



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

**Суперкомпьютерное решение задачи разностной схемы  
для уравнения Пуассона задачи Дирихле**

ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЗАДАНИЙ ПО КУРСУ  
«СУПЕРКОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ И ТЕХНОЛОГИИ»

Студент дневного отделения магистратуры  
Лю Хайлинь

Преподаватель:  
Доцент, Попова Н.Н.

Москва, 2025

# Содержание

<b>1 Введение</b>	<b>3</b>
1.1 Актуальность . . . . .	3
1.2 Постановка задачи . . . . .	3
<b>2 Математическая постановка и численные методы</b>	<b>5</b>
2.1 Основное уравнение . . . . .	5
2.2 Метод фиктивных областей . . . . .	5
2.3 Метод конечных разностей . . . . .	6
2.4 Метод сопряжённых градиентов с предобусловливанием . . . . .	7
<b>3 Программная реализация и результатов выполнения</b>	<b>9</b>
3.1 Последовательная . . . . .	9
3.2 Распараллеливание с OpenMP . . . . .	9
3.3 Распараллеливание с MPI . . . . .	13
3.4 Распараллеливание с MPI + OpenMP . . . . .	15
<b>4 Заключение</b>	<b>17</b>

# 1 Введение

## 1.1 Актуальность

Решение уравнения Пуассона имеет фундаментальное значение в математическом моделировании процессов теплопередачи, электростатического потенциала, диффузии и гидродинамики. Для большинства реальных задач, где область имеет сложную геометрию или неоднородные граничные условия, аналитические решения отсутствуют, поэтому требуется численное моделирование.

Современные вычислительные методы и технологии параллельных вычислений позволяют эффективно решать краевые задачи больших размерностей. Применение параллельных моделей, таких как OpenMP и MPI, обеспечивает значительное ускорение вычислений на многоядерных и многопроцессорных архитектурах.

Разработка и исследование параллельных численных методов для решения уравнения Пуассона представляет собой актуальную задачу вычислительной математики и суперкомпьютерных технологий, поскольку такие подходы находят широкое применение в инженерных расчётах, моделировании физических полей и решении обратных задач.

## 1.2 Постановка задачи

### Цели работы

- Разработать и реализовать численный метод решения уравнения Пуассона в трапециевидной области с помощью метода конечных разностей.
- Исследовать эффективность параллельных реализаций на основе OpenMP, MPI и гибридной модели MPI+OpenMP.
- Провести сравнительный анализ ускорений и масштабируемости разработанных реализаций.

### Задачи работы

- Построить разностную схему для уравнения Пуассона с использованием метода фиктивных областей.
- Реализовать итерационный метод сопряжённых градиентов для решения полученной системы линейных уравнений.

- Реализовать параллельные версии программы с применением OpenMP, MPI и их гибридного сочетания.
- Провести вычислительные эксперименты для различных размеров сеток и числа процессов.
- Проанализировать влияние параллелизма на скорость сходимости и общую производительность.

## 2 Математическая постановка и численные методы

### 2.1 Основное уравнение

Рассматривается двумерное уравнение Пуассона задачи Дирихле:

$$-\Delta u = f(x, y), \quad (x, y) \in D, \quad (2.1)$$

где  $\Delta$  — оператор Лапласа:

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}. \quad (2.2)$$

Для заданной области  $D$  с границей  $\partial D$  формулируются граничные условия Дирихле:

$$u(x, y) = 0, \quad (x, y) \in \partial D. \quad (2.3)$$

Правая часть уравнения полагается постоянной:

$$f(x, y) = 1. \quad (2.4)$$

Область  $D$  представляет собой трапецию с вершинами

$$A(0, 0), \quad B(3, 0), \quad C(2, 3), \quad D(0, 3),$$

вложенную в прямоугольник  $\Pi = [0, 3] \times [0, 3]$ .

### 2.2 Метод фиктивных областей

Для нахождения приближённого решения задачи (2.1)–(2.3) используется метод конечных разностей. Поскольку рассматриваемая область  $D$  имеет сложную форму (трапецию), прямое построение сетки, совпадающей с её границей, является затруднительным. Для этого применяется **метод фиктивных областей**, позволяющий рассматривать область  $D$  как часть прямоугольной области  $\Pi$ , в которой удобно строить регулярную сетку.

