Exercises: Parallel merge and application to sort

Jean-Louis Roch

NB Ce problème est constitué de cinq parties indépendantes.

Dans tout le problème, on considère une machine parallèle de type CREW: les processeurs peuvent accéder en lecture à n'importe quelle donnée; mais les écritures sur une même case mémoire sont en exclusion mutuelle.
On s'intéresse dans la suite à des algorithmes parallèles de fusion et de tri de tableaux basés sur des comparaisons entre les éléments. Les coûts sont évalués uniquement en *nombre de comparaisons entre éléments* (NB les comparaisons entre deux indices de tableaux ne sont pas prises en compte). Pour un algorithme parallèle et une instance de taille n, on note:

- $W_1(n)$: le nombre maximal de comparaisons effectuées; i.e. le temps de son exécution séquentielle, noté aussi parfois $T_1(n)$;
- $D(n)$ sa profondeur, i.e. le nombre maximal de comparaisons entre éléments sur un chemin critique dans le graphe de précédence; i.e. le temps de l'exécution parallèle sur un nombre infini de processeurs identiques, noté aussi $T_\infty(n)$).

Dans toute la suite, on considère le problème MERGE suivant:

- Entrée : deux tableaux **triés** $A = [a_0, \ldots, a_{n-1}]$ et $B = [b_0, \ldots, b_{m-1}]$ (par ordre croissant).
  On suppose de plus tous les éléments a_i et b_j **distincts** : $a_i \neq b_j$ pour tout $0 \leq i < n$ et $0 \leq j < m$.
  On a donc $a_0 < a_1 < \ldots < a_{n-1}$ et $b_0 < b_1 < \ldots < b_{m-1}$.
- Sortie : un tableau $X = [x_0, \ldots, x_{n+m-1}]$ trié (i.e. $x_0 < x_1 < \ldots < x_{n+m-1}$) contenant les éléments de A et B.

I. Complexité de MERGE et algorithme séquentiel

On étudie ici un minorant de la complexité séquentielle de MERGE en nombre de comparaisons.

1. Justifier que pour deux tableaux quelconques A (n éléments) et B (m éléments), il y a $C_{n+m}^n = \frac{(n+m)!}{n!.m!}$ configurations possibles pour le tableau X résultat de la fusion de A et B.

2. En déduire un minorant de la complexité de MERGE (on ne demande pas ici d'équivalent).

3. On rappelle la formule de Stirling : $n! \simeq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$. En déduire un minorant de complexité dans le cas où $n = m$.

4. On considère dans cette question l'algorithme suivant de fusion séquentielle (classique):

```
for (k=0, ptA=0, ptB=0 ; (ptA ≠ n) && (ptB ≠ m); k += 1) {
   if (B[ptB] < A[ptA] ) { X[k] = B[ptB] ; ptB += 1 ; }
   else { X[k] = A[ptA] ; ptA += 1 ; }
}
while (ptA ≠ n) { X[k] = A[ptA] ; ptA += 1 ; k += 1 ; } ;
while (ptB ≠ m) { X[k] = B[ptB] ; ptB += 1 ; k += 1 ; } ;
```

4.a. Justifier que cet algorithme effectue $W_1(n,m) \leq n+m-1$ comparaisons; expliciter un pire cas.

4.b. Quelle est en pire cas sa profondeur D en nombre de comparaisons (i.e. temps parallèle sur un nombre infini de processeurs) ?

II. Un algorithme parallèle D&C pour MERGE

5. On considère l'algorithme parallèle *MergePar* de type "Diviser pour Régner" suivant :

1. On se ramène au cas $n \geq m > 0$ (sinon on appelle MergePar(B, A, X) et si $m = 0$ on a fini).
2. On partitionne le tableau A en deux sous-tableaux $A_1 = [a_0, \ldots, a_{n/2-1}]$ et $A_2 = [a_{n/2}, \ldots, a_{n-1}]$.
3. Soit $\alpha = a_{n/2}$; on sépare B en deux sous-tableaux B_1 et B_2 : $B_1 = [b_0, \ldots, b_{j-1}]$ contient les éléments de B plus petits que α et $B_2 = [b_j, \ldots, b_{m-1}]$ les éléments de B plus grands que α; i.e.
   - si $b_0 > \alpha$ alors B_1 est vide et $B_2 = B$;
   - sinon si $b_{m-1} < \alpha$ alors $B_1 = B$ et B_2 est vide;
   - sinon: j est l'unique indice tel que $b_{j-1} < \alpha < b_j$.
4. On fusionne récursivement A_1 et B_1 dans $X[0, \ldots, n/2+j-1]$;
   et, en parallèle, on fusionne récursivement A_2 et B_2 dans $X[n/2+j, \ldots, n+m-1]$.

5.a. Justifier brièvement que l'algorithme MergePar fusionne correctement les tableaux triés A et B (tous les éléments étant supposés distincts).

5.b. Expliquer comment calculer -en séquentiel- l'indice j utilisé pour partitionner B en faisant $O(\log_2 m)$ comparaisons: on ne demande pas l'algorithme, juste le principe.

