Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра нелинейных динамических систем и процессов управления

Орлова Анна Олеговна, 614 группа

Вариант 2

Отчёт по заданию в рамках курса

«Суперкомпьютерное моделирование и технологии»

Численное решение краевой задачи для уравнения Пуассона с потенциалом в прямоугольной области

Москва
2022

Содержание

1 Постановка задачи

В прямоугольнике $\Pi = [A_1, A_2] \times [B_1, B_2]$, граница Γ которого состоит из отрезков

$$\gamma_R = \{(A_2, y), B_1 \leqslant y \leqslant B_2\}, \quad \gamma_L = \{(A_1, y), B_1 \leqslant y \leqslant B_2\},$$
$$\gamma_T = \{(x, B_2), A_1 \leqslant x \leqslant A_2\}, \quad \gamma_B = \{(x, B_1), A_1 \leqslant x \leqslant A_2\},$$

рассматривается дифференциальное уравнение Пуассона с потенциалом

$$-\Delta u + q(x, y)u = F(x, y), \tag{1}$$

в котором оператор Лапласа

$$\Delta u = \frac{\partial}{\partial x}\Big(k(x, y)\frac{\partial u}{\partial x}\Big) + \frac{\partial}{\partial y}\Big(k(x, y)\frac{\partial u}{\partial y}\Big).$$

Для выделения единственного решения уравнение (1) дополняется граничными условиями. На каждом отрезке границы прямоугольника Π задаются следующие условия:

$$\gamma_R : u(x, y) = \varphi(x, y), \quad \gamma_L : u(x, y) = \varphi(x, y),$$
$$\gamma_T : u(x, y) = \varphi(x, y), \quad \gamma_B : u(x, y) = \varphi(x, y).$$

Функции $F(x, y)$, $\varphi(x, y)$, коэффициент $k(x, y)$, потенциал $q(x, y)$ считаются известными. Требуется найти функцию $u(x, y)$, удовлетворяющую уравнению (1) и граничным условиям.

В соответствии с вариантом задания рассматриваю следующие данные:

- $A_1 = 0$, $A_2 = 4$, $B_1 = 0$, $B_2 = 3$,
- $u(x, y) = u_2(x, y) = \sqrt{4 + xy}$,
- $k(x, y) = k_3(x, y) = 4 + x + y$,
- $q(x, y) = q_2(x, y) = x + y$,
- граничные условия γ_R (1 тип), γ_L (1 тип), γ_T (1 тип), γ_B (3 тип).

Задача. *Задача практикума заключается в восстановлении известной гладкой функции $u(x, y)$ по её образу $F(x, y) = -\Delta u + q(x, y)u$ и её граничным значениям.*
Нужно восстановить следующий вид функции $u(x, y)$:

$$u(x, y) = \sqrt{4 + xy}$$

2 Определение функций $F(x, y)$, $\varphi(x, y)$

Определим функцию $F(x, y)$. Для этого вычислим оператор Лапласа, используя явные вид функций $u(xy)$, $k(x, y)$:

$$\frac{\partial u}{\partial x} = \frac{y}{2\sqrt{4+xy}}$$

$$\frac{\partial u}{\partial y} = \frac{x}{2\sqrt{4+xy}}$$

$$\Delta u = \frac{\partial}{\partial x}\left(k(x,y)\frac{y}{2\sqrt{4+xy}}\right) + \frac{\partial}{\partial y}\left(k(x,y)\frac{x}{2\sqrt{4+xy}}\right) =$$
$$= \frac{\partial}{\partial x}\left(\frac{(4+x+y)y}{2\sqrt{4+xy}}\right) + \frac{\partial}{\partial y}\left(\frac{(4+x+y)x}{2\sqrt{4+xy}}\right) =$$
$$= \frac{x}{2\sqrt{4+xy}} + \frac{y}{2\sqrt{4+xy}} - \frac{x^2(x+y+4)}{4(xy+4)^{\frac{3}{2}}} - \frac{y^2(x+y+4)}{4(xy+4)^{\frac{3}{2}}}.$$

Учитывая, что $q(x, y)u = (x + y)\sqrt{4 + xy}$, то

$$F(x,y) = (x+y)\sqrt{4+xy} - \frac{x+y}{2\sqrt{4+xy}} + \frac{(x^2+y^2)(x+y+4)}{4(xy+4)^{\frac{3}{2}}}. \tag{2}$$

Теперь определим функцию $\varphi(x, y)$.

$$\varphi(x,y) = \begin{cases} 2\sqrt{1+y}, & x = 4, \widetilde{y} \in [0,3]; \\ 2, & x = 0, \widetilde{y} \in [0,3]; \\ \sqrt{4+3x}, & \widetilde{x} \in [0,4], y = 3; \\ \frac{-x}{4} + 2, & \widetilde{x} \in [0,4], y = 0. \end{cases} \tag{3}$$

3 Разностная схема решения задачи

Краевые задачи для уравнения Пуассона с потенциалом (1) предлагается численно решать методом конечных разностей. В расчетной области Π определяется равномерная прямоугольная сетка $\bar{\omega}_h = \bar{\omega}_1 \times \bar{\omega}_2$, где

$$\bar{\omega}_1 = \{x_i = ih_1,\ i = \overline{0, M}\},\ \bar{\omega}_2 = \{y_j = jh_2,\ j = \overline{0, N}\}.$$

Здесь $h_1 = 4/M$, $h_2 = 3/N$. Через ω_h обозначим множество внутренних узлов сетки $\bar{\omega}_h$, т.е. множество узлов сетки прямоугольника, не лежащих на границе Γ.

Рассмотрим линейное пространство H функций, заданных на сетке $\bar{\omega}_h$. Обозначим через w_{ij} значение сеточной функции $w \in H$ в узле сетки $(x_i, y_j) \in \bar{\omega}_h$. Будем считать, что в пространстве H задано скалярное произведение и евклидова норма

$$[u, v] = \sum_{i=0}^{M} h_1 \sum_{j=0}^{N} h_2 \rho_{ij} u_{ij} v_{ij} = h_1 h_2 \sum_{i=0}^{M} \sum_{j=0}^{N} \rho_{ij} u_{ij} v_{ij}, \quad \|u\|_E = \sqrt{[u, u]}. \tag{4}$$

Весовая функция $\rho_{ij} = \rho^{(1)}(x_i)\rho^{(2)}(y_j)$, где

$$\rho^{(1)}(x_i) = \left[\begin{array}{rl} 1, & 1 \leqslant i \leqslant M-1 \\ 1/2, & i = 0,\ i = M \end{array}\right. \qquad \rho^{(2)}(y_j) = \left[\begin{array}{rl} 1, & 1 \leqslant j \leqslant N-1 \\ 1/2, & j = 0,\ j = N \end{array}\right.$$

В методе конечных разностей дифференциальная задача математической физики заменяется конечно-разностной операторной задачей вида

$$Aw \; = \; B, \tag{5}$$

где $A : H \rightarrow H$ – оператор, действующий в пространстве сеточных функций, $B \in H$ – известная правая часть. Задача (5) называется разностной схемой. Решение этой задачи считается численным решением исходной дифференциальной задачи.

