Parallel Prefix Scan Algorithm Implementation in C/mpi Report

Introduction

The program implements the parallel prefix scan algorithm using MPI (Message Passing Interface). The prefix scan is a process where each element in a sequence is replaced by the sum of all previous elements, including itself. This program distributes the task across multiple processes using MPI without the use of the **MPI_Scan** function.

Key Steps in the Code

Initialization and Input (Step 1 & 2)

- MPI Initialization: The MPI environment is initialized using **MPI_Init**, and each process retrieves its rank (ID) and size (number of processes).
- Data Input (Process 0): Process 0 takes input for n, the number of elements to be processed. It generates n random integers and stores them in the data array.
- Broadcasting n: The value of n is broadcast to all processes using **MPI_Bcast**, ensuring all processes know the size of the data.

```c
MPI_Comm_size(MPI_COMM_WORLD, &size);

// Step 1: Process 0 generates the random data
if (rank == 0) {
    // Get the number of elements to generate
    printf("Enter the number of elements to generate: ");
    scanf("%d", &n);

    // Ensure n is divisible by the number of processes
    if (n % size != 0) {
        printf("Number of elements must be divisible by the number of processes.\n");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    // Allocate memory for n integers
    data = (int*)malloc(n * sizeof(int));

    // Seed the random number generator and generate n random numbers
    srand(time(0));
    for (int i = 0; i < n; i++) {
        data[i] = rand() % 100;  // Random integers between 0 and 99
    }

    // Print the generated array
    print_array(data, n, "Array generated");
}
```

Data Distribution (Step 3 & 4):

The total data is divided evenly among the processes, ensuring each process handles n / size elements.

MPI_Scatter distributes portions of the data array to each process.

```
// Allocate memory for the local array in each process
local_data = (int*)malloc(local_n * sizeof(int));

// Step 4: Scatter the data to all processes
MPI_Scatter(data, local_n, MPI_INT, local_data, local_n, MPI_INT, 0, MPI_COMM_WORLD);

// Step 5: Each process computes the prefix sum of its local portion (up phase)
for (int i = 1; i < local_n; i++) {
    local_data[i] += local_data[i - 1];  // Add previous element to current
}
```

Local Prefix Sum Computation (Up Phase) (Step 5)

Each process computes the local prefix sum for its portion of the data. This is done by iterating through the local array and adding the current element to the previous one.

```
// Step 5: Each process computes the prefix sum of its local portion (up phase)
for (int i = 1; i < local_n; i++) {
    local_data[i] += local_data[i - 1];  // Add previous element to current
}
```

Gathering Local Results (Step 6)

Once each process computes its local prefix sum, the partial results are gathered by process 0 using **MPI_Gather.**

```
// Step 6: Gather the local prefix sums back to process 0
MPI_Gather(local_data, local_n, MPI_INT, data, local_n, MPI_INT, 0, MPI_COMM_WORLD);
```

Global Prefix Sum (Down Phase) (Step 7):

- In the **down phase**, process 0 combines the local prefix sums to create the global prefix sum. This is done by adjusting the prefix sums of each subsequent process by adding the last value of the previous process.

```
// Combine the prefix sums from all processes (down phase)
for (int i = 1; i < size; i++) {
    for (int j = 0; j < local_n; j++) {
        data[i * local_n + j] += data[(i - 1) * local_n + (local_n - 1)];
    }

    // Print the down phase result at each step
    printf("Down phase after process %d: ", i);
    print_array(data, n, "Down phase");
}
```

Final Output:

- Process 0 prints both the original input and the final global prefix sum to the screen.

```
// Print the final global prefix sum
printf("\nFinal global prefix sum:\n");
print_array(data, n, "Global prefix sum");
```

Output Example:

- **Input**: [23, 45, 12, 9, 18, 55]
- **Global Prefix Sum**: [23, 68, 80, 89, 107, 162]

