Chapter 9: Algorithms for Parallel (PRAM) Computers

Reference: Computer Algorithms by Horowitz, Sahni and Rajasekaran

9.1 Computational Model

- Most popular theoretical Model: PRAM (parallel random-access machine)
  - X processors, $P_0, P_1, \ldots, P_{x-1}$ share global memory and a common clock
  - May execute different instructions at the same time, e.g. read/write at the same time
  - Key assumption: running time can be measured as the number of parallel memory accesses.
    i.e. In 1 cpu machine, to do 1 statement, say, 1 unit of time.
    In X cpu machines, to do 1 statement (each cpu in parallel), also, 1 unit of time

- Versions of PRAM algorithms
  - Concurrent read (CR): may read same location at the same time
  - Exclusive read (ER) - no two processors can read same location at the same time
  - Also, concurrent write (CW) and exclusive write (EW)
  - Read from and write to same location at the same time is not allowed
  - Types of algorithms: EREW, CREW, ERCW, CRCW
    Not many algorithms on ERCW
A PRAM that supports EREW is called EREW-PRAM (similarly, we can define CRCW-PRAM etc)

- Most algorithms assume \( n, \log n, \) or \( n/\lg n \) number of processors. In practice, this is not a realistic assumption.

- For CW, assume all CPU write the same value.
  Other assumptions: last CPU value in memory, arbitrary value etc.

- Simple CRCW algorithm: To compute \( a[0] = a[1] \parallel a[2] \parallel \ldots \parallel a[n] \)
  for each processor \( i \ (1 \leq i \leq n) \) in parallel
  
  if \( A[i] = 1 \) then \( A[0] := A[i]; \)

- Running time analysis: For a given problem \( X \) with input size \( n \),
  Let the run time of a parallel algorithm using \( p \) processors be \( T(n,p) \)
  Let the run time of a best known sequential algorithm be \( S(n) \)

  - The total work of a parallel algorithm is: \( p \cdot T(n,p) \)
  - The speedup of a parallel algorithm is \( S(n)/T(n,p) \)
  - If speedup is \( O(p) \) then the algorithm is said to have a linear speedup

  - A parallel algorithm is said to be work-optimal if \( p \cdot T(n,p) = O(S(n)) \)

  - A parallel algorithm is work-optimal if and only if it has linear speedup

  - For above example: \( S(n) = O(n) \), \( n \cdot T(n,n) = n \cdot O(1) \) \( \rightarrow \) work optimal.
9.2 Prefix computation

- Let $\oplus$ = binary, associative operator, i.e. $(x \oplus y) \oplus z = x \oplus (y \oplus z)$

- Example : $+$, $*$, AND, OR etc

- Problem : Given $n$ elements, $x_1, x_2, \ldots, x_n$. Compute $n$ elements $x_1 \oplus x_2, \ldots, x_1 \oplus x_2 \oplus x_3 \ldots x_{n-1} \oplus x_n$

- Algorithm : using $n$ CPUs; assume $n$ is $2^k$

  for each CPU $i$ in parallel /* initialize */

  $y[i] = x_i$

  for $i = 0$ to $k-1$ do

  for each CPU $j$ where $j > 2^i$ in parallel

  $y[j] = y[j] \oplus y[j-2^i]$

- Example : input <3 1 4 5 2 3 6 7>

<table>
<thead>
<tr>
<th>index</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial y[]</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>i = 0 y[]</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>i = 1 y[]</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>13</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>i = 2 y[]</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>13</td>
<td>15</td>
<td>18</td>
<td>24</td>
<td>31</td>
</tr>
</tbody>
</table>

- Analysis

  Above algorithm is EREW (or CREW) algorithm

  The run time of best sequential algorithm is $O(n)$
  The run time of above algorithm is, $T(n,n) = O(lg n)$
  The total work is $O(n \ lg n)$
  It is not work optimal!
To get work optimal algorithm, we need to use only \((n/\log n)\) CPUs

Work optimal algorithm : using \((n/\log n)\) CPUs

1. Assign \(\log n\) elements to each CPU

2. Each CPU computes the prefixes of its assigned \(\log n\) elements using a simple sequential algorithm

3. From Step 2, use last element from each CPU, i.e. total \((n/\log n)\) elements.

Use \((n/\log n)\) CPUs to compute prefixes of theses \(n/\log n\) elements using previous parallel algorithm (see example, record results in a new array)

