Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра исследования операций

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

Третье задание по курсу «Суперкомпьютерное моделирование и технологии»

Выполнил:
студент 614 группы
Грузицкий М.А.
Вариант 10

Москва 2021

Содержание

1 Постановка задачи 3
2 Определение функций $F(x, y)$, $\psi(x, y)$ 5
3 Разностная схема решения задачи 6
4 Метод решения СЛАУ 10
5 Результаты последовательной работы программы 11
6 Результаты параллельной работы программы 13
Список использованной литературы 17
Приложение А 18
Приложение Б 24

1 Постановка задачи

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

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

рассматривается дифференциальное уравнение Пуассона с потенциалом

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

в котором оператор Лапласа

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

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

- граница γ_R: $\psi(x,y) = \left(k\frac{\partial u}{\partial n}\right)(x,y) + u(x,y)$;
- граница γ_L: $\psi(x,y) = \left(k\frac{\partial u}{\partial n}\right)(x,y)$;
- граница γ_T: $\psi(x,y) = \left(k\frac{\partial u}{\partial n}\right)(x,y) + u(x,y)$;
- граница γ_B: $\psi(x,y) = \left(k\frac{\partial u}{\partial n}\right)(x,y)$.

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

Требуется:

1. пользуясь явным видом функций $u(x,y)$, $k(x,y)$, $q(x,y)$ определить правую часть уравнения Пуассона $F(x,y)$ и граничные условия $\psi(x,y)$;
2. собрать разностную схему для уравнения Пуассона с граничными условиями;
3. разработать последовательный код программы, вычисляющий приближенное решение разностной схемы методом наименьших невязок, проверить точность схемы, выполнив расчеты на сгущающихся сетках (M, N) = (20, 20), (40, 40), (80, 80), (160, 160);
4. используя средства библиотеки MPI, разработать параллельный код программы, вычисляющий приближенное решение разностной схемы методом наименьших невязок, проверить качество работы алгоритма,

выполнив расчеты на сетке (M, N) = (160, 160) на одном, четырех и шестнадцати процессах;

5. провести исследование параллельных характеристик MPI-программы, выполнив расчеты на вычислительных комплексах IBM BlueGene/P и IBM Polus;
6. разработать гибридный MPI / OpenMP код программы, провести исследование параллельных характеристик гибридной программы и сравнить полученные результаты с программой, не использующей директивы OpenMP.

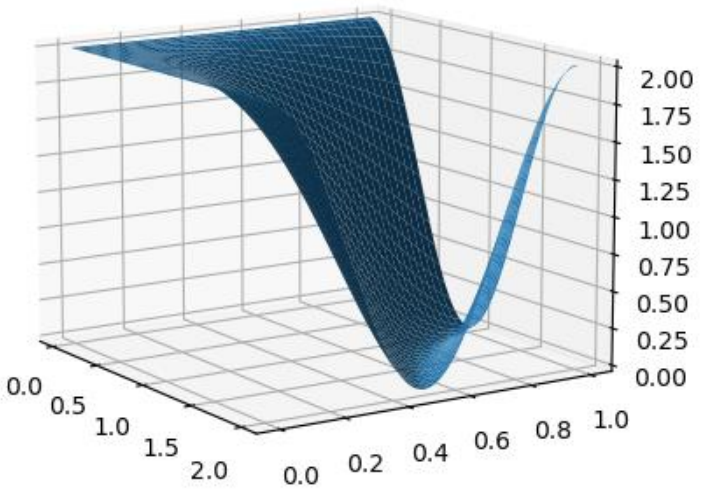
В соответствии с вариантом задания рассматриваются следующие данные:

- $A_1 = 0, A_2 = 2, B_1 = 0, B_2 = 1$;
- $u(x, y) = u_4(x, y) = 1 + \cos(\Pi xy)$;
- $k(x, y) = k_3(x, y) = 4 + x + y$;
- $q(x, y) = q_0(x, y) = 0$;
- граничные условия $\gamma_R, \gamma_L, \gamma_T, \gamma_B$ показаны выше.

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.

В соответствии с вариантом, требуется восстановить следующую функцию:

$$u(x, y) = 1 + \cos(\Pi xy).$$



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

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

$$\frac{\partial u}{\partial x} = -\Pi y \sin \Pi xy, \quad \frac{\partial u}{\partial y} = -\Pi x \sin \Pi xy,$$

$$\begin{aligned}
\Delta u = \frac{\partial}{\partial x}\left((4+x+y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left((4+x+y)\frac{\partial u}{\partial y}\right) \\
= \frac{\partial}{\partial x}\big((4+x+y)(-\Pi y \sin \Pi xy)\big) \\
+ \frac{\partial}{\partial y}\big((4+x+y)(-\Pi x \sin \Pi xy)\big) \\
= \frac{\partial}{\partial x}(-4\Pi y \sin \Pi xy - \Pi xy \sin \Pi xy - \Pi y^2 \sin \Pi xy) \\
+ \frac{\partial}{\partial y}(-4\Pi x \sin \Pi xy - \Pi xy \sin \Pi xy - \Pi x^2 \sin \Pi xy) \\
= -\Pi^2 \cos \Pi xy\,(x^3 + y^3 + 4x^2 + 4y^2 + x^2 y + xy^2) \\
- \Pi \sin \Pi xy (x+y)
\end{aligned}$$

Учитывая, что $q(x,y)u = 0$, правая часть уравнения (1) имеет следующий вид:

$$\begin{aligned}
F(x,y) = -\Delta u \\
= \Pi^2 \cos \Pi xy\,(x^3 + y^3 + 4x^2 + 4y^2 + x^2 y + xy^2) \\
+ \Pi \sin \Pi xy (x+y).
\end{aligned} \tag{2}$$

Определим функцию $\psi(x,y)$.

$$\psi(x,y) = \begin{cases} (6+\tilde{y})(-\Pi\tilde{y}\sin(2\Pi\tilde{y}) + 1 + \cos(2\Pi\tilde{y}), & x = 2, \tilde{y} \in [0;1] \\ 0, & x = 0, \tilde{y} \in [0;1] \\ (5+\tilde{x})(-\Pi\tilde{x}\sin(2\Pi\tilde{x}) + 1 + \cos(2\Pi\tilde{x}), & \tilde{x} \in [0;2], y = 1 \\ 0, & \tilde{x} \in [0;2], y = 0 \end{cases} \tag{3}$$

3 Разностная схема решения задачи

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

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

Здесь $h_1 = 2/M$, $h_2 = 1/N$. Через ω_h обозначим множество внутренних узлов сетки $\overline{\omega}_h$, т.е. множество узлов сетки прямоугольника, не лежащих на границе Г.

