



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

СУПЕРКОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ И ТЕХНОЛОГИИ

Отчет по заданию №4 «Задача для трёхмерного гиперболического уравнения в прямоугольном параллелепипеде»

ВАРИАНТ №3

*Богатенкова Анастасия Олеговна
628 группа*

28 ноября 2022 г.

Содержание

1	Математическая постановка задачи	2
2	Численный метод решения задачи	3
3	Описание программной реализации	5
4	Графики решений и погрешности	6
5	Результаты расчётов	7
5.1	Результаты запусков MPI	8
5.2	Результаты запусков MPI+OpenMP	12
5.3	Результаты запусков MPI+CUDA	20
5.4	Сравнение результатов	24
5.5	Профилирование MPI+CUDA	25
6	Выводы	26

1 Математическая постановка задачи

В трёхмерной 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

$$\frac{\partial^2 u}{\partial t^2} = \Delta u \quad (1)$$

с начальными условиями

$$u|_{t=0} = \varphi(x, y, z) \quad (2)$$

$$\frac{\partial u}{\partial t}|_{t=0} = 0 \quad (3)$$

при условии, что на границах области заданы однородные граничные условия первого рода

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

либо периодические граничные условия

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

В полученном варианте задания (№3) граничные условия выглядят следующим образом:

$$\begin{aligned} u(0, y, z, t) &= 0, & u(L_x, y, z, t) &= 0 \\ u(x, 0, z, t) &= u(x, L_y, z, t), & u_y(x, 0, z, t) &= u_y(x, L_y, z, t) \\ u(x, y, 0, t) &= 0, & u(x, y, L_z, t) &= 0 \end{aligned} \quad (4)$$

Аналитическое уравнение функции u :

$$\begin{aligned} u(x, y, z, t) &= \sin\left(\frac{\pi}{L_x}x\right) \cdot \sin\left(\frac{2\pi}{L_y}y\right) \cdot \sin\left(\frac{3\pi}{L_z}z\right) \cdot \cos(a_t t) \\ a_t &= \pi \sqrt{\frac{1}{L_x^2} + \frac{4}{L_y^2} + \frac{9}{L_z^2}} \end{aligned}$$

2 Численный метод решения задачи

Введём на Ω сетку: $\omega_{h\tau} = \bar{\omega}_h \times \omega_\tau$, где

$$T = T_0$$

$$L_x = L_{x_0}, \quad L_y = L_{y_0}, \quad L_z = L_{z_0}$$

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

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

Через ω_h обозначим множество внутренних, а через γ_h – множество граничных узлов сетки $\bar{\omega}_h$.

Для аппроксимации исходного уравнения (1) с начальными условиями (2)–(3) и граничными условиями (4) воспользуемся следующей системой уравнений:

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

Здесь Δ_h – семиточечный разностный аналог оператора Лапласа:

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

Приведённая выше разностная схема является явной – значения u_{ijk}^{n+1} на $(n+1)$ -м шаге можно явным образом выразить через значения на предыдущих слоях.

Для начала счёта (т.е. нахождения u_{ijk}^2) должны быть заданы значения $u_{ijk}^0, u_{ijk}^1, (x_i, y_j, z_k) \in \omega_h$. Из условия (2) имеем:

$$u_{ijk}^0 = \varphi(x_i, y_j, z_k), \quad (x_i, y_j, z_k) \in \omega_h$$

Простейшая замена условия (3) уравнением $\frac{u_{ijk}^1 - u_{ijk}^0}{\tau} = 0$ имеет лишь первый порядок аппроксимации по τ . Аппроксимацию второго порядка по τ и h дает разностное уравнение:

$$\frac{u_{ijk}^1 - u_{ijk}^0}{\tau} = \frac{\tau}{2} \Delta_h \varphi(x_i, y_j, z_k), \quad (x_i, y_j, z_k) \in \omega_h$$

$$u_{ijk}^1 = u_{ijk}^0 + \frac{\tau^2}{2} \Delta_h \varphi(x_i, y_j, z_k)$$

Разностная аппроксимация для периодических граничных условий выглядит следующим образом:

$$\begin{aligned} u_{0jk}^{n+1} &= u_{Njk}^{n+1}, \\ u_{i0k}^{n+1} &= u_{iNk}^{n+1}, \\ u_{ij0}^{n+1} &= u_{ijN}^{n+1}, \end{aligned}$$

$$\begin{aligned} u_{1jk}^{n+1} &= u_{N+1jk}^{n+1} \\ u_{i1k}^{n+1} &= u_{iN+1k}^{n+1} \\ u_{ij1}^{n+1} &= u_{ijN+1}^{n+1} \end{aligned}$$

$$i, j, k = 0, \dots, N.$$

Для вычисления $u^0, u^1 \in \gamma_h$ допускается использование аналитического значения u , которое задается в программе еще для вычисления погрешности решения задачи.

