INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Stamp / Signature of the Invigilator

EXAMINATION (Mid Semester)										**SEMESTER (Autumn)**			
Roll Number										**Section**		**Name**	
Subject Number	C	S	6	0	0	2	7	**Subject Name**		***Parallel Algorithms***			
Department / Center of the Student											**Additional sheets**		

Important Instructions and Guidelines for Students

1. You must occupy your seat as per the Examination Schedule/Sitting Plan.
2. Do not keep mobile phones or any similar electronic gadgets with you even in the switched off mode.
3. Loose papers, class notes, books or any such materials must not be in your possession, even if they are irrelevant to the subject you are taking examination.
4. Data book, codes, graph papers, relevant standard tables/charts or any other materials are allowed only when instructed by the paper-setter.
5. Use of instrument box, pencil box and non-programmable calculator is allowed during the examination. However, exchange of these items or any other papers (including question papers) is not permitted.
6. Write on both sides of the answer script and do not tear off any page. **Use last page(s) of the answer script for rough work.** Report to the invigilator if the answer script has torn or distorted page(s).
7. It is your responsibility to ensure that you have signed the Attendance Sheet. Keep your Admit Card/Identity Card on the desk for checking by the invigilator.
8. You may leave the examination hall for wash room or for drinking water for a very short period. Record your absence from the Examination Hall in the register provided. Smoking and the consumption of any kind of beverages are strictly prohibited inside the Examination Hall.
9. Do not leave the Examination Hall without submitting your answer script to the invigilator. **In any case, you are not allowed to take away the answer script with you.** After the completion of the examination, do not leave the seat until the invigilators collect all the answer scripts.
10. During the examination, either inside or outside the Examination Hall, gathering information from any kind of sources or exchanging information with others or any such attempt will be treated as '**unfair means**'. Do not adopt unfair means and do not indulge in unseemly behavior.

Violation of any of the above instructions may lead to severe punishment.

Signature of the Student

To be filled in by the examiner

Question Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total**
Marks Obtained											

Marks obtained (in words)	**Signature of the Examiner**	**Signature of the Scrutineer**

CS60027 Parallel Algorithms, Autumn 2023–2024

Mid-Semester Test

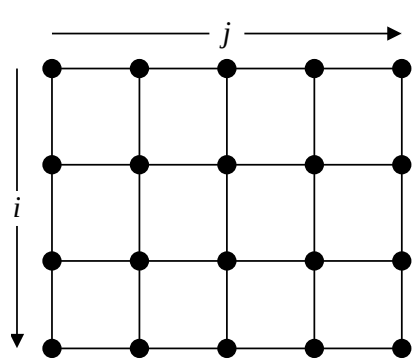
26–September–2023 | 09:00am–11:00am | Maximum marks: 60

[*Write your answers in the question paper itself. Be brief and precise. Answer all questions.*]

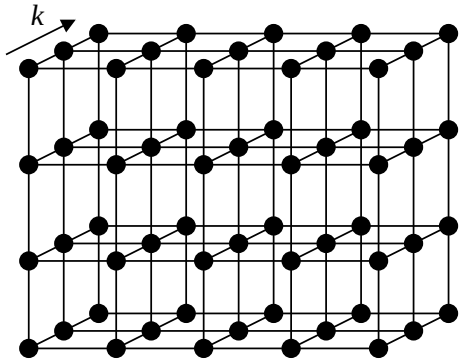
Do not write anything on this page.

Questions start from the next page.

1. A **three-dimensional** $r \times s \times t$ **mesh** is obtained by making t copies of an $r \times s$ mesh and joining the corresponding nodes in a line. An example is given below. We number the nodes in the 3-d mesh as $P_{i,j,k}$, where $1 \leqslant i \leqslant r$, $1 \leqslant j \leqslant s$, and $1 \leqslant k \leqslant t$.



4 x 5 mesh



4 x 5 x 3 mesh

Let $A = (a_1, a_2, \ldots, a_n)$ be an array of n integers, and we want to compute the sum $s = \sum_{i=1}^{n} a_i$. You are given an $m \times m \times m$ mesh with $p = m^3$ processors. Assume that $p \leqslant n$. For the sake of simplicity, you may further assume that n is an integral multiple of p.

(a) Propose an efficient algorithm to compute the sum s on the given 3-d mesh. Clearly specify how the elements of A are initially distributed among the p processors. Also clearly specify the computations and communications involved in the algorithm. What is the running time of your algorithm? **(10)**

Solution Let $c = n/p$. We give disjoint chunks of A with c elements to the processors.