При построении разностной схемы следует аппроксимировать все уравнения краевой задачи их разностными аналогами – сеточными уравнениями, связывающими значения искомой сеточной функции в узлах сетки. Полученные таким образом уравнения должны быть функционально независимыми, а их общее количество – совпадать с числом неизвестных, т.е. с количеством узлов сетки.

Уравнение (1) во всех внутренних точках сетки аппроксимируется разностным уравнением

$$-\Delta_h w_{ij} + q_{ij} w_{ij} = F_{ij}, \quad i = \overline{1, M-1},\ j = \overline{1, N-1}, \tag{6}$$

в котором $F_{ij} = F(x_i, y_j)$, $q_{ij} = q(x_i, y_j)$, разностный оператор Лапласа

$$\begin{aligned} \Delta_h w_{ij} = \frac{1}{h_1}\Big(k(x_i + 0.5h_1, y_j)\frac{w_{i+1j} - w_{ij}}{h_1} - k(x_i - 0.5h_1, y_j)\frac{w_{ij} - w_{i-1j}}{h_1}\Big) + \\ + \frac{1}{h_2}\Big(k(x_i, y_j + 0.5h_2)\frac{w_{ij+1} - w_{ij}}{h_2} - k(x_i, y_j - 0.5h_2)\frac{w_{ij} - w_{ij-1}}{h_2}\Big). \end{aligned} \tag{7}$$

Введем обозначения правой и левой разностных производных по переменным x, y соответственно:

$$\begin{aligned} w_{x,ij} = \frac{w_{i+1j} - w_{ij}}{h_1}, \quad w_{\overline{x},ij} = w_{x,i-1j} = \frac{w_{ij} - w_{i-1j}}{h_1}, \\ w_{y,ij} = \frac{w_{ij+1} - w_{ij}}{h_2}, \quad w_{\overline{y},ij} = w_{y,ij-1} = \frac{w_{ij} - w_{ij-1}}{h_2}, \end{aligned} \tag{8}$$

а также определим сеточные коэффициенты

$$a_{ij} = k(x_i - 0.5h_1, y_j), \quad b_{ij} = k(x_i, y_j - 0.5h_2). \tag{9}$$

С учетом принятых обозначений (8) - (9) разностный оператор Лапласа можно представить в более компактном и удобном виде

$$\Delta_h w_{ij} = \big(a w_{\overline{x}}\big)_{x,ij} + \big(b w_{\overline{y}}\big)_{y,ij}\ . \tag{10}$$

В самом деле, из (7) с учётом (8) - (9) получаем

$$\begin{aligned} \Delta_h w_{ij} = \frac{1}{h_1}(a_{i+1j} w_{x,ij} - a_{ij} w_{\overline{x},ij}) + \frac{1}{h_2}(b_{ij+i} w_{y,ij} - b_{ij} w_{\overline{y},ij}) = \\ = \frac{1}{h_1}(a_{i+1j} w_{\overline{x},i+1j} - a_{ij} w_{\overline{x},ij}) + \frac{1}{h_2}(b_{ij+i} w_{\overline{y},ij+1} - b_{ij} w_{\overline{y},ij}) = \\ = \big(a w_{\overline{x}}\big)_{x,ij} + \big(b w_{\overline{y}}\big)_{y,ij}\ . \end{aligned} \tag{11}$$

Аппроксимация граничных условий для задачи описанной вариантом имеет вид:

$$w_{ij} = \varphi(x_i, y_j). \tag{12}$$

Переменные w_{ij}, заданные равенством (12), исключаются из разностной схемы, а соответствующие узлы $P_{ij}(x_i, y_j)$ – из расчетной сетки $\overline{\omega}_h$. В скалярном произведении (4) слагаемые, отвечающие данным граничным узлам, считаются равными нулю.

Замечание. Разностные схемы (5), аппроксимирующие все описанные выше краевые задачи для уравнения Пуассона с положительным потенциалом, обладают самосопряженным и положительно определенным оператором A и имеют единственное решение при любой правой части.

Соберём все уравнения, получим систему линейных алгебраических уравнений (СЛАУ), состоящую из $(M-1)\times(N-1)$ уравнений и $(M-1)\times(N-1)$ неизвестной:

