CS60027 Parallel Algorithms, Autumn 2023–2024

Class Test 1

13–September–2023 | 06:30pm–07:30pm | Maximum marks: 25

Roll no: ____________ **Name:** ________________________

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

1. In the class, we have seen a combination of an $O(1)$-time non-optimal algorithm (let us call this the *basic maximum algorithm*) and an $O(\log n)$-time optimal algorithm, to design an optimal $O(\log\log n)$-time algorithm for computing the maximum of an array with n distinct elements, on a common CRCW PRAM. The combination used doubly-logarithmic depth trees. This exercise explores the same idea of accelerated cascading, but using another type of trees. Although these new trees cannot yield a superfast running time of $O(\log\log n)$, we can do better than $O(\log n)$ time without sacrificing optimality.

(a) Let $t \geqslant 2$ be a positive integer fixed beforehand (t is not necessarily a constant, that is, t may be a function of n). Instead of a balanced binary tree, consider a balanced t-ary tree. For simplicity, assume that n is a power of t. Then, each internal node in the tree has exactly t child nodes. The n input elements are fed to the n leaf nodes of the tree (one element per node). Each internal node uses the basic maximum algorithm for computing the maximum of the values from its t child nodes. The computation proceeds from the bottom to the top of the tree. The root eventually computes the final result. Deduce that the parallel running time of this algorithm is $O\left(\frac{\log n}{\log t}\right)$, and the work done by this algorithm is $O(nt)$. **(5)**

Solution Let h be the height of the tree. This implies $t^h = n$, that is, $h = \frac{\log n}{\log t}$. Each level takes $O(1)$ parallel time.

For computing the work, let n' be a the number of nodes at some depth. The work done at that depth is $O(t^2 n')$. Summing over all internal nodes gives a total work of $O\left(t^2(n/t) + t^2(n/t^2) + t^2(n/t^3) + \cdots + t^2\right) = O\left(nt\left(1 + \frac{1}{t} + \left(\frac{1}{t}\right)^2 + \cdots\right)\right) = O\left(\frac{nt}{1-\frac{1}{t}}\right) = O(nt)$, because $1 - \frac{1}{t} \geqslant \frac{1}{2}$ for all $t \geqslant 2$.

(b) Find a value of t, for which the parallel running time of the algorithm of Part (a) is $O(\sqrt{\log n})$. What is the work done (as a function of n only) for this value of t? **(5)**

Solution We want $\frac{\log n}{\log t} = \sqrt{\log n}$, that is, $t = 2^{\sqrt{\log n}}$. The work done for this value of t is $O\left(n 2^{\sqrt{\log n}}\right)$.

(c) Use the idea of accelerated cascading that combines the algorithm of Part (a) with an optimal algorithm, for designing an optimal $O(\sqrt{\log n})$-time algorithm to compute the maximum on a common CRCW PRAM. Use the value of t derived in Part (b). Show that your algorithm is optimal, and has a parallel running time of $O(\sqrt{\log n})$. **(5)**

Solution First, use the optimal BBT algorithm until the problem size reduces to $n' = n/2^{\sqrt{\log n}}$. This requires $O(\sqrt{\log n})$ (sequential) iterations, each taking $O(1)$ parallel running time. The work done in this stage is $O(n)$.

Then run the algorithm of Part (a) on the n' max candidates. Take $t = 2^{\sqrt{\log n'}}$. By Part (b), the running time of this stage is $O(\sqrt{\log n'}) = O(\sqrt{\log n})$. Moreover, the work done in this stage is $O\left(n' 2^{\sqrt{\log n'}}\right) = O\left(n' 2^{\sqrt{\log n}}\right) = O(n)$.

Note: If we take $t = 2^{\sqrt{\log n}}$ in the second stage, the same asymptotic bounds follow.