Stage 0: Each node computes the sum of the c elements given to it. All these additions run sequentially in each node, but in parallel across all the nodes. So the (parallel) running time of this step in $\Theta(c)$.

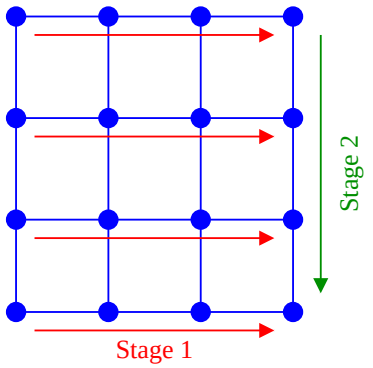
For each fixed k, the nodes $P_{i,j,k}$ form a 2-d mesh called the k-th layer.

Stage 1: Partial sums are relayed in each layer towards right. Each node, upon receiving the partial sum from its left neighbor, adds its own partial sum to the sum received, and sends the updated partial sum to its right neighbor. This takes $\Theta(m)$ time.

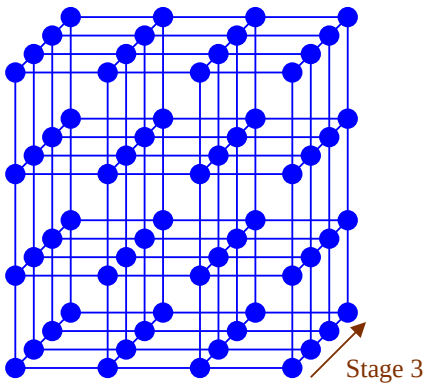
Stage 2: Then, the rightmost nodes in each layer relays their updated partial sums downward. Each node, upon receiving the partial sum from its top neighbor, adds its partial sum to the sum received, and sends this updated partial sum to its neighbor below. Eventually, the nodes $P_{m,m,k}$ store the sums of all the array elements given to all the nodes in the respective layers. This stage also runs in $\Theta(m)$ time.

Stage 3: Finally, the nodes $P_{m,m,k}$ keeps on relaying and updating the partial sums along the third dimension. This stage also takes $\Theta(m)$ time. The final sum s is available at the node $P_{m,m,m}$.

The communication pattern is illustrated in the figure below.



Communication in each layer



Communication across layers

The overall running time (including computation and communication) of this algorithm is

$$\Theta(c+m) = \Theta\left(\frac{n}{p} + m\right) = \Theta\left(\frac{n}{m^3} + m\right).$$

(b) The sum s can be computed sequentially in $T^*(n) = \Theta(n)$ time. Let the running time of your algorithm in Part (a) be $T_p(n)$. The speedup is $S_p(n) = \frac{T^*(n)}{T_p(n)}$. For what value of m is $S_p(n)$ maximized? Justify. **(5)**

Solution Intuitively, the speedup is maximized if the computation and communication times are asymptotically the same, that is, $\frac{n}{m^3} = m$, that is, $m = n^{\frac{1}{4}}$. The corresponding speedup is $\Theta(n^{\frac{1}{4}})$.

If $m = n^{\frac{1}{4}-\varepsilon}$, then the computation time dominates giving $T_p(n) = \Theta(n^{\frac{1}{4}+3\varepsilon})$, that is, $S_p(n) = \Theta(n^{\frac{1}{4}-3\varepsilon})$.

If $m = n^{\frac{1}{4}+\varepsilon}$, then the communication time dominates giving $T_p(n) = \Theta(n^{\frac{1}{4}+\varepsilon})$, that is, $S_p(n) = \Theta(n^{\frac{1}{4}-\varepsilon})$.

2. Use an **EREW PRAM** for solving the following parts. **Do not use sorting** in any of these parts. Your algorithms need not be optimal.

(a) Suppose that n processors of the PRAM need to read a variable x in the shared memory simultaneously. But concurrent reading is not allowed. Sequential reading takes $O(n)$ time. Explain how the simultaneous reading of x can be done in $O(\log n)$ time. What is the work done? **(3)**

Solution In the first time unit, one copy of x is made. In the second time unit, copies are made in parallel for each of the two copies of x. In the third time unit, four new copies are created in parallel. After $\log n$ units of time, n copies of x are available. These copies can be exclusively read by the n processors. The total work done is $O(n)$.

