RenPar'7,
Actes des 7es Rencontres
Francophones du Parallélisme,
PIP–FPMs Mons, Belgique, 30 mai–2 juin 1995
Dekeiser, Libert et Manneback Editeurs

Analyse de Flot de Données : Régions de Tableaux IN et OUT

Béatrice CREUSILLET, creusillet@cri.ensmp.fr

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

Résumé

Deux nouveaux types d'ensemble d'éléments de tableaux, ainsi que leur calcul, sont introduits. Les régions IN et OUT représentent l'ensemble des éléments de tableaux dont la valeur est importée et exportée par l'instruction ou la procédure considérée. Les applications possibles sont nombreuses : compilation des communications pour les machines à passage de message, privatisation de tableaux, optimisation à la compilation de l'utilisation des caches ou des mémoires locales dans les machines à hiérarchie de mémoires,...

1. Introduction

La compilation des programmes scientifiques en vue de leur exécution sur des machines massivement parallèles, ou à hiérarchie de mémoires nécessite une analyse intra- et inter-procédurale précise du flot des éléments de tableaux.

Une technique d'analyse récente [7] a ouvert des perspectives intéressantes dans ce domaine, car elle permet de connaître de manière exacte le flot des éléments de tableaux dans les programmes à contrôle statique. Cette dernière contrainte a été levée [9, 6], mais au prix d'une perte d'exactitude dans l'analyse. De plus, la complexité de cette méthode la rend difficilement applicable à des programmes de taille réelle, et elle ne permet pas une analyse interprocédurale.

D'autres méthodes consistent à calculer des résumés approchés des effets des procédures sur les éléments de tableaux [10, 4]. Elles permettent de traiter des programmes réels comportant des appels de procédure grâce à leur faible complexité pratique. Mais elles pèchent par leur imprécision, les analyses étant insensibles au flot de contrôle du programme et au flot des données scalaires entières apparaissant dans les expressions d'indice des tableaux.

Dans le cadre du paralléliseur PIPS [8], nous avons étendu l'approche de Triolet [10] sur les régions de tableau de manière à calculer les effets résumés *exacts* des instructions et des procédures sur les éléments de tableaux [1, 2]. Ces régions `READ/WRITE` sont utilisées pour compiler efficacement le langage HPF [5], mais s'avèrent insuffisantes pour d'autres optimisations comme la privatisation de tableau.

Nous introduisons donc deux nouveaux types de région exacte, les régions `IN` et `OUT`, qui représentent respectivement les ensembles d'éléments de tableaux dont la valeur est *importée* ou *exportée* par le fragment de programme considéré. Dans le cadre d'une machine massivement parallèle, ces régions permettent de calculer les communications à générer avant et après l'exécution d'une portion de code. Elles permettent également de privatiser des sections de tableaux [2]. Pour une machine à hiérarchie de mémoires, elles indiquent les données qui seront (ré)-utilisées, et qui devront donc, autant que possible, être préchargées ou conservées dans les caches ou les mémoire locales ; tandis que les éléments de tableaux référencés, mais n'apparaissant pas dans les regions `IN` et `OUT`, devront être traités comme des variables locales.

Cet article est organisé de la façon suivante. Une première section rappelle ce que sont les régions exactes. Les régions `IN` et `OUT` sont ensuite introduites dans une deuxième partie. Leur calcul est détaillé pour certaines constructions du langage FORTRAN dans les sections 4 et 5. Nous conclurons par l'état actuel de l'implantation.

2. Régions, régions exactes

Une région est un ensemble d'éléments de tableau décrit par des équations et inéquations affines formant un polyèdre convexe [10]. D'autres

caractéristiques ont été introduites pour représenter les effets des instructions et des procédures sur les éléments de tableaux :

- l'*action* réalisée sur les éléments de la région ; READ (R) pour une utilisation ou WRITE (W) pour une définition ;
- l'*approximation* de la région ; MAY si la région est une surapproximation de l'ensemble des éléments de tableaux effectivement référencés ; MUST si la région représente exactement cet ensemble (région *exacte*).