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

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

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

$$\rho^{(1)}(x_i) = \begin{bmatrix} 1, & 1 \le i \le M-1 \\ 1/2, & i = 0,\ i = M \end{bmatrix} \quad \rho^{(2)}(y_j) = \begin{bmatrix} 1, & 1 \le j \le N-1 \\ 1/2, & j = 0,\ j = N \end{bmatrix}$$

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

$$Aw = B, \quad (5)$$

где $A: H \to H$ – 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

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

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

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

а также определим сеточные коэффициенты

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

С учетом принятых обозначений разностный оператор Лапласа можно представить в более компактном и удобном виде

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

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

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

Аппроксимация граничных условий второго и третьего рода на правой и левой сторонах прямоугольника имеет вид:

$$\frac{2}{h_1}\left(a w_{\bar{x}}\right)_{Mj} + \left(q_{Mj} + \frac{2}{h_1}\right) w_{Mj} - \left(b w_{\bar{y}}\right)_{y,Mj} = F_{Mj} + \frac{2}{h_1}\psi_{\mathrm{Mj}}; \tag{11}$$

$$-\frac{2}{h_1}\left(a w_{\bar{x}}\right)_{1j} + q_{0j} \cdot w_{0j} - \left(b w_{\bar{y}}\right)_{y,0j} = F_{0j} + \frac{2}{h_1}\psi_{0\mathrm{j}}, \quad j = \overline{1, N-1}; \tag{12}$$

На верхней и нижней сторонах соответственно имеем:

$$\frac{2}{h_2}\left(\mathrm{b}w_{\overline{y}}\right)_{iN} + \left(q_{iN} + \frac{2}{h_2}\right)w_{iN} - \left(aw_{\overline{x}}\right)_{x,iN} = F_{iN} + \frac{2}{h_2}\psi_{\mathrm{iN}}; \quad (13)$$

$$-\frac{2}{h_2}\left(bw_{\overline{y}}\right)_{i1} + q_{i0}\cdot w_{i0} - \left(aw_{\overline{x}}\right)_{x,\mathrm{i0}} = F_{i0} + \frac{2}{h_2}\psi_{\mathrm{i0}}, \quad i = \overline{1, M-1}; \quad (14)$$

Сеточных уравнений (6-14) недостаточно, чтобы определить разностную схему для задачи с граничными условиями Неймана и Дирихле. Требуются сеточные уравнения для угловых точек прямоугольника П. Они имеют следующий вид:

- В вершине $P(0,0)$:

$$-\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{10} - \frac{2}{h_2}\left(bw_{\overline{y}}\right)_{01} + q_{00}\cdot w_{00} = F_{00} + \left(\frac{2}{h_1} + \frac{2}{h_2}\right)\psi_{00}; \quad (15)$$

- В вершине $P(2,0)$:

$$\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{M0} - \frac{2}{h_2}\left(bw_{\overline{y}}\right)_{M1} + \left(q_{M0} + \frac{2}{h_1}\right)w_{M0} = F_{M0} + \left(\frac{2}{h_1} + \frac{2}{h_2}\right)\psi_{\mathrm{M0}}; \quad (15)$$

- В вершине $P(2,1)$:

$$\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{MN} + \frac{2}{h_2}\left(bw_{\overline{y}}\right)_{MN} + \left(q_{MN} + \frac{2}{h_1} + \frac{2}{h_2}\right)w_{MN} = F_{MN} + \left(\frac{2}{h_1} + \frac{2}{h_2}\right)\psi_{\mathrm{MN}}; \quad (16)$$

- В вершине $P(0,1)$:

$$-\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{1N} + \frac{2}{h_2}\left(bw_{\overline{y}}\right)_{0N} + \left(q_{M0} + \frac{2}{h_2}\right)w_{0N} = F_{0N} + \left(\frac{2}{h_1} + \frac{2}{h_2}\right)\psi_{\mathrm{0N}}; \quad (15)$$

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.

Собрав все уравнения, получим разностную схему для уравнения Пуассона с граничными условиями:

$$
\begin{cases}
-\Delta_h w_{ij} + q_{ij} w_{ij} = F_{ij}, \qquad i = \overline{1, M-1}, \qquad j = \overline{1, N-1}; \\
\frac{2}{h_1}\,(a w_{\bar{x}})_{Mj} + \left(q_{Mj} + \frac{2}{h_1}\right) w_{Mj} - (b w_{\bar{y}})_{y,Mj} = F_{Mj} + \frac{2}{h_1}\,\psi_{\mathrm{Mj}}, \qquad i = \overline{1, M-1}; \\
-\frac{2}{h_1}\,(a w_{\bar{x}})_{1j} + q_{0j} \cdot w_{0j} - (b w_{\bar{y}})_{y,0j} = F_{0j} + \frac{2}{h_1}\,\psi_{0\mathrm{j}}, \quad i = \overline{1, M-1}; \\
\frac{2}{h_2}\,(\mathrm{b} w_{\bar{y}})_{iN} + \left(q_{iN} + \frac{2}{h_2}\right) w_{iN} - (a w_{\bar{x}})_{x,iN} = F_{iN} + \frac{2}{h_2}\,\psi_{\mathrm{iN}}, j = \overline{1, N-1}; \\
-\frac{2}{h_2}\,(b w_{\bar{y}})_{i1} + q_{i0} \cdot w_{i0} - (a w_{\bar{x}})_{x,\mathrm{i0}} = F_{i0} + \frac{2}{h_2}\,\psi_{\mathrm{i0}}, j = \overline{1, N-1}; \\
-\frac{2}{h_1}\,(a w_{\bar{x}})_{10} - \frac{2}{h_2}\,(b w_{\bar{y}})_{01} + q_{00} \cdot w_{00} = F_{00} + \left(\frac{2}{h_1} + \frac{2}{h_2}\right)\psi_{00}; \\
\frac{2}{h_1}\,(a w_{\bar{x}})_{M0} - \frac{2}{h_2}\,(b w_{\bar{y}})_{M1} + \left(q_{M0} + \frac{2}{h_1}\right) w_{M0} = F_{M0} + \left(\frac{2}{h_1} + \frac{2}{h_2}\right)\psi_{\mathrm{M0}}; \\
\frac{2}{h_1}\,(a w_{\bar{x}})_{MN} + \frac{2}{h_2}\,(b w_{\bar{y}})_{MN} + \left(q_{MN} + \frac{2}{h_1} + \frac{2}{h_2}\right) w_{MN} = F_{MN} + \left(\frac{2}{h_1} + \frac{2}{h_2}\right)\psi_{\mathrm{MN}}; \\
-\frac{2}{h_1}\,(a w_{\bar{x}})_{1N} + \frac{2}{h_2}\,(b w_{\bar{y}})_{0N} + \left(q_{M0} + \frac{2}{h_2}\right) w_{0N} = F_{0N} + \left(\frac{2}{h_1} + \frac{2}{h_2}\right)\psi_{0\mathrm{N}}.
\end{cases}
$$

Систему линейных алгебраических уравнений (СЛАУ) можно представить в операторном виде (5), в котором оператор A определен левой частью линейных уравнений, функция B – правой.

