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

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

в рамках курса «Суперкомпьютерно моделирование и технологии»

Вариант 4

Выполнила: Цянь Чэнсыцзинь, 614 группа

Москва
2022

Содержание

1 Постановка задачи.

В прямоугольнике $\Pi = \{A_1 \le x \le A_2,\ B_1 \le y \le B_2\}$, граница Γ состоит из отрезков

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

рассматривается дифференциальное уравнение Пуассона с потенциалом

$$-\Delta u + q(x,y)u = F(x,y), \tag{1.1}$$ eq_1.

в котором оператор Лапласа

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

Для выделения единственного решения уравнение (1.1) дополняется граничными условиями. На каждом отрезке границы прямоугольника Π задаются следующие условия:

1. условия первого типа (условия Дирихле):

$$u(x,y) = \varphi(x,y); \tag{1.2}$$ eq_1.

2. условия третьего типа:

$$\Big(k\,\frac{\partial u}{\partial n}\Big)(x,y) + \alpha u(x,y) = \psi(x,y), \tag{1.3}$$ eq_1.

где n – единичная внешняя нормаль к границе прямоугольника.

Функции $F(x,y)$, $\varphi(x,y)$, $\psi(x,y)$, коэффициент $k(x,y)$, потенциал $q(x,y)$ и параметр $\alpha \ge 0$ считаются известными. Требуется найти функцию $u(x,y)$, удовлетворяющую уравнению (1.1) и граничным условиям.

В соответствии с моим вариантом (**вариант 4**) задания рассматриваю следующие данные:

- $A_1 = 0,\ A_2 = 2,\ B_1 = 0,\ B_2 = 1,$
- $u(x,y) = u_4(x,y) = 1 + \cos(\pi x y),$
- $k(x,y) = k_3(x,y) = 4 + x + y,$
- $q(x,y) = q_0(x,y) = 0,$
- граничные условия $\gamma_R = (1.3)$, $\gamma_L = (1.2)$, $\gamma_T = (1.3)$, $\gamma_B = (1.2)$.

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.*

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

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

$$\frac{\partial u}{\partial x} = -\pi y \sin(\pi xy), \quad \frac{\partial u}{\partial y} = -\pi x \sin(\pi xy)$$

$$\Delta u = \frac{\partial}{\partial x}\Big((4+x+y)\cdot\big(-\pi y\sin(\pi xy)\big)\Big) + \frac{\partial}{\partial y}\Big((4+x+y)\cdot\big(-\pi x\sin(\pi xy)\big)\Big) =$$

$$= -(\pi y)^2\big(4+x+y\big)\cos(\pi xy) - (\pi x)^2\big(4+x+y\big)\cos(\pi xy) - \pi y\sin(\pi xy) - \pi x\sin(\pi xy).$$

Учитывая, что $q(x,y) = 0, u = 1+\cos(\pi xy)$, то

$$F(x,y) = \pi^2 y^2\big(4+x+y\big)\cos(\pi xy) + \pi^2 x^2\big(4+x+y\big)\cos(\pi xy) + \pi y\sin(\pi xy) + \pi x\sin(\pi xy). \quad (2.1)$$

eq_2.

левое γ_L и нижнее γ_B граничное условие:

$$\gamma_L = u(0,y) = \varphi(0,y) = 2, \quad \gamma_B = u(x,0) = \varphi(x,0) = 2,$$

правое γ_R граничное условие (n = x):

$$\gamma_R = \psi(2,y) = (6+y)\big(-\pi y\sin(2\pi y)\big) + 1 + \cos(2\pi y),$$

верхнее γ_T граничное условие (n = y):

$$\gamma_T = \psi(x,1) = (5+x)\big(-\pi x\sin(\pi x)\big) + 1 + \cos(\pi x).$$

3 Разностная схема решения задачи.

Краевые задачи для уравнения Пуассона с потенциалом (1.1) предлагается численно решать методом конечных разностей. В расчетной области Π определяется равномерная прямоугольная сетка $\bar{\omega}_h = \bar{\omega}_1 \times \bar{\omega}_2$, где

$$\bar{\omega}_1 = \{x_i = A_1 + ih_1,\ i = \overline{0,M}\},\ \bar{\omega}_2 = \{y_j = B1 + jh_2,\ j = \overline{0,N}\}.$$

Здесь $h_1 = (A_2 - A_1)/M$, $h_2 = (B_2 - B_1)/N$. Через ω_h обозначим множество внутренних узлов сетки $\bar{\omega}_h$.

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

пространстве H задано скалярное произведение и евклидова норма

$$\big[u,v\big]=\sum_{i=0}^{M}h_1\sum_{j=0}^{N}h_2\rho_{ij}u_{ij}v_{ij}=h_1h_2\sum_{i=0}^{M}\sum_{j=0}^{N}\rho_{ij}u_{ij}v_{ij},\quad \|u\|_E=\sqrt{[u,u]}. \tag{3.1}$$ eq_3.

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

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

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

$$Aw\ =\ B, \tag{3.2}$$ eq_3.

