

Отчёт по курсу «Суперкомпьютерное моделирование и технологии» для студентов первого и второго потоков, вариант 7, Жэнь Цзежуй 618-1

## 1 Математическая постановка задачи

Требуется приближённо решить двумерную задачу Дирихле для уравнения Пуассона в области сложной формы — так называемой «области-сапожок»:

$$-\Delta u(x, y) = 1, \quad (x, y) \in D, \quad u|_{\partial D} = 0.$$

Область  $D$  определяется как разность двух квадратов:

$$D = \{(x, y) : -1 < x < 1, -1 < y < 1\} \setminus \{(x, y) : 0 < x < 1, 0 < y < 1\}.$$

То есть из квадрата  $(-1, 1) \times (-1, 1)$  удалён его правый верхний квадрат  $(0, 1) \times (0, 1)$ . Таким образом граница области имеет Г-образный вид, напоминающий сапог.

На границе задаётся условие Дирихле  $u = 0$ .

## 2 Численный метод решения

Для решения используется **метод фиктивных областей**. Вне  $D$  вводится малая проницаемость  $1/\varepsilon$ , что обеспечивает приближение граничного условия  $u = 0$ . В прямоугольнике  $\Pi = [-1, 1] \times [-1, 1]$  решается модифицированное уравнение

$$-\nabla \cdot (k(x, y) \nabla v) = F(x, y), \quad v|_{\Gamma} = 0,$$

где

$$k(x, y) = \begin{cases} 1, & (x, y) \in D, \\ 1/\varepsilon, & (x, y) \notin D, \end{cases} \quad F(x, y) = \begin{cases} 1, & (x, y) \in D, \\ 0, & (x, y) \notin D. \end{cases}$$

Параметр  $\varepsilon = h^2$  обеспечивает выполнение граничного условия на криволинейной границе.

На равномерной сетке

$$x_i = -1 + ih, \quad y_j = -1 + jh, \quad i, j = 0..N, \quad h = \frac{2}{N},$$

применяется аппроксимация второго порядка:

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} = 1.$$

На узлах вне области  $D$  значение функции принимается равным нулю. Полученная система  $A w = B$  является разреженной, симметричной и положительно определённой.

### 3 Последовательный код программы (Задание 1)

В последовательной версии для решения разреженной симметричной положительно определённой системы

$$Aw = f,$$

полученной после дискретизации уравнения Пуассона в области сложной формы, используется **метод сопряжённых градиентов**. Матрица  $A$  не хранится явно; оператор применяется «на лету» с использованием пятиточечного шаблона.

Начальное приближение выбирается как  $w^{(0)} = 0$ . На каждой итерации вычисляются следующие величины:

$$\begin{aligned} r^{(k)} &= f - Aw^{(k)}, & p^{(k)} &= r^{(k)} + \beta^{(k-1)} p^{(k-1)}, \\ \alpha^{(k)} &= \frac{(r^{(k)}, r^{(k)})_E}{(p^{(k)}, Ap^{(k)})_E}, & w^{(k+1)} &= w^{(k)} + \alpha^{(k)} p^{(k)}, \\ r^{(k+1)} &= r^{(k)} - \alpha^{(k)} Ap^{(k)}, & \beta^{(k)} &= \frac{(r^{(k+1)}, r^{(k+1)})_E}{(r^{(k)}, r^{(k)})_E}. \end{aligned}$$

Итерационный процесс завершается, когда энергетическая норма невязки удовлетворяет условию

$$\|r^{(k)}\|_E < 10^{-6}.$$

Компиляция и запуск:

```
1 g++ -O3 -std=c++11 task1.cpp -o task1
2 ./task1
```

Результаты:

Расчёты выполнены на сетках  $10 \times 10$ ,  $20 \times 20$  и  $40 \times 40$ . Получены следующие значения количества итераций, невязки и времени работы:

$M$	$N$	Итераций	Остаток	Время (с)
10	10	132	$9.5789 \times 10^{-7}$	0.000588
20	20	572	$9.7168 \times 10^{-7}$	0.004901
40	40	2353	$9.9796 \times 10^{-7}$	0.059938

Метод сопряжённых градиентов существенно ускоряет сходимость по сравнению с методом скорейшего спуска: рост числа итераций соответствует теоретической оценке  $O(\sqrt{N})$  для задач типа Пуассона.

