There are few key steps in my codes:

1. I have used two lists to repeatedly store and calculate the result. Initial[] as the foundation and calcu[] as the calculation result for each step.

2. Using mod operator to calculate how many grids each core to calculate.

```
int sep;
sep = (m - 1) / size;
if ((m - 1) % size != 0) { sep++; }
```

3.Communication between different cores:

In each loop, most of the cores will send its tails and head to neighborhood, also will receive from its neighborhood. Expect the first and last core, each core will receive and send two message to different neighbor in each loop.

T	dt	steps	grids=m+1	dt/dx^2	Cores	Runing time(s)
0.5	0.005	100	11	0.5	4	0.000141145
0.5	0.00005	10000	101	0.5	4	0.020325
0.5	0.00005	10000	101	0.5	8	0.0204618
0.5	0.0000005	1000000	1001	0.5	4	10.3782
0.5	0.0000005	1000000	1001	0.5	8	9.98308

It seems that introducing more cores did not significantly increase improve the calculating speed under these schemes, the time for communication is indeed a big consume of time.

```
#include <iostream>
#include <mpi.h>
#include <math.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);

  float T, dt, dx,mm,t;
  int m, n,m1,n1;
  int i, j, count = 0;
  double pi = 3.14159265358979323846;
  m = 1000;
  n = 5;
  mm = 1000;
  dx = 1 / mm;  // x \in (0,1)
  T = 0.5;     // T=0.5
  dt = 0.0000005;

  float initial[m + 1+size+1], calcu[m + 1+size+1];

  initial[0] = 0;
  initial[m] = 0;
  //cout << "I am "<<rank<<" out of "<<size<<" and closest multiple of 3 to me is ..."<<endl;


  for (j = 1; j < m; j++) {
       initial[j]= sin(2*pi*j*dx)+2* sin(5 * pi * j * dx)+ 3*sin(20 * pi * j * dx);
  }


  //MPI_Send(&(c1[0]), 7, MPI_INT, 0, 1, MPI_COMM_WORLD);
  //MPI_Recv(&(c2), 7, MPI_INT, 0, 2, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

  int sep;
  sep = (m - 1) / size;
  float dtxx = (dt / dx) / dx;
  if ((m - 1) % size != 0) { sep++; }

  n = T/dt;
  cout << "i am n " << n << endl;
  cout << "sep " << sep << endl;
  cout << "dtxx   size " << dtxx << "  "<<size << endl;
  //cout << "I am "<<rank<<" out of "<<size<<endl;"
  double  t_start, t_end;
  MPI_Barrier(MPI_COMM_WORLD);
  t_start = MPI_Wtime();

  for (i = 0; i < n; i++) { // n steps on time variable
    calcu[0] = 0;
    calcu[m] = 0;
    initial[0] = 0;
    initial[m] = 0;

    for (j = 1; j < m; j++) { //calculation;
        if (  (rank * sep < j) && ((rank + 1) * sep >= j) ) {
            calcu[j] = initial[j] + dtxx * (initial[j - 1] - 2 * initial[j] + initial[j + 1]);
            //cout << "I am " << rank << " calclulating calcu " << j << endl;
        }
    }

    if (i == n - 1) {continue;}

    for (j = 1; j < m; j++) { //renew inital;
        if ((rank * sep < j) && ((rank + 1) * sep >= j)) {
            initial[j]= calcu[j];

        }

        //send calcu's head
        if ((j == rank*sep +1) && (rank > 0)) {
            MPI_Send(&(calcu[j]), 1, MPI_FLOAT, (rank - 1), (i + 1) * rank, MPI_COMM_WORLD);
            //cout << "I am " << rank << " sending calclu[]" << j << " to "<<rank-1 << endl;
        }

        //send calcu's tail
        if ((j == (rank + 1) * sep)&&(rank<size-1)) {
            MPI_Send(&(calcu[j]), 1, MPI_FLOAT, (rank+1), (i + 1) * rank, MPI_COMM_WORLD);
            //cout << "I am " << rank << " sending calclu[]" << j << " to " << rank + 1 << endl;
        }

        //receive other's head, his tail
        if ((j == (rank+1) * sep + 1) && (rank<size-1)) {
            MPI_Recv(&(initial[j]), 1, MPI_FLOAT, rank+1, (i + 1) * (rank+1), MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            //cout << "I am " << rank << " receiving calclu[]" << j << " from " << rank + 1 << endl;
        }
        //receive other's tail, his head
        if ((j == (rank - 1) * sep ) && (rank >0)) {
            MPI_Recv(&(initial[j]), 1, MPI_FLOAT, rank - 1, (i + 1) * (rank - 1), MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            //cout << "I am " << rank << " receiving calclu[]" << j << " from " << rank - 1 << endl;
        }
    }
        }


        if (rank > 0 && rank < size - 1 ) {
          MPI_Send(&(calcu[(rank)*sep+1]), sep, MPI_FLOAT, 0, rank, MPI_COMM_WORLD);
      }

        if (rank == size - 1) {
            i = m - 1 - (rank * sep);
```

```
       MPI_Send(&(calcu[(rank)*sep + 1]), i, MPI_FLOAT, 0, rank, MPI_COMM_WORLD);

     }

    if (rank == 0) {
      for (i = 1; i < size-1; i++) {
        MPI_Recv(&(calcu[i*sep+1]), sep, MPI_FLOAT, i , i , MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      }
      MPI_Recv(&(calcu[(size-1) * sep + 1]), m-1-sep*(size-1), MPI_FLOAT, size - 1, size - 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      calcu[0] = 0;
      calcu[m] = 0;



      cout<<"This is the result: "<<endl;
      for (i = 0; i <= m; i++) {
        cout << calcu[i] << " ";
      }
      cout << " " << endl;

      cout << "This is the true result: " << endl;
      cout << 0 << " ";

      for (i = 1; i < m; i++) {
        cout << exp(-4 * pi * pi * T) * sin(2 * pi * i * dx) + 2 * exp(-25 * pi * pi * T) * sin(5 * pi * i * dx) + 3 * exp(-400 * pi * pi * T) * sin(20 * pi * i * dx)<< " ";
      }

      cout << 0 << " ";
      cout << " " << endl;


    }
    MPI_Barrier(MPI_COMM_WORLD);
    t_end = MPI_Wtime();
    cout << "Running time  " << t_end - t_start << endl;

     MPI_Finalize();

    }
```