(b) Let $A = (a_1, a_2, \ldots, a_n)$ be an *unsorted* array of integers. Recall that for an integer x, the rank of x in A, denoted $\text{rank}(x : A)$, is the number of elements of A, that are $\leqslant x$. Propose an $O(\log n)$-time $O(n)$-work algorithm for computing $\text{rank}(x : A)$. Note that x is a shared variable. **(4)**

Solution The algorithm is given below.

First make n copies of x as in Part (a) (all these copies are called x).
For $i = 1, 2, \ldots, n$, pardo:
    If $A(i) \leqslant x$, set $C(i) = 1$, else set $C(i) = 0$.
Compute the sum s of the elements of the array C, in parallel.
Return s.

The initial copy part takes $O(\log n)$ time using $O(n)$ work. Populating the array C can be done in $O(1)$ time using $O(n)$ work. Finally, the sum s can be computed in $O(\log n)$ time using $O(n)$ work. The total work done is $O(n)$.

(c) Let A be as in Part (b). We want to compute the array $R = (r_1, r_2, \ldots, r_n)$, where $r_i = \text{rank}(a_i : A)$. Propose an $O(\log n)$-time algorithm for this problem. What is the work done by this algorithm? **(4)**

Solution For $i = 1, 2, \ldots, n$, we compute $r_i = \text{rank}(a_i : A)$ in parallel, using the algorithm of Part (b). The running time remains $O(\log n)$. The work done becomes $O(n^2)$.

(d) We want to determine whether A contains one or more duplicate (that is, repeated) elements. Given the array R computed in Part (b), solve this problem in $O(\log n)$ time using $O(n)$ work. **(4)**

Solution If A does not contain repeated elements, then R is a permutation of $1, 2, 3, \ldots, n$, so the sum S of the elements of R will be $n(n+1)/2$. On the other hand, if A contains repeated elements, we will have $S > n(n+1)/2$. We can compute S in $O(\log n)$ time using $O(n)$ work. The computation of $n(n+1)/2$ and the comparison of S with this quantity can be done in $O(1)$ sequential time.

3. Let T be a rooted tree on n nodes numbered $1,2,3,\ldots,n$. The tree is given as an array $P=(p_1,p_2,\ldots,p_n)$, where p_i is the parent of i. If r is the root, then $p_r=0$. Develop **CREW PRAM** algorithms for solving the following parts. Write the complete WT presentation of each algorithm in this exercise. Do not use any algorithm covered in the class, as a subroutine.

(a) Propose an algorithm to identify the root and store it in a global variable r, in $O(1)$ time. What is the work done by your algorithm? **(3)**

Solution We do not need pointer jumping for solving this problem, because there is a single root node. The following algorithm can be used.

For $i=1,2,\ldots,n$, pardo:
  If $P[i]=0$, set $r=i$.

No concurrent write is required here, because there is a unique i (the root) for which $P(i)=0$.

The work done by this algorithm is $O(n)$.

(b) The level of a node in T is its distance from the root of T. Thus the root has level 0, the children of the root have level 1, the grandchildren of the root have level 2, and so on. Propose an $O(\log n)$-time algorithm to compute the array $L=(l_1,l_2,\ldots,l_n)$, where l_i is the level of the node i in T. What is the work done by your algorithm? **(6)**

Solution We assume that the root r is computed as in Part (a), Then, we do pointer jumping. $Q[i]$ stores an ancestor of i, and $D[i]$ stores the distance of $Q[i]$ from i.

For $i=1,2,\ldots,n$ pardo:
  If $i=r$, then set $L[i]=0$, else set $L[i]=-1$.
  Set $Q[i]=P[i]$.
  If $i=r$, then set $D[i]=0$, else set $D[i]=1$.
  While $L[i]=-1$, repeat:
    If $L[Q[i]]\neq -1$, then
      set $L[i]=D[i]+L[Q[i]]$,
    else
      set $Q[i]=Q[Q[i]]$, and
      set $D[i]=2\times D[i]$.

The work done by this algorithm is $O(n\log n)$.

(c) Propose an $O(\log n)$-time algorithm to 2-color T. What is the work done by your algorithm? **(6)**

Solution Let the colors be 1 and 2. We create an array $C = (c_1, c_2, \ldots, c_n)$, where $c_i \in \{1,2\}$ is the color of the node i. We assume that the level array L computed in Part (b) is available.

```
For i = 1,2,...,n, pardo:
    If L[i] is even,
        set C[i] = 1,
    else
        set C[i] = 2.
```

This algorithm has $O(1)$ running time, and the work done is $O(n)$.

Combining the results of Parts (a) and (b), we conclude that the 2-coloring algorithm has $O(\log n)$ overall running time and does $O(n \log n)$ work in total.

