Question-5: Parallel Matrix Chain Multiplication Problem (28 points)

Solution Approach

1. Parallelizing the standard DP approach to solve the matrix Chain Multiplication problem.
2. So, the trick is to split it up into diagonal parallelization. This is because, each diagonal depends only on the values of its previous diagonals, and initially the principal diagonal is just the dimensions of each matrix.
3. Basically, start from the principal diagonal (n elements) and split the work up amongst processors.
4. This essentially means that I will be operating on matrix chain lengths from l = 2 to l = n, as this is what defines the diagonal that I'm currently on.
5. On each diagonal, I will split it up into nearly equal chunks. Similar to my approach for Question-3, at any given time, each process will have to make 1 extra calculation as opposed to others.
6. After each diagonal is done, I will iterate over each element in that diagonal, and get the process that computed it to broadcast to all other processes so that they can update their own DP matrix, for subsequent diagonal calculations.
7. At the end, I will get the root process to print out the value of DP[1][n], where the optimal number of multiplications are stored.

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 optimal chain multiplication of at most 1 element more than others in any given diagonal computation iteration. This is a nearly equal distribution of data amongst different processors.

Due to absence of `MPI_Scatter` and its variants, this program is very fast.

Total Time Complexity of Approach

O(N^3/p), since each processor computes N/p * $N * 2$ chunks.

Total Message Complexity of Approach

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

For broadcasting initial data, it is O(log p) using `MPI_Bcast`, and (n+1) * sizeof(int) data is received by each process.

For broadcasting computed data in each diagonal iteration, it is O(N^2/p * log p) using `MPI_Bcast`, as each process needs to make N/p broadcasts (which is the number of elements it processes), and there are N such diagonals.

All things considered, it comes out to be **O(log p + N^2/p log p))**.

Space Requirements of Solution

O(N^2), since each processor requires its own DP array, and chain dims array (O(N)).

Performance Scaling from 1 to 12 processes

Size of testcase: *N = 3000*

Number of Processors	Time Elapsed
1	48.6291s
2	45.8646s
3	44.8987s
4	41.346s
5	37.8041s
6	33.8467s
7	31.4334s
8	29.5713s
9	28.7728s
10	28.1578s
11	27.0328s
12	25.4644s