Московский государственный университет имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра математических методов прогнозирования

Задание по курсу «Суперкомпьютерное моделирование и технологии»

Решение краевой задачи для уравнения Пуассона методом конечных разностей.

Выполнил:

студент 617 группы

Г.В. Кормаков

Москва, 2021

Содержание

1 Постановка задачи

В прямоугольнике $\Pi = \{(x,y) : A_1 \leqslant x \leqslant A_2, B_1 \leqslant y \leqslant B_2\}$, 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

$$\Delta u = \frac{\partial}{\partial x}\left(k(x,y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(k(x,y)\frac{\partial u}{\partial y}\right) \tag{2}$$

Для выделения единственного решения уравнение дополняется граничными условиями. На каждом отрезке границы прямоугольника Π задается условие одним из двух способов - условиями Дирихле (1-ого типа) или условиями второго (Неймана) и третьего типа.

Для выданного варианта 8 задания краевые условия задаются следующими условиями: третьего типа на правой и левой границе и второго на верхней и нижней на сетке (см. раздел 2). Общая формула условий третьего типа выглядит следующим образом:

$$\left(k\frac{\partial u}{\partial \boldsymbol{n}}\right)(x,y) + \alpha u(x,y) = \psi(x,y), \tag{3}$$

где $\boldsymbol{n}$ - единичная внешняя нормаль к границе прямоугольника.

Краевое условие второго типа (условие Неймана) содержится в краевом условии третьего типа (случай $\alpha = 0$ в 3).

Функции $F(x,y), \varphi(x,y), \psi(x,y)$, коэффициент $k(x,y)$, потенциал $q(x,y)$ и параметр $\alpha \geqslant 0$ считаются известными, функцию $u(x,y)$, удовлетворяющую уравнению 1 и граничным условиям, определенным вариантом задания 8, требуется найти.

Важно отметить, что нормаль $\boldsymbol{n}$ не определена в угловых точках прямоугольника. Краевое условие третьего типа будет рассматриваться лишь в тех точках границы, где нормаль существует.

2 Численный метод решения

2.1 Общая схема метода конечных разностей

В качестве метода решения задачи Пуассона 1 с потенциалом предлагается использовать метод конечных разностей.

Для этого область Π дискретизуем сеткой $\overline{\omega}_h = \overline{\omega}_1 \times \overline{\omega}_2$, где $\overline{\omega}_1$ – разбиение сетки по оси Ox с шагом $h_1 = \frac{A_2 - A_1}{M}$ и $\overline{\omega}_2$ – разбиение сетки по оси Oy с шагом $h_2 = \frac{B_2 - B_1}{N}$:

$$\overline{\omega}_1 = \left\{x_i = A_1 + ih_1, i = \overline{0, M}\right\}, \overline{\omega}_2 = \left\{y_j = B_1 + jh_2, j = \overline{0, N}\right\}$$

Также примем обозначение ω_h для внутренних узлов сетки $\overline{\omega}_h$

Рассматривается 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

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

где ρ_{ij} – весовая функция $\rho_{ij} = \rho^{(1)}(x_i)\,\rho^{(2)}(y_j)$ с

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

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

$$Aw = B \tag{4}$$

где $A : H \to H$ – оператор, действующий в пространстве сеточных функций, $B \in H-$ известная правая часть. Задача 4 называется разностной схемой. Решение этой задачи считается численным решением исходной дифференциальной задачи.

2.2 Конкретный вид разностной схемы для варианта 8

Уравнение 1 приближается для всех внутренних точек ω_h сетки разностной схемой следующего вида:

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

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

$$\Delta_h w_{ij} = \frac{1}{h_1}\left(k(x_i + 0.5h_1, y_j)\frac{w_{(i+1)j} - w_{ij}}{h_1} - k(x_i - 0.5h_1, y_j)\frac{w_{ij} - w_{(i-1)j}}{h_1}\right) +$$
$$+\frac{1}{h_2}\left(k(x_i, y_j + 0.5h_2)\frac{w_{i(j+1)} - w_{ij}}{h_2} - k(x_i, y_j - 0.5h_2)\frac{w_{ij} - w_{i(j-1)}}{h_2}\right)$$

В разностном операторе Лапласа введём обозначения для правых и левых разностных производных по координатам:

$$w_{x,ij} \equiv \frac{w_{(i+1)j} - w_{ij}}{h_1}, \quad w_{\overline{x},ij} \equiv w_{x,(i-1)j} = \frac{w_{ij} - w_{(i-1)j}}{h_1}$$
$$w_{y,ij} \equiv \frac{w_{i(j+1)} - w_{ij}}{h_2}, \quad w_{\overline{y},ij} \equiv w_{y,i(j-1)} \equiv \frac{w_{ij} - w_{i(j-1)}}{h_2}$$

и зададим сеточные коэффициенты

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

Тогда разностный оператор Лапласа равен

$$\Delta_h w_{ij} = \frac{1}{h_1}\left(k(x_i + 0.5h_1, y_j)\, w_{x,ij} - k(x_i - 0.5h_1, y_j)\, w_{x,(i-1)j}\right) +$$
$$+\frac{1}{h_2}\left(k(x_i, y_j + 0.5h_2)\, w_{y,ij} - k(x_i, y_j - 0.5h_2)\, w_{y,i(j-1)}\right) =$$
$$= \frac{a_{(i+1)j} w_{x,ij} - a_{ij} w_{x,(i-1)j}}{h_1} + \frac{b_{i(j+1)} w_{y,ij} - b_{ij} w_{y,i(j-1)}}{h_2} =$$
$$= \frac{a_{(i+1)j} w_{\overline{x},(i+1)j} - a_{ij} w_{\overline{x},ij}}{h_1} + \frac{b_{i(j+1)} w_{\overline{y},i(j+1)} - b_{ij} w_{\overline{y},ij}}{h_2} \equiv (a w_{\overline{x}})_{x,ij} + (b w_{\overline{y}})_{y,ij} \tag{5}$$

Итого, получаем $(M-1)\cdot(N-1)$ уравнений во внутренних точках:

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

Рассмотрим конкретные граничные условия, заданные в варианте 8 (см. таблицу 1).

Для правой (схема 7) и левой (схема 8) границы (γ_R, γ_L соответственно) задаются условия третьего типа. Разностный вариант (с учётом членов для соответствия порядка погрешности аппроксимации с основным уравнением 5) для них выглядит следующим образом:

$$\frac{2}{h_1}(a w_{\overline{x}})_{Mj} + \left(q_{Mj} + \frac{2\alpha_R}{h_1}\right) w_{Mj} - (b w_{\overline{y}})_{y,Mj} = F_{Mj} + \frac{2}{h_1}\psi_{Mj}^{(R)}, \quad j = \overline{1, N-1} \tag{7}$$
$$-\frac{2}{h_1}(a w_{\overline{x}})_{1j} + \left(q_{0j} + \frac{2\alpha_L}{h_1}\right) w_{0j} - (b w_{\overline{y}})_{y,0j} = F_{0j} + \frac{2}{h_1}\psi_{0j}^{(L)}, \quad j = \overline{1, N-1} \tag{8}$$

Для верхней (схема 9) и нижней (схема 10) границы (γ_T, γ_B соответственно) задаются условия второго типа (Неймана).

$$\frac{2}{h_2}(b w_{\overline{y}})_{iN} + q_{iN} w_{iN} - (a w_{\overline{x}})_{x,iN} = F_{iN} + \frac{2}{h_2}\psi_{iN}^{(T)}, \quad i = \overline{1, M-1} \tag{9}$$
$$-\frac{2}{h_2}(b w_{\overline{y}})_{i1} + q_{i0} w_{i0} - (a w_{\overline{x}})_{x,i0} = F_{i0} + \frac{2}{h_2}\psi_{i0}^{(B)}, \quad i = \overline{1, M-1} \tag{10}$$

Также в угловых точках не определена нормаль, поэтому необходимо их учесть отдельными уравнениями.

Для точки $P\left(A_1, B_1\right)$ прямоугольника П

$$-\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{10} - \frac{2}{h_2}\left(bw_{\overline{y}}\right)_{01} + \left(q_{00} + \frac{2\alpha_L}{h_1}\right) w_{00} = F_{00} + \left(\frac{2}{h_1} + \frac{2}{h_2}\right)\psi_{00} \qquad (11)$$

Для точки $P\left(A_2, B_1\right)$ прямоугольника П

$$\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{M0} - \frac{2}{h_2}\left(bw_{\overline{y}}\right)_{M1} + \left(q_{M0} + \frac{2\alpha_R}{h_1}\right) w_{M0} = F_{M0} + \left(\frac{2}{h_1} + \frac{2}{h_2}\right)\psi_{M0} \qquad (12)$$

Для точки $P\left(A_2, B_2\right)$ прямоугольника П

$$\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{MN} + \frac{2}{h_2}\left(bw_{\overline{y}}\right)_{MN} + \left(q_{MN} + \frac{2\alpha_R}{h_1}\right) w_{MN} = F_{MN} + \left(\frac{2}{h_1} + \frac{2}{h_2}\right)\psi_{MN} \qquad (13)$$

Для точки $P\left(A_1, B_2\right)$ прямоугольника П

$$-\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{1N} + \frac{2}{h_2}\left(bw_{\overline{y}}\right)_{0N} + \left(q_{0N} + \frac{2\alpha_L}{h_1}\right) w_{0N} = F_{0N} + \left(\frac{2}{h_1} + \frac{2}{h_2}\right)\psi_{0N} \qquad (14)$$

Уточним, что обозначили за $\psi^{(T)}, \psi^{(B)}$ фунции краевых условий второго типа (для данного варианта), а за $\psi^{(R)}, \psi^{(L)}$ – функции краевых условий третьего типа. В угловых точках вектор нормали не определён, поэтому краевые условия равны значению функции $u(x, y)$.

Вариант задания	Граничные условия γ_R	γ_L	γ_T	γ_B	Решение u(x, y)	Коэфф. $k(x, y)$	Потенциал $q(x, y)$
8	3 тип	3 тип	2 тип	2 тип	$u_2(x, y)$	$k_3(x, y)$	$q_2(x, y)$

Таблица 1: Условия задания

Запишем **финальную систему**, убедившись, что она определена.

$$\begin{aligned}
-\left(aw_{\overline{x}}\right)_{x,ij}-\left(bw_{\overline{y}}\right)_{y,ij}+q_{ij}w_{ij}&=F_{ij}, i=\overline{1,M-1}, j=\overline{1,N-1}\\
\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{Mj}+\left(q_{Mj}+\frac{2}{h_1}\right)w_{Mj}-\left(bw_{\overline{y}}\right)_{y,Mj}&=F_{Mj}+\frac{2}{h_1}\psi^{(R)}_{Mj},\ j=\overline{1,N-1}\\
-\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{1j}+\left(q_{0j}+\frac{2}{h_1}\right)w_{0j}-\left(bw_{\overline{y}}\right)_{y,0j}&=F_{0j}+\frac{2}{h_1}\psi^{(L)}_{0j},\ j=\overline{1,N-1}\\
\frac{2}{h_2}\left(bw_{\overline{y}}\right)_{iN}+q_{iN}w_{iN}-\left(aw_{\overline{x}}\right)_{x,iN}&=F_{iN}+\frac{2}{h_2}\psi^{(T)}_{iN},\ i=\overline{1,M-1}\\
-\frac{2}{h_2}\left(bw_{\overline{y}}\right)_{i1}+q_{i0}w_{i0}-\left(aw_{\overline{x}}\right)_{x,i0}&=F_{i0}+\frac{2}{h_2}\psi^{(B)}_{i0},\ i=\overline{1,M-1}\\
-\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{10}-\frac{2}{h_2}\left(bw_{\overline{y}}\right)_{01}+\left(q_{00}+\frac{2}{h_1}\right)w_{00}&=F_{00}+\left(\frac{2}{h_1}+\frac{2}{h_2}\right)\frac{\psi^{(L)}_{00}+\psi^{(B)}_{00}}{2}\\
\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{M0}-\frac{2}{h_2}\left(bw_{\overline{y}}\right)_{M1}+\left(q_{M0}+\frac{2}{h_1}\right)w_{M0}&=F_{M0}+\left(\frac{2}{h_1}+\frac{2}{h_2}\right)\frac{\psi^{(R)}_{M0}+\psi^{(B)}_{M0}}{2}\\
\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{MN}+\frac{2}{h_2}\left(bw_{\overline{y}}\right)_{MN}+\left(q_{MN}+\frac{2}{h_1}\right)w_{MN}&=F_{MN}+\left(\frac{2}{h_1}+\frac{2}{h_2}\right)\frac{\psi^{(R)}_{MN}+\psi^{(T)}_{MN}}{2}\\
-\frac{2}{h_1}\left(aw_{\overline{x}}\right)_{1N}+\frac{2}{h_2}\left(bw_{\overline{y}}\right)_{0N}+\left(q_{0N}+\frac{2}{h_1}\right)w_{0N}&=F_{0N}+\left(\frac{2}{h_1}+\frac{2}{h_2}\right)\frac{\psi^{(L)}_{0N}+\psi^{(T)}_{0N}}{2}
\end{aligned}$$

Получили $(N-1)(M-1)+2(M-1)+2(N-1)+4=MN-M-N+1+2M+2N-4+4=MN+N+M+1$ уравнений. Число неизвестных w_{ij} равно $(M+1)(N+1)=MN+N+M+1$. И система гарантирует единственность решения для данной разностной схемы.

