**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**"Уфимский университет науки и технологий"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Параллельное программирование

**Отчет по лабораторной работе № 3**

**Тема:** «Параллельный алгоритм решения краевой задачи для уравнения теплопроводности»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМИ-354 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Мова И.А. |  |  |  |
| Принял | Спеле В.В. |  |  |  |

**Уфа 2023**

**Цель:** научиться использовать принцип геометрического параллелизма для разработки и программной реализации параллельных алгоритмов решения краевых задач для уравнений эволюционного типа на примере параллельной реализации явной численной схемы для решения краевой задачи уравнения теплопроводности.

**Теоретический материал**

*MPI (Message Passing Interface****)*** – это стандартизованная библиотека функций, призванная обеспечить совместную работу параллельных процессов путем организации передачи сообщений между ними.

*Основные понятия MPI*

* Каждая программа представляет собой совокупность одновременно работающих процессов, которые могут обмениваться сообщениями;
* Все процессы объединяются в группы;
* Обмен сообщениями возможен между процессами одной группы, которой поставлена в соответствие своя область связи;
* Коммуникатор – идентификатор области;
* При запуске программы все доступные ей процессы объединяются в начальную группу с общей областью связи, имеющей коммуникатор MPI\_COMM\_WORLD.

*Особенности реализации MPI для C/C++*

1. Первая строка программы: #include “mpi.h”;
2. В MPI принят ANSI C стандарт;
3. Нумерация массивов начинается с 0;
4. Массивы хранятся по строкам;
5. Логические переменные являются переменными типа integer со значением 0 в случае false и любым не нулевым значением, обозначающем true.

*Базовые функции MPI*

1. int MPI\_Init(int \*argc, char \*\*argv[]); – инициализация параллельной части программы.

Возвращает предопределенные константы

MPI\_SUCCESS – возвращается в случае успешного выполнения,

MPI\_ERR\_ARG – ошибка неправильного задания аргумента,

MPI\_ERR\_INTERN – внутренняя ошибка (нехватка памяти),

MPI\_ERR\_UNKNOWN – неизвестная ошибка.

1. int MPI\_Finalize(void); – завершение параллельной части программы.
2. int MPI\_Comm\_size(MPI\_Comm comm, int\* size); – определение числа процессов size в коммуникационной группе с коммуникатором comm.
3. int MPI\_Comm\_rank(MPI\_comm comm, int\* rank); – определение номера rank вызвавшего ее процесса, входящего в коммуникационную группу с коммуникатором comm
4. int MPI\_Send(void\* sbuf, int count, MPI\_Datatype datatype, int dest, int tag, MPI\_Comm comm) – передача сообщений от одного процесса к другому

sbuf – адрес в памяти, начиная с которого размещаются передаваемые данные;

count – количество передаваемых элементов;

datatype – тип передаваемых элементов;

dest – номер процесса-получателя сообщения;

tag – метка передаваемого сообщения;

comm – коммуникатор.

1. int MPI\_Recv(void\* rbuf, int count, MPI\_Datatype datatype, int source, int tag, MPI\_comm comm, MPI\_Status \*status) – прием сообщений от одного процесса к другому.

*Входные параметры:*

count – количество получаемых элементов;

datatype – тип получаемых элементов;

source – номер процесса-отправителя сообщения;

tag – метка принимаемого сообщения;

comm – коммуникатор.

*Выходные параметры:*

rbuf – адрес в памяти, начиная с которого размещаются принимаемые данные;

status – структура, содержащая информацию о принятом сообщении.

Структура status имеет три поля.

Status.MPI\_SOURCE – номер процесса-отправителя;

Status.MPI\_TAG – метка принимаемого сообщения;

Status.MPI\_ERROR – код завершения приема сообщения.

*Функция совмещенного приема/передачи.*

int MPI\_Sendrecv(void\* sbuf, int scount, MPI\_Datatype sdatatype, int dest, int stag, void\* rbuf, int rcount, MPI\_Datatype rdatatype, int source, int rtag, MPI\_comm comm, MPI\_Status \*status);

*Входные параметры:*

sbuf – адрес в памяти, начиная с которого размещаются передаваемые данные;

scount – количество передаваемых элементов;

sdatatype – тип отправляемых элементов;

dest – номер процесса-получателя сообщения;

stag – метка отправляемого сообщения;

rcount – количество получаемых элементов;

rdatatype – тип получаемых элементов;

source – номер процесса-отправителя сообщения;

rtag – метка принимаемого сообщения;

comm – коммуникатор.

