

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

Отчёт по заданию 3 в рамках курса
«Суперкомпьютерное моделирование и
технологии»

Выполнила:
Лапенко Юлия Андреевна
Группа 627
Вариант 2

1 Описание условия

В трехмерной замкнутой области $\Omega = [0 \leq x \leq L_x] \times [0 \leq y \leq L_y] \times [0 \leq z \leq L_z]$ для $(0 < t \leq T]$ требуется найти решение $u(x, y, z, t)$ уравнения в частных производных:

$$\frac{\partial^2 u}{\partial t^2} = \Delta u$$

с начальными условиями

$$\begin{aligned} u|_{t=0} &= \varphi(x, y, z) \\ \frac{\partial u}{\partial t}|_{t=0} &= 0 \end{aligned}$$

при условии, что на границах области заданы следующие граничные условия:

$$\begin{aligned} u(0, y, z, t) &= 0, & u(L_x, y, z, t) &= 0, \\ u(x, 0, z, t) &= 0, & u(x, L_y, z, t) &= 0, \\ u(x, y, 0, t) &= u(x, y, L_z, t), & u_z(x, y, 0, t) &= u_z(x, y, L_z, t). \end{aligned}$$

2 Численный метод решения задачи

Для численного решения задачи введем на Ω сетку $\bar{\omega}_{h\tau} = \omega_h \times \omega_\tau$, где

$$T = T_0,$$

$$L_x = L_{x0}, L_y = L_{y0}, L_z = L_{z0}$$

$$\bar{\omega}_h = \{(x_i = ih_x, y_j = jh_y, z_k = kh_z), i, j, k = 0, 1, \dots, N, h_x N = L_x, h_y N = L_y, h_z N = L_z\},$$

$$\omega_\tau = \{t_n = n\tau, n = 0, 1, \dots, K, \tau K = T\}.$$

Обозначения: ω_h – множество внутренних узлов сетки $\bar{\omega}_h$, а γ_h – множество граничных сетки.

Аналитическая функция $u(x, y, z, t)$ для варианта 2 выглядит так:

$$u(x, y, z, t) = \sin\left(\frac{\pi}{L_x}x\right) \cdot \sin\left(\frac{\pi}{L_y}y\right) \cdot \sin\left(\frac{2\pi}{L_z}z\right) \cdot \cos(a_t \cdot t + 2\pi), \quad a_t = \pi \sqrt{\frac{1}{L_x^2} + \frac{1}{L_y^2} + \frac{4}{L_z^2}}$$

Задание значений на слое u_{ijk}^0 происходит следующим образом:

$$u_{ijk}^0 = \varphi(x_i, y_j, z_k), \quad \text{где } (x_i, y_j, z_k) \in \bar{\omega}_h, \quad \varphi(x_i, y_j, z_k) = u(x_i, y_j, z_k, 0)$$

Значения в узлах сетки u_{ijk}^1 вычисляются так:

$$\begin{aligned} u_{ijk}^1 &= u_{ijk}^0 + \frac{\tau^2}{2} \Delta_h \varphi(x_i, y_j, z_k), \quad \text{где } (x_i, y_j, z_k) \in \omega_h, \\ u_{ijk}^1 &= u(x_i, y_j, z_k, \tau), \quad \text{где } (x_i, y_j, z_k) \in \gamma_h. \end{aligned}$$

Разностная аппроксимация для граничного условия по z :

$$u_{ij0}^{n+1} = u_{ijN}^{n+1}, \quad u_{ij1}^{n+1} = u_{ijN+1}^{n+1}, \quad \text{где } i, j, k = 0, 1, \dots, N.$$

Значения на последующих слоях вычисляются следующим образом:

$$\frac{u_{ijk}^{n+1} - 2u_{ijk}^n + u_{ijk}^{n-1}}{\tau^2} = \Delta_h u^n, \quad (x_i, y_j, z_k) \in \omega_h, \quad 1, 2, \dots, K-1,$$