Par exemple, la région :

$$\texttt{<A(}\phi_1\texttt{,}\phi_2\texttt{)-W-MUST-\{}\phi_1\texttt{==I,}\phi_1\texttt{==}\phi_2\texttt{\}>}$$

où ϕ_1 et ϕ_2 représentent respectivement la première et la deuxième dimension de A, correspond à une référence en écriture à l'élément A(I,I).

Lorsque l'on a plusieurs références à des éléments d'un même tableau, la région résumée s'obtient en fusionnant les régions initiales. Or, l'union de deux polyèdres convexes n'est pas, en général, un polyèdre convexe. L'opérateur $\bar{\cup}$ utilisé est donc l'enveloppe convexe. Celle-ci peut contenir des points n'appartenant pas aux polyèdres de départ. La région résultante est alors une région MAY. Par exemple, pour A(I), A(I-1) et A(I+1), la région résumée est :

$$\texttt{<A(}\phi_1\texttt{)-R-MUST-\{I-1<=}\phi_1\texttt{, }\phi_1\texttt{<=I+1\}>}$$

qui contient exactement les éléments de départ. Par contre, pour A(I-1) et A(I+1), on obtient :

$$\texttt{<A(}\phi_1\texttt{)-R-MAY-\{I-1<=}\phi_1\texttt{, }\phi_1\texttt{<=I+1\}>}$$

C'est une région MAY car elle contient l'élément supplémentaire A(I).

Les autres opérateurs dont nous aurons besoin sont l'intersection et la différence de deux régions. L'intersection de deux polyèdres convexes étant un polyèdre convexe, l'intersection de deux région MUST est encore une région MUST. Par contre, la différence de deux polyèdres convexes n'est pas forcément un polyèdre convexe. Dans ce cas, l'opérateur $\ominus$ choisi donne une surapproximation de la différence des deux régions initiales, et la région résultante est une région MAY.

```
1.       K = FOO()
   C <W(φ1,φ2)-W-MUST-{1<=φ1<=N,K<=φ2<=K+N}>
   C <W(φ1,φ2)-R-MUST-{1<=φ1<=N,K<=φ2<=K+N}>
2.       DO I = 1,N
   C <W(φ1,φ2)-W-MUST-{1<=φ1<=N,φ2==K,1<=I<=N}>
3.          DO J = 1,N
   C <W(φ1,φ2)-W-MUST-{φ1==J,φ2==K,1<=J<=N, 1<=I<=N}>
4.             W(J,K) = J + K
            ENDDO
5.          K = K + 1
   C <W(φ1,φ2)-W-MUST-{1<=φ1<=N,φ2==K,1<=I<=N}>
   C <W(φ1,φ2)-R-MUST-{1<=φ1<=N,K-1<=φ2<=K,1<=I<=N}>
6.          DO J = 1,N
   C <W(φ1,φ2)-W-MUST-{φ1==J,φ2==K,1<=J<=N,1<=I<=N}>
7.             W(J,K) = J*J - K*K
   C <W(φ1,φ2)-R-MUST-{φ1==J,K-1<=φ2<=K,1<=J<=N,1<=I<=N}>
8.             A(I) = A(I)+W(J,K)+W(J,K-1)
            ENDDO
         ENDDO
```

FIG. 1 - READ/WRITE regions

Les contraintes linéaires définissant une région peuvent comporter des variables du programme dont la valeur dépend de l'état mémoire[1] considéré (ex. ϕ_1==I). Nous dirons alors que la region est exprimée dans cet état mémoire. Pour appliquer un opérateur à deux régions, il faut qu'elles soient exprimées dans le même état mémoire. Nous noterons $\widetilde{T}_{\sigma_1 \to \sigma_2}$ l'opérateur permettant de transformer une région exprimée dans l'état σ_1 en une région équivalente exprimée dans l'état σ_2.