$$\begin{cases}
-\Delta_h w_{ij} + q_{ij}w_{ij} = F_{ij}, & i=\overline{2,M-2},\ j=\overline{1,N-2}\\
-(aw_{\bar{x}})_{i,1} - \frac{1}{h_2}\left[(bw_{\bar{y}})_{i,0} - \frac{1}{h_2}b_{i,1}w_{i,1}\right] + q_{i,0}w_{i,0} = F_{i,0} + \frac{2}{h_2^2}\varphi_{i,0}, & i=\overline{1,M-1},\ j=0\\
-(aw_{\bar{x}})_{i,N-1} + \frac{1}{h_2}\left[(bw_{\bar{y}})_{i,N-1} + \frac{1}{h_2}b_{i,N}w_{i,N-1}\right] + q_{i,N-1}w_{i,N-1} = & \\
\qquad\qquad = F_{i,N-1} + \frac{1}{h_2^2}b_{i,N-1}\varphi_{i,N-1}, & i=\overline{2,M-2},\ j=N-1\\
(bw_{\bar{y}})_{1,j} - \frac{1}{h_1}\left[(aw_{\bar{x}})_2 j - \frac{1}{h_1}a_{1j}w_{1j}\right] + q_{1,j}w_{1,j} = F_{1,j} + \frac{1}{h_1^2}a_{1j}\varphi_{0j}, & i=1,\ j=\overline{1,N-2}\\
-(bw_{\bar{y}})_{M-1,j} + \frac{1}{h_1}\left[(aw_{\bar{x}})_{M-1,j} - \frac{1}{h_1}a_{M,j}w_{M-1,j}\right] + & \\
\qquad\qquad +q_{M-1,j}w_{M-1,j} = F_{M-1,j} + \frac{1}{h_1^2}a_{M-1,j}\varphi_{M,j}, & i=M-1,\ j=\overline{1,N-2}\\
-\frac{1}{h_1}\left[(aw_{\bar{x}})_{2,0} - \frac{1}{h_1}a_{1,0}w_{1,0}\right] - \frac{2}{h_2}\left[(bw_{\bar{y}})_{1,2} - \frac{1}{h_2}w_{1,0}\right] + q_{1,0}w_{1,0} & \\
\qquad\qquad = F_{1,0} + \frac{1}{h_1^2}a_{1,0}\varphi_{0,0} + \frac{2}{h_2^2}\varphi_{1,0}, & i=1,\ j=0\\
-\frac{1}{h_1}\left[(aw_{\bar{x}})_{2,N-1} - \frac{1}{h_1}a_{1,N-1}w_{1,N-1}\right] + & \\
\quad +\frac{1}{h_2}\left[(bw_{\bar{y}})_{1,N-1} + \frac{1}{h_2}b_{1N}w_{1,N-1}\right] + q_{1,N-1}w_{1,N-1} = & \\
\qquad\qquad = F_{1,N-1} + \frac{1}{h_1^2}a_{1,N-1}\varphi_{0,N-1} + \frac{1}{h_2^2}b_{1N}\varphi_{1,N-1}, & i=1,\ j=N-1\\
\frac{1}{h_1}\left[(aw_{\bar{x}})_{M-2,0} + \frac{1}{h_1}a_{M-1,0}w_{M-1,0}\right] - & \\
\quad -\frac{1}{h_2}\left[(bw_{\bar{y}})_{M-1,1} - \frac{1}{h_2}b_{M-1,0}w_{M-1,1}\right] + q_{M-1,1}w_{M-1,1} = & \\
\qquad\qquad = F_{M-1,0} + \frac{1}{h_1^2}a_{M-1,0}\varphi_{M,1} + \frac{2}{h_2^2}\varphi_{M-1,0}, & i=M-1,\ j=0\\
\frac{1}{h_1}\left[(aw_{\bar{x}})_{M-2,N-1} + \frac{1}{h_1}a_{M-1,N-1}w_{M-1,N-1}\right] + & \\
+\frac{1}{h_2}\left[(bw_{\bar{y}})_{M-1,N-2} + \frac{1}{h_2}b_{M-1,N-1}w_{M-1,N-1}\right] + q_{M-1,N-1}w_{M-1,N-1} = & \\
\qquad\qquad = F_{M-1,N-1} + \frac{1}{h_1^2}a_{M-1,N-1}\varphi_{M,N} + \frac{1}{h_2^2}b_{M-1,N-1}\varphi_{M,N}, & i=M-1,\ j=N-1
\end{cases} \tag{13}$$

Дальнейшая наша задача решить систему уравнений (13).

4 Метод решения СЛАУ.

Приближенное решение системы уравнений (5) для сформулированных выше краевых задач может быть получено итерационным методом наименьших невязок. Этот метод позволяет получить последовательность сеточных функций $w^{(k)} \in H$, $k = 1, 2, \ldots$, сходящуюся по норме пространства H к решению разностной схемы, т.е.

$$\|w - w^{(k)}\|_E \to 0, \quad k \to +\infty.$$

Начальное приближение $w^{(0)}$ можно выбрать любым способом, например, равным нулю во всех точках расчетной сетки.

Метод является одношаговым. Итерация $w^{(k+1)}$ вычисляется по итерации $w^{(k)}$ согласно равенствам:

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} - \tau_{k+1} r_{ij}^{(k)}, \tag{14}$$

где невязка $r^{(k)} = Aw^{(k)} - B$, итерационный параметр

$$\tau_{k+1} = \frac{\left[Ar^{(k)}, r^{(k)}\right]}{\left\|Ar^{(k)}\right\|_E^2}.$$

В качестве условия остановки итерационного процесса возьмем неравенство

$$\|w^{(k+1)} - w^{(k)}\|_E < \varepsilon,$$

где ε – положительное число, определяющее точность итерационного метода. Константу ε для данной задачи возьмем равной 10^{-6}.

5 Создание MPI программы.

Описание основных идей программы.

1. В программе были реализованы функции q, k, F, u – согласно описанному варианту. Эти функции участвуют в уравнениях из системы (13).

2. Реализована функция умножения матрицы A на вектор w. Сетка w и Aw хранятся в виде векторов.

3. Реализована функция заполнения матрицы B.

4. В цикле выполняли вычисления вектора r как разность Aw и B, расчет τ, затем новое значение вектора w. Цикл повторялся до тех пор, пока норма разности вектора w на текущей и прошлой итерациях не становилась меньше заданной точности.

5. Для реализации MPI программы использовались стандартные функции библиотеки MPI.

6. Происходит деление разностной схемы на подрешетки. На каждой подрешетке действует свой процесс. Все процессы выполняют один код, но у каждого будут свои значения в вычисляемой области. Для разделения решетки на подрешетки требовались следующий действия:

   (a) Определить количество процессоров и найти, какой степени двойки оно соответствует.

   (b) Найти два наибольших числа, дающих в сумме найденную степень (если степень четная, то числа будут одинаковыми – половиной степени; иначе будут отличаться на единицу). Полученные числа дают размер сетки независимых процессоров: наибольшее из двух чисел – количество строк, наименьшее – количество столбцов.

   (c) Далее вычисляется, какое количество точек в строке и в столбце получит каждый процессор.

7. Каждый процессор вычисляет положенный ему прямоугольник расчетной области.

8. После выхода из параллельной секции матрицы с результатами суммируются и результат рассылается по всем процессам (операция Allreduce)

Ниже приведены таблицы с результатами расчетов на ПВС IBM Polus.

Точность 10^{-6}.