Testing and Results

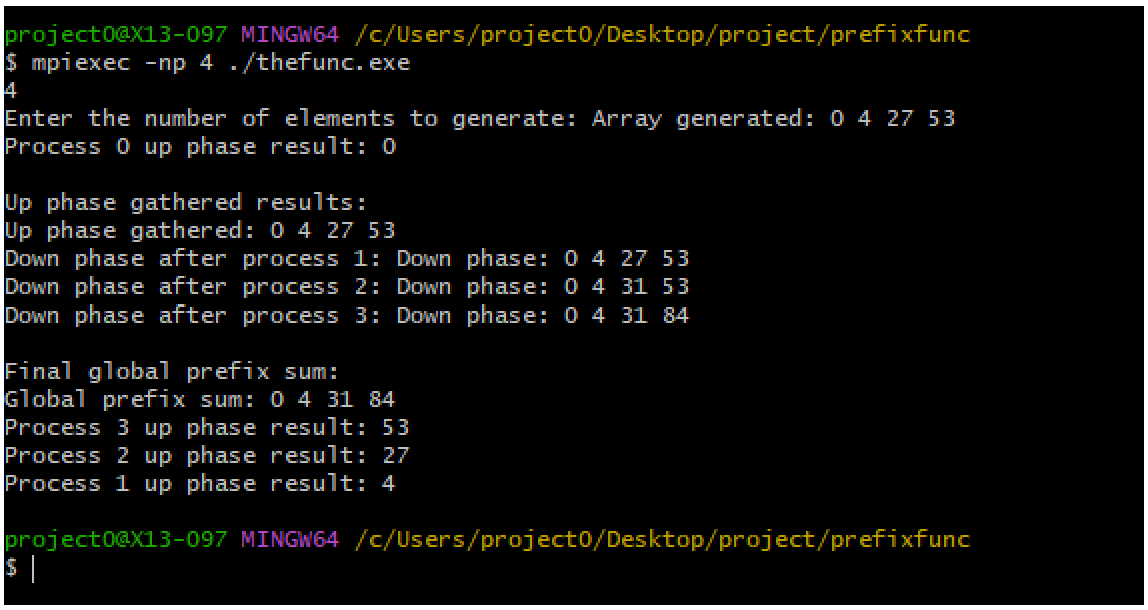
Test 1

I ran the program using 4 processes and the figure below shows the results. The user or keyboard input of the number of elements in the arrays was 4.

The generated arrays: [0, 4, 27, 53]

The Partial sums (up phase gathered results: [0, 4, 27, 53]

The final result (Global prefix sum): [0, 4, 31, 84]

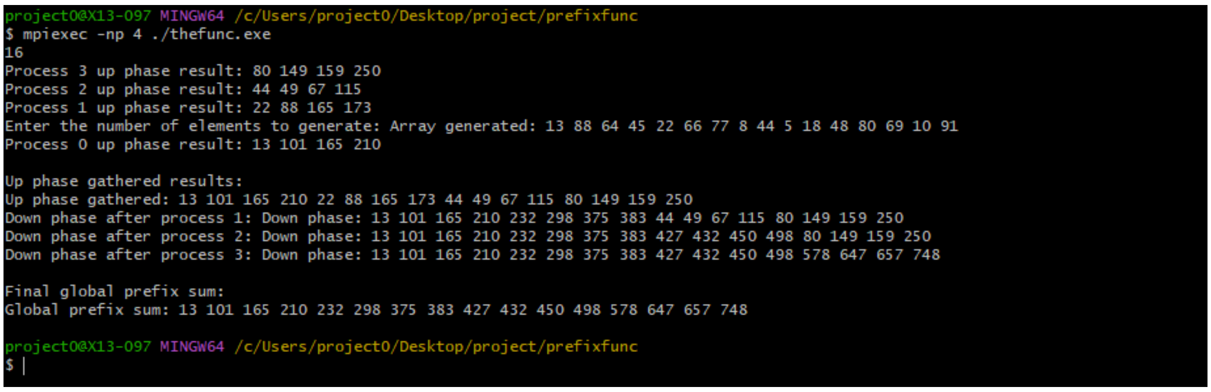


```
project0@X13-097 MINGW64 /c/Users/project0/Desktop/project/prefixfunc
$ mpiexec -np 4 ./thefunc.exe
4
Enter the number of elements to generate: Array generated: 0 4 27 53
Process 0 up phase result: 0

Up phase gathered results:
Up phase gathered: 0 4 27 53
Down phase after process 1: Down phase: 0 4 27 53
Down phase after process 2: Down phase: 0 4 31 53
Down phase after process 3: Down phase: 0 4 31 84

Final global prefix sum:
Global prefix sum: 0 4 31 84
Process 3 up phase result: 53
Process 2 up phase result: 27
Process 1 up phase result: 4

project0@X13-097 MINGW64 /c/Users/project0/Desktop/project/prefixfunc
$ |
```

Test 2

I ran the program using 4 processes and the figure below shows the results. The user or keyboard input of the number of elements in the arrays was 16.

The diagram below shows the results.



```
project0@X13-097 MINGW64 /c/Users/project0/Desktop/project/prefixfunc
$ mpiexec -np 4 ./thefunc.exe
16
Process 3 up phase result: 80 149 159 250
Process 2 up phase result: 44 49 67 115
Process 1 up phase result: 22 88 165 173
Enter the number of elements to generate: Array generated: 13 88 64 45 22 66 77 8 44 5 18 48 80 69 10 91
Process 0 up phase result: 13 101 165 210

Up phase gathered results:
Up phase gathered: 13 101 165 210 22 88 165 173 44 49 67 115 80 149 159 250
Down phase after process 1: Down phase: 13 101 165 210 232 298 375 383 44 49 67 115 80 149 159 250
Down phase after process 2: Down phase: 13 101 165 210 232 298 375 383 427 432 450 498 80 149 159 250
Down phase after process 3: Down phase: 13 101 165 210 232 298 375 383 427 432 450 498 578 647 657 748

Final global prefix sum:
Global prefix sum: 13 101 165 210 232 298 375 383 427 432 450 498 578 647 657 748

project0@X13-097 MINGW64 /c/Users/project0/Desktop/project/prefixfunc
$ |
```

<u>Conclusion</u>

This program successfully implements the parallel prefix scan algorithm by distributing data among processes, computing local prefix sums, and combining results into a global prefix sum without using MPI_Scan. The use of **MPI_Scatter** and **MPI_Gather** efficiently manages data distribution and collection across processes.