### 4 OpenMP

#### 4.1 Задание 2

В рамках задания 2 был реализован численный решатель для двумерного уравнения Пуассона на области-сапожке.

### 4.1.1 Используемый метод

Алгоритм относится к классу предобусловленных методов сопряженных градиентов и включает следующие этапы:

- вычисление остатка  $r = F - Aw$ ;
- применение диагонального предобуславливания

$$z = D^{-1}r,$$

где  $D$  — диагональная часть оператора  $A$ ;

- вычисление шага

$$\alpha = \frac{(r, z)}{(p, Ap)};$$

- обновление решения и остатка;
- обновление направления поиска

$$p_{k+1} = z_{k+1} + \beta_k p_k, \quad \beta_k = \frac{(r_{k+1}, z_{k+1})}{(r_k, z_k)}.$$

Использование диагонального предобуславливания Якоби улучшает спектральные свойства матрицы дискретизации и ускоряет сходимость по сравнению с обычным методом сопряженных градиентов.

### 4.1.2 Результаты расчёта для сетки $40 \times 40$

Threads	TOL	Количество итераций	Остаток	Время (s)
1	1e-6	3028	$9.99433 \times 10^{-7}$	1.01529
4	1e-6	3028	$9.99433 \times 10^{-7}$	0.356572
16	1e-6	3028	$9.99433 \times 10^{-7}$	0.258314

## 4.2 Задание 3

**Цель.** Повысить точность и ускорить сходимость за счёт использования потоковой (физически корректной) дискретизации и решателя **метода сопряжённых градиентов** с предобуславливанием Якоби.

В задаче `task3` коэффициенты  $a_{i\pm 1/2,j}$  и  $b_{i,j\pm 1/2}$  учитывают долю ячеек, принадлежащей области  $D$ , что обеспечивает точное соблюдение граничных условий и симметрию оператора.

### 4.2.1 Основные фрагменты

Предобуславливание и скалярное произведение:

```

1 for (int i=1; i<M; ++i)
2 for (int j=1; j<N; ++j)
3     z[i][j] = r[i][j] / diag[i][j];
4

```

```

5 | double rz = 0.0;
6 | for (int i=1; i<M; ++i)
7 | for (int j=1; j<N; ++j)
8 |     rz += r[i][j] * z[i][j];

```

Вычисление  $\alpha$  и обновление  $w, r$ :

```

1 applyA(p, Ap);
2 double pAp = dotE(p, Ap);
3 double alpha = rz / pAp;
4
5 for (int i=1; i<M; ++i)
6 for (int j=1; j<N; ++j){
7     w[i][j] += alpha * p[i][j];
8     r[i][j] -= alpha * Ap[i][j];
9 }

```

Вычисление  $\beta$  и обновление направления  $p$ :

```

1 applyDinv(r, z);
2 double rz_new = dotE(r, z);
3 double beta = rz_new / rz;
4
5 for (int i=1; i<M; ++i)
6 for (int j=1; j<N; ++j)
7     p[i][j] = z[i][j] + beta * p[i][j];
8
9 rz = rz_new;