Таким образом, алгоритм численного решения задачи выглядит следующим образом:

1. Вычислить значения u^0, u^1 на границе:

$$u^{0,1}|_{\gamma_h} \leftarrow 0, (x_i, y_j, z_k) \in \gamma_h \text{ для условий 1-го рода,}$$

$$u^0|_{\gamma_h} \leftarrow u(x_i, y_j, z_k, 0), (x_i, y_j, z_k) \in \gamma_h,$$

$$u^1|_{\gamma_h} \leftarrow u(x_i, y_j, z_k, \tau), (x_i, y_j, z_k) \in \gamma_h \text{ для периодических условий.}$$

2. Вычислить значения u^0 внутри области:

$$u_{ijk}^0 \leftarrow \varphi(x_i, y_j, z_k), (x_i, y_j, z_k) \in \omega_h$$

3. Вычислить значения u^1 внутри области:

$$u_{ijk}^1 \leftarrow u_{ijk}^0 + \frac{\tau^2}{2} \Delta_h \varphi(x_i, y_j, z_k), (x_i, y_j, z_k) \in \omega_h$$

4. Повторять для $K - 1$ шагов:

вычисляем значения u^{n+1} на основе двух предыдущих шагов внутри области:

$$u_{ijk}^{n+1} \leftarrow 2u_{ijk}^n - u_{ijk}^{n-1} + \tau^2 \cdot \Delta_h u^n, (x_i, y_j, z_k) \in \omega_h$$

вычисляем граничные значения u^{n+1} :

$$u_{ijk}^{n+1} \leftarrow 0, (x_i, y_j, z_k) \in \gamma_h \text{ для условий 1-го рода,}$$

$$u_{ijk}^{n+1} \leftarrow u(x_i, y_j, z_k, (n+1) \cdot \tau), (x_i, y_j, z_k) \in \gamma_h \text{ для периодических условий.}$$

3 Описание программной реализации

Реализовано три программы: последовательная, параллельная с использованием MPI+OpenMP и параллельная с использованием MPI+CUDA. Программа принимает на вход число точек сетки по одной из осей N , длину отрезка решения по одной из осей L ($L_x = L_y = L_z = L$), а также имя файла для вывода результатов *out_file*. Программа выводит в файл число точек N , число процессов MPI, погрешность решения на последнем временном слое и время работы программы.

Параллельная версия программы выполнена следующим образом:

1. Сетка разбивается на блоки, каждый из которых достаётся процессу для вычисления функции;
2. Каждый процесс вычисляет своих соседей (номера процессов-соседей и прямоугольники-границы блоков);
3. Каждый процесс заполняет свой блок начальными значениями u_0 , u_1 с использованием аналитической функции и формул предыдущей главы.
4. В цикле происходит вычисление функции на очередном временном слое, при этом происходят пересылки прямоугольников-границ блоков для вычисления оператора Лапласа.
5. В конце вычислений с помощью *MPI_Reduce* вычисляется погрешность вычисления функции на последнем временном слое как максимум из локальных ошибок каждого из процессов.

В параллельной версии MPI+OpenMP все циклы распараллеливались с помощью директивы *#pragma omp parallel for*. В параллельной версии MPI+CUDA для работы с памятью и реализации редукции использовалась библиотека *thrust*.

Проверка корректности выполнения параллельных версий программ проверялась сопоставлением результатов работы программ с результатами работы последовательной версии. Ошибки вычисления функции на каждом временном слое выводились в файл мастер-процессом и сравнивались с ошибками последовательной версии. Полученные одинаковые значения позволяют предположить правильность работы программ.