где $A:H\to H$ – оператор, действующий в пространстве сеточных функций, $B\in H$ – известная правая часть. Задача (3.2) 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 (1.1) во всех внутренних точках сетки аппроксимируется разностным уравнением

$$-\Delta_h w_{ij}+q_{ij}w_{ij}=F_{ij},\quad i=\overline{1,M-1},\ j=\overline{1,N-1}, \tag{3.3}$$ eq_3.

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

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

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

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

а также определим сеточные коэффициенты

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

С учетом принятых обозначений разностный оператор Лапласа можно представить в более компактном и удобном виде $\Delta_h w_{ij} = \big(aw_{\overline{x}}\big)_{x,ij} + \big(bw_{\overline{y}}\big)_{y,ij}$.

Аппроксимация граничных условий первого типа имеет вид:

$$w_{ij} = \varphi(x_i, y_j). \tag{3.4}$$

eq_3.

Замечание. Переменные w_{ij}, заданные равенством (3.4), исключаются из разностной схемы, а соответствующие узлы $P_{ij}(x_i, y_j)$ – из расчетной сетки $\overline{\omega}_h$. В скалярном произведении (3.1) слагаемые, отвечающие данным граничным узлам, считаются равными нулю.

Аппроксимация граничных условий третьего типа на правой и верхней сторонах прямоугольника имеет вид:

$$\begin{aligned}
&(2/h_1)(aw_{\overline{x}})_{Mj} + \big(q_{Mj} + 2\alpha_R/h_1\big)w_{Mj} - \big(bw_{\overline{y}}\big)_{y,Mj} = F_{Mj} + (2/h_1)\psi_{Mj},\ j = \overline{1, N-1}.\\
&(2/h_2)(bw_{\overline{y}})_{iN} + \big(q_{iN} + 2\alpha_T/h_2\big)w_{iN} - \big(aw_{\overline{x}}\big)_{x,iN} = F_{iN} + (2/h_2)\psi_{iN},\ i = \overline{1, M-1}.
\end{aligned} \tag{3.5}$$

eq_3.

Здесь α_R, α_T– параметры в граничных условиях третьего типа, которые мы будем считать неизменными вдоль отрезков γ_R, γ_T соответственно.

Сеточных уравнений (3.3)-(3.5) недостаточно, чтобы определить разностную схему для задачи с граничными условиями (1.2), (1.3). Требуются сеточные уравнения для угловой точки (A_2, B_2) прямоугольника П. Она имеет следующий вид:

$$\begin{aligned}
(2/h_1)(aw_{\overline{x}})_{MN} + (2/h_2)(bw_{\overline{y}})_{MN} + \big(q_{MN} + 2\alpha_R/h_1 + 2\alpha_T/h_2\big)w_{MN} =\\
= F_{MN} + \big(2/h_1 + 2/h_2\big)\psi_{MN}
\end{aligned} \tag{3.6}$$

eq_3.

– в вершине $P(A_2, B_2)$ прямоугольника. Здесь $\psi_{MN} = \frac{h_1\psi(A_2-0,B_2)+h_2\psi(A_2,B_2-0)}{h_1+h_2}$, где $\psi(x_0 \pm 0, y) = \lim\limits_{x\to x_0\pm 0}\psi(x,y), \psi(x, y_0 \pm 0) = \lim\limits_{y\to y_0\pm 0}\psi(x,y)$.

Замечание. Разностные схемы (3.2), 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.

Замечание. В краевых условиях третьего типа числа α_R, α_T всюду считать равными единице.

Соберём все уравнения, получим систему линейных алгебраических уравнений (СЛАУ),

состоящую из $(M)\times(N)$ уравнений и $(M)\times(N)$ неизвестной:

$$\begin{cases}
-\Delta_h w_{ij} + q_{ij}w_{ij} = F_{ij}, & i=\overline{2,M-1},\ j=\overline{2,N-1}\\
-(aw_{\bar{x}})_{x,i1} - \frac{1}{h_2}\left[(bw_{\bar{y}})_{i2} - \frac{1}{h_2}b_{i1}w_{i1}\right] + q_{i1}w_{i1} = F_{i1} + \frac{b_{i1}}{h_2^2}\varphi_{i0}, & i=\overline{2,M-1},\ j=1\\
-(bw_{\bar{y}})_{y,1j} - \frac{1}{h_1}\left[(aw_{\bar{x}})_{2j} - \frac{1}{h_1}a_{1j}w_{1j}\right] + q_{1j}w_{1j} = F_{1j} + \frac{a_{1j}}{h_1^2}\varphi_{0j}, & i=1, \quad j=\overline{2,N-1}\\
\frac{2}{h_2}(bw_{\bar{y}})_{iN} + (q_{iN} + \frac{2}{h_2})w_{iN} - (aw_{\bar{x}})_{x,iN} = F_{iN} + \frac{2}{h_2}\psi_{iN}, & i=\overline{2,M-1},\ j=N\\
\frac{2}{h_2}(aw_{\bar{x}})_{Mj} + (q_{Mj} + \frac{2}{h_1})w_{Mj} - (bw_{\bar{y}})_{y,Mj} = F_{Mj} + \frac{2}{h_1}\psi_{Mj}, & i=M, \quad j=\overline{2,N-1}\\
\frac{2}{h_1}(aw_{\bar{x}})_{MN} + \frac{2}{h_2}(bw_{\bar{y}})_{MN} + \left(q_{MN} + \frac{2}{h_1} + \frac{2}{h_2}\right)w_{MN} = & \\
\qquad = F_{MN} + \left(\frac{2}{h_1} + \frac{2}{h_2}\right)\psi_{MN}, & i=M, \quad j=N\\
-\frac{1}{h_1}\left[(aw_{\bar{x}})_{21} - \frac{1}{h_1}a_{11}w_{11}\right] - \frac{1}{h_2}\left[(bw_{\bar{y}})_{12} - \frac{1}{h_2}b_{11}w_{11}\right] + q_{11}w_{11} & \\
\qquad = F_{11} + \frac{a_{11}}{h_1^2}\varphi_{01} + \frac{b_{11}}{h_2^2}\varphi_{10}, & i=1, \quad j=1\\
\frac{2}{h_2}(bw_{\bar{y}})_{1N} + (q_{1N} + \frac{2}{h_2})w_{1N} - \frac{1}{h_1}\left[(aw_{\bar{x}})_{2N} - \frac{1}{h_1}a_{1N}w_{1N}\right] = & \\
\qquad = F_{1N} + \frac{2}{h_2}\psi_{1N} + \frac{a_{1N}}{h_1^2}\varphi_{0N}, & i=1, \quad j=N\\
\frac{2}{h_1}(aw_{\bar{x}})_{M1} + (q_{M1} + \frac{2}{h_2})w_{M1} - \frac{1}{h_2}\left[(bw_{\bar{y}})_{M2} - \frac{1}{h_2}b_{M1}w_{M1}\right] = & \\
\qquad = F_{M1} + \frac{2}{h_2}\psi_{M1} + \frac{a_{M1}}{h_2^2}\varphi_{M0}. & i=M, \quad j=1
\end{cases} \tag{3.7}$$

eq_3.

Вытянем сеточную функцию $w_{i,j},\ i=\overline{1,M},\ j=\overline{1,N}$ в вектор:

$$w^T = \left(w_{1,1}, w_{1,2}, ..., w_{1,N}; w_{2,1}, w_{2,2}, ..., w_{2,N}; ...; ...; w_{M,1}, w_{M,2}, ..., w_{M,N}\right),$$

Распишем подробную систему (3.7) выписав линейно по $w_{i,j}$ (с учетом того, что в моем варианте: $q_{i,j} = q(x_i, y_j) = 0$, и $\varphi(x_i, 0) = \varphi(0, y_j) = 2$):

$i=1,\ j=1$:

$$\left(\frac{a_{2,1}+a_{1,1}}{h_1^2} + \frac{b_{1,2}+b_{1,1}}{h_2^2}\right)w_{1,1} - \frac{b_{1,2}}{h_2^2}w_{1,2} - \frac{a_{2,1}}{h_1^2}w_{2,1} = F_{1,1} + \frac{a_{1,1}}{h_1^2}\varphi 0,1 + \frac{b_{1,1}}{h_2^2}\varphi_{1,0},$$

$i=1,\ j=\overline{2,N-1}$:

$$-\frac{b_{1,j}}{h_2^2}w_{1,j-1} + \left(\frac{a_{2,j}+a_{1,j}}{h_1^2} + \frac{b_{1,j+1}+b_{1,j}}{h_2^2}\right)w_{1,j} - \frac{b_{1,j+1}}{h_2^2}w_{1,j+1} - \frac{a_{2,j}}{h_1^2}w_{2,j} = F_{1,j} + \frac{a_{1,j}}{h_1^2}\varphi_{0,j},$$

$i=1,\ j=N$:

$$-\frac{2b_{1,N}}{h_2^2}w_{1,N-1} + \left(\frac{2b_{1,N}}{h_2^2} + \frac{2}{h_2} + \frac{a_{2,N}+a_{1,N}}{h_1^2}\right)w_{1,N} - \frac{a_{2,N}}{h_1^2}w_{2,N} = F_{1,N} + \frac{2}{h_2}\psi_{1,N} + \frac{a_{1,N}}{h_1^2}\varphi_{0,N},$$

$i=\overline{2,M-1},\ j=1$:

$$-\frac{a_{i,1}}{h_1^2}w_{i-1,1} + \left(\frac{a_{i+1,1}+a_{i,1}}{h_1^2} + \frac{b_{i,2}+b_{i,1}}{h_2^2}\right)w_{i,1} - \frac{b_{i,2}}{h_2^2}w_{i,2} - \frac{a_{i+1,1}}{h_1^2}w_{i+1,1} = F_{i,1} + \frac{b_{i,1}}{h_2^2}\varphi_{i,0},$$