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:

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

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

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

В качестве условия остановки итерационного процесса можно взять неравенство

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

где ε – положительное число, определяющее точность итерационного метода.

Константа ε в данной задаче равна 10^{-6}.

5 Результаты последовательной работы программы

Описание программы (представлена в приложении А):

В глобальном пространстве имен реализованы функции $u(x,y)$, $k(x,y)$, $q(x,y)$, $F(x,y)$ согласно варианту. Также функции скалярного произведения векторов, разности двух векторов и функция нормы вектора.

Реализована функция, заполняющая матрицу B и матрицу Aw.

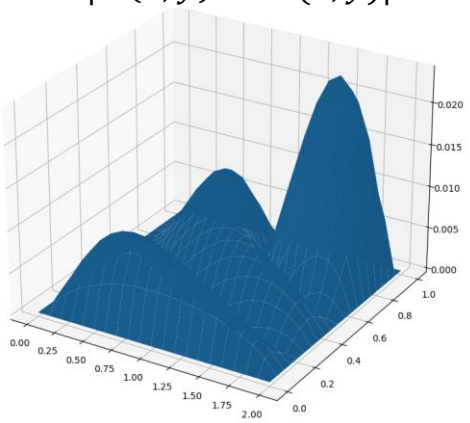
1. Матрица w в начале заполняется нулями и задаются граничные условия. Заполняется матрица B, равная $F(x,y)$.
2. В цикле на каждой итерации вычисляются:
   a. $w^{(k)}$
   b. $Aw^{(k)} = -\Delta_h w_{ij} + q_{ij} w_{ij}$
   c. $r^{(k)} = Aw^{(k)} - B$
   d. $Ar^{(k)}$
   e. τ_{k+1}
   f. $w_{ij}^{(k+1)} = w_{ij}^{(k)} - \tau_{k+1} r_{ij}^{(k)}$

Цикл выполняется пока не достигнута необходимая точность.

Результаты работы последовательного кода программы. С увеличением размерности сетки, точность восстановления исходной функции увеличивается.

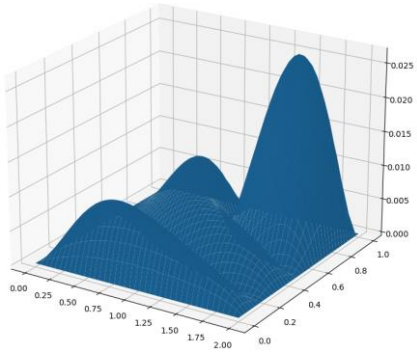
1. M = 20, N = 20

$$|u(x,y) - w(x,y)|$$



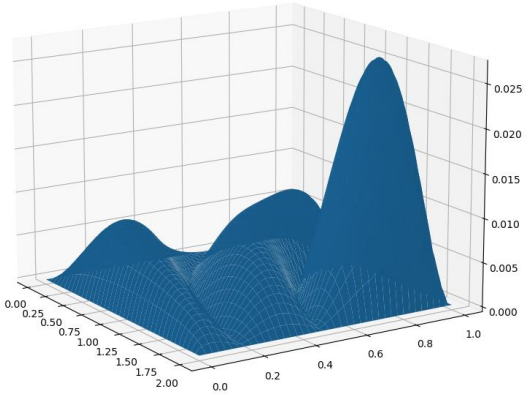
2. M = 40, N = 40

$$|u(x,y) - w(x,y)|$$



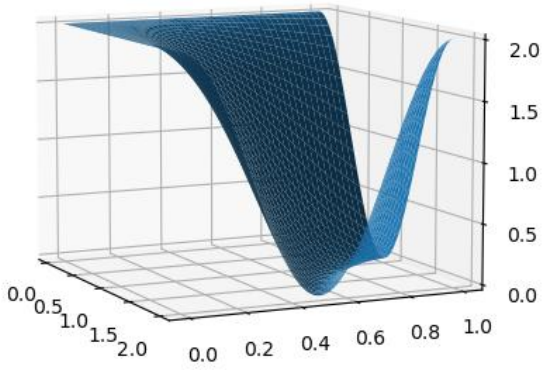
3. M = 80, N = 80

$$|u(x,y) - w(x,y)|$$

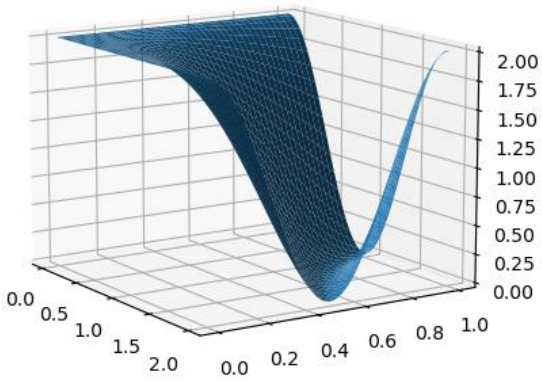


4. M = 160, N = 160

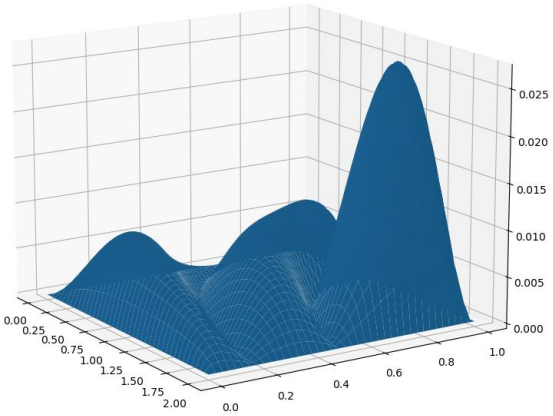
$$w(x,y)$$



$$u(x,y) = 1 + \cos(\Pi xy)$$



$$|u(x,y) - w(x,y)|$$



6 Результаты параллельной работы программы

Принцип работы параллельной MPI программы (представлена в приложении Б):

