HPC for Mathematicians - Week 4 Assignment

Aikaterini Karoni

February 10, 2021

Exercise 3

In the following code the halo swapping technique is implemented to solve the 1D heat equation in parallel. The heat equation is:

$$u_t = u_{xx} \text{ for } x \in (0, 1), t \in (0, T)$$

subject to the initial condition

$$u(x, t = 0) = \sin(2\pi x) + 2\sin(5\pi x) + 3\sin(20\pi x)$$

and the homogeneous Dirichlet Boundary Conditions

$$u(x = 0, t) = u(x = 1, t) = 0$$

The parallel code's results validity was verified by comparing them to those of the serial code and were found to be the same. Also the running time was computed for different numbers of processors and the results are presented in the graph below.

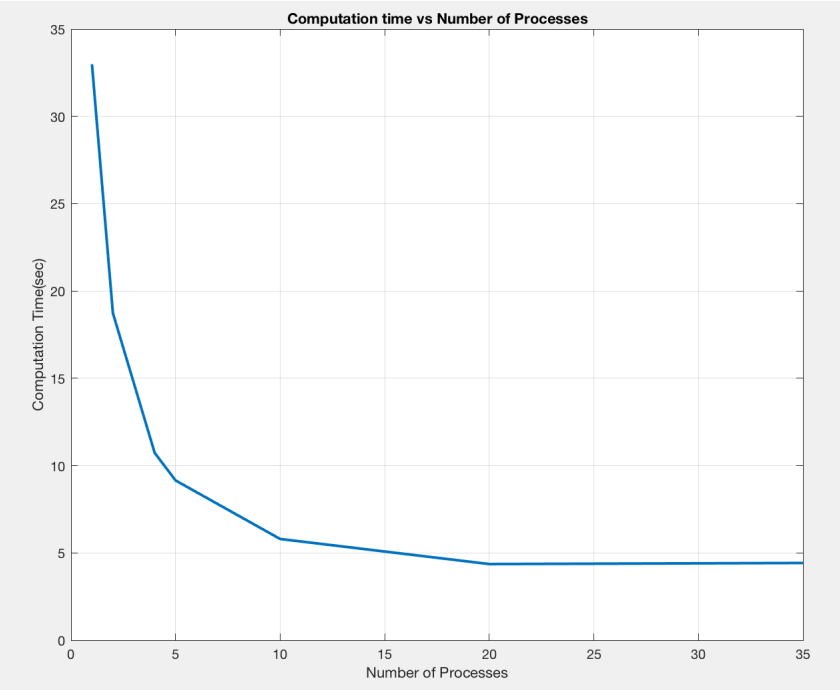

Figure 1:

The code written to perform the parallel computations is the following:

Listing 1: C++ code using listings

```
1
2  #include <iostream>
3  #include <stdlib.h>
4  #include <mpi.h>
5  #include <time.h>
6  #include <cmath>
7
8  using namespace std;
9
10 static const double PI = 3.1415926536;
11
12 int main(int argc, char* argv[]){
13
14   int rank, size, ierr;
15   MPI_Comm comm;
16   comm  = MPI_COMM_WORLD;
17   MPI_Init(NULL,NULL);
18   MPI_Comm_rank(comm, &rank);
19   MPI_Comm_size(comm, &size);
20   MPI_Request request;    // for non-blicking send-receive later on
21   MPI_Status status;
22
23   int M = 100;  // M length intervals
24   int N = 10000; // N time intervals
25   int Jn = ((M+1)-2)/size + 2;    // remember M ->not #points, but #interv
26   double T = atof(argv[1]);  // get final time from input argument
27   double U[2][M+1];  // stores the numerical values of function U; two ro
28   double Ul[2][Jn];  // local array Ul that each of the processes works w
29   double Usol[M+1];  // stores true solution
30   double dt = T/N;
31   double dx = 1./M;
32   double dtdx = dt/(dx*dx);
33   double t1,t2;
34
35   t1 =  MPI_Wtime();
36
37   // initialize numerical array with given conditions
38   U[0][0]=0, U[0][M]=0, U[1][0]=0, U[1][M]=0;          // U(t,x=0) = U(t,x=
39   for(int m=1; m<M; ++m){
40       U[0][m] = sin(2*PI*m*dx) + 2*sin(5*PI*m*dx) + 3*sin(20*PI*m*dx);
41   }
42
43   for (int m=0; m<Jn; m++){
44       Ul[0][m]=U[0][rank*(Jn-2)+m];
45    }
```

```
46
47    if (rank == 0){     // left boundary condition
48       Ul[1][0] = 0;
49    }
50
51    if (rank == size-1){ // right boundary condition
52       Ul[1][Jn-1] = 0;
53    }
54
55    // use numerical scheme to obtain the future values of U on the M+1 spa
56    for(int i=1; i<=N; ++i){
57            for (int m=1; m<Jn-1; ++m){
58             Ul[1][m] = Ul[0][m] + dtdx *(Ul[0][m-1] - 2*Ul[0][m] + Ul[0][m
59        }
60
61
62        if (rank !=0){     // each process apart from the 1st one sends its
63                           // element to the previous one
64              MPI_Send(&Ul[1][1],1, MPI_DOUBLE, rank-1, 2, comm);
65        }
66
67            if (rank != size-1){  // each process apart from the last one r
68                               // the second element sent by the next proces
69              MPI_Recv(&Ul[1][Jn-1],1, MPI_DOUBLE, rank+1, 2, comm, MPI_ST
70            }
71
72        if (rank !=size-1){     // each process apart from the last one send
73                                // its previous to last element to the next
74              MPI_Send(&Ul[1][Jn-2],1, MPI_DOUBLE, rank+1, 2, comm);
75        }
76
77            if (rank != 0){  // each process apart from the first one recei
78                               // the second element sent by the previous pr
79                 MPI_Recv(&Ul[1][0],1, MPI_DOUBLE, rank-1, 2, comm, MPI_ST
80            }
81
82        for(int m=0; m<=Jn-1; m++){
83            Ul[0][m] = Ul[1][m];
84            }
85
86    }
87
88    if (rank==0){
89       for (int m=0; m<Jn; m++){
90           U[1][m] = Ul[1][m];
91       }
92    }
93
```

```
94    if (rank!=0){
95       MPI_Send(&Ul[1][0],Jn, MPI_DOUBLE, 0, 2, comm);
96    }
97
98    if (rank==0){
99       for (int r=1; r<=size-1; r++){
100          MPI_Recv(&U[1][r*(Jn-2)],Jn, MPI_DOUBLE, r, 2, comm, MPI_STATUS_
101      }
102   }
103
104   if (rank==0){
105     cout<< "\ndx="<<dx<<", dt="<<dt<<", dt/dx ="<< dtdx<<endl;
106     // print out array entries of numerical solution next to true solutio
107     cout << "\nTrue and numerical values at M="<<M<<" space points at tim
108     cout << "\nTrue values          Numerical solutions\n"<<endl;
109     for(int m=0; m<=M; ++m){
110        Usol[m] = exp(-4*PI*PI*T)*sin(2*PI*m*dx) + 2*exp(-25*PI*PI*T)*sin
111        cout << Usol[m] << "            " << U[1][m] << endl;
112        // note that we did not really need to store the true solution in
113     }
114   }
115
116   t2 = MPI_Wtime();
117
118   if (rank==0){
119      cout << "time to run= " << t2-t1 << endl;
120   }
121   MPI_Finalize();
122
123 }
```