CSE 6220 Homework 2

Karl Hiner, @khiner6

1

Determine if the parallel prefix algorithm can be used to compute prefix sums of a sequence of n numbers based on the binary operation $\bigoplus$ defined as:

The parallel prefix algorithm can be used to compute the prefix sums of a sequence of n numbers if and only if $\oplus$ is a binary associative operator.

(a) $a \oplus b = 2a + b$

Checking associativity,

$$\begin{aligned}(a \oplus b) \oplus c &\stackrel{?}{=} a \oplus (b \oplus c) \\ 2(2a+b)+c &\stackrel{?}{=} 2a + (2b+c) \\ 4a+2b+c &\neq 2a+2b+c\end{aligned}$$

This shows the binary operation defined as $a \oplus b = 2a + b$ *is not* associative, and thus *cannot* be used by the parallel prefix algorithm to compute prefix sums.

(b) $a \oplus b = \sqrt{a^2 + b^2}$

$$\begin{aligned}(a \oplus b) \oplus c &\stackrel{?}{=} a \oplus (b \oplus c) \\ \sqrt{\left(\sqrt{a^2+b^2}\right)^2 + c^2} &\stackrel{?}{=} \sqrt{a^2 + \left(\sqrt{b^2+c^2}\right)^2} \\ \left(\sqrt{a^2+b^2}\right)^2 + c^2 &\stackrel{?}{=} a^2 + \left(\sqrt{b^2+c^2}\right)^2 \\ a^2+b^2+c^2 &= a^2+b^2+c^2\end{aligned}$$

This shows the binary operation defined as $a \oplus b = \sqrt{a^2 + b^2}$ *is* associative, and thus *can* be used by the parallel prefix algorithm to compute prefix sums.

2

In the game of Photosynthesis, points are given for trees that receive sunlight. Consider n trees $T_0, T_1, ..., T_{n-1}$ planted along a single row of spaces, and sunlight is colinear with this row of trees. The tree placement is modeled by an array A of size n, where $A[i]$ denotes the height of the tree T_i. A tree tall enough to get sunlight exposure scores photosynthesis points according to its height, i.e., T_i is given $A[i]$ points. However, a tree can also be blocked from sunlight by an earlier tree *of equal height or taller*, in which case the blocked tree receives no points.

Design a parallel algorithm to compute the total number of points for a configuration given by A, and compute its runtime.

Assume the array A is block-distributed across all nodes. Let p_k refer to the node holding the n/p elements $\left[A[\frac{kn}{p}], ..., A[\frac{(k+1)n}{p} - 1]\right] \equiv A_k$. The algorithm can then be performed by the following steps:

1. Compute the local maximum tree height, $\hat{A}_k = \max(A_k)$ for each node p_k. This takes $O\left(\frac{n}{p}\right)$ time.
2. Use a parallel prefix reduction with $\oplus = \max(\hat{A}_i, \hat{A}_j)$. (Note that the max operation is binary associative, and thus can be used for parallel prefix.) After the reduction, each node p_k holds the maximum tree height, $\hat{A}_{<k}$, of all previous nodes $p_{j<k}$. Parallel prefix takes $O(\log p)$ time, with communication cost $O(p)$.
3. Compute the local total tree score for each node, as follows:
   - Initialize the running score $s_k = 0$, and running height max $m_k = \hat{A}_{<k}$.
   - For each tree height in the local list A_k, if $A_k[i] < m_k$, $s_k \mathrel{+}= A_k[i]$, else set $m_k = A_k[i]$.
4. Use the parallel sum algorithm to sum all local tree scores to find the total tree score, $S = \sum_k s_k$. This final step takes $O(\log p)$ time.

TODO 1 & 2 can actually be combined into one par prefix. TODO add all times

3

A sequence of nested parenthesis is said to be well-formed if 1) there are an equal number of left and right parenthesis, and 2) each right parenthesis is matched by a left parenthesis that occurs to its left in the sequence. For example, ((( ) ( )) ( )) is well-formed, but ( )) (is not.

There is a nested parenthesis sequence of length n distributed across p processors using block decomposition. Design a parallel algorithm to determine if it is well-formed and specify its run-time.

4

Let A be an array of n elements and L be a boolean array of the same size. We want to assign a unique rank in the range $1, 2, ..., n$ to each element of A such that for any $i < j$:

- If $L[i] = L[j]$, $A[i]$ has lower rank than $A[j]$.
- If $L[i] = 0$ and $L[j] = 1$, $A[i]$ has lower rank than $A[j]$.
- If $L[i] = 1$ and $L[j] = 0$, $A[j]$ has lower rank than $A[i]$.

Design a parallel algorithm to compute the ranks and specify its run-time. (Hint: Think of L as specifying labels. Then, all elements with 0 label receive lower ranks than any element with label 1. Within the same label, ranks are given in left to right order as per array A.)

5

Invent Segmented Parallel Prefix: Segmented parallel prefix is a generalization of the parallel prefix problem where the prefix sums need to be restarted at specified positions. Consider array X containing n numbers and a boolean array B of the same size. We wish to compute prefix sums on X but the sum resets at every position i where $B[i] = 1$. Formally, we wish to compute array S of size n such that

$$S[0] = X[0]$$
$$S[i] = \begin{cases} s[i-1] + X[i], & \text{if B[i] = 0} \\ X[i], & \text{if B[i] = 1} \end{cases}$$

Design parallel segmented prefix algorithm and specify its run-time.

(Hint: The problem can be transformed into a standard prefix sums problem.)

NOTE from Feb. 8 class: Can be solved using matrix raised to power of iteration - see class slides.