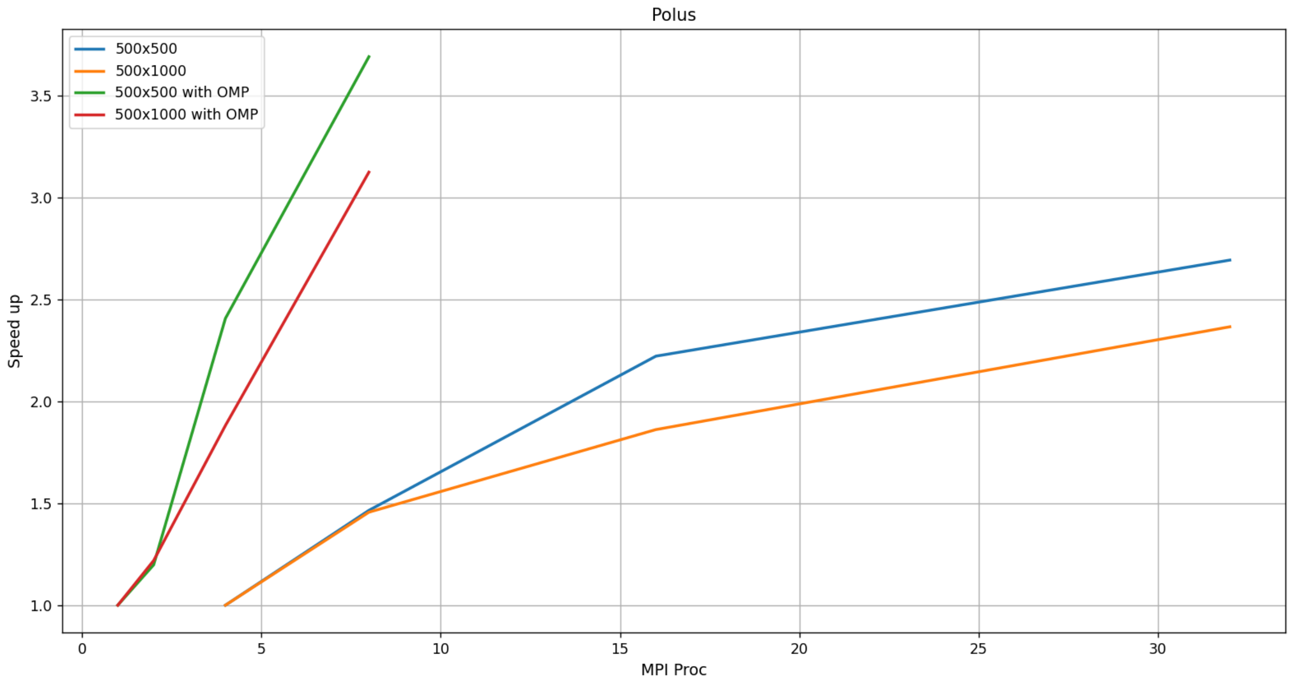
OpenMP порождает 4 нити.

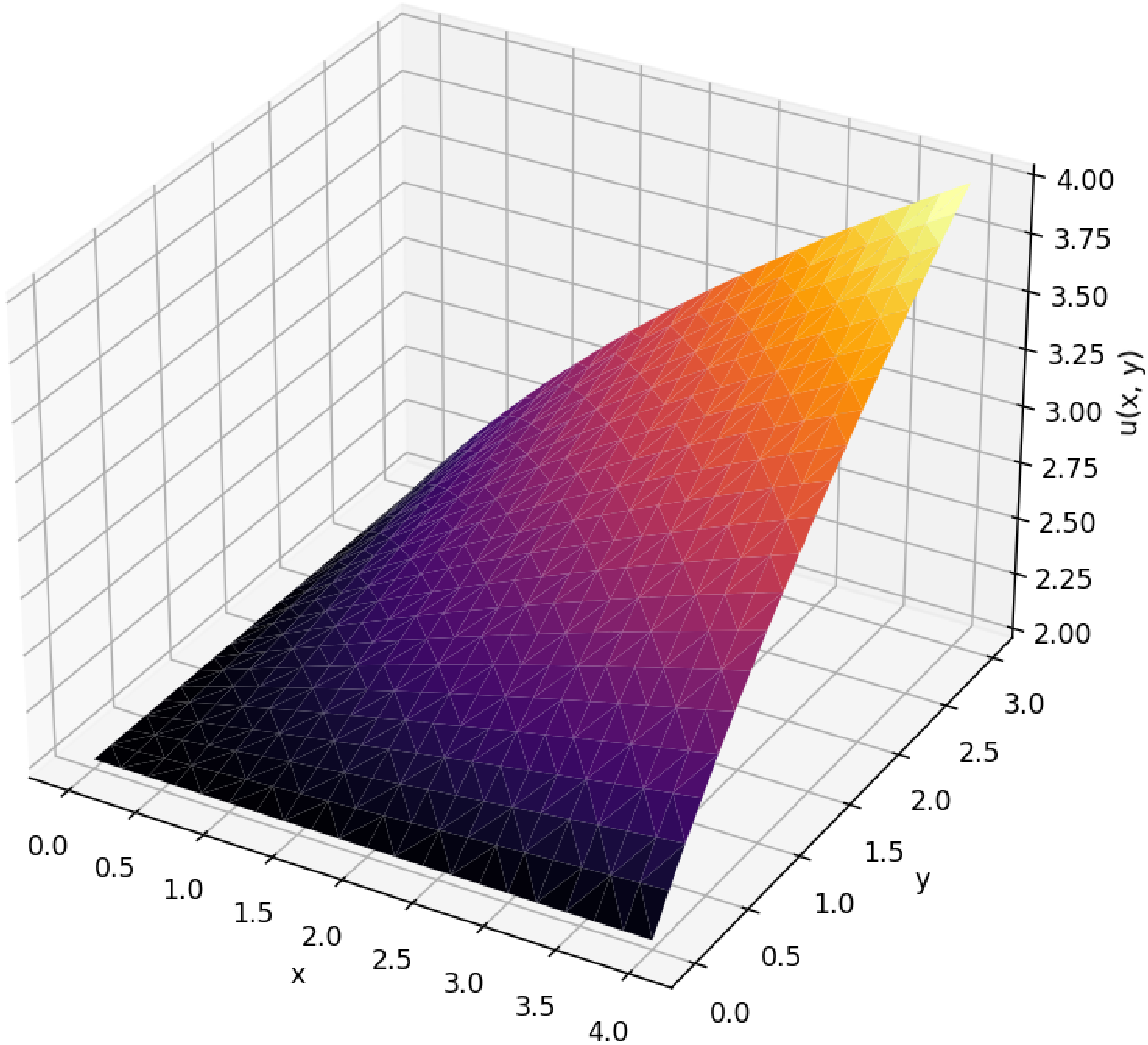
Table 1: Результаты расчетов на ПВС IBM Polus реализация MPI+OpenMP

