

Отчёт по заданию: Решение двумерного уравнения Пуассона методом фиктивных областей с использованием OpenMP, MPI и их гибридного сочетания (MPI+OpenMP)

Вариант: №9

Студент: Ма Синьюэ 617

Цель работы

Целью данной работы является численное решение двумерного уравнения Пуассона с использованием метода фиктивных областей, а также разработка и исследование параллельных алгоритмов на основе технологий OpenMP, MPI и их гибридного сочетания (MPI+OpenMP).

Постановка задачи

Для успешного выполнения задания требуется:

1. Разработать последовательный код программы, вычисляющий приближённое решение разностной схемы методом скорейшего спуска, выполнить расчёты на сгущающихся сетках:

$$(M, N) = (10, 10), (20, 20), (40, 40).$$

2. Используя средства OpenMP, разработать параллельный код программы, вычисляющий приближённое решение разностной схемы, проверить качество работы алгоритма, выполнив расчёты на сетке

$$(M, N) = (40, 40)$$

при числе потоков 1, 4, и 16, провести сравнение с последовательным вариантом алгоритма.

3. Заполнить таблицу с результатами расчётов OpenMP-программы.
4. Разработать и реализовать алгоритм **двумерного разбиения** прямоугольного контейнера $\Pi = [A_1, B_1] \times [A_2, B_2]$ на домены (подобласти) так, чтобы
 - отношение количества узлов по переменным x и y в каждом домене принадлежало диапазону $[1/2, 2]$;

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

5. Используя средства библиотеки MPI и разработанный алгоритм разбиения, реализовать параллельный вариант метода сопряжённых градиентов для решения двумерного уравнения Пуассона. Проверить качество работы алгоритма, выполнив расчёты на сетке

$$(M, N) = (40, 40)$$

на одном, двух и четырёх процессах, провести сравнение с последовательным вариантом алгоритма по числу итераций и времени решения.

6. Выполнить серию вычислительных экспериментов на более крупных сетках

$$(M, N) = (400, 600), \quad (M, N) = (800, 1200)$$

при различном числе MPI-процессов и заполнить Таблицу 2 с результатами расчётов (число итераций, время решения и ускорение относительно последовательного запуска).

7. Разработать гибридный код программы, добавив в MPI-код директивы OpenMP, проверить качество работы алгоритма, выполнив расчёты на сетке

$$(M, N) = (40, 40)$$

на одном и двух процессах с четырьмя нитями, провести сравнение с последовательным вариантом алгоритма.

8. Заполнить Таблицу 3 с результатами расчётов гибридной программы на ПВС IBM Polus: для сеток

$$(M, N) = (400, 600), \quad (M, N) = (800, 1200)$$

при числе MPI-процессов 2 и 4 и количестве OpenMP-нитей в процессе 1, 2, 4 и 8, указав число итераций, время решения и ускорение относительно последовательного варианта алгоритма.

Метод и реализация

Область, сетка и правая часть

Рассматривается эллиптическая область

$$D = \{(x, y) : x^2 + 4y^2 < 1\},$$

вложенная в прямоугольный контейнер

$$A_1 = -1, \quad B_1 = 1, \quad A_2 = -0.6, \quad B_2 = 0.6.$$

Сетка равномерная:

$$x_i = A_1 + i h_x, \quad i = 0, \dots, M; \quad y_j = A_2 + j h_y, \quad j = 0, \dots, N,$$

где $h_x = \frac{B_1 - A_1}{M}$, $h_y = \frac{B_2 - A_2}{N}$. Правая часть строится как индикатор области:

$$B_{ij} = \begin{cases} F_{\text{VAL}} = 1, & (x_i, y_j) \in D, \\ 0, & (x_i, y_j) \notin D. \end{cases}$$

Метод фиктивных областей и выбор ε

Коэффициент теплопроводности задаётся по правилу

$$k(x, y) = \begin{cases} 1, & (x, y) \in D, \\ \frac{1}{\varepsilon}, & (x, y) \notin D, \end{cases} \quad \boxed{\varepsilon = (\max\{h_x, h_y\})^2}.$$

(Именно такая формула ε используется в коде: `eps = max(h1, h2)*max(h1, h2);`).