4 Графики решений и погрешности

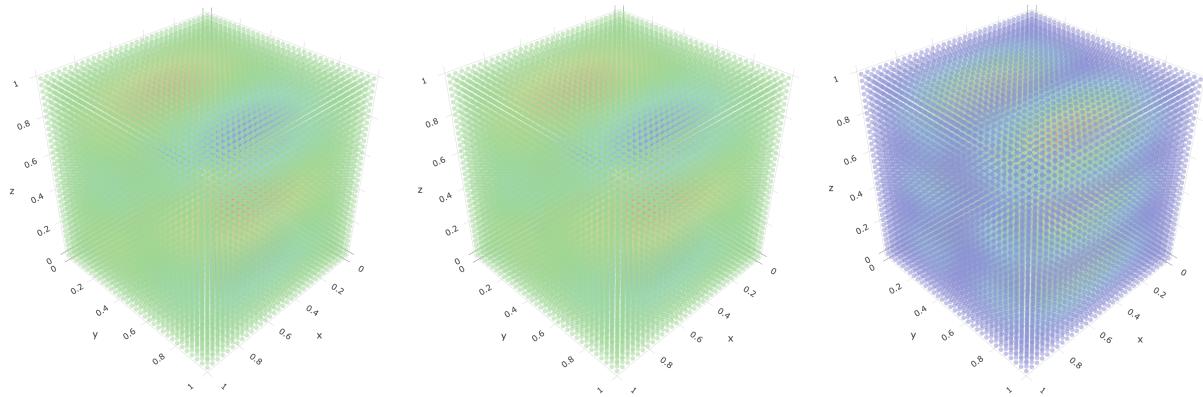


Рис. 1: Графики аналитической функции, вычисленной функции и погрешности вычисления для $L_x = L_y = L_z = 1$

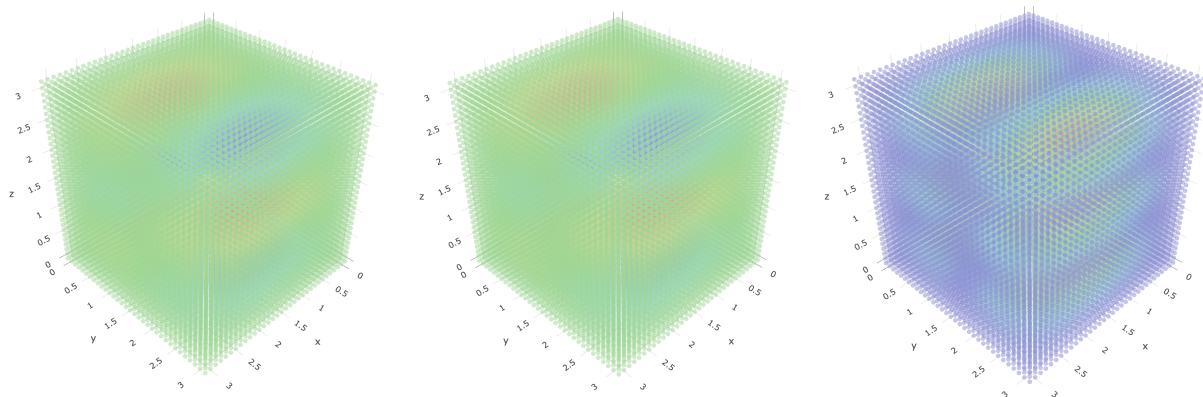


Рис. 2: Графики аналитической функции, вычисленной функции и погрешности вычисления для $L_x = L_y = L_z = \pi$

5 Результаты расчётов

В таблицах и на графиках ниже представлены результаты расчётов на суперкомпьютере Polus для $L_x = L_y = L_z = L, T = 1, K = 1000$. Для каждого набора p, N, L время работы программы усреднялось по 5 запускам. В таблицах представлены усреднённые значения времени и ускорения.

Подсчитаны время выполнения разных реализаций программы с разными аргументами. Ускорение подсчитывалось разными способами: работа программы сравнивалась с последовательной программой без MPI, а также с MPI-программой, запущенной на 1 MPI-процессе. Результаты сравнения с последовательной программой хуже результатов сравнения с MPI-программой, что вполне ожидаемо в силу усложнения кода и накладных расходов, связанных с передачей данных.