Таким образом, матрица оператора A определяется коэффициентами перед неизвестными в левой части, а матрицы B – в правой.

2.3 Метод решения СЛАУ

Приближенное решение системы уравнений 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.

$$\left\|w-w^{(k)}\right\|_E\xrightarrow[k\to+\infty]{}0$$

Метод является одношаговым. Итерация обновления $w^{(k+1)}$ записывается в виде

$$w^{(k+1)}_{ij}=w^{(k)}_{ij}-\tau_{k+1}r^{(k)}_{ij},$$

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 используем неравенство $\left\|w^{(k+1)} - w^{(k)}\right\|_E < \varepsilon$, где ε - положительное число, определяющее точность итерационного метода. Константу ε для данной задачи предлагается взять равной 10^{-6}.

2.4 Вид функций

В таблице 1 приведены конкретные функции для данного варианта. Они задаются следующим образом

$$u_2(x,y) = \sqrt{4+xy}, \Pi = [0,4] \times [0,3] \tag{15}$$

$$k_3(x,y) = 4 + x + y \tag{16}$$

$$q_2(x,y) = \begin{cases} x+y, & x+y \geqslant 0 \\ 0, & x+y < 0 \end{cases} \tag{17}$$

3 Нахождение $F(x,y), \psi(x,y)$

Поскольку для численной реализации нам предлагается сравнить приближеннное решение с истинным, то необходимо найти аналитический вид функции $F(x,y)$ и краевых условий[1] для известного решения $u(x,y) = u_2(x,y) = \sqrt{4+xy}$, $\Pi = [0,4] \times [0,3]$, $k(x,y) = k_3(x,y) = 4+x+y$ и потенциалом $q(x,y) = q_2(x,y) = \max(0, x+y)$. Для определённости, возьмём вектор нормали $\boldsymbol{n}$, направленным извне.

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

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

Аналогично для $\dfrac{\partial}{\partial y}$.

[1]В случае варианта 8 необходимо найти вид только функции $\psi(x,y)$

Тогда

$$\begin{aligned}
&F(x,y) = -\Delta u + q(x,y)u = \\
&= -\frac{\partial}{\partial x}\left(k(x,y)\frac{y}{2\sqrt{4+xy}}\right) - \frac{\partial}{\partial y}\left(k(x,y)\frac{x}{2\sqrt{4+xy}}\right) + q(x,y)u = \\
&= -\frac{y}{2u(x,y)} + k(x,y)\frac{y^2}{2u^3(x,y)} - \frac{x}{2u(x,y)} + k(x,y)\frac{x^2}{4u^3(x,y)} + q(x,y)u = \\
&= \frac{-2u^2y + ky^2 - 2u^2x + kx^2 + 4qu^4}{4u^3} = \\
&= \frac{-2(4+xy)(x+y) + (4+x+y)y^2 - 2(4+xy)x + (4+x+y)x^2 + 4qu^4}{4(4+xy)^{3/2}} = \\
&= \frac{x^3 - x^2(2y-4-y) - x(8+2y^2-y^2) + y(-8+4y+y^2) + 4q(4+xy)^2}{4(4+xy)^{3/2}} = \\
&= \frac{x^3 - x^2(y-4) - x(y^2+8) + y(y^2+4y-8) + 4\max(0,x+y)(4+xy)^2}{4(4+xy)^{3/2}}
\end{aligned} \tag{18}$$

Приведём пример вычислений функции $\psi(x,y)$ для в «общем виде» (знак $\pm$ не значит обязательное наличие члена, в зависимости от направления, одна из компонент нормали может равняться 0).

$$\begin{aligned}
&\psi(x,y) = \left(k\frac{\partial u}{\partial \boldsymbol{n}}\right)(x,y) + \alpha u(x,y) = k\left(\pm\frac{y}{2u} \pm \frac{x}{2u}\right) + \alpha u = \\
&= \frac{(4+x+y)(\pm x \pm y) + 2\alpha(4+xy)}{2\sqrt{4+xy}}
\end{aligned}$$

Для γ_R, γ_L $\alpha = 1$, для γ_T, γ_B $\alpha = 0$. Также учтём знаки нормали к поверхности в соответствующих направлениях. Таким образом, в обозначениях уравнений 11 - 13

$$\psi^{(R)}(x,y) = \frac{y(4+x+y) + 2(4+xy)}{2\sqrt{4+xy}};\ \psi^{(L)}(x,y) = \frac{-y(4+x+y) + 2(4+xy)}{2\sqrt{4+xy}} \tag{19}$$

$$\psi^{(T)}(x,y) = \frac{x(4+x+y)}{2\sqrt{4+xy}};\ \psi^{(B)}(x,y) = \frac{-x(4+x+y)}{2\sqrt{4+xy}} \tag{20}$$

4 Описание программной реализации

4.1 Формализация требований на домены

Перед непосредственной реализацией важно понять, как осуществлять разбиение на блоки-домены Π_{ij}, соблюдая следующие условия:

1. отношение количества узлов по переменым x и y в каждом домене принадлежало диапазону $[1/2, 2]$

2. количество узлов по переменым x и y любых двух доменов отличалось не более, чем на единицу.

Условия в данной формулировке приведены в постановке задания. Уточним более корректно смысл каждого пункта[2]. Обозначим количество узлов сетки в одном домене $n_x(i)$ и $n_y(j)$ соответственно по x и y. Количество узлов, вообще говоря, зависит от числа процессоров p, выделенных на задачу, в приведённых обозначениях считаем, что размер зависит от числа процессов, выделенных линейно на каждую ось (далее будет объяснён алгоритм разбиения).

Первое условие утверждает, что для любого домена его форма должна быть похожа на прямоугольную или квадратную. Формальнее,

$$\frac{n_x(i)}{n_y(j)} \in [0.5, 2] \quad \forall \Pi_{ij} \tag{21}$$

Данная конфигурация позволяет минимизировать потери во времени на перессылки.

Второе условие даёт равномерность разбиения на домены. Т.е., вообще говоря, размеры доменов по оси x не должны отличаться более, чем на 1, и по y аналогично. Таким образом, $n_x(i), n_y(j)$, фактически должны быть константными, но из-за того, что размеры сетки могут быть не кратны числу процессоров, на граничных блоках мы получаем иные размеры. Именно на то, чтобы отличия в размерах были в этом случае минимальны и направлено ограничение 2.

Формальнее,

$$|n_x(i_1) - n_x(i_2)| \leqslant 1,\ |n_y(j_1) - n_y(j_2)| \leqslant 1,\ \forall (i_1, j_1), (i_2, j_2) \tag{22}$$

4.2 Алгоритм разбиения на блоки

Приведём алгоритм, удовлетворяющий условиям 21,22. Обозначим за p_x число процессоров, работающих в ячейках по оси x, и p_y – по y.

Для гарантированного выполнения условия 22 возьмём последовательность размеров блоков по каждой оси, отличающихся только на 1 точку. Т.е. будем чередовать по оси x домены с размером a_x и a_x+1 (по y соответственно обозначим за a_y и a_y+1).

[2]Корректность гарантирована объяснениями преподавателей на лекции, посвящённой постановке задачи

Таким образом, получаем следующие разложения

$$\sum_{i=1}^{p_x} n_x(i) = k_1 a_x + k_2(a_x + 1) = M,\ k_1 + k_2 = p_x \Rightarrow a_x p_x + k_2 = M \tag{23}$$

$$\sum_{j=1}^{p_y} n_y(j) = s_1 a_y + s_2(a_y + 1) = N,\ s_1 + s_2 = p_y \Rightarrow a_y p_y + s_2 = N \tag{24}$$

Условие 21 в новых обозначениях формулируется как $0.5 \leqslant \frac{a_x}{a_y} \leqslant 2$, т.к. в приведённой формулировке алгоритма чередоваться бо́льшие блоки будут одновременно по осям.

Из формул 23, 24 видно, что необходимо поделить с остатком M и N на количество процессоров по осям. При этом не стоит забывать, что должно соблюдаться неравенство $p_x p_y \leqslant p$.

Предлагается следующая схема: берём $p_x = 2^k$ процессоров на ось x и $p_y = \lfloor \frac{p}{2^k} \rfloor$ процессоров на ось y.[3] Для выполнения условия 21 логичным кажется выбирать эти числа, исходя из пропорции 25.

$$\frac{2^k}{\frac{p}{2^k}} = \frac{M}{N} \Rightarrow \frac{2^{2k}}{p} = \frac{M}{N} \Rightarrow 2^{2k} = \frac{pM}{N} \Rightarrow k = \left\lfloor \frac{\log_2\left(\frac{pM}{N}\right)}{2} \right\rfloor \tag{25}$$

В случае $p = 2^n$ выражение 25 переходит в $k = \left\lfloor \frac{n + \log_2\left(\frac{M}{N}\right)}{2} \right\rfloor$.

Таким образом, итоговая схема получения блоков, удовлетворяющих условиям 21 (из-за выбора k согласно 25) и 22 (из-за выбора определённой последовательности), выглядит следующим образом:

1. Задаём $p_x = 2^{\left\lfloor \frac{\log_2\left(\frac{pM}{N}\right)}{2} \right\rfloor}$, $p_y = \left\lfloor \frac{p}{p_x} \right\rfloor$.
2. Получаем $a_x = \lfloor \frac{M}{p_x} \rfloor$, $a_y = \lfloor \frac{N}{p_y} \rfloor$ и фиксируем $k_2 = M \mod p_x$, $s_2 = N \mod p_y$
3. Далее генерируем $k_1 = p_x - k_2$ доменов по оси x с размером a_x и $s_1 = p_y - s_2$ доменов по оси y с размером a_y.
4. Затем начинается генерация k_2 доменов по оси x с размером $a_x + 1$ и s_2 доменов по оси y с размером $a_y + 1$

[3] Данная формула носит общий характер, в рамках текущего задания $p = 2^n$ и можно сразу сказать, что $p_y = 2^{n-k}$

4.3 Реализация с использованием MPI и OpenMP

В рамках экспериментов была осуществлена реализация описанной разностной схемы с использованием MPI и OpenMP. Реализация написана на языке C. Классическая MPI реализация приведена в 1, с методом сопряжённых градиентов в 9.

