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	6	Subject Name	*Parallel and Distributed 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

CS60026 Parallel and Distributed Algorithms, Autumn 2018–2019

Mid-Semester Test

17–September–2018 NR-121/122/221/222, 02:00–04:00 pm Maximum marks: 50

Write your answers in the question paper itself. Be brief and precise. Answer <u>all</u> questions.
If you use any algorithm/result/formula covered in the class, just mention it, do not elaborate.

1. Let A, B be $n \times n$ matrices, and we want to compute their product $C = AB$. Although this problem can be solved in $o(n^3)$ time, let us take the best sequential running time of a simple matrix-multiplication algorithm as $T^*(n) = \Theta(n^3)$. Consider the following parallel algorithm in the WT presentation level for computing C.

1. For all $i, j, k \in \{1, 2, 3, \ldots, n\}$ pardo: Set $C(i, j, k) = A(i, k)B(k, j)$.
2. For all $i, j \in \{1, 2, 3, \ldots, n\}$ pardo:

   Use the balanced-binary tree method to compute $C(i, j) = \sum_{k=1}^{n} C(i, j, k)$ in parallel.

Let us use p processors to run this algorithm. Recall that p is called an *optimal number of processors* if it is as large as possible and the speedup (relative to $T^*(n)$) is still $\Theta(p)$. Deduce the optimal number of processors for this algorithm. What is the parallel running time of this algorithm if it uses the optimal number of processors? **(7+3)**

Solution Step 1 runs in $\Theta(1)$ time and does $\Theta(n^3)$ work. The BBT method of Step 2 to compute each n-fold sum takes $\Theta(\log n)$ time and does $\Theta(n)$ work, so Step 2 runs in $\Theta(\log n)$ parallel time and makes a total of $\Theta(n^3)$ work. To sum up, the give algorithm has

$$W(n) = \Theta(n^3),$$
$$T(n) = \Theta(\log n).$$

By Brent's WT scheduling principle, the parallel running time on p processors is then

$$T_p(n) = O\left(\frac{n^3}{p} + \log n\right),$$

that is, the speedup is

$$S_p(n) = \frac{T^*(n)}{T_p(n)} = O\left(\frac{pn^3}{n^3 + p\log n}\right).$$

It follows that $S_p(n) = \Theta(p)$ if $p\log n = \Theta(n^3)$, that is, $p = \Theta\left(\frac{n^3}{\log n}\right)$. Plugging in this value of p in the expression for $T_p(n)$ gives $T_p(n) = \Theta(\log n)$.

2. An online merchant stores its item information in a secure array $A = (a_1, a_2, \ldots, a_n)$, where each a_i is a non-negative integer (which may stand for the count of the i-th item for sale). At the beginning of each day (5 am), the company makes a copy $B = (b_1, b_2, \ldots, b_n)$ of A. Daily transactions modify the copy B. During the maintenance hour (4–5 am), the changes made in the copy B are carefully examined by the audit department, and subsequently B is copied back to A before 5 am. Since n is large, the audit department asks for your help to design an efficient parallel algorithm to find out which entries of B changed during the day. Let these indices be $1 \leqslant i_1 \leqslant i_2 \leqslant \cdots \leqslant i_k \leqslant n$ for some k (depending on the day). Your task is to compute k and the index array $C = (i_1, i_2, \ldots, i_k)$ so that the audit department can focus on these items only. Propose an $O(\log n)$-time $O(n)$-work CREW PRAM algorithm to solve this problem. The two arrays A and B (and their size n) are only provided as input to your algorithm. **(10)**

Solution We use two additional arrays U, V of size n. The algorithm has the following steps.

1. For $i = 1, 2, 3, \ldots, n$, pardo: If $a_i \neq b_i$, set $U(i) = 1$, else set $U(i) = 0$.
2. Compute the prefix sums of U to V.
3. Set $k = V(n)$.
4. For $i = 1, 2, 3, \ldots, n$, pardo: If $U(i) = 1$, set $C(V(i)) = i$.

Steps 1 and 4 take $O(1)$ time and do $O(n)$ work. Step 3 requires $O(1)$ time and $O(1)$ work. Finally, Step 2 can be implemented to run in $O(\log n)$ time using $O(n)$ work (on a CREW or EREW PRAM).