*Выходные параметры:*

rbuf – адрес в памяти, начиная с которого размещаются принимаемые данные;

status – структура, содержащая информацию о принятом сообщении.

Для анализа данных, полученных в ходе проведения лабораторной работы, будут использоваться следующие определения:

* Отношение времени выполнения параллельной программы на одном процессоре (ядре) ко времени выполнения параллельной программы на процессорах называется ускорением при использовании p процессоров:

* Отношение ускорения к количеству процессоров называется эффективностью при использовании процессоров:

*Функции коллективного взаимодействия*

1. int ***MPI\_Bcast***(void \*buf, int count, MPI\_Datatype datatype, int root, MPI\_Comm comm);Функция предназначена для рассылки данных, хранящихся на одном процессе, всем остальным процессам группы.

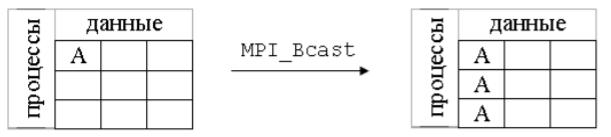


Рисунок 1. MPI\_Bcast

*Входные параметры:*

buf – адрес в памяти, начиная с которого размещаются передаваемые данные;

count – количество рассылаемых элементов;

datatype – тип отправляемых данных;

root – номер процесса-отправителя сообщения;

comm – коммуникатор.

1. int ***MPI\_Gather***(void \*sbuf, int scount, MPI\_Datatype stype, void \*rbuf, int rcount, MPI\_Datatype rtype, int root, MPI\_Comm comm);

Функция предназначена для сбора данных со всех процессов на одном (так называемый, "совок“).



Рисунок 2. MPI\_Gather

*Входные параметры*:

sbuf – адрес в памяти на каждом процессе, начиная с которого размещаются отправляемые данные;

scount – количество элементов, отправляемых с каждого процесса;

stype – тип отправляемых данных;

rcount – количество принимаемых элементов от каждого процесса;

rtype – тип принимаемых данных;

root – номер процесса, на котором осуществляется сборка сообщений;

comm – коммуникатор.Выходные параметры:

rbuf – адрес в памяти на процессе с номером root, начиная с которого размещаются принимаемые сообщения.

1. Вариант функции с варьируемым кол-вом собираемых элементов:

int ***MPI\_Gatherv***(void \*sbuf, int scount, MPI\_Datatype stype, void \*rbuf, int \*rcounts, int \*displs, MPI\_Datatype rtype, int root, MPI\_Comm comm);

*Характерные отличия:*

rcounts – массив длин принимаемых от процессов сообщений;

displs – массив позиций в приемном буфере, по которым размещаются принимаемые сообщения.

1. int ***MPI\_Scatter***(void \*sbuf, int scount, MPI\_Datatype stype, void \*rbuf, int rcount, MPI\_Datatype rtype, int root, MPI\_Comm comm);

Функция предназначена для рассылки данных с одного процесса всем остальным процессам (так называемый, “разбрызгиватель”).

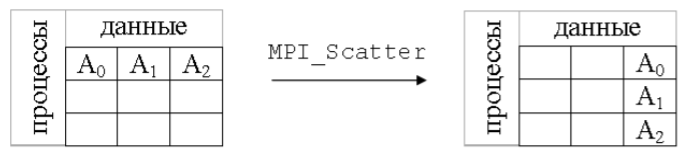


Рисунок 3. MPI\_Scatter

*Входные параметры:*

sbuf – адрес в памяти на процессе-отправителе сообщения, начиная с которого размещаются отправляемые данные;

scount – количество элементов, отправляемых каждому процессу;

stype – тип отправляемых данных;

rcount – количество элементов, принимаемых каждым процессом (длина принимаемого сообщения);

rtype – тип принимаемых данных;

root – номер процесса-отправителя сообщения;

comm – коммуникатор.Выходные параметры:

rbuf – адрес в памяти на каждом процессе, начиная с которого размещаются принимаемые сообщения.

