Optimisation des boucles, II/II

Compilation avancée et optimisation de programmes

Paul Feautrier

ENS de Lyon
Paul.Feautrier@ens-lyon.fr

16 novembre 2005

Ordonnancement

- Soit E un ensemble de tâches, opérations, instructions ... à exécuter à l'aide d'un certain nombre de machines, opérateurs, processeurs ...
- Ordonnancer = fixer pour chaque opération la date de départ et l'opérateur utilisé.
- Représentation par table.

Opération	Durée	Date	Opérateur
`x = y + z`	1	0	+
`u = v + w`	2	0	*
`a = x + u`	1	2	+

Exécution parallèle = les dates de départ sont égales.

Ordonnancement d'un bloc de base

On se donne:

- La liste des opérations $T_i, i = 1, n$ et leur durée d_i.
- Les *contraintes de précédence* $T_i \text{ avant } T_j$ (généralisation des dépendances).
- Résoudre:

$$\begin{array}{rcl} & \min & L \\ \forall i : L & \geq & t_i \\ \forall i : t_i & \geq & 0 \\ \forall i, j, T_i \text{ avant } T_j : t_j & \geq & t_i + d_i \end{array}$$

- On peut utiliser la programmation linéaire, mais il y a plus simple.

Tri topologique

- Construire le graphe de la relation de précédence, qui est un DAG.
- Tant qu'il reste des sommets dans le graphe,
- Choisir un sommet t_i sans prédécesseur,
- Lui affecter la date

$$\max_{T_j \text{ avant } T_i} t_j + d_j.$$

  Le calcul est possible parce que tous les T_j ont déja été ordonnancé.
- Enlever T_i du graphe.

Contraintes de ressources

- L'ordonnancement ci-dessus ne tient pas compte des *contraintes de ressources*.
- r: un *type* de ressource (e.g. un additionneur flottant).
- N_r le nombre de ressources de type r.
- $T_i \in r$: T_i utilise une ressource de type r.

$$\forall i : \mathrm{Card}\,\{T_i \in r \mid t_i \leq t < t_i + d_i\}.$$

Complexité

- Résultat classique: le problème de l'ordonnancement sous contraintes de ressources est NP-complet dès qu'il y a un type avec plus de 2 ressources.
- On doit se contenter de solutions approchées ou utiliser la programamtion linéaire en nombres entiers:
- $X_{it} \in \{0, 1\},\ X_{it} = 1$ ssi $t_i = t$.
- $\forall t : \sum_t X_{it} = 1$.
- $t_i = \sum_t t.X_{it}$, permet d'écrire les contraintes de précédence.

$$\forall t, r : \sum_{T_i \in r} \sum_{k=t-d_i+1}^{t} X_{ik} \leq N_r.$$

Algorithmes de liste I/II

- **Principe**: on simule le fonctionnement d'un multiprocesseur *glouton*: un processeur n'est oisif que s'il n'y a pas de travail pour lui. On classe les opérations par ordre de priorité selon un critère arbitraire.
- **Notations**:
  - K_r la tache qui utilise actuellement la ressource r.
  - t_r: la date à laquelle K_r r a été lancée.
  - d_r: la durée d'exécution de K_r.

Algorithmes de liste, II/II

- Pour $t = 0, L$ (une borne supérieure de la durée de l'ordonnancement):
  - Pour toute ressource r:
    - Si $t < t_r + d_r$, la ressource est occupé, ne rien faire.
    - Si $t = t_r + d_r$, enlever la tache exécutée par r du graphe de précédence.
    - Si $t \geq t_r + d_r$, chercher la tache de plus haute priorité qui utilise r et n'a pas de prédécesseur.
    - Si on en trouve une, $t_r = t$ et r exécute cette tache.