Пусть область  $D$  принадлежит прямоугольнику  $\Pi$ , заданному как

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

а  $\widehat{D} = \Pi \setminus D$  — фиктивная область (область вне  $D$ , но внутри  $\Pi$ ).

Тогда исходное уравнение (2.1) переписывается в обобщённом виде:

$$-\frac{\partial}{\partial x} \left( k(x, y) \frac{\partial v}{\partial x} \right) - \frac{\partial}{\partial y} \left( k(x, y) \frac{\partial v}{\partial y} \right) = F(x, y), \quad (x, y) \in \Pi, \quad (2.5)$$

с граничным условием:

$$v(x, y) = 0, \quad (x, y) \in \Gamma = \partial \Pi, \quad (2.6)$$

где  $k(x, y)$  — коэффициент теплопроводности, а  $F(x, y)$  — обобщённая правая часть, определяемые следующим образом:

$$k(x, y) = \begin{cases} 1, & (x, y) \in D, \\ \frac{1}{\varepsilon}, & (x, y) \in \widehat{D}, \end{cases} \quad F(x, y) = \begin{cases} f(x, y), & (x, y) \in D, \\ 0, & (x, y) \in \widehat{D}, \end{cases} \quad (2.7)$$

где  $\varepsilon$  — малый параметр, обеспечивающий плавное выключение уравнения в фиктивной области.

### 2.3 Метод конечных разностей

Для построения численного решения уравнения (2.5) используется **метод конечных разностей**, который заключается в аппроксимации производных разностями на равномерной прямоугольной сетке.

В прямоугольной области  $\Pi = [A_1, B_1] \times [A_2, B_2]$  задаётся равномерная сетка

$$x_i = A_1 + ih_x, \quad i = 0, 1, \dots, M, \quad y_j = A_2 + jh_y, \quad j = 0, 1, \dots, N, \quad (2.8)$$

где шаги сетки определяются как

$$h_x = \frac{B_1 - A_1}{M}, \quad h_y = \frac{B_2 - A_2}{N}.$$

В каждой внутренней узловой точке  $(x_i, y_j)$  производные в уравнении (2.5) аппроксимируются центральными разностями. Получаем разностную схему в виде:

$$\begin{aligned} & -\frac{1}{h_x^2} \left( k_{i+\frac{1}{2}, j} (u_{i+1, j} - u_{i, j}) - k_{i-\frac{1}{2}, j} (u_{i, j} - u_{i-1, j}) \right) \\ & -\frac{1}{h_y^2} \left( k_{i, j+\frac{1}{2}} (u_{i, j+1} - u_{i, j}) - k_{i, j-\frac{1}{2}} (u_{i, j} - u_{i, j-1}) \right) = F_{i, j}. \end{aligned} \quad (2.9)$$

где  $u_{i, j} \approx u(x_i, y_j)$ , а коэффициенты  $k_{i \pm \frac{1}{2}, j}$  и  $k_{i, j \pm \frac{1}{2}}$  вычисляются как средние значения  $k(x, y)$  между соседними узлами:

$$k_{i+\frac{1}{2}, j} = \frac{1}{2} (k_{i, j} + k_{i+1, j}), \quad k_{i, j+\frac{1}{2}} = \frac{1}{2} (k_{i, j} + k_{i, j+1}). \quad (2.10)$$

Правая часть  $F_{i,j}$  определяется как значение функции  $F(x, y)$  в узле  $(x_i, y_j)$ :

$$F_{i,j} = F(x_i, y_j). \quad (2.11)$$

На границе  $\Gamma = \partial\Pi$  реализуются граничные условия Дирихле:

$$u_{i,j} = 0, \quad (x_i, y_j) \in \Gamma. \quad (2.12)$$

Таким образом, для всех внутренних точек сетки ( $i = 1, \dots, M - 1, j = 1, \dots, N - 1$ ) уравнение (2.9) задаёт систему линейных алгебраических уравнений относительно неизвестных  $u_{i,j}$ :

$$A u = F,$$

где  $A$  — дискретный оператор Лапласа, модифицированный коэффициентами  $k(x, y)$ , а  $F$  — вектор правых частей.