$\boldsymbol{i = \overline{2, M-1},\ j = \overline{2, N-1}:}$

$$-\frac{a_{i,j}}{h_1^2}w_{i-1,j} - \frac{b_{i,j}}{h_2^2}w_{i,j-1} + \big(\frac{a_{i+1,j} + a_{i,j}}{h_1^2} + \frac{b_{i,j+1} + b_{i,j}}{h_2^2}\big)w_{i,j} - \frac{b_{i,j+1}}{h_2^2}w_{i,j+1} - \frac{a_{i+1,j}}{h_1^2}w_{i+1,j} = F_{i,j},$$

$\boldsymbol{i = \overline{2, M-1},\ j = N:}$

$$-\frac{a_{i,N}}{h_1^2}w_{i-1,N} - \frac{2b_{i,N}}{h_2^2}w_{i,N-1} + \big(\frac{2b_{i,N}}{h_2^2} + \frac{2}{h_2} + \frac{a_{i+1,N} + a_{i,N}}{h_1^2}\big)w_{i,N} - \frac{a_{i+1,N}}{h_1^2}w_{i+1,N} = F_{i,N} + \frac{2}{h_2}\psi_{iN},$$

$\boldsymbol{i = M,\ j = 1:}$

$$-\frac{2a_{M,1}}{h_1^2}w_{M-1,1} + \big(\frac{2a_{M,1}}{h_1^2} + \frac{2}{h_1} + \frac{b_{M,2} + b_{M,1}}{h_2^2}\big)w_{M.1} - \frac{b_{M,2}}{h_2^2}w_{M,2} = F_{M,1} + \frac{2}{h_1}\psi_{M,1} + \frac{b_{M,1}}{h_2^2}\varphi_{M.0}.$$

$\boldsymbol{i = M,\ j = \overline{2, N-1}:}$

$$-\frac{2a_{M,j}}{h_1^2}w_{M-1,j} - \frac{b_{M,j}}{h_2^2}w_{M,j-1} + \big(\frac{2a_{M,j}}{h_1^2} + \frac{2}{h_1} + \frac{b_{M,j+1} + b_{M,j}}{h_2^2}\big)w_{M,j} - \frac{b_{M,j+1}}{h_2^2}w_{M,j+1} = F_{M.j} + \frac{2}{h_1}\psi_{M,j},$$

$\boldsymbol{i = M,\ j = N:}$

$$-\frac{2a_{M,N}}{h_1^2}w_{M-1,N} - \frac{2b_{M,N}}{h_2^2}w_{M,N-1} + \big(\frac{2a_{M,N}}{h_1^2} + \frac{2b_{M,N}}{h_2^2} + \frac{2}{h_1} + \frac{2}{h_2}\big)w_{M,N} = F_{M,N} + \big(\frac{2}{h_1} + \frac{2}{h_2}\big)\psi_{M.N}$$

Введём следующие обозначения:

$$c_{i,j} = \frac{a_{i,j} + a_{i+1,j}}{h_1^2} + \frac{b_{i,j} + b_{i,j+1}}{h_2^2},$$

$$d_{i,N} = \frac{2b_{i,N}}{h_2^2} + \frac{2}{h_2} + \frac{a_{i,N} + a_{i+1,N}}{h_1^2}, \qquad e_{M,j} = \frac{2a_{M,j}}{h_1^2} + \frac{2}{h_1} + \frac{b_{M,j} + b_{M,j+1}}{h_2^2},$$

$$f_{1,N} = \frac{2a_{M,N}}{h_1^2} + \frac{2b_{M,N}}{h_2^2} + \frac{2}{h_1} + \frac{2}{h_2}.$$

4 Метод решения СЛАУ.

Приближенное решение системы уравнений (3.2) для сформулированных выше краевых задач может быть получено итерационным методом наименьших невязок. Этот метод позволяет получить последовательность сеточных функций $w^{(k)} \in H$, $k = 1, 2, \ldots$, сходящуюся по норме пространства H к решению разностной схемы, т.е.

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

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

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

равенствам:

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} - \tau_{k+1} r_{ij}^{(k)}, \tag{4.1}$$

где невязка $r^{(k)} = Aw^{(k)} - B$, итерационный параметр $\tau_{k+1} = \frac{\left[Ar^{(k)}, r^{(k)}\right]}{\left\|Ar^{(k)}\right\|_E^2}$. В качестве условия остановки итерационного процесса возьмем неравенство

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

где ε – положительное число, определяющее точность итерационного метода.

5 Краткое описание проделанной работы и программной реализации

Согласно постановке задания №3 необходимо написать одну последовательную программу и две параллельные программы для решения поставленной задачи, одна из которых будет использовать стандарт MPI, а другая будет гибридной и будет использовать совместно технологии MPI и OpenMP.