Коэффициенты схемы a_{ij} , b_{ij}

Для ячейки (i, j) вычисляются доли пересечения её граней с областью D . Длины пересечения отрезков с эллипсом:

$$x = \text{const} = x_0 : \quad y \in \left[-\sqrt{\frac{1-x_0^2}{4}}, \sqrt{\frac{1-x_0^2}{4}} \right], \quad y = \text{const} = y_0 : \quad x \in \left[-\sqrt{1-4y_0^2}, \sqrt{1-4y_0^2} \right].$$

Пусть $\ell_{ij}^{(v)}$ — часть вертикального ребра длиной h_y , лежащая в D , а $\ell_{ij}^{(h)}$ — часть горизонтального ребра длиной h_x , лежащая в D . Тогда (ровно как в функции `fictitious_regions_setup`):

$$a_{ij} = \begin{cases} 1, & \ell_{ij}^{(v)} = h_y, \\ \frac{1}{\varepsilon}, & \ell_{ij}^{(v)} = 0, \\ \frac{\ell_{ij}^{(v)}}{h_y} + \frac{1 - \frac{\ell_{ij}^{(v)}}{h_y}}{\varepsilon}, & \text{иначе,} \end{cases} \quad b_{ij} = \begin{cases} 1, & \ell_{ij}^{(h)} = h_x, \\ \frac{1}{\varepsilon}, & \ell_{ij}^{(h)} = 0, \\ \frac{\ell_{ij}^{(h)}}{h_x} + \frac{1 - \frac{\ell_{ij}^{(h)}}{h_x}}{\varepsilon}, & \text{иначе.} \end{cases}$$

Действие оператора A

Разностный оператор для внутренних узлов ($i = 1..M - 1, j = 1..N - 1$) реализован как

$$(Aw)_{ij} = -\frac{1}{h_x} \left(a_{i+1,j} \frac{w_{i+1,j} - w_{i,j}}{h_x} - a_{i,j} \frac{w_{i,j} - w_{i-1,j}}{h_x} \right) - \frac{1}{h_y} \left(b_{i,j+1} \frac{w_{i,j+1} - w_{i,j}}{h_y} - b_{i,j} \frac{w_{i,j} - w_{i,j-1}}{h_y} \right).$$

(См. функцию `apply_A`).

Диагональный предобусловливатель D^{-1}

Используется диагональное предобусловливание (см. `apply_Dinv`):

$$D_{ij} = \frac{a_{i+1,j} + a_{i,j}}{h_x^2} + \frac{b_{i,j+1} + b_{i,j}}{h_y^2}, \quad z_{ij} = (D^{-1}r)_{ij} = \frac{r_{ij}}{D_{ij}}.$$

Метод сопряжённых градиентов (PCG)

Итерационный процесс в `solve`:

$$\begin{aligned} r^{(0)} &= B, \quad z^{(0)} = D^{-1}r^{(0)}, \quad p^{(1)} = z^{(0)}, \quad \langle u, v \rangle = \sum_{i,j} u_{ij}v_{ij} h_x h_y; \\ \text{на шаге } k : \quad \alpha_k &= \frac{\langle z^{(k-1)}, r^{(k-1)} \rangle}{\langle Ap^{(k)}, p^{(k)} \rangle}, \quad w^{(k)} = w^{(k-1)} + \alpha_k p^{(k)}, \\ r^{(k)} &= r^{(k-1)} - \alpha_k A p^{(k)}, \quad z^{(k)} = D^{-1}r^{(k)}, \quad \beta_{k+1} = \frac{\langle z^{(k)}, r^{(k)} \rangle}{\langle z^{(k-1)}, r^{(k-1)} \rangle}, \\ p^{(k+1)} &= z^{(k)} + \beta_{k+1} p^{(k)}. \end{aligned}$$

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

где $\|\cdot\|_E$ — евклидова норма, а δ задаётся пользователем (в тестах $\delta = 10^{-6}$).

Параллельная реализация OpenMP

Параллелизация выполняется с помощью директив:

```
#pragma omp parallel for collapse(2)
```

Они применяются в циклах:

- вычисление коэффициентов a_{ij}, b_{ij}, F_{ij} ;
- операция Aw ;
- обновление векторов r, z, p, w .

Время измеряется с помощью `omp_get_wtime()`.

Параллельная реализация MPI

Для распараллеливания разностной схемы используется библиотека MPI. Разбиение прямоугольника по условию задания (пункт 4) осуществляется по переменной x на несколько подобластей, каждая из которых обрабатывается отдельным MPI-процессом.