4. Each CPU updates its \(\log n\) elements using results from Step 3

Example : input <3 1 4 5 2 3 6 7>

<table>
<thead>
<tr>
<th></th>
<th>CPU 1</th>
<th>CPU 2</th>
<th>CPU 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Step 1</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Step 2</td>
<td>3</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Step 3</td>
<td>8</td>
<td>18</td>
<td>31</td>
</tr>
<tr>
<td>Step 4</td>
<td>3</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

Analysis

Step 1 : \(O(\log n)\)
Step 2 : \(O(\log n)\)
Step 3 : \(O(\log (n/\log n)) = O(\log n)\)
Step 4 : \(O(\log n)\)
Total work : \((n/\log n) \times O(\log n) = O(n)\)

It is work optimal!
9.3 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:

\[
M_{i,j} = 1 \text{ if } x_i < x_j; \text{ otherwise, } 0
\]

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

Step 1: If n is small, compute the maximum directly and return it

Step 2: Partition n elements & n processors into k groups, say, 
  \( G_1, G_2, \ldots, G_k \) (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 use divide and conquer strategy

  In step 2, each sub problem has size k (k processors & k elements)

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

  The total running time is \( T(n) = T(k) + O(1) \)

  Recall the quiz#1 problems: \( T(n) = (\log \log n) \)

  Total work: \( n \times O(\log \log n) = O(n \log \log n) \)

  It is not work optimal!

  Note: There is a work optimal randomized algorithm.
9.4 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 $O(m)$

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

  Step 1: If $m=1$, merge two sequence with 1 comparison

  Step 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 $(u_{m+i}, u_{i+1})$, 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) \quad odd = (4, 12) \quad even = (9, 18)

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

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

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

(2, 4) (8, 12) (5, 9) (11, 18)

(2, 8, 4, 12) (5, 11, 9, 18)

(2, 4, 8, 12) (5, 9, 11, 18)

(2, 5, 4, 9, 8, 11, 12, 18)

(2, 4, 5, 8, 9, 11, 12, 18)
• Theorem: The previous algorithm correctly merge two sorted sequences of arbitrary numbers

Prove by induction on $2^K$ elements on each list!

• Analysis

This algorithm use 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: $2m \cdot O(\log m) = O(m \log m)$

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

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

• Major steps:

Step 1: Partition $X_1$ in $(m / \log m)$ parts, $A_1$, $A_2$, ..., $A_z$, 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 $B_1$, $B_2$, ..., $B_z$.

\[
X_1 \mid A_1 \mid A_2 \mid A_3 \mid \ldots \mid A_z
\]

\[
X_2 \mid B_1 \mid B_2 \mid B_3 \mid \ldots \mid B_z
\]

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 ceiling($|B_i|/(\log m)$) 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 total of $2z$ parts in $B_1$, $B_2$, ..., $B_z$, i.e. each $B_i$ may have at most one part $< \log m$ elements.

We need total of $2z$ CPUs, each pair needs $O(\log m)$ to merge.

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

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

It is work optimal!
Chapter 10: Algorithms for fixed connection parallel machines: Mesh

Reference: Computer Algorithms by Horowitz, Sahni and Rajasekaran

10.1 Computation Model

- A mesh is an a X b grid in which there is a processor at each grid point
- The edges are bi-directional communication links, i.e. two separated uni-directional links.
- Each processor can be labeled with a tuple (i, j)
- Each processor has some local memory, and can perform basic operations
- There is a global clock which synchronize all processors
- We only consider square meshes, i.e. a = b, and linear array

A 4 by 4 mesh (16 processors)       A Linear array of p processors

1,1   1,2   1,3   1,4

4,1   4,2   4,3   4,4

1   2   3   p-1   p

o-----o------o--- …,---o-------o
10.2 Packet Routing

- Primitive IPC operation – packet routing
- A packet contains data + source processor + destination processor
- A link can handle only one packet at one unit time
- A processor may receive multiple packets (from different links) and send multiple packets (to different links) at the same time
- A processor may queue some packets in its local storage
- Each processor uses the same packet routing algorithm
- Partial Permutation Routing (PPR) is a special case of general routing problem. In PPR, each processor is the origin (and destination) of at most one packet.
- The performance of an algorithm is measured by run time, i.e. time to complete all operations, and maximum queue length, i.e. the maximum number of packets in a processor queue.

Packet routing on a linear array of $p$ processors

**Problem 1**: At most one packet originated from each processor.

The problem can be solved in $\leq p - 1$ steps.

Every processor starts to send packet using the shortest route. There is no contention for any link. The maximum distance is $p - 1$ links.

**Problem 2**: Each processor is the destination for exactly one packet.

Note: A processor may have multiple packets to be routed to multiple processors
Farthest destination first strategy: At each time unit, each processor chooses at most two packets from its queue, one to farthest left (if any) and one to farthest right (if any).

