe convexe des régions initiales, exprimée dans l'état-mémoire précédant l'exécution de la séquence linéaire B. Il préserve les approximations `MUST` lorsque les projections dûes au calcul des* $\widetilde{T}_k^{-1}$ *se limitent à des substitutions exactes.*

Propriété 2 *L'algorithme précédent donne des régions plus précises que lorsque seuls les effets sur les variables scalaires entières sont utilisés*[4]*, dans le sens où elles contiennent moins d'éléments n'appartenant pas aux régions initiales.*

Ces propriétés ont été démontrées dans [1].

2.3. Exemple d'application

Cette section montre les améliorations obtenues par rapport à l'algorithme n'utilisant que les effets des procédures et des instrutions sur les variables scalaires entières. Reprenons pour cela l'exemple de la figure 1.

On s'intéresse aux accès en écriture des éléments du tableau `WORK` (régions `W`). La région R correspondant à la deuxième boucle `J` est donnée par :

```
<WORK(PHI1,PHI2)-W-MUST-
     {1<=I,I<=N,1<=PHI1,PHI1<=N,PHI2==K}>
```

L'instruction précédente, `CALL INC1(K)`, modifie `K`. Si l'on utilise simplement les effets de ce `CALL` (i.e. `K` modifié) pour exprimer la région R dans l'état-mémoire précédant cette instruction, on obtient la régions $\widetilde{R}$:

```
<WORK(PHI1,PHI2)-W-MAY-{1<=I,I<=N,1<=PHI1,PHI1<=N}>
```

qui ne contient plus d'information sur la deuxième dimension des éléments référencés, ce qui explique que ce soit une région `MAY`. La région R_0 obtenue pour l'ensemble de la séquence linéaire est finalement :

```
<WORK(PHI1,PHI2)-W-MAY-{1<=I,I<=N,1<=PHI1,PHI1<=N}>
```

Au contraire, si l'on utilise l'algorithme décrit dans la section 2.2, on obtient la région $\widetilde{R}$:

```
<WORK(PHI1,PHI2)-W-MUST-
     {1<=I,I<=N,1<=PHI1,PHI1<=N,PHI2==K+1}>
```

car le transformer de l'instruction `CALL INC1(K)` est `T(K) {K==K+1}`. La région R_0 est alors :

```
<WORK(PHI1,PHI2)-W-MUST-
     {1<=I,I<=N,1<=PHI1,PHI1<=N,K<=PHI2,PHI2<=K+1}>
```

qui représente avec précision les éléments de tableaux effectivement référencés.

3. Autres travaux

Les régions fournissent une représentation très générale pour résumer les références aux éléments

3. Utilisation des équations où le coefficient multiplicateur de la variable selon laquelle on substitue est 1 ou -1.

4. Et non les effets sur leur valeur, soit : $\widetilde{T}_k^{-1} : R_{k+1} \longrightarrow \widetilde{R}_k = proj_{\sigma_k}(R_{k+1})$

de tableaux. La contre-partie est le coût théorique élevé des opérations d'intersection et d'union.

D'autres travaux [4, 6, 2] se sont intéressés au calcul de résumés approchés des effets des procédures sur les tableaux. Ces approches établissent un compromis entre la variété des formes d'ensembles d'éléments de tableaux qu'elles permettent de représenter, et le coût des opérations sur ces ensembles.

D'autres auteurs ont introduit des représentations exactes des références aux éléments de tableaux accédés. Li et Yew [9] conservent des listes d'*atomes* pour les représenter, ce qui revient à ne pas effectuer d'union, mais nécessite plus d'espace mémoire. Tang [12] représente l'image exacte des références aux éléments de tableaux sous la forme d'un problème de programmation en nombres entiers, sous des hypothèses assez fortes : toutes les variables scalaires entières intervenant dans les expressions d'indices de tableaux doivent s'exprimer linéairement les unes en fonction des autres, et ne doivent pas varier dans le corps des boucles. Enfin, divers algorithmes de privatisation ou d'expansion de tableaux ont été proposés [5, 10, 11, 14].

Mais aucun de ces auteurs ne donne de précision sur le comportement de leur méthode en présence de variables scalaires entières variant à l'intérieur du corps des boucles et des procédures analysées. Havlak et Kennedy indiquent juste dans [6] que si une procédure modifie une variable scalaire entière dont dépendent les indices de tableaux, les ensembles d'éléments de tableaux correspondant prennent une valeur indéfinie.

4. Conclusion et travaux à venir

Un nouvel algorithme de calcul des régions résumées d'une séquence linéaire d'instructions complexes a été présenté dans cet article. Il prend en compte les effets des instructions et des procédures sur les valeurs des variables scalaires entières. Cela permet d'une part d'obtenir des régions plus précises et, d'autre part de conserver leurs approximations `MUST` dans un plus grand nombre de cas.

Une deuxième étape, en cours d'achèvement, a consisté à définir le calcul des régions pour les autres structures de contrôle du langage FORTRAN, en tenant compte de leur sémantique.

Enfin, ces régions seront utilisées pour implémenter un algorithme de privatisation de tableaux, destiné à améliorer le taux de parallélisation de PIPS [3].

Références

1. Béatrice Apvrille. PIPS : Amélioration du calcul des régions – rapport d'avancement. Rapport Interne E/181/CRI, CRI, École des Mines de Paris, janvier 1994.
2. V. Balasundaram and Ken Kennedy. A technique for summarizing data access and its use in parallelism enhancing transformations. In *ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 41–53, June 1989.
3. W. Blume and R. Eigenmann. Performance analysis of parallelizing compilers on the Perfect Benchmarks programs. *IEEE Transactions on Parallel and Distributed Systems*, 3(6):643–656, November 1992.
4. D. Callahan and Ken Kennedy. Analysis of interprocedural side effects in a parallel programming environment. *Journal of Parallel and Distributed Computing*, 5:517–550, 1988.
5. Paul Feautrier. Array expansion. In *International Conference on Supercomputing*, pages 429–441, July 1988.
6. Paul Havlak and Ken Kennedy. An implementation of interprocedural bounded regular section analysis. *IEEE Transactions on Parallel and Distributed Systems*, 2(3):350–360, July 1991.
7. François Irigoin. Interprocedural analyses for programming environments. In *Workshop on Environments and Tools for Parallel Scientific Computing*, September 1992.
8. François Irigoin, Pierre Jouvelot, and Rémi Triolet. Semantical interprocedural parallelization : An overview of the PIPS project. In *International Conference on Supercomputing*, June 1991.
9. Z. Li and P.-C. Yew. Efficient interprocedural analysis and program restructuring for parallel programs. *ACM SIGPLAN Notices*, 23(9):85–99, 1988.
10. Zhiyuan Li. Array privatization for parallel execution of loops. In *International Conference on Supercomputing*, pages 313–322, July 1992.
11. Dror E. Maydan, Saman P. Amarasinghe, and Monica S. Lam. Array data-flow analysis and its use in array privatization. In *Symposium on Principles of Programming Language*, January 1993.
12. Peiyi Tang. Exact side effects for interprocedural dependence analysis. In *International Conference on Supercomputing*, pages 137–146, July 1993.
13. Rémi Triolet. *Contribution à la parallélisation automatique de programmes Fortran comportant des appels de procédures.* PhD thesis, Université Paris VI, 1984.
14. Peng Tu and David Padua. Automatic array privatization. In *Language and Compilers for Parallel Computing*, August 1993.