где Δ_h – семиточечный разностный аналог оператора Лапласа:

$$\Delta_h u^n = \frac{u_{i-1,j,k}^n - 2u_{i,j,k}^n + u_{i+1,j,k}^n}{h_x^2} + \frac{u_{i,j-1,k}^n - 2u_{i,j,k}^n + u_{i,j+1,k}^n}{h_y^2} + \frac{u_{i,j,k-1}^n - 2u_{i,j,k}^n + u_{i,j,k+1}^n}{h_z^2}$$

3 Реализация решения задачи

Для начала нужно вычислить, как узлы сетки будут распределены по процессорам. Реализуем блочное разбиение. Заведем для этого класс `DivisionInfo`, который будет содержать следующую информацию:

- **CubeVolume Volume** – количество узлов в сетке $\bar{\omega}_h$ по каждому измерению, а также диапазоны индексов из глобальной сетки $\bar{\omega}_h$ по каждому измерению.
- **CubeCoordinate Coord** – координаты процессора по каждому измерению внутри сетки из блоков.
- **CubeNumber CubeNum** – количество блоков по каждому измерению (эти данные одинаковые для всех процессоров).

Чтобы вычислить количество шагов K , необходимое для сходимости численного метода, воспользуемся условием устойчивости для трехмерной схемы «крест»:

$$a_t \tau \sqrt{\frac{1}{h_x^2} + \frac{1}{h_y^2} + \frac{1}{h_z^2}} \leq 1$$

Из этого следует:

$$a_t \tau \sqrt{\frac{1}{h_x^2} + \frac{1}{h_y^2} + \frac{1}{h_z^2}} = a_t \frac{T}{K} \sqrt{\frac{1}{h_x^2} + \frac{1}{h_y^2} + \frac{1}{h_z^2}} \leq 1 \Rightarrow a_t T \sqrt{\frac{1}{h_x^2} + \frac{1}{h_y^2} + \frac{1}{h_z^2}} \leq K.$$

Заменив неравенство на равенство, получим значение K .

После того как эти данные вычислены, можно вычислить значения в сетках u_0 и u_1 для каждого процессора и посчитать погрешность как максимум из модулей разности между значениями в узлах сетки и значениями функции u в точках (x_i, y_j, z_k, t_n) . Посчитанные погрешности будет собирать процессор с рангом 0, используя функцию `MPI_Reduce` с аргументом `MPI_MAX`.

Далее будем обозначать за u_2 сетку, значения в узлах которой нам нужно вычислить на текущем шаге, за u_1 и u_0 – сетки, хранящие значения в узлах на прошлом и позапрошлом слое по времени соответственно.

Каждый процессор хранит свои локальные части глобальных сеток u_1 и u_0 , поэтому

вычисление значений внутренних узлов сетки u_2 тривиально. Трудности возникают при подсчете значений в граничных узлах этой сетки. Для подсчета нужно организовать обмен точек с соседями по каждой координате. Рассмотрим следующие случаи:

1. Вычисление границ по координате X . Для этого каждый процессор берет сначала ту грань блока, которая соответствует плоскости YZ при меньшем значении координаты X (такие плоскости будем называть левыми стенками), и заполняет двумерный массив соответствующими граничными значениями сетки u_1 . Далее вычисляется ранг соседей слева и справа по координате X (для первого блока соседом слева будет последний блок). С помощью функции `MPI_Sendrecv` производится передача двумерного массива соседу слева и одновременное получение массива от соседа справа. Далее пересчитываются значения в правой стенке YZ . Аналогично, после соответствующих обменов, считаем левые стенки. Для граничных блоков в сетке значения в крайних стенках равны 0 (граничное условие первого рода).
2. Для вычисления стенок XZ по координате Y производятся аналогичные действия.
3. Для вычисления стенок XY по координате Z придется изменить алгоритм, поскольку по Z граничные условия периодические. Для этого первый блок заполняет левую стенку двумерного массива XY значениями $u_1[i, j, 1]$, в то время как остальные блоки используют $u_1[i, j, 0]$. Передаем эту стенку соседу слева, получаем от правого соседа его левую стенку. Далее производим вычисления в правых стенках XY всех блоков. Чтобы отправить свою правую стенку соседу слева последний блок будет использовать уже посчитанные значения $u_2[i, j, Z_{max}]$, в то время как остальные блоки отправляют $u_1[i, j, Z_{max}]$. Левая стенка первого блока полностью копирует правую стенку из массива, присланного последним блоком.