3. Let n be a power of 2, and $A(x)$ and $B(x)$ two polynomials (with numeric coefficients) of degree $n-1$. The polynomials are presented to you as two arrays $A = (a_0, a_1, a_2, \ldots, a_{n-1})$ and $B = (b_0, b_1, b_2, \ldots, b_{n-1})$, where a_i and b_i are the coefficients of x^i in $A(x)$ and $B(x)$, respectively. Propose an $O(\log n)$-time parallel algorithm to compute the product polynomial $C(x) = A(x)B(x)$. What PRAM model do you use? What is the work done by your algorithm? **(6+2+2)**

Solution Let us use a divide-and-conquer algorithm. Consider the half-sized polynomials $A_{lo} = \left(a_0, a_1, a_2, \ldots, a_{\frac{n}{2}-1}\right)$, $A_{hi} = \left(a_{\frac{n}{2}}, a_{\frac{n}{2}+1}, \ldots, a_{n-1}\right)$, $B_{lo} = \left(b_0, b_1, b_2, \ldots, b_{\frac{n}{2}-1}\right)$, and $B_{hi} = \left(b_{\frac{n}{2}}, b_{\frac{n}{2}+1}, \ldots, b_{n-1}\right)$. We have

$$\begin{aligned} A(x) &= x^{\frac{n}{2}} A_{hi}(x) + A_{lo}(x), \\ B(x) &= x^{\frac{n}{2}} B_{hi}(x) + B_{lo}(x). \end{aligned}$$

Therefore their product is

$$C(x) = x^n A_{hi}(x) B_{hi}(x) + x^{\frac{n}{2}} \left(A_{hi}(x) B_{lo}(x) + A_{lo}(x) B_{hi}(x) \right) + A_{lo}(x) B_{lo}(x).$$

This gives the following recursive algorithm.

1. Compute $A_{hi}, A_{lo}, B_{hi}, B_{lo}$ by halving the input arrays.
2. Compute the four products $D_3 = A_{hi}B_{hi}, D_2 = A_{hi}B_{lo}, D_1 = A_{lo}B_{hi}, D_0 = A_{lo}B_{lo}$ recursively *in parallel*.
3. For $i = 0, 1, 2, \ldots, 2n-2$, pardo: Set $C(i) = 0$.
4. For $i = 0, 1, 2, \ldots, n-2$, pardo:
   - Add $D_3(i)$ to $C(n+i)$.
   - Add $D_2(i)$ to $C(\frac{n}{2}+i)$.
   - Add $D_1(i)$ to $C(\frac{n}{2}+i)$.
   - Add $D_0(i)$ to $C(i)$.

We have the recurrence relations:

$$\begin{aligned} T(n) &= T(n/2) + \Theta(1), \\ W(n) &= 4W(n/2) + \Theta(n). \end{aligned}$$

The solutions of these are:

$$\begin{aligned} T(n) &= \Theta(\log n), \\ W(n) &= \Theta(n^2). \end{aligned}$$

Since concurrent write is not needed, this algorithm runs on a CREW PRAM. Step 2 requires concurrent reads by the recursive calls. However, if we make copies of half-sized polynomials for each recursive call (can be done in $\Theta(1)$ parallel time using $\Theta(n)$ work), then this algorithm also runs on an EREW PRAM.

Note: This problem can also be solved non-recursively on a CREW/EREW PRAM within the same time and work bounds. Compute and store all the products $p_{ij} = a_i b_j$. Then, compute the coefficients of C in parallel using the formula $c_k = \sum_{\substack{i,j \\ i+j=k}} p_{ij}$ by the balanced binary tree method. The handling of the indices i, j with $i + j = k$ is a little bit problematic, but can in principle be done.