```

#### 4.2.2 Компиляция и запуск

```

1 g++ -O3 -std=c++11 -fopenmp task3.cpp -o task3
2 ./task3

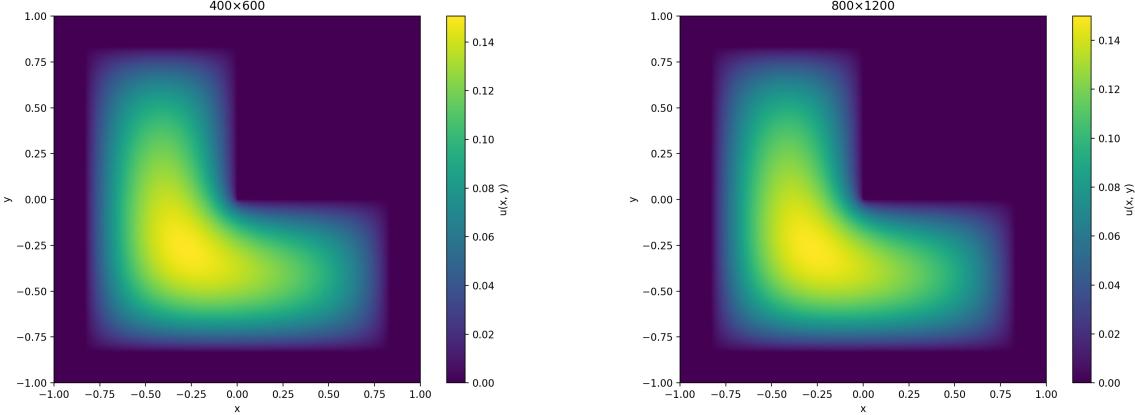
```

#### 4.2.3 Визуализация

На Рис. 1 видно, что максимум функции  $u(x, y)$  расположен внутри области, а на границе значение строго равно нулю. Более мелкая сетка даёт более гладкий профиль решения.

#### 4.2.4 Производительность и ускорение

Число потоков	M×N	Итераций	Время (с)	Ускорение
1	400×600	2749	256.698221	1.00
1	800×1200	5237	1948.584467	1.00



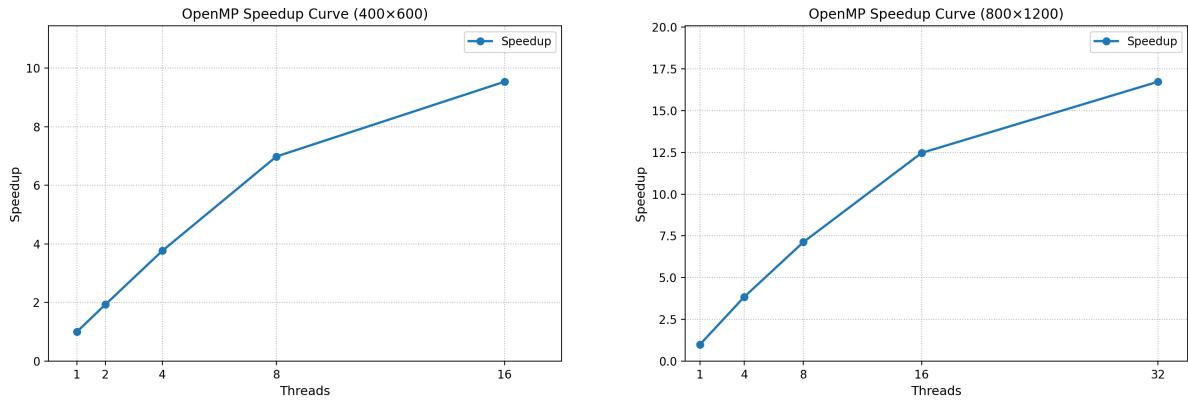
(a) Тепловая карта для сетки  $400 \times 600$

(b) Тепловая карта для сетки  $800 \times 1200$

Рис. 1: Тепловая карта OpenMP

Число потоков	$M \times N$	Итераций	Время (с)	Ускорение
2	$400 \times 600$	2749	132.982183	1.93
4	$400 \times 600$	2749	68.166429	3.77
8	$400 \times 600$	2749	36.760456	6.99
16	$400 \times 600$	2749	26.923483	9.54
4	$800 \times 1200$	5237	505.865782	3.852
8	$800 \times 1200$	5237	273.394149	7.127
16	$800 \times 1200$	5237	156.307519	12.473
32	$800 \times 1200$	5237	116.491348	16.731

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



(a)  $400 \times 600$

(b)  $800 \times 1200$

Рис. 2: Графики ускорения OpenMP

## 5 MPI

В данном разделе представлена распределённая версия решателя, основанная на двумерной декартовой топологии MPI и методе сопряжённых градиентов с пре-дубуславливанием Якоби.