После поочередного выполнения этих действий считаем погрешность по описанному ранее алгоритму и высылаем процессу с рангом 0. На каждом шаге по времени будет выводиться максимальная погрешность. Всего шагов по времени сделаем 20, как сказано в описании задания. Время выполнения программы вычисляется как максимум времен каждого процессора по отдельности (процессор с рангом 0 собирает времена с помощью редукции по максимуму).

Задействовать OpenMP мы будем для циклов, их в коде большое количество. Для примера можно рассмотреть функцию `updateYXLeft()` из файла `task3.cpp`. Распараллеливаемые циклы, выполняющиеся друг за другом, можно поместить в единую секцию `#pragma omp parallel`, а внутри нее размещать прагмы `#pragma omp for`, что поможет снизить накладные расходы на запуск нитей. Также для циклов можно добавлять клаузу `nowait`, что позволит нитям не дожидаться завершения цикла и перейти к следующему циклу, поскольку между циклами неявно появляется барьер.

| N_p | N^3 | T | S | δ |
|-------|---------|-----------|----------|--------------|
| 1 | 128^3 | 10.53250 | 1.00000 | 0.0000006051 |
| 4 | 128^3 | 3.16360 | 3.32928 | 0.0000006017 |
| 8 | 128^3 | 1.44000 | 7.31424 | 0.0000006017 |
| 16 | 128^3 | 1.02310 | 10.29469 | 0.0000006017 |
| 32 | 128^3 | 0.55020 | 19.14304 | 0.0000006017 |
| 1 | 256^3 | 108.45930 | 1.00000 | 0.0000000379 |
| 4 | 256^3 | 40.80650 | 2.65789 | 0.0000000378 |
| 8 | 256^3 | 19.11920 | 5.67279 | 0.0000000378 |
| 16 | 256^3 | 17.01840 | 6.37306 | 0.0000000378 |
| 32 | 256^3 | 4.53360 | 23.92344 | 0.0000000378 |
| 1 | 512^3 | 867.30670 | 1.00000 | 0.0000000024 |
| 4 | 512^3 | 370.99470 | 2.33779 | 0.0000000024 |
| 8 | 512^3 | 187.85770 | 4.61683 | 0.0000000024 |
| 16 | 512^3 | 97.66530 | 8.88040 | 0.0000000024 |
| 32 | 512^3 | 48.87710 | 17.74464 | 0.0000000024 |

| N_p | Thr | N^3 | T | S | δ |
|-------|-----|---------|-----------|---------|--------------|
| 1 | 4 | 128^3 | 7.90810 | 1.00000 | 0.0000006051 |
| 2 | 4 | 128^3 | 4.13980 | 1.91026 | 0.0000006034 |
| 4 | 4 | 128^3 | 2.28040 | 3.46786 | 0.0000006017 |
| 8 | 4 | 128^3 | 1.16510 | 6.78749 | 0.0000006017 |
| 1 | 4 | 256^3 | 86.98200 | 1.00000 | 0.0000000379 |
| 2 | 4 | 256^3 | 49.28200 | 1.76499 | 0.0000000378 |
| 4 | 4 | 256^3 | 23.81850 | 3.65187 | 0.0000000378 |
| 8 | 4 | 256^3 | 12.01360 | 7.24029 | 0.0000000378 |
| 1 | 4 | 512^3 | 737.95810 | 1.00000 | 0.0000000024 |
| 2 | 4 | 512^3 | 402.86250 | 1.83179 | 0.0000000024 |
| 4 | 4 | 512^3 | 231.57870 | 3.18664 | 0.0000000024 |
| 8 | 4 | 512^3 | 113.69860 | 6.49048 | 0.0000000024 |

Таблица 1: Результаты вычислений для $L_x = L_y = L_z = 1$ с использованием MPI и MPI+OpenMP

4 Исследование масштабируемости

Будем запускать получившуюся программу на системе Polus. Сначала запустим версию, включающую только MPI. Для значений $L_x = L_y = L_z = 1$ также запустим гибридную версию MPI+OpenMP на 4 потоках (для $L_x = L_y = L_z = \pi$ публиковать таблицу с гибридной версией не будем, поскольку принципиально запуск этой версии ничем не отличается). Результаты представлены в таблицах 1 и 2.

