Exercises: Parallel merge and application to sort
Jean-Louis Roch

For this problem, a CREW Parallel Random Access Machine is considered: any processor can read data at any address; but write operations on a given address are in mutual exclusion (concurrent write are prohibited). In the sequel, merge and sort algorithms are based in comparisons between elements. Costs of algorithms are uniquely evaluated in number of comparisons between elements; comparisons between array indexes are not taken into account. For a parallel algorithm and an input of size \( n \), the following notations are used:

- \( W_1(n) \): the maximum number of comparisons performed; i.e. the time of the sequential execution, sometimes denoted \( T_1(n) \);
- \( D(n) \) the depth, i.e. the maximum number of comparisons between elements that are in dependence (critical path in the precedence DAG); i.e. the time of a parallel execution on an unbounded number of identical processors, sometimes denoted \( T_\infty(n) \).

The MERGE problem is defined as follows:

- Input: two sorted arrays \( A = [a_0,\ldots,a_{n-1}] \) and \( B = [b_0,\ldots,b_{m-1}] \) (by increasing order).
  Moreover, all elements \( a_i \) are assumed distincts: \( a_i \neq b_j \) for any \( 0 \leq i < n \) and \( 0 \leq j < m \).
  Thus: \( a_0 < a_1 < \ldots < a_{n-1} \) and \( b_0 < b_1 < \ldots < b_{m-1} \).
- Output: a sorted array \( X = [x_0,\ldots,x_{n+m-1}] \) (i.e. \( x_0 < x_1 < \ldots < x_{n+m-1} \)) that contains the elements of both \( A \) and \( B \).

I. Complexity of MERGE and sequential algorithm

This question provides a lower bound on the minimum number of comparisons required for MERGE.

1. Let \( A \) and \( B \) be two arbitrary arrays with respectively \( n \) and \( m \) elements; justify that there are \( C_{n+m}^n = \frac{(n+m)!}{n! \cdot m!} \) possible configurations for the array \( X \) that results from \( \text{MERGE}(A, B) \).

2. En déduire un minorant de la complexité de \( \text{MERGE} \) (on ne demande pas ici d’équivalent). Deduc a lower bound on the complexity of \( \text{MERGE} \).

3. Let remind Stirling formula: \( n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \). Provide a lower bound for \( \text{MERGE} \) when \( n = m \).

4. In this question, we consider the classical sequential merge algorithm:

```java
for (k=0, ptA=0, ptB=0 ; (ptA \neq n) && (ptB \neq m) ; k += 1) {
    else { X[k] = A[ptA] ; ptA += 1 ; }
}
while (ptA \neq n) { X[k] = A[ptA] ; ptA += 1 ; k += 1 ; }
while (ptB \neq m) { X[k] = B[ptB] ; ptB += 1 ; k += 1 ; }
```
4.a. Justify that this algorithm performs $W_1(n, m) \leq n + m - 1$ comparisons; explicit a worst case.

4.b. What is, in worst case, the depth $D$ in number of comparisons (i.e. parallel time on an unbound number of processors)?

II. A parallel Divide&Conquer algorithm for MERGE

5. We consider the following parallel Divide&Conquer algorithm for MERGE:

1. We assume that $n \geq m > 0$ (else $\text{MergePar}(B, A, X)$ is called; if $m = 0$, the algorithm is completed).

2. The array $A$ is split into two sub-arrays $A_1 = [a_0, \ldots, a_{n/2-1}]$ and $A_2 = [a_{n/2}, \ldots, a_{n-1}]$.

3. Let $\alpha = a_{n/2}$; $B$ is split into two subarrays $B_1$ and $B_2$: $B_1 = [b_0, \ldots, b_{j-1}]$ the elements of $B$ lesser than $\alpha$ and $B_2 = [b_j, \ldots, b_{m-1}]$ the elements of $B$ larger than $\alpha$; i.e.
   - if $b_0 > \alpha$ then $B_1$ is empty and $B_2 = B$;
   - else if $b_{m-1} < \alpha$ then $B_1 = B$ and $B_2$ is empty;
   - else: $j$ is the unique index such that $b_{j-1} < \alpha < b_j$.

4. $A_1$ and $B_1$ are recursively merged in $X[0, \ldots, n/2 + j - 1]$; and $A_2$ and $B_2$ are recursively merged in parallel in $X[n/2 + j, \ldots, n + m - 1]$.