The problem can be solved by the farthest destination first strategy in $\text{Max}(p-i,i-1)$ step for processor $i$, i.e. $O(p-1)$

Proof:

Let consider only packets that are moving from left to right.

A packet to processor $p$ cannot be delayed according to the strategy, so, it will reach destination at $p-i$ units.

A packet to processor $p-1$ can only be delayed by packet to $p$, so it will reach destination at $p-1-i$ units + 1 delayed unit $\leq p-i$ units

Packet routing (PPR) on a Mesh (Assume $p \times p$ processors)

Algorithm:

Let $q$ be an arbitrary packet with $(i,j)$ as its origin and $(u,v)$ as its destination.

Phase 1: Travel along column $j$ to row $u$

Phase 2: Travel along row $u$ to its destination $(u,v)$

Running time analysis:

The lower bound is $2(p-1)$ steps, i.e. from $(1,1)$ to $(p,p)$

Phase 1 can be done in $(p-1)$ steps (by problem 1 above)
Phase 2 can be done in $(p-1)$ steps (by problem 2 above)

Total: $2(p-1)$ steps $\Rightarrow$ optimal algorithm
Problem: in the worst case, the queue size is $p/2$.

i.e. In each time unit, 2 packets arrive and only one can be sent out.
There are atmost $p$ packets in one column $\rightarrow$ may need to queue up to $p/2$ packets.

10.3 Fundamental Algorithms

- Broadcasting problem: To broadcast a message to all processors.

Assume a processor can duplicate message.

Phase 1: Send the message along the row.
Phase 2: For each processor in the row, send message along its column.

Total time $O(p)$

- Prefix computation problem for $p \times p$ mesh; Total time $O(p)$

Assume there is a number in each processor, i.e. $p^2$ numbers $\{x_{1,1}, x_{1,2}, \ldots, x_{1,p}, x_{2,1}, x_{2,2}, \ldots, x_{2,p}, \ldots, x_{p,1}, x_{p,2}, \ldots, x_{p,p}\}$

Phase 1: Each row $i$, compute prefix

$$x_{i,1}, x_{i,1} \oplus x_{i,2}, \ldots, x_{i,1} \oplus x_{i,2} \oplus \ldots \oplus x_{i,p}$$

Phase 2: Compute prefix for number in column $p$.

Assume final numbers in column $p$ are $\alpha_1, \alpha_2, \ldots, \alpha_p$

Phase 3: For each processor $(i,p)$ in column $p$, broadcast the number $\alpha_{i-1}$ to all other elements in the same row $i$. The number $\alpha_{i-1}$ will be added to each number in row $i$. 

123
Example : (assume each small square is a processor in 4 x 4 mesh)

\[
\begin{array}{cccc}
0 & 1 & 1 & 2 \\
1 & 0 & 2 & 1 \\
1 & 0 & 0 & 2 \\
0 & 1 & 2 & 3 \\
\end{array}
\quad \rightarrow \quad
\begin{array}{cccc}
0 & 1 & 2 & 4 \\
1 & 1 & 3 & 4 \\
1 & 1 & 1 & 3 \\
0 & 1 & 3 & 6 \\
\end{array}
\quad \rightarrow \quad
\begin{array}{cccc}
0 & 1 & 2 & 4 \\
1 & 1 & 3 & 8 \\
1 & 1 & 1 & 11 \\
0 & 1 & 3 & 17 \\
\end{array}
\quad \rightarrow \quad
\begin{array}{cccc}
0 & 1 & 2 & 4 \\
5 & 5 & 7 & 8 \\
9 & 9 & 9 & 11 \\
11 & 12 & 14 & 17 \\
\end{array}
\]

10.4 Merging

- Odd-Even Merge on a Linear Array

Assume two sorted lists with 2m numbers :

\[a_1 \ a_2 \ a_3 \ a_4 \ \ldots \ a_{m-1} \ a_m \ b_1 \ b_2 \ b_3 \ b_4 \ \ldots \ b_{m-1} \ b_m\]

Assume there are 2m processors and each processor has a number.

1\textsuperscript{st} m processors hold 1\textsuperscript{st} list of m sorted numbers,
2\textsuperscript{nd} m processors hold 2\textsuperscript{nd} list of m sorted numbers.