1. Вариант функции с варьируемым кол-вом рассылаемых элементов:

int ***MPI\_Scatterv***(void \*sbuf, int \*scounts, int \*displs, MPI\_Datatype stype, void \*rbuf, int rcount, MPI\_Datatype rtype, int root, MPI\_Comm comm);

*Характерные отличия:*

scounts – массив, содержащий количество элементов в каждой части, на которые разбивается сообщение;

displs – массив позиций, определяющий начальные положения каждой части сообщения.

1. int ***MPI\_Reduce***(void \*sbuf, void \*rbuf, int count, MPI\_Datatype datatype, MPI\_Op op, int root, MPI\_Comm comm);

Функция проделывает операцию op над данными, хранящимися в sbuf в каждом процессе группы, результат которой записывается в rbuf в процесс с номером root.

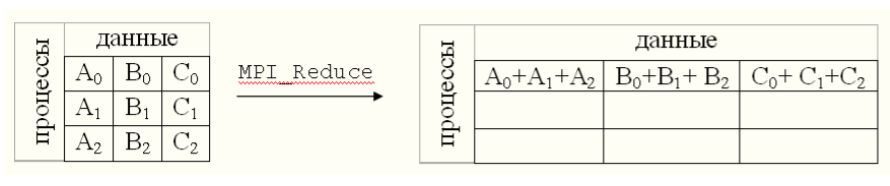


Рисунок 4. MPI\_Reduce

*Входные параметры:*

sbuf – адрес в памяти на каждом процессе, по которому хранятся исходные данные для распределенной операции;

count – количество элементов в sbuf;

datatype – тип данных, над которыми производится распределенная операция;

root – номер процесса, на котором осуществляется размещение результата выполнения операции;

op – название распределенной операции;

comm – коммуникатор.

*Выходные параметры:*

rbuf – адрес в памяти, по которому размещаются результаты выполнения операции.

***12 предопределенных операций:***

MPI\_MAX – поиск поэлементного максимума;

MPI\_MIN – поиск поэлементного минимума;

MPI\_SUM – вычисление суммы векторов;

MPI\_PROD – вычисление поэлементного произведения векторов;

MPI\_LAND – логическое “И”;

MPI\_LOR – логическое “ИЛИ”;

MPI\_LXOR – логическое исключающее “ИЛИ”;

MPI\_BAND – бинарное “И”;

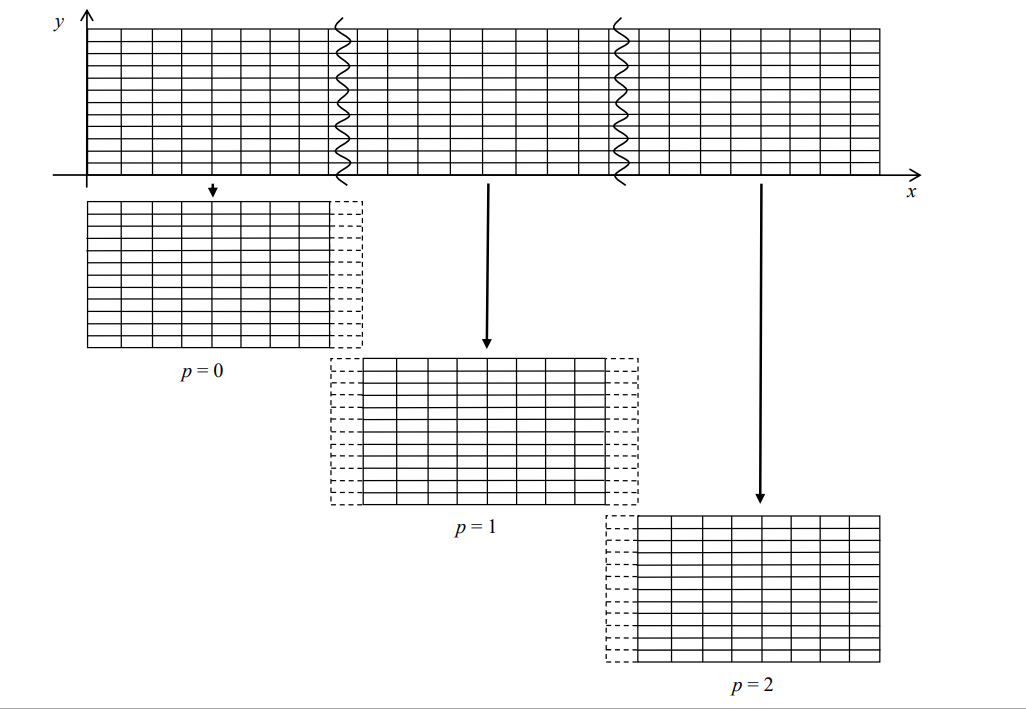
MPI\_BOR – бинарное “ИЛИ”;