5.1 Последовательная программа

Шаг 1: Инициализировать матрицу B; **InitializeMatrixB()**
Шаг 2: Выберите начальное значение итерации $w^{(0)}$;
Шаг 3: Вычислите $Aw^{(k)}$; – **CalculateAw()**
Шаг 4: Вычислите невязку $r^{(k)} = Aw^{(k)} - B$; – **Diff2Vector()**
Шаг 5: Вычислите $\tau_{k+1} = \frac{\left[Ar^{(k)}, r^{(k)}\right]}{\left\|Ar^{(k)}\right\|_E^2}$, – **DotProduct(), NormVector()**
Шаг 6: Обновите $w_{ij}^{(k+1)} = w_{ij}^{(k)} - \tau_{k+1} r_{ij}^{(k)}$; – **Update_w()**
Шаг 7: Сравните значения $\|w^{(k+1)} - w^{(k)}\|_E$ и EPS, если $\|w^{(k+1)} - w^{(k)}\|_E$ меньше или равно EPS, алгоритм завершается, а $w_{ij}^{(k+1)}$ является окончательным численным решением, если $\|w^{(k+1)} - w^{(k)}\|_E$ больше EPS, повторяем шаги 3-6.

5.2 Параллельная программа: MPI

5.2.1 Алгоритм двумерного разбиения расчетной области Π на домены

tion_5.2

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

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

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

Предположим, что всего имеется p процессов, есть p_x процессов в направлении оси x и p_y процессов в направлении оси y. В каждом блоке процесса есть количество узлы n_x и n_y в направлениях x и y. Следовательно, условие (5.2.1) означает

$$1. \quad \frac{n_x(i)}{n_y(j)} \in [1/2, 2], \quad \forall i \in \overline{1, p_x},\ j \in \overline{1, p_y}$$

$$2. \quad |n_x(i) - n_x(k)| \leq 1, \quad \forall i, k \in \overline{1, p_x}; \quad |n_y(j) - n_y(z)| \leq 1, \quad \forall j, z \in \overline{1, p_y}.$$

где $p = p_x * p_y$.

Во-первых, давайте разберемся с условием 2. Возьмем, к примеру, $p_x = 4, p_y = 8$ и количество узлов 500*500. Каждый блок получает $500/4 = 125$ узлов по оси x. Для оси y каждому блоку сначала выделяется $500/8 = 62$ узла, а оставшиеся $500\%8 = 4$ узла выделяются блокам, координата j которых меньше остатка 4. Т.е. блока процесса с координатами $j = 0, 1, 2, 3$ имеются 63 узла по оси y, а остальные блоки процесса 62.

Затем проверяем условие 1. В этом примере, $\frac{n_x(i)}{n_y(j)} = 125/63 \approx 2$, выполняется условие 1. Следовательно, если $min|p_x - p_y|$ & $p_x * p_y = p$ выполнено, и узлы распределены для блоков по осям x и y указанным выше образом, то деление области удовлетворяет условиям 1 и 2.

Часть разбиения области реализуется следующей функцией:

- **struct ProcInfo**

  чтобы сохранить необходимую информацию, требуемую каждым блоком процесса, создаем структуру ProcInfo.

```
// A structure containing the information needed for a process block
struct ProcInfo {
  int num_procs;
  int dims_x, dims_y; // Record how many processes are on the x and y axes

  int rank; // the rank of the process block
  int coords[2]; // The coordinates of this process block in the MPI topology

  int size_x, size_y; // The size of the process block
  // The startand end coordinates of the process block
  int i_beg, i_end; int j_beg, j_end;

  // Process block size containing neighbor block boundary information.
   real_size_x = size_x + 2; real_size_y = size_y + 2
  int real_size_x, real_size_y;

  // The rank of the process block's upper, lower, left, and right neighbors
  int nb_left, nb_right, nb_up, nb_down;
};
```