## 5.1 Цель

Цель MPI-версии — выполнить вычисления на нескольких процессах с корректным разбиением области, обменом граничными слоями (halo-exchange) и глобальными редукциями, сохранив при этом математически эквивалентный предобусловленный метод сопряжённых градиентов, использованный в OpenMP-версии.

## 5.2 Декомпозиция области

Область  $[A_1, B_1] \times [A_2, B_2]$  разбивается между процессами с использованием декартовой топологии MPI:

$$P_x \times P_y = P, \quad \frac{(M+1)/P_x}{(N+1)/P_y} \approx 1,$$

что обеспечивает баланс вычислений и коммуникаций.

Каждый процесс получает:

- локальные размеры  $n_x, n_y$ ;
- глобальные смещения  $i_{\text{start}}, j_{\text{start}}$ ;
- четырёх соседей: left/right и up/down;
- пользовательские типы `row_type` и `col_type`.

## 5.3 Обмен граничными значениями (halo-exchange)

Для корректного применения оператора  $A$  выполняется обмен граничными полосами:

- горизонтальный обмен с типом `row_type`;
- вертикальный обмен с типом `col_type`;
- обмен реализован через MPI\_Sendrecv.

Этот шаг выполняется на каждой итерации метода сопряжённых градиентов (МСГ) перед вызовом оператора  $A$ .

## 5.4 Оператор $A$ и предобуславливание

Поточно-ориентированная дискретизация использует коэффициенты  $a_{i\pm 1/2,j}$  и  $b_{i,j\pm 1/2}$ , вычисляемые на основе доли локальной ячейки, лежащей внутри области  $D$ . Значения в “фиктивной” области масштабируются параметром  $1/\varepsilon$ , где  $\varepsilon = h_{\max}^2$ .

Предобуславливание (Якоби) имеет вид:

$$z_{ij} = \frac{r_{ij}}{D_{ij}}.$$

## 5.5 Итерационный процесс предобусловленного метода сопряжённых градиентов

Алгоритм полностью совпадает с OpenMP-версией, но все скалярные величины вычисляются с использованием MPI\_Allreduce.

Итерации продолжаются, пока выполняются условия:

$$\|r^{(k)}\|_E \leq \varepsilon_{\text{rel}} \|r^{(0)}\|_E, \quad \|w^{(k)} - w^{(k-1)}\|_E < \delta.$$

В расчётах использованы параметры:

$$\varepsilon_{\text{rel}} = 10^{-6}, \quad \delta = 10^{-8}.$$

## 5.6 Основные фрагменты

Предобусловливание и вычисление скалярных произведений с использованием глобальных редукций MPI:

```
1 applyDinv_local(d, r, z);
2
3 double rz = dotE_global(d, r, z);
```

Обмен граничными слоями (halo-exchange) перед применением оператора A:

```
1 exchange_halo(d, p);
2 applyA_local(d, p, Ap);
```

Вычисление  $\alpha$  и обновление  $w, r$ :

```
1 double pAp = dotE_global(d, p, Ap);
2 double alpha = rz / pAp;
3
4 for (int i = 1; i <= d.nx; ++i)
5   for (int j = 1; j <= d.ny; ++j) {
6     int gi = d.istart + (i - 1);
7     int gj = d.jstart + (j - 1);
8     if (gi <= 0 || gi >= d.M_global ||
9         gj <= 0 || gj >= d.N_global) continue;
10
11    int id = IDX(d, i, j);
12    w_prev[id] = w[id];
13    w[id]      += alpha * p[id];
14    r[id]      -= alpha * Ap[id];
15 }
```