MPI\_BXOR – бинарное исключающее ИЛИ;

MPI\_MAXLOC – поиск индексированного максимума;

MPI\_MINLOC – поиск индексированного минимума.

Декомпозиция пространственной области по оси Ox при трех процессах



Необходимо найти решение краевой задачи уравнения теплопроводности

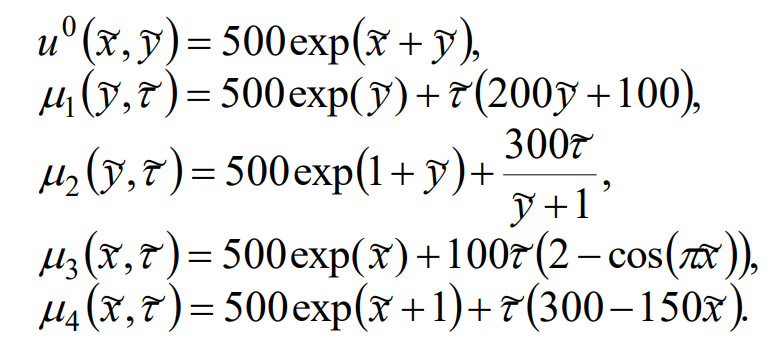
с начальным условием

и граничными условиями

Удельная теплоемкость, плотность и коэффициент теплопроводности промаются равными

Согласно теории конечных разностей:

Условие устойчивости:

**Практическая часть**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | | |
|  |  |  |
| 1 | 619,45 | 591,89 | 576,73 |
| 2 | 316,93 | 289,22 | 274,85 |
| 4 | 165,15 | 156,31 | 142,17 |
| 8 | 88,92 | 89,23 | 79,96 |
| 16 | 50,73 | 42,74 | 36,43 |
| 32 | 32,53 | 20,27 | 18,72 |
| 64 | 18,93 | 10,88 | 9,68 |

Таблица 1. Результаты замеров времени при разном числе процессов.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | |  | | |
|  |  |  |  |  |  |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1,954533 | 2,046504 | 2,098345 | 0,977266 | 1,023252 | 1,049172 |
| 4 | 3,750833 | 3,786642 | 4,056622 | 0,937708 | 0,94666 | 1,014156 |
| 8 | 6,966374 | 6,633307 | 7,212731 | 0,870797 | 0,829163 | 0,901591 |
| 16 | 12,21072 | 13,84862 | 15,83118 | 0,76317 | 0,865539 | 0,989449 |
| 32 | 19,04242 | 29,2003 | 30,80823 | 0,595076 | 0,912509 | 0,962757 |
| 64 | 32,72319 | 54,40165 | 59,57955 | 0,5113 | 0,850026 | 0,93093 |

Таблица 2. Ускорение и эффективность при разном числе процессов

Рисунок 5. Графики зависимости ускорения от числа процессов при разных размерностях сетки

Рисунок 6. Графики зависимости эффективности от числа процессов при разных размерностях сетки