1. В глобальном пространстве имен реализованы функции $u(x,y)$, $k(x,y)$, $q(x,y)$, $F(x,y)$ согласно варианту, а также функции скалярного произведения векторов, разности двух векторов и функция нормы вектора.
2. Реализована функция, заполняющая матрицу B и матрицу Aw.
3. Создается декартова топология (сетка), с помощью функции MPI_Dims_create(commSize, 2, dims) с размерностью 2, которая помогает пользователю выбрать выгодное распределение процессов по каждой координате в зависимости от числа процессов.
4. Создается новый коммуникатор с помощью функции MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, true, &MPI_COMM_CART).
5. Для каждого процесса определяется размер подсетки и ее начальное положение (сдвиг по x и по y).
6. Матрица w в начале заполняется нулями и задаются граничные условия в подсетках, где это необходимо.
7. Матрица B, равная $F(x,y)$, заполняется в каждой подсетке.
8. В цикле на каждой итерации в каждом процессе вычисляются:
   a. $w^{(k)}$
   b. $Aw^{(k)} = -\Delta_h w_{ij} + q_{ij} w_{ij}$
   c. $r^{(k)} = Aw^{(k)} - B$
   d. $Ar^{(k)}$
   e. τ_{k+1}
   f. $w_{ij}^{(k+1)} = w_{ij}^{(k)} - \tau_{k+1} r_{ij}^{(k)}$

Цикл выполняется пока не достигнута необходимая точность.

Результаты работы параллельного кода программы на сетке $(M,N) = (160,160)$ на 1, 4, 16 процессах.

Число процессоров	Число итераций	Время решения
1	1593	114.149
4	1080	18.5819
16	565	2.59223

Исследование параллельных характеристик

Таблица с результатами расчетов на ПВС IMB Blue Gene/P для MPI программы

Число процессоров N_p	Число точке сетки $M \times N$	Время решения T	Ускорение S
128	500×1000	10.2297	1
256	500×1000	5.07834	2.01
512	500×1000	4.0462	2.53
128	1000×1000	26.5936	1
256	1000×1000	5.08288	5.23
512	1000×1000	4.45616	5.97

Таблица с результатами расчетов на ПВС IMB Blue Gene/P для гибридной MPI/OpenMP программы (8 нитей)

Число процессоров N_p	Число точке сетки $M \times N$	Время решения T	Ускорение S
128	500×1000	3.50224	1
256	500×1000	1.67182	2.09
512	500×1000	1.44369	2.43
128	1000×1000	9.28549	1
256	1000×1000	1.66639	5.57
512	1000×1000	1.54392	6.01

Таблица с результатами расчетов на ПВС IMB Polus

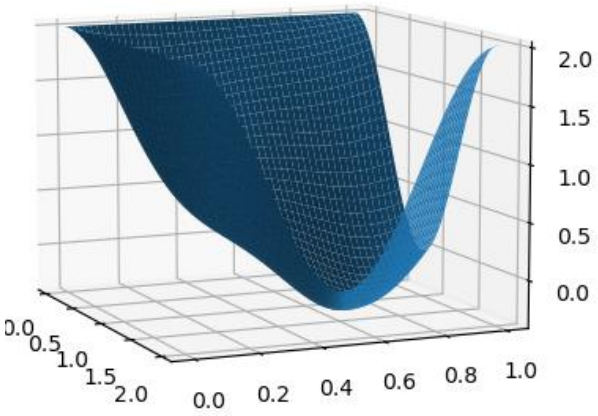
Число процессоров N_p	Число точке сетки $M \times N$	Время решения T	Ускорение S
4	500×500	47.0581	1
8	500×500	16.8597	2.79
16	500×500	15.3588	3.06
32	500×500	3.24503	14.5
4	500×1000	42.4628	1
8	500×1000	37.2763	1.14
16	500×1000	20.6121	2.06
32	500×1000	7.80147	5.44

По результатам, приведенным в таблицах, можно сделать вывод, что время работы гибридной MPI/OpenMP программы намного ниже. Чем больше используется параллельных процессов, тем программа выполняет вычисления быстрее, но разница между 256 и 512 процессорами уже не существенна.

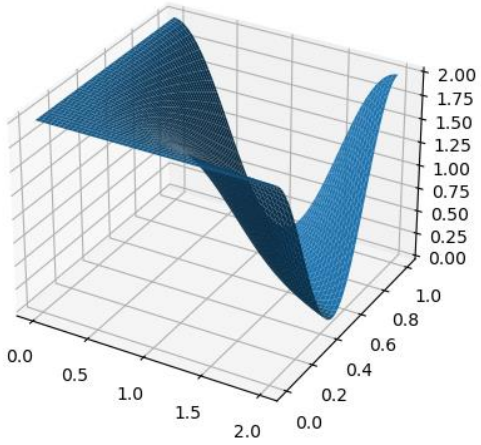
С увеличением размерности сетки увеличивалась погрешность вычислений. На наибольшей размерности сетки восстановленный график функции заметно отличается от исходного графика функции. Присутствует погрешность вычислений около 0.8.

