Workshop 3

Peter Whalley

February 1, 2021

1 Exercise 2

This a basic example of practice of basic MPI functions and collection communication. We perform the task of calculating matrices **A** and **B** which are $N \times N$ matrices defined by

$$A_{i,j} = (N - j + i + 1)i$$
$$B_{i,j} = (j + i)(N - j + 1),$$

where N is the number of processors. Our script outputs the value of $A \times B$ and does this all in parallel as follows.

Listing 1: C++ code using listings

```
#include <iostream> //includes packages
#include <mpi.h>
using namespace std;
int main()
{
  int rank, ierr, size;
  MPI_Comm comm;
  comm = MPI_COMM_WORLD;
  MPI_Init(NULL,NULL);
  MPI_Comm_rank(comm, &rank);
  MPI_Comm_size(comm, &size);
  int i, j, k, l, m, n, o;
  const int root = 0;
  int N = size;
  int A[N];
  int B[N][N];
  int D[N][N];
  int C[N];
  if (rank == root) {
    for (i=0; i < N; i++)
    {
      for (j=0; j < N; j++)
      {
      B[i][j] = ((j+1) + (i+1))*(N-(j+1)+1);
      }
    }
  }
  //Broadcast B
  MPI_Bcast(&B, N*N, MPI_INT,root ,comm);

  //Barrier
  MPI_Barrier(comm);

  for (j=0; j < N; j++)
    {
    A[j] = (N-(j+1)+(rank+1)+1)*(rank+1);

  }
  //multiplying each processors row of A by the entire matrix B
  //and storing it as C.
  for (l = 0; l< N; l++)

    { C[l] = 0;
      for  (m = 0; m<N ; m++)
      {
        C[l] += A[m]*B[m][l];
      }
    }
  //Gathering all the rows of D.
  MPI_Gather(&C,N,MPI_INT,&D,N,MPI_INT,root , comm);
  //Barrier
  MPI_Barrier(comm);
  //Reconstruct Matrix.
  //Printing Matrix
  if (rank == root){
  for(n = 0;n<N;n++)
   {
```

```
58            for(o=0;o<N;o++)
59            {
60                cout<<D[n][o]<< " ";
61            }
62        cout<<endl;
63        }
64    }
65      MPI_Finalize();
66  }
```

It does this by initialising MPI and assigning ranks and the size (number of processors) using functions MPI_Init, MPI_Comm_rank and MPI_Comm_size in lines 7 to 11. It then assigns the integer N to be size (the number of processors). Further to this we generate the matrix B on the root process (the process with rank 0), then uses MPI_Bcast to broadcast B to all other processes and this is done in lines 20 to 29. We have then generated each row of matrix A in parallel, i.e. the rank 0th processor creates the first row, the rank 1st processor creates the second row etc. Note that we have only allocated A an array of size N rather than a matrix of size $(N \times N)$ to save memory as we only need to store the respective rows of A on each process for matrix multiplication. Then each process multiples their row of A by the entire matrix B and stores their answer in a vector C, this is done in lines 41 to 48. After this is done the root process gathers all of the vectors C and puts them in the right order in matrix D. This is done using the function MPI_Gather. We have also included MPI_Barrier functions in lines 32 and 52 to ensure that all the processes are at the same point after broadcasting and gathering. Also in lines 54 to 64 we have just included a nice format for printing the matrix in the terminal.

2 Exercise 3

Here is an example of using MPI parallelisation for a double integral solver. We evaluate the integral

$$\int_0^b \int_0^a x\sin(x^2) + y\sin(y^2)dxdy = \frac{1}{2}(-b\cos(a^2) - a\cos(b^2) + a + b).$$

I have used a 2D version of the trapezium rule and I have implemented this for uniform strips and squares, by dividing up the domain into rectangles of width $1/N_x$ and length $1/N_y$.
The trapezium rule is stated as follows

$$\int_a^b \int_c^d f(x,y)dydx = \frac{1}{4N_xN_y}[f(a,c) + f(b,c) + f(a,d) + f(b,d) + 2\sum_{i=1}^{N_x} f(x_i,c) + 2\sum_{i=1}^{N_x} f(x_i,d)$$
$$+2\sum_{j=1}^{N_y} f(a,y_j) + 2\sum_{j=1}^{N_y} f(b,y_j) + 4\sum_{j=1}^{N_y}\sum_{i=1}^{N_x} f(x_i,y_j)]$$

Then I have parallelised it by performing the calculation

$$\sum_{i=0}^{\lfloor (N_x-(r+1))/N \rfloor} f(x_{(r+1)+i*N}, c) + \sum_{i=1}^{\lfloor (N_x-(r+1))/N \rfloor} f(x_{(r+1)+i*N}, d)$$
$$+ \sum_{j=1}^{\lfloor (N_y-(r+1))/N \rfloor} f(a, y_{(r+1)+j*N}) + \sum_{j=1}^{\lfloor (N_y-(r+1))/N \rfloor} f(b, y_{(r+1)+j*N}) + \sum_{j=1}^{\lfloor (N_x-(r+1))/N \rfloor} \sum_{i=1}^{N_x} f(x_i, y_{(r+1)+j*N})]$$

on process r out of N and hence splits up the tasks between all N processors. I have then used the function MPI_Reduce with the operator SUM to sum up all these values to have an approximation of the total integral. I have not had time to perform further analysis of this algorithm in combination with task 1.

Listing 2: C++ code using listings

```
#include <iostream> //includes packages
#include <math.h>
#include <string>
#include <mpi.h>

using namespace std; //using standard function names.

//Predefining the function to integrate.
double f(double x,double y)
{ double a = x*sin(x*x) + y*sin(y*y);
  return a;
}

int main(void)
{
  float a;
  float b;
  int N_x;
  int N_y;
  float int_approx;
  float int_exact;
  float numerical_error;




```

```
    a = 100;
    b = 100;
    N_x = 100000;
    N_y = 100000;

    double h_x;
    double h_y;
    int i,j;
    double sum;
    double finalsum;
    int rank, ierr, size;
    MPI_Comm comm;
    const int root = 0;
    comm = MPI_COMM_WORLD;
    MPI_Init(NULL,NULL);
    MPI_Comm_rank(comm, &rank);
    MPI_Comm_size(comm, &size);


    h_x = a/ N_x;
    h_y = b/ N_y;

    sum = 0.0;

    for (i = rank+1; i<N_x; i+=size)
    {
      sum += 2*(f(i*h_x,0) + f(i*h_x,b));
    }
    for (i = rank+1; i<N_y; i+=size){

      sum += 2*(f(0,i*h_y) + f(a,i*h_y));
    }
    for (i = rank+1; i<N_x; i+=size)
    { for (j=1;j<N_y;j++){
      sum += 4*f(i*h_x,j*h_y);
    }
    }
    //Reducing to a final sum.
    MPI_Reduce(&sum,&finalsum,1,MPI_DOUBLE,MPI_SUM,root,comm);
```

```
  if (rank == root){




    int_approx = (finalsum + f(0,0) + f(a,0) + f(a,b) + f(0,b)) * h_x * h_y /
    cout <<"The approximate integral = " <<int_approx << endl;

    int_exact = 0.5*(-b*cos(a*a) - a*cos(b*b) + a + b);
    numerical_error = abs(int_exact-int_approx);
    cout <<"The numerical error = " << numerical_error << endl;

  }
  MPI_Finalize();
}
```