Step 1: Group odd part and even part of \(a_i\) (also, same for \(b_i\))

i.e. \[a_1 \ a_3 \ \ldots \ a_{m-1} \ a_2 \ a_4 \ \ldots \ a_m \ b_1 \ b_3 \ \ldots \ b_{m-1} \ b_2 \ b_4 \ \ldots \ b_m\]

Run time : O(m/2)

Step 2 : Group odd parts (and even parts) of both lists

i.e. \[a_1 \ a_3 \ \ldots \ a_{m-1} \ b_1 \ b_3 \ \ldots \ b_{m-1} \ a_2 \ a_4 \ \ldots \ a_m \ b_2 \ b_4 \ \ldots \ b_m\]

Run time : O(m/2)
Step 3: Odd parts (and even parts) are merged recursively to get two sorted lists

\[ i.e. \quad o_1 \ o_2 \ldots o_{m-1} \quad e_1 \ e_2 \ldots e_{m-1} \quad e_m \]

Run time: \( T(m/2) \)

Step 4: Shuffled odd and even numbers

\[ i.e. \quad o_1 \ e_1 \ o_2 \ e_2 \ o_3 \ldots \ e_{m-2} \ o_{m-1} \ e_{m-1} \ o_m \ e_m \]

Run time: \( O(m) \)

Step 5: Compare adjacent elements and swap numbers if out of order

\[ i.e. \quad o_1 \ e_1 \ o_2 \ e_2 \ o_3 \ldots \ e_{m-2} \ o_{m-1} \ e_{m-1} \ o_m \ e_m \]

Run time: \( O(1) \)

Total Run time \( T(m) = T(m/2) + 2m + 1 = O(m) \)

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

- Odd-even transposition sort on linear array

Assume each processor $i$ has a number $x_i$

For $i = 1$ to $p$ do

  If $i$ is odd : compare keys at processors $2j-1$ and $2j$ for all $j$
  If $i$ is even : compare keys at processors $2j$ and $2j+1$ for all $j$

This can be done in $O(p)$, skip the proof!

Example :

\[
\begin{align*}
  i = 1 & : \underline{5} \, \underline{4} \, 8 \, 1 \, 2 \, 6 \, 3 \, 7 \\
  i = 2 & : \underline{4} \, \underline{5} \, 1 \, 8 \, 2 \, 6 \, 3 \, 7 \\
  i = 3 & : 4 \, 1 \, \underline{5} \, 2 \, 8 \, 3 \, 6 \, 7 \\
  i = 4 & : 1 \, \underline{4} \, 2 \, 5 \, 3 \, \underline{8} \, 6 \, 7 \\
  i = 5 & : \underline{1} \, \underline{2} \, 4 \, 3 \, 5 \, 6 \, \underline{8} \, 7 \\
  i = 6 & : \underline{1} \, \underline{2} \, \underline{3} \, 4 \, 5 \, 6 \, 7 \, 8
\end{align*}
\]
Shearsort on a Mesh  (use snakelike order)

Assume each processor \( i \) has a number \( x_i \)

For \( i = 1 \) to \( \log(p^2) + 1 \) do
  - If \( i \) is even : sort each columns in increasing order from top to bottom
  - If \( i \) is odd : sort each rows; alternate rows are sorted in reverse order.

Use previous algorithm to sort \( p \) elements in \( O(p) \)
This can be done in \( O(p \log p) \), skip the proof!

Example :

\[
\begin{array}{cccc}
15 & 12 & 8 & 32 \\
7 & 13 & 6 & 17 \\
2 & 16 & 25 & 19 \\
18 & 5 & 11 & 3 \\
\end{array} \rightarrow \begin{array}{cccc}
8 & 12 & 15 & 32 \\
17 & 13 & 7 & 6 \\
2 & 16 & 19 & 25 \\
18 & 11 & 5 & 3 \\
\end{array}
\]

\[
\begin{array}{cccc}
2 & 11 & 5 & 3 \\
8 & 12 & 7 & 6 \\
17 & 13 & 15 & 25 \\
18 & 16 & 19 & 32 \\
\end{array} \rightarrow \begin{array}{cccc}
2 & 3 & 5 & 11 \\
12 & 8 & 7 & 6 \\
13 & 15 & 17 & 25 \\
32 & 19 & 18 & 16 \\
\end{array}
\]

\[
\begin{array}{cccc}
2 & 3 & 5 & 6 \\
12 & 8 & 7 & 11 \\
13 & 15 & 17 & 16 \\
32 & 19 & 18 & 25 \\
\end{array} \rightarrow \begin{array}{cccc}
2 & 3 & 5 & 6 \\
12 & 11 & 8 & 7 \\
13 & 15 & 16 & 17 \\
32 & 25 & 19 & 18 \\
\end{array}
\]

Chapter 11: Routing Messages in a Network

11.1 Computation Model

- In a distributed real-time system, the problem of determining whether a set of messages can be sent on-time is an important issue.

