Workshop 4

Peter Whalley

February 10, 2021

1 Exercise 3

In this exercise, we use halo-swapping technique to solve a 1D heat equation in parallel and analyse the speed up with number of processors. The equation we solve is

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

subject to the initial conditions

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

with homgeneous Dirchlet boundary conditions

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

The code below illustrates my parallel implementation of a numerical solver for the heat equation. I have set $N = 2000000$, the number of time intervals to ensure numerical stability and constructed M such that it is divisible by a large amount of numbers between 1 and 36 for testing. I have made a local vector U of size J for each process to save memory and then iterated the central differencing procedure on each local process with halo-swapping for boundary terms. I have then sent all the final local iterates to the root process and concatenated them into a complete solution. I have compared the results with the serial code and they get the same numerical results for the same M and N.

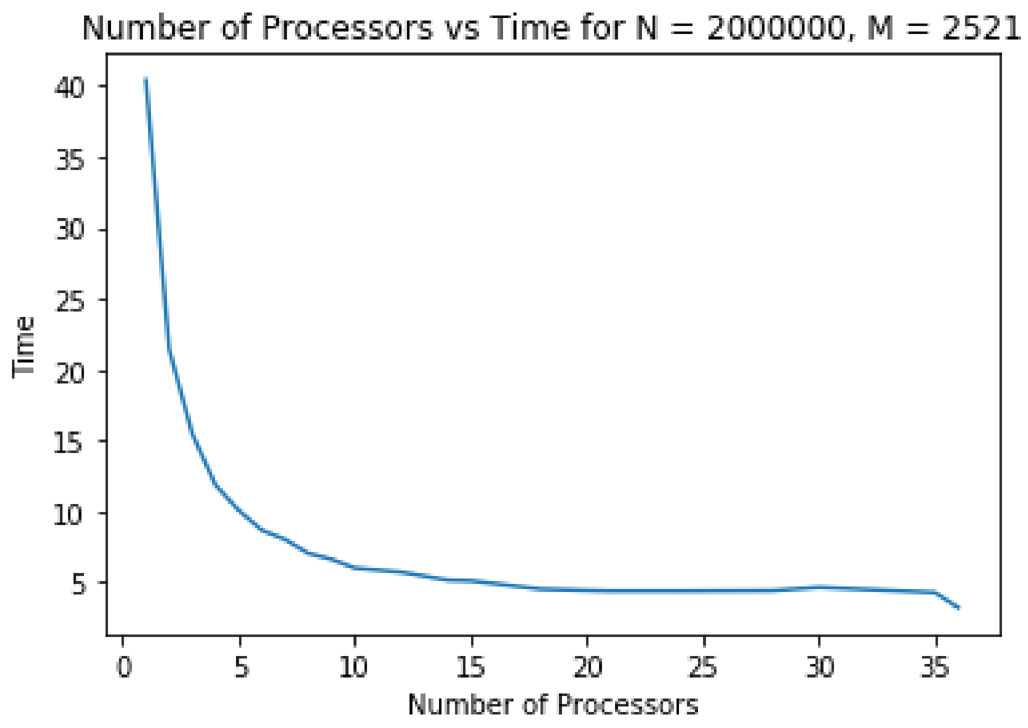
Listing 1: C++ code using listings

```
#include <iostream>
#include <cmath>
#include <mpi.h>

using namespace std;

static const double PI = 3.1415926536;
```

```

int main(int argc, char* argv[]){

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);

int N = 2000000; // N time intervals
int M = 2*3*4*5*7*3 + 1;
int J = (M-1)/size +2;
double T = atof(argv[1]);  // get final time from input argument
double U[J];  // stores the numerical values of function U; two rows to also
double Unew[J];
double UFinal[M+1]; //Final U
double Usol[M+1];  // stores true solution
double dt = T/N;
double dx = 1./M;
double dtdx = dt/(dx*dx);
cout<< "\ndx="<<dx<<", dt="<<dt<<", dt/dx ="<< dtdx<<endl;

double t1, t2; //timings

t1 = MPI_Wtime();
UFinal[0] = 0;
UFinal[M] = 0;
// initialize numerical array with given conditions
if (rank == 0){
     U[0] = 0;
     Unew[0] = 0;
}
if (rank == size - 1){
     U[J-1] = 0;
     Unew[J-1] = 0;
}
//Initial condition on each process.
```

```
for(int m=0; m<J; ++m){
     U[m] = sin(2*PI*(m+rank*(J-2))*dx) + 2*sin(5*PI*(m+rank*(J-2))*dx) + 3*si
}

for(int i=1; i<=N; ++i){
     for (int m=1; m<J-1; ++m){
          Unew[m] = U[m] + dtdx * (U[m-1] - 2*U[m] + U[m+1]);
     }
     // update "old" values
     for(int m=1; m<J-1; ++m){
          U[m] = Unew[m];
     }
     if (rank < size-1){
     MPI_Send(&U[J-2],1,MPI_DOUBLE,rank+1,2,comm);
}
     if (rank > 0){
     MPI_Send(&U[1],1,MPI_DOUBLE,rank-1,2,comm);
     MPI_Recv(&U[0],1,MPI_DOUBLE,rank-1,2,comm,MPI_STATUS_IGNORE);
}
     if (rank < size-1){
     MPI_Recv(&U[J-1],1,MPI_DOUBLE,rank+1,2,comm,MPI_STATUS_IGNORE);

     }

}
//Sending all the J vectors to the root process.
if (rank!=root){
MPI_Send(&U,J,MPI_DOUBLE,root,rank,MPI_COMM_WORLD);
}

//On the root process collecting all J and collating them in a final solution
if (rank == root){
for (int j=0;j<J;j++){
UFinal[j] = U[j];
}
for (int i=1; i<size; i++){
MPI_Recv(&UFinal[(J-2)*i],J,MPI_DOUBLE,i,i,MPI_COMM_WORLD, MPI_STATUS_IGNORE)
}
t2 = MPI_Wtime();
cout <<"Time taken with "<<size<<" processes = "<<t2-t1<<endl;
// print out array entries of numerical solution next to true solution
cout << "\nTrue and numerical values at M="<<M<<" space points at time T="<<T
cout << "\nTrue values           Numerical solutions\n"<<endl;
for(int m=0; m<=M; ++m){
      Usol[m] = exp(-4*PI*PI*T)*sin(2*PI*m*dx) + 2*exp(-25*PI*PI*T)*sin(5*PI*m*
      cout << Usol[m] << "                    " << UFinal[m] << endl;
      // note that we did not really need to store the true solution in the arr
}
}

MPI_Finalize();
}
```

I have used *MPI_WTime*() to time the results for different numbers of processes and I have plotted the results below.



The results seem to plateau, due the serial part of the code dominating the runtime. This causes the parallelisation procedure to not improve the speed up of the algorithm by much further. However compared to serial code (1 process) there is a significant speed up with up to 10 processes.