Analyse de dépendances
Introduction à la parallélisation automatique

Lectures et avertissement

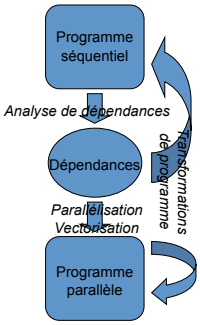
- La référence
  - Randy Allen and Ken Kennedy. *Optimizing compilers for modern architectures*. Morgan Kaufmann. 2002.
  - Plus particulièrement chapitres 2, 5, 6.
- Mais les aspects avancés (“méthodes polyédriques”) ne sont pas traités. Voir Paul Feautrier. Automatic Parallelization in the Polytope Model http://www.prism.uvsq.fr/rapports/1996/document_1996_8.ps. Synthèse + bibliographie
- Un cours raisonnablement approfondi demanderait plusieurs séances.
- TOUS LES EXEMPLES SONT DANS LE TD

Plan

- Introduction
- Dépendances
- Parallélisation et transformations

Introduction

- Méthodes systématiques et automatisables
  - Analyse d'un programme séquentiel, fonctionnel, parallèle
  - Pour la définition d'un ordonnancement compatible avec la sémantique du programme
  - Exprimable dans une syntaxe parallèle
- Aide à la parallélisation manuelle.
- Les paralléliseurs automatiques et leurs limites.



4

Syntaxe parallèle

- Constructions séquentielles + doall
  - Par(seq)

    doall i = 1, n

    <Bloc d'instructions séquentiel (i)>

    end do
  - Exécutions parallèle de B1, …, Bn
  - Chaque Bi conserve son séquencement
  - Synchronisation à la fin du doall
  - Traduction immédiate en espace d'adressage unique
- Instructions vectorielles

5

Programme séquentiel et nids de boucles

- Programme séquentiel
  - Boucles et séquences : pas de conditionnelles, ni de procédures
  - Instruction : occurrence textuelle
  - Opération : occurrence dynamique. Exemple A(1, 3, 2)=….

```
      DO i=1, N
        DO j=i, N+1
          DO k=j-i, N
(s)         A(i, j,k) = …
(t)         B(i,j,k) = …
          ENDDO
        ENDDO
        DO r=1, N
(u)       C(i,r) = …
        ENDDO
      ENDDO
```

Nid de boucles parfaitement imbriqué

Instruction (statique)

Nid de boucles imparfaitement imbriqué

6

Introduction

- Contexte
  - Nids de boucles
- Grain très fin
  - Vectorisation
  - SIMD
  - Machines parallèles si grands vecteurs
- Grain moyen
  - Transcription directe en espace d'adressage unique

```
do i=1,N
    do j =1, N
        A(i,j) = B(i,j) +1
    end do
end do

do i=1,N
    A(i,1:N) = B(i,1:N) +1
end do

doall i=1,N
    do j =1, N
        A(i,j) = B(i,j) +1
    end do
end do
```

7

Dépendances

- Soient s et t deux opérations d'un programme P.
- Collision : s et t sont en collision si
  - s et t accèdent au même emplacement mémoire et l'une au moins écrit.
- Dépendance : il existe une dépendance de s vers t si
  - s et t sont en collision

    ET
  - s avant t dans l'exécution de P

  Notation : s -> t
- L'ensemble des dépendances définit le *graphe de dépendances développé* du programme

8

Typologie

s -> t

- Dépendance de flot - RAW - dépendance vraie

  s écrit, t lit
- Anti-dépendance - WAR

  s lit, t écrit
- Dépendance de sortie - WAW

  s et t écrivent

9

Parallélisation par tri topologique

- TD 7.1
- Pas directement utilisable pour les boucles

10

Repères d'instructions et prédicat de séquencement

- Repère d'instruction
  - décrit une opération dans un nid de boucles
  - (nom_inst, i1, i2, …, in) où i1, i2, …, in sont les indices des boucles qui englobent l'instruction
- Prédicat de séquencement

  s, t deux instructions englobées dans n boucles

  L' ordre d'exécution des opérations est :