- A network is a directed graph $G = (V,E)$, where each vertex is a node of the network and an edge is a communication link.

- If $(u,v)$ is an edge, then there is a transmitter in node $u$ and a receiver in node $v$.

- Assume a node can simultaneously send and receive several messages provided that they are transmitted on different communication links.

- A set of $n$ messages $M = \{M_1, M_2,..., M_n\}$ needs to be routed through the network.

- Each message $M_i$ is represented by the quintuple $(s_i, e_i, l_i, r_i, d_i)$.

  The message originates from node $s_i$ at time $r_i$ and is to be sent to node $e_i$ by time $d_i$.

  The message consists of $l_i$ packets of information (where each packet can be sent in one unit time).

- Assume each message must be completely received by a node before it can be forwarded to another node.

- Centralized distributed algorithm: There is a coordinator that constructs a route for the messages and that the routed will be broadcasted to each node.

- On-line (or fully distributed) algorithm: Each node in the network routes messages without any knowledge of the messages in other nodes.
• Problem: Given a network G and a set of messages M.

If the set of messages M can be routed through the network G such that each message $M_i$ is sent from node $s_i$ to the node $e_i$ in the time interval $[r_i, d_i]$?

• We study the complexity of the problem under various restrictions on the four parameters of the messages: $s_i, e_i, r_i$ and $d_i$.

• Example: \( G = \{\{1,2,3,4,5\}, \{(1,2), (1,3), (3,4), (2,4), (4,5)\}\} \)

<table>
<thead>
<tr>
<th>$M_i$</th>
<th>$s_i$</th>
<th>$e_i$</th>
<th>$l_i$</th>
<th>$r_i$</th>
<th>$d_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>$M_2$</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>$M_3$</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>$M_4$</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

• A feasible nonpreemptive transmission:

<table>
<thead>
<tr>
<th>Edges</th>
<th>Times</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,2)</td>
<td></td>
<td>M_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1,3)</td>
<td></td>
<td>M_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2,4)</td>
<td></td>
<td>M_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3,4)</td>
<td></td>
<td>M_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4,5)</td>
<td></td>
<td>M_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• If you change $M_4$ to (4, 5, 2, 3, 5), then there is no feasible non-preemptive transmission.
There is a feasible preemptive transmission:

<table>
<thead>
<tr>
<th>Edges\Times</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,2)</td>
<td></td>
<td>M₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1,3)</td>
<td>M₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2,4)</td>
<td></td>
<td></td>
<td>M₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3,4)</td>
<td>M₂</td>
<td></td>
<td></td>
<td>M₃</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4,5)</td>
<td></td>
<td>M₃</td>
<td>M₄</td>
<td>M₃</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11.2 General Network Results

- Given an arbitrary network and a set of messages with same $s_i$, $e_i$, $r_i$ and $d_i$.

MRNS problem: Determining whether there is a feasible preemptive transmission is NP-complete.

- Proof:

The MRNS problem is NP

A node may need to preempt a message when a new message arrives and needs to send over the same link. So, each message may be preempted only $O(n)$ times.

Guess preemption points for each message on each edge. Verify the feasibility in polynomial time.

Use 3-Partition Problem to show that MRNS problem is NP-hard

3-Partition Problem: Given a list $A=(a_1, a_2, \ldots, a_{3z})$ of $3z$ integers such that $\Sigma A = zB$ and $B/4 < a_i < B/2$ for each $1 \leq i \leq 3z$. Can $A$ be partitioned into $z$ sets, $S_1, S_2, \ldots, S_z$, such that $\Sigma S_i = B$. Note: Each set must have exactly 3 elements from $A$. 
3-Partition \(\propto\) MRNS

Given arbitrary instance of 3-Partition, i.e. \(A=(a_1, a_2, \ldots, a_{3z})\)
Construct an instance of MRNS as follows:

\[
G:
\]

\[
0 \rightarrow 1 \rightarrow 2 \rightarrow z+1
\]

\[
\cdots
\]

\[
M: \{M_1, M_2, \ldots, M_{4z}\} \quad \text{where}
\]