Двумерное разбиение области

Пусть всего P процессов и $(M - 1) \times (N - 1)$ внутренних узлов по координатам x и y . В программе сначала выбирается двумерная решётка процессов $P_x \times P_y$ (функция `choose_process_grid(P, Px, Py)`), где $P_x P_y = P$ и решётка по возможности близка к квадратной.

Затем функция `decompose_2d(M, N, Px, Py, rank, i_start, i_end, j_start, j_end)` делит множество внутренних индексов

$$i = 1, \dots, M - 1, \quad j = 1, \dots, N - 1$$

между процессами так, что каждому процессу с координатами (p_x, p_y) в решётке соответствует прямоугольный блок

$$i_{\min}^{(p)} \leq i \leq i_{\max}^{(p)}, \quad j_{\min}^{(p)} \leq j \leq j_{\max}^{(p)},$$

где

$$n_x^{(p)} = i_{\max}^{(p)} - i_{\min}^{(p)} + 1, \quad n_y^{(p)} = j_{\max}^{(p)} - j_{\min}^{(p)} + 1$$

— число внутренних узлов по x и y на данном процессе. Размеры блоков по каждому направлению отличаются не более чем на единицу:

$$|n_x^{(p)} - n_x^{(q)}| \leq 1, \quad |n_y^{(p)} - n_y^{(q)}| \leq 1$$

для любых процессов p, q . Тем самым выполняется требование задания о двумерном разбиении прямоугольника: каждый MPI-процесс отвечает за собственный подпрямоугольник сетки.

Для удобства используется локальная нумерация

$$l_i = i - i_{\min}^{(p)} + 1, \quad l_j = j - j_{\min}^{(p)} + 1,$$

где $l_i = 1, \dots, n_x^{(p)}$, $l_j = 1, \dots, n_y^{(p)}$. Дополнительно по каждому направлению вводятся *призрачные* (halo) слои с индексами $l_i = 0$, $l_i = n_x^{(p)} + 1$ и $l_j = 0$, $l_j = n_y^{(p)} + 1$, в которых хранятся значения граничных узлов, полученные от соседних процессов при обмене данными.

Локальные операции и обмен граничной информацией

Коэффициенты a_{ij}, b_{ij} и правая часть B_{ij} вычисляются локально в процедуре `fictitious_regions_se` только для принадлежащих процессу индексов $i = i_{\min}^{(p)}, \dots, i_{\max}^{(p)}$ (плюс один слой по x для корректного вычисления потоков через границы). Формулы полностью совпадают с последовательным случаем.

Действие оператора A над вектором w реализовано в функции `apply_A_local`: во внутренних узлах ($l_i = 1..n_p, j = 1..N - 1$) используется та же разностная формула, что и в последовательной версии, но все вычисления ведутся только по локальным индексам. Перед каждым вызовом `apply_A_local` выполняется обмен граничными слоями между соседними процессами (`exchange_halos()`), где используются пары вызовов `MPI_Sendrecv`. На крайних процессах граничные слои, соответствующие границе контейнера, задаются равными нулю, что соответствует граничным условиям Дирихле.

Диагональный предобусловливатель D^{-1} также строится локально (`apply_Dinv_local`): для каждого внутреннего узла процесс вычисляет

$$D_{ij} = \frac{a_{i+1,j} + a_{i,j}}{h_x^2} + \frac{b_{i,j+1} + b_{i,j}}{h_y^2}, \quad z_{ij} = \frac{r_{ij}}{D_{ij}},$$

используя только локальные данные и призрачные слои.

Параллельный метод сопряжённых градиентов

Векторные операции метода сопряжённых градиентов выполняются локально, а глобальные скалярные произведения вычисляются в два шага:

1. каждый процесс считает свою локальную сумму $\sum_{l_i,j} u_{ij}v_{ij} h_x h_y$ (функция `dot_local`);
2. все локальные суммы суммируются при помощи `MPI_Allreduce` с операцией `MPI_SUM`, формируя глобальное скалярное произведение.

Аналогично собирается глобальная норма $\|w^{(k+1)} - w^{(k)}\|_E$, которая используется в критерии остановки:

$$\|w^{(k+1)} - w^{(k)}\|_E < \delta, \quad \delta = 10^{-6}.$$

После каждого шага все процессы получают одинаковое значение нормы, поэтому решение синхронно прекращает итерации.