Версия программы, использующая только MPI, неплохо масштабируется. Идеального ускорения, равного количеству процессов, добиться не получается, поскольку присутствуют накладные расходы на обмен стенками между блоками и сбор информации о погрешностях на каждом процессе. С использованием OpenMP получилось добиться еще большего ускорения, что можно заметить, если сравнить время работы программы на одном и том же размере и количестве процессоров в левой и правой подтаблице таблицы 1.

Для наглядности также продемонстрируем графики аналитического (рисунок 1) и посчитанного (рисунок 2) решений для маленькой сетки, где $N = 8$, на 20-м шаге. На рисунке 3 размещен график погрешности.

На дальнейших рисунках размещены графики зависимости времени и ускорения от числа процессов.

| N_p | N^3 | T | S | δ |
|-------|---------|-----------|----------|--------------|
| 1 | 128^3 | 10.18750 | 1.00000 | 0.0000053072 |
| 4 | 128^3 | 3.10560 | 3.28036 | 0.0000052140 |
| 8 | 128^3 | 1.42980 | 7.12512 | 0.0000052140 |
| 16 | 128^3 | 1.05500 | 9.65640 | 0.0000052140 |
| 32 | 128^3 | 0.47200 | 21.58369 | 0.0000052140 |
| 1 | 256^3 | 107.73360 | 1.00000 | 0.0000003351 |
| 4 | 256^3 | 40.62910 | 2.65164 | 0.0000003331 |
| 8 | 256^3 | 19.22530 | 5.60374 | 0.0000003331 |
| 16 | 256^3 | 10.83830 | 9.94008 | 0.0000003331 |
| 32 | 256^3 | 4.30430 | 25.02930 | 0.0000003331 |
| 1 | 512^3 | 868.56020 | 1.00000 | 0.0000000210 |
| 4 | 512^3 | 371.60010 | 2.33735 | 0.0000000209 |
| 8 | 512^3 | 187.75920 | 4.62593 | 0.0000000209 |
| 16 | 512^3 | 97.05730 | 8.94894 | 0.0000000209 |
| 32 | 512^3 | 51.16600 | 16.97534 | 0.0000000209 |