5.c Justifier brièvement la récurrence : $\begin{cases} D(m,n) = D(n,m) & \text{si } n < m \\ D(n,m) \leq D(n/2,m) + O(\log m) & \text{si } n \geq m \\ D(n,0) = O(1) \end{cases}$

En déduire que le temps parallèle de cet algorithme est $D(n,m) = O(\log^2(n+m))$.

5.d. On admet que le nombre d'opérations effectuées par l'algorithme MergePar est $W(n,m) = n+m+o(n+m)$ (on ne demande pas de le justifier). Majorer le temps d'exécution sur p processeurs identiques en utilisant un algorithme d'ordonnancement glouton (par vol de travail).

III. Un algorithme très parallèle pour MERGE

6. Le but de cette partie est de construire un algorithme parallèle MergeParFast de temps parallèle (profondeur) constante, mais qui effectue un grand nombre de comparaisons.
Pour simplifier, on pose $a_{-1} = b_{-1} = -\infty$ et $a_n = b_m = +\infty$.
Soit $i \in \{0, \ldots, n-1\}$ un indice arbitraire dans A; soit alors $k \in \{0, \ldots, m\}$ l'indice dans B tel que $b_{k-1} < a_i$ et $b_k > a_i$.

6.a. Justifier que $x_{i+k} = a_i$.

6.b. Expliquer comment calculer l'indice k associé à a_i en temps parallèle $O(1)$ avec m comparaisons.

6.c. En déduire un algorithme de fusion de temps parallèle $O(1)$; préciser le nombre de comparaisons nécessaires. **Indication :** *ranger en parallèle tous les éléments de A et de B dans X.*

IV. Un algorithme en cascade efficace pour MERGE

7. Le but de cette partie est d'améliorer l'algorithme précédent pour construire un algorithme de fusion parallèle très rapide mais qui effectue seulement $W_1(n,m) = O(n+m)$ comparaisons.

Pour $i = 0, \ldots, \lfloor\sqrt{n}\rfloor$, on pose $\alpha_i = a_{i\sqrt{n}}$. De même, pour $j = 0, \ldots, \lfloor\sqrt{m}\rfloor$, on pose $\beta_j = b_{j\sqrt{m}}$.
On pose $\alpha_{-1} = \beta - 1 = -\infty$ et $\alpha_{\lfloor\sqrt{n}\ \rfloor+1} = \beta_{\lfloor\sqrt{m}\ \rfloor+1} = +\infty$.
Enfin, pour $i = 0, \ldots, \lfloor\sqrt{n}\rfloor$, on définit l'indice $\mu_i \in \{0, \ldots, \lfloor\sqrt{m}\ \rfloor + 1\}$ tel que: $\beta_{\mu_i - 1} < \alpha_i < \beta_{\mu_i}$.
et pour $j = 0, \ldots, \lfloor\sqrt{m}\rfloor$, on définit l'indice $\nu_j \in \{0, \ldots, \lfloor\sqrt{n}\ \rfloor + 1\}$ tel que: $\alpha_{\nu_j - 1} < \beta_j < \alpha_{\nu_j}$.

7.a. En utilisant la question 6, montrer que tous les indices μ_i et ν_j peuvent être calculés en $O(1)$ avec $O(n+m)$ comparaisons.

7.b. En déduire un algorithme parallèle pour MERGE de temps parallèle $O(\log\log n)$; et qui effectue $O(n \log\log n)$ comparaisons.

7.c. Construire un algorithme qui résout MERGE en temps parallèle $D(n,m) = O(\log\log n)$ et qui effectue $O(n+m)$ comparaisons seulement.

V. Application à un tri par fusion parallèle

Cette partie, indépendante des précédentes, utilise un algorithme de fusion (qui peut être vu comme un oracle) pour réaliser le tri. L'algorithme récursif de tri par partition-fusion (MERGE-SORT) est le suivant:

```
Algorithme TRI ( T [0 ..  n-1] ) {
   if (n == 1) return T ;
   else {
      A[0..  n/2 - 1] = TRI( T[0 ..  n/2-1] ) ;
      B[0..  n- n/2 - 1] = TRI( T[n/2 ..  n-1] ) ;
      return MERGE(A, B ) ;
   }
}
```

8. On note $D^{(M)}(n)$ (resp. $W_1^{(M)}(n)$) le temps parallèle (resp. nombre d'opérations) de l'algorithme MERGE utilisé. Donner le temps parallèle $D(n)$ et le nombre d'opérations $W_1(n)$ de cet algorithme TRI lorsque MERGE est réalisé par ;

9.a l'algorithme séquentiel de la question 4;

9.b l'algorithme parallèle de la question 5 pour lequel $D^{(M)}(n) = \log^2 n$ et $W_1^{(M)}(n) = O(n)$;

9.c l'algorithme parallèle de la question 8 pour lequel $D^{(M)}(n) = \log\log n$ et $W_1^{(M)}(n) = O(n)$.