Вся логика метода PCG вынесена в функцию `gradient_solver_mpi`, в которой используются описанные выше локальные операции и глобальные коллективные вызовы MPI.

Гибридная реализация MPI+OpenMP

Гибридный вариант объединяет распределение данных по MPI и внутрипроцессную распараллелизацию по OpenMP. Разбиение области по переменной x , локальная нумерация узлов, вычисление коэффициентов a_{ij}, b_{ij}, B_{ij} , обмен граничными слоями и все коллективные

операции (`MPI_Allreduce`, `MPI_Sendrecv`) полностью совпадают с чисто MPI-реализацией, описанной выше.

Внутри каждого MPI-процесса наиболее трудоёмкие циклы распараллеливаются директивами OpenMP вида

```
#pragma omp parallel for
```

а при накоплении локальных норм и скалярных произведений — с использованием редукции:

```
#pragma omp parallel for reduction(+:local_sum)
```

К таким циклам относятся:

- вычисление локальных коэффициентов a_{ij}, b_{ij}, B_{ij} в процедуре `fictitious_regions_setup_local`;
- применение оператора A к вектору w (`apply_A_local`);
- применение диагонального предобуславливателя (`apply_Dinv_local`);
- обновление векторов w, r, z, p и вычисление локального вклада в норму $\|w^{(k+1)} - w^{(k)}\|_E$ в основной итерации метода.

Число нитей OpenMP в каждом процессе задаётся переменной среды `OMP_NUM_THREADS` или опцией `affinity[core(N)]` в командном файле LSF. Временные характеристики гибридного варианта измеряются на процессе с рангом 0 с помощью функции `MPI_Wtime()`, что позволяет корректно сравнивать полученное ускорение с последовательной и чисто MPI-реализацией на сетках $(M, N) = (40, 40), (400, 600)$ и $(800, 1200)$.

Результаты расчётов

1. Последовательная версия

Размер сетки	Число итераций	Время решения
10×10	17	0.0001
20×20	31	0.0005
40×40	60	0.0034

2. OpenMP

Размер сетки	Число нитей	Число итераций	Время решения
40×40	1	60	0.005
40×40	4	60	0.004
40×40	16	60	0.009

Размер сетки ($M \times N$)	Число нитей	Число итераций	Время решения	Ускорение
400×600	2	546	3.098	1.59
400×600	4	546	1.776	2.78
400×600	8	546	1.170	4.21
400×600	16	546	0.977	5.05
800×1200	4	989	12.214	3.19
800×1200	8	989	7.554	5.15
800×1200	16	989	6.034	6.45
800×1200	32	989	6.216	6.27

3. MPI

Размер сетки	Число процессов	Число итераций	Время с MPI	Время с Openmp
40×40	1	60	0.00328	0.005
40×40	2	60	0.00186	0.004
40×40	4	60	0.00189	0.004

Размер сетки	Число процессов	Число итераций	Время решения	Ускорение
400×600	2	546	2.646	1.86
400×600	4	546	1.432	3.44
400×600	8	546	0.905	5.45
400×600	16	546	0.438	11.25
800×1200	4	989	9.689	4.02
800×1200	8	989	5.022	7.75
800×1200	16	989	2.906	13.40
800×1200	32	989	2.382	16.35

4. MPI+OpenMP

Размер сетки	Число процессов	Число нитей	Число итераций	Время с Openmp и MPI	Время с MPI
40×40	1	1	60	0.0125	0.00328
40×40	1	4	60	0.0100	0.00186
40×40	2	4	60	0.1204	0.00189

Размер сетки	Число процессов	Число нитей	Число итераций	Время решения	Ускорение
400×600	2	1	546	1.911	2.58
400×600	2	2	546	1.151	4.28
400×600	2	4	546	0.597	8.26
400×600	2	8	546	0.313	15.75
800×1200	4	1	989	7.636	5.10
800×1200	4	2	989	4.205	9.26
800×1200	4	4	989	2.138	18.21
800×1200	4	8	989	1.103	35.30

