**SF2568: Homework 2**

Yue Jiao

911024-7799

1. One to all collective communication
   1. In this question, we want to design a point-to-point communication algorithm between P nodes so that the total communication steps satisfy . The algorithm works in the following way.

Assuming that the node contains the data that shall be broadcasted. Then repeat the following step.

Let node which contains the data send to the node .

So, after each step of communication, the number of nodes that contains the data doubles.

The iteration terminates when all nodes contain the data. With total nodes, the communication complexity will be .

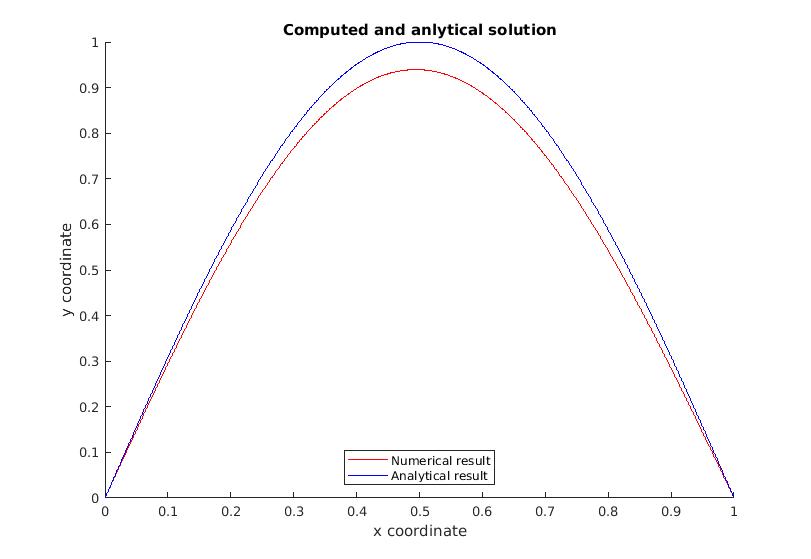
* 1. Let be the number of words that should be broadcasted. Let be the latency. Let be the time needed to send one word. Then we can see that each step of communication takes the time . If is the smallest integer that satisfies then exact steps are needed to broadcast data to all nodes. So, the total communication time shall be:
  2. A similar procedure like the broadcasting operation can be used to do the scatter operation.

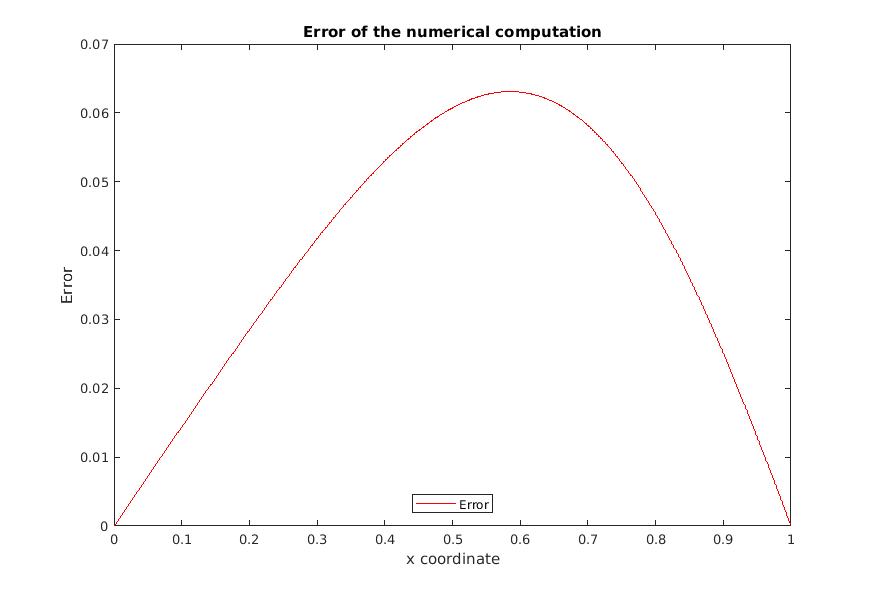
First give all data a label that identify the order of the data. The label shall be the rank of the node. Let be the biggest positive integer that satisfies . Then let . Then repeat the following steps.

For each node from the node to the node receive all data from node . If there is no such a node then nothing should be received. These communications shall be synchronized so that all communication should be done before the beginning of next step. When the communication is done, let .

The iteration terminates when . Then the node should have received data from all nodes. Then it can sort the data with the help of their labels.

