Parallel Algorithms for (PRAM) Computers
&
Some Parallel Algorithms

Reference : Horowitz, Sahni and Rajasekaran, *Computer Algorithms*

---

3 Maximum Selection

- Problem : Given n numbers, $x_1, x_2, \ldots, x_n$. Find the largest number.
- Algorithm : $O(1)$ CRCW algorithm; use $n^2$ CPUs; assume all numbers are distinct

Step 1: For each CPU $i,j$ (for each $1 \leq i,j \leq n$) in parallel:
  - $M_{i,j} = 1$ if $x_i < x_j$; otherwise, 0

Step 2: For each row, use $n$ CPUs to compute OR of $n$ elements

Step 3: (cont. from step 2) If $i^{th}$ row is 0, return $x_i$. 
• Example: input <3 1 4 5 2>

After Step 1: Matrix M

<table>
<thead>
<tr>
<th>INDEX</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

After Step 2: Row 4 return 0

After Step 3: return maximum value 5

• Analysis

Total running time: O(1)

Total work: \(n^2 \times O(1) = O(n^2)\)

Sequential Algorithm: O(n)

It is not work optimal!
Recursive Algorithm: \(O(\log \log n)\) CRCW algorithm; use \(n\) CPUs;

Assume \(n\) is always a perfect square, i.e. \(k^2 = n\) (or \(k = n^{1/2}\)).

If this is not true, take the smallest \(k\) such that \(k^2 \geq n\)

**Step 1:** If \(n = 1\), return \(x_1\)

**Step 2:** Partition \(n\) elements & \(n\) processors into \(k\) groups, say,

\[ G_1, G_2, \ldots, G_k (\text{assume } k^2 = n). \]

In parallel, call the algorithm recursively to find maximum element \(m_i\) of each group \(G_i\)

**Step 3:** Use previous algorithm with \(n\) CPUs to find the maximum of \(m_1, m_2, \ldots, m_k\)

---

**Analysis**

This algorithm uses divide and conquer strategy.

In step 2, each sub problem has size \(k\) or \(n^{1/2}\) (\(n^{1/2}\) processors & \(n^{1/2}\) elements)

In step 3, the running time is \(O(1)\)

WLOG, we assume that \(n = 2^q\) and \(T(2) = O(1)\).

The total running time \(T(n)\) satisfy the recurrence

\[
\begin{align*}
T(n) &= T(n^{1/2}) + O(1) \\
T(2) &= O(1)
\end{align*}
\]

which solves to \(T(n) = (\log \log n)\)
4 Merging

- Problem: Given 2 sorted sequences $X_1 = k_1, k_2, \ldots, k_m$ and $X_2 = k_{m+1}, k_{m+2}, \ldots, k_{2m}$.

Assume each sequence has $m$ distinct elements, and $m$ is an integral power of 2.

The goal is to produce a sorted sequence of $2m$ elements.

- Best sequential algorithm is $O(m)$.
For each $k_j \in X_1$, we know that it is the rank $\#j$ element in $X_1$. We allocate a single processor to perform a binary search on $X_2$ and figure out $q$ (the number of elements in $X_2$ that are less than $k_j$). Then we know that $k_j$ is the rank $\#(j+q)$ element in $X_1 \cup X_2$.

For each element in $X_2$, a similar procedure can be used to compute its rank in $X_1 \cup X_2$.

We can use $2m$ processors, one for each element. An overall rank can be found for each element using binary search in $O(\log m)$ time. Merging can be done in $O(\log m)$ time. It is not work optimal!

**Theorem:** Merging of two sorted sequences each of length $m$ can be completed in $O(\log m)$ time using $m$ CREW PRAM processors.

**Recursive EREW Algorithm:** Odd-Even Merge – using $2m$ processors

**Algorithm A**

1. If $m=1$, merge two sequence with 1 comparison
2. Partition $X_1$ into their odd and even parts,
   
   i.e. $X_1^{\text{odd}} = k_1, k_3, k_5, \ldots, k_{m-1}$ and $X_1^{\text{even}} = k_2, k_4, \ldots, k_m$

   Similarly, partition $X_2$ into $X_2^{\text{odd}}$ and $X_2^{\text{even}}$
Step 3: Recursively merge \( X_1^{\text{odd}}, X_2^{\text{odd}} \) (and \( X_1^{\text{even}}, X_2^{\text{even}} \)) using \( m \) processors.