2. Let $A = (a_0, a_1, a_2, \ldots, a_{n-1})$ be an array of n integers (not necessarily distinct). We want to compute the prefix sums $s_i = a_0 + a_1 + a_2 + \cdots + a_i$ for all $i = 0, 1, 2, \ldots, n-1$. For simplicity, assume that $n = 2^d$. Consider a synchronous d-dimensional hypercube H. As usual, the processors in H are numbered by d-bit integers. The i-th processor is supplied the value of a_i, and will eventually store the partial sum s_i (for each $i = 0, 1, 2, \ldots, n-1$). In this exercise, you design an $O(\log n)$-time algorithm for doing the computations on the hypercube H. A skeleton of the algorithm for Processor i is given below. If $i = i_{d-1} i_{d-2} \ldots i_1 i_0$, we denote $i^{(h)} = i_{d-1} \ldots i_{h+1} \overline{i_h} i_{h-1} \ldots i_0$, where $\overline{i_h}$ is the complement of the bit i_h. Notice that the for loop is sequential, so the body of the loop would take $O(1)$ parallel time for each value of h.

```
Initialize my_result and my_message.
For $h = 0, 1, 2, \ldots, d-1$, repeat:
    Exchange my_message with $i^{(h)}$.
    Update my_result and my_message.
```

Fill out the details of the algorithm. Also demonstrate how the algorithm works for $n = 8$ (that is, on a 3-d hypercube.) **(10)**

Solution Each processor maintains two variables s (my_result) and x (my_message). A value received from a neighboring processor is stored in y. The algorithm for the i-th processor is given below.

```
Initialize $s = a_i$ and $x = a_i$.
For $h = 0, 1, 2, \ldots, d-1$, repeat:
```

If $i_h = 0$, do:	If $i_h = 1$, do:
Send x to processor $i^{(h)}$.	Send x to processor $i^{(h)}$.
Receive y from processor $i^{(h)}$.	Receive y from processor $i^{(h)}$.
Set $x = x + y$.	Set $x = x + y$.
	Compute $s = s + x$.

Let us see how this works for $d = 3$.

Initialization

Node	s	x
0	a_0	a_0
1	a_1	a_1
2	a_2	a_2
3	a_3	a_3
4	a_4	a_4
5	a_5	a_5
6	a_6	a_6
7	a_7	a_7

After iteration for $h = 0$

Node	s	x
0	a_0	$a_0 + a_1$
1	$a_0 + a_1$	$a_0 + a_1$
2	a_2	$a_2 + a_3$
3	$a_2 + a_3$	$a_2 + a_3$
4	a_4	$a_4 + a_5$
5	$a_4 + a_5$	$a_4 + a_5$
6	a_6	$a_6 + a_7$
7	$a_6 + a_7$	$a_6 + a_7$

After iteration for $h = 1$

Node	s	x
0	a_0	$a_0 + a_1 + a_2 + a_3$
2	$a_0 + a_1 + a_2$	$a_0 + a_1 + a_2 + a_3$
1	$a_0 + a_1$	$a_0 + a_1 + a_2 + a_3$
3	$a_0 + a_1 + a_2 + a_3$	$a_0 + a_1 + a_2 + a_3$
4	a_4	$a_4 + a_5 + a_6 + a_7$
6	$a_4 + a_5 + a_6$	$a_4 + a_5 + a_6 + a_7$
5	$a_4 + a_5$	$a_4 + a_5 + a_6 + a_7$
7	$a_4 + a_5 + a_6 + a_7$	$a_4 + a_5 + a_6 + a_7$

After iteration for $h = 2$

Node	s	x
0	a_0	$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$
4	$a_0 + a_1 + a_2 + a_3 + a_4$	$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$
1	$a_0 + a_1$	$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$
5	$a_0 + a_1 + a_2 + a_3 + a_4 + a_5$	$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$
2	$a_0 + a_1 + a_2$	$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$
6	$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6$	$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$
3	$a_0 + a_1 + a_2 + a_3$	$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$
7	$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$	$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$