Composition
Mineure Calcul Parallèle

E. Goubault

30 mars 1998

Introduction

Au cours d'une résolution numérique de l'équation des ondes par un schéma implicite, on a à trouver la solution d'un système d'équations linéaires $Ax = b$. Après un algorithme de type pivot de Gauss que l'on n'étudiera pas ici, on se retrouve avec un système triangulaire supérieur $Ux = b$. On se propose ici de programmer la résolution parallèle de ce dernier système sous diverses hypothèses.

Soit donc à résoudre le système triangulaire supérieur

$$Ux = b$$

avec,

$$U = \begin{pmatrix} U_{1,1} & U_{1,2} & U_{1,3} & \cdots & U_{1,n} \\ 0 & U_{2,2} & U_{2,3} & \cdots & U_{2,n} \\ & & & \cdots & \\ 0 & 0 & \cdots & 0 & U_{n,n} \end{pmatrix}$$

et $U_{i,i} \neq 0$ pour tout $1 \leq i \leq n$.

On procède par "remontée" c'est à dire que l'on calcule successivement,

$$\begin{aligned} x_n &= \frac{b_n}{U_{n,n}} \\ x_i &= b_i - \frac{\sum_{j=i+1}^{n} U_{i,j} x_j}{U_{i,i}} \end{aligned}$$

pour $i = n-1, n-2, \cdots, 1$.

A) Remontée en mode SIMD

On écrit les équations précédentes en pseudo-code C de la façon suivante:

```
x[n]=b[n]/U[n,n];
for (i=n-1;i>=1;i--)
   {
      x[i]=0;
      for (j=i+1;j<=n;j++)
         L: x[i]=x[i]+U[i,j]*x[j];
      x[i]=b[i]-x[i]/U[i,i];
   }
```

où `n` est la taille de la matrice U représentée par le tableau `U[i,j]`, où `x[i]` (`i` entre 1 et `n`) représente le vecteur "inconnu" x et `b[i]`, le vecteur second membre du système d'équations linéaires.

On suppose dans cette première partie que l'on dispose de $n-1$ processeurs.

Graphe de dépendance dans l'espace d'itérations

1) Quel est le graphe de dépendance dans l'espace d'itérations du code C dans les boucles imbriquées, calculant la remontée ? On rappelle que ce graphe est le graphe de dépendance dans l'espace des `(i,j)` entre les différentes instances du corps de la boucle `L`. De quel(s) type(s) de dépendance(s) s'agit-il ?

Amélioration du code

Tel que le code séquentiel est écrit il parait difficile de le paralléliser en mode SIMD. On suppose que dans ce mode, on dispose d'une instruction `forall` qui a la même sémantique que l'instruction C `for` si ce n'est que le corps de boucle qui la suit, si ce corps n'a pas de dépendance pour les indices gérés par ce `forall`, est exécuté en parallèle et de façon synchrone sur chacun des processeurs.

2) Y a t-il une "rotation de code" (et autres transformations) que l'on peut effectuer qui permettrait de paralléliser le code séquentiel donné plus haut ? Reécrivez le code en conséquence.

Indication: On pourra comme dans le cas séquentiel commencer par calculer `x[n]`. Ensuite, l'idée sera de placer chacune des composantes des tableaux rentrant en compte pour le calcul de la ième équation linéaire ($1 \leq i \leq n-1$)[1] sur le ième processeur P_i. On pourra donc propager les valeurs déjà résolues des `x[j]`, `j`$\geq$`i`+1 aux processeurs P_k, $k \leq$`i`, qui effectueront une même étape de sommation élémentaire.

Complexité

3) En supposant que le coût de chaque opération élémentaire (comme +, -, /, *) est unitaire ainsi que le coût d'un accès à un élément de tableau qui n'est pas sur la mémoire que gère le processus appelant, et en négligeant le coût d'initialisation des boucles, quel est le gain entre le code parallélisé et le code séquentiel du début?