Таблица 2: Результаты вычислений для $L_x = L_y = L_z = \pi$

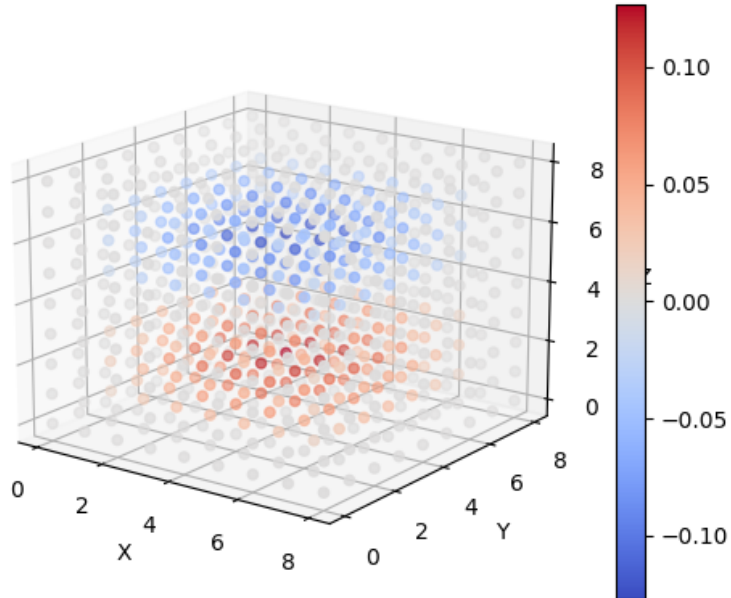


Рис. 1: Значения в узлах сетки для аналитического решения

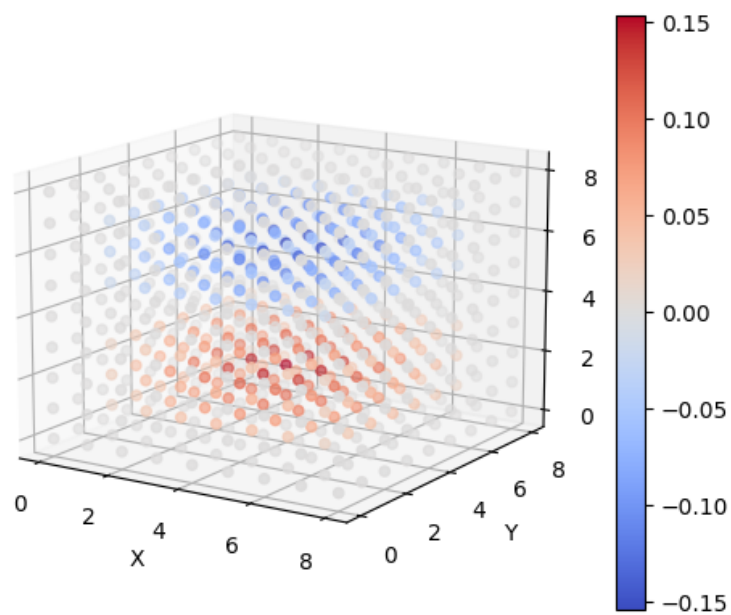


Рис. 2: Значения в узлах сетки для посчитанного решения

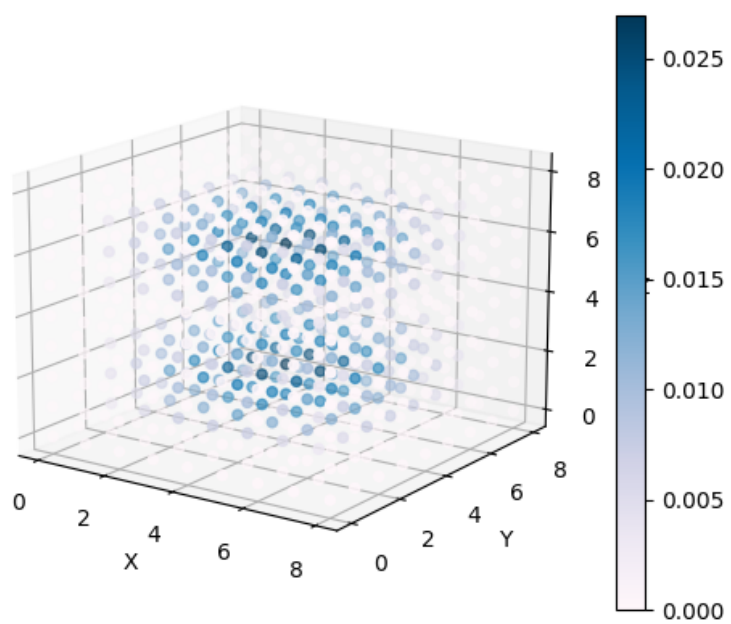


Рис. 3: Погрешность в узлах сетки (абсолютные значения)