Также (из-за неэффективности вычисления на системе Blue Gene/P) реализована схема оптимизации с помощью сопряжённых градиентов (реализация с MPI в приложении 2). Метод сопряжённых градиентов (везде далее CG) может показывать нестабильность в случае нессиметричной матрицы СЛАУ, результаты сравнения продемонстрированы в сводных таблицах. Также для сравнения приводится время вычисления решения на ноутбуке с 4 ядрами (8-ю логическими процессорами)[4]

Реализация с использованием OpenMP отличается добавлением в циклах директив pragma omp parallel for.

Коротко опишем ключевые этапы программы с MPI. С помощью MPI_Cart_create создаётся топология на прямоугольной сетке, совпадающей по реализации с разбиением, предлагаемом в разделе 4.2.

Далее проверяется, какая ошибка каждого блока по разностной схеме от истинного решения, известного заранее. После проверки начинается основной цикл оптимизации. Критериев останова является получение нормы относительной ошибки меньше заданного значения. В случае сопряжённых градиентов - получение отношения нормы остатков к норме матрицы правой части меньше заданного числа.

В цикле заполняются блоки, расширенные по одной точке по каждому направлению (для получения от соседних процессов информации об их граничных точках). Обмен происходит с помощью асинхронной отправки и синхронного получения.

Стоит отметить, что на локальном запуске не требовалось наличия функции MPI_Wait(), однако Blue Gene/P потребовал написания данной реализации из-за специфики архитектуры.

После этого вычисляется остаток на текущем шаге и перед получением произведения Ar также проводится необходимая синхронизация.

[4] Asus ZenBook BX433F, Processor Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz. Installed RAM 16.0 GB (15.8 GB usable). System type 64-bit operating system, x64-based processor.

Затем вычисляется коэффициент (коэффициенты, в случае сопряжённых градиентов) для оптимизации и делается шаг.

В конце программы происходит барьерная синхронизация и вывод результатов по времени на стандартный поток вывода и по данным в необходимые для визуализации файлы.

5 Результаты на системах Blue Gene/P и Polus

Для данной задачи выполнены подсчёты ускорения программы на системах Blue Gene/P и Polus.

Под ускорением программы, запущенной на p MPI-процессах, понимается величина:

$$S_p = \frac{T_m}{T_p}$$

где T_m– время работы на минимальном числе MPI-процессов, T_p– время работы программы на p MPI-процессах.

Результаты запусков стандартной реализации на системе Blue Gene/P с использованием MPI и OpenMP приведены в 2 и 3.

Результаты использования только MPI 2 показывают лишь замедление времени исполнения при увеличении числа узлов. Возможные варианты объяснения видятся следующие:

- Топология, создаваемая MPI_Cart, становится ресурсоёмкой при нескольких обменах в ходе исполнения программы
- Медленный метод сходимости
- Загрузка суперкомпьютера
- Компиляция без оптимизации под архитектуру

Также видна закономерность в том, что на бóльшей сетке метод оптимизации сходится быстрее[5].

[5] Отметим, что чтобы уместиться в разрешённые временные рамки, была изменена начальная инициализация.

Число процессоров p	Сетка $M \times N$	Время T (мин)	Ускорение S
128	500×1000	5.031	1
256	500×1000	6.023	0.83
512	500×1000	9.889	0.51
128	$\mathbf{1000 \times 1000}$	**1.975**	**1**
256	1000×1000	2.341	0.84
512	1000×1000	3.488	0.57
(локально, CG) 4	$\mathbf{500 \times 1000}$	**4.491**	**1.12**
(локально, CG) 4	1000×1000	20.674	0.10

Таблица 2: Результаты расчетов MPI версии на ПВС IBM Blue Gene/P

Рассмотрим результаты для запусков программы, скомпилированной с OpenMP директивами.

Число процессоров p	Сетка $M \times N$	Время T (мин)	Ускорение S
128	500×1000	5.391	1
256	500×1000	3.214	1.68
512	$\mathbf{500 \times 1000}$	**3.213**	**1.68**
128	1000×1000	9.424	1
256	1000×1000	5.573	1.69
512	$\mathbf{1000 \times 1000}$	**3.554**	**2.65**
(локально, CG) 4	500×1000	8.905	0.61
(локально, CG) 4	1000×1000	11.094	0.85

Таблица 3: Результаты расчетов MPI+OpenMP версии на ПВС IBM Blue Gene/P

Для сравнения также приводится время, потраченное на локальном компьютере с 4 ядрами на тех же размерах сеток. Для корректности сравнения приведём результаты запуска метода сопряжённых градиентов на системе Blue Gene/P (см. 4, 5)

Число процессоров p	Сетка $M \times N$	Время T (мин)	Ускорение S
128	500×1000	0.824	1
256	500×1000	0.470	1.75
512	$\mathbf{500 \times 1000}$	**0.470**	**1.75**
128	1000×1000	1.856	1
256	1000×1000	1.053	1.76
512	$\mathbf{1000 \times 1000}$	**0.608**	**3.05**
(локально, CG) 4	500×1000	4.491	0.183
(локально, CG) 4	1000×1000	20.674	0.090

Таблица 4: Результаты расчетов MPI версии **с CG** на ПВС IBM Blue Gene/P

Число процессоров p	Сетка $M \times N$	Время T (мин)	Ускорение S
128	500×1000	5.742	1
256	500×1000	4.081	1.41
512	$\mathbf{500 \times 1000}$	**3.089**	**1.86**
128	1000×1000	0.422	1
256	$\mathbf{1000 \times 1000}$	**0.310**	**1.36**
512	1000×1000	3.495	0.121
(локально, CG) 4	500×1000	8.905	0.64
(локально, CG) 4	1000×1000	11.094	0.04

Таблица 5: Результаты расчетов MPI+OpenMP версии **с CG** на ПВС IBM Blue Gene/P

На системе ПВС IBM Polus с заданной точностью $1e-6$ за 15 минут не удалось получить результаты по некоторым конфигурациям, поэтому проведено сравнение только метода с CG оптимизацией.

Число процессоров p	Сетка $M \times N$	Время T (мин)	Ускорение S
4	500×500	1.391	1
8	500×500	0.669	2.08
16	500×500	0.363	3.83
32	$\mathbf{500 \times 500}$	**0.257**	**5.41**
4	500×1000	4.569	1
8	500×1000	2.647	1.73
16	500×1000	1.465	3.12
32	$\mathbf{500 \times 1000}$	**0.791**	**5.78**
(локально, CG) 4	500×500	2.616	0.53
(локально, CG) 4	*500 × 1000*	*4.491*	*1.02*

Таблица 6: Таблица с результатами расчетов MPI версии (только CG) на ПВС IBM Polus

На основании таблиц 2 - 6 можно сделать следующие выводы.

На Blue Gene/P

- OpenMP версия показывает ускорение почти в два раза. Без указания директив *pragma* у параллельных циклов (результаты таблицы 2) нет должного ресурса параллелизма и при увеличении числа процессоров наблюдается увеличение времени.
- Реализация метода оптимизации сопряжённых направлений ускоряет вычисление в 5 раз (из сравнения 2 с 4)
- При добавлении директив OpenMP наблюдается увеличение времени работы. Особенно выделяется случай с 512 процессорами и сеткой размера 1000×1000.

В случае прямоугольной сетки увеличение времени объясняется наружением симметричности матрицы СЛАУ, что не позволяет воспользоваться преимуществами метода сопряжённых градиентов.

В случае с 512 процессорами увеличение времени работы можно объяснить тем, что размеры блоков (исходя из вывода 25 и дальнейшего алгоритма) различаются почти в 2 раза (1000/32 и 1000/16), что является граничным случаем условий на домены и также нарушает симметрию.

На Polus

- Метод конечных разностей сходится явно дольше, чем на Blue Gene/P (наблюдалось достижение точности 6e-5 только к 15 минуте подсчёта).
- Идеально демонстрируется ускорение за счёт взятия бóльшего числа процессоров (получение ускорения в 5-6 раз при взятии 32 процессоров).
- Вычислительная мощность на прямоугольной сетке совпадает (или даже уступает производительности ноутбука с приведённым выше описанием).

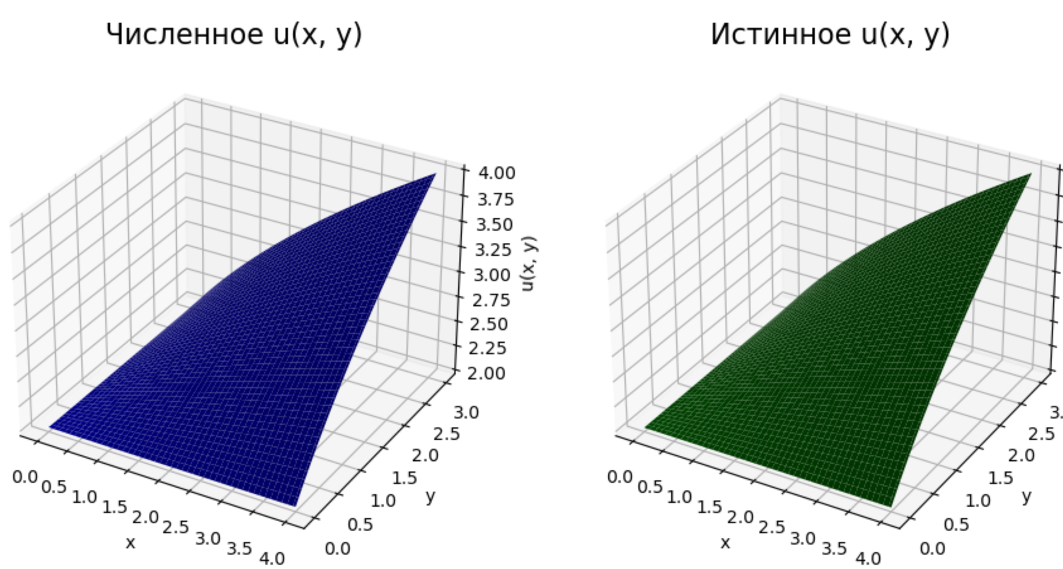

Рис. 1: Результат на сетке 500×1000

Итоговые результаты функций для сеток с наибольшим числом узлов приведены на рисунках 1, 2.

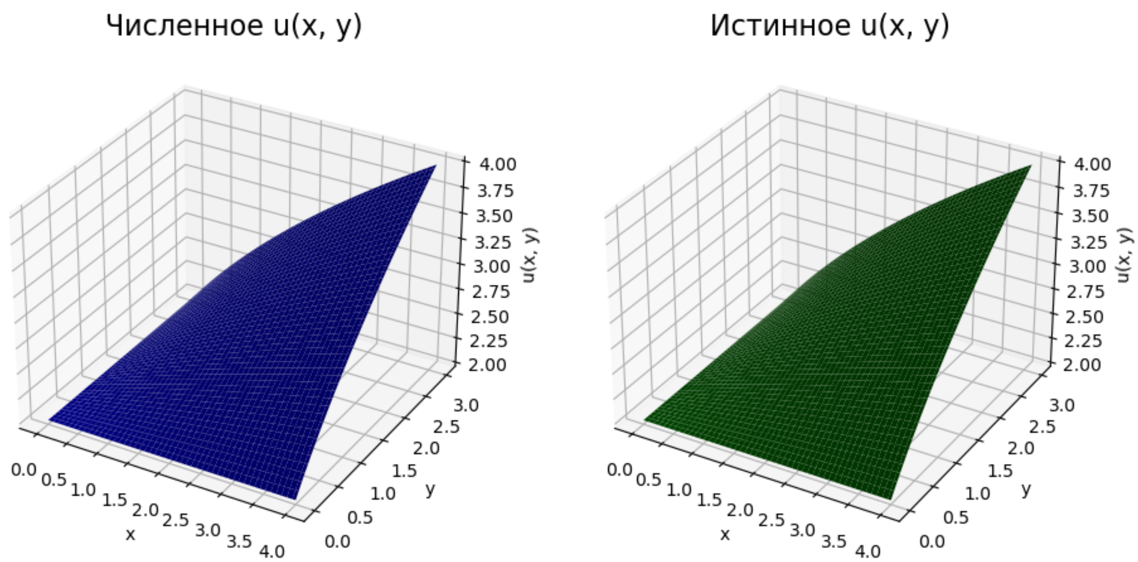

Рис. 2: Результат на сетке 1000×1000

6 Заключение

В ходе проведённых экспериментов была реализована программа для численного решения СЛАУ методом конечных разностей. Вычислительный эксперимент, с целью получения информативных результатов, дополнен сравнением с результатами, полученными методом сопряжённых градиентов и запуском на локальном компьютере.