4. (a) Suppose we are given an array S of size n representing the nodes of a set of linked lists. Each node i has at most one successor whose index is $S[i]$. Suppose that array indices start from 1, and $S[i] = 0$ if the node with index i has no successor. Show how to rank the nodes within all the lists in $O(\log n \log\log n)$ time and $O(n)$ operations. Give an analysis of the time complexity and the number of operations. **(3)**

Solution The following algorithm done in the class works with a slight modification. Here, we fix the number of iterations while the algo in the book keeps track of the size of the list, and exits the loop when it falls below $n/\log n$.

1. Repeat the following for $O(\log\log n)$ iterations:
   (a) Color the nodes using 3 colors using $O(\log n)$ time and $O(n)$ work.
   (b) Remove the set I of local minima.
2. Apply the pointer jumping algorithm to the resulting set of nodes.
3. Restore the original list and ranks by inserting back the independent sets in reversed order.

Analysis: Suppose there are k lists with lengths $n_1, \ldots, n_k$ such that $n_1 + \cdots + n_k = n$.

Analysis of the loop: Runtime analysis is exactly the same as the original algo, giving $O(\log n \log\log n)$ time. Number of operations in iteration ℓ is at most $O\left(\sum_{i=1}^{k} \left(\frac{4}{5}\right)^{\ell} \cdot n_i\right) = O\left(\left(\frac{4}{5}\right)^{\ell} \cdot n\right)$; so the total number of operations in the loop is $O(n)$.

After the loop, at Step 2, the i-th list has size $O(n_i/\log n_i)$. The number of operations in Step 2 is $\sum_{i=1}^{k} O\left(\frac{n_i}{\log n_i} \log\left(\frac{n_i}{\log n_i}\right)\right) = O(n)$. Runtime of Step 2 is $O(\log n)$ by standard analysis of pointer jumping.

Analysis of step 3 is the same as in the original algo.

(b) Suppose that you are given a list represented by an array S of size n as follows. Index 1 is the first node of the list. However, each index > 1 is not necessarily a node of the list. If index i is a node of the list, then $S[i]$ is the index of its successor, or 0 if it is the last node of the list. You are also given another array N of n numbers. Moreover, suppose that you are given an array R such that for each index i in the list, $R[i]$ contains the serial number of i from the start of the list. That is, if i is the k-th node of the list, then $R[i] = k$ (in particular, $R[1] = 1$). In all of the three input arrays, entries corresponding to indices which are not parts of the list can be arbitrary numbers. Assume that the entries of the array R are distinct. Give an algorithm that finds the minimum $N[i]$ for any index i that is a node of the list. Your algorithm should run in $O(\log n)$ time and use $O(n)$ operations. Demonstrate your algorithm on the following input:

$S = (5,3,0,0,2,9)$
$N = (0,3,-3,-2,-1,5)$
$R = (1,3,4,5,2,7)$. **(12)**

Solution

1. for $i = 1$ to n pardo:
   $\mathsf{minim}[i] \leftarrow N[i]$;
2. $k \leftarrow 1$
3. for $1 \leqslant i \leqslant n$ pardo:
   while(1) do:
   if $R[i] \notin [1,n]$ or $(R[i]-1) \bmod k \neq 0$ or $S[1] = 0$:
   exit;
   if $S[i] \neq 0$ and $S[i] \leqslant n$:
   $\mathsf{minim}[i] \leftarrow \min\{\mathsf{minim}[i], \mathsf{minim}[S[i]]\}$;
   $S[i] \leftarrow S[S[i]]$;
   if $i = 1$:
   $k \leftarrow 2 \times k$;
4. return $\mathsf{minim}[1]$.

By standard analysis of pointer jumping, runtime = $O(\log n)$.

Total number of operations in Steps 1 and 2: $O(n)$. Number of operations in the ℓ-th iteration of the loop is at most $\lceil n/2^{\ell-1} \rceil$. Number of operations in Step 4 is $O(1)$. Thus the total number of operations is $O(n)$.

Demonstration on the given input:

Stage	Active indices	S at the end of the iteration	minim at the end of the iteration
Before the loop	–	$(5,3,0,0,2,9)$	$(0,3,-3,-2,-1,5)$
Iteration 1	$\{1,2,3,4,5\}$	$(2,0,0,0,3,9)$	$(-1,-3,-3,-2,-1,5)$
Iteration 2	$\{1,2,4\}$	$(0,0,0,0,3,9)$	$(-3,-3,-3,-2,-1,5)$

Output: -3.