Techniques de l'algorithmique et de la programmation parallèle

Master II Informatique Grenoble

30 Octobre 2003

Transformée de Fourier en parallèle

Un grand nombre d'applications utilisent la transformée de Fourier rapide (FFT). Un algorithme classique de calcul suit le schéma "papillon" (cf figure).

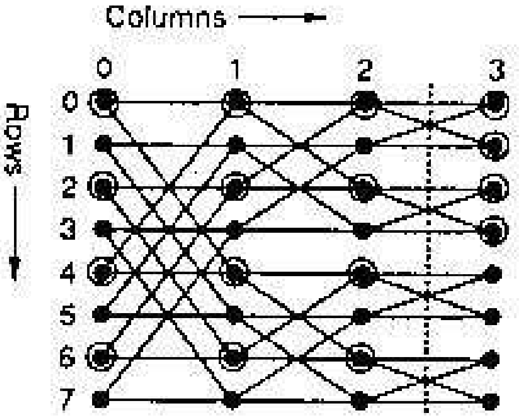

Figure 1: Graphe de précédence pour $n = 8$.

Soit $n = 2^q$ la taille du vecteur d'entrée. Le graphe de précédence du calcul est défini par $n(1 + \log n)$ nœuds (dans tout cet exercice, log désigne le logarithme en base 2), disposés selon une grille à n lignes et $1 + \log n$ colonnes. Les nœuds en colonne 0 sont les entrées du calcul ; ceux en colonne $\log n$ sont les sorties après les calculs qui interviennent à chaque nœud intermédiaire (on supposera que chaque noeud applique sur ses entrées la fonction CalculeFFT(), que l'on supposera de coût unitaire).
On suppose qu'à l'initialisation `Data(i,0)`, $0 \leq i < n$, contient le vecteur en entrée. L'algorithme séquentiel est le suivant :

```
Pour j:=1 à  log n faire
  Pour i:=0 à n-1 faire
    Data(i, j) := CalculeFFT( Data(i, j-1),  Data( Voisin(i, j), j-1 ) ) ;
```

où la primitive `Voisin(i, j)` prend en entrée deux entiers i et j et retourne l'entier $\bar{\imath}_j$ défini par :

$$\text{si } \; i = \sum_{l=0}^{\log n - 1} \alpha_l 2^l \; \text{ alors } \; \bar{\imath}_j = \sum_{l=0, l \neq \log n - j}^{\log n - 1} \alpha_l 2^l + (1 - \alpha_{\log n - j}) \times 2^{\log n - j}.$$

Autrement dit, $\forall \; 0 \leq i < n, \forall \; 1 \leq j \leq \log n$, le nœud (i, j) est relié aux deux nœuds $(i, j-1)$ et $(\bar{\imath}_j, j-1)$ où $\bar{\imath}_j$ est obtenu en complémentant à 2 le j-ème bit *de poids fort* de i écrit en base deux. Ainsi, on a :

- le nœud (i, j) ne dépend que des nœuds : $(i, j-1)$ et $(\texttt{Voisin}(i, j), j-1)$.
- le nœud (i, j) ne sert au calcul que des nœuds : $(i, j+1)$ et $(\texttt{Voisin}(i, j+1), j+1)$.

La parallélisation sur i étant triviale, une idée naturelle est de découper le graphe papillon en lignes. Le problème consiste à trouver une distribution astucieuse des lignes pour optimiser les communications.

1. Préliminaire. Calculer le temps parallèle et la complexité de communications. Quel est le nombre minimal de processeurs nécessaires ?

2. Distribution cyclique. On suppose que l'on répartit le papillon selon un schéma cyclique : le processeur $k, k = 0 \dots p-1$ reçoit la ligne numéro ($k \bmod p$).

2.1. Faire un schéma du graphe dans le cas $n = 8$ (graphe à 8 lignes et 4 colonnes) et $p = 2$ en indiquant quelles lignes sont sur quel processeur et en différenciant les arètes correspondant à des communications des autres arètes.

2.2 Montrer qu'avec cette distribution de lignes, seuls les calculs sur les colonnes de 1 à $\log(n/p)$ peuvent se faire localement, *i.e.* sans communication. *Indication :* tenir compte du fait que $2^{\log p} = p = 0 \pmod p$ et que c'est la plus petite puissance de 2 vérifiant cette propriété.

2.3. On modélise le coût d'une communication point-à-point selon la loi $g.N + s$ où s est le temps d'initialisation ("latence") de la communication, et g la bande passante (inverse du débit). N est le nombre de données transmises.

Calculer le temps que passe un processeur en communications. Calculer le temps passé en calculs. Conclusion ?

2.4. Écrire le squelette d'un programme MPI pour réaliser ce type de calcul. On supposera qu'au départ le processeur 0 (le maître) possède tout le vecteur `data` à n composantes.

a. écrire la répartition initiale. On pourra utiliser des primitives de communication globales type MPI.

b. écrire les boucles de calcul à l'aide de primitives "send" et "receive" suivantes :

- `send(i, data)` : effectue l'envoi vers i des données "data"
- `receive(i, result)` : reçoit depuis i les données dans "result"

3. Distribution par blocs. On suppose maintenant que l'on effectue une répartition des lignes par blocs : le processeur k reçoit les lignes $(n/p) \times k \dots (n/p) \times k + (n/p) - 1$.

3.1. Quelles étapes peuvent désormais se dérouler sans communication ?

3.2. En déduire le volume de communications et de calculs par processeur. Comparer à la distribution précédente.

4. Solution mixte. On suppose désormais que $p < \sqrt{n}$.

4.1. Compte tenu des résultats précédents, on s'intéresse à fusionner les deux schémas en effectuant un échange global ("all-to-all") après la $\frac{\log n}{2}$-ième étape. Expliquer pourquoi on fait ce choix compte tenu des questions précédentes. On précisera notamment la distribution initiale des données et la distribution du résultat final.
Préciser le temps parallèle et la complexité de communications avec $\sqrt{n}$ processeurs. Donner le nombre total d'opérations et le volume de communications; qu'en déduisez-vous ?

4.2. On se donne une primitive "pseudo-MPI" all2all :

```
all2all(envoi[0..p-1], recep[0..p-1])
```

Chacun envoie "envoi[i]" vers le processeur i et reçoit dans le tableau "recep" les valeurs que lui envoient les autres processeurs (recep[j] reçoit les données émises par le processeur j).
Écrire le squelette d'un programme MPI qui effectue le calcul selon ce schéma.

4.3. On modélise le coût d'un échange global par la formule $s + g.N$ où s et g ont le même sens que ci-dessus et N est le nombre de données envoyées par **un** processeur.
Calculer le coût des communications dans ce schéma, comparer avec les deux autres et conclure.