Рисунки

1. Рисунок приближенного решения на сетке 800×1200

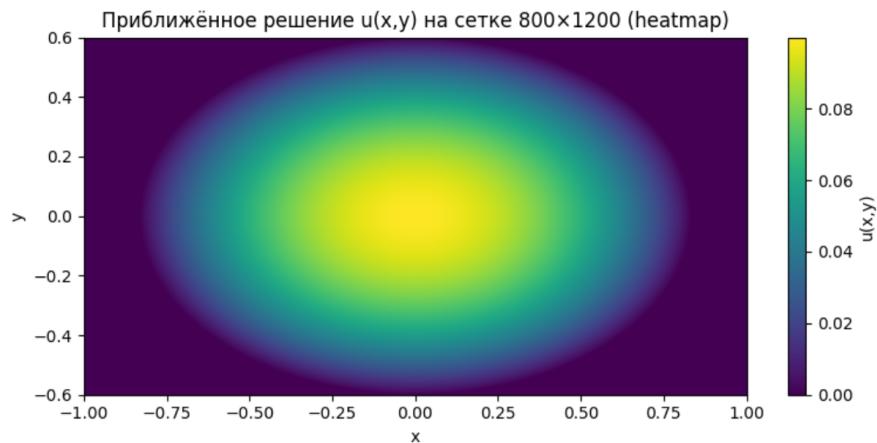


Рис. 1: Приближённое решение на сетке 800×1200 (вид сверху, 2D-представление).

Приближённое решение на сетке 800×1200

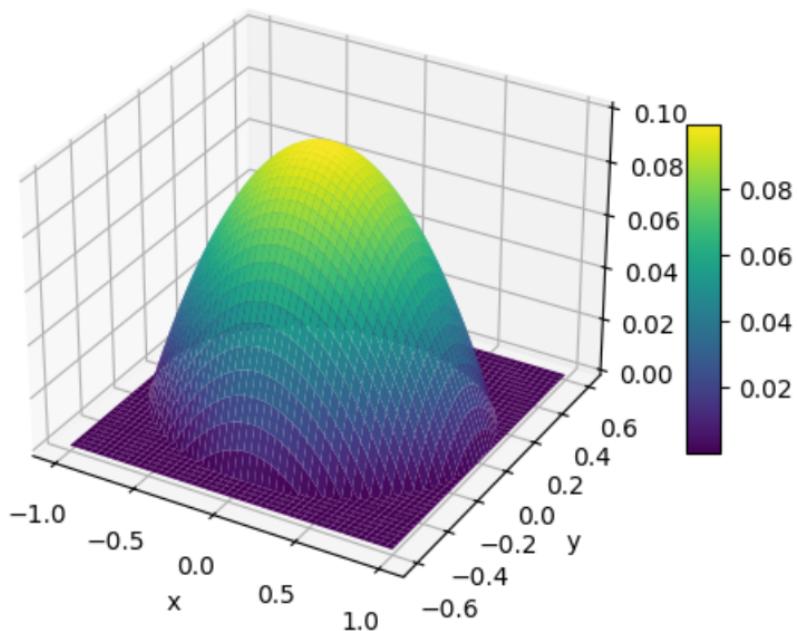


Рис. 2: Приближённое решение на сетке 800×1200 (3D-график).

