MPI Program: Modifications

Introduction

The problem under consideration is the Mandelbrot set, which is defined as –

$$z_{i+1} = z_i^2 + C$$

This equation is of the form $\alpha + i\beta$, which contains a real axis, α and an imaginary axis, β, forming a complex number.

In the code provided as a part of the assessment brief, the parameters defined are as follows –

The initial code uses `MPI_Reduce` to collate the results, which sends more data than required, because each participating process in the call sends an entire computational domain. This could potentially be improved to have a more effective runtime and a significant reduction of computational cost. Therefore, in the modified code, each worker process sends only the values calculated, rather than sending an entire computational domain. Consequently, there is no need to call `MPI_Reduce` to collate the results at the end of calculation, as there is no value that is calculated unnecessarily, thus, optimising the computational cost.

The results for the original and the modified code were compared using 2, 4, 8, 12 and 16 MPI processes on one node, and 32 MPI processes on two nodes.

Code modifications

The code was modified as follows:

- A buffer space was created for sending and receiving data, such that the buffer space of the size of one column in the complex plane with two additional values. These two values are essentially worker rank and iteration `i`.
- The point-to-point communication was modified, such that the worker processes send their results to the manager process after calculating each column. In an event where missing data is returned, these were changed to -1.
- The function `do_communication` is modified to `modified_function`, within which all of these modifications are made.

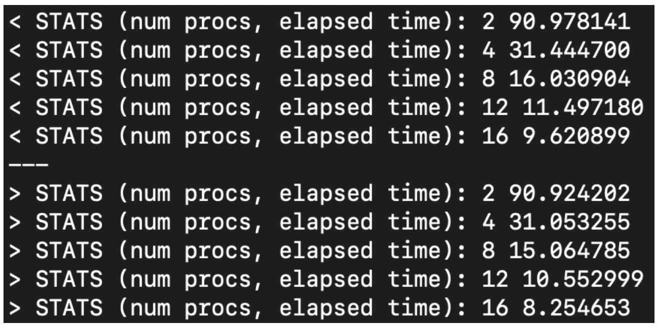
The code snippet is described below –

```
void modified_function(int myRank){
  MPI_Group worldGroup, workerGroup;
  MPI_Comm  workerComm;
  int zeroArray={0};

  int sendBuffer[N_IM + 3];
  int receiveBuffer[N_IM + 3];
  int index=0;
  int i, j;

  MPI_Comm_group(MPI_COMM_WORLD, &worldGroup);
  MPI_Group_excl(worldGroup, 1, &zeroArray, &workerGroup);
  MPI_Comm_create(MPI_COMM_WORLD, workerGroup, &workerComm);
  /* Packing i (iteration) into a 1D buffer for sending and receiving. This
includes the worker rank, the iteration i,
  and the nIter values for the rest of the column*/
  for (i = 0; i < N_RE; i++) {
    if (myRank != 0) {
        sendBuffer[0] = myRank;
        sendBuffer[1] = i;
        for (j = 0; j < N_IM + 1; j++) {
            sendBuffer[j + 2] = nIter[i][j];
        }
        MPI_Send(sendBuffer, N_IM + 3, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    else {
        for (int workerRank = 1; workerRank < N_RE + 1; workerRank++) {
            MPI_Recv(receiveBuffer, N_IM + 3, MPI_INT, workerRank, 0,
MPI_COMM_WORLD)
            int worker_i = receiveBuffer[1];
            for (j = 0; j < N_IM; j++) {
                nIter[worker_i][j] = receiveBuffer[j + 2];
            }
        }
    }
    }
```

Results



```
< STATS (num procs, elapsed time): 2 90.978141
< STATS (num procs, elapsed time): 4 31.444700
< STATS (num procs, elapsed time): 8 16.030904
< STATS (num procs, elapsed time): 12 11.497180
< STATS (num procs, elapsed time): 16 9.620899
---
> STATS (num procs, elapsed time): 2 90.924202
> STATS (num procs, elapsed time): 4 31.053255
> STATS (num procs, elapsed time): 8 15.064785
> STATS (num procs, elapsed time): 12 10.552999
> STATS (num procs, elapsed time): 16 8.254653
```

Fig. 1. The difference in results between initial (on the above) and modified code (in the bottom) for one node processes

As observed from results, the time taken by the modified code is lesser than that of the initial code. These results are plotted as a graph to demonstrate the comparison in Fig. 2.