Let \( L_1 = u_1, u_2, \ldots, u_m \)
\( (L_2 = u_{m+1}, u_{m+2}, \ldots, u_{2m}) \)
be the result.

Step 4: Form a sequence \( L = u_1, u_{m+1}, u_2, \)
\( u_{m+2}, u_3, u_{m+3}, \ldots, u_m, u_{2m} \)
Compare every pair \( (u_{m+i}, u_{1+i}) \),
i.e. \( (u_{m+1}, u_2), (u_{m+2}, u_3), \ldots \)
Interchange elements if they are out of order.

Output the resultant sequence.

- Example: \( m = 4 \)

\[
X_1 = (2, 5, 8, 11) \quad X_2 = (4, 9, 12, 18)
\]
Odd = (2, 8) \quad even = (5, 11)
odd = (4, 12) \quad even = (9, 18)
\[
\begin{align*}
(2, 8) & \quad (4, 12) \\
(2) & \quad (8) \\
(2, 4) & \quad (8, 12)
\end{align*}
\]

\[
(2, 8, 4, 12) \quad (5, 11, 2, 18)
\]

\[
(2, 4, 8, 12) \quad (5, 9, 11, 18)
\]

\[
(2, 4, 5, 8, 9, 11, 12, 18)
\]

\[
(2, 4, 5, 8, 9, 11, 12, 18)
\]
Theorem: The previous algorithm $A$ correctly merge two sorted sequences of arbitrary numbers (Proof: Assignment #1)

- **Analysis**

  This algorithm $A$ uses divide and conquer strategy

  **Step 1**, $O(1)$

  **Step 2**, Partition can be done by $2m$ CPUs at the same time in $O(1)$

  **Step 3**, There are two sub-problems. Using $m$ CPUs in parallel to solve each sub-problem. A sub-problem has 2 sorted lists and a list is with $m/2$ elements.

  **Step 4**, Using $m$ CPUs in parallel in $O(1)$

  The total running time is

  $T(m) = T(m/2) + O(1) = O(\log m)$

  Note: $T(m)$ means running time to merge 2 sorted lists, each with $m$ elements

  Total work: $2m \ast O(\log m) = O(m \log m)$

  It is not work optimal!
• A work optimal CREW merging algorithm

• Goal: use $O(m/\log m)$ CPUs to obtain $O(\log m)$ algorithm

**Algorithm B**
Step 1: Partition $X_1$ in $(m/\log m)$ parts, $A_1, A_2, \ldots, A_z$, where $z = (m/\log m)$. Note: each $A_i$ has $(\log m)$ elements.

Step 2: Let $u_i$ be the largest element in $A_i$, i.e. last element of $A_i$. Assign a cpu to each $u_i$. Use binary search, $O(\log m)$, to search the correct position of $u_i$ in $X_2$. This divides $X_2$ into $z$ parts $B_1, B_2, \ldots, B_z$.

Now: we only need to merge $A_i$ with $B_i$ for $1 \leq i \leq z$
Step 3: If $|B_i| = O(\log m)$, then $A_i$ and $B_i$ can be merged in $O(\log m)$; Otherwise, partition $B_i$ in $\lceil |B_i|/(\log m) \rceil$ parts. Now, use similar strategy as Step 2, i.e. assign a cpu to each sub-part of $B_i$, use largest key to find correct position in $A_i$, i.e. $O(\log \log m)$.

There are at most $2^z$ parts in $B_1, B_2, \ldots, B_z$, (think about WHY?) and each part has at most $\log m$ elements. We need at most $2^z$ CPUs, each pair needs $O(\log m)$ to merge.

The total running time of Algorithm B is $O(\log m)$

Total work:

$2(\frac{m}{\log m}) \cdot O(\log m) = O(m)$

It is work optimal!
5 Sorting

Odd Even Merge Sort:

Algorithm C

Step 1: If n <= 1 return X

Step 2: Use n CPUs to partition input X of n elements into 2 lists, X₁ and X₂. Each with n/2 elements

Step 3: Use n/2 CPUs to sort X₁ recursively and n/2 CPUs to sort X₂ recursively. Let X₁* and X₂* be the result sorted lists.

Step 4: Use Odd-Even merge to merge two sorted lists using n CPUs.

Analysis of Algorithm C:

EREW algorithm using n CPUs

T(n) = O(1) + T(n/2) + O(log n)

= log(n) + log(n/2) + log(n/4) + … + log(n/2ⁱ) ; i = log n

<= log(n) * log(n)

T(n) = O(log² n)

Total work O(n log² n)

It is not work optimal