Получены следующие результаты:

- В таблицах 1, 2, 3, 4 и на рисунках 3, 4, 5, 6 представлены результаты запусков MPI программы без распараллеливания с помощью OpenMP.
- В таблицах 5, 6, 7, 8 и на рисунках 7, 8, 9, 10 представлены результаты запусков гибридной программы MPI+OpenMP, запуски проводились с 8 нитями на процесс.
- В таблицах 9, 10, 11, 12 и на рисунках 11, 12, 13, 14 представлены результаты запусков MPI+OpenMP с 128 нитями на процесс. Они не всегда лучше результатов с 8 нитями, так как на вычислительном комплексе на каждом процессоре 8 ядер, и 8 нитей на процесс является оптимальным значением для запуска.
- В таблицах 13, 14, 15, 16 и на рисунках 15, 16, 17, 18 представлены результаты запусков MPI+CUDA с 128 нитями на процесс.

Кроме того, в таблицах 17, 18 для наглядности представлены сравнительные результаты времени и ускорения различных версий программы относительно последовательной программы без MPI. Судя по результатам, MPI+CUDA позволяет выполнять вычисления более эффективно за более короткое время.

Для реализации MPI+CUDA с помощью *nvprof* было выполнено профилирование и подсчитано время, затраченное на работу различных функций программы. В таблицах 19, 20 представлены результаты профилирования.

5.1 Результаты запусков MPI

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.890	1.000	$9.3 \cdot 10^{-6}$
2	128^3	1.007	1.876	$9.3 \cdot 10^{-6}$
4	128^3	0.556	3.397	$9.3 \cdot 10^{-6}$
6	128^3	0.491	3.851	$9.3 \cdot 10^{-6}$
1	256^3	14.314	1.000	$2.1 \cdot 10^{-6}$
2	256^3	7.441	1.924	$2.1 \cdot 10^{-6}$
4	256^3	3.808	3.759	$2.1 \cdot 10^{-6}$
6	256^3	2.826	5.065	$2.1 \cdot 10^{-6}$
1	512^3	110.868	1.000	$2.9 \cdot 10^{-7}$
2	512^3	57.147	1.940	$2.9 \cdot 10^{-7}$
4	512^3	29.622	3.743	$2.9 \cdot 10^{-7}$
6	512^3	20.645	5.370	$2.9 \cdot 10^{-7}$

Таблица 1: Результаты для Polus при $L = 1$ (сравнение с 1-процессной MPI программой)

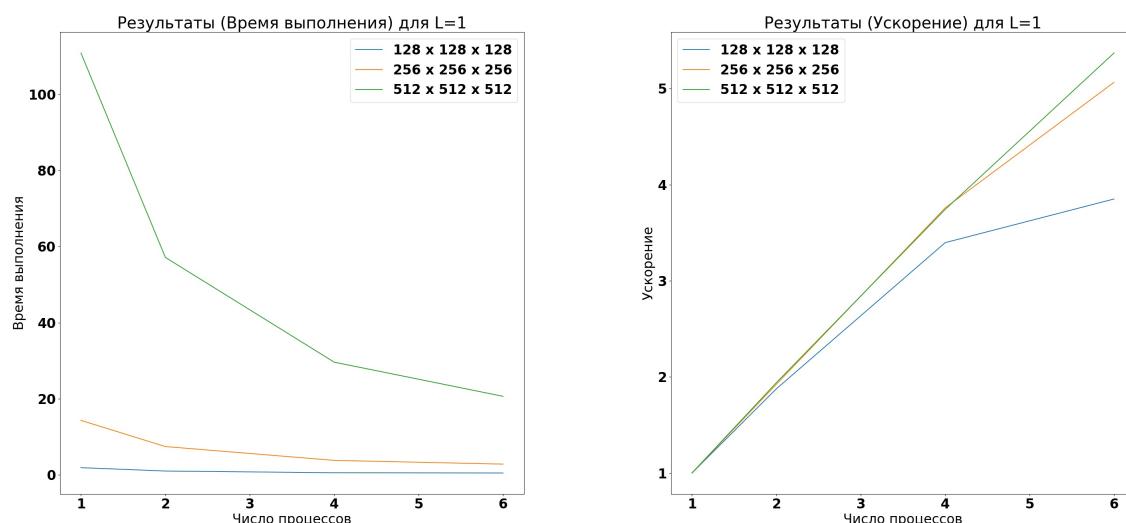


Рис. 3: Графики времени работы и ускорения для Polus при $L = 1$ (сравнение с 1-процессной MPI программой)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.895	1.000	$9.8 \cdot 10^{-7}$
2	128^3	1.007	1.882	$9.8 \cdot 10^{-7}$
4	128^3	0.543	3.492	$9.8 \cdot 10^{-7}$
6	128^3	0.470	4.030	$9.8 \cdot 10^{-7}$
1	256^3	14.354	1.000	$2.4 \cdot 10^{-7}$
2	256^3	7.468	1.922	$2.4 \cdot 10^{-7}$
4	256^3	3.876	3.703	$2.4 \cdot 10^{-7}$
6	256^3	2.859	5.021	$2.4 \cdot 10^{-7}$
1	512^3	110.897	1.000	$5.8 \cdot 10^{-8}$
2	512^3	57.079	1.943	$5.8 \cdot 10^{-8}$
4	512^3	29.485	3.761	$5.8 \cdot 10^{-8}$
6	512^3	20.612	5.380	$5.8 \cdot 10^{-8}$