La figure 1 donne un exemple de programme avec ses régions READ et WRITE. L'utilisation de ces régions ne permettrait pas de conclure que le tableau W est privatisable, c'est-à-dire que les éléments qui sont définis à une itération ne sont utilisés que lors de la même itération.

3. Régions IN et OUT

La région IN d'une instruction complexe contient les éléments de tableaux importés, c'est-à-dire dont la valeur est lue avant d'être éventuellement réinitialisée au sein de cette même instruction. Par exemple, dans la séquence d'instructions (7,8) de la figure 1, seul W(J,K-1) est importé car W(J,K) est défini dans l'instruction 7 avant d'être utilisé par l'instruction 8.

1. On rappelle qu'un état mémoire associe une valeur à chaque variable du programme pour une exécution donnée, et pour un instant donné de cette exécution.

La région OUT d'une instruction complexe représente les éléments de tableaux *vivants* ou exportés, c'est-à-dire qui sont définis par cette instruction, et utilisés dans les instructions suivantes. Dans notre exemple, l'instruction 7 exporte W(J,K) vers l'instruction 8, mais une itération I n'exporte aucun des éléments qu'elle définit vers les itérations suivantes.

La propagation des régions IN sur le graphe de contrôle et le graphe des appels du programme est ascendante, tandis que celle des régions OUT est descendante. Ainsi, la région IN d'une instruction complexe (DO, IF) est calculée à partie de ses composantes (corps, branche gauche ou droite) ; tandis que les régions OUT des instructions d'une séquence sont calculées à partir de la région OUT de la séquence (la région OUT du programme principal étant l'ensemble vide).

Nous nous limiterons dans le cadre de cet article à la description du calcul interprocédural pour une instruction ou une séquence d'instructions.

4. Calcul des régions IN

La région IN d'une *instruction d'affectation* est identique à sa région READ, car les valeurs des éléments de tableau référencés en lecture ne peuvent provenir de l'affectation elle-même.

Nous cherchons la région IN_B associée la *séquence d'instructions* $B = S_1, \ldots, S_n$ et exprimée dans l'état mémoire σ_B précédant l'exécution de B. C'est l'ensemble des éléments de tableau qui sont lus et dont la valeur provient des instructions précédant B.

Pour chaque instruction S_k, nous supposons connues ses régions W_k et IN_k, exprimées dans l'état σ_k précédant S_k ($\sigma_1 = \sigma_B$). Nous noterons IN'_k la région IN correspondant à la séquence $S_k, \ldots, S_n$. La région IN_B est alors définie par :

$$\begin{cases} IN'_n = IN_n \\ IN'_k = IN_k \,\overline{\cup}\, [\, \widetilde{T}_{\sigma_{k+1}\to\sigma_k}(IN'_{k+1}) \;\ominus\; W_k \,] \\ IN_B = IN'_1 \end{cases}$$

L'opérateur $\widetilde{T}_{\sigma_{k+1}\to\sigma_k}$ permet ici d'exprimer la région IN correspondant à $S_{k+1}, \ldots, S_n$ dans le même état mémoire σ_k que les régions WRITE et IN de S_k. $\widetilde{T}_{\sigma_{k+1}\to\sigma_k}(IN'_{k+1}) \;\ominus\; W_k$ représente la région importée par la sous-séquence $S_{k+1}, \ldots, S_n$, mais non définie par l'instruction S_k. La fusion avec IN_k donne donc bien l'ensemble des éléments lus par la séquence $S_k, \ldots, S_n$ avant d'être éventuellement redéfinis par cette même séquence.