- **DomainDecomp(), FindNumproc_X()**
  Информация о параметрах структуры ProcInfo получается в функции DomainDecomp(). В этой функции мы используем **MPI_Car_create()** для создания виртуального топологии решетка, **MPI_Comm_rank()** и **MPI_Cart_coords()** для получения ранг и координаты координаты данной блока в виртуальной решетке. Затем мы используем функцию **FindNumproc_X()**, чтобы получить количество процессов по осям x и y, также вычислить размер и начальные координаты i_beg, j_beg каждого блока. Наконец, ранг верхнего/нижнего/левого/правого блока процесса получается через функцию **MPI_CART_SHIFT()** для подготовки к обмену информацией между блоками процесса.

```
// Find the closest two factors a*b=num_procs && min|a-b|
int FindNumproc_X(int num_procs) {
  int s = (int)sqrt(num_procs);
  for (int i = s; i > 0; i--) {
    if (num_procs % i == 0) {
      return i;
    }
  }
}

// Decompose the region and obtain the necessary information about the
   process block itself
void DomainDecomp(int M, int N, MPI_Comm* Grid_Comm, ProcInfo* Proc) {
  // The MPI library is being activated...
  MPI_Comm_size(MPI_COMM_WORLD, &(Proc->num_procs));
  MPI_Comm_rank(MPI_COMM_WORLD, &(Proc->rank));

  // Determine how many processes are in each of the x, y directions
  int dims[2] = { 0,0 };
  dims[0] = FindNumproc_X(Proc->num_procs);
  dims[1] = Proc->num_procs / dims[0];
  Proc->dims_x = dims[0]; Proc->dims_y = dims[1];

  // Creating MPI Topology...
  const int ndims = 2;
  int periods[2] = { 0,0 };
  MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periods, 0, Grid_Comm);
  MPI_Comm_rank(*Grid_Comm, &(Proc->rank));
  MPI_Cart_coords(*Grid_Comm, Proc->rank, ndims, Proc->coords);

  // Determine the number of nodes in each domain
  Proc->size_x = M / dims[0]; Proc->size_y = N / dims[1];
  if (Proc->coords[0] < (M % dims[0])) Proc->size_x += 1;
  if (Proc->coords[1] < (N % dims[1])) Proc->size_y += 1;
```

```
34
35  // Calculate the starting point and ending point of each domain
36  // i_beg
37  if (Proc->coords[0] < M % dims[0]) {
38    Proc->i_beg = Proc->coords[0] * (M / dims[0]) + Proc->coords[0];
39  }
40  else { // Proc->coords[0] >= M % dims[0]
41    Proc->i_beg = Proc->coords[0] * (M / dims[0]) + M % dims[0];
42  }
43  Proc->i_end = Proc->i_beg + Proc->size_x - 1;
44  // j_beg
45  if (Proc->coords[1] < N % dims[1]) {
46    Proc->j_beg = Proc->coords[1] * (N / dims[1]) + Proc->coords[1];
47  }
48  else {
49    Proc->j_beg = Proc->coords[1] * (N / dims[1]) + N % dims[1];
50  }
51  Proc->j_end = Proc->j_beg + Proc->size_y - 1;
52
53  // real_size is the process block size with neighbor block boundary
      information
54  Proc->real_size_x = Proc->size_x + 2; Proc->real_size_y = Proc->size_y + 2;
55
56  // Get the rank of a neighboring process block
57  MPI_Cart_shift(*Grid_Comm, 0, 1, &(Proc->nb_left), &(Proc->nb_right));
58  MPI_Cart_shift(*Grid_Comm, 1, 1, &(Proc->nb_down), &(Proc->nb_up));
59 }
```

5.2.2 Межпроцессного взаимодействия

- **ExchangeBorder()**

  Эта функция завершает процесс обмена граничной информацией между блоками

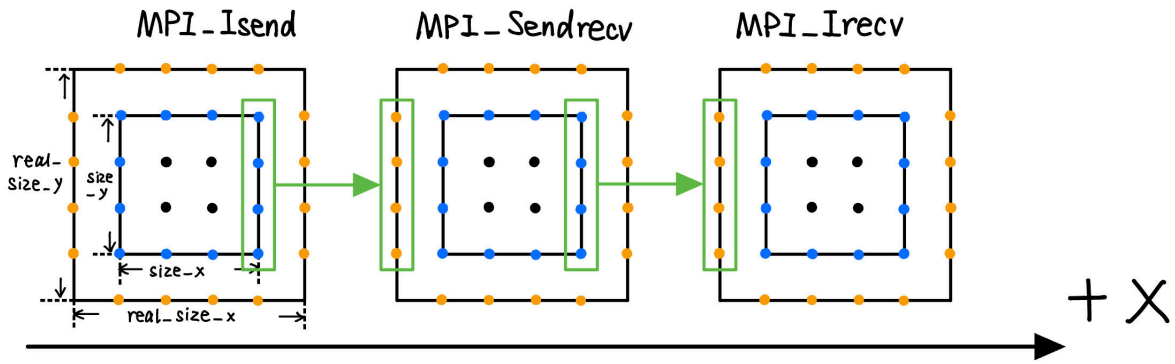

fig_1

Рис. 5.1: обмен граничными данными по +X

процесса. Для предотвращения «Deadlock» во время связи, обмен данными будет осуществляться по направлениям +Y/-Y/+X/-X соответственно. Возьмём рис. (5.1)

в качестве примера, чтобы проиллюстрировать этот процесс.

Точки в каждом блоке процесса делятся на три категории: внутренние точки (черные точки на рисунке), граничные точки (синие точки), полученные точки (желтые точки). При движении вдоль +X, если блок процесса находится на левой границе, он отправляет только свои правые граничные точки соседу справа; если блок процесса находится на правой границе, он получает граничные точки только от левого соседа; если процесс блок не находится в границе, он не только отправляет свою правую граничные точки правому соседу, но и получает информацию от левого соседа. То же самое для направления -X/+Y/-Y.