  (s, i1, i2, …, in) << (t, i'1, i'2, …, i'n) ssi

  (i1, i2, …, in) < (i'1, i'2, …, i'n) dans l'ordre lexicographique

  OU

  {(i1, i2, …, in) = (i'1, i'2, …, i'n) ET s avant t dans l'ordre syntaxique de P}
- Exemples : TD 7.2.1

11

Niveau des dépendances et GDR

- Niveau de la dépendance

  La dépendance

  (s, i1, i2, …, in) -> (t, i'1, i'2, …, i'n)

  est de niveau k si k est le premier indice tel que ik < i'k
  - Dépendance inter-itération - loop carried dependency

  Est de niveau $\propto$ si (i1, i2, …, in) = (i'1, i'2, …, i'n)
  - Dépendance intra-itération - loop-independent dependency
- Graphe de dépendance réduit (GDR)
  - Multi-graphe
  - Nœuds = instructions (statiques)
  - Arcs = dépendances étiquetées par leur niveau
- Exemples : TD 7.2.2

12

Equivalence

- *«Définition»* Deux programmes sont équivalents s'ils produisent le même résultat
- Transformation
  - Ré-ordonnancement, ré-indexation, parallélisation,…
  - Les instances dynamiques ne changent pas : pas d'instructions ajoutées ou supprimées
  - La structure de contrôle et les indices peuvent changer
- « *Proposition* » Une transformation d'un programme P qui préserve le graphe de dépendances développé fournit un programme équivalent à P
- La réciproque est fausse TD 7.4.1

13

Instructions vectorielles

- Pour ce cours on se limitera à

  X(a:b:c) = OP (Y1(a:b:c),…, Yk(a:b:c))

Équivalent à

```
doall i=a, b
  X(i) = OP (Y1(i), …, Yk(i))
end do
```

- Opération « par coordonnées » - α−extension

14

Doall et prédicat de séquencement

- Filtrer l'ordre lexicographique.
- Voir TD 7 - appendice

15

Parallélisation de boucles

- *Une boucle do de profondeur k est équivalente à une boucle doall si elle ne porte pas de dépendance de niveau k.*
- Preuve :
  - Les ensembles Read et Write des opérations ne sont pas modifiés
  - Le prédicat de séquencement n'est modifié qu'au niveau k
  - Donc le GDD n'est pas modifié
  - Détails dans TD 7 - appendice

16

Tranformations de boucles

- Une dépendance inter-itérations interdit la parallélisation.
- Des transformations (valides) peuvent faire apparaître de nouvelles opportunités de parallélisation.

17

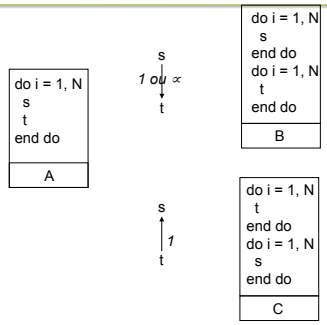
Distribution de boucles

- Boucle à un seul niveau :
- Exemples TD 7.3

  Une boucle contenant au moins deux instructions est équivalente à la séquence de deux boucles s'il n'existe pas de cycle de dépendances dans le GDR.
- Le séquencement doit suivre la direction des dépendances

18

Distribution de boucles



19

Distribution de boucles

- En distribuant la boucle, le séquencement correct des opérations en dépendance de niveau 1 est garanti : s'il existe (s,i) -> (t,i') au niveau 1 ou ∝ dans A, on a i < i' ou i = i'. Dans B, (s,i) s'exécute avant (t,j) quels que soient i et j.
- La distribution ne crée pas de dépendance nouvelle.
  Les seules dépendances nouvelles possibles dans B sont (s,i) -> (t,i') avec i > i'. Si une telle dépendance existait, la dépendance inverse (flot et anti échangés) existerait dans A de t vers s, donc le GDR de A comporterait un cycle.
- Même raisonnement pour C.
- Cette transformation ne modifie pas les instructions