2. Рисунки ускорений

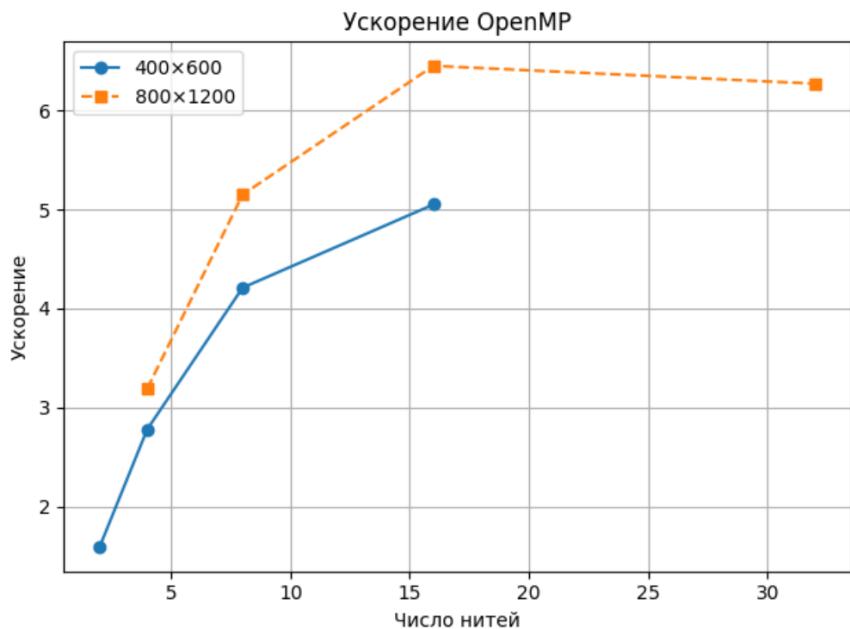


Рис. 3: Ускорение OpenMP-программы для сеток 400×600 и 800×1200 .

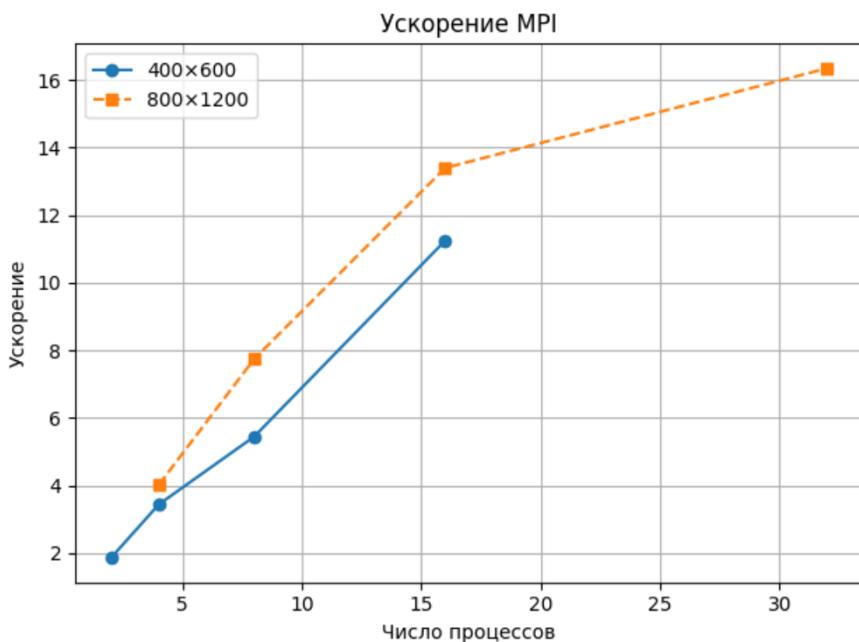


Рис. 4: Ускорение MPI-программы для сеток 400×600 и 800×1200 .

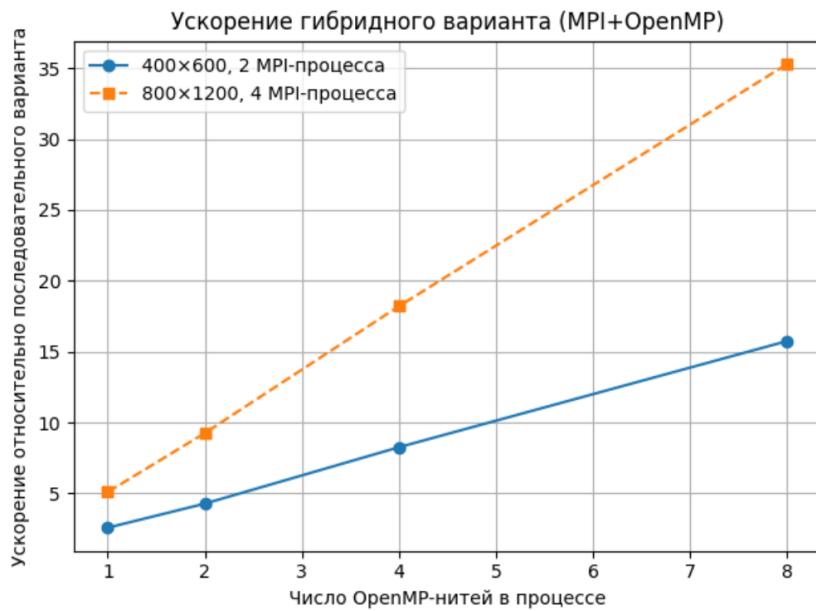


Рис. 5: Ускорение MPI и Openmp-программы для сеток 400×600 и 800×1200 .