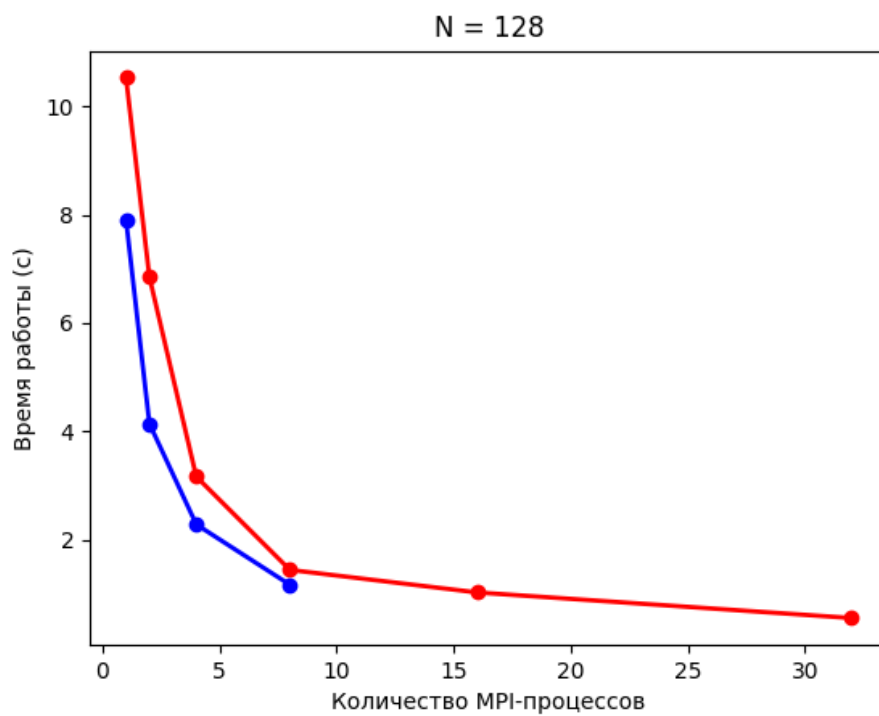


Рис. 4: График зависимости времени работы программы от количества процессов при $N = 128$ (красный цвет – только MPI, синий – MPI + 4 потока OpenMP)

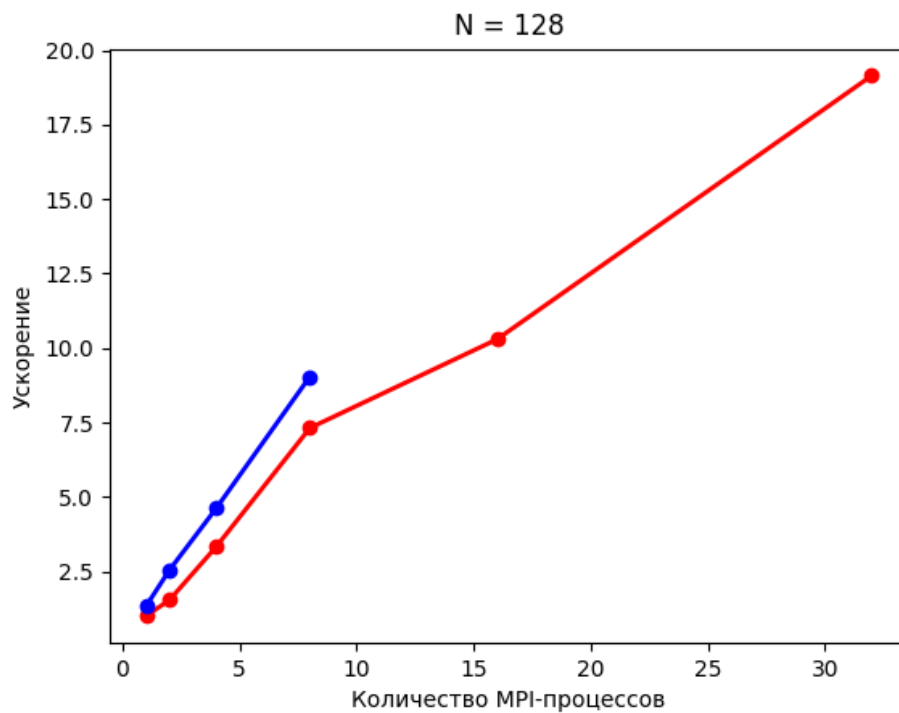


Рис. 5: График зависимости ускорения программы от количества процессов при $N = 128$