Исходя из описанного выше процесса межпроцессного взаимодействия, коммуникационные функции MPI, которые необходимо использовать, следующие неблокирующих/блокирующих функции: **MPI_Isend(), MPI_Sendrecv(), MPI_Irecv()**, и используем **MPI_Wait()** для обеспечения завершения неблокирующей отправки и получения сообщений.

```
// Exchange process block boundary information
void ExchangeBorder(double* w_plus, MPI_Comm* Grid_Comm, ProcInfo* Proc) {
        ............
        ............
  int i, j;
  int TAG_X = 0, TAG_Y = 1;

  MPI_Status status;
  MPI_Request request;

  //To avoid deadlock, next, the boundary information will be transmitted
   along the +Y / -Y / +X / -X directions respectively
  // +Y:
  if ((Proc->nb_down < 0) && (Proc->nb_up >= 0)) { // The process block is at
     the lower border
    for (i = 1; i <= Proc->size_x; i++) {
      send_up[i] = w_plus[(i + 1) * size_y - 2];
    }
    MPI_Isend(send_up, size_x, MPI_DOUBLE, Proc->nb_up, TAG_Y, *Grid_Comm, &
   request);
  }
  else if ((Proc->nb_down >= 0) && (Proc->nb_up >= 0)) {
    for (i = 1; i <= Proc->size_x; i++) {
      send_up[i] = w_plus[(i + 1) * size_y - 2];
    }

    MPI_Sendrecv(send_up, size_x, MPI_DOUBLE, Proc->nb_up, TAG_Y, recv_down,
   size_x, MPI_DOUBLE, Proc->nb_down, TAG_Y, *Grid_Comm, &status);
    for (i = 1; i <= Proc->size_x; i++) {
```

```
      w_plus[i * size_y] = recv_down[i];
    }
  }
  else if ((Proc->nb_down >= 0) && (Proc->nb_up < 0)) { // The process block
   is at the upper border
    MPI_Irecv(recv_down, size_x, MPI_DOUBLE, Proc->nb_down, TAG_Y, *Grid_Comm
   , &request);
    MPI_Wait(&request, &status);
    for (i = 1; i <= Proc->size_x; i++) {
      w_plus[i * size_y] = recv_down[i];
    }
  }
        ............
        ............
}
```

5.2.3 Реализация итеративного алгоритма

В отличие от последовательной программы, реализация MPI реализует каждый шаг итерационного алгоритма на основе каждой блока процесса. Но значения tau, norm(diff_w) вычисляются по всем точкам, поэтому мы используем **MPI_Allreduce()** для получения глобального tau, norm(diff_w) через операцию **MPI_SUM**. Обратите внимание, что перед расчетом Aw/Ar требуется необходимый обмен граничными значениями.

```
// Implement an iterative algorithm
double Solve(int M, int N, double h_x, double h_y, MPI_Comm* Grid_Comm, ProcInfo*
    Proc, int* num_iter) {
        ............
        ............
  // Given an initial iteration value
  InitMatrix_with_value(w, 0.82, Proc->real_size_x, Proc->real_size_y);

  // Initialize the right side of the equation: matrix B
  InitializeB(B, M, N, h_x, h_y, Proc);
  do {
    // w^(k)
    Update_pre_w(pre_w, w, Proc->real_size_x, Proc->real_size_y);
    // Exchanging boundary information between process blocks
    ExchangeBorder(pre_w, Grid_Comm, Proc);
    // Aw^{k}
    CalculateAw(Aw, pre_w, M, N, h_x, h_y, Proc);
    // r^{k} = Aw^{k} - B
    Diff2Vector(r, Aw, B, Proc->real_size_x, Proc->real_size_y);
    // Exchanging boundary information between process blocks
    ExchangeBorder(r, Grid_Comm, Proc);
    // Ar^{k}
```

```
    CalculateAw(Ar, r, M, N, h_x, h_y, Proc);
    // tau
    // step 1. Each process block calculates its own tau numerator and denominator
    tau_local_numerator = DotProduct(Ar, r, M, N, h_x, h_y, Proc);
    tau_local_denominator = DotProduct(Ar, Ar, M, N, h_x, h_y, Proc);
    // step 2. After reduction, compute the public global tau
    MPI_Allreduce(&tau_local_numerator, &tau_global_numerator, 1, MPI_DOUBLE,
   MPI_SUM, *Grid_Comm);
    MPI_Allreduce(&tau_local_denominator, &tau_global_denominator, 1, MPI_DOUBLE,
   MPI_SUM, *Grid_Comm);
    tau_global = tau_global_numerator / tau_global_denominator;
    // w^{k+1}_{ij}
    Update_w(w, pre_w, tau_global, r, M, N, Proc);
    *(num_iter) = *(num_iter)+1;
    // ||w^{k+1} - w^{k}|| < EPS
    Diff2Vector(diff_w, w, pre_w, Proc->real_size_x, Proc->real_size_y);
    // Calculate the error within each process block
    diff_local = DotProduct(diff_w, diff_w, M, N, h_x, h_y, Proc);
    // After reduction, compute the public global error
    MPI_Allreduce(&diff_local, &diff_global, 1, MPI_DOUBLE, MPI_SUM, *Grid_Comm);
    error_norm = sqrt(diff_global);

  } while (error_norm > EPS);
        ............
        ............
}
```

5.2.4 Гибридное программирование openMP+MPI

В этой программе на основе программы MPI добавлены следующие две директивы openMP:

1 #pragma omp parallel for default(shared) schedule(dynamic)

2 #pragma omp parallel for default(shared) schedule(dynamic) reduction(+:res)

Первая директива используется для распараллеливания цикла for. Здесь чтобы переменные каждой нити не смешивались друг с другом, можно использовать **private** или объявить переменные внутри цикла. Вторая директива openMP используется в функции **DotProduct()**. reduction объявляет, что res является частной переменной нити, после завершения параллелизма выполняется операция суммирования, и результат возвращается в одноименную переменную основной нити.