Ordonnancement d'un nid de boucles

- Comme le nombre d'opérations peut être très grand (ou inconnu, ou infini), on ne peut plus tabuler l'ordonnancement.
- On doit représenter l'ordonnancement par une formule.
- Opération: $\langle S, i \rangle$, $i \in D_S$.
- Ordonnancement affine:

  $$\theta(S, i) = h_S.i + k_S,$$

  où h_S est un vecteur de même dimension que i, et où k_s est un scalaire, en général entiers.
- Première contrainte: $\theta(S, i) \geq 0$.
- NB: on peut utiliser des ordonnancements plus compliqués, par exemple affines par morceaux.

Contrainte de causalité

L'ordre associé à l'ordonnancement doit respecter les dépendances:

$$\langle S, \rangle \; \delta \; \langle T, j \rangle \Rightarrow \theta(S, i) + d(S) \leq \theta(T, j).$$

- Il s'agit ici de la relation de dépendance détaillée:

$$\begin{array}{rcl} i \in D_S, & & j \in D_T \\ f_S(i) & = & g_T(j), \\ i & \ll_p & j \end{array}$$

  - f_S, g_T : fonctions d'indice ;
  - $\ll_p$: ordre lexicographique à la profondeur p.

Un exemple I/III

Boucle de Lamport:

```
for(i=1; i<=n; i++)
    for(j=1; j<=n; j++)
        a[i][j] = 0.5*(a[i-1][j] + a[i][j-1]);
```

- Ordonnancement: $\theta(i, j) = \alpha i + \beta j$. Il n'y a pas besoin de terme constant.
- Durée de l'opération: 1.
- Dépendance: $(i, j) \rightarrow (i + 1, j)$ et $(i, j) \rightarrow (i, j + 1)$.
- Noter qu'aucune des boucles n'est parallèle.

Un exemple II/III

Résolution

$$\begin{aligned} \alpha i + \beta j + 1 &\leq \alpha(i+1) + \beta j, \\ \alpha i + \beta j + 1 &\leq \alpha i + \beta(j+1) \end{aligned}$$

- Soit $\alpha \geq 1, \beta \geq 1$.
- On a intérêt à prendre les plus petites valeurs possibles, donc: $\theta(i,j) = i + j$.

Un exemple III/III

Génération du code parallèle

- Rappel: la valeur de $i+j$ est la date à laquelle l'opération (i,j) doit être exécutée.
- Le temps doit avancer de sa valeur minimale, 2, à sa valeur maximale, $2n$.
- A chaque instant, on exécute en parallèle toutes les opérations telles que $i+j=t$.

```
for(t=2; t<=2*n; t++)
    //for(j=max(1, t-n); j <= min(n, t-1); j++)
        a[t-j][j] = 0.5 * (a[t-j-1][j] + a[t-j][j-1]);
```

- Vérifier: pas de dépendance dans la boucle parallèle.

Dépendances uniformes

- L'exemple de Lamport est le cas d'un nid de boucle parfait à dépendances uniformes: $\langle S, i\rangle \rightarrow \langle S, i + d_k\rangle,\ k = 1, n$.
- Conditions de causalité: $h_S.d_k \geq 1,\ k = 1, n$.
- Il y a toujours une solution, parce que les vecteurs d_k sont lexicopositifs. On peut trouver la meilleure par programmation linéaire.

Theorem

Un nid de boucle de profondeur ≥ 2 à dépendances uniformes contient toujours du parallélisme.

Démonstration.

Ranger les vecteurs de dépendance par ordre lexicographique croissant, et construire l'ordonnancement par étape. $\square$

Cas général

- La condition de causalité peut représenter jusqu'à $\mathrm{Card}\ D_S \times \mathrm{Card}\ D_T$ contraintes linéaires sur les coefficients des ordonnancements.
- Ce nombre peut être très grand, ou inconnu, ou même infini.
- Il faut trouver un moyen pour *résumer* ces contraintes en un ensemble fini.
- C'est possible parce que les contraintes sont linéaires.