M = 1000, N = 1000

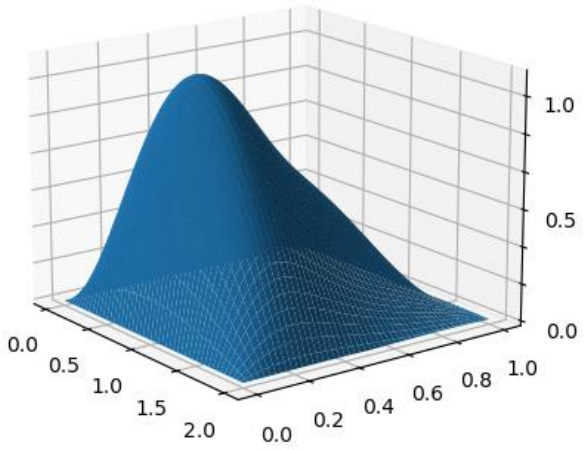
$w(x, y)$



$u(x, y) = 1 + \cos(\Pi xy)$



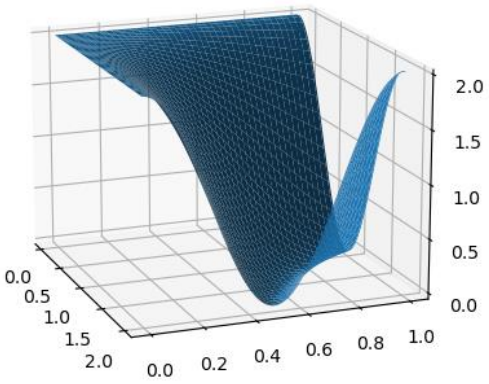
$|u(x, y) - w(x, y)|$



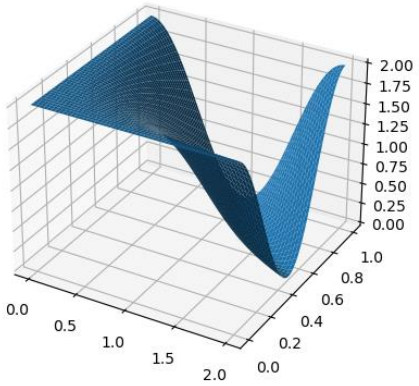
Наилучшие результаты получены на сетке 160x160, когда график уже визуально неотличим от исходного и присутствует незначительная погрешность. Также присутствует небольшая погрешность на сетке 500x500. Можно сделать вывод, что оптимально выбирать размерность сетки между этими значениями. Стоит отметить, что ошибка вычислений на сетке 160x160 при параллельной работе программы выше, чем при последовательном вычислении, но время выполнения программы заметно меньше.

M = 160, N = 160

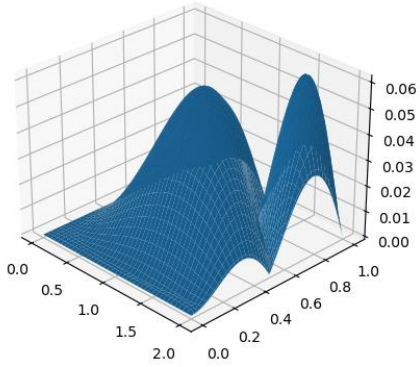
$w(x, y)$



$u(x, y) = 1 + \cos(\Pi xy)$

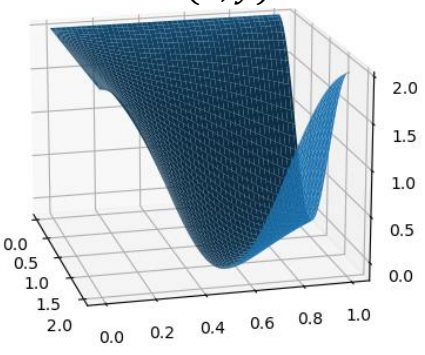


$|u(x, y) - w(x, y)|$

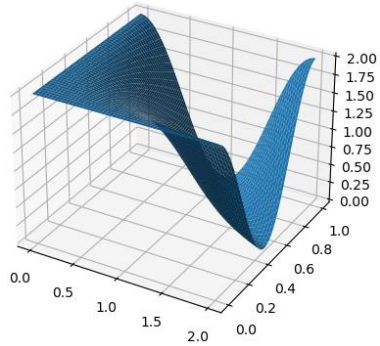


M = 500, N = 500

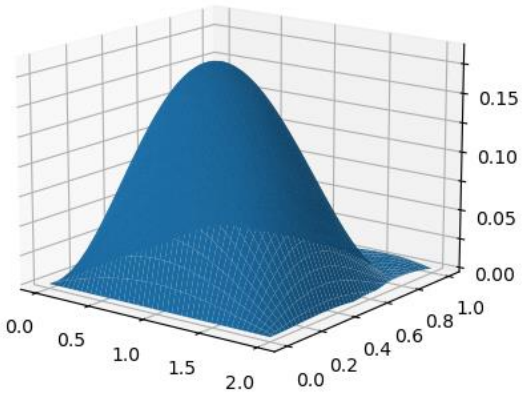
$w(x, y)$



$u(x, y) = 1 + \cos(\Pi xy)$



$|u(x, y) - w(x, y)|$



Список использованной литературы

1. АнтоновА.С. Технологии параллельного программирования MPI и OpenMP: Учеб. пособие.Предисл.: В.А.Садовничий.—Издательство Московского университета М.:, 2012.—С.344.

2. Высокопроизводительные вычисления на ВМК МГУ // [Электронный ресурс] – URL: http://hpc.cmc.msu.ru/