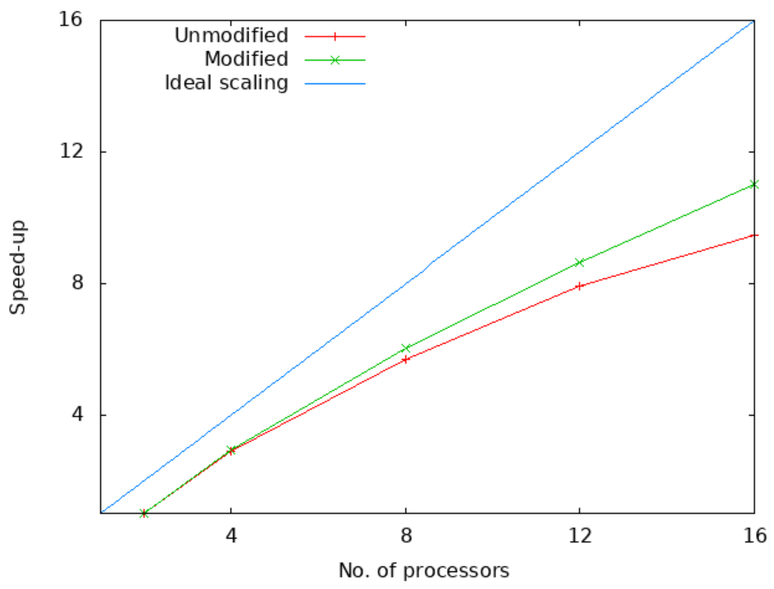


Fig. 2. Graphical representation of comparison of unmodified and modified code for 2, 4, 8, 12 and 16 MPI processes on one node

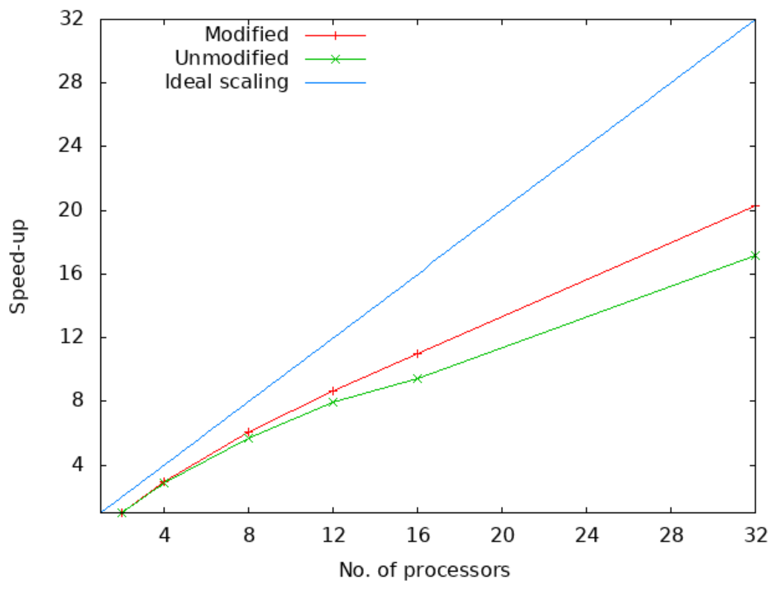


Fig. 3. Graphical representation of comparison of unmodified and modified code for 2, 4, 8, 12 and 16 MPI process on one node, and 32 MPI processes on two nodes

Table 1. Computational time on each of the nodes, as described in the previous sections, tabulated, along with the ratio of T_0 and T_i

MPI Process	Unmodified		Modified	
	Time elapsed	T_0/T_i	Time elapsed	T_0/T_i
2	90.978	1.00	90.924	1.00
4	31.444	2.89	31.053	2.93
8	16.030	5.67	15.064	6.04
12	11.497	7.91	10.553	6.62
16	9.621	9.46	8.257	11.02
32 (two nodes)	5.303	17.15	4.485	20.28

Where, $T_0 = 90.985\,s$ is the time taken by the code when no parallel computing were implemented, and T_i is the time taken by i nodes.

Conclusion

The ratio T_0/T_i is essentially how well the program performs as compared to processes run in series, as opposed to parallel. It could be observed that the unmodified program performs 17.15 times better on 32 MPI processes with two nodes, whereas the modified program performs 20.28 times better on the same process run.

It is apparent that the modified program performs 18% better than the unmodified program on the two-node iteration, whereas, it performs 16.5% better than the unmodified program on the 16 MPI process, one-node iteration.