6 Результаты на системе Polus

По описанной выше программе я провела несколько тестов на полюсах с разным количеством процессов. Чтобы завершить программу на Полюсе за 20 минут, я пыталась много раз и выбрала начальное значение итерации $w^{(0)} = 0.82$ с точностью $EPS = 5e - 6$.

В следующих двух таблицах время работы программы сравнивается при разном количестве процессов. По таблицам можем получить следующие результаты:

- По результатам, представленным в (Табл. 1), можно обнаружить, что при увеличении количества процессов на 4, 8, 16 и 32 соответственно экспоненциально уменьшается и время работы программы. При количестве процессов 32 не получается ускорение = 8. Возможная причина в том, что при увеличении количества процессов увеличиваются коммуникационные издержки между процессами, такие как Ised(), Irecv(), Allreduce() при обмене граничные данные блока процессов.

- При одном и том же количестве процессов, по мере увеличения данных, в моем варианте соответственно увеличивается время работы программы, что логично, т.к. увеличивается количество данных, обрабатываемых программой.

- Таблица 2 может быть получена из гибридной программы (OpenMP+MPI). По сравнению с таблицей 1, когда количество процессов в таблице 2 * количество нитей = количеству процессов в таблице 1, скорость работы гибридной программы выше, чем у программы, использующей только MPI, а ускорение составляет около 1,2. Это связано с тем, что мы используем технологию openMP в гибридной программе, которая позволяет нам распределять огромные вычисления по разным нитям, и этим нитям не нужно взаимодействовать друг с другом, что экономит накладные расходы на связь.

Число процессов MPI	Число точек сетки M × N	Время(s) решения	Ускорение
4	500 × 500	1112.64	1
8	500 × 500	495.322	2.24
16	500 × 500	281.639	3.95
32	500 × 500	154.578	7.18
4	500 × 1000	1204.588	1
8	500 × 1000	567.061	2.12
16	500 × 1000	304.805	3.96
32	500 × 1000	178.434	6.75

Таблица 1: Таблица с результатами расчетов на ПВС IBM Polus (MPI код)

Число процессов MPI	Количество OMP-нитей в процессе	Число точек сетки M × N	Время решения (s)	Ускорение
1	4	500 × 500	882.625	1
2	4	500 × 500	459.591	1.92
4	4	500 × 500	223.738	3.94
8	4	500 × 500	119.497	7.38
1	4	500 × 1000	945.218	1
2	4	500 × 1000	507.27	1.86
4	4	500 × 1000	237.915	3.97
8	4	500 × 1000	152.889	6.18

Таблица 2: Таблица с результатами расчетов на ПВС IBM Polus (MPI + OpenMP)

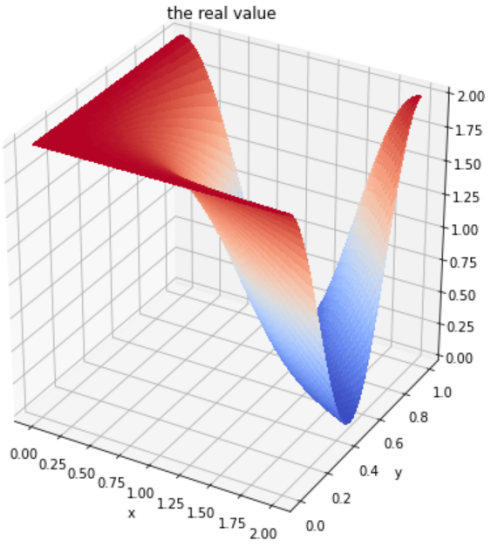

Рис. 6.1: real value

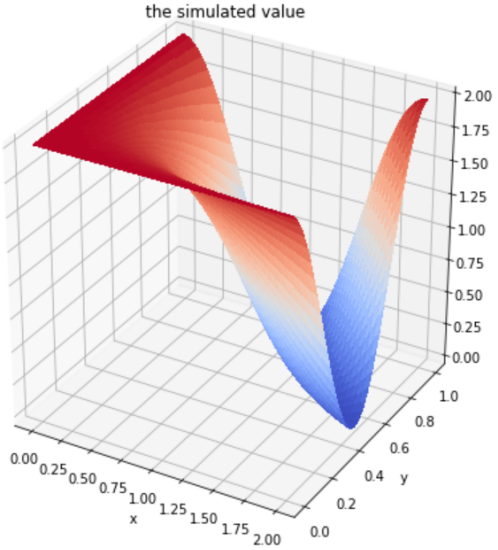

Рис. 6.2: simulated value