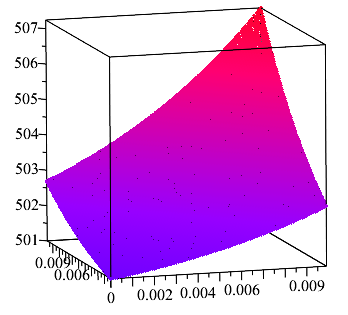


Рисунок 6. Поле температур при и

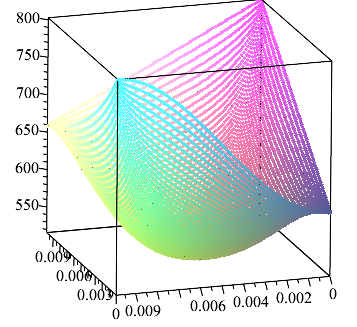


Рисунок 7. Поле температур при и

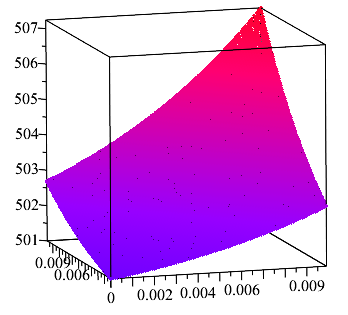


Рисунок 6. Поле температур при и

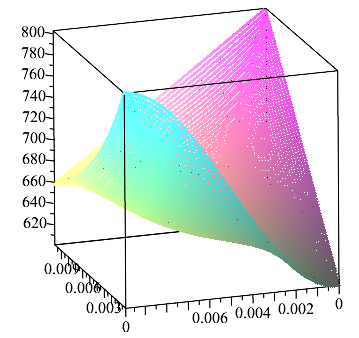


Рисунок 7. Поле температур при и

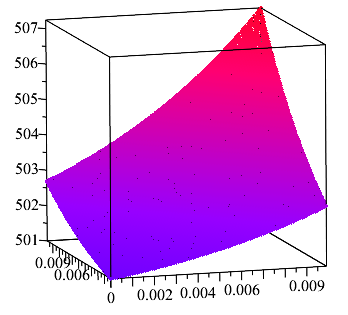


Рисунок 8. Поле температур при и

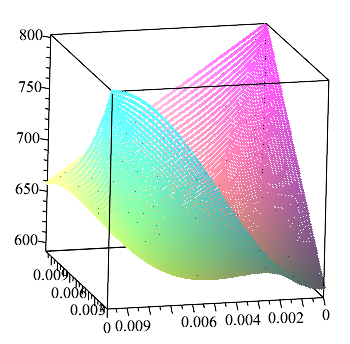


Рисунок 9. Поле температур при и

**Вывод**: в ходе лабораторной работы был реализован алгоритм решения двумерного уравнения теплопроводности по явной схеме, проведен анализ его эффективности.

**Код программы.**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <iostream>

#include <fstream>

#define Pi 3.1415926535897

#define K 0.00001864472034

#define L 0.01

using namespace std;

//задано по 2 варианту

double u0(double x, double y) {

return 500. + exp(x + y);

}

double mu1(double y, double t) {

return 500. + exp(y) + t \* (200. \* y + 100.);

}

double mu2(double y, double t) {

return 500. + exp(1 + y) + 300. \* t / (y + 1.);

}

double mu3(double x, double t) {

return 500. + exp(x) + 100. \* t \* (2. - cos(PI \* x));

}

double mu4(double x, double t) {

return 500. + exp(x + 1) + t \* (300. - 150. \* x);

}

double c(double u) {

return 1. / (2.25e-3 - 6.08e-10 \* u \* u);

}

double rho(double u) {

return 7860. + 41500. / u;

}

double lambda(double u) {

return 1.45 + 2.3e-2 \* u - 2.e-6 \* u \* u;

}

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

{

int p, o, s = 0.0;

double\* u, \* u1;

int MyID, NumProc, ierror;

MPI\_Status status;

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

if (ierror != MPI\_SUCCESS)

{

printf("MPI initialization error!");

exit(1);

}

MPI\_Comm\_size(MPI\_COMM\_WORLD, &NumProc);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &MyID);

int\* rcounts, \* displs;

if (MyID == 0) { rcounts = (int\*)calloc(NumProc, sizeof(int)); displs = (int\*)calloc(NumProc, sizeof(int)); }

int I = atoi(argv[1]);

double T = atof(argv[2]);

double dx = L / I;

double tau = (0.9 \* dx \* dx) / (4.0 \* K);

p = (I - 2) / NumProc;

o = (I - 2) % NumProc;

if (MyID < o) { p++; }

else { s = o; }

u = new double[(p + 2) \* I];

if (MyID)

u1 = (double\*)calloc((p + 2) \* I, sizeof(double));

else

{

u1 = (double\*)calloc(I \* I, sizeof(double));

