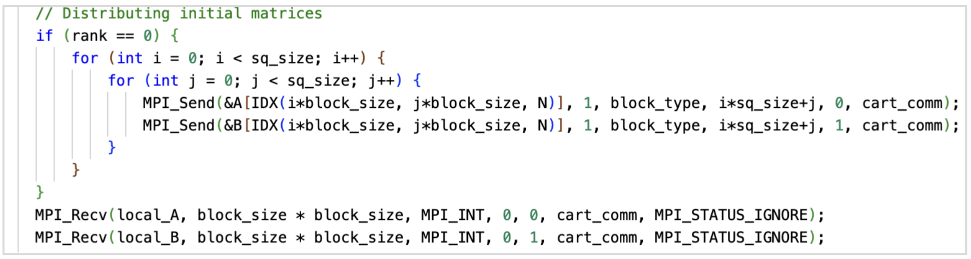
Parallel Programming MPI Part 2 - **Cannon Algorithm Task B**

Pavithra Purushothaman (fd0001571)
Matrikel Nummer: 1535949

In my implementation of **Cannon Algorithm's Task B**, I modified the code to work with **submatrices** instead of individual elements. The primary goal was to **optimize parallel computation** and reduce communication overhead. Instead of each process handling a single matrix value, I adapted the code so that each process operates on a **block of the matrix**.

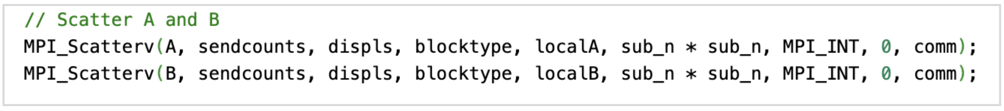
The most significant change was replacing **element-wise distribution** with **submatrix distribution.** In Task A, I used MPI_Send() and MPI_Recv() to send individual elements. In Task B, I replaced this with **MPI_Scatterv()**, allowing processes to receive larger **blocks of the matrix** at once.

Task A: Too many messages being sent and received due to element-wise distribution.



```
// Distributing initial matrices
if (rank == 0) {
    for (int i = 0; i < sq_size; i++) {
        for (int j = 0; j < sq_size; j++) {
            MPI_Send(&A[IDX(i*block_size, j*block_size, N)], 1, block_type, i*sq_size+j, 0, cart_comm);
            MPI_Send(&B[IDX(i*block_size, j*block_size, N)], 1, block_type, i*sq_size+j, 1, cart_comm);
        }
    }
}
MPI_Recv(local_A, block_size * block_size, MPI_INT, 0, 0, cart_comm, MPI_STATUS_IGNORE);
MPI_Recv(local_B, block_size * block_size, MPI_INT, 0, 1, cart_comm, MPI_STATUS_IGNORE);
```

Task B: This distributes large matrix blocks instead of sending one value at a time, reducing the number of communications.



```
// Scatter A and B
MPI_Scatterv(A, sendcounts, displs, blocktype, localA, sub_n * sub_n, MPI_INT, 0, comm);
MPI_Scatterv(B, sendcounts, displs, blocktype, localB, sub_n * sub_n, MPI_INT, 0, comm);
```

This approach was chosen because,

- **Better Load Balancing:** Every process now computes **multiple values** instead of just one.
- **Lower Communication Overhead:** Instead of sending thousands of messages, **fewer large messages** are sent.
- **Scalability:** This approach scales well for **larger matrices** (N = 8000), compared to Task A.

Instead of replacing all MPI functions from task A, I adapted existing ones to support **submatrices**:

- **Replaced MPI_Send()** with **MPI_Scatterv()** for efficient matrix distribution.
- **Replaced MPI_Recv()** with **MPI_Gather()** to collect computed results efficiently.
- **Used MPI_Cart_create()** to define a **2D Cartesian grid**, ensuring ranks are properly assigned in a logical manner.
- **Used MPI_Cart_shift()** to implement the **shifting** required in Cannon's Algorithm.

The switch from element-wise distribution to **submatrices** significantly improved **parallel efficiency**. The new approach ensures that all processes work **independently**, reducing waiting time and avoiding **Rank 0 bottlenecks**.

References:
1. Used lecture slides for understanding different communication mechanisms for sending block data.