5.a. Briefly justify that $\text{MergePar}$ correctly merges the two sorted arrays $A$ and $B$ (all elements are assumed distincts).

5.b. Explain how to compute, in sequential and with $O(\log_2 m)$ comparisons, the index $j$ used to partition $B$; the algorithm is not asked, just its principle.

5.c Briefly justify the recurrence:

\[
\begin{align*}
D(m, n) &= D(n, m) & \text{si } n < m \\
D(n, m) &\leq D(n/2, m) + O(\log m) & \text{si } n \geq m \\
D(n, 0) &= O(1)
\end{align*}
\]

Deduce that the depth of this parallel algorithm is: $D(n, m) = O(\log^2(n + m))$.

5.d. We admit that the number of operation performed by $\text{MergePar}$ is $W(n, m) = n + m + o(n + m)$ (no justification is asked). Give an upper bound on the execution time on $p$ identical processors by using a greedy work-stealing algorithm.

III. An ultrafast parallel algorithm for MERGE

6. This question aims to design a parallel algorithm $\text{MergeParFast}$ with constant depth, but that performs a large number of comparions.

For the sake of simplicity, it is assumed that $a_{-1} = b_{-1} = -\infty$ and $a_n = b_m = +\infty$.

Let $i \in \{0, \ldots, n-1\}$ an arbitrary index in $A$; let $k \in \{0, \ldots, m\}$ be the unique index in $B$ such that $b_{k-1} < a_i$ and $b_k > a_i$. 

2
6.a. Justify that $x_{i+k} = a_i$.

6.b. Give an algorithm to compute the index $k_i$ related to $a_i$ in depth $O(1)$ with $m$ comparisons.

6.c. Deduce a merge algorithm with parallel depth $O(1)$; what is the number of comparisons performed? 

Hint: in parallel, rank all elements of $A$ and $B$ in $X$.

IV. An efficient cascading algorithm for MERGE

7. This question improves previous algorithm of question 6 in order to obtain a very fast parallel merge algorithm that performs an asymptotic optimal number $W_1(n, m) = O(n + m)$ of comparisons.

For $i = 0, \ldots, \lfloor \sqrt{n} \rfloor$, let $\alpha_i = a_i \sqrt{n}$. Conversely, for $j = 0, \ldots, \lfloor \sqrt{m} \rfloor$, let $\beta_j = b_j \sqrt{m}$. Let $\alpha_{-1} = \beta_{-1} = -\infty$ et $\alpha_{\lfloor \sqrt{n} \rfloor + 1} = \beta_{\lfloor \sqrt{m} \rfloor + 1} = +\infty$.

Finally, for $i = 0, \ldots, \lfloor \sqrt{n} \rfloor$, let the index $\mu_i \in \{0, \ldots, \lfloor \sqrt{m} \rfloor + 1\}$ be the one such that: $\beta_{\mu_i-1} < \alpha_i < \beta_{\mu_i}$.

and for $j = 0, \ldots, \lfloor \sqrt{m} \rfloor$, let the index $\nu_j \in \{0, \ldots, \lfloor \sqrt{n} \rfloor + 1\}$ be such that: $\alpha_{\nu_j-1} < \beta_j < \alpha_{\nu_j}$.

7.a. Using question 6, prove that all the index $\mu_i$ and $\nu_j$ can be computed all together in depth $O(1)$ with $O(n + m)$ comparisons.

7.b. Deduce a parallel algorithm for MERGE with depth $O(\log \log n)$; and that performs $O(n \log \log n)$ comparisons.

7.c. Give an algorithm that computes MERGE in parallel depth $D(n, m) = O(\log \log n)$ and that performs $O(n + m)$ comparisons only.

V. Application to parallel merge-sort

This part is dependent from the previous ones; it uses a blackbox merge algorithm to compute the sort. The recursive merge-sort algorithm (MERGE-SORT) is the following:

Algorithm SORT ( 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. We denote $D^{(M)}(n)$ (resp. $W_1^{(M)}(n)$) the parallel depth (resp. work or number of operations) of the used MERGE algorithm. Explicit the depth and work of the above MERGE-SORT algorithm when the MERGE operations is performed by:

9.a. the sequential algorithm of question 4;

9.b. the parallel algorithm of question 5 for which: $D^{(M)}(n) = \log^2 n$ and $W_1^{(M)}(n) = O(n)$;

9.c. the parallel algorithm of question 8 for which: $D^{(M)}(n) = \log \log n$ et $W_1^{(M)}(n) = O(n)$.