Вычисление нормы шага и нормы невязки:

```
1 double rr = dotE_global(d, r, r);
2 double norm = sqrt(rr);
3
4 for (int i = 1; i <= d.nx; ++i)
5   for (int j = 1; j <= d.ny; ++j) {
6     int gi = d.istart + (i - 1);
7     int gj = d.jstart + (j - 1);
```

```

8   if (gi <= 0 || gi >= d.M_global ||
9      gj <= 0 || gj >= d.N_global) continue;
10    int id = IDX(d, i, j);
11    dw[id] = w[id] - w_prev[id];
12  }
13
14 double dxE = sqrt(dotE_global(d, dw, dw));

```

Вычисление  $\beta$  и обновление направления  $p$ :

```

1 applyDinv_local(d, r, z);
2 double rz_new = dotE_global(d, r, z);
3
4 double beta = rz_new / rz;
5 rz = rz_new;
6
7 for (int i = 1; i <= d.nx; ++i)
8 for (int j = 1; j <= d.ny; ++j) {
9    int gi = d.istart + (i - 1);
10   int gj = d.jstart + (j - 1);
11   if (gi <= 0 || gi >= d.M_global ||
12      gj <= 0 || gj >= d.N_global) continue;
13
14   int id = IDX(d, i, j);
15   p[id] = z[id] + beta * p[id];
16 }

```

## 5.7 Сбор глобального решения

После завершения итераций каждый процесс отправляет:

- размеры локального блока,
- позицию в глобальной сетке,
- локальное решение  $w_{loc}$ .

Процесс `rank = 0` собирает данные через `MPI_Gatherv` и формирует глобальный массив:

$$w_{\text{global}}(i, j), \quad 0 \leq i \leq M, 0 \leq j \leq N.$$

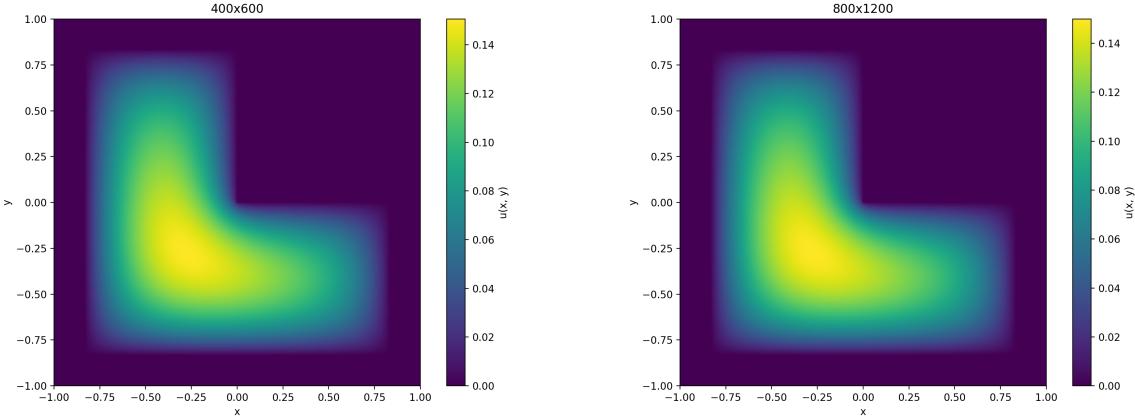
Результат записывается в файл:

`result_mpi_MxN_tP.txt`.

Визуализация результатов приведена на Рис. 3.