Приложение А

Код программы без распараллеливания процессов.

```
#include <math.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <stdio.h>
#include <vector>
#define A1 0
#define A2 2
#define B1 0
#define B2 1

using namespace std;

double u(double x, double y)
{
   return 1 + cos(M_PI * x * y);
}

double q(double x, double y)
{
            return 0;
}

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

double F(double x, double y)
{
   return M_PI * M_PI * cos(M_PI * x * y) * (x*x*x + y*y*y + 4*x*x + 4*y*y + x*x*y +
x*y*y) + M_PI * sin(x + y);
}

// Скалярное произведение векторов
double scalar(double *w0, double *w1, int M, int N, double h1, double h2)
{
   double res = 0;

   for(int i = 0; i <= M; i++)
   {
      for (int j = 0; j <= N; j++)
      {
         if(i == 0 && i == M && j == 0 && j == N)
            res +=  h1*h2*w0[i * (N + 1) + j] * w1[i * (N + 1) + j]/4;
         else
            res +=  h1*h2*w0[i * (N + 1) + j] * w1[i * (N + 1) + j];
```

```
        }
    }
    return res;
}

// Норма вектора
double norma(double *w, int M, int N, double h1, double h2)
{
    return sqrt(scalar(w, w, M, N, h1, h2));
}

// Разность двух векторов
void diff(double* res, double *w1, double *w0, int M, int N)
{
    for(int i = 0; i <= M; i++)
    {
        for (int j = 0; j <= N; j++)
        {
            if(i == 0 || i == M || j == 0 || j == N)
                res[i * (N + 1) + j] = 0;
            else
                res[i * (N + 1) + j] = w1[i * (N + 1) + j] - w0[i * (N + 1) + j];
        }
    }
}

// Матрица B
void B_(double* B, int M, int N, double h1, double h2)
{
    for(int i = 0; i <= M; i++)
        for (int j = 0; j <= N; j++)
        {
            B[i * (N + 1) + j] = F(A1 + i * h1, B1 + j * h2);
        }
}

// Матрица Aw
void A_(double* A, double *w, int M, int N, double h1, double h2)
{
    double aw, bw;

    for(int i = 0; i <= M; i++)
    {
        for (int j = 0; j <= N; j++)
        {
            if(i == 0 || i == M || j == 0 || j == N)
            {
                A[i * (N + 1) + j] = w[i * (N + 1) + j];
            }
            else
            {
```

```
            aw = k(A1 + (i + 0.5) * h1, B1 + j * h2) * (w[(i + 1) * (N + 1) + j] - w[i * (N + 1) +
j])/h1
            - k(A1 + (i - 0.5) * h1, B1 + j * h2) * (w[i * (N + 1) + j] - w[(i - 1) * (N + 1) +
j])/h1;

            bw = k(A1 + i * h1, B1 + (j + 0.5) * h2) * (w[i * (N + 1) + j + 1] - w[i * (N + 1) +
j])/h2
            - k(A1 + i * h1, B1 + (j - 0.5) * h2) * (w[i * (N + 1) + j] - w[i * (N + 1) + j - 1])/h2;

            A[i * (N + 1) + j] = -aw/h1 - bw/h2;
         }
      }
   }
}

int main()
{
   int M = 160;
   int N = 160;
   double e = pow(10, -6);
   cout << "e = " << e << "\n";

   double aw, bw;

   // Шаги сетки
   double h1 = (double)2/M;
   double h2 = (double)1/N;

   double *w = new double [(M + 1) * (N + 1)];

   for (int i = 0; i < (M + 1) * (N + 1); i++)
   {
      w[i] = 0;
   }

   // Задаём условия на границах
   for(int i = 0; i <= M; i++)
   {
      for (int j = 0; j <= N; j++)
      {

         aw = k(A1 + (i + 0.5) * h1, B1 + j * h2) * (u(A1+(i + 1)*h1,B1 + j*h2) -
u(A1+i*h1,B1 + j*h2))/h1
            + k(A1 + (i - 0.5) * h1, B1 + j * h2) * (u(A1+i*h1,B1 + j*h2) - u(A1+(i - 1)*h1,B1
+ j*h2))/h1;

         bw = k(A1 + i * h1, B1 + (j + 0.5) * h2) * (u(A1+i*h1,B1 + (j+1)*h2) - u(A1+i*h1,B1
+ j*h2))/h2
            + k(A1 + i * h1, B1 + (j - 0.5) * h2) * (u(A1+i*h1,B1 + j*h2) - u(A1+i*h1,B1 + (j-
1)*h2))/h2;
```

```
        if (i==0 && j==0)//00
        w[0] = u(A1 + i*h1, B1  + j * h2);
          w[0] =  ((2.)*((k(A1 + (1 + 0.5) * h1, B1 + j * h2) * (u(1,0) - u(0,0)))/(h1))
              +(2.)*(k(A1 + i * h1, B1 + (1 + 0.5) * h2) * (u(0,1) -
u(0,0))/(h2))+F(A1+i*h1,B1 + j*h2))/(2./h1);

        if (i==0 && (j>0 && j<N))//левая