20

L'algorithme d'Allen-Kennedy

Préliminaires

- G un graphe. Une composante fortement connexe de G est un ensemble de sommets tel qu'il existe un chemin entre deux quelconques de ces sommets.
- Il existe des algorithmes pour trouver les composantes fortement connexes maximales (CFCM) $\{\pi_1, \pi_2, \ldots, \pi_m\}$ –Tarjan
- Les arcs de G induisent naturellement un graphe sur $\{\pi_1, \pi_2, \ldots, \pi_m\}$ noté G_π
- G_π est un DAG

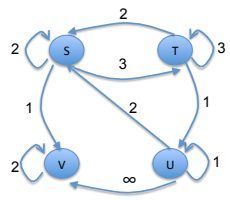
21

Principe de l'algorithme: distribution des boucles

- Calculer les CFCM du GDR.
- Ordonner les CFCM suivant l'ordre topologique.
- Pour i=1: nb_CFCM, créer une boucle dont le corps est la restriction de la boucle initiale aux instructions de π_i.

Exemple

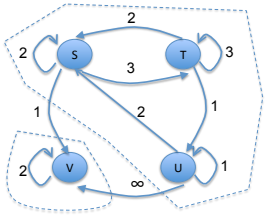
GDR




```
do i
   do j
      do k
            S
            T
      end do
      U
      V
   end do
end do
```


Exemple

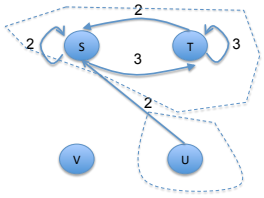
GDR




```
do i
   AK (π1,2)
end do
doall i
   do j
      AK (π2,3)
   end do
end doall
```

Exemple

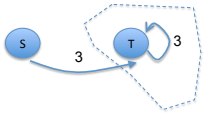
GDR




```
do i
   doall j
      U
   end doall
   do j
      AK(π3, 3)
   end do
end do
doall i
   do j
      V
   end do
end doall
```


Exemple

GDR




```
do i
   doall j
      U
   end doall
   do j
      doall k
        S
      end doall
      do k
        T
      end do
   end do
end do
doall i
   do j
      V
   end do
end doall
```


L'algorithme d'Allen-Kennedy

```
AK(G, k) {
   Remove all edges of level < k.
   Compute the SCCs of G.
   For each CFCM C in topological order
      if C has one vertex S and no edge
        generate all loops as DOALL and generate the code for S.
      else
        n = level_min(C).
        generate (n - k) DOALL loops and a DO loop for level n
        call AK(C, n + 1)
   EndIf
EndForEach
}
```

L'algorithme d'Allen-Kennedy

- On peut montrer qu'il est optimal dans la classe des algorithmes qui ne transforment pas le code des instructions
- D'autres transformations peuvent créer de meilleures opportunités de parallélisme
- Voir par exemple http://perso.ens-lyon.fr/alain.darte/DEA04/loops.pdf

28

Autres transformations

- Echange de boucles – TD 7.4
  - But : Modifier la granularité du parallélisme
  - Moyen : Modifier le parcours de l'espace d'itération pour faire apparaître du parallélisme
- Torsion de boucles (Loop skewing) _ TD 7.5
  - But : faire apparaître du parallélisme.
  - Moyen : Modifier le parcours de l'espace d'itération pour faire apparaître du parallélisme
- Et autres modifications du parcours de l'espace d'itération…
- Agrégation de boucles
  - But : diminuer le surcoût de synchronisation
- Transformations du code séquentiel
  - But : faire apparaître du parallélisme.
- Problèmes
  - Critères d'optimalité : nombre de boucles parallèles, accélération
  - Atteindre l'optimum

29

Limites de l'analyse de dépendances

- L'analyse automatique implique
  - La résolution de systèmes linéaires en nombres entiers
  - D'où des approximations, qui doivent être conservatives
  - C'est une limite réelle pour les méthodes de transformation avancées
- La compilation séparée bloque l'analyse de dépendances inter-procédurales
- Les accès par indirection interdisent toute analyse de dépendances
  - Pointeurs
  - Indices tableaux : A(L(i))

30