## 5.8 Результаты расчёта для сетки $40 \times 40$

Threads	TOL	Количество итераций	Время (с)
1	1e-6	162	0.028048
2	1e-6	162	0.020376
4	1e-6	162	0.008651



(a) Термальная карта для сетки  $400 \times 600$

(b) Термальная карта для сетки  $800 \times 1200$

Рис. 3: Термальная карта MPI

## 5.9 Производительность и ускорение

Число потоков	$M \times N$	Итераций	Время (с)	Ускорение
1	$400 \times 600$	2327	64.907571	1.000
1	$800 \times 1200$	4486	472.104478	1.000

Число потоков	$M \times N$	Итераций	Время (с)	Ускорение
2	$400 \times 600$	2327	31.796831	2.041
4	$400 \times 600$	2327	16.208656	4.004
8	$400 \times 600$	2327	8.369234	7.756
16	$400 \times 600$	2327	5.433574	11.944
4	$800 \times 1200$	4486	124.989525	3.776
8	$800 \times 1200$	4486	64.099953	7.365
16	$800 \times 1200$	4486	32.918410	14.345
32	$800 \times 1200$	4486	23.771534	19.856

Таблица 2: Таблица с результатами расчетов на ПВС IBM Polus (MPI код).

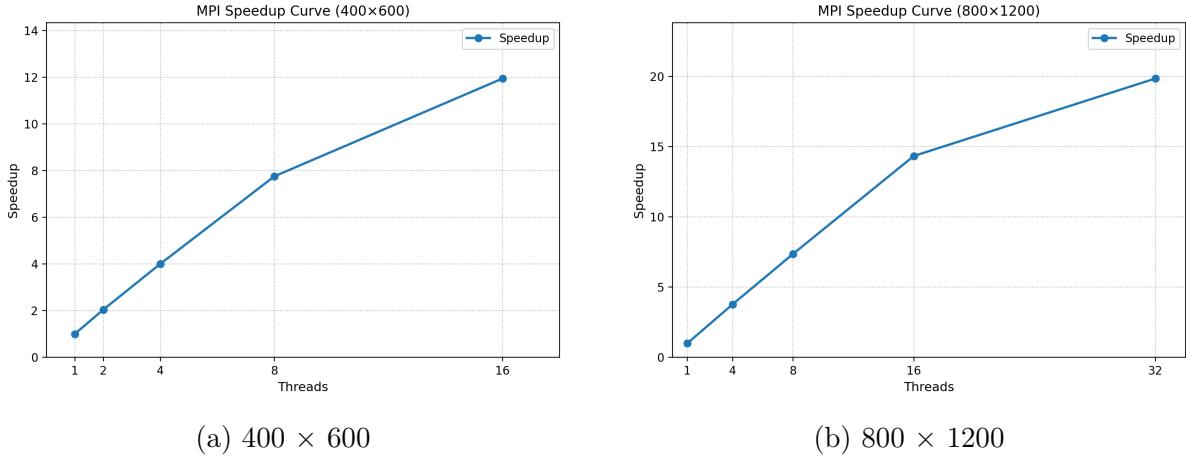


Рис. 4: Графики ускорения MPI

## 6 MPI+OpenMP

В данном разделе представлена гибридная версия решателя, основанная на двумерной декартовой топологии MPI и методе сопряжённых градиентов с якобианским предобуславливанием. Вычисления внутри каждого MPI-процесса дополнительно распараллелены с использованием OpenMP.

### 6.1 Цель

Цель реализации MPI+OpenMP-версии — выполнить вычисления на нескольких процессах с корректным разбиением области, обменом граничными слоями (halo-exchange) и глобальными редукциями, сохраняя при этом математически эквивалентный предобусловленный метод сопряжённых градиентов.

### 6.2 Декомпозиция области

Область  $[A_1, B_1] \times [A_2, B_2]$  разбивается между процессами с использованием двумерной декартовой топологии MPI:

$$P_x \times P_y = P, \quad \frac{M+1}{P_x} \approx \frac{N+1}{P_y} \approx 1,$$

что обеспечивает баланс вычислений и коммуникаций.

Каждый MPI-процесс получает:

- локальные размеры  $n_x, n_y$ ;
- глобальные смещения  $i_{\text{start}}, j_{\text{start}}$ ;
- четырёх соседей (left/right, up/down);
- пользовательские типы данных `row_type` и `col_type` для обмена граничными слоями.