Также описан аналитически метод, лежащий в основе разбиения на блоки MPI_Cart_create, получены аналитические выражения для правых частей и описана разностная схема.

Для представления результатов также использовались стандартные методы визуализации python, получающие на вход файлы, записанные каждым доменом (процессором) отдельно.

7 Литература

Источники

[1] Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. ЧИСЛЕННЫЕ МЕТОДЫ. - М.: Наука, 1987.

8 Приложение 1. Код MPI программы

```c
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
#include <time.h>
#include <math.h>
#include <unistd.h>


void print_matrix(int M, int N, double (*A)[N+2]){
    printf("Matrix:\n");
    for (int i=N+1; i>=0; i--){
        for (int j=0; j<=M+1; j++){
            printf("%3.2f ", A[i][j]);
        }
        printf("\n");
    }
}

void print_matrix_to_file(FILE *file,
                          int M, int N,
                          double (*A)[N+2]){
    for (int i=N+1; i>=0; i--){
        for (int j=0; j<=M+1; j++){
            fprintf(file,"%g ", A[i][j]);
        }
        fprintf(file, "\n");
    }
}

double u_2(double x, double y){
    return sqrt(4 + x * y);
}

double k_3(double x, double y){
    return 4 + x + y;
}

double q_2(double x, double y){
    double sum = x + y;
    if (sum < 0) {
        return 0;
    } else {
        return sum;
    }
}

double F(double x, double y){
    return ((pow(x, 3) - x*x*(y - 4) - x*(y*y + 8) +
    y*(y*y + 4*y - 8) + 4*q_2(x, y)*pow((4 + x*y), 2)) /
    (4 * pow((4 + x*y), 1.5)));
}

double psi_R(double x, double y){
    return (y*(4 + x + y) + 2*(4 + x*y)) / (2*sqrt(4 + x*y));
}


double psi_L(double x, double y){
    return (-y*(4 + x + y) + 2*(4 + x*y)) / (2*sqrt(4 + x*y));
}
```

```c


double psi_T(double x, double y){
    return (x*(4 + x + y)) / (2*sqrt(4 + x*y));
}


double psi_B(double x, double y){
    return -psi_T(x, y);
}

double rho_1(int i,
             int M,
             int left_border,
             int right_border){
    if ((left_border && i == 1) || (right_border && i == M))
        return 0.5;
    return 1;
}

double rho_2(int j,
             int N,
             int bottom_border,
             int top_border){
    if ((bottom_border && j == 1) || (top_border && j == N))
        return 0.5;
    return 1;
}

double dot_product(int M, int N,
                   double (*U)[N + 2], double (*V)[N + 2],
                   double h1, double h2,
                   int left_border, int right_border,
                   int top_border, int bottom_border
                   ){
    double answer = 0.;
    double rho, r1, r2;

    for (int i=1; i <= M; i++){
        for (int j=1; j <= N; j++){
            r1 = rho_1(i, M, left_border, right_border);
            r2 = rho_2(j, N, bottom_border, top_border);
            rho = r1 * r2;
            answer += (rho * U[i][j] * V[i][j] * h1 * h2);
        }
    }
    return answer;
}


double norm(int M, int N, double (*U)[N + 2],
            double h1, double h2,
            int left_border, int right_border,
            int top_border, int bottom_border){
    return sqrt(dot_product(M, N, U, U, h1, h2,
                            left_border, right_border,
                            top_border, bottom_border));
}


void B_right(int M, int N, double (*B)[N+2],
       double h1, double h2,
```

```
        double x_start, double y_start,
        int left_border, int right_border,
        int top_border, int bottom_border){
    int i, j;

    for(i = 0; i <= M + 1; i++)
        for (j = 0; j <= N + 1; j++)
            B[i][j] = F(x_start + (i - 1) * h1, y_start + (j - 1) * h2);

    if (left_border){
        for (j = 1; j <= N; j++) {
            B[1][j] = (F(x_start, y_start + (j - 1) * h2) +
                    psi_L(x_start, y_start + (j - 1) * h2) * 2/h1);
        }
    }
    if (right_border){
        for (j = 1; j <= N; j++) {
            B[M][j] = (F(x_start + (M - 1)*h1, y_start + (j - 1) * h2) +
                       psi_R(x_start + (M - 1)*h1, y_start + (j - 1) * h2) * 2/h1);
        }
    }
    if (top_border){
        for (i = 1; i <= M; i++) {
            B[i][N] = (F(x_start + (i - 1)*h1, y_start + (N - 1)*h2) +
                       psi_T(x_start + (i - 1)*h1, y_start + (N - 1)*h2) * 2/h2);
        }
    }

    if (bottom_border){
        for (i = 1; i <= M; i++) {
            B[i][1] = (F(x_start + (i - 1)*h1, y_start) +
                       psi_B(x_start + (i - 1)*h1, y_start) * 2/h2);
        }
    }
    if (left_border && top_border){
        B[1][N] = (F(x_start, y_start + (N - 1)*h2) +
                (2/h1 + 2/h2) * (psi_L(x_start, y_start + (N - 1)*h2) +
                                psi_T(x_start, y_start + (N - 1)*h2)) / 2);
    }
    if (left_border && bottom_border){
        B[1][1] = (F(x_start, y_start)
                + (2/h1 + 2/h2) * (psi_L(x_start, y_start) + psi_B(x_start, y_start)) / 2);
    }
    if (right_border && top_border){
        B[M][N] = (F(x_start + (M - 1)*h1, y_start + (N - 1)*h2) +
                (2/h1 + 2/h2) * (psi_R(x_start + (M - 1)*h1, y_start + (N - 1)*h2) +
                                psi_T(x_start + (M - 1)*h1, y_start + (N - 1)*h2)) / 2);
    }
    if (right_border && bottom_border){
        B[M][1] = (F(x_start + (M - 1)*h1, y_start) +
                (2/h1 + 2/h2) * (psi_R(x_start + (M - 1)*h1, y_start) +
                                psi_B(x_start + (M - 1)*h1, y_start)) / 2);
    }
}


double aw_x_ij(int N,
```

```
            double (*w)[N+2],
            double x_start, double y_start,
            int i, int j,
            double h1, double h2
            ){
    return (1/h1) * (k_3(x_start + (i + 0.5 - 1) * h1,y_start + (j - 1) * h2) * (w[i + 1][j] - w[i][j]) / h1
                - k_3(x_start + (i - 0.5 - 1) * h1,y_start + (j - 1) * h2) * (w[i][j] - w[i - 1][j]) / h1);
}

double aw_ij(int N,
             double (*w)[N+2],
             double x_start, double y_start,
             int i, int j,
             double h1, double h2
){
    return (k_3(x_start + (i - 0.5 - 1) * h1,y_start + (j - 1) * h2) * (w[i][j] - w[i - 1][j]) / h1);
}

double bw_y_ij(int N,
            double (*w)[N+2],
            double x_start, double y_start,
            int i, int j,
            double h1, double h2
){
    return (1/h2) * (k_3(x_start + (i - 1) * h1,y_start + (j + 0.5 - 1) * h2) * (w[i][j + 1] - w[i][j]) / h2
                - k_3(x_start + (i - 1) * h1,y_start + (j - 0.5 - 1) * h2) * (w[i][j] - w[i][j - 1]) / h2);
}

double bw_ij(int N,
             double (*w)[N+2],
             double x_start, double y_start,
             int i, int j,
             double h1, double h2
){
    return (k_3(x_start + (i - 1) * h1,y_start + (j - 0.5 - 1) * h2) * (w[i][j] - w[i][j-1]) / h2);
}

void Aw_mult(int M, int N,
            double (*A)[N+2], double (*w)[N+2],
            double h1, double h2,
            double x_start, double y_start,
            int left_border, int right_border,
            int top_border, int bottom_border
            )
{
    double aw_x, bw_y;
    int i, j;
    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){
                A[i][j] = w[i][j];
            } else {
                aw_x = aw_x_ij(N, w, x_start, y_start, i, j, h1, h2);
                bw_y = bw_y_ij(N, w, x_start, y_start, i, j, h1, h2);
                A[i][j] = -aw_x - bw_y + q_2(x_start + (i - 1) * h1,
                                            y_start + (j - 1) * h2) * w[i][j];
```

```
            }
        }
    }

    // Left interior border filling
    if (left_border){
        for (j = 1; j <= N; j++) {
            aw_x = aw_ij(N, w, x_start, y_start, 2, j , h1, h2);
            bw_y = bw_y_ij(N, w, x_start, y_start, 1, j, h1, h2);
            A[1][j] = -2*aw_x / h1 - bw_y + (q_2(x_start,
                                                  y_start + (j - 1) * h2) + 2/h1) *
                                                      w[1][j];
        }
    }

    // Right interior border
    if (right_border){
        for (j = 1; j <= N; j++) {
            aw_x = aw_ij(N, w, x_start, y_start, M, j, h1, h2);
            bw_y = bw_y_ij(N, w, x_start, y_start, M, j, h1, h2);
            A[M][j] = 2*aw_x / h1 - bw_y + (q_2(x_start + (M - 1) * h1,
                                                 y_start + (j - 1) * h2) + 2/h1) *
                                                     w[M][j];
        }
    }

    // Top border
    if (top_border){
        for (i = 1; i <= M; i++) {
            aw_x = aw_x_ij(N, w, x_start, y_start, i, N, h1, h2);
            bw_y = bw_ij(N, w, x_start, y_start, i, N, h1, h2);
            A[i][N] = -aw_x + 2*bw_y / h2 + q_2(x_start + (i - 1) * h1,
                                                 y_start + (N - 1) * h2) * w[i][N];
        }
    }

    // Bottom border
    if (bottom_border){
        for (i = 1; i <= M; i++) {
            aw_x = aw_x_ij(N, w, x_start, y_start, i, 1, h1, h2);
            bw_y = bw_ij(N, w, x_start, y_start, i, 2, h1, h2);
            A[i][1] = -aw_x - 2*bw_y / h2 + q_2(x_start + (i - 1)* h1, y_start
                ) * w[i][1];
        }
    }
    if (left_border && bottom_border){
        aw_x = aw_ij(N, w, x_start, y_start, 2, 1, h1, h2);
        bw_y = bw_ij(N, w, x_start, y_start, 1, 2, h1, h2);
        A[1][1] = -2*aw_x / h1 - 2*bw_y / h2 + (q_2(x_start, y_start) + 2/h1)
            * w[1][1];
    }
    if (left_border && top_border){
        aw_x = aw_ij(N, w, x_start, y_start, 2, N, h1, h2);
        bw_y = bw_ij(N, w, x_start, y_start, 1, N, h1, h2);
        A[1][N] = -2*aw_x / h1 + 2*bw_y / h2 + (q_2(x_start, y_start + (N - 1)
             * h2) + 2/h1)* w[1][N];
    }

    if (right_border && bottom_border){
        aw_x = aw_ij(N, w, x_start, y_start, M, 1, h1, h2);
        bw_y = bw_ij(N, w, x_start, y_start, M, 2, h1, h2);
```

```
        A[M][1] = 2*aw_x / h1 - 2 * bw_y / h2 + (q_2(x_start + (M - 1) * h1, y_start) + 2/h1) * w[M][1];
    }
    if (right_border && top_border) {
        aw_x = aw_ij(N, w, x_start, y_start, M, N, h1, h2);
        bw_y = bw_ij(N, w, x_start, y_start, M, N, h1, h2);
        A[M][N] = 2*aw_x / h1 + 2 * bw_y / h2 + (q_2(x_start + (M - 1) * h1,
                                                y_start + (N - 1) * h2) + 2/h1) * w[M][N];
    }
}


void calculate_r(int M, int N,
                double (*r)[N+2],
                double (*Aw)[N+2],
                double (*B)[N+2]
                ){
    int i, j;

    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)
                r[i][j] = 0;
            else
                r[i][j] = Aw[i][j] - B[i][j];
        }
    }
}


void get_idx_n_idx(int *idx,
                  int *n_idx,
                  int process_amnt,
                  int grid_size,
                  int coordinate){
    if (grid_size % process_amnt == 0) {
        *n_idx = grid_size / process_amnt;
        *idx = coordinate * (grid_size / process_amnt);
    }
    else
    {
        if (coordinate == 0){
            *n_idx = grid_size % process_amnt + grid_size / process_amnt;
            *idx = 0;
        } else
        {
            *n_idx = grid_size / process_amnt;
            *idx = grid_size % process_amnt + coordinate * (grid_size / process_amnt);
        }
    }
}


#define A1 0.0
#define A2 4.0
#define B1 0.0
#define B2 3.0
#define EPS_REL 1e-6
#define DOWN_TAG 1000
#define MAX_ITER 100000
```

```c
void send_recv_borders(int n_x, int n_y,
                      const int process_amounts[2],
                      double x_idx,
                      double y_idx,
                      const int my_coords[2],
                      int tag,
                      double (*w)[n_y+2],
                      double b_send[n_x],
                      double l_send[n_y],
                      double t_send[n_x],
                      double r_send[n_y],
                      double b_rec[n_x],
                      double l_rec[n_y],
                      double t_rec[n_x],
                      double r_rec[n_y],
                      int left_border, int right_border,
                      int top_border, int bottom_border,
                      double h1, double h2,
                      MPI_Comm MPI_COMM_CART
                      ){

    int i, j;
    int neighbour_coords[2];
    int neighbour_rank;
    MPI_Request request[4] = {MPI_REQUEST_NULL, MPI_REQUEST_NULL,
        MPI_REQUEST_NULL, MPI_REQUEST_NULL};
    MPI_Status status;

    // Bottom border send
    if ((process_amounts[1] > 1) && !bottom_border) {
        for (i = 0; i < n_x; i++)
            b_send[i] = w[i+1][1];

        neighbour_coords[0] = my_coords[0];
        neighbour_coords[1] = my_coords[1] - 1;

        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Isend(b_send, n_x, MPI_DOUBLE,
                  neighbour_rank, tag + DOWN_TAG,
                  MPI_COMM_CART, &request[0]);
    }

    // Left border send
    if ((process_amounts[0] > 1) && !left_border) {
        for (j = 0; j < n_y; j++)
            l_send[j] = w[1][j+1];

        neighbour_coords[0] = my_coords[0] - 1;
        neighbour_coords[1] = my_coords[1];

        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Isend(l_send, n_y, MPI_DOUBLE,
                  neighbour_rank, tag,
                  MPI_COMM_CART, &request[1]);
    }

    // Top border
    if ((process_amounts[1] > 1) && !top_border) {
        for (i = 0; i < n_x; i++)
            t_send[i] = w[i+1][n_y];
```

```
        neighbour_coords[0] = my_coords[0];
        neighbour_coords[1] = my_coords[1] + 1;

        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Isend(t_send, n_x, MPI_DOUBLE,
                  neighbour_rank, tag,
                  MPI_COMM_CART, &request[2]);
    }

    // Right border
    if ((process_amounts[0] > 1) && !right_border) {
        for (j = 0; j < n_y; j++)
            r_send[j] = w[n_x][j+1];

        neighbour_coords[0] = my_coords[0] + 1;
        neighbour_coords[1] = my_coords[1];

        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Isend(r_send, n_y, MPI_DOUBLE,
                  neighbour_rank, tag,
                  MPI_COMM_CART, &request[3]);
    }

    // Receive borders
    // Bottom border
    if ((bottom_border && (process_amounts[1] > 1)) || (process_amounts[1] ==
         1)) {
        for (i = 1; i <= n_x; i++)
            w[i][0] = u_2(A1 + (x_idx + i - 1) * h1, B1 + (y_idx - 1) * h2);
    } else {
        neighbour_coords[0] = my_coords[0];
        neighbour_coords[1] = my_coords[1] - 1;
        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Recv(b_rec, n_x, MPI_DOUBLE,
                 neighbour_rank, tag, MPI_COMM_CART, &status);

        for (i = 1; i <= n_x; i++)
            w[i][0] = b_rec[i - 1];
    }

    // Left border
    if ((left_border && (process_amounts[0] > 1)) || (process_amounts[0] ==
        1)) {
        for (j = 1; j <= n_y; j++){
            w[0][j] = u_2(A1 + (x_idx - 1) * h1, B1 + (y_idx + j - 1) * h2);
        }

    } else {
        neighbour_coords[0] = my_coords[0] - 1;
        neighbour_coords[1] = my_coords[1];

        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Recv(l_rec, n_y, MPI_DOUBLE,
                 neighbour_rank, tag, MPI_COMM_CART, &status);

        for (j = 1; j <= n_y; j++)
            w[0][j] = l_rec[j - 1];
    }

    // Top border
```

```
    if ((top_border && (process_amounts[1] > 1)) || (process_amounts[1] == 1)
        ) {
        for (i = 1; i <= n_x; i++)
            w[i][n_y + 1] = u_2(A1 + (x_idx + i - 1) * h1, B1 + (y_idx + n_y)
                * h2);
    } else {
        neighbour_coords[0] = my_coords[0];
        neighbour_coords[1] = my_coords[1] + 1;
        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Recv(t_rec, n_x, MPI_DOUBLE,
                 neighbour_rank, tag + DOWN_TAG,
                 MPI_COMM_CART, &status);

        for (i = 1; i <= n_x; i++)
            w[i][n_y + 1] = t_rec[i - 1];
    }

    // Right border
    if ((right_border && (process_amounts[0] > 1)) || (process_amounts[0] ==
        1)) {
        for (j = 1; j <= n_y; j++)
            w[n_x + 1][j] = u_2(A1 + (x_idx + n_x)*h1, B1 + (y_idx + j - 1) *
                h2);
    } else {
        neighbour_coords[0] = my_coords[0] + 1;
        neighbour_coords[1] = my_coords[1];
        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Recv(r_rec, n_y, MPI_DOUBLE,
                 neighbour_rank, tag, MPI_COMM_CART, &status);

        for (j = 1; j <= n_y; j++)
            w[n_x + 1][j] = r_rec[j - 1];
    }

    for (int i = 0; i < 4; i++) {
        MPI_Wait(&request[i], &status);
    }
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Program receive %d numbers. Should be 2: M, N\n", argc);
        return -1;
    }

    int M = atoi(argv[argc - 2]);
    int N = atoi(argv[argc - 1]);
    if ((M <= 0) || (N <= 0)) {
        printf("M and N should be integer and > 0!!!\n");
        return -1;
    }
    printf("M = %d, N = %d\n", M, N);

    int my_rank;
    int n_processes;
    int process_amounts[2] = {0, 0};
    int write[1] = {0};

    double h1 = (A2 - A1) / M;
    double h2 = (B2 - B1) / N;
    double cur_eps = 1.0;