Méthode des sommets

Patrice Quinton, circa. 1987

- La relation de dépendance détaillée définit un polyèdre dans le produit cartésien $D_S \times D_T$.

$$\begin{array}{rcl} i \in D_S, & & j \in D_T \\ f_S(i) & = & g_T(j), \\ i & \ll_p & j \end{array}$$

- Théorème de Minkowski: tout polyèdre est combinaison convexe d'un nombre fini de points, ses sommets.
- Pour qu'une fonction affine soit non négative dans un polyèdre, il faut et il suffit qu'elle soit non négative en ses sommets.
- Trouver les sommets du polyèdre des dépendances (algorithme de Chernikova).
- Ecrire la condition de causalité en ces points.
- Résoudre par programmation linéaire.

Méthode de Farkas I/II

Paul Feautrier, 1992

- Lemme de Farkas: pour qu'une fonction affine soit non négative dans un polyèdre, il faut et il suffit qu'elle soit combinaison affine positive des contraintes définissant le polyèdre:

$$Ax + b \geq 0 \Rightarrow cx + d \geq 0$$

est équivalent à:

$$cx + d = \lambda_0 + \lambda.(Ax + b) \;\; \lambda_0, \lambda \geq 0.$$

- antécédent: les contraintes définissant le polyèdre des dépendances.
- conséquent: le délai $h_T.j + k_T - h_S.i - k_S - d_S$.

Méthode de Farkas II/II

- Ecrire l'identité de Farkas.
- Identifier les coefficients de i et j et les termes constants.
- On obtient un système d'équations linéaires dont les inconnues sont les coefficents des ordonnancements h_S, k_S et les multiplicateurs de Farkas.
- Comme les inconnues sont positives, on doit résoudre par une méthode de programmation linéaire.
- On peut ajouter une fonction objectif (par exemple, la latence totale).

Exemple

```
for(i=0; i<n; i++)
    for(j=0; j<n; j++)
M:      c[i] += a[i][j]*b[j];
```

- Ordonnancement: $\theta(M, i, j) = \alpha i + \beta j$.
- Dépendance:

$$i \geq 0, j \geq 0, \; i' \geq 0, j' \geq 0$$

$$i = i', \; j' \geq j + 1.$$

$$\alpha i' + \beta j' - \alpha i - \beta j - 1 = \lambda_0 + \lambda_1 i + \lambda_2 j + \lambda_3 i' + \lambda_4 j' + \lambda_5 (j' - j - 1) + \mu (i' - i).$$

$$\alpha = \lambda_3 - \mu, \;\; -\alpha = \lambda_1 + \mu,$$

$$\beta = \lambda_4 + \lambda_5, \;\; -\beta = \lambda_2 - \lambda_5,$$

$$-1 = \lambda_0 - \lambda_5.$$

$$\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 0, \;\; \lambda_5 = \beta = 1 \;\; \alpha = 0.$$

Echec et Mat ...

```
for(i=0; i<n; i++)
    for(j=0; j<n; j++)
M:          s += a[i][j];
```

- Ordonnancement: $\theta(M, i, j) = \alpha i + \beta j$.
- Dépendance:

$$i \geq 0, j \geq 0, \;\; i' \geq 0, j' \geq 0$$

$$i' \geq i + 1,$$

$$i = i', \; j' \geq j + 1.$$

$$\alpha i' + \beta j' - \alpha i - \beta j - 1 = \lambda_0 + \lambda_1 i + \lambda_2 j + \lambda_3 i' + \lambda_4 j' + \lambda_5 (i' - i - 1).$$

$$\beta j' - \beta j - 1 = \mu_0 + \mu_1 i + \mu_2 j + \mu_3 j' + \mu_5 (j' - j - 1)$$

La première identité entraine $\beta = 0$ alors que la seconde entraine $\beta \geq 1$. Il y a contradiction.
Il n'y a pas d'ordonnancement affine légal.

Caches

Localité spatiale / temporelle

Associativité

Stratégie de remplacement

- LRU, FIFO, RANDOM.
- write througt, write back.

TLB

Mémoires locales, mémoire virtuelle