          w[j] = ((2.)*(k(A1 + (1- 0.5) * h1, B1 + j * h2) * (u(A1 + (1) * h1,B1 + j * h2) -
u(A1 + 0 * h1, B1 + j * h2))/(h1))
                +(bw)+F(A1+i*h1,B1 + j*h2))/(2./h1)+u(A1+i*h1,B1 + j*h2);

        if (i==M && j==0)//M0
          w[(N + 1) * M] = u(A1 + i*h1, B1  + j * h2);
          w[(N + 1) * M] = ((2.)*(k(A1 + (i - 0.5) * h1, B1 + j * h2) * (u(A1+i*h1,B1 + j*h2)
- u(A1+(i - 1)*h1,B1 + j*h2))/(h1))
                          +(2.)*(k(A1 + i * h1, B1 + (1 - 0.5) * h2) * (u(A1+i*h1,B1 + 1*h2) -
u(A1+i*h1,B1 + (1-1)*h2))/(h2))
                          +(2./h1*u(A1+i*h1,B1 + j*h2))+F(A1+i*h1,B1 + j*h2))/(2./h1+2./h2);

        if (i==M && (j>0 && j<N))//правая
          w[(N + 1) * M + j] = ((2.)*((k(A1 + (i - 0.5) * h1, B1 + j * h2) * (u(A1+(i)*h1,B1 +
j*h2) - u(A1+(i-1)*h1,B1 + j*h2)))/(h1))
                            -(bw)+(2./h1)*u(A1+i*h1,B1 + j*h2))/(2./h1);

        if(i==0 && j==N)//0N
          w[N] = ((2.)*((k(A1 + (1 - 0.5) * h1, B1 + j * h2) * (u(A1+1*h1,B1 + j*h2) -
u(A1+(1 - 1)*h1,B1 + j*h2)))/(h1))
                        +(2.)*(k(A1 + i * h1, B1 + (j - 0.5) * h2) * (u(A1+i*h1,B1 + j*h2) -
u(A1+i*h1,B1 + (j-1)*h2))/(h2))
                        +(2./h2*u(A1+i*h1,B1 + j*h2))+F(A1+i*h1,B1 + j*h2))/(2./h1+2./h2);

        if(i==M && j==N)//MN
          w[(N + 1) * M + N] = ((2.)*((k(A1 + (i - 0.5) * h1, B1 + j * h2) * (u(A1+i*h1,B1 +
j*h2) - u(A1+(i - 1)*h1,B1 + j*h2)))/(h1))
                        +(2.)*(k(A1 + i * h1, B1 + (j - 0.5) * h2) * (u(A1+i*h1,B1 + j*h2) -
u(A1+i*h1,B1 + (j-1)*h2))/(h2))
                        +((2./h1+2./h2)*u(A1+i*h1,B1 + j*h2)))/(2./h1+2./h2);

        if(j==0 && (i>0 && i<M))//нижняя
          w[i * (N + 1)] = ((2.)*((k(A1 + i * h1, B1 + (1 - 0.5) * h2) * (u(A1+i*h1,B1 + 1*h2)
- u(A1+i*h1,B1 + (1-1)*h2)))/(h2))
                          +(aw))/(2./h2)+u(A1+i*h1,B1 + j*h2);

        if(j==N && ((i>0 && i<M)))//верхняя
          w[i * (N + 1) + N] = ((2.)*((k(A1 + i * h1, B1 + (j - 0.5) * h2) * (u(A1+i*h1,B1 +
j*h2) - u(A1+i*h1,B1 + (j-1)*h2)))/(h2))
                            +(aw)+(2./h2)*u(A1+i*h1,B1 + j*h2))/(2./h2);
      }
    }
```

```
double *r = new double [(M + 1) * (N + 1)];
double *Ar = new double [(M + 1) * (N + 1)];
double *Aw = new double [(M + 1) * (N + 1)];
double *w_ = new double [(M + 1) * (N + 1)];
double *s = new double [(M + 1) * (N + 1)];
double *B = new double [(M + 1) * (N + 1)];
double tau;

cout << "M = " << M << "\n" << "N = " << N << "\n";

//На первой итерации зададим нулевые значения
for (int i = 0; i <= (M + 1) * (N + 1); i++)
{
   w_[i] = 0;
}

B_(B, M, N, h1, h2);

int count = 0;
bool flag = true;

while(flag)
{
   count++;

   if (count > 1)
   {
      for (int i = 0; i < (M + 1) * (N + 1); i++)
         w_[i] = w[i];
   }
   else
   {
      for (int i = 0; i < (M + 1) * (N + 1); i++)
         w_[i] = 0;
   }

   A_(Aw, w, M, N, h1, h2);

   diff(r, Aw, B, M, N);

   A_(Ar, r, M, N, h1, h2);

   tau = scalar(Ar, r, M, N, h1, h2)/scalar(Ar, Ar, M, N, h1, h2);

   for (int i = 0; i <= M; i++)
   {
      for (int j = 0; j <= N; j++)
      {
         if(i != 0 && i != M && j != 0 && j != N)
            w[i * (N + 1) + j] = w[i * (N + 1) + j] - tau * r[i * (N+1) + j];
```

```
            }
        }

        diff(s, w, w_, M, N);

        if (norma(s, M, N, h1, h2) < e)
            flag = false;
    }

    cout << "Itterations: " << count << "\n" << "Tau: " << tau << "\n" << "Eps: " << e << "\n";

    //Запись результатов в файл
    ofstream myfile;
    myfile.open("result_test_1.txt");
    for(int i = 2; i <= M-2; i++)
    {
        for(int j = 2; j <= N-2; j++)
        {
            myfile << w[i * (N + 1) + j] << "\n";
        }
    }
    myfile.close ();

    delete[] w;
    delete[] w_;
    delete[] s;
    delete[] r;
    delete[] Ar;
    delete[] Aw;
    delete[] B;

    cout << "Free memmory\n";

    return 0;
}
```