1. This question is about to transpose a vector in the memory which has been calculated by matrix-vector multiplication.
   1. Since we are having a processes meshing, each row of the meshing should contains the result . What we want is that the column should contains . This can be done by letting the first processes read from the corresponding process . Then let each column of processes do the data broadcasting describe above from the process to all processes where .
   2. The time needed for the first communications is . Then each broadcasting takes the time where is the smallest integer that satisfies . These broadcastings happen times so the total broadcasting time is and the total communication time is then:
2. I choose to let and . So, the analytical function corresponds to this situation is that . The result and the error are plotted below:





The code is listed below and uploaded in this address:

https://pastebin.ubuntu.com/p/hyTXdHhxgm/

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <mpi.h>

#define MIN(a,b) ((a) < (b) ? (a) : (b))

#define PI 3.1415926535897

#define N 1000

#define SMX 10000000

inline double r(double x) {

return -x\*x;

}

inline double f(double x) {

return x\*x\*sin(PI\*x) - PI\*PI\*sin(PI\*x);

}

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

{

int rc, P, p;

MPI\_Status\* status;

status = (MPI\_Status \*) malloc (sizeof(MPI\_Status));

rc = MPI\_Init(&argc, &argv);

rc = MPI\_Comm\_size(MPI\_COMM\_WORLD, &P);

rc = MPI\_Comm\_rank(MPI\_COMM\_WORLD, &p);

if (N < P) {

fprintf(stdout, "Too few discretization points...\n");

exit(1);

}

int L = N/P;

int R = N%P;

int I = (N+P-p-1)/P;

double h = 1.0/(N+1.0);

double \*rr, \*ff, \*unew, \*u;

rr = (double \*) malloc ((N+2)\*sizeof(double));

ff = (double \*) malloc ((N+2)\*sizeof(double));

unew = (double \*) malloc(I\*sizeof(double));

u = (double \*) calloc(I+2, sizeof(double));

for(int i=0; i<N+2; i++) {

rr[i] = r(((double)i)\*h);

ff[i] = f(((double)i)\*h);

//printf("i: %d, rr: %F, ff: %F\n", i, rr[i], ff[i]);

}

for (int step = 0; step < SMX; step++) {

if (p%2) {

if (p+1 != P) {

rc = MPI\_Send(unew + I - 1, 1, MPI\_DOUBLE, p+1, 0, MPI\_COMM\_WORLD);

rc = MPI\_Recv(u + I + 1, 1, MPI\_DOUBLE, p+1, 0, MPI\_COMM\_WORLD, status);

}

if (0 != p) {

rc = MPI\_Send(unew + 0, 1, MPI\_DOUBLE, p-1, 0, MPI\_COMM\_WORLD);

rc = MPI\_Recv(u + 0, 1, MPI\_DOUBLE, p-1, 0, MPI\_COMM\_WORLD, status);

}

} else {

if (0 != p) {

rc = MPI\_Recv(u + 0, 1, MPI\_DOUBLE, p-1, 0, MPI\_COMM\_WORLD, status);

rc = MPI\_Send(unew + 0, 1, MPI\_DOUBLE, p-1, 0, MPI\_COMM\_WORLD);

}

if (p+1 != P) {

rc = MPI\_Recv(u + I + 1, 1, MPI\_DOUBLE, p+1, 0, MPI\_COMM\_WORLD, status);

rc = MPI\_Send(unew + I - 1, 1, MPI\_DOUBLE, p+1, 0, MPI\_COMM\_WORLD);

}

}

for(int i = 0; i < I; i++) {

int n = p\*L + MIN(p,R) + i + 1;

unew[i] = (u[i]+u[i+2]-h\*h\*ff[n])/(2.0-h\*h\*rr[n]);

}

for(int i=0; i < I; i++) {

u[i+1] = unew[i];

}

}

int\* signal;

signal = (int\*) malloc (sizeof(int));

int tmp;

MPI\_Barrier(MPI\_COMM\_WORLD);

printf("done computing\n");

if (p == 0) {

FILE\* file = fopen("res.txt", "a");

for(int i = 0; i < I; i++) {

fprintf(file, "%F\n", unew[i]);

}

fclose(file);

tmp = 1;

rc = MPI\_Send(&tmp, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

} else {

rc = MPI\_Recv(signal, 1, MPI\_INT, p-1, 0, MPI\_COMM\_WORLD, status);

if(\*signal == p) {

FILE\* file = fopen("res.txt", "a");

for(int i = 0; i < I; i++) {

fprintf(file, "%.8F\n", unew[i]);

}

fclose(file);

if(p!=P-1) {

tmp = p+1;

rc = MPI\_Send(&tmp, 1, MPI\_INT, p+1, 0, MPI\_COMM\_WORLD);

}

}

}

MPI\_Finalize();

return 0;

}

The result is plotted with following Matlab code:

data = textread('res.txt');

data = [0; data; 0];

x = (0:1:1001)/(1001);

u = sin(pi\*x);

figure();

hold on

plot(x, data, 'r');

plot(x, u, 'b');

legend('Numerical result', 'Analytical result', 'Location', 'South')

title('Computed and anlytical solution')

xlabel('x coordinate')

ylabel('y coordinate')

axis([0 1 0 1])

figure();

plot(x, abs(u'-data), 'r');

legend('Error', 'Location', 'South')

title('Error of the numerical computation')

xlabel('x coordinate')

ylabel('Error')

axis([0 1])