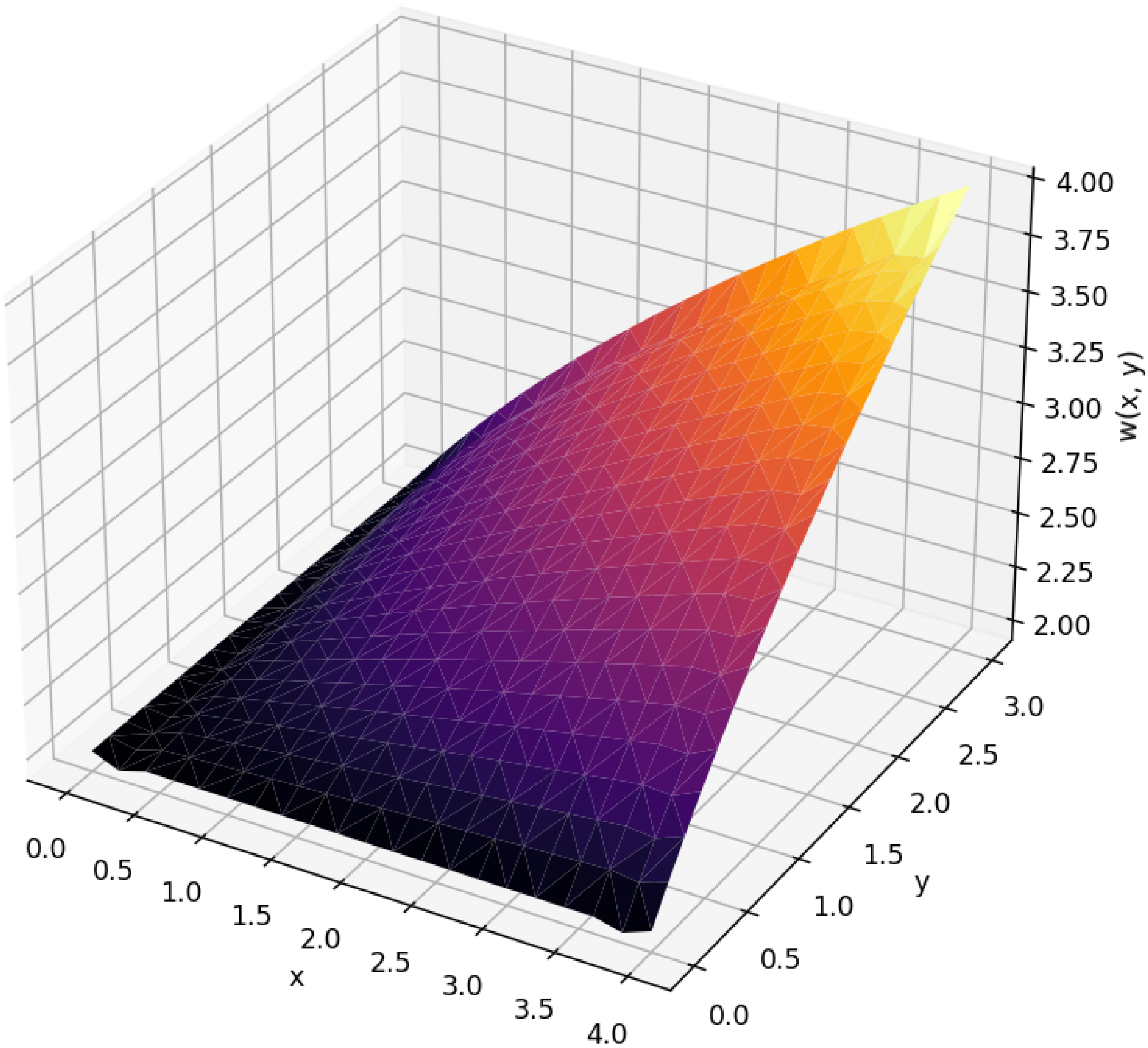
Число проц-ов N_p	Число точек сетки	Время решения, с. T	Ускорение S
1	500×500	82.708302	1.0
2	500×500	69.038390	1.198
4	500×500	34.369342	2.406
8	500×500	22.419090	3.689
1	500×1000	138.018794	1.0
2	500×1000	113.268240	1.219
4	500×1000	73.384320	1.881
8	500×1000	44.187978	3.123

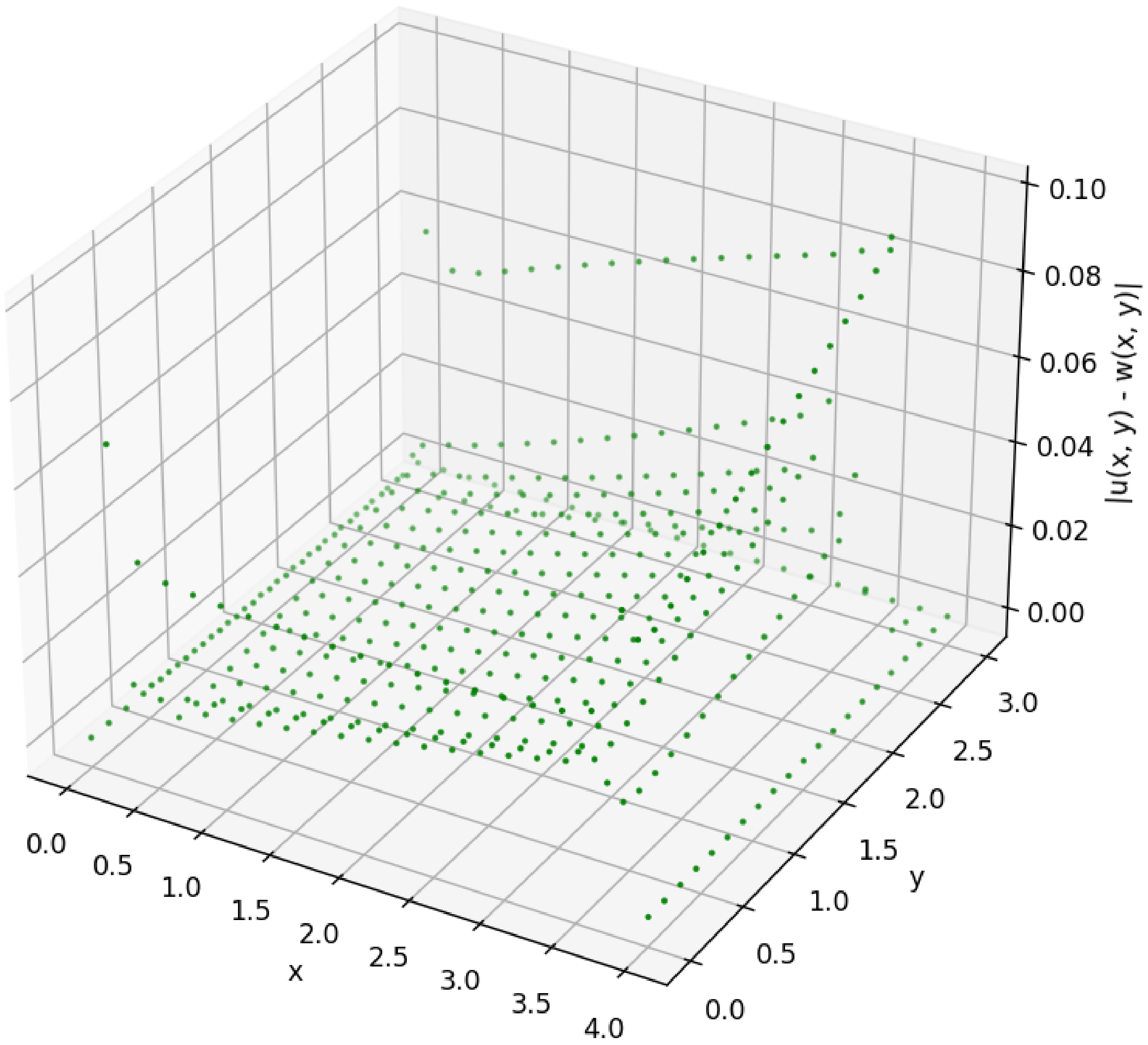
Table 2: Таблица с результатами расчетов на ПВС IBM Polus MPI

