Question-3: Prefix Sum (16 points)

Solution Approach

1. If there are 'p' processes and 'N' elements, then I will split the main array into N/p chunks, and send each chunk to a process. Padding with 0s if N is not divisible by p.

2. Now, prefix sum of each chunk will be computed by a process in O(N/p), but of course, they won't be reflective of the correct (GLOBAL) prefix sum as a whole.

3. To fix this, I will make each of them send the last element of their local prefix sum back to the root process, and ask the root process to do a O(p) run over this, and prefix sum this up.

4. Then, I will send each one of these elements to the respective local processes, and ask them to add this number to all their prefix sums in O(N/p).

5. Lastly, ask each process to send their prefix sum arrays back to the root process, and it will return the final array after gathering them.

Highlights of Program

If N is not divisible by p, then the first few processes take up 1 extra element until all the remaining elements are used up. Thus, every process computes prefix sum of at most 1 element more than others. This is a nearly equal distribution of data amongst different processors.

I have also coded up a generic `inplace_prefix_sum` function that computes the prefix sum of an array inplace, from a `start_ptr` to an `end_ptr`.

This program uses `MPI_Scatter`, which is inherently slow. This seems to be the bottleneck in the program. An alternative way would be to use `MPI_Bcast`, but that would force every single process to store entire input array, which is simply unnecessary. However, ignoring the fact that *someone* has to split the array; once the array has been split, the program is incredibly fast. ‘

Total Time Complexity of Approach

Ignoring the time complexity of the MPI operations; the time complexity of my approach is $\mathbf{O(N/p + p)}$, where the N/p term is due to each processor computing N/p chunk of the prefix sum, and p for root process to compute the offsets between chunks, to get the final prefix sum.

Total Message Complexity of Approach

For broadcasting number of elements, it is O(log p) using `MPI_Bcast`.

For scattering initial data, it is O(N) using `MPI_Scatter`.

For gathering last elements, it is O(p) using `MPI_Gather`.

For scattering adjusted values, it is O(p) using `MPI_Scatter`.

For gathering final results, it is O(N) using `MPI_Gather`.

All things considered, it comes out to be $\mathbf{O(N + p + \log p)}$.

Space Requirements of Solution

Root process requires O(N + p) space complexity, since it needs to store a main array of size N (to get data initially, and to gather final prefix sums), and a size p array to get the offsets between chunks.

Every other process requires only O(N/p), since all they need to store is an array of size N/p for their own chunk.

Performance Scaling from 1 to 12 processes

Size of testcase: N = *1000000*

Number of Processors	Time Elapsed
1	0.0109392s
2	0.0103918s
3	0.00720181s
4	0.00541813s
5	0.00440225s
6	0.00396056s
7	0.00339356s
8	0.00293015s
9	0.00264365s
10	0.00247898s
11	0.00233081s
12	0.00223038s