```

```
    MPI_Init(&argc, &argv);
    MPI_Status status;
    MPI_Request request;

    // For the cartesian topology
    MPI_Comm MPI_COMM_CART;
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &n_processes);

    // Creating rectangular supports
    MPI_Dims_create(n_processes, 2, process_amounts);

    printf("p_x = %d, p_y = %d\n", process_amounts[0], process_amounts[1]);
    int periods[2] = {0, 0};

    // Create cartesian topology in communicator
    MPI_Cart_create(MPI_COMM_WORLD, 2,
                    process_amounts, periods,
                    1, &MPI_COMM_CART);

    int my_coords[2];
    // Receive corresponding to rank process coordinates
    MPI_Cart_coords(MPI_COMM_CART, my_rank, 2, my_coords);

    int x_idx, n_x;
    get_idx_n_idx(&x_idx, &n_x, process_amounts[0], M+1, my_coords[0]);

    int y_idx, n_y;
    get_idx_n_idx(&y_idx, &n_y, process_amounts[1], N+1, my_coords[1]);

    double start_time = MPI_Wtime();

    // Create each block of size n_x and n_y with borders
    double *t_send = malloc(sizeof(double[n_x]));
    double *t_rec = malloc(sizeof(double[n_x]));
    double *b_send = malloc(sizeof(double[n_x]));
    double *b_rec = malloc(sizeof(double[n_x]));

    double *l_send = malloc(sizeof(double[n_y]));
    double *l_rec = malloc(sizeof(double[n_y]));
    double *r_send = malloc(sizeof(double[n_y]));
    double *r_rec = malloc(sizeof(double[n_y]));

    int i, j;
    int n_iters = 0;
    double block_eps;

    double (*w)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*w_pr)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*B)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double tau = 0;
    double global_tau = 0;
// double alpha_k, beta_k;
    double denumenator;
    double whole_denum;
    double global_alpha, global_beta;
    double eps_local, eps_r;

    double (*Aw)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*r_k)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*Ar)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*w_w_pr)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
```

```
    int left_border = 0;
    int top_border = 0;
    int right_border = 0;
    int bottom_border = 0;


    if (my_coords[0] == 0)
        left_border = 1;

    if (my_coords[0] == (process_amounts[0] - 1))
        right_border = 1;

    if (my_coords[1] == 0)
        bottom_border = 1;

    if (my_coords[1] == (process_amounts[1] - 1))
        top_border = 1;

    printf("L%d, R%d, T%d, B%d, 'x'%d, 'y'%d\n",
           left_border, right_border, top_border, bottom_border, my_coords[0],
               my_coords[1]);
    printf("%d %d\n", x_idx, y_idx);

    double (*Au)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*U)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    for (i = 0; i <= n_x + 1; i++)
        for (j = 0; j <= n_y + 1; j++)
            U[i][j] = u_2(A1 + (x_idx + i - 1) * h1, B1 + (y_idx + j - 1) * h2
                );

    Aw_mult(n_x, n_y, Au, U, h1, h2,
            A1 + x_idx * h1, B1 + y_idx * h2,
            left_border, right_border,
            top_border, bottom_border);

    B_right(n_x, n_y, B,
            h1, h2,
            A1 + x_idx * h1,
            B1 + y_idx * h2,
            left_border, right_border,
            top_border, bottom_border);

    double error_mean = 0;
    int amnt = 0;
    for (i = 1; i <= n_x; i++)
        for (j = 1; j <= n_y; j++){
            error_mean += fabs(Au[i][j] - B[i][j]);
            amnt += 1;
        }
    printf("ERROR FROM B = %3.2f\n", error_mean / amnt);

    for (i = 0; i <= n_x + 1; i++)
        for (j = 0; j <= n_y + 1; j++)
            w[i][j] = 0; // u_2(A1 + (x_idx + i - 1) * h1, B1 + (y_idx + j -
                1) * h2);

    int tag = 0;
    while ((cur_eps > EPS_REL) && (n_iters < MAX_ITER)) {
        if (my_rank == 0) {
            if (n_iters % 1000 == 0)
                printf("%g \n", cur_eps);
```

```
        }
        n_iters++;

        for (i = 0; i <= n_x + 1; i++) {
            for (j = 0; j <= n_y + 1; j++) {
                if (i == 0 || j == 0 || i == n_x + 1 || j == n_y + 1) {
                    w_pr[i][j] = 0;
                } else {
                    w_pr[i][j] = w[i][j];
                }
            }
        }

        send_recv_borders(n_x, n_y, process_amounts,
                          x_idx, y_idx, my_coords, tag,
                          w,
                          b_send, l_send, t_send, r_send,
                          b_rec, l_rec, t_rec, r_rec,
                          left_border, right_border,
                          top_border, bottom_border,
                          h1, h2, MPI_COMM_CART);

        Aw_mult(n_x, n_y,
                Aw, w,
                h1, h2,
                A1 + x_idx * h1, B1 + y_idx * h2,
                left_border, right_border,
                top_border, bottom_border);

        calculate_r(n_x, n_y, r_k, Aw, B);

        send_recv_borders(n_x, n_y, process_amounts,
                          x_idx, y_idx, my_coords, tag,
                          r_k,
                          b_send, l_send, t_send, r_send,
                          b_rec, l_rec, t_rec, r_rec,
                          left_border, right_border,
                          top_border, bottom_border,
                          h1, h2, MPI_COMM_CART);

        Aw_mult(n_x, n_y,
                Ar, r_k,
                h1, h2,
                A1 + x_idx * h1, B1 + y_idx * h2,
                left_border, right_border,
                top_border, bottom_border);

        tau = dot_product(n_x, n_y, Ar, r_k, h1, h2,
                          left_border, right_border,
                          top_border, bottom_border
                          );
        denumenator = dot_product(n_x, n_y, Ar, Ar, h1, h2,
                                  left_border, right_border,
                                  top_border, bottom_border);
        MPI_Allreduce(&tau, &global_tau, 1,
                      MPI_DOUBLE, MPI_SUM, MPI_COMM_CART);
        MPI_Allreduce(&denumenator, &whole_denum, 1,
                      MPI_DOUBLE, MPI_SUM, MPI_COMM_CART);
        global_tau = global_tau / whole_denum;


        for (i = 1; i <= n_x; i++)
```

```
            for (j = 1; j <= n_y; j++) {
                w[i][j] = w[i][j] - global_tau * r_k[i][j];
            }

        calculate_r(n_x, n_y, w_w_pr, w, w_pr);
        block_eps = norm(n_x, n_y, w_w_pr, h1, h2,
                         left_border, right_border,
                         top_border, bottom_border);

        MPI_Allreduce(&block_eps, &cur_eps, 1,
                      MPI_DOUBLE, MPI_SUM, MPI_COMM_CART);
    }

    // Waiting for all processes
    MPI_Barrier(MPI_COMM_WORLD);
    double end_time = MPI_Wtime();

    if (my_rank != 0) {
        MPI_Recv(write, 1, MPI_INT, my_rank - 1, 0, MPI_COMM_WORLD, &status);
    } else {
        printf("TIME = %f\n", end_time - start_time);
        printf("Number of iterations = %d\n", n_iters);
        printf("Tau = %f\n", tau);
        printf("Eps = %f\n", EPS_REL);
    }

// usleep(500);
    if (my_rank != n_processes - 1)
        MPI_Send(write, 1, MPI_INT, my_rank + 1, 0, MPI_COMM_WORLD);

    FILE *dim0, *dim1, *grid, *u_file, *true_u_file;
    char u_file_name[FILENAME_MAX];
    sprintf(u_file_name, "u_%d_%d.csv", my_coords[0], my_coords[1]);
    char true_u_file_name[FILENAME_MAX];
    sprintf(true_u_file_name, "true.u_%d_%d.csv", my_coords[0], my_coords[1]);

    dim0 = fopen("dim0.csv", "w");
    dim1 = fopen("dim1.csv", "w");
    grid = fopen("grid.csv", "w");
    u_file = fopen(u_file_name, "w");
    true_u_file = fopen(true_u_file_name, "w");


    for (int j = y_idx; j < y_idx + n_y; j++) {
        for (int i = x_idx; i < x_idx + n_x; i++) {
            fprintf(u_file, "%g ", w[i - x_idx + 1][j - y_idx + 1]);
            fprintf(true_u_file, "%g ", u_2(A1 + i*h1, B1 + j*h2));
        }
        fprintf(u_file, "\n");
        fprintf(true_u_file, "\n");
    }

    if (my_rank == 0) {
        for (int j = 0; j <= N; j++) {
            fprintf(dim0, "%g ", B1 + j*h2);
        }

        for (int i = 0; i <= M; i++) {
            fprintf(dim1, "%g ", A1 + i*h1);
        }
        fprintf(grid, "%d %d", process_amounts[1], process_amounts[0]);
```

```
    }
    fclose(dim0);
    fclose(dim1);
    fclose(grid);
    fclose(u_file);
    fclose(true_u_file);

    free(Au);
    free(U);
    free(w);
    free(w_pr);
    free(B);
    free(Ar);
    free(r_k);
    free(Aw);
    free(w_w_pr);

    free(t_send);
    free(t_rec);
    free(b_send);
    free(b_rec);
    free(r_send);
    free(r_rec);
    free(l_send);
    free(l_rec);
    MPI_Finalize();
    return 0;
}
```