Число проц-ов N_p	Число точек сетки	Время решения, с. T	Ускорение S
4	500×500	88.372147	1.0
8	500×500	60.302955	1.465
16	500×500	39.799426	2.221
32	500×500	32.830875	2.692
4	500×1000	150.601277	1.0
8	500×1000	103.462851	1.456
16	500×1000	80.674803	1.861
32	500×1000	63.674806	2.365




x
y
u(x, y)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00


4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
w(x, y)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x
0.0
0.5
1.0
1.5
2.0
2.5
3.0
y


|u(x, y) - w(x, y)|
0.00
0.02
0.04
0.06
0.08
0.10
x
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0

6 Приложение 1.

Реализация гибридная MPI/OpenMP. Для кода OpenMP убрать соответствующие вставки.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>
#include <omp.h>

#define M 500
#define N 1000

double u(double x, double y)
{
    return sqrt(4 + x*y);
}

double q(double x, double y)
{
        double t = x + y;
    return t >= 0 ? t : 0;
}

double k(double x, double y){
    return 4 + x + y;
}

/** phi_1(x) = u(4, y), R*/
double phi_1(double x, double y)
{
    return 2.0 * sqrt(1 + y);
}

/** phi_2(y) = u(0, y), L*/
double phi_2(double x, double y)
{
    return 2.0;
}

/** phi_3(x) = u(x, 3), T*/
double phi_3(double x, double y)
{
    return sqrt(4 + x*3);
}

/** phi_4(x) = u/dn(x, 3) + u, B*/
double phi_4(double x, double y)
{
    return -x/4 + 2;
}

double F(double x, double y)
```

```
{
    return -(x + y)/(2*sqrt(x*y + 4)) + (y*y + x*x)*k(x, y)/(4*pow(x*y + 4, 1.5)) +
}

int getColumns(int degree, int rank, int k)
{
    int numCol = 1;
    int columns[3];

    for(int i = 0; i < degree/2; i++)
    {
        numCol = numCol*2;
    }

    columns[0] = (N + 1) % numCol;
    columns[1] = (N + 1) / numCol;

    if ((rank + 1)%numCol)
    {
        columns[2] = (rank + 1) % numCol - 1;
    }
    else
    {
        columns[2] = numCol - 1;
    }

    if (k == 0)
        return columns[0];
    else if (k == 1)
        return columns[1];
    if (k == 2)
        return columns[2];
    return -1;
}

int getRows(int degree, int rank, int k)
{
    int numRow = 1, numCol = 1;
    int rows[3];

    for(int i = 0; i < degree/2; i++)
    {
        numCol = numCol * 2;
    }

    for(int i = 0; i < degree - degree/2; i++)
    {
        numRow = numRow * 2;
    }

    rows[0] = (M + 1) % numRow;
    rows[1] = (M + 1) / numRow;
```

```
    if ((rank + 1) % numCol)
    {
        rows[2] = (rank + 1) / numCol;
    }
    else
    {
        rows[2] = (rank + 1) / numCol - 1;
    }

    if (k == 0)
        return rows[0];
    else if (k == 1)
        return rows[1];
    if (k == 2)
        return rows[2];
    return -1;
}

double ro(int i, int j)
{
    if ((i == 0 && j == 0) || (i == M && j == 0) || (i == 0 && j == N))
        return 0.25;
    if (j == 0)
        return 0.5;
    return 1.0;
}

double iterPar(double u[], double v[], int i, int j, double h1, double h2)
{
    double res = 0.0;

    if ((i == 0 && j == 0) || (i == M && j == 0) || (i == 0 && j == N))
    {
        res = ro(i, j) * ro(i, j) * u[i * (N + 1) + j] * v[i * (N + 1) + j];
    }
    else if(j == 0)
    {
        res = ro(i, j) * ro(i, j) * u[i * (N + 1) + j] * v[i * (N + 1) + j];
    }
    else if(i >= 1 && i <= M - 1 && j >= 1 && j <= N - 1)
    {
        res = ro(i, j) * ro(i, j) * u[i * (N + 1) + j] * v[i * (N + 1) + j];
    }

    return h1 * h2 * res;
}

double getB(int i, int j, double h1, double h2)
{
    double B = 0.0;
    double x = h1 * i, xl = h1 * (i - 1), xr = h1 * (i + 1);
    double y = h2 * j, yt = h2 * (j + 1), yb = h2 * (j - 1);
```

```
    if(i >= 2 && i <= M - 2 && j >= 1 && j <= N - 2)    //internal
    {
        B = F(x, y);
    }
    else if(i == 1 && j >= 1 && j <= N - 2)              //left side a1b1 -> a1b2
    {
        B = F(x, y) + 2.0/(h1) * phi_2(xl, y);
        //B = F(x, y) + 1.0/(h1*h1) * k(x - 0.5 * h1, y) * phi_2(xl, y);
    }
    else if(i == M - 1 && j >= 1 && j <= N - 2)          //right side a2b1 -> a2b2
    {
        B = F(x, y) + 2.0/(h1) * phi_1(xr, y);
        //B = F(x, y) + 1.0/(h1*h1) * k(x + 0.5 * h1, y) * phi_1(xr, y);
    }
    else if(i >= 2 && i <= M - 2 && j == N - 1)          //top side a1b2 -> a2b2
    {
        B = F(x, y) + 2.0/(h2) * phi_3(x, yt);//
        //B = F(x, y) + 1.0/(h2*h2) * k(x, y + 0.5 * h2) * phi_3(x, yt);//
    }
    else if(i >= 2 && i <= M - 2 && j == 0)              //bottom side a1b1 -> a2b1
    {
        B = F(x, y) + (2.0/h2) * phi_4(x, y);
        //B = F(x, y) + (1.0/h2*h2) *  k(x - 0.5 * h1, y) * phi_4(x, y);
    }
    else if(i == 1 && j == 0)                            //point 1, 0
    {
        B = F(x, y) + 2.0/h2 * phi_4(xl, y) + 2.0/h1 * phi_2(xl, y);
    }
    else if(i == 1 && j == N - 1)                              //point 1, N-1
    {
        B = F(x, y) + 2.0/h2 * phi_3(xl, y) + 2.0/h1 * phi_2(xl, y);
    }
    else if(i == M - 1 && j == 0)                              //point M-1, 0
    {
        B = F(x, y) + 2.0/h2 * phi_4(xr, y) + 2.0/h1 * phi_1(xr, y);
    }
    else if(i == M - 1 && j == N - 1)                          //point M-1, N-1
    {
        B = F(x, y) + 2.0/h1 * phi_1(xr, yt) + 2.0/h2 * phi_3(xr, yt);
    }

    return B;