B) Remontée en mode MIMD

Dans cette partie, on veut rendre explicite les communications entre les processeurs qui gèrent chacun une partie du tableau `U`. De fait, on suppose que l'on a n processeurs $P_1, \cdots, P_n$ et que l'on répartit les tableaux de la façon suivante,

P_1	P_2	P_3	$\cdots$	P_{n-1}	P_n
$b[1]$	$b[2]$	$b[3]$	$\cdots$	$b[n-1]$	$b[n]$
$U[1,n]$	$U[2,n]$	$U[3,n]$	$\cdots$	$U[n-1,n]$	$U[n,n]$
$U[1,n-1]$	$U[2,n-1]$	$U[3,n-1]$	$\cdots$	$U[n-1,n-1]$	
$\cdots$	$\cdots$	$\cdots$	$\cdots$		
$U[1,1]$					

Code en PVM

[1] C'est celle qui sert à déterminer `x[i]` à partir des `x[j]` (`j`$\geq$`i`+1) lors de la remontée.

1) Programmer le code de remontée correspondant en PVM (on pourra s'inspirer de ce qui a été décrit en A)2)). On supposera que P_n est le maître qui contient au début tout le tableau `U` et le tableau `b` et crée ses esclaves $P_1, \cdots, P_{n-1}$. A la fin il doit attendre le résultat de tous ses esclaves.

Dans toute la suite du problème, on supposera que le coût de l'envoi et le coût de la réception (hors attente) est égal à la taille des données envoyées multipliée par α ($0 \leq \alpha \leq 1$) et ne dépend ni du processeur ni de l'historique des calculs.

Dépendance des tâches

Chacun des P_i contient donc un processus unique (que l'on peut aussi appeler P_i), qui va se charger de calculer $x[i]$.

2) Représentez le graphe de dépendances de tâches en y ajoutant les tâches de communication. Pour ce faire, on découpera le processus P_i en sous-tâches correspondant aux réceptions, sommations et envois de résultats partiels.

Complexité

3) Quel est le temps de calcul global du code PVM ? On supposera le coût de chaque opération élémentaire comme unitaire et on négligera tout ce qui est initialisation de boucle, `spawn` etc. Comment cela se compare-t-il avec A)3) ?

Ordonnancement dans le cas de deux processeurs

Le cas que nous avons étudié était idéal. On suppose dans cette question que l'on n'a pas autant de processeurs que de lignes dans la matrice U. Pour fixer les idées on choisit $p = 2$ processeurs. Les P_i apparaissant dans le tableau plus haut en début de la partie B) correspondent donc à des processus (ou des tâches).

4) Ordonnancez au mieux les tâches sur les 2 processeurs tout en respectant la localisation des données. On pourra s'inspirer de la méthode du cours (après avoir attribué des coûts aux sous-tâches) ou essayer a posteriori de justifier en quelques lignes l'optimalité de son choix.

Ordonnancement statique

5) On suppose ici que la matrice U a $n = kp$ lignes (où $k \geq 2$ et p est le nombre de processeurs).

a) Sans écrire le code, justifier de la complexité (temps global d'exécution) de la solution qui consiste à répartir les lignes de `U` sur les processeurs modulo `p` (donc les lignes `U[1,*]`, `U[1+p,*]`, $\cdots$, `U[1+(k-1)*p,*]` sur P_1 etc.).

b) On met maintenant des lignes adjacentes de `U` sur chacun des processeurs, c'est à dire que l'on découpe `U` en bandes horizontales. Ecrire un squelette de code PVM permettant de gérer cette décomposition de domaines. Comment faut-il choisir au mieux les tailles des domaines ? Comparer avec 5)a).

6) Question facultative: Donnez un principe, en quelques phrases, qui permettrait de faire de l'équilibrage de charges dynamique en PVM pour cette remontée (par exemple, si on avait autorisé d'autres applications indépendantes à consommer aléatoirement des ressources calcul pendant l'exécution du code).