## 2.4 Метод сопряжённых градиентов с предобусловливанием

После дискретизации получаем СЛАУ

$$Au = F, \quad A = A^\top \succ 0, \quad (2.13)$$

эквивалентную задаче минимизации квадратичного функционала

$$J(u) = \frac{1}{2} (Au, u) - (F, u). \quad (2.14)$$

Градиент функционала равен

$$\nabla J(u) = Au - F = -(F - Au) = -r, \quad r := F - Au, \quad (2.15)$$

поэтому движение в направлении  $r$  (или его предобусловленного варианта  $z = D^{-1}r$ ) уменьшает  $J$ .

Для ускорения сходимости применяется диагональный предобусловливатель  $D \approx A$ :

$$(Dw)_{i,j} = \left( \frac{k_{i+1,j} + k_{i,j}}{h_x^2} + \frac{k_{i,j+1} + k_{i,j}}{h_y^2} \right) w_{i,j}, \quad 1 \leq i \leq M - 1, 1 \leq j \leq N - 1, \quad (2.16)$$

что даёт простое обращение  $z = D^{-1}r$  покомпонентно.

Строить направления  $p^{(k)}$ , попарно  $A$ -сопряжённые,

$$(Ap^{(k)}, p^{(\ell)}) = 0, \quad k \neq \ell, \quad (2.17)$$

и искать  $u^{(k+1)} = u^{(k)} + \alpha_k p^{(k)}$  с оптимальным шагом

$$\alpha_k = \frac{(r^{(k)}, z^{(k)})}{(Ap^{(k)}, p^{(k)})}, \quad Dz^{(k)} = r^{(k)}. \quad (2.18)$$

Обновление направления выполняется по формуле

$$\beta_k = \frac{(r^{(k)}, z^{(k)})}{(r^{(k-1)}, z^{(k-1)})}, \quad p^{(k)} = z^{(k)} + \beta_k p^{(k-1)}. \quad (2.19)$$

После шага  $\alpha_k$  остаток обновляется как

$$r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)}. \quad (2.20)$$

Критерий остановки берём, например, по норме остатка:

$$\|r^{(k)}\|_2 \leq \varepsilon \quad \text{или} \quad \frac{\|r^{(k)}\|_2}{\|F\|_2} \leq \varepsilon_{\text{rel}}. \quad (2.21)$$

Задача решаем по следующему алгоритму:

---

**Algorithm 1** Предобусловленный метод сопряжённых градиентов (PCG)

---

**Require:**  $A, F, u^{(0)}, D, \varepsilon, k_{\max}$

- 1:  $r^{(0)} \leftarrow F - Au^{(0)}$
  - 2:  $z^{(0)} \leftarrow D^{-1}r^{(0)}$
  - 3:  $p^{(0)} \leftarrow z^{(0)}$
  - 4:  $\rho^{(0)} \leftarrow (r^{(0)}, z^{(0)})$
  - 5: **for**  $k = 0, 1, 2, \dots, k_{\max} - 1$  **do**
  - 6:      $q \leftarrow Ap^{(k)}$
  - 7:      $\alpha_k \leftarrow \rho^{(k)} / (p^{(k)}, q)$
  - 8:      $u^{(k+1)} \leftarrow u^{(k)} + \alpha_k p^{(k)}$
  - 9:      $r^{(k+1)} \leftarrow r^{(k)} - \alpha_k q$
  - 10:    **if**  $\|r^{(k+1)}\|_2 \leq \varepsilon$  **then**
  - 11:       **break**
  - 12:    **end if**
  - 13:     $z^{(k+1)} \leftarrow D^{-1}r^{(k+1)}$
  - 14:     $\rho^{(k+1)} \leftarrow (r^{(k+1)}, z^{(k+1)})$
  - 15:     $\beta_{k+1} \leftarrow \rho^{(k+1)} / \rho^{(k)}$
  - 16:     $p^{(k+1)} \leftarrow z^{(k+1)} + \beta_{k+1} p^{(k)}$
  - 17: **end for**
  - 18: **return**  $u^{(k+1)}$
-