### 6.3 Обмен граничными значениями (halo-exchange)

Для корректного применения оператора  $A$  выполняется обмен граничными слоями между соседними поддоменами:

- горизонтальный обмен с использованием типа `row_type`;
- вертикальный обмен с использованием типа `col_type`;
- обмен реализован неблокирующими вызовами MPI.

Данный шаг выполняется на каждой итерации метода сопряжённых градиентов перед применением оператора  $A$ .

### 6.4 Оператор $A$ и предобусловливание

Дискретизация оператора Лапласа осуществляется по пятиточечному шаблону:

$$(Au)_{ij} = -u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} + 4u_{i,j}.$$

Вычисления выполняются только в узлах, принадлежащих вычислительной области, что определяется логической маской.

В качестве предобусловливателя используется якобиановское предобусловливание:

$$z_{ij} = \frac{r_{ij}}{D_{ij}},$$

где  $D_{ij}$  — диагональный элемент матрицы оператора. Для пятиточечного оператора Лапласа  $D_{ij} = 4$ .

### 6.5 Итерационный процесс предобусловленного метода сопряжённых градиентов

Итерационный алгоритм соответствует классическому PCG-методу. Все скалярные произведения вычисляются с использованием глобальных редукций MPI (`MPI_Allreduce`).

Итерации продолжаются до выполнения условия:

$$\|r^{(k)}\|_2 < \varepsilon, \quad \varepsilon = 10^{-8}.$$

### 6.6 Основные фрагменты реализации

Предобусловливание и вычисление скалярных произведений с использованием глобальных редукций:

```
1 applyDinv(d, r, z);
2 double rz = dotE_global(d, r, z);
```

Обмен граничными слоями перед применением оператора  $A$ :

```
1 exchange_halo(d, p);
2 applyA_local(d, p, Ap);
```

Вычисление шага  $\alpha$  и обновление решения и невязки:

```

1 double pAp    = dotE_global(d, p, Ap);
2 double alpha = rz / pAp;
3
4 w[id] += alpha * p[id];
5 r[id] -= alpha * Ap[id];

```

Вычисление коэффициента `beta` и обновление направления:

```

1 applyDinv(d, r, z);
2 double rz_new = dotE_global(d, r, z);
3 double beta   = rz_new / rz;
4
5 p[id] = z[id] + beta * p[id];
6 rz = rz_new;

```

Все локальные циклы в приведённых фрагментах распараллелены с использованием директив OpenMP.

## 6.7 Сбор глобального решения

После завершения итерационного процесса каждый процесс хранит локальную часть решения  $w_{\text{loc}}$ . Процесс с рангом 0 собирает данные и формирует глобальное решение  $w_{\text{global}}(i, j)$ . Результаты расчёта используются для последующей визуализации.

Визуализация результатов приведена на Рис. 5.

