HPC for Mathematicians - Week 3 Assignment

Aikaterini Karoni

February 2, 2021

Exercise 2

The following code calculates the product of two NxN matrices A and B, where:

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

where i and j range form 1 to N and are the row and column of each element respectively.

The code used to perform the parallel multiplication is the following:

Listing 1: C++ code using listings

```

#include <iostream>
#include <stdlib.h>
#include <mpi.h>
#include <time.h>

using namespace std;


int main(){

  int rank, size, ierr;
  MPI_Comm comm;

  comm   = MPI_COMM_WORLD;

  MPI_Init(NULL,NULL);
  MPI_Comm_rank(comm, &rank);
  MPI_Comm_size(comm, &size);

  // for non-blocking send-receive later on
  MPI_Request request;
  MPI_Status status;

  int N = size;
  int A[N][N];
  int B[N][N];
  int C[N];
  int D[N][N];

// root process generates matrix B
  if (rank == 0) {
     for (int i = 0; i < N; i++) {
         int ii = i + 1;
         for (int j = 0; j < N; j++) {
             int jj = j + 1;
            B[i][j] = (jj+ii) * (N-jj+1);
          }
      }
  }

// Broadcast matrix data to the other processes
  MPI_Bcast(&B, N*N, MPI_INT, 0 , comm);

// Each process creates their corresponding row of matrix A
  for (int j = 0; j < N; j++) {
      int jj = j + 1;
      int rr = rank + 1;
      A[rank][j] = (N-jj+rr+1) * rr;
  }

// Each process multiplies their row of A with the entire row of B
// and  stores the answer in vector c.
  for (int j = 0; j < N; j++) {
      C[j] = 0;
      for (int k = 0; k < N; k++) {
          C[j] += A[rank][k] * B[k][j];
      }
  }
 // Root process gathers all vectors C from each process and puts them
 // into matrix D
  MPI_Gather(&C, N, MPI_INT, &D, N, MPI_INT, 0, comm);

  if (rank == 0) {
     for (int i = 0; i < N; i++) {
         for (int j = 0; j < N; j++) {
             cout << "(i,j)= "<< "(" <<i << ","<< j << ")" << " and  " << "D[i][j]= "<< D[i][j] << endl;
         }
     }
  }


  MPI_Finalize();

}
```

As we can see, the root process (rank 0) generates the matrix B and broadcasts it to all other processes using the command $MPI\_Bcast(\&B, N*N, MPI\_INT, 0, comm)$. Each process then creates their corresponding row of matrix A and multiplies the row with all of B to obtain a vector C. Finally, using the *MPI_Gather* command, the vectors C from each process are gathered by the root process into one matrix D, whose elements are printed in an output file.

1 Exercise 3

In this exercise we use the trapezoid rule to copmute the following integral:

$$\int_0^b \int_0^a x\sin(x^2) + y\sin(y^2)dxdy = \frac{1}{2}(-b\cos(a^2) - a\cos(b^2) + a + b) \qquad [1.1]$$

According to the traapezoid rule we have:

$$\begin{aligned}\int_0^b \int_0^a x\sin(x^2) + y\sin(y^2)dxdy = \frac{dxdy}{4}\Big[& f(0,0) + f(a,0) + f(0,b) + f(a,b)) \qquad [1.2]\\ & + 2\sum_i f(x_i,0) + 2\sum_i f(x_i,b) + 2\sum_j f(0,y_j) + 2\sum_j f(a,y_j)\\ & + 4\sum_j\sum_i f(x_i,y_j)\Big]\end{aligned}$$

The following code performs that copmutation using MPI. More specifically we parallelize the computaion of the sums in eq.1.2 and tehn using the *MPI_Reduce* command with the *MPI_SUM* argument, we compute the final sum.

Listing 2: C++ code using listings

```
#include <iostream>
#include <stdlib.h>
#include <mpi.h>
#include <time.h>
#include <math.h>

using namespace std;
template <typename func_type>

double trapezoid(double a, double b, int n, func_type f, int rank, int si
{
  double dx   = a/n;
  double dy   = b/n;
  double sum1 = 0.0;        //sum_[f(xi,0) + f(xi,b)]
  double sum2 = 0.0;        //sum_[f(0,yj)+ f(a,yj)]
  double sum3 = 0.0;        //sum_f(xi,yj)

  for (int i = 1+rank; i<n; i+=size){
      sum1 += f(i*dx,0.0) + f(i*dx,b);
  }
  sum1 = 2.0 * sum1;

  for (int j = 1+rank; j<n; j+=size){
      sum2 += f(0.0,j*dy)+f(a,j*dy);
  }
  sum2 = 2.0 * sum2;

  for (int i = 1+rank; i<n; i+=size){
      for (int j = 1; j<n; j++){
          sum3 += f(i*dx,j*dy);
      }
  }
  sum3 = 4.0 * sum3;

  return (sum1 + sum2 + sum3);

}

double f(double x, double y)
{
        double z = x*sin(x*x) + y*sin(y*y);
//        double z = x*sin(pow(x,x)) + y*sin(pow(y,y));
  return z;
}

int main()
{

  int rank, size, ierr;
  MPI_Comm comm;

  comm   = MPI_COMM_WORLD;

  MPI_Init(NULL,NULL);
  MPI_Comm_rank(comm, &rank);
  MPI_Comm_size(comm, &size);
  MPI_Request request;    // for non-blicking send-receive later on
  MPI_Status status;

  double a    = 100;
  double b    = 100;
  int n       = 100000;
```

```cpp
  double dx  = a / n;
  double dy  = b / n;
  double res = trapezoid(a, b, n ,f,rank,size);
  double sum123;

  MPI_Reduce(&res,&sum123, 1, MPI_DOUBLE, MPI_SUM, 0, comm);
  double integral_appr = (dx*dy/4)*(f(0.0,0.0) +f(a,0.0) +f(0.0,b)+f(a,b)

  if (rank == 0){
     cout << "trapezoid integral approximation = "<< integral_appr << end
     cout << "analytical solution = "<< 0.5 * (-b*cos(a*a)-a*cos(b*b)+ a
<< endl;
  }

  MPI_Finalize();

}
```