}

double getAw(double w[], int i, int j, double h1, double h2)
{
    double aw;
    double x = h1 * i, xl = h1 * (i - 1), xr = h1 * (i + 1);
    double y = h2 * j, yt = h2 * (j + 1), yb = h2 * (j - 1);

    if(i >= 2 && i <= M - 2 && j >= 1 && j <= N - 2)    //internal
    {
```

```
        aw = 1.0/(h1*h1) * (k(x + 0.5 * h1, y) * (w[(i + 1) * (N + 1) + j] - w[i *
        - k(x - 0.5 * h1, y) * (w[i * (N + 1) + j] - w[(i - 1) * (N + 1) + j]))
        + 1.0/(h2*h2) * (k(x, y + 0.5 * h2) * (w[i * (N + 1) + j + 1] - w[i * (N +
        + k(x, y - 0.5 * h2) * (w[i * (N + 1) + j] - w[i * (N + 1) + j - 1]))
        + q(x, y) * w[i * (N + 1) + j];
    }
    else if(i == 1 && j >= 1 && j <= N - 2)           //left side a1b1 -> a1b2
    {
        aw = -1.0/(h1*h1) * (k(xr - 0.5 * h1, y) * (w[(i + 1) * (N + 1) + j] - w[i
        - 1.0/(h1*h1) * k(x - 0.5 * h1, y) * w[i * (N + 1) + j])
        - 1.0/(h2*h2) * (k(x, y + 0.5 * h2) * (w[i * (N + 1) + j + 1] - w[i * (N +
        - k(x, y - 0.5 * h2) * (w[i * (N + 1) + j] - w[i * (N + 1) + j - 1]))
        + q(x, y) * w[i * (N + 1) + j];
    }
    else if(i == M - 1 && j >= 1 && j <= N - 2)       //right side a2b1 -> a2b2
    {
        aw = 1.0/(h1*h1) * (k(xl + 0.5 * h1, y)) * (w[i * (N + 1) + j] - w[(i - 1)
        + 1.0/(h1*h1) * (k(x + 0.5 * h1, y)) * w[i * (N + 1) + j]
        - 1.0/(h2*h2) * (k(x, y + 0.5 * h2) * (w[i * (N + 1) + j + 1] - w[i * (N +
         - k(x, y - 0.5 * h2) * (w[i * (N + 1) + j] - w[i * (N + 1) + j - 1]))
         + q(x, y) * w[i * (N + 1) + j];
    }
    else if(i >= 2 && i <= M - 2 && j == N - 1)       //top side a1b2 -> a2b2 !!
    {
        aw = -1.0/(h1*h1) * (k(x + 0.5 * h1, y) * (w[(i + 1) * (N + 1) + j] - w[i *
        - k(x - 0.5 * h1, y) * (w[i * (N + 1) + j] - w[(i - 1) * (N + 1) + j]))
        + 1.0/(h2*h2) * (k(x, yb + 0.5 * h2) * (w[i * (N + 1) + j]
        - w[i * (N + 1) + j - 1]) + k(x, y + 0.5 * h2) * w[i * (N + 1) + j])
        + q(x, y) * w[i * (N + 1) + j];
    }
    else if(i == 1 && j == N - 1)                     //point a1b2 !! -1
    {
        aw = -1.0/(h1*h1) * k(xr - 0.5 * h1, y) * (w[(i + 1) * (N + 1) + j] - w[i *
        + 1.0/(h1*h1) * k(x - 0.5 * h1, y) * w[i * (N + 1) + j]
        + 1.0/(h2*h2) * (k(x, yb + 0.5 * h2) * (w[i * (N + 1) + j] - w[i * (N + 1)
        + k(x, y + 0.5 * h2) * w[i * (N + 1) + j])
        + q(x, y) * w[i * (N + 1) + j] ;
    }
    else if(i == M - 1 && j == N - 1)                 //point a2b2 !!
    {
        aw = 1.0/(h1*h1) * (k(xl + 0.5 * h1, y) * (w[i * (N + 1) + j] - w[(i - 1) *
        + k(x + 0.5 * h1, y) * w[i * (N + 1) + j])
        + 1.0/(h2*h2) * (k(x, yb + 0.5 * h2) * (w[i * (N + 1) + j] - w[i * (N + 1)
        + k(x, y + 0.5* h2) * w[i * (N + 1) + j])
        + q(x, y) * w[i * (N + 1) + j];
    }
    else if(i >= 2 && i <= M - 1 && j == 0)           //bottom side a1b1 -> a2b1
    {
        aw = -2.0/(h2*h2) * k(x, j + 0.5 * h2) * (w[i * (N + 1) + j + 1] - w[i * (N
        - 1.0/(h1*h1) * (k(x + 0.5 * h1, y) * (w[(i + 1) * (N + 1) + j] - w[i * (N
        - k(x - 0.5 * h1, y) * (w[i * (N + 1) + j] - w[(i - 1) * (N + 1) + j]))
         + (q(x, y) + 2/h2) * w[i * (N + 1) + j] ;
    }
```

```
    else if(i == 1 && j == 0)                                    //point a1b1
    {
        aw = -1.0/(h1*h1) * k(xr - 0.5 * h1, y) * (w[(i + 1) * (N + 1) + j] - w[i *
        + 1.0/(h1*h1) * k(x - 0.5 * h1, y) * w[i * (N + 1) + j] //(y_size + 2)
        - 2.0/(h2*h2) * k(x, yt - 0.5 * h2) * (w[i * (N + 1) + j + 1] - w[i * (N +
        + (q(x, y) + 2/h2) * w[i * (N + 1) + j];
    }
    else if(i == M - 1 && j == 0)                                //point a2b1
    {
        aw = 1.0/(h1*h1) * (k(xl + 0.5 * h1, y) * (w[i * (N + 1) + j] - w[(i - 1) *
        + k(x + 0.5 * h1, y) * w[i * (N + 1) + j])
        - 2.0/(h2*h2) * k(x, yt - 0.5 * h2) * (w[i * (N + 1) + j + 1] - w[i * (N +
        + (q(x, y) + (2/h2)) * w[i * (N + 1) + j] ;
    }
    else if (i == 0 || i == M || j == N)
    {
        aw = w[i * (N + 1) + j];
    }

    return aw;