Листинг 1: neyman_pde_mpi.c

9 Приложение 2. Код MPI программы с оптимизацией сопряжёнными градиентами

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
#include <time.h>
#include <math.h>
#include <unistd.h>


void print_matrix(int M, int N, double (*A)[N+2]){
    printf("Matrix:\n");
    for (int i=N+1; i>=0; i--){
        for (int j=0; j<=M+1; j++){
            printf("%3.2f ", A[i][j]);
        }
        printf("\n");
    }
}

void print_matrix_to_file(FILE *file,
                          int M, int N,
                          double (*A)[N+2]){
    for (int i=N+1; i>=0; i--){
        for (int j=0; j<=M+1; j++){
```

```
            fprintf(file,"%g ", A[i][j]);
        }
        fprintf(file, "\n");
    }
}

double u_2(double x, double y){
    return sqrt(4 + x * y);
}

double k_3(double x, double y){
    return 4 + x + y;
}

double q_2(double x, double y){
    double sum = x + y;
    if (sum < 0) {
        return 0;
    } else {
        return sum;
    }
}

double F(double x, double y){
    return ((pow(x, 3) - x*x*(y - 4) - x*(y*y + 8) +
    y*(y*y + 4*y - 8) + 4*q_2(x, y)*pow((4 + x*y), 2)) /
    (4 * pow((4 + x*y), 1.5)));
}

double psi_R(double x, double y){
    return (y*(4 + x + y) + 2*(4 + x*y)) / (2*sqrt(4 + x*y));
}


double psi_L(double x, double y){
    return (-y*(4 + x + y) + 2*(4 + x*y)) / (2*sqrt(4 + x*y));
}


double psi_T(double x, double y){
    return (x*(4 + x + y)) / (2*sqrt(4 + x*y));
}


double psi_B(double x, double y){
    return -psi_T(x, y);
}

double rho_1(int i,
             int M,
             int left_border,
             int right_border){
    if ((left_border && i == 1) || (right_border && i == M))
        return 0.5;
    return 1;
}

double rho_2(int j,
             int N,
             int bottom_border,
             int top_border){
    if ((bottom_border && j == 1) || (top_border && j == N))
```

```
        return 0.5;
    return 1;
}

double dot_product(int M, int N,
                   double (*U)[N + 2], double (*V)[N + 2],
                   double h1, double h2,
                   int left_border, int right_border,
                   int top_border, int bottom_border
                   ){
    double answer = 0.;
    double rho, r1, r2;

    for (int i=1; i <= M; i++){
        for (int j=1; j <= N; j++){
            r1 = rho_1(i, M, left_border, right_border);
            r2 = rho_2(j, N, bottom_border, top_border);
            rho = r1 * r2;
            answer += (rho * U[i][j] * V[i][j] * h1 * h2);
        }
    }
    return answer;
}


double norm(int M, int N, double (*U)[N + 2],
            double h1, double h2,
            int left_border, int right_border,
            int top_border, int bottom_border){
    return sqrt(dot_product(M, N, U, U, h1, h2,
                            left_border, right_border,
                            top_border, bottom_border));
}


void B_right(int M, int N, double (*B)[N+2],
       double h1, double h2,
       double x_start, double y_start,
       int left_border, int right_border,
       int top_border, int bottom_border){
    int i, j;

    for(i = 0; i <= M + 1; i++)
        for (j = 0; j <= N + 1; j++)
            B[i][j] = F(x_start + (i - 1) * h1, y_start + (j - 1) * h2);

    if (left_border){
        for (j = 1; j <= N; j++) {
            B[1][j] = (F(x_start, y_start + (j - 1) * h2) +
                    psi_L(x_start, y_start + (j - 1) * h2) * 2/h1);
        }
    }
    if (right_border){
        for (j = 1; j <= N; j++) {
            B[M][j] = (F(x_start + (M - 1)*h1, y_start + (j - 1) * h2) +
                       psi_R(x_start + (M - 1)*h1, y_start + (j - 1) * h2) * 2/
                           h1);
        }
    }
    if (top_border){
        for (i = 1; i <= M; i++) {
            B[i][N] = (F(x_start + (i - 1)*h1, y_start + (N - 1)*h2) +
```

```
                        psi_T(x_start + (i - 1)*h1, y_start + (N - 1)*h2) * 2/h2);
        }
    }

    if (bottom_border){
        for (i = 1; i <= M; i++) {
            B[i][1] = (F(x_start + (i - 1)*h1, y_start) +
                        psi_B(x_start + (i - 1)*h1, y_start) * 2/h2);
        }
    }
    if (left_border && top_border){
        B[1][N] = (F(x_start, y_start + (N - 1)*h2) +
                (2/h1 + 2/h2) * (psi_L(x_start, y_start + (N - 1)*h2) +
                                psi_T(x_start, y_start + (N - 1)*h2)) / 2);
    }
    if (left_border && bottom_border){
        B[1][1] = (F(x_start, y_start)
                + (2/h1 + 2/h2) * (psi_L(x_start, y_start) + psi_B(x_start, y_start)) / 2);
    }
    if (right_border && top_border){
        B[M][N] = (F(x_start + (M - 1)*h1, y_start + (N - 1)*h2) +
                (2/h1 + 2/h2) * (psi_R(x_start + (M - 1)*h1, y_start + (N - 1)*h2) +
                                psi_T(x_start + (M - 1)*h1, y_start + (N - 1)*h2)) / 2);
    }
    if (right_border && bottom_border){
        B[M][1] = (F(x_start + (M - 1)*h1, y_start) +
                (2/h1 + 2/h2) * (psi_R(x_start + (M - 1)*h1, y_start) +
                                psi_B(x_start + (M - 1)*h1, y_start)) / 2);
    }
}


double aw_x_ij(int N,
             double (*w)[N+2],
             double x_start, double y_start,
             int i, int j,
             double h1, double h2
             ){
    return (1/h1) * (k_3(x_start + (i + 0.5 - 1) * h1,y_start + (j - 1) * h2) * (w[i + 1][j] - w[i][j]) / h1
                  - k_3(x_start + (i - 0.5 - 1) * h1,y_start + (j - 1) * h2) * (w[i][j] - w[i - 1][j]) / h1);
}

double aw_ij(int N,
              double (*w)[N+2],
              double x_start, double y_start,
              int i, int j,
              double h1, double h2
){
    return (k_3(x_start + (i - 0.5 - 1) * h1,y_start + (j - 1) * h2) * (w[i][j] - w[i - 1][j]) / h1);
}

double bw_y_ij(int N,
             double (*w)[N+2],
             double x_start, double y_start,
             int i, int j,
```

