Assignment – 1

Group-13 Report

Ayyappa Reddy Maramreddy – R11888944

Jason Heinrich – R11913725

Rishabh Kumar – R11785189

Description:

The provided code is an MPI Program written in C that which demonstrates the parallel computation using two processes. The main goal of this code is to design and distribute the computation of row sums of a 100x100 matrix across two MPI Processes of having (With Process IDs 0 and 1). The code uses non-blocking communication primitives like (‘MPI_Send’ and ‘MPI_Irecv’) to overlap communication and computation.

In our method of implementation, we introduced asynchronous communication for data transfer from Process 0 to Process1 instead of sending all the data at once, each row of data was sent individually using MPI_Isend () within a loop.

On the receiving end (Process 1), we adapted the code to create MPI_Irecv () handlers for each row upfront, forming a queue of asynchronous receives. A subsequent loop was employed to wait for each message and immediately perform row summation, eliminating the need for a barrier or wait call after the loop. As long as the time taken to sum the rows was significantly less than the time to send data, this approach ensured efficient processing.

Upon Completing the row summation in Process 1, the data was sent back to the process 0 in a single block using MPI_Send (). Meanwhile, Process 0, having waited for data to be sent to Process 1, used MPI_Recv () to receive the processed data back. Finally, Process 0 printed the resulting row sums in 10 columns. This modification enhanced concurrency by allowing processes to overlap communication and computation, potentially improving the overall performance.

Then Finally the Program finalizes MPI using MPI_Finalize ().

Overall, Our MPI program code demonstrates the parallelization of a simple matrix computation task across two processes. Showcasing the use of MPI Communication functions and non-blocking operations to achieve overlapping of communication and computation.

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

// Macro to generate data
#define generate_data(i,j) (i)*(i)*(j)
void main(int argc, char **argv)
{
    int i, j, pid, np, mtag, data[100][100], row_sum[100];
    MPI_Status status;
    MPI_Request req_s, req_r[50];  // Array of requests for multiple non-blocking receives

    // Initialize MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &pid); // Get process rank
    MPI_Comm_size(MPI_COMM_WORLD, &np);  // Get total number of processes

    // Process with ID 0
    if(pid == 0) {
        // Generate first half of the data
        for(i=0; i<50; i++)
            for(j=0; j<100; j++)
                data[i][j] = generate_data(i,j);

        // Sending first half to process 1

        mtag = 1;
        for(i=0; i<50; i++)
            MPI_Isend(data[i], 100, MPI_INT, 1, mtag, MPI_COMM_WORLD, &req_s);
```

Fig 1. Initializing MPI and Implementation of Process 0

```c
        // Generate the second half of the data using the given generate function
        for(i=50; i<100; i++)
            for(j=0; j<100; j++)
                data[i][j] = generate_data(i,j);

        // Compute row sums for the second half of the data
        for(i=50; i<100; i++){
            row_sum[i] = 0;
            for(j=0; j<100; j++)
                row_sum[i] += data[i][j];
        }

        MPI_Wait(&req_s, &status); // Wait for the data send to complete

        // Non-blocking receive of computed row_sums from process 1
        mtag = 2;
        MPI_Recv(row_sum, 50, MPI_INT, 1, mtag, MPI_COMM_WORLD, &status);

        //MPI_Wait(&req_r[i], &status);  // Wait for the receive to complete

        // Print all row sums
        for(i=0; i<100; i++) {
            printf(" %d ", row_sum[i]);
            if(i%10 == 9) printf("\n");
        }
    }
    // Process with ID 1
    else { /*** pid == 1 ***/
        // Receive the first half of the data from process 0 and compute row sums
```

Fig 2. Generating second half of the data

```
63          mtag = 1;
64          for(i=0; i<50; i++) {
65              MPI_Irecv(data[i], 100, MPI_INT, 0, mtag, MPI_COMM_WORLD, &req_r[i]);
66          }
67
68          for(i=0; i<50; i++) {
69              MPI_Wait(&req_r[i], &status);  // Wait for the receive to complete
70
71              row_sum[i] = 0;
72              for(j=0; j<100; j++)
73                  {
74                  row_sum[i] += data[i][j];
75                  }
76
77              //printf("row: ");
78              //printf(" %d ", i);
79              //printf(" %d ", row_sum[i]);
80
81          }
82
83
84          // Send the computed row sums to process 0
85          mtag = 2;
86          MPI_Send(row_sum, 50, MPI_INT, 0, mtag, MPI_COMM_WORLD);
87          //MPI_Wait(&req_s, &status);  // Wait for the send to complete
88      }
89
90      MPI_Finalize(); // Finalize MPI
91  }
```

Fig 3. implementation of Process 1 and MPI_Finalize ()

Output:

```
PS C:\Users\harle\OneDrive\Desktop\Parallel> mpiexec -n 2 Assign1
 328350  328450  328550  328650  328750  328850  328950  329050  329150  329250
 329350  329450  329550  329650  329750  329850  329950  330050  330150  330250
 330350  330450  330550  330650  330750  330850  330950  331050  331150  331250
 331350  331450  331550  331650  331750  331850  331950  332050  332150  332250
 332350  332450  332550  332650  332750  332850  332950  333050  333150  333250
 333350  333450  333550  333650  333750  333850  333950  334050  334150  334250
 334350  334450  334550  334650  334750  334850  334950  335050  335150  335250
 335350  335450  335550  335650  335750  335850  335950  336050  336150  336250
 336350  336450  336550  336650  336750  336850  336950  337050  337150  337250
 337350  337450  337550  337650  337750  337850  337950  338050  338150  338250
```

Fig 4. Result