INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Stamp / Signature of the Invigilator

EXAMINATION (End 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

End-Semester Test

15–November–2018 F116/F142/NC341/NC342, 02:00–05:00 pm Maximum marks: 60

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

1. Let $A = (a_{i,j})$ be an invertible $n \times n$ matrix with real-valued entries. Your task is to compute $B = A^{-1}$. One popular algorithm for this computation involves Gaussian elimination. Start with $B = I_n$ (where I_n is the $n \times n$ identity matrix). Applying elementary row operations (exchanging two rows, multiplying a row by a non-zero value, subtracting a multiple of a row from another), convert A to the identity matrix I_n. Apply the same sequence of elementary row operations to B. When A is reduced to I_n, B changes from I_n to A^{-1}. Consider this algorithm as an optimal sequential algorithm for inverting a matrix.

(a) Develop an $O(n \log n)$-time CREW-PRAM algorithm to implement the above matrix-inversion method. Is your algorithm optimal? **(6 + 2)**

Solution Here is a WT-level presentation of the algorithm.

Initialization

1. For $i, j \in \{1,2,3,\ldots,n\}$, **pardo**: Set $b_{i,j} := \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$

Row-reduction loop

2. For $i = 1,2,3,\ldots,n$, **do**:
   (a) Find one $k \in \{i, i+1, i+2, \ldots, n\}$ such that $a_{k,i} \neq 0$ (see below).
   (b) If $k \neq i$, for all $j = 1,2,3,\ldots,n$, **pardo**:
   Swap $a_{i,j}$ with $a_{k,j}$, and $b_{i,j}$ with $b_{k,j}$.
   (c) For $j \in \{1,2,3,\ldots,n\}$, **pardo**:
   Divide $a_{i,j}$ by $a_{i,i}$, and $b_{i,j}$ by $a_{i,i}$.
   (d) For $k \in \{1,2,3,\ldots,n\}$ with $k \neq i$ and $a_{k,i} \neq 0$, and for $j \in \{1,2,3,\ldots,n\}$, **pardo**:
   Subtract $a_{k,i}a_{i,j}$ from $a_{k,j}$, and $a_{k,i}b_{i,j}$ from $b_{k,j}$.

It remains to elaborate how Step 2(a) can be implemented. Use an array $U[i-1 \ldots n]$ initialized to $U[i-1] = 0$, and $U_k = \begin{cases} 0 & \text{if } a_{k,i} = 0 \\ 1 & \text{if } a_{k,i} \neq 0 \end{cases}$ for $i \leqslant k \leqslant n$. Compute the prefix sums of U in an array V. Identify k in parallel as the index $k \in \{i, i+1, i+2, \ldots, n\}$ such that $V[k-1] = 0$ and $V[k] = 1$. Since A is invertible, this k must exist. Also, k is uniquely identified, so concurrent write is not needed.

Parallel running time

Step 1 takes $O(1)$ time. For each i in the row-reduction loop, Step (a) takes $O(\log n)$ time, whereas Steps (b)–(d) take $O(1)$ time. Since the row-reduction loop (Step 2) is sequential, the total running time is $O(n \log n)$.

Optimality

We have $T^*(n) = \Theta(n^3)$. Step 1 requires $\Theta(n^2)$ effort. For each i in the row-reduction loop, Step (a) can be implemented so as to perform only $O(n)$ work, whereas Steps (b) and (c) do $O(n)$ work, and Step (d) does $O(n^2)$ work. Thus, the overall row-reduction process makes $O(n^3)$ work.

Therefore, this algorithm is optimal.

(b) If you are given a CRCW PRAM, how can you improve the running time to $O(n)$? Mention which type of CRCW PRAM you use (only common, priority and arbitrary variants are allowed). Is your improved algorithm optimal? **(4 + 2 + 2)**

Solution We can indeed use any k with $a_{k,i} \neq 0$ in Step 2(a) as the pivot (assuming that floating-point approximation errors and the possible instability of the algorithm are not an issue). So whenever $a_{k,i} \neq 0$, attempt a concurrent write of k. A priority-CRCW PRAM succeeds is writing the first such k, whereas an arbitrary-CRCW PRAM writes any k with $a_{k,i} \neq 0$. So Step 2(a) now runs in $O(1)$ time, and the running time of the overall algorithm reduces to $O(n)$.

The modified Step 2(a) still requires $O(n)$ work, so the total work done by the modified algorithm remains the same, namely, $O(n^3)$, that is, this CRCW PRAM algorithm is optimal too.

(c) What are the optimal numbers of processors for your algorithms of Part (a) and Part (b)? **(2 + 2)**

Solution Let us use the formula $p_{opt} = \Theta(T^*(n)/T(n))$ to obtain the optimal number of processors in each case.

Part (a): $p_{opt} = \Theta(\frac{n^2}{\log n})$.

Part (b): $p_{opt} = \Theta(n^2)$.