```
            double h1, double h2
){
    return (1/h2) * (k_3(x_start + (i - 1) * h1,y_start + (j + 0.5 - 1) * h2)
         * (w[i][j + 1] - w[i][j]) / h2
                  - k_3(x_start + (i - 1) * h1,y_start + (j - 0.5 - 1) * h2) *
                      (w[i][j] - w[i][j - 1]) / h2);
}

double bw_ij(int N,
              double (*w)[N+2],
              double x_start, double y_start,
              int i, int j,
              double h1, double h2
){
    return (k_3(x_start + (i - 1) * h1,y_start + (j - 0.5 - 1) * h2) * (w[i][
        j] - w[i][j-1]) / h2);
}

void Aw_mult(int M, int N,
            double (*A)[N+2], double (*w)[N+2],
            double h1, double h2,
            double x_start, double y_start,
            int left_border, int right_border,
            int top_border, int bottom_border
            )
{
    double aw_x, bw_y;
    int i, j;
    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){
                A[i][j] = w[i][j];
            } else {
                aw_x = aw_x_ij(N, w, x_start, y_start, i, j, h1, h2);
                bw_y = bw_y_ij(N, w, x_start, y_start, i, j, h1, h2);
                A[i][j] = -aw_x - bw_y + q_2(x_start + (i - 1) * h1,
                                          y_start + (j - 1) * h2) * w[i][j];
            }
        }
    }

    // Left interior border filling
    if (left_border){
        for (j = 1; j <= N; j++) {
            aw_x = aw_ij(N, w, x_start, y_start, 2, j , h1, h2);
            bw_y = bw_y_ij(N, w, x_start, y_start, 1, j, h1, h2);
            A[1][j] = -2*aw_x / h1 - bw_y + (q_2(x_start,
                                              y_start + (j - 1) * h2) + 2/h1) *
                                                   w[1][j];
        }
    }

    // Right interior border
    if (right_border){
        for (j = 1; j <= N; j++) {
            aw_x = aw_ij(N, w, x_start, y_start, M, j, h1, h2);
            bw_y = bw_y_ij(N, w, x_start, y_start, M, j, h1, h2);
            A[M][j] = 2*aw_x / h1 - bw_y + (q_2(x_start + (M - 1) * h1,
                                             y_start + (j - 1) * h2) + 2/h1) *
                                                 w[M][j];
        }
    }
```

```

    // Top border
    if (top_border){
        for (i = 1; i <= M; i++) {
            aw_x = aw_x_ij(N, w, x_start, y_start, i, N, h1, h2);
            bw_y = bw_ij(N, w, x_start, y_start, i, N, h1, h2);
            A[i][N] = -aw_x + 2*bw_y / h2 + q_2(x_start + (i - 1) * h1,
                                                y_start + (N - 1) * h2) * w[i][N];
        }
    }

    // Bottom border
    if (bottom_border){
        for (i = 1; i <= M; i++) {
            aw_x = aw_x_ij(N, w, x_start, y_start, i, 1, h1, h2);
            bw_y = bw_ij(N, w, x_start, y_start, i, 2, h1, h2);
            A[i][1] = -aw_x - 2*bw_y / h2 + q_2(x_start + (i - 1)* h1, y_start
                ) * w[i][1];
        }
    }
    if (left_border && bottom_border){
        aw_x = aw_ij(N, w, x_start, y_start, 2, 1, h1, h2);
        bw_y = bw_ij(N, w, x_start, y_start, 1, 2, h1, h2);
        A[1][1] = -2*aw_x / h1 - 2*bw_y / h2 + (q_2(x_start, y_start) + 2/h1)
            * w[1][1];
    }
    if (left_border && top_border){
        aw_x = aw_ij(N, w, x_start, y_start, 2, N, h1, h2);
        bw_y = bw_ij(N, w, x_start, y_start, 1, N, h1, h2);
        A[1][N] = -2*aw_x / h1 + 2*bw_y / h2 + (q_2(x_start, y_start + (N - 1)
             * h2) + 2/h1)* w[1][N];
    }

    if (right_border && bottom_border){
        aw_x = aw_ij(N, w, x_start, y_start, M, 1, h1, h2);
        bw_y = bw_ij(N, w, x_start, y_start, M, 2, h1, h2);
        A[M][1] = 2*aw_x / h1 - 2 * bw_y / h2 + (q_2(x_start + (M - 1) * h1,
            y_start) + 2/h1) * w[M][1];
    }
    if (right_border && top_border) {
        aw_x = aw_ij(N, w, x_start, y_start, M, N, h1, h2);
        bw_y = bw_ij(N, w, x_start, y_start, M, N, h1, h2);
        A[M][N] = 2*aw_x / h1 + 2 * bw_y / h2 + (q_2(x_start + (M - 1) * h1,
                                                     y_start + (N - 1) * h2) + 2/
                                                         h1) * w[M][N];
    }
}


void calculate_r(int M, int N,
                 double (*r)[N+2],
                 double (*Aw)[N+2],
                 double (*B)[N+2]
                 ){
    int i, j;

    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)
                r[i][j] = 0;
            else
                r[i][j] = Aw[i][j] - B[i][j];
```

```
        }
    }
}


void get_idx_n_idx(int *idx,
                   int *n_idx,
                   int process_amnt,
                   int grid_size,
                   int coordinate){
    if (grid_size % process_amnt == 0) {
        *n_idx = grid_size / process_amnt;
        *idx = coordinate * (grid_size / process_amnt);
    }
    else
    {
        if (coordinate == 0){
            *n_idx = grid_size % process_amnt + grid_size / process_amnt;
            *idx = 0;
        } else
        {
            *n_idx = grid_size / process_amnt;
            *idx = grid_size % process_amnt + coordinate * (grid_size /
                process_amnt);
        }
    }
}


#define A1 0.0
#define A2 4.0
#define B1 0.0
#define B2 3.0
#define EPS_REL 1e-6
#define DOWN_TAG 1000
#define MAX_ITER 100000


void send_recv_borders(int n_x, int n_y,
                       const int process_amounts[2],
                       double x_idx,
                       double y_idx,
                       const int my_coords[2],
                       int tag,
                       double (*w)[n_y+2],
                       double b_send[n_x],
                       double l_send[n_y],
                       double t_send[n_x],
                       double r_send[n_y],
                       double b_rec[n_x],
                       double l_rec[n_y],
                       double t_rec[n_x],
                       double r_rec[n_y],
                       int left_border, int right_border,
                       int top_border, int bottom_border,
                       double h1, double h2,
                       MPI_Comm MPI_COMM_CART
){

    int i, j;
    int neighbour_coords[2];
    int neighbour_rank;
```

```
    MPI_Request request[4] = {MPI_REQUEST_NULL, MPI_REQUEST_NULL,
        MPI_REQUEST_NULL, MPI_REQUEST_NULL};
    MPI_Status status;

    // Bottom border send
    if ((process_amounts[1] > 1) && !bottom_border) {
        for (i = 0; i < n_x; i++)
            b_send[i] = w[i+1][1];

        neighbour_coords[0] = my_coords[0];
        neighbour_coords[1] = my_coords[1] - 1;

        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Isend(b_send, n_x, MPI_DOUBLE,
                  neighbour_rank, tag + DOWN_TAG,
                  MPI_COMM_CART, &request[0]);
    }

    // Left border send
    if ((process_amounts[0] > 1) && !left_border) {
        for (j = 0; j < n_y; j++)
            l_send[j] = w[1][j+1];

        neighbour_coords[0] = my_coords[0] - 1;
        neighbour_coords[1] = my_coords[1];

        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Isend(l_send, n_y, MPI_DOUBLE,
                  neighbour_rank, tag,
                  MPI_COMM_CART, &request[1]);
    }

    // Top border
    if ((process_amounts[1] > 1) && !top_border) {
        for (i = 0; i < n_x; i++)
            t_send[i] = w[i+1][n_y];

        neighbour_coords[0] = my_coords[0];
        neighbour_coords[1] = my_coords[1] + 1;

        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Isend(t_send, n_x, MPI_DOUBLE,
                  neighbour_rank, tag,
                  MPI_COMM_CART, &request[2]);
    }

    // Right border
    if ((process_amounts[0] > 1) && !right_border) {
        for (j = 0; j < n_y; j++)
            r_send[j] = w[n_x][j+1];

        neighbour_coords[0] = my_coords[0] + 1;
        neighbour_coords[1] = my_coords[1];

        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Isend(r_send, n_y, MPI_DOUBLE,
                  neighbour_rank, tag,
                  MPI_COMM_CART, &request[3]);
    }

    // Receive borders
    // Bottom border
```

```
    if ((bottom_border && (process_amounts[1] > 1)) || (process_amounts[1] ==
         1)) {
        for (i = 1; i <= n_x; i++)
            w[i][0] = u_2(A1 + (x_idx + i - 1) * h1, B1 + (y_idx - 1) * h2);
    } else {
        neighbour_coords[0] = my_coords[0];
        neighbour_coords[1] = my_coords[1] - 1;
        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Recv(b_rec, n_x, MPI_DOUBLE,
                 neighbour_rank, tag, MPI_COMM_CART, &status);

        for (i = 1; i <= n_x; i++)
            w[i][0] = b_rec[i - 1];
    }

    // Left border
    if ((left_border && (process_amounts[0] > 1)) || (process_amounts[0] ==
        1)) {
        for (j = 1; j <= n_y; j++){
            w[0][j] = u_2(A1 + (x_idx - 1) * h1, B1 + (y_idx + j - 1) * h2);
        }

    } else {
        neighbour_coords[0] = my_coords[0] - 1;
        neighbour_coords[1] = my_coords[1];

        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Recv(l_rec, n_y, MPI_DOUBLE,
                 neighbour_rank, tag, MPI_COMM_CART, &status);

        for (j = 1; j <= n_y; j++)
            w[0][j] = l_rec[j - 1];
    }

    // Top border
    if ((top_border && (process_amounts[1] > 1)) || (process_amounts[1] == 1)
        ) {
        for (i = 1; i <= n_x; i++)
            w[i][n_y + 1] = u_2(A1 + (x_idx + i - 1) * h1, B1 + (y_idx + n_y)
                * h2);
    } else {
        neighbour_coords[0] = my_coords[0];
        neighbour_coords[1] = my_coords[1] + 1;
        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
        MPI_Recv(t_rec, n_x, MPI_DOUBLE,
                 neighbour_rank, tag + DOWN_TAG,
                 MPI_COMM_CART, &status);

        for (i = 1; i <= n_x; i++)
            w[i][n_y + 1] = t_rec[i - 1];
    }

    // Right border
    if ((right_border && (process_amounts[0] > 1)) || (process_amounts[0] ==
        1)) {
        for (j = 1; j <= n_y; j++)
            w[n_x + 1][j] = u_2(A1 + (x_idx + n_x)*h1, B1 + (y_idx + j - 1) *
                h2);
    } else {
        neighbour_coords[0] = my_coords[0] + 1;
        neighbour_coords[1] = my_coords[1];
        MPI_Cart_rank(MPI_COMM_CART, neighbour_coords, &neighbour_rank);
```

```
        MPI_Recv(r_rec, n_y, MPI_DOUBLE,
                 neighbour_rank, tag, MPI_COMM_CART, &status);

        for (j = 1; j <= n_y; j++)
            w[n_x + 1][j] = r_rec[j - 1];
    }

    for (int i = 0; i < 4; i++) {
        MPI_Wait(&request[i], &status);
    }
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Program receive %d numbers. Should be 2: M, N\n", argc);
        return -1;
    }

    int M = atoi(argv[argc - 2]);
    int N = atoi(argv[argc - 1]);
    if ((M <= 0) || (N <= 0)) {
        printf("M and N should be integer and > 0!!!\n");
        return -1;
    }
    printf("M = %d, N = %d\n", M, N);

    int my_rank;
    int n_processes;
    int process_amounts[2] = {0, 0};
    int write[1] = {0};

    double h1 = (A2 - A1) / M;
    double h2 = (B2 - B1) / N;
    double cur_eps = 1.0;