### 3 Программная реализация и результатов выполнения

#### 3.1 Последовательная

На основе описанных выше методов реализован решатель для уравнения Пуасона с краевыми условиями Дирихле. Решатель построен на использовании класса `PoissonSolver`, определённого в файле `include/conjugate_gradient.hpp`. Данный класс реализует все этапы метода предобусловленного сопряжённого градиента (PCG), включая построение сетки, инициализацию, применение оператора  $A$ , предобуславливание и итерационный процесс поиска решения. Метод конечных разностей применяется для аппроксимации оператора Лапласа, а фиктивные области позволяют обрабатывать граничные участки трапециевидной геометрии. Итерации выполняются до достижения нормы остатка, меньшей заданного допуска  $\varepsilon$ .

Основной файл `task.cpp` организует ввод параметров сетки  $(M, N)$ , запуск решателя и измерение времени работы программы. После завершения вычислений решение сохраняется в виде CSV-файла.

При  $(M, N) = (10, 10), (20, 20), (40, 40)$  используем последовательную программу для вычисления приближенного решения. Получены следующие результаты:

Сетка ( $M \times N$ )	Число итераций	Время(s)
$(10 \times 10)$	20	0.000837
$(20 \times 20)$	47	0.007440
$(40 \times 40)$	108	0.069108

Таблица 1: Последовательная программа на разных сеток

#### 3.2 Распараллеливание с OpenMP

В дальнейшем на основе последовательной реализации была добавлена поддержка параллельных вычислений с использованием технологии OpenMP. Основная идея заключалась в распараллеливании наиболее трудоёмких двойных циклов по индексам  $(i, j)$ , возникающих при вычислении операторов  $Au$ ,  $Ap$ , а также при обновлениях векторов  $u$ ,  $r$ ,  $p$  и  $z$ . Для этого в код были добавлены директивы `#pragma omp parallel for collapse(2)`,

позволяющие автоматически разделять итерации по потокам. Такое решение обеспечивает равномерную загрузку вычислительных ядер и значительно ускоряет процесс решения при больших размерах сетки.

Кроме того, в местах, где использовались операции суммирования (например, при вычислении норм и скалярных произведений), были добавлены конструкции `reduction(+:sum)`, что позволило корректно и безопасно объединять результаты вычислений от разных потоков. Таким образом, все основные участки с независимыми вычислениями были распараллелены, при этом численная точность метода и устойчивость итерационного процесса сохраняются.

При  $(M, N) = (40, 40)$  используем последовательную и параллельную программу с 1, 4, 16-нитей. Получены следующие результаты.

Количество нитей	Сетка ( $M \times N$ )	Число итераций	Время (s)	Ускорение
-	$(40 \times 40)$	108	0.069108	1.00
1	$(40 \times 40)$	108	0.069582	1.00
4	$(40 \times 40)$	108	0.020581	3.36
16	$(40 \times 40)$	108	0.012877	5.37

Таблица 2: Распараллеливание с OpenMP с разными количествами нитей

Как показано в таблице 2, при небольших размерах сетки  $(M, N)$  объём вычислений остаётся относительно малым, поэтому использование параллельных программ OpenMP не приводит к заметному ускорению по сравнению с последовательной версией. Это связано с тем, что при малом количестве узлов затраты на создание потоков, их синхронизацию и переключение между ними становятся соизмеримыми с самим временем вычислений, что снижает общую эффективность параллелизации.

При увеличении размеров задачи влияние накладных расходов уменьшается, и преимущества параллельных вычислений становятся более очевидными. В дальнейшем мы рассмотрим результаты для более крупных сеток  $(M, N) = (400, 600)$  и  $(800, 1200)$ , где применение OpenMP демонстрирует существенное ускорение по сравнению с последовательной реализацией.