2. The parallel algorithms of Exercise 1 do not run in polylogarithmic time. If it is given that A is a *lower triangular*[1] invertible $n \times n$ matrix with real-valued entries, develop a polylogarithmic-time CREW-PRAM algorithm to invert A. Deduce the running time and the work done by your algorithm. **(7 + 3)**

(**Hint:** Break each of A and A^{-1} into four $\frac{n}{2} \times \frac{n}{2}$ blocks, and use the fact that A^{-1} is again lower triangular.)

Solution Write $A = \begin{pmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{pmatrix}$ and $B = A^{-1} = \begin{pmatrix} B_{11} & 0 \\ B_{21} & B_{22} \end{pmatrix}$, where each A_{ij} and each B_{ij} are $\frac{n}{2} \times \frac{n}{2}$ blocks. Since $AB = I_n$, we have $A_{11}B_{11} = I_{n/2}$, $A_{21}B_{11} + A_{22}B_{21} = 0$, and $A_{22}B_{22} = I_{n/2}$. This, in turn, implies that $B_{11} = A_{11}^{-1}$, $B_{22} = A_{22}^{-1}$, and $B_{21} = -A_{22}^{-1}A_{21}A_{11}^{-1}$. Since A_{11} and A_{22} are again lower triangular, we recursively compute their inverses in parallel. Subsequently, B_{21} can be computed by two multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices. This can be done in $O(\log n)$ time using $O(n^3)$ work. This implies that

$$\begin{aligned} T(n) &= T(n/2) + O(\log n), \\ W(n) &= 2W(n/2) + O(n^3). \end{aligned}$$

By the master theorem of divide-and-conquer recurrences, we therefore have:

$$\begin{aligned} T(n) &= O(\log^2 n), \\ W(n) &= O(n^3). \end{aligned}$$

[1] A square matrix is called *lower triangular* if all the entries above its main diagonal are zero. The main diagonal of an *invertible* lower triangular matrix must consist of non-zero entries only.

3. An array A of n bits consists of n_0 zero bits followed by n_1 one bits (so $n = n_0 + n_1$). For simplicity, assume that $n_0, n_1 > 0$, and that n is a perfect fourth power, so you can write $n = s^2 = t^4$ for some positive integers s, t. In the following two parts, develop EREW-PRAM algorithms to compute the counts n_0 and n_1.

(a) If only $s = \sqrt{n}$ processors are available, solve the problem in $O(1)$ time.[2] **(5)**

Solution Since $n_1 = n - n_0$, it suffices to compute n_0 only.

Break A into s chunks each of size s. Call the i-th chunk A_i. Perform the following steps.

1. For $i = 1, 2, 3, \ldots, s-1$, **pardo**:
   If the last bit of A_i is zero, and the first bit of A_{i+1} is one, return $n_0 = is$.
2. For $i = 1, 2, 3, \ldots, s$, **pardo**:
   If the first bit of A_i is zero, and the last bit of A_i is one, record this i (this happens for a unique i).
3. For $j = 1, 2, 3, \ldots, s-1$ and for the recorded i, **pardo**:
   If the j-th bit of A_i is zero, and the $(j+1)$-st bit of A_i is one, return $n_0 = (i-1)s + j$.

(b) If only $t = \sqrt[4]{n}$ processors are available, solve the problem again in $O(1)$ time. **(5)**

Solution We repeatedly use partitioning of A until its size reduces to t. We use steps analogous to those in Part (a).

Break A into t chunks each of size $t^3 = n^{3/4}$. Run (the equivalent of) Step 1 for these chunks. If n_0 is found, we are done. Otherwise, we confine the search to a single chunk B of size t^3, discovered by running Step 2.

Break B into t subchunks each of size t^2. If Step 1 succeeds on B, we are done, else the search is now restricted to one subchunk C of size t^2, again identified by running Step 2.

Finally, break C into t chunks each of size t, and carry out the three steps of Part (a).

Errata: These results pertain to the CREW PRAM model. My error (What is it?) was pointed out by *Sumeet Shirgure*.

[2] An algorithm for Part (b) can solve Part (a) by keeping $s - t$ processors idle. Here, you are required to utilize all the available processors effectively. Indeed, the work done by your algorithms for the two parts should be $\Theta(s)$ and $\Theta(t)$, respectively.

4. [*Counting sort*] Let A be an array of n integers, each in the range $1,2,3,\ldots,k$. It is given that $k = O(\log n)$. Propose an optimal $O(\log n)$-time CREW-PRAM algorithm to sort A. The sorted output is to be written in a separate array B. Your algorithm should be stable, that is, equal keys preserve the order in B as they appear in A (this matters if the elements of A contain satellite data other than the keys in the range $1,2,3,\ldots,k$). **(10)**

Solution We can compute the prefix sums for each key $1,2,3,\ldots,k$, and subsequently relocate the elements of A to appropriate locations in B. This approach takes $O(\log n)$ time, but does $O(nk) = O(n \log n)$ work. In order to reduce the work to $O(n)$ (this is optimal since sequential counting sort runs in $O(n+k) = O(n)$ time), we use the ideas of accelerated cascading and partitioning.