    MPI_Init(&argc, &argv);
    MPI_Status status;
    MPI_Request request;

    // For the cartesian topology
    MPI_Comm MPI_COMM_CART;
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &n_processes);

    // Creating rectangular supports
    MPI_Dims_create(n_processes, 2, process_amounts);

    printf("p_x = %d, p_y = %d\n", process_amounts[0], process_amounts[1]);
    int periods[2] = {0, 0};

    // Create cartesian topology in communicator
    MPI_Cart_create(MPI_COMM_WORLD, 2,
                    process_amounts, periods,
                    1, &MPI_COMM_CART);

    int my_coords[2];
    // Receive corresponding to rank process coordinates
    MPI_Cart_coords(MPI_COMM_CART, my_rank, 2, my_coords);

    int x_idx, n_x;
    get_idx_n_idx(&x_idx, &n_x, process_amounts[0], M+1, my_coords[0]);

```

```
    int y_idx, n_y;
    get_idx_n_idx(&y_idx, &n_y, process_amounts[1], N+1, my_coords[1]);

    double start_time = MPI_Wtime();

    // Create each block of size n_x and n_y with borders
    double *t_send = malloc(sizeof(double[n_x]));
    double *t_rec = malloc(sizeof(double[n_x]));
    double *b_send = malloc(sizeof(double[n_x]));
    double *b_rec = malloc(sizeof(double[n_x]));

    double *l_send = malloc(sizeof(double[n_y]));
    double *l_rec = malloc(sizeof(double[n_y]));
    double *r_send = malloc(sizeof(double[n_y]));
    double *r_rec = malloc(sizeof(double[n_y]));

    int i, j;
    int n_iters = 0;
    double block_eps;

    double (*w)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*w_pr)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*B)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double tau = 0;
    double global_tau = 0;
    double alpha_k, beta_k;
    double denumenator;
    double whole_denum;
    double global_alpha, global_beta;
    double eps_local, eps_r;

    double (*Aw)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*r_k_1)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*r_k)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*Ar)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*z)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*Az)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*w_w_pr)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));

    int left_border = 0;
    int top_border = 0;
    int right_border = 0;
    int bottom_border = 0;


    if (my_coords[0] == 0)
        left_border = 1;

    if (my_coords[0] == (process_amounts[0] - 1))
        right_border = 1;

    if (my_coords[1] == 0)
        bottom_border = 1;

    if (my_coords[1] == (process_amounts[1] - 1))
        top_border = 1;

    printf("L%d, R%d, T%d, B%d, 'x'%d, 'y'%d\n",
          left_border, right_border, top_border, bottom_border, my_coords[0],
               my_coords[1]);
    printf("%d %d\n", x_idx, y_idx);
```

```
    double (*Au)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    double (*U)[n_y + 2] = malloc(sizeof(double[n_x + 2][n_y + 2]));
    for (i = 0; i <= n_x + 1; i++)
        for (j = 0; j <= n_y + 1; j++)
            U[i][j] = u_2(A1 + (x_idx + i - 1) * h1, B1 + (y_idx + j - 1) * h2
                );

    Aw_mult(n_x, n_y, Au, U, h1, h2,
            A1 + x_idx * h1, B1 + y_idx * h2,
            left_border, right_border,
            top_border, bottom_border);

    B_right(n_x, n_y, B,
            h1, h2,
            A1 + x_idx * h1,
            B1 + y_idx * h2,
            left_border, right_border,
            top_border, bottom_border);
    double norm_b, all_norm_b;
    norm_b = dot_product(n_x, n_y, B, B, h1, h2,
                         left_border, right_border,
                         top_border, bottom_border);
    MPI_Allreduce(&norm_b, &all_norm_b, 1,
                  MPI_DOUBLE, MPI_SUM, MPI_COMM_CART);
    all_norm_b = sqrt(all_norm_b);

    double error_mean = 0;
    int amnt = 0;
    for (i = 1; i <= n_x; i++)
        for (j = 1; j <= n_y; j++){
            error_mean += fabs(Au[i][j] - B[i][j]);
            amnt += 1;
        }
    printf("ERROR FROM B = %3.2f\n", error_mean / amnt);

    for (i = 0; i <= n_x + 1; i++)
        for (j = 0; j <= n_y + 1; j++)
            w[i][j] = 0; // u_2(A1 + (x_idx + i - 1) * h1, B1 + (y_idx + j -
                1) * h2);

    int tag = 0;
    while ((cur_eps > EPS_REL) && (n_iters < MAX_ITER)) {
        if (my_rank == 0) {
            if (n_iters % 1000 == 0)
                printf("%g \n", cur_eps);
        }
        n_iters++;

        for (i = 0; i <= n_x + 1; i++) {
            for (j = 0; j <= n_y + 1; j++) {
                if (i == 0 || j == 0 || i == n_x + 1 || j == n_y + 1) {
                    w_pr[i][j] = 0;
                } else {
                    w_pr[i][j] = w[i][j];
                }
            }
        }

        send_recv_borders(n_x, n_y, process_amounts,
                          x_idx, y_idx, my_coords, tag,
                          w,
                          b_send, l_send, t_send, r_send,
```

```
                         b_rec, l_rec, t_rec, r_rec,
                         left_border, right_border,
                         top_border, bottom_border,
                         h1, h2, MPI_COMM_CART);

        Aw_mult(n_x, n_y,
                Aw, w,
                h1, h2,
                A1 + x_idx * h1, B1 + y_idx * h2,
                left_border, right_border,
                top_border, bottom_border);

// Make initialization for CG
        if (n_iters == 1) {
            calculate_r(n_x, n_y, r_k_1, B, Aw);

            send_recv_borders(n_x, n_y, process_amounts, x_idx, y_idx,
                my_coords, tag,
                              r_k_1,
                              b_send, l_send, t_send, r_send,
                              b_rec, l_rec, t_rec, r_rec,
                              left_border, right_border, top_border,
                                  bottom_border,
                              h1, h2, MPI_COMM_CART);
            for (i = 0; i <= n_x + 1; i++)
                for (j = 0; j <= n_y + 1; j++)
                    z[i][j] = r_k_1[i][j];
        }

        Aw_mult(n_x, n_y,
                Ar, r_k_1,
                h1, h2,
                A1 + x_idx * h1, B1 + y_idx * h2,
                left_border, right_border,
                top_border, bottom_border);

        Aw_mult(n_x, n_y,
                Az, z,
                h1, h2,
                A1 + x_idx * h1, B1 + y_idx * h2,
                left_border, right_border,
                top_border, bottom_border);

        alpha_k = dot_product(n_x, n_y, r_k_1, r_k_1, h1, h2,
                              left_border, right_border,
                              top_border, bottom_border);
        denumenator = dot_product(n_x, n_y, Az, z, h1, h2,
                                  left_border, right_border,
                                  top_border, bottom_border);

        MPI_Allreduce(&alpha_k, &global_alpha, 1,
                      MPI_DOUBLE, MPI_SUM, MPI_COMM_CART);
        MPI_Allreduce(&denumenator, &whole_denum, 1,
                      MPI_DOUBLE, MPI_SUM, MPI_COMM_CART);
        global_alpha = global_alpha / whole_denum;

        for (i = 1; i <= n_x; i++)
            for (j = 1; j <= n_y; j++) {
                w[i][j] = w[i][j] + global_alpha * z[i][j];
                r_k[i][j] = r_k_1[i][j] - global_alpha * Az[i][j];
            }

```

```
        send_recv_borders(n_x, n_y, process_amounts, x_idx, y_idx, my_coords,
            tag,
                          r_k,
                          b_send, l_send, t_send, r_send,
                          b_rec, l_rec, t_rec, r_rec,
                          left_border, right_border, top_border, bottom_border,
                          h1, h2, MPI_COMM_CART);
        beta_k = dot_product(n_x, n_y, r_k, r_k, h1, h2,
                             left_border, right_border,
                             top_border, bottom_border
                             );
        denumenator = dot_product(n_x, n_y, r_k_1, r_k_1, h1, h2,
                                  left_border, right_border,
                                  top_border, bottom_border);

        MPI_Allreduce(&beta_k, &global_beta, 1,
                      MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
        MPI_Allreduce(&denumenator, &whole_denum, 1,
                      MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
        global_beta = global_beta / whole_denum;

        for (i = 1; i <= n_x; i++)
            for (j = 1; j <= n_y; j++) {
                z[i][j] = r_k[i][j] + global_beta * z[i][j];
                r_k_1[i][j] = r_k[i][j];
            }
        send_recv_borders(n_x, n_y, process_amounts, x_idx, y_idx, my_coords,
            tag,
                          z,
                          b_send, l_send, t_send, r_send,
                          b_rec, l_rec, t_rec, r_rec,
                          left_border, right_border, top_border, bottom_border,
                          h1, h2, MPI_COMM_CART);

        block_eps = dot_product(n_x, n_y, r_k_1, r_k_1, h1, h2,
                                left_border, right_border,
                                top_border, bottom_border);
        MPI_Allreduce(&block_eps, &cur_eps, 1,
                      MPI_DOUBLE, MPI_SUM, MPI_COMM_CART);
        cur_eps = sqrt(cur_eps) / all_norm_b;
    }

    // Waiting for all processes
    MPI_Barrier(MPI_COMM_WORLD);
    double end_time = MPI_Wtime();

    if (my_rank != 0) {
        MPI_Recv(write, 1, MPI_INT, my_rank - 1, 0, MPI_COMM_WORLD, &status);
    } else {
        printf("TIME = %f\n", end_time - start_time);
        printf("Number of iterations = %d\n", n_iters);
        printf("Tau = %f\n", tau);
        printf("Eps = %f\n", EPS_REL);
    }

// usleep(500);
    if (my_rank != n_processes - 1)
        MPI_Send(write, 1, MPI_INT, my_rank + 1, 0, MPI_COMM_WORLD);

    FILE *dim0, *dim1, *grid, *u_file, *true_u_file;
    char u_file_name[FILENAME_MAX];
    sprintf(u_file_name, "u_%d_%d.csv", my_coords[0], my_coords[1]);
```

```c
    char true_u_file_name[FILENAME_MAX];
    sprintf(true_u_file_name, "true.u_%d_%d.csv", my_coords[0], my_coords[1])
        ;

    dim0 = fopen("dim0.csv", "w");
    dim1 = fopen("dim1.csv", "w");
    grid = fopen("grid.csv", "w");
    u_file = fopen(u_file_name, "w");
    true_u_file = fopen(true_u_file_name, "w");


    for (int j = y_idx; j < y_idx + n_y; j++) {
        for (int i = x_idx; i < x_idx + n_x; i++) {
            fprintf(u_file, "%g ", w[i - x_idx + 1][j - y_idx + 1]);
            fprintf(true_u_file, "%g ", u_2(A1 + i*h1, B1 + j*h2));
        }
        fprintf(u_file, "\n");
        fprintf(true_u_file, "\n");
    }

    if (my_rank == 0) {
        for (int j = 0; j <= N; j++) {
            fprintf(dim0, "%g ", B1 + j*h2);
        }

        for (int i = 0; i <= M; i++) {
            fprintf(dim1, "%g ", A1 + i*h1);
        }
        fprintf(grid, "%d %d", process_amounts[1], process_amounts[0]);
    }
    fclose(dim0);
    fclose(dim1);
    fclose(grid);
    fclose(u_file);
    fclose(true_u_file);

    free(Au);
    free(U);
    free(w);
    free(w_pr);
    free(B);
    free(Az);
    free(z);
    free(Ar);
    free(r_k);
    free(r_k_1);
    free(Aw);
    free(w_w_pr);

    free(t_send);
    free(t_rec);
    free(b_send);
    free(b_rec);
    free(r_send);
    free(r_rec);
    free(l_send);
    free(l_rec);
    MPI_Finalize();
    return 0;
}
```

Листинг 2: neyman_pde_mpi_cg.c