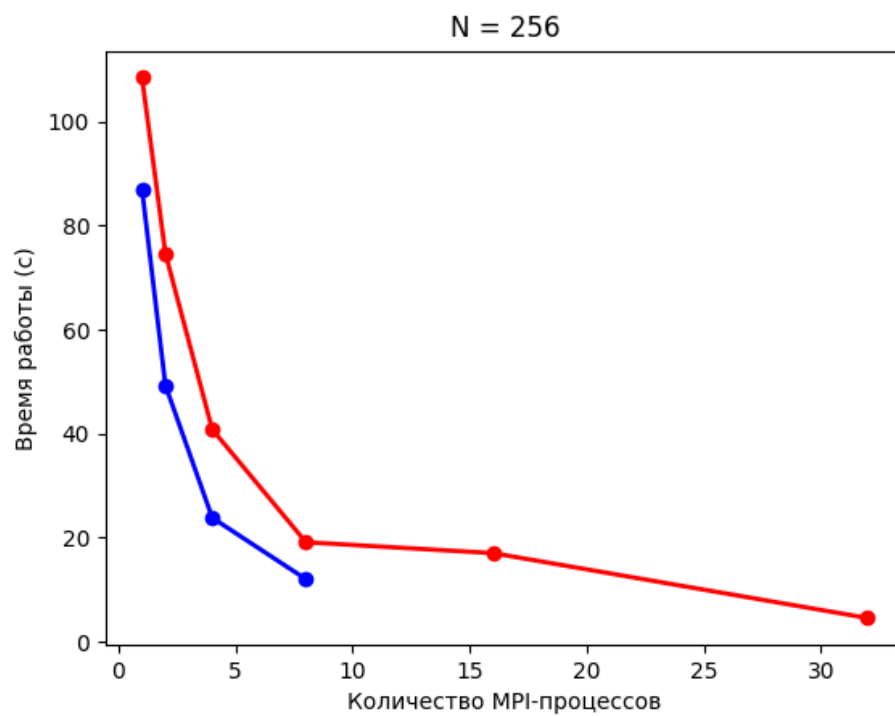


Рис. 6: График зависимости времени работы программы от количества процессов при $N = 256$ (красный цвет — только MPI, синий — MPI + 4 потока OpenMP)

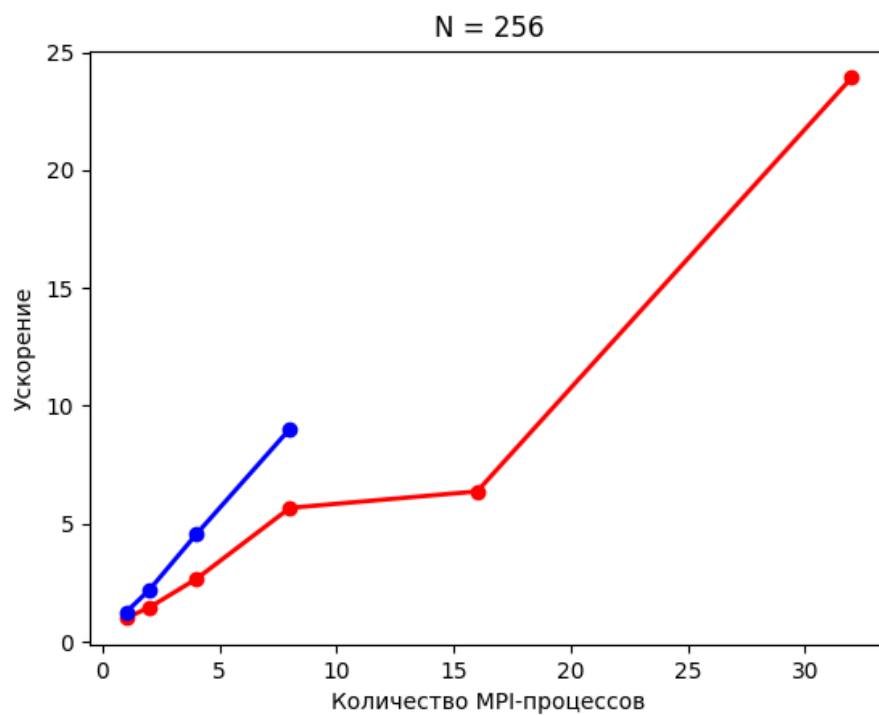


Рис. 7: График зависимости ускорения программы от количества процессов при $N = 256$