<table>
<thead>
<tr>
<th>(M_i)</th>
<th>(s_i)</th>
<th>(e_i)</th>
<th>(l_i)</th>
<th>(r_i)</th>
<th>(d_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M_1)</td>
<td>0</td>
<td>z+1</td>
<td>(a_1)</td>
<td>0</td>
<td>5B</td>
</tr>
<tr>
<td>(M_2)</td>
<td>0</td>
<td>z+1</td>
<td>(a_2)</td>
<td>0</td>
<td>5B</td>
</tr>
<tr>
<td>(\cdots)</td>
<td>(\cdots)</td>
<td>(\cdots)</td>
<td>(\cdots)</td>
<td>(\cdots)</td>
<td>(\cdots)</td>
</tr>
<tr>
<td>(M_{3z})</td>
<td>0</td>
<td>z+1</td>
<td>(a_{3z})</td>
<td>0</td>
<td>5B</td>
</tr>
<tr>
<td>(M_{3z+1})</td>
<td>0</td>
<td>z+1</td>
<td>2B</td>
<td>0</td>
<td>5B</td>
</tr>
<tr>
<td>(\cdots)</td>
<td>(\cdots)</td>
<td>(\cdots)</td>
<td>(\cdots)</td>
<td>(\cdots)</td>
<td>(\cdots)</td>
</tr>
<tr>
<td>(M_{4z})</td>
<td>0</td>
<td>z+1</td>
<td>2B</td>
<td>0</td>
<td>5B</td>
</tr>
</tbody>
</table>

Note: All messages have same \(s_i\), \(e_i\), \(r_i\) and \(d_i\)
The transformation can be done in polynomial time
Based on input size \(a_1, a_2, \ldots, a_{3z}\)
Property:

- There are $z$ paths from node 0 to node $z+1$
- $M_{3z+1}, M_{3z+2}, \ldots, M_{4z}$ are enforcer message.
- There is a feasible schedule if and only if exactly one enforcer message is routed on each path

If 3-Partition problem has a solution, then MRNS problem has a feasible transmission

The message $M_{3z+k}$ and corresponding 3 messages $M_{k1}, M_{k2}, M_{k3}$ for set $S_k = \{a_{k1}, a_{k2}, a_{k3}\}$ are routed through path $0 \rightarrow k \rightarrow z+1$

<table>
<thead>
<tr>
<th>Edges\Times</th>
<th>0</th>
<th>B</th>
<th>2B</th>
<th>3B</th>
<th>4B</th>
<th>5B</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,k)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$M_{3z+k}$</td>
</tr>
<tr>
<td>(k,z+1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$M_{3z+k}$</td>
</tr>
</tbody>
</table>

Obvious to see that it is a feasible transmission.

If MRNS problem has a feasible transmission then 3-Partition problem has a solution

If there is a feasible transmission, then each edge $(0,k)$, for $1 \leq k \leq z$, must not have any idle time & handle all messages on or before 3B; otherwise, messages cannot reach their destination node $z+1$ before due date 5B.

$\rightarrow$ exactly one enforcer message + 3 other messages of total length B
11.3 Unidirectional Ring Network Results

- Consider only unidirectional ring network:
  
i.e. $G = (\{1,2,\ldots,m\},\{(1,2),(2,3),\ldots,(m,1)\})$

11.3.1 Nonpreemptive transmission

Consider four parameters:

- origin nodes ($s_i$), destination nodes ($e_i$),
- release times ($r_i$) & deadlines ($d_i$)

Problem 1: Polynomial time when any one of the four parameters is allowed to be arbitrary

**Algorithm A**: Earlier Available Message Strategy

**Algorithm B**: Earlier Available Message and Farthest Destination Strategy

**Algorithm C**: Earlier Available Message and Earliest Deadline Strategy
<table>
<thead>
<tr>
<th>si</th>
<th>ei</th>
<th>ri</th>
<th>di</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed</td>
<td>arbitrary</td>
<td>fixed</td>
<td>fixed</td>
<td>B</td>
</tr>
<tr>
<td>arbitrary</td>
<td>fixed</td>
<td>fixed</td>
<td>fixed</td>
<td>A</td>
</tr>
<tr>
<td>fixed</td>
<td>fixed</td>
<td>fixed</td>
<td>arbitrary</td>
<td>C</td>
</tr>
<tr>
<td>fixed</td>
<td>fixed</td>
<td>arbitrary</td>
<td>fixed</td>
<td>A</td>
</tr>
</tbody>
</table>

Example: Consider same destination nodes, release times, deadlines

Let: \( G = \{(1,2,3,4), \{(1,2),(2,3),(3,4),(4,1)\}\)  
\( M = \{M_1,M_2,M_3,M_4\} \)