Количество нитей	Сетка ( $M \times N$ )	Число итераций	Время (s)	Ускорение
-	(400 × 600)	1812	176.748	1.00
2	(400 × 600)	1812	89.1394	1.98
4	(400 × 600)	1812	44.3287	3.99
8	(400 × 600)	1812	24.2446	7.29
16	(400 × 600)	1812	16.9253	10.44
-	(800 × 1200)	3754	1466.55	1.00
4	(800 × 1200)	3754	375.709	3.90
8	(800 × 1200)	3754	198.446	7.39
16	(800 × 1200)	3754	131.231	11.17
32	(800 × 1200)	3754	67.1255	21.85

Таблица 3: Распараллеливание с OpenMP при больших размеров сетки

Анализ представленных данных показывает, что при увеличении размеров сетки эффективность распараллеливания возрастает значительно. Для сетки  $(400 \times 600)$  ускорение при 16 потоках достигает значения  $S = 10.44$ , что близко к линейному масштабированию. При дальнейшем увеличении задачи до  $(800 \times 1200)$  достигается ускорение  $S = 21.85$  при 32 потоках, что указывает на хорошую масштабируемость метода и эффективное использование вычислительных ресурсов.

Такое поведение объясняется тем, что при больших размерах задачи доля вычислений в общем времени исполнения резко возрастает, а накладные расходы, связанные с управлением потоками, становятся незначительными. Таким образом, реализованный OpenMP-решатель демонстрирует высокую эффективность на крупномасштабных сетках и способен обеспечивать почти идеальную параллельную производительность при увеличении числа вычислительных ядер.

Для наглядного представления полученного приближённого решения уравнения Пуассона наибольшего размера сетки  $(M, N) = (800, 1200)$  были построены визуализации в двумерном и трёхмерном форматах. На рисунке 1 показано распределение значений функции  $u(x, y)$  на плоскости, что позволяет оценить структуру решения и поведение функции внутри области. Трёхмерная визуализация, представленная на рисунке 2, иллюстрирует рельеф поверхности решения и демонстрирует плавное изменение потенциала по области.