Reprenons l'exemple de la séquence d'instructions (7,8) de la figure 1. Nous avons :

$$\begin{array}{lcl} IN_7 & = & \varnothing \\ W_7 & = & \texttt{<W(}\phi_1\texttt{,}\phi_2\texttt{)-W-MUST-\{}\phi_1\texttt{==J,}\phi_2\texttt{==K\}>} \\ IN_8 & = & \texttt{<W(}\phi_1\texttt{,}\phi_2\texttt{)-IN-MUST-\{}\phi_1\texttt{==J,K-1<=}\phi_2\texttt{<=K\}>} \end{array}$$

ce qui donne :

$$\begin{array}{lcl} IN'_8 & = & IN_8 \\ & = & \texttt{<W(}\phi_1\texttt{,}\phi_2\texttt{)-IN-MUST-\{}\phi_1\texttt{==J,K-1<=}\phi_2\texttt{<=K\}>} \\ IN'_7 & = & IN_7 \,\overline{\cup}\, (IN'_8 \;\ominus\; W_7) \\ & = & \texttt{<W(}\phi_1\texttt{,}\phi_2\texttt{)-IN-MUST-\{}\phi_1\texttt{==J,}\phi_2\texttt{==K-1\}>} \end{array}$$

La séquence d'instructions (7,8) importe donc bien uniquement l'élément W(J,K-1).

5. Calcul des régions OUT

La propagation des régions OUT est descendante. Nous supposons donc connue la région OUT_B associée à la *séquence d'instructions* B, et exprimée dans l'état mémoire précédant celle-ci. Nous cherchons les régions OUT_k associées à chacune des instructions composant la séquence. Ces régions contiennent les éléments définis par l'instruction considérée, et exportés soit vers les instructions suivantes de la séquence, soit vers l'extérieur de la séquence.

Nous noterons OUT'_k l'ensemble des éléments de tableaux définis par les instructions $S_1, \ldots, S_k$, et dont la valeur est réutilisée *après* l'exécution de B. Les équations permettant de calculer les OUT_k sont alors :

$$\begin{cases} OUT'_n = \widetilde{T}_{\sigma_B\to\sigma_n}(OUT_B) \\ OUT_n = W_n \cap OUT'_n \end{cases}$$

et, $\forall\, k \in [1..n-1]$,

$$\begin{cases} OUT'_k = \widetilde{T}_{\sigma_{k+1}\to\sigma_k}(OUT'_{k+1} \ominus W_{k+1}) \\ OUT_k = W_k \cap [\, OUT'_k \,\overline{\cup}\, \widetilde{T}_{\sigma_{k+1}\to\sigma_k}(IN'_{k+1}) \,] \end{cases}$$

OUT_B est la région OUT associée à B, et donc à $S_1, \ldots, S_N$, et exprimée dans l'état mémoire précédant l'exécution de B (σ_B). $\widetilde{T}_{\sigma_B\to\sigma_n}(OUT_B)$ est donc la région associée à $S_1, \ldots, S_N$, mais exprimée dans l'état mémoire σ_n précédant S_n, ce qui est la définition de OUT'_n. Nous avons donc bien :

$$OUT'_n = \widetilde{T}_{\sigma_B\to\sigma_n}(OUT_B)$$

La restriction à la sous-séquence $S1, \ldots, S_k$ de la région `OUT` de B, contient les éléments exportés à l'extérieur de B par la séquence $S1, \ldots, S_{k+1}$, moins les éléments déjà exportés par l'instruction S_{k+1} ; le tout exprimé dans l'état mémoire σ_k, soit :

$$OUT'_k = \widetilde{T}_{\sigma_{k+1} \to \sigma_k}(OUT'_{k+1} \ominus W_{k+1})$$

Enfin, la région `OUT` de l'instruction S est la région écrite par S et exportée vers l'extérieur de B, $W_k \cap OUT'_k$, à laquelle il faut ajouter la région écrite par S et exportée vers $S_{k+1}, \ldots, S_n$, soit $W_k \cap \widetilde{T}_{\sigma_{k+1} \to \sigma_k}(IN'_{k+1})$. Nous avons finalement :

$$OUT_k = W_k \cap [\, OUT'_k \,\overline{\cup}\, \widetilde{T}_{\sigma_{k+1} \to \sigma_k}(IN'_{k+1}) \,]$$