<table>
<thead>
<tr>
<th>( M_i )</th>
<th>( s_i )</th>
<th>( e_i )</th>
<th>( l_i )</th>
<th>( r_i )</th>
<th>( d_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_1 )</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>( M_2 )</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>( M_3 )</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>( M_4 )</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>12</td>
</tr>
</tbody>
</table>

Use Earlier Available Message Strategy:

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
(1,2) & M_3 & M_4 & //////////////////////////////////////////////////////////////\\n(2,3) & M_1 & M_2 & M_3 & M_4 & //////////////\\n(3,4) & ////////////// M_1 & ///// M_2 & M_3 & M_4\\n\end{array}
\]
Problem 2 : NP-complete when any two of the four parameters are allowed to be arbitrary

<table>
<thead>
<tr>
<th>s_i</th>
<th>e_i</th>
<th>r_i</th>
<th>d_i</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed</td>
<td>fixed</td>
<td>arbitrary</td>
<td>arbitrary</td>
<td>NP complete</td>
</tr>
<tr>
<td>fixed</td>
<td>arbitrary</td>
<td>fixed</td>
<td>arbitrary</td>
<td>NP complete</td>
</tr>
<tr>
<td>fixed</td>
<td>arbitrary</td>
<td>arbitrary</td>
<td>fixed</td>
<td>NP complete</td>
</tr>
<tr>
<td>arbitrary</td>
<td>fixed</td>
<td>arbitrary</td>
<td>fixed</td>
<td>NP complete</td>
</tr>
<tr>
<td>arbitrary</td>
<td>fixed</td>
<td>fixed</td>
<td>arbitrary</td>
<td>NP complete</td>
</tr>
<tr>
<td>arbitrary</td>
<td>arbitrary</td>
<td>fixed</td>
<td>fixed</td>
<td>NP complete</td>
</tr>
</tbody>
</table>

Example : Consider fixed destination nodes & release times

Given arbitrary instance of 3-Partition, i.e. A=(a_1, a_2, ..., a_{3z})
Construct an instance of MRNS as follows :

G : \( \{1, 2, \ldots, 2z+2\}, \{(1, 2), (2, 3), \ldots, (2z+1, 2z+2)\} \)

M: \{M_1, M_2, \ldots, M_{4z+1}\} where

<table>
<thead>
<tr>
<th>M_i</th>
<th>s_i</th>
<th>e_i</th>
<th>l_i</th>
<th>r_i</th>
<th>d_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_1</td>
<td>2z</td>
<td>2z+2</td>
<td>a_1</td>
<td>0</td>
<td>2zB</td>
</tr>
<tr>
<td>M_2</td>
<td>2z</td>
<td>2z+2</td>
<td>a_2</td>
<td>0</td>
<td>2zB</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>M_{3z}</td>
<td>2z</td>
<td>2z+2</td>
<td>a_{3z}</td>
<td>0</td>
<td>2zB</td>
</tr>
<tr>
<td>M_{3z+1}</td>
<td>1</td>
<td>2z+2</td>
<td>B</td>
<td>0</td>
<td>(2Z+1)B</td>
</tr>
<tr>
<td>M_{3z+2}</td>
<td>3</td>
<td>2z+2</td>
<td>B</td>
<td>0</td>
<td>(2Z-1)B</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>M_{4z+1}</td>
<td>2z+1</td>
<td>2z+2</td>
<td>B</td>
<td>0</td>
<td>B</td>
</tr>
</tbody>
</table>

Diagram:

1 -- 2
2 -- 3
3 -- 1

2z+2
Property:

- The enforcer messages $M_{3z+1}$ to $M_{4z+1}$ are urgent, i.e. the message must be transmitted without any delay on each node.
- The transmission of the enforcer messages will leave exactly $z$ disjoint interval, each with length $B$, on the edges $(2z, 2z+1)$ and $(2z+1, 2z+2)$.
- Those intervals are for the transmission of messages $M_1$ to $M_{3z}$
- It is obvious the transmission is feasible if and only if messages $M_1$ to $M_{3z}$ can be partitioned into these $z$ intervals of size $B$

<table>
<thead>
<tr>
<th>Edges \ Times</th>
<th>0</th>
<th>B</th>
<th>2B</th>
<th>3B</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,2)</td>
<td>M_{3z+1}</td>
<td>////////////////////////////////////////////////////////////////////////////////////</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1,3)</td>
<td>///////////////</td>
<td>M_{3z+1}</td>
<td>////////////////////////////////////////////////////////////////////////////////////</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2,4)</td>
<td>M_{3z+2}</td>
<td>///////////////</td>
<td>M_{3z+1}</td>
<td>////////////////////////////////////////////////////////////////////////////////////</td>
<td></td>
</tr>
<tr>
<td>(2z, 2z+1)</td>
<td>M_{4z}</td>
<td>M_{4z-1}</td>
<td>…………………………</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2z+1, 2z+2)</td>
<td>M_{4z+1}</td>
<td>M_{4z}</td>
<td>M_{4z-1}</td>
<td>………..</td>
<td>M_{3z+1}</td>
</tr>
</tbody>
</table>