Таблица 2: Результаты для Polus при $L = \pi$ (сравнение с 1-процессной MPI программой)

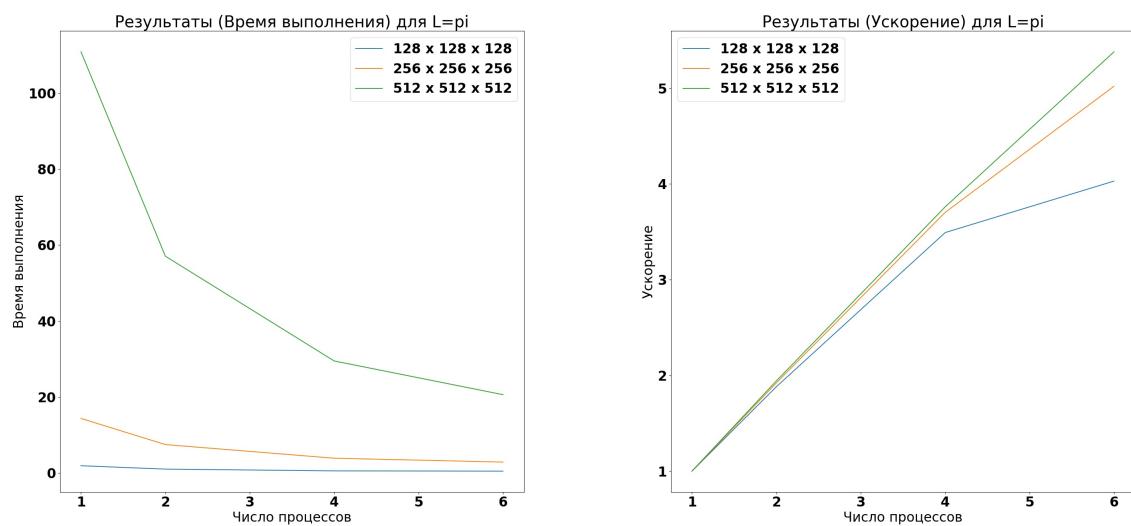


Рис. 4: Графики времени работы и ускорения для Polus при $L = \pi$ (сравнение с 1-процессной MPI программой)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.213	1.000	$9.3 \cdot 10^{-6}$
2	128^3	1.007	1.204	$9.3 \cdot 10^{-6}$
4	128^3	0.556	2.181	$9.3 \cdot 10^{-6}$
6	128^3	0.491	2.472	$9.3 \cdot 10^{-6}$
1	256^3	8.761	1.000	$2.1 \cdot 10^{-6}$
2	256^3	7.441	1.177	$2.1 \cdot 10^{-6}$
4	256^3	3.808	2.300	$2.1 \cdot 10^{-6}$
6	256^3	2.826	3.100	$2.1 \cdot 10^{-6}$
1	512^3	66.157	1.000	$2.9 \cdot 10^{-7}$
2	512^3	57.147	1.158	$2.9 \cdot 10^{-7}$
4	512^3	29.622	2.233	$2.9 \cdot 10^{-7}$
6	512^3	20.645	3.205	$2.9 \cdot 10^{-7}$

Таблица 3: Результаты для Polus при $L = 1$ (сравнение с последовательной программой без MPI)