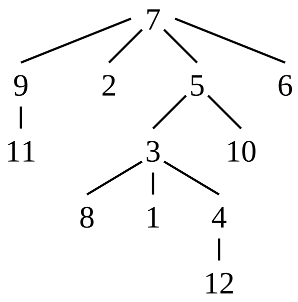
4. In the class, we have seen a common-CRCW PRAM algorithm to find the maximum of an array A of n numbers in $O(1)$ time using n^2 processors. For simplicity, let n be a perfect fifth power. Propose a common-CRCW PRAM algorithm to compute the maximum of A in $O(1)$ time but using only $n^{6/5}$ processors. **(10)**

Solution Let us call the algorithm using quadratic number of processors (as discussed in the class) the *basic maximum* algorithm. Given this algorithm, we can design the following algorithm.

1. Break A into $n^{4/5}$ chunks B_i each of size $n^{1/5}$.
2. Compute the $n^{4/5}$ maximums $B = (b_1, b_2, \ldots, b_{n^{4/5}})$ of B_i in parallel by the basic maximum algorithm.
3. Break B into $n^{3/5}$ chunks C_i each of size $n^{1/5}$.
4. Compute the $n^{3/5}$ maximums $C = (c_1, c_2, \ldots, c_{n^{3/5}})$ of C_i in parallel by the basic maximum algorithm.
5. Compute and return the maximum of C by the basic maximum algorithm.

Clearly, each step requires $O(1)$ time (given sufficiently many processors). In order to estimate the processor requirement, we note that Step 2 requires $(n^{1/5})^2 \times n^{4/5} = n^{6/5}$ processors, Step 4 requires $(n^{1/5})^2 \times n^{3/5} = n$ processors, and Step 5 requires $(n^{3/5})^2 = n^{6/5}$ processors.

5. Let $T = (V, E)$ be a rooted tree with $V = \{1, 2, 3, \ldots, n\}$, and with some $r \in V$ specified as the root. Each node in T has a specific ordering of its children, so we can talk about the first, second, third, ... children of a node. Also, the next sibling of the i-th child u of v is the $(i+1)$-st child of v. The tree T is supplied to you by n, r and two arrays fc and ns of size n each (and by nothing else). For each $u \in V$, the element $fc(u)$ is the first child of u, or 0 is u does not have a child. Likewise, for each $u \in V$, the element $ns(u)$ is the next sibling of u, or 0 if u does not have a next sibling. The following figure gives an example ($n = 12$ and $r = 7$, with left-to-right child ordering). Your task is to compute an array p of size n such that $p(u)$ stores the parent of u for all $u \in V$ (we have $p(r) = 0$ for the root r). Propose an $O(\log n)$-time EREW PRAM algorithm to solve this problem. What is the work done by your algorithm? **(8+2)**



Input

	1	2	3	4	5	6	7	8	9	10	11	12
fc	0	0	8	12	3	0	9	0	11	0	0	0
ns	4	5	10	0	6	0	0	1	2	0	0	0

Output

	1	2	3	4	5	6	7	8	9	10	11	12
p	3	7	5	3	7	7	0	3	7	5	9	4

Solution First, each parent copies its number to its first child. Then, the first child copies the parent number to its siblings by the pointer-jumping technique. Let v have k children $u_1, u_2, u_3, \ldots, u_k$. The parent sets $p(u_1) = v$. Then, u_1 copies v to $p(u_2)$, then u_1, u_2 copy v to $p(u_3)$ and $p(u_4)$, then u_1, u_2, u_3, u_4 copy v to $p(u_5), p(u_6), p(u_7), p(u_8)$, and so on. It is clear that all siblings of u_1 eventually get the parent number v in $O(\log k)$ parallel steps. Since $k \leqslant n - 1$, the desired parallel running time follows.

For $u = 1, 2, 3, \ldots, n$ pardo:

1. Initialize $p(u) = 0$.
2. If $fc(u) \neq 0$, set $p(fc(u)) = u$.
3. Make a copy $S(u) = ns(u)$.
4. While $S(u) \neq 0$, repeat: /* Sequential pointer-jumping loop */
   - If $p(u) \neq 0$, copy $p(u)$ to $p(S(u))$.
   - Set $S(u) = S(S(u))$.

The work done by this algorithm is $O(n \log n)$ (since each node has $\leqslant n - 1$ children). If the *busy wait* based upon the condition $p(u) \neq 0$ can be avoided, the work done can be derived to be $O(n)$.

Use this space for leftover answers and rough work

Use this space for leftover answers and rough work