Приложение Б

Код программы с параллельными процессами.

```
#include <cmath>
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <mpi.h>
//#include <omp.h>
#include <unistd.h>
#include <fstream>
#define EPS 0.000001
#define A1 0
#define A2 2
#define B1 0
#define B2 1

using namespace std;

void get_cmd_params(int _argc, char **_argv, int &_M, int &_N)
{
   // Размерность сетки
   _M = 160;
   _N = 160;

   // Обработка параметров командной строки
   for (int i = 1; i < _argc; i++)
   {
      string option(_argv[i]);

      if (option.compare("-M") == 0)
      {
         _M = atoi(_argv[++i]);
      }

      if (option.compare("-N") == 0)
      {
         _N = atoi(_argv[++i]);
      }
   }
}

double u(double x, double y)
{
   return (1 + cos(M_PI * x * y));
}

double q(double x, double y)
{
```

```
    return 0;
}

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

double F(double x, double y)
{
    return (M_PI * M_PI * cos(M_PI * x * y) * (x*x*x + y*y*y + 4*x*x + 4*y*y + x*x*y +
x*y*y) + M_PI * sin(x + y));
}

// Скалярное произведение векторов
double scalar(double *w0, double *w1, int M, int N, double h1, double h2)
{
    double res = 0;
    for(int i = 0; i <= M; i++)
    {
        for (int j = 0; j <= N; j++)
        {
            if(i == 0 && i == M && j == 0 && j == N)
                res +=  h1*h2*w0[i * (N + 2) + j] * w1[i * (N + 2) + j]/4;
            else
                res +=  h1*h2*w0[i * (N + 2) + j] * w1[i * (N + 2) + j];
        }
    }
    return res;
}

// Норма вектора
double norm(double *w, int M, int N, double h1, double h2)
{
    return sqrt(scalar(w, w, M, N, h1, h2));
}

// Разность двух векторов
void diff(double* res, double *w1, double *w0, int M, int N)
{
    int i, j;
    //# pragma omp parallel for
    for(i = 0; i <= M + 1; i++)
    {
        for (j = 0; j <= N + 1; j++)
        {
            if(i == 0 || i == M + 1 || j == 0 || j == N + 1)
                res[i * (N + 2) + j] = 0;
            else
                res[i * (N + 2) + j] = w1[i * (N + 2) + j] - w0[i * (N + 2) + j];
        }
```

```
    }
}

// Матрица B
void B_(double* B, int M, int N, double h1, double h2, double x_p, double y_p)
{
    int i, j;
    //# pragma omp parallel for
    for(i = 0; i <= M + 1; i++)
        for (j = 0; j <= N + 1; j++)
        {
            B[i * (N + 2) + j] = F(x_p + i * h1, y_p + j * h2);
        }
}

// Матрица Aw
void A_(double* A, double *w, int M, int N, double h1, double h2, double x_p, double y_p)
{
    double aw, bw;
    for(int i = 0; i <= M + 1; i++)
    {
        for (int j = 0; j <= N + 2; j++)
        {
            if(i == 0 || i == M + 1 || j == 0 || j == N + 1)
            {
                A[i * (N + 2) + j] = w[i * (N + 2) + j];
            }
            else
            {
                aw = k(x_p + (i + 0.5) * h1, y_p + j * h2)*(w[(i + 1) * (N + 2) + j] - w[i * (N + 2) +
j]) / h1
                - k(x_p + (i - 0.5) * h1, y_p + j * h2) * (w[i * (N + 2) + j] - w[(i - 1) * (N + 2) + j]) /
h1;
                bw = k(x_p + i * h1, y_p + (j + 0.5) * h2) * (w[i * (N + 2) + j + 1] - w[i * (N + 2) +
j]) / h2
                - k(x_p + i * h1, y_p + (j - 0.5) * h2) * (w[i * (N + 2) + j] - w[i * (N + 2) + j - 1]) /
h2;
                A[i * (N + 2) + j] = -aw/h1 - bw/h2;
            }
        }
    }
}

int main(int argc, char **argv)
{
    int M, N;

    get_cmd_params(argc, argv, M, N);

    // ID процесса
    int rank;
```

```
    // Всего процессов
    int commSize;
    int rank_n;
    int dims[2];
    dims[0] = 0;
    dims[1] = 0;

    // Шаг сетки по x и y
    double h1 = (double)2/M;
    double h2 = (double)1/N;
    double eps_global = 1.0;

    // Инициализация процессов
    MPI_Init(&argc, &argv);
    MPI_Status status;
    MPI_Request request;
    MPI_Comm MPI_COMM_CART;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &commSize);

    // Конструктор декартовой топологии резмерности 2, количество узлов решетки =
количеству процессов
    MPI_Dims_create(commSize, 2, dims);

    cout << rank << " : " << " DIMS " << dims[0] << " " << dims[1] << "\n";

    // Создается новый коммуникатор с декартовой топологией
    int periods[2];
    periods[0] = 0;
    periods[1] = 0;
    MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, true, &MPI_COMM_CART);

    int coords[2];
    int coords_n[2];
    // Возвращает координаты процесса в декартовой топологии
    MPI_Cart_coords(MPI_COMM_CART, rank, 2, coords);

    cout << rank << " : " << " Coords " << coords[0] << " " << coords[1] << "\n";

    // Для каждого процесса сдвиг и размер по x (i)
    int x_shift;
    int x_size;
    // Для каждого процесса сдвиг и размер по y (j)
    int y_shift;
    int y_size;

    // Делим число процессоров по внутренней сетке расчётов
    if (M % dims[0] == 0)
    {
        x_size = M / dims[0];
        x_shift = coords[0] * (M / dims[0]);
```

```
    }
    else
    {
        if (coords[0] == 0)
        {
            x_size = M % dims[0] + M / dims[0];
            x_shift = 0;
        }
        else
        {
            x_size = M / dims[0];
            x_shift = M % dims[0] + coords[0] * (M / dims[0]);
        }
    }

    if (N % dims[1] == 0)
    {
        y_size = N / dims[1];
        y_shift = coords[1] * (N / dims[1]);
    }
    else
    {
        if (coords[1] == 0)
        {
            y_size = N % dims[1] + N / dims[1];
            y_shift = 0;
        }
        else
        {
            y_size = N / dims[1];
            y_shift = N % dims[1] + coords[1] * (N / dims[1]);
        }
    }

     cout << rank << " x_shift " << x_shift << " y_shift " << y_shift << " x_size " << x_size <<
" y_size " << y_size << "\n";

    double start_time = MPI_Wtime();

    int i,j;
    int k = 0;

    double *w = new double [(x_size + 2) * (y_size + 2)];
    double *w_ = new double [(x_size + 2) * (y_size + 2)];
    double *B = new double [(x_size + 2) * (y_size + 2)];
    double tau;
    double eps_local, eps_r;

    double *Aw = new double [(x_size + 2) * (y_size + 2)];
    double *r = new double [(x_size + 2) * (y_size + 2)];
    double *Ar = new double [(x_size + 2) * (y_size + 2)];
```

```
double *s = new double [(x_size + 2) * (y_size + 2)];

// Задаем начальные значения
//# pragma omp parallel for
for (i = 0; i <= x_size + 1; i++)
{
   for (j = 0; j <= y_size + 1; j++)
   {
      // На первой итерации значение не имеет значения
      w[i * (y_size + 2) + j] = 0;
   }
}

B_(B, x_size, y_size, h1, h2, A1 + x_shift * h1, B1 + y_shift * h2);

while (eps_global > EPS)
{
   k++;

   //# pragma omp parallel for
   for (i = 0; i <= x_size + 1; i++)
   {
      for (j = 0; j <= y_size + 1; j++)
      {
         if(i == 0 || j == 0 || i == x_size + 1 || i == y_size + 1)
         {
            w_[i * (y_size + 2) + j] = 0;

         } else {
            w_[i * (y_size + 2) + j] = w[i * (y_size + 2) + j];
         }
      }
   }

   // Верхняя
   if ((dims[1] > 1 && coords[1] == dims[1] - 1) || dims[1] == 1)
   {
      for (i = 1; i <= x_size; i++)
      {
         w[i * (y_size + 2) + y_size + 1] = u(A1 + (x_shift + i) * h1, B2);
      }
   }

   // Нижняя
   if((coords[1] == 0 && dims[1] > 1) || dims[1] == 1)
   {
      for (i = 1; i <= x_size; i++)
      {
         w[i * (y_size + 2)] = u(A1 + (x_shift + i) * h1, B1);
      }
   }
```

```
        // Левая
        if ((dims[0] > 1 && coords[0] == 0) || dims[0] == 1)
        {
            for (j = 1; j <= y_size; j++)
            {
                w[j] = u(A1, B1 + (y_shift + j) * h2);
            }
        }

        // Правая
        if ((dims[0] > 1 && coords[0] == (dims[0] - 1)) || dims[0] == 1)
        {
            for (j = 1; j <= y_size; j++)
            {
                    w[(x_size + 1) * (y_size + 2) + j] = u(A2, B1 + (y_shift + j) * h2);
            }
        }

        A_(Aw, w, x_size, y_size, h1, h2, A1 + x_shift * h1, B1 + y_shift * h2);

        diff(r, Aw, B, x_size, y_size);

        A_(Ar, r, x_size, y_size, h1, h2, A1 + x_shift * h1, B1 + y_shift * h2);

        tau = scalar(Ar, r, x_size, y_size, h1, h2)/scalar(Ar, Ar, x_size, y_size, h1, h2);

        //# pragma omp parallel for
        for (i = 1; i <= x_size; i++)
        {
            for (j = 1; j <= y_size; j++)
            {
                w[i * (y_size + 2) + j] = w[i*(y_size + 2) + j] - tau * r[i * (y_size + 2) + j]/2;
            }
        }

        diff(s, w, w_, x_size, y_size);

        eps_local = norm(s, x_size, y_size, h1, h2);

        MPI_Allreduce(&eps_local, &eps_global, 1, MPI_DOUBLE, MPI_SUM,
MPI_COMM_WORLD);
    }

    MPI_Barrier(MPI_COMM_WORLD);
    double end_time = MPI_Wtime();

    if (rank == 0) cout << "TIME " << end_time - start_time << "\n";

    // Результаты
    if (rank == 0) cout << "Itterations: " << k << "\n";
```

```
    if (rank == 0) cout << "Tau: " << tau << "\n";
    if (rank == 0) cout << "Eps: " << EPS << "\n";

    cout << flush;
    usleep(500);

    // Освобождение памяти
    delete[] w;
    delete[] w_;
    delete[] B;
    delete[] r;
    delete[] Ar;
    delete[] Aw;
    delete[] s;

    MPI_Finalize();
    return 0;
}
```