}

for (int j = MyID \* p + s; j < (MyID + 1) \* p + s + 2; j++)

{

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

{

u[i + (j - MyID \* p - s) \* I] = u0(i / (double)I, j / (double)I);

}

}

if (!MyID)

{

displs[0] = 0;

for (int i = 0; i < NumProc; i++)

{

if (i < o)

rcounts[i] = (p + 3) \* I;

else

rcounts[i] = (p + 2) \* I;

if (i < NumProc - 1)

displs[i + 1] = displs[i] + (rcounts[i] - 2 \* I);

}

}

MPI\_Gatherv(u, (p + 2) \* I, MPI\_DOUBLE, u1, rcounts, displs, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

if (MyID == 0)

{

ofstream out("u0.txt");

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

{

for (int j = 0; j < I; j++)

out << i \* dx << “\t” << j \* dx << “\t” << u1[i \* I + j] << "\n";

out << endl;

}

out.close();

}

double tstart = MPI\_Wtime();

for (double t = tau; t < T; t += tau)

{

for (int j = 1; j < p + 1; j++)

for (int i = 1; i < I - 1; i++)

{

int idx = i + j \* I;

double lambda1 = lambda((u[idx + 1] + u[idx]) / 2.);

double lambda2 = lambda((u[i - 1 + j \* I] + u[idx]) / 2.);

double lambda3 = lambda((u[i + (j + 1) \* I] + u[idx]) / 2.);

double lambda4 = lambda((u[i + (j - 1) \* I] + u[idx]) / 2.);

double cc = c(u[idx]);

double roc = rho(u[idx]);

u1[idx] = u[idx] + tau / (cc \* roc) \*

(lambda1 \* (u[idx + 1] - u[idx]) / (dx \* dx)

- lambda2 \* (u[idx] - u[idx - 1]) / (dx \* dx)

+ lambda3 \* (u[idx + I] - u[idx]) / (dx \* dx)

- lambda4 \* (u[idx] - u[idx - I]) / (dx \* dx));

}

if (MyID)

MPI\_Sendrecv(&u1[I + 1], I - 2, MPI\_DOUBLE, MyID - 1, 1, &u1[1], I - 2, MPI\_DOUBLE, MyID - 1, 1, MPI\_COMM\_WORLD, &status);

if (NumProc - MyID - 1)

MPI\_Sendrecv(&u1[p \* I + 1], I - 2, MPI\_DOUBLE, MyID + 1, 1, &u1[(p + 1) \* I + 1], I - 2, MPI\_DOUBLE, MyID + 1, 1, MPI\_COMM\_WORLD, &status);

if (!MyID)

for (int i = 1; i < I - 1; i++)

{

u1[i] = mu3(i / (double)I, t / T);

}

if (!(NumProc - MyID - 1))

for (int i = 1; i < I - 1; i++)

{

u1[(p + 1) \* I + i] = mu4(i / (double)I, t / T);

}

for (int j = 0; j < p + 2; j++)

{

u1[j \* I] = mu1((j + MyID \* p + s) / (double)I, t / T);

u1[(j + 1) \* I - 1] = mu2((j + MyID \* p + s) / (double)I, t / T);

}

for (int j = 0; j < p + 2; j++)

{

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

{

u[i + j \* I] = u1[i + j \* I];

}

}

}

MPI\_Gatherv(u, (p + 2) \* I, MPI\_DOUBLE, u1, rcounts, displs, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

if (MyID == 0)

{

double time = MPI\_Wtime() - tstart;

cout << "Time: " << time << endl;

ofstream out("u1.txt");

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

{

for (int j = 0; j < I; j++)

out << i \* dx << “\t” << j \* dx << “\t” << u1[i \* I + j] << "\n";

}

out.close();

}

MPI\_Finalize();

return 0;

}