Comme illustration, considérons le corps de la deuxième boucle `J`, avec $OUT_B = \varnothing$ (ce qui signifie que le tableau `W` n'est pas réutilisé en dehors de la boucle). Nous avons:

$$\begin{array}{rcl} OUT_8 & = & OUT'_8 = OUT_B = \varnothing \\ OUT'_7 & = & (OUT'_7 \ominus W_7 = \varnothing \\ OUT_7 & = & W_7 \cap [\, OUT'_7 \,\overline{\cup}\, IN'_7 \,] \\ & = & W_7 \cap IN_7 \\ & = & \texttt{<W(}\phi_1\texttt{,}\phi_2\texttt{)-W-MUST-\{}\phi_1\texttt{==J,}\phi_2\texttt{==K\}>} \\ & \cap & \texttt{<W(}\phi_1\texttt{,}\phi_2\texttt{)-IN-MUST-\{}\phi_1\texttt{==J,K-1<=}\phi_2\texttt{<=K\}>} \\ & = & \texttt{<W(}\phi_1\texttt{,}\phi_2\texttt{)-OUT-MUST-\{}\phi_1\texttt{==J,}\phi_2\texttt{==K\}>} \end{array}$$

Ce qui signifie que l'instruction 7 exporte `W(J,K)`.

On montrerait de même que si la région `OUT` de la boucle `I` est vide, alors celle du corps dela boucle l'est aussi. Comme la région `IN` du corps est également vide, le tableau `W` serait privatisé [2].

6. Conclusion

Nous avons introduit deux nouveaux types de région de tableaux exacte : les régions `IN` et `OUT`, qui représentent les ensembles d'éléments de tableaux dont la valeur est importée ou exportée par le fragment de programme considéré.

L'implantation actuelle couvre l'ensemble des structures de contrôle du langage FORTRAN, ainsi que la propagation interprocédurale. Une première série d'expériences sur les programmes du Perfect Club [3] a montré que ces analyses étaient effectuées en un temps raisonnable, malgré la complexité théorique exponentielle des opérations portant sur les polyèdres.

D'autres expériences seront nécessaires pour déterminer si la représentation des régions sous forme de polyèdres est suffisamment précise pour l'analyse des programmes scientifiques, ou si une représentation sous forme d'unions finies de polyèdres est à envisager. Le coût en terme de temps de calcul et d'occupation mémoire serait certainement très important, mais cela permettrait d'éviter les approximations lors de l'utilisation des opérations d'union et de différence, et donc d'augmenter l'exactitude de l'analyse.

Bibliographie

1. Béatrice Apvrille. Calcul de régions de tableaux exactes. In *Rencontres Francophones du Parallélisme*, pages 65–68, June 1994.
2. Béatrice Apvrille-Creusillet. Régions exactes et privatisation de tableaux. Rapport de DEA Systèmes Informatiques, Université Paris VI, septembre 1994. Disponible sous `http://www.cri.ensmp.fr/~creusil`.
3. M. Berry et al. The PERFECT Club benchmarks: Effective performance evaluation of supercomputers. Technical Report CSRD-827, Center for Supercomputing Research and Development, University of Illinois, May 1989.
4. D. Callahan and K. Kennedy. Analysis of interprocedural side effects in a parallel programming environment. *Journal of Parallel and Distributed Computing*, 5:517–550, 1988.
5. Fabien Coelho. Compilation of I/O communications for HPF. In *Frontiers'95*, February 1995. Available via `http://www.cri.ensmp.fr/~coelho`.
6. Jean-François Collard. Automatic parallelization of while-loops using speculative execution. *International Journal of Parallel Programming*, 23(2):191–219, 1995.
7. Paul Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1):23–53, September 1991.
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. Vadim Maslov. Lazy array data-flow analysis. In *Symposium on Principles of Programming Language*, pages 311–325, January 1994.
10. Rémi Triolet. Contribution à la parallélisation automatique de programmes fortran comportant des appels de procédures. Thèse de doctorat, Université Paris VI, 1984.