$2zB$  $(2z+1)B$
11.3.2 Preemptive transmission

Consider four parameters:

- origin nodes \( s_i \), destination nodes \( e_i \),
- release times \( r_i \) & deadlines \( d_i \)

**Problem 1**: Polynomial time when any one of the four parameters is allowed to be arbitrary

Same results as nonpreemptive transmission.

**Problem 2**: NP-complete when any two of the four parameters are allowed to be arbitrary

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_i )</td>
<td>( e_i )</td>
<td>( r_i )</td>
<td>( d_i )</td>
<td>Complexity</td>
</tr>
<tr>
<td>fixed</td>
<td>fixed</td>
<td>arbitrary</td>
<td>arbitrary</td>
<td>NP complete</td>
</tr>
<tr>
<td>fixed</td>
<td>arbitrary</td>
<td>fixed</td>
<td>arbitrary</td>
<td>?? (Open)</td>
</tr>
<tr>
<td>fixed</td>
<td>arbitrary</td>
<td>arbitrary</td>
<td>fixed</td>
<td>NP complete</td>
</tr>
<tr>
<td>arbitrary</td>
<td>fixed</td>
<td>arbitrary</td>
<td>fixed</td>
<td>?? (Open)</td>
</tr>
<tr>
<td>arbitrary</td>
<td>fixed</td>
<td>fixed</td>
<td>arbitrary</td>
<td>NP complete</td>
</tr>
<tr>
<td>arbitrary</td>
<td>arbitrary</td>
<td>fixed</td>
<td>fixed</td>
<td>NP complete</td>
</tr>
</tbody>
</table>

Same results as nonpreemptive transmission except two cases.
11.3.3 On-line algorithms

- On-line Algorithms: Each node in the network routes messages without any knowledge of the messages in other nodes.

- Algorithm A, B & C are on-line algorithms

- For both non-preemptive and preemptive transmissions, there are on-line algorithms when three parameters are fixed (see 11.3.1 & 11.3.2)

- No on-line algorithms can exist if only two parameters are fixed.

- Example: Assume fixed origin nodes and destination nodes. Release times and deadlines are arbitrary.

Let: 
\[ G=\{(1,2,3),\{(1,2),(2,3),(3,1)\}\} \]
\[ M=\{M_1,M_2,M_3\} \]

<table>
<thead>
<tr>
<th>( M_i )</th>
<th>( s_i )</th>
<th>( e_i )</th>
<th>( l_i )</th>
<th>( r_i )</th>
<th>( d_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_1 )</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>( M_2 )</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>11</td>
</tr>
</tbody>
</table>

At time 0, there are two possible cases:

- \( M_1 \) is transmitted at time 0 on the edge (1,2)

Let \( M_3 = (1,3,4,2,10) \) releases at time 2

\( M_3 \) is an urgent message and \( M_2 \) will miss its deadline
But if $M_2$ is transmitted first, then we have a feasible transmission:

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
(1,2) & M_2 & | & M_3 & | & M_1 & /\cdots/ \cdots/ \cdots/ \cdots/ \\
(2,3) & /\cdots/ \cdots/ & M_2 & /\cdots/ & | & M_3 & | & M_1 & /\cdots/
\end{array}
\]

- If $M_1$ is not transmitted at time 0 on the edge (1,2)

Let $M_3 = (1, 3, 4, 1, 9)$ release at time 1

$M_3$ is a urgent message, then either $M_1$ or $M_2$ will miss its deadline

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
(1,2) & M_2 & | & M_3 & | & M_2 & M_1 & /\cdots/ \cdots/ \cdots/ \cdots/ \\
(2,3) & /\cdots/ \cdots/ \cdots/ \cdots/ & M_3 & | & M_2 & | & M_1
\end{array}
\]

But if $M_1$ is transmitted at 0, then we have a feasible transmission:

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
(1,2) & M_1 & | & M_3 & | & M_2 & /\cdots/ \cdots/ \cdots/ \cdots/ \\
(2,3) & /\cdots/ & M_1 & /\cdots/ \cdots/ \cdots/ \cdots/ & M_3 & | & M_2 & /\cdots/
\end{array}
\]