PARALLEL AND DISTRIBUTED ALGORITHMS
BY
DEBDEEP MUKHOPADHYAY
AND
ABHISHEK SOMANI

http://cse.iitkgp.ac.in/~debdeep/courses_iitkgp/PAlgo/index.htm

PRAM ALGORITHMS:
MERGING AND GRAPH COLORING
AN OPTIMAL MERGING

We have seen a parallel merging in the last class which takes $O(\log n)$ time with $n$ processors.

What would be a cost-optimal algorithm to perform the merging?

The technique is based on a general strategy which is called partitioning.

* As opposed to Divide and Conquer, the crux lies here in suitably partitioning the problem which helps combining to get the result easy.

OPTIMAL MERGE-PARTITIONING

Spawn $k(m) = \frac{m}{\log m}$ processors.

Each processor divides the array $B$ into subarrays, $B_p$, where $|B_p| = \log m$.

It finds rank of $b_{\log m}$ in $A$ using the binary search algorithm, and let $j(i) = \text{rank}(b_{\log m}|A)$.

The configurations of $A_j$ and $B$ shown above.
EXAMPLE

Let $A = \{4,6,7,10,12,15,18,20\}$, $B = \{3,9,16,21\}$

$m = 4$, $k(m) = \frac{4}{\log 4} = 2$.

$B = ((3,9),(16,21))$

$\text{rank}(9,A) = 3$.

Thus $A = ((4,6,7),(10,12,15,18,20))$

Note each element of $A_1$ and $B_1$ is larger than each element in $A_0$ or $B_0$.

Hence, we can merge $A$ and $B$ by merging separately the pairs $(A_0,B_0)$ and $(A_1,B_1)$.

PROOF

Let $\text{rank}(b_{\text{log } m}|A) = j(i)$

Thus we have: $a_{(j)} < b_{\text{log } m} < a_{(j)+1}$.

This result implies that:

$b_{\text{log } m+1} > b_{\text{log } m} > a_{(j)}$ (thus each element of $B_1$ is larger than each element of $A_{j-1}$)

Likewise, $a_{(j)+1} > b_{\text{log } m}$ (thus each element of $A_j$ is larger than each element of $B_{j-1}$)
TIMING ANALYSIS

The finding of \( j(i) \)'s takes \( O(n) \) time, since the binary search is applied to all the elements of \( A \) in parallel.

Thus the total number of operations required to execute this step (of doing the binary search and partitioning) is

\[ O((\log n)\frac{x m}{\log(2m)}) = O(m+n) \]

Consider the merging of two arrays each of length \( n \).

After the partitioning step we end up with an independent set of merging sub-problems.

- This outcome is the essence of partitioning.
- We would like to handle each merging subproblem in \( O(\log n) \) time, so the algorithm stays cost optimal.

THE MERGING SUBPROBLEMS

Consider the merging subproblem corresponding to \((A_i, B_i)\). Recall \( |B_i| = \log n \), for all indices \( i \).

If \( |A_i| = O(\log n) \), we can merge the pair \((A_i, B_i)\) using an optimal sequential algorithm in \( O(\log n) \) time.

Otherwise, we apply the previous algorithm to partition \( A_i \) into blocks each of which is of size \( O(\log n) \) (in this case \( A_i \) plays the role of \( B \), and \( B_i \) plays the role of \( A \)).

- This step will take \( O(\log \log n) \) time using \( O(|A_i|) \) operations.
- Thus we can make each of the subsequences to be of length \( O(\log n) \)
- Then we apply the best sequential algorithm to merge the subsequences in \( O(\log n) \) using number of processors \( \sum_{i} \left( \frac{|A_i|}{\log n} \right) = n/\log n \), the number of resources also does not increase asymptotically.