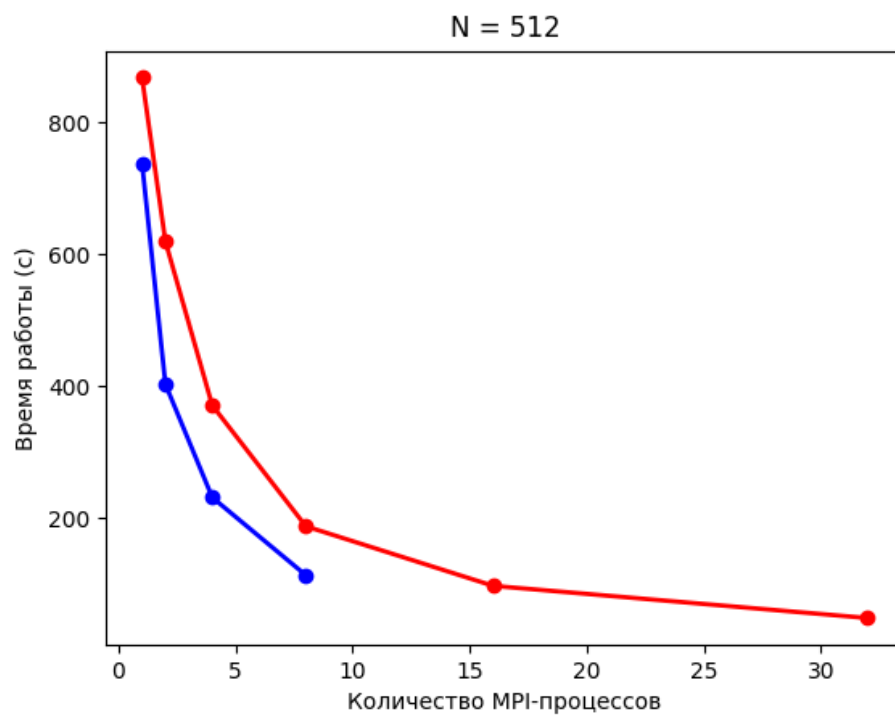


Рис. 8: График зависимости времени работы программы от количества процессов при $N = 512$ (красный цвет – только MPI, синий – MPI + 4 потока OpenMP)

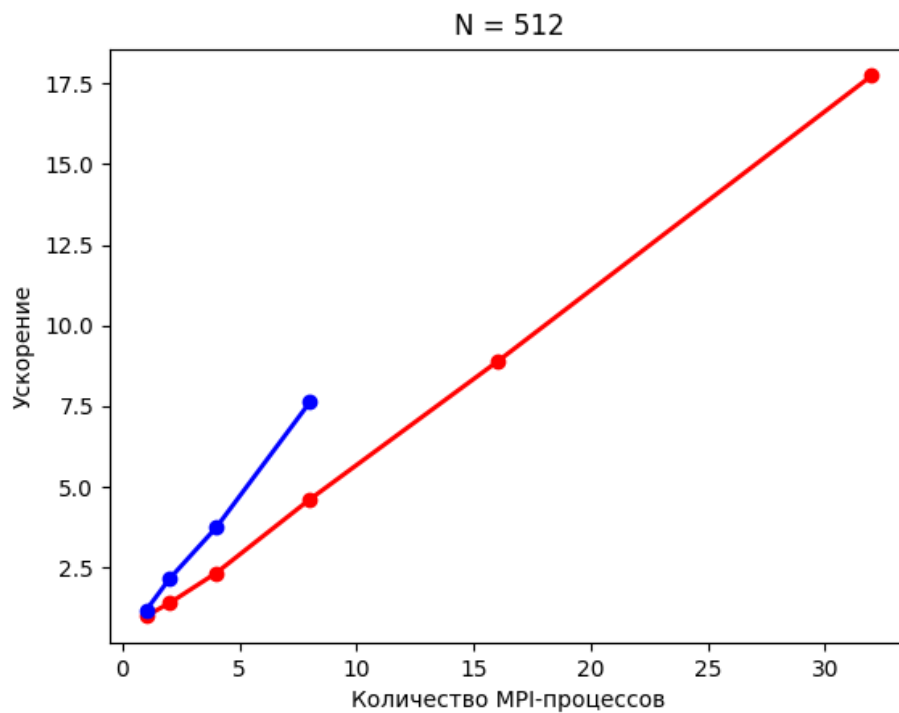


Рис. 9: График зависимости ускорения программы от количества процессов при $N = 512$