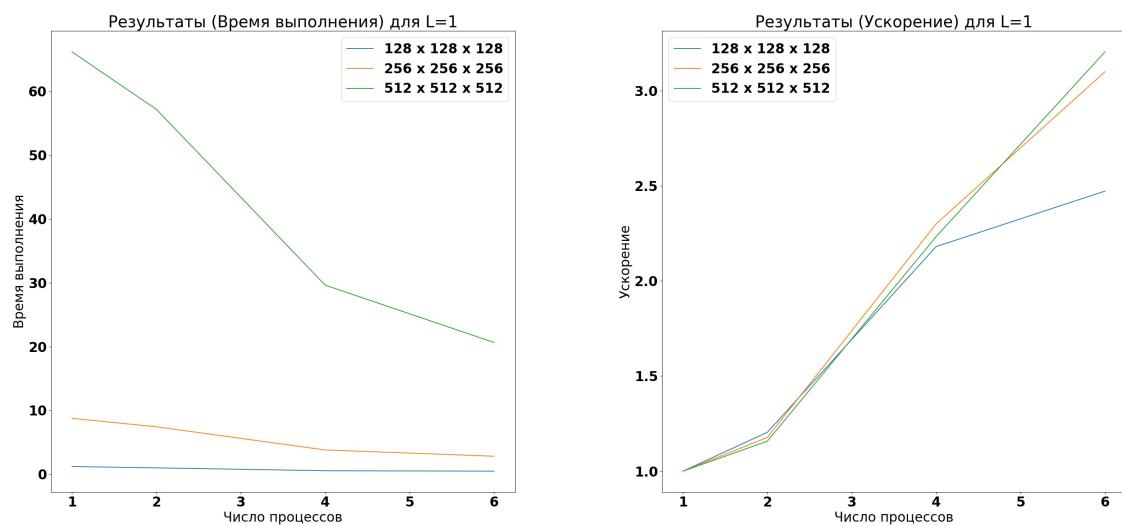


Рис. 5: Графики времени работы и ускорения для Polus при $L = 1$ (сравнение с последовательной программой без MPI)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.214	1.000	$9.8 \cdot 10^{-7}$
2	128^3	1.007	1.205	$9.8 \cdot 10^{-7}$
4	128^3	0.543	2.236	$9.8 \cdot 10^{-7}$
6	128^3	0.470	2.581	$9.8 \cdot 10^{-7}$
1	256^3	8.762	1.000	$2.4 \cdot 10^{-7}$
2	256^3	7.468	1.173	$2.4 \cdot 10^{-7}$
4	256^3	3.876	2.260	$2.4 \cdot 10^{-7}$
6	256^3	2.859	3.065	$2.4 \cdot 10^{-7}$
1	512^3	66.396	1.000	$5.8 \cdot 10^{-8}$
2	512^3	57.079	1.163	$5.8 \cdot 10^{-8}$
4	512^3	29.485	2.252	$5.8 \cdot 10^{-8}$
6	512^3	20.612	3.221	$5.8 \cdot 10^{-8}$

Таблица 4: Результаты для Polus при $L = \pi$ (сравнение с последовательной программой без MPI)

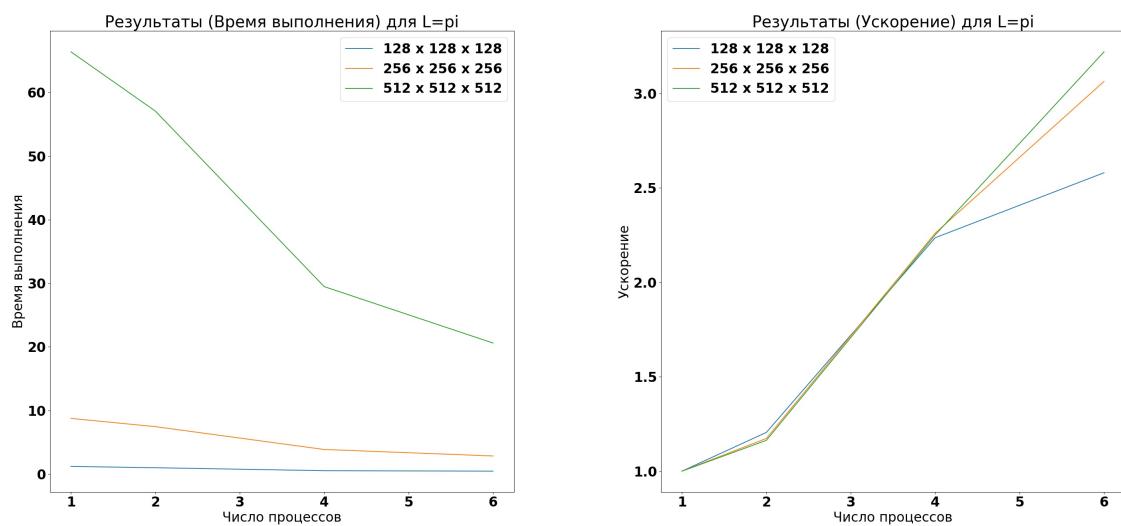


Рис. 6: Графики времени работы и ускорения для Polus при $L = \pi$ (сравнение с последовательной программой без MPI)