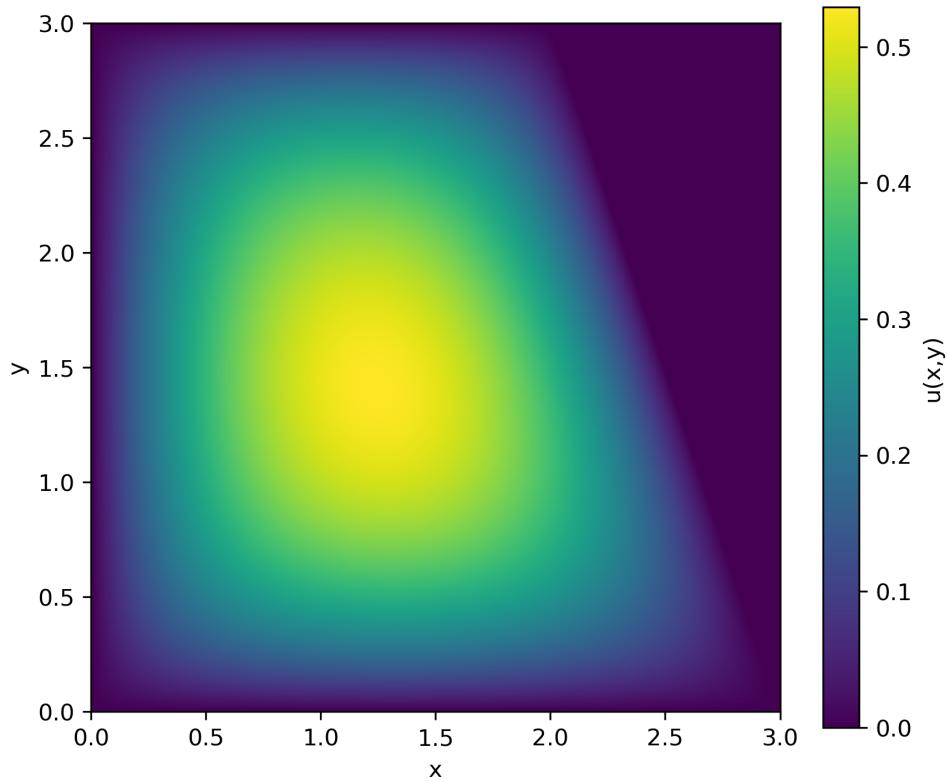


Рис. 1: Двумерная визуализация решения для сетки  $(800 \times 1200)$

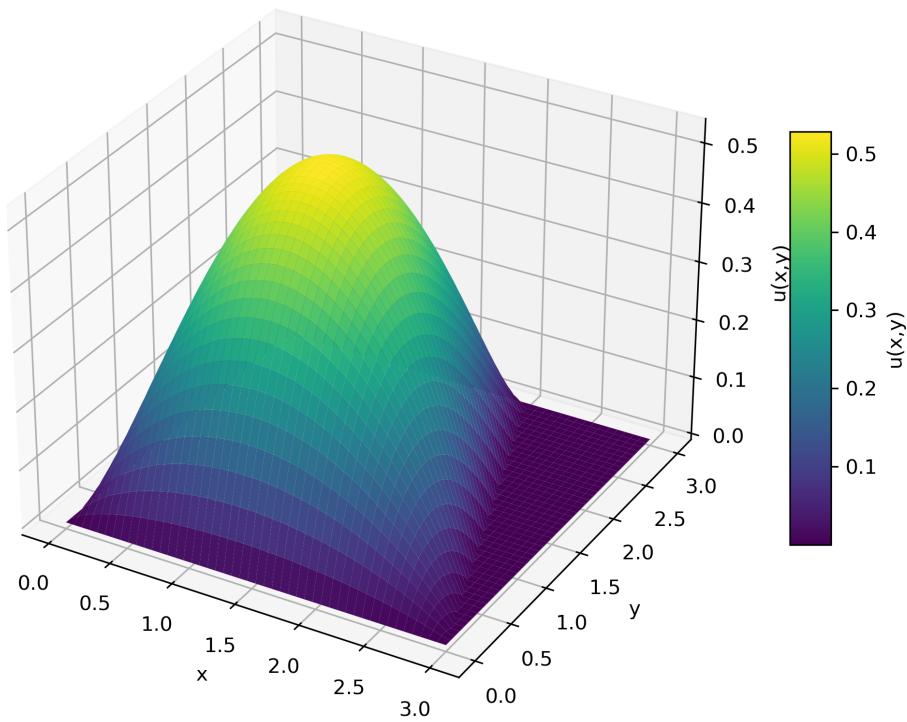


Рис. 2: Трёхмерная визуализация решения для сетки  $(800 \times 1200)$

### 3.3 Распараллеливание с MPI

Для распределения глобальной прямоугольной сетки по процессам реализован класс `DomainDecomposer`, который строит блочную декомпозицию  $p_x \times p_y$  и удовлетворяет двум условиям: (1) отношение сторон каждого локального блока не превышает 2:1, (2) длины подотрезков по каждой оси отличаются не более чем на 1 узел. Для этого выбираются  $p_x, p_y$  как близкие к  $\sqrt{P}$  множители степени двойки с учётом соотношения  $M:N$ , а разбиение по каждой оси выполняется как равномерное (блоки размера  $\lfloor \frac{\text{size}}{\text{blocks}} \rfloor$  или на единицу больше).

На построенной декомпозиции создаётся декартова топология процессов `MPI_Cart_create` с размерами  $(p_x, p_y)$  и без периодичности. Каждому рангу сопоставляются координаты `coords` и его локальный прямоугольник индексов  $[x_{\min} : x_{\max}] \times [y_{\min} : y_{\max}]$  (в глобальной нумерации). Далее для каждого процесса конструируется свой `MPIPoissonSolver` на локальной области (с добавлением гало-слоёв по соседям), после чего выполняется итерационный метод.

Ниже показан пример декомпозиции для сетки  $800 \times 1200$  при  $P = 16$ :