}

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

    int iter = 0,
        rank,
        rank_omp = 4,
        size,
        root = 0,
        i, j;

    double epsilon = 1e-6,
           eps_local,
           eps_global = 0.0,
           tau,
           tau_local[2] = {0.0, 0.0},
           tau_global[2] = {0.0, 0.0},
           maxTime,
           time,
           start;

    double A1 = 0.0,
           A2 = 4.0,
           B1 = 0.0,
           B2 = 3.0,
           h1 = (A2 - A1) / M,
           h2 = (B2 - B1) / N;

    double w[(M + 1) * (N + 1)],
           r2[(M + 1) * (N + 1)],
           Ar2[(M + 1) * (N + 1)],
           wk[(M + 1) * (N + 1)],
```

```
            error[(M + 1) * (N + 1)],
            r[(M + 1) * (N + 1)],
            Ar[(M + 1) * (N + 1)];

    for(int i = 0; i <= M; i++)
        {
            for (int j = 0; j <= N; j++)
            {
                w[i * (N + 1) + j] = 0.0;
                wk[i * (N + 1) + j] = 0.0;
                r[i * (N + 1) + j] = 0.0;
                r2[i * (N + 1) + j] = 0.0;
                Ar[i * (N + 1) + j] = 0.0;
                Ar2[i * (N + 1) + j] = 0.0;
                error[i * (N + 1) + j] = 0.0;
            }
        }

    omp_set_num_threads(rank_omp);

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    start = MPI_Wtime();

    int degree = 0;
    int a = size;

    int x_size[2];
    int y_size[2];
    int coord[2];
    int procRow[2];
    int procCol[2];

    while (a != 1)
    {
        a /= 2;
        degree++;
    }

    procRow[0] = getRows(degree, rank, 0); procRow[1] = getRows(degree, rank, 1); c
    procCol[0] = getColumns(degree, rank, 0); procCol[1] = getColumns(degree, rank,

    if(((coord[0] + 1) <= procRow[0]) && ((coord[1] + 1) <= procCol[0]))
    {
        x_size[0] = coord[0] * (procRow[1] + 1);
        x_size[1] = (coord[0] + 1) * (procRow[1] + 1);
        y_size[0] = coord[1] * (procCol[1] + 1);
        y_size[1] = (coord[1] + 1) * (procCol[1] + 1);
    }
    else if (((coord[0] + 1) > procRow[0]) && ((coord[1] + 1 <= procCol[0])))
    {
        x_size[0] = procRow[0] + procRow[1] * coord[0];
```

```
        x_size[1] = procRow[0] * (procRow[1] + 1) + (coord[0] - procRow[0] + 1) * p
        y_size[0] = coord[1] * (procCol[1] + 1);
        y_size[1] = (coord[1] + 1) * (procCol[1] + 1);
    }
    else if (((coord[0] + 1) <= procRow[0]) && ((coord[1] + 1) > procCol[0]))
    {
        x_size[0] = coord[0] * (procRow[1] + 1);
        x_size[1] = (coord[0] + 1) * (procRow[1] + 1);
        y_size[0] = procCol[0] * (procCol[1] + 1) + (coord[1] - procCol[0]) * procC
        y_size[1] = procCol[0] * (procCol[1] + 1) + (coord[1] - procCol[0] + 1) * p
    }
    else
    {
        x_size[0] = procRow[0] + procRow[1] * coord[0]; //procRow[0] * (procRow[1]
        x_size[1] = procRow[0] * (procRow[1] + 1) + (coord[0] - procRow[0] + 1) * p
        y_size[0] = procCol[0] * (procCol[1] + 1) + (coord[1] - procCol[0]) * procC
        y_size[1] = procCol[0] * (procCol[1] + 1) + (coord[1] - procCol[0] + 1) * p
    }

    do{
        iter++;

        //r = A*w - B
        #pragma omp for firstprivate( i, j) shared(r2)
        for(int i = x_size[0]; i < x_size[1]; i++)
        {
            for (int j = y_size[0]; j < y_size[1]; j++)
            {
                r2[i * (N + 1) + j] = getAw(w, i, j, h1, h2) - getB(i, j, h1, h2);
            }
        }

        MPI_Allreduce(r2, r, (M + 1) * (N + 1), MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD

        //A*r
        #pragma omp for firstprivate(i, j) shared(Ar2)
        for(int i = x_size[0]; i < x_size[1]; i++)
        {
            for (int j = y_size[0]; j < y_size[1]; j++)
            {
                Ar2[i * (N + 1) + j] = getAw(r, i, j, h1, h2);
            }
        }

        MPI_Allreduce(Ar2, Ar, (M + 1) * (N + 1), MPI_DOUBLE, MPI_SUM, MPI_COMM_WOR

        tau_local[0] = 0.0;
        tau_local[1] = 0.0;

        //r_k+1
        #pragma omp for firstprivate(i, j) shared(tau_local)
        for(int i = x_size[0]; i < x_size[1]; i++)
        {
```

```
            for (int j = y_size[0]; j < y_size[1]; j++)
            {
                tau_local[0] += iterPar(Ar, r, i, j, h1, h2);
                tau_local[1] += iterPar(Ar, Ar, i, j, h1, h2);
            }
        }

        MPI_Allreduce(tau_local, tau_global, 2, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD

        tau = tau_global[0] / tau_global[1];

        #pragma omp for firstprivate(i, j) shared(Ar2, r2, w, wk, error)
        for(i = 0; i <= M; i++)
        {
            for(j = 0; j <= N; j++)
            {
                wk[i * (N + 1) + j] = (w[i * (N + 1) + j] - tau * r[i * (N + 1) + j
                if (i == 0 || i == M || j == N)
                {
                    error[i * (N + 1) + j] = 0.0;
                }
                else
                {
                    error[i * (N + 1) + j] = (wk[i * (N + 1) + j] - w[i * (N + 1) +
                }
                w[i * (N + 1) + j] = wk[i * (N + 1) + j];
                r2[i * (N + 1) + j] = 0;
                Ar2[i * (N + 1) + j] = 0;
            }
        }

        eps_local = 0.0;

        #pragma omp for firstprivate(i, j) shared(eps_local)
        for(int i = x_size[0]; i < x_size[1]; i++)
        {
            for (int j = y_size[0]; j < y_size[1]; j++)
            {
                eps_local += sqrt(iterPar(error, error, i, j, h1, h2));
            }
        }
        MPI_Allreduce(&eps_local, &eps_global, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WOR

    } while(eps_global > epsilon);


    time = MPI_Wtime() - start;
    MPI_Reduce(&time, &maxTime, 1, MPI_DOUBLE, MPI_MAX, root, MPI_COMM_WORLD);
    MPI_Finalize();

    if(rank == root)
    {
        printf("Iterations: %d, time: %f seconds\n", iter, maxTime);
```

```
        printf("Number of processes: %d, epsilon: %f, M: %d, N: %d\n", size, epsilon
    }

    return 0;
}
```