1. Let $m = \lceil \log n \rceil$. Partition A into chunks $A_1, A_2, A_3, \ldots, A_{n'}$ of size m, where $n' = \lceil n/m \rceil$ (the last chunk may contain less than m elements). In the rest of this algorithm, we use i to denote a chunk number ($1 \leqslant i \leqslant n'$), l to denote a position in a chunk ($1 \leqslant l \leqslant m$), j to denote a key ($1 \leqslant j \leqslant k$), and t to denote a position in A ($1 \leqslant t \leqslant n$).
2. For $i = 1,2,3,\ldots,n'$, **pardo**:
   (a) Use the sequential stable counting sort algorithm to sort A_i.
   (b) As a byproduct, we get the counts $C_{i,j}$ of the keys $j \in \{1,2,3,\ldots,k\}$ in the chunk A_i.
   (c) Set $p = 0$ (the previous key seen).
   (d) For $l = 1,2,3,\ldots,m$, **do**:
   If $(A_i[l] = p)$, set $U_{i,l} = U_{i,l-1} + 1$, else set $U_{i,l} = 1$ and $p = A_i[l]$.
3. For $j = 1,2,3,\ldots,k$, **pardo**:
   (a) Compute the prefix sums of the n' counts $C_{1,j}, C_{2,j}, C_{3,j}, \ldots, C_{n',j}$ in an array D_j.
   (b) Also take $D_j[0] = 0$.
4. Compute the prefix sums of $D_1[n'], D_2[n'], D_3[n'], \ldots, D_k[n']$ in an array E. Also take $E[0] = 0$.
5. For $t = 1,2,3,\ldots,n$, **pardo**:
   (a) Compute the index i of the chunk A_i, to which a_t belongs (in fact, $i = 1 + \lfloor (t-1)/m \rfloor$).
   (b) Also compute the position l of a_t in A_i (in fact, $l = t - (i-1)m$).
   (c) Set $B[E[a_t - 1] + D_{a_t}[i-1] + U_{i,l}] = a_t$.

Let us now enumerate the running times and the works done for the individual steps.

Step	Parallel running time	Work done
Step 1 (computation of m and n')	$O(1)$	$O(1)$
Steps 2(a),(b) for each i	$O(m+k) = O(\log n)$	$O(m+k) = O(\log n)$
Steps 2(c),(d) for each i	$O(m) = O(\log n)$	$O(m) = O(\log n)$
Step 2 (total)	$O(\log n)$	$O(n' \log n) = O(n)$
Step 3 (total)	$O(\log n') = O(\log n)$	$O(kn') = O(n)$
Step 4 (total)	$O(\log k) = O(\log \log n)$	$O(k) = O(\log n)$
Step 5 (total)	$O(1)$	$O(n)$

5. Let C be an arithmetic circuit given by a sequence $(h_1, h_2, h_3, \ldots, h_n)$, where each h_i is one of the following: (i) a small constant integer (positive/negative/zero) [input], (ii) $h_i = h_j + h_k$ for some $j,k < i$ [addition gate], (iii) $h_i = h_j - h_k$ for some $j,k < i$ [subtraction gate], (iv) $h_i = h_j h_k$ for some $j,k < i$ [multiplication gate]. Your task is to determine whether the output h_n of the arithmetic circuit C is zero or non-zero.

(a) Prove that this problem is P-Complete. **(7)**

Solution A given arithmetic circuit can be evaluated in linear time, so ARITHMETIC_CIRCUIT is in P. We now reduce CVP to ARITHMETIC_CIRCUIT. Let $(g_1, g_2, g_3, \ldots, g_n)$ be an instance of CVP. The reduced instance contains the corresponding values $h_1, h_2, h_3, \ldots, h_n$ along with some new variables.

If g_i is an input, take h_i equal to the value of g_i (zero or one, which are small integers).

If g_i is an AND gate, say, $g_i = g_j \wedge g_k$ for some $j,k < i$, take $h_i = h_j h_k$.

If g_i is an OR gate, say, $g_i = g_j \vee g_k$ for some $j,k < i$, take $h_i = h_j + h_k - h_j h_k$. This construction requires some intermediate arithmetic gates. For example, we can realize this as $y_i = h_j + h_k$, $z_i = h_j h_k$, and $h_i = y_i - z_i$.

If g_i is a NOT gate, say, $g_i = g'_j$ for some $j < i$, we take $h_i = 1 - h_j$. This calls for the introduction of a new input whose value is 1 and a subtraction gate: $w_i = 1$ and $h_i = w_i - h_j$.

Let g_n be the output of the CVP circuit. Then, h_n is the output of the converted arithmetic circuit. The construction ensures that h_n evaluates to the same value (zero or one) as g_n.

Clearly, this reduction can be done by an NC algorithm.

(b) Why cannot you use the polylogarithmic-time raking-based tree-contraction algorithm to evaluate an arithmetic circuit? **(3)**

Solution The raking-based tree-contraction algorithm can process an expression that can be represented as a binary tree. In such a tree, the fan-out of each non-root node is one (that is, an expression evaluated at a non-root node is used exactly once). An instance of ARITHMETIC_CIRCUIT may have intermediate values of fan-out larger than one, and if so, this circuit cannot be represented by a binary tree.