Пример декомпозиции области для  $800 \times 1200$ ,  $P = 16$

```
===== Domain Decomposition Summary =====
Global grid: 800 x 1200
Total processes: 16      Grid layout: 4 x 4
X partitions (4):      0 200 400 600
Y partitions (4):      0 300 600 900
=====
[Rank  0] coords=(0,0) x:[  0, 199] y:[  0, 299]
[Rank  1] coords=(0,1) x:[  0, 199] y:[ 300, 599]
[Rank  2] coords=(0,2) x:[  0, 199] y:[ 600, 899]
[Rank  3] coords=(0,3) x:[  0, 199] y:[ 900,1200]
[Rank  4] coords=(1,0) x:[ 200, 399] y:[  0, 299]
[Rank  5] coords=(1,1) x:[ 200, 399] y:[ 300, 599]
[Rank  6] coords=(1,2) x:[ 200, 399] y:[ 600, 899]
[Rank  7] coords=(1,3) x:[ 200, 399] y:[ 900,1200]
[Rank  8] coords=(2,0) x:[ 400, 599] y:[  0, 299]
[Rank  9] coords=(2,1) x:[ 400, 599] y:[ 300, 599]
[Rank 10] coords=(2,2) x:[ 400, 599] y:[ 600, 899]
[Rank 11] coords=(2,3) x:[ 400, 599] y:[ 900,1200]
[Rank 12] coords=(3,0) x:[ 600, 800] y:[  0, 299]
[Rank 13] coords=(3,1) x:[ 600, 800] y:[ 300, 599]
[Rank 14] coords=(3,2) x:[ 600, 800] y:[ 600, 899]
[Rank 15] coords=(3,3) x:[ 600, 800] y:[ 900,1200]
```

Класс `MPIPoissonSolver` наследует базовый класс `PoissonSolver` и добавляет распределённые операции на основе MPI. В конструкторе создаётся декартова топология процессов, а для каждого процесса определяются соседи по четырём направлениям: `left`, `right`, `bottom`, `top`. Это позволяет эффективно организовать обмен граничными слоями (halo exchange) между соседними поддоменами.

**Обмен граничными слоями.** Для корректного вычисления разностного оператора Лапласа вблизи границ локальной области каждая подзадача хранит дополнительные строки и столбцы — «призрачные» узлы. Метод `exchange_halo()` выполняет асинхронный обмен этими данными между соседними процессами с помощью пар вызовов `MPI_Isend/MPI_Irecv` и последующего ожидания `MPI_Waitall`. После завершения обмена граничные значения копируются в соответствующие ячейки сетки, обеспечивая непрерывность решения на стыках поддоменов.

**Редукция и вычисление глобальных норм.** Поскольку каждая подзадача хранит только часть сетки, скалярные произведения и нормы (например,  $\|r\|_2$  или  $(r, z)$ ) вычисляются локально и затем объединяются во всей топологии с помощью `MPI_Allreduce`.

**Завершение и сборка решения.** После завершения итерационного процесса каждый процесс хранит свою часть приближённого решения  $u(x, y)$ . Далее с помощью операции `MPI_Gatherv` данные передаются на `rank 0`, который собирает их в единую глобальную матрицу решения.

Мы также протестировали производительность MPI на малых и больших сетках:

Количество процессов	Сетка ( $M \times N$ )	Число итераций	Время (s)	Ускорение
(Без MPI)	$(40 \times 40)$	108	0.069108	1.00
1	$(40 \times 40)$	108	0.069582	1.00
2	$(40 \times 40)$	108	0.036846	1.88
4	$(40 \times 40)$	108	0.020216	3.42

Таблица 4: Распараллеливание с MPI с разными количествами процессов

Количество процессов	Сетка ( $M \times N$ )	Число итераций	Время (s)	Ускорение
1	(400 × 600)	1812	176.963	1.00
2	(400 × 600)	1812	88.8531	1.99
4	(400 × 600)	1818	45.4095	3.90
8	(400 × 600)	1818	23.4544	7.54
16	(400 × 600)	1808	12.1644	14.55
1	(800 × 1200)	3754	1468.37	1.00
4	(800 × 1200)	3756	374.655	3.92
8	(800 × 1200)	3754	196.448	7.47
16	(800 × 1200)	3769	100.992	14.54
32	(800 × 1200)	3763	51.8997	28.29

Таблица 5: Распараллеливание с MPI при больших размеров сетки

Анализ экспериментальных результатов показывает, что MPI-параллелизация демонстрирует высокую эффективность и почти линейное ускорение при увеличении числа процессов. Для небольшой сетки ( $40 \times 40$ ) ускорение ограничено, поскольку объём вычислений невелик и накладные расходы на коммуникацию между процессами сопоставимы с самим временем вычислений. Однако уже при размере ( $400 \times 600$ ) наблюдается почти идеальная масштабируемость: при 16 процессах ускорение достигает  $S = 14.55$ .