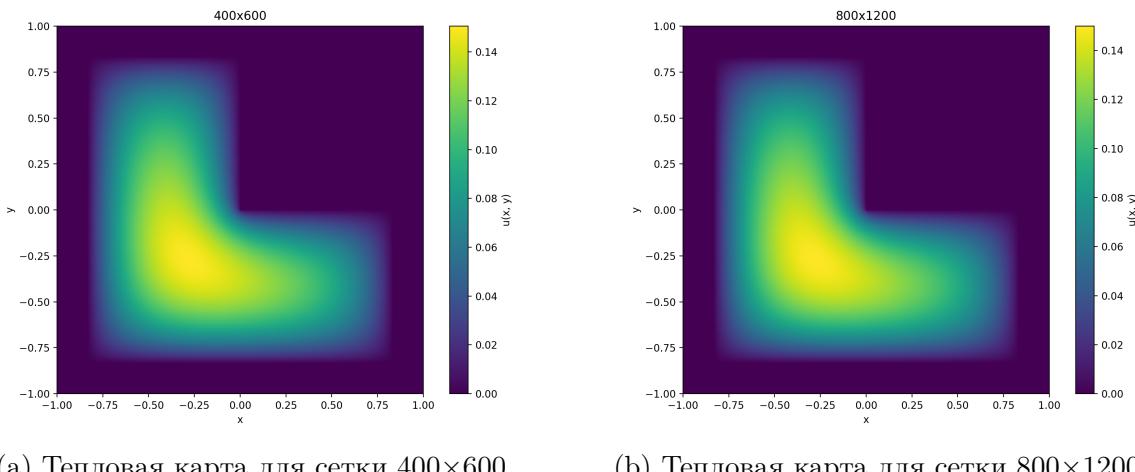


Рис. 5: Тепловая карта MPI+OpenMP

## 6.8 Результаты расчёта для сетки $40 \times 40$

Количество MPI	Количество OpenMP-нитей в процессе	Число точек сетки ( $M \times N$ )	Число итераций	Время решения
1	4	$40 \times 40$	90	0.004712
2	4	$40 \times 40$	90	0.003478

## 6.9 Производительность и ускорение

Количество процессов MPI	Количество OpenMP-нитей в процессе	Число точек сетки ( $M \times N$ )	Число итераций	Время решения	Ускорение
2	1	$400 \times 600$	1412	7.500332	1.00
2	2	$400 \times 600$	1412	6.343893	1.18
2	4	$400 \times 600$	1412	4.000413	1.87
2	8	$400 \times 600$	1412	2.988352	2.51
4	1	$800 \times 1200$	2922	38.700510	1.00
4	2	$800 \times 1200$	2922	27.167954	1.42
4	4	$800 \times 1200$	2922	19.100784	2.03
4	8	$800 \times 1200$	2922	62.186709	-

Таблица 3: Таблица с результатами расчетов на ПВС IBM Polus (MPI+OpenMP код).

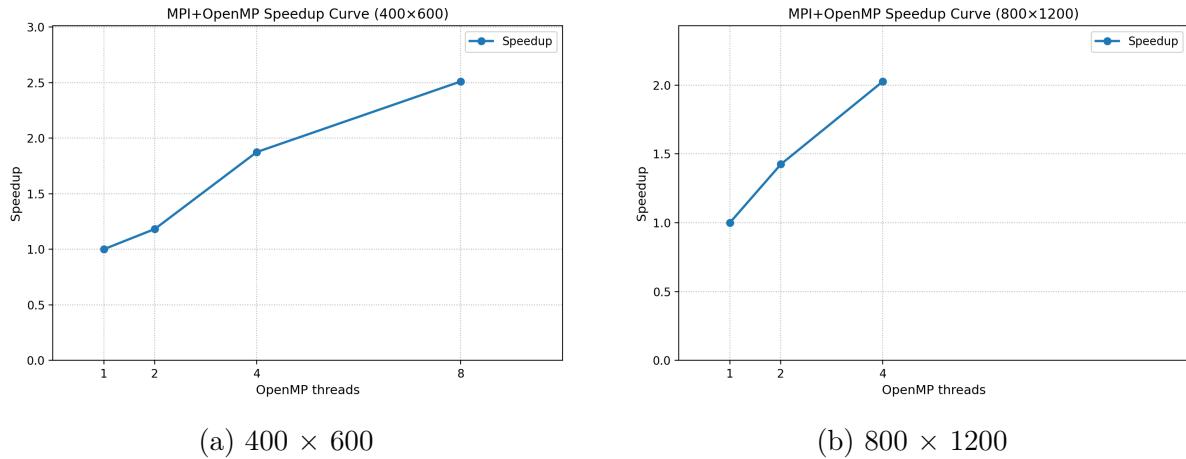


Рис. 6: Графики ускорения MPI+OpenMP

## 7 Ссылка на репозиторий Git

<https://github.com/JerryRen-41/SuperComputer>