Для ещё более крупной сетки ( $800 \times 1200$ ) поведение остаётся аналогичным: при 32 процессах достигается ускорение  $S = 28.29$ , что подтверждает отличную масштабируемость решателя и эффективность схемы обмена граничными слоями. Незначительные расхождения в числе итераций (в пределах нескольких шагов) связаны с особенностями вычислений в плавающей арифметике: порядок операций при редукции скалярных произведений в MPI может незначительно влиять на накопленную погрешность и, соответственно, на критерий сходимости.

### 3.4 Распараллеливание с MPI + OpenMP

Для дальнейшего повышения производительности комбинируется распределённая (MPI) и потоковая (OpenMP) параллелизация. В этой гибридной схеме каждая подзадача,

запущенная на своём MPI-процессе, дополнительно использует несколько потоков OpenMP для распараллеливания вложенных циклов по узлам сетки. Такой подход позволяет эффективно задействовать как межузловой, так и внутрипроцессорный параллелизм.

Процессы	Нити	Сетка	Число итераций	Время (с)	Ускорение
-	-	(40 × 40)	108	0.069108	1.00
1	4	(40 × 40)	108	0.036172	1.91
2	4	(40 × 40)	108	0.020501	3.37

Таблица 6: Распараллеливание с MPI+OpenMP с разными количествами процессов и нитей

Процессы	Нити	Сетка	Число итераций	Время (с)	Ускорение
2	1	(400 × 600)	1812	89.5524	1.00
2	2	(400 × 600)	1812	58.5149	1.53
2	4	(400 × 600)	1812	43.7185	2.05
2	8	(400 × 600)	1812	37.5361	2.39
4	1	(800 × 1200)	3756	374.991	1.00
4	2	(800 × 1200)	3756	246.903	1.52
4	4	(800 × 1200)	3756	186.469	2.01
4	8	(800 × 1200)	3756	157.681	2.38

Таблица 7: Распараллеливание с MPI+OpenMP при больших размерах сетки

Как видно из представленных данных, добавление OpenMP-параллелизма поверх MPI обеспечивает дополнительное ускорение, особенно при увеличении числа нитей. Для небольшой сетки (40 × 40) выигрыш умеренный, однако при увеличении размеров задачи (400 × 600) и (800 × 1200) достигается устойчивое ускорение около 2.4 раза при использовании 8 потоков.

## 4 Заключение

В данной работе был реализован полнофункциональный решатель для уравнения Пуассона с краевыми условиями Дирихле, основанный на методе предобусловленного сопряжённого градиента (PCG) и методе фиктивных областей. Разработанный программный комплекс демонстрирует сочетание академической строгости и инженерной проработанности.

Основные достоинства реализации заключаются в ясной и модульной структуре кода, удобной для дальнейшего расширения и адаптации, а также в визуализируемом механизме разбиения расчётной области. Благодаря классу `DomainDecomposer` обеспечивается автоматическое построение сбалансированных поддоменов с сохранением пропорций, что делает систему легко масштабируемой на вычислительных кластерах.

Важной особенностью проекта является наличие средств автоматизации — скриптов `run.sh` для локальных тестов и `run_polus.sh` для запуска на суперкомпьютере. Это делает разработанный решатель не только исследовательским инструментом, но и практически применимым инженерным решением, готовым к использованию в производственных НРС-средах.

Проведённые эксперименты подтвердили эффективность реализации: в последовательном, OpenMP, MPI и гибридном (MPI+OpenMP) режимах наблюдается высокая сходимость и отличная масштабируемость. Визуализация решения на сетках до  $(800 \times 1200)$  позволила наглядно продемонстрировать корректность численного метода и гладкость полученного потенциала.

Итогом работы стало создание удобного, надёжного и производительного вычислительного комплекса, способного эффективно решать задачи математической физики на больших сетках. Полученные результаты подтверждают, что комбинация строгих численных методов, современных технологий параллельных вычислений и инженерного подхода позволяет разрабатывать системы, одинаково ценные как с академической, так и с прикладной точки зрения.