5.2 Результаты запусков MPI+OpenMP

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	0.609	1.000	$9.3 \cdot 10^{-6}$
2	128^3	0.361	1.689	$9.3 \cdot 10^{-6}$
4	128^3	0.185	3.297	$9.3 \cdot 10^{-6}$
6	128^3	0.172	3.543	$9.3 \cdot 10^{-6}$
1	256^3	4.337	1.000	$2.1 \cdot 10^{-6}$
2	256^3	4.159	1.043	$2.1 \cdot 10^{-6}$
4	256^3	2.179	1.990	$2.1 \cdot 10^{-6}$
6	256^3	1.592	2.724	$2.1 \cdot 10^{-6}$
1	512^3	32.728	1.000	$2.9 \cdot 10^{-7}$
2	512^3	29.229	1.120	$2.9 \cdot 10^{-7}$
4	512^3	12.273	2.667	$2.9 \cdot 10^{-7}$
6	512^3	11.529	2.839	$2.9 \cdot 10^{-7}$

Таблица 5: Результаты для Polus MPI+OpenMP с 8 нитями при $L = 1$ (сравнение с 1-процессной MPI программой)

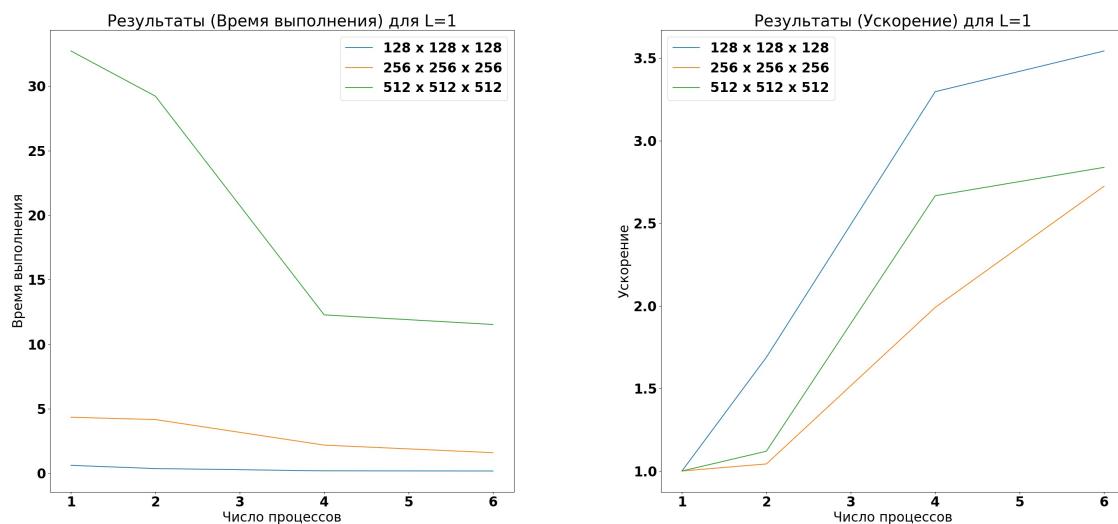


Рис. 7: Графики времени работы и ускорения для Polus MPI+OpenMP с 8 нитями при $L = 1$ (сравнение с 1-процессной MPI программой)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	0.617	1.000	$9.8 \cdot 10^{-7}$
2	128^3	0.388	1.591	$9.8 \cdot 10^{-7}$
4	128^3	0.206	2.998	$9.8 \cdot 10^{-7}$
6	128^3	0.235	2.630	$9.8 \cdot 10^{-7}$
1	256^3	4.344	1.000	$2.4 \cdot 10^{-7}$
2	256^3	4.069	1.067	$2.4 \cdot 10^{-7}$
4	256^3	2.091	2.078	$2.4 \cdot 10^{-7}$
6	256^3	1.475	2.945	$2.4 \cdot 10^{-7}$
1	512^3	32.699	1.000	$5.8 \cdot 10^{-8}$
2	512^3	30.240	1.081	$5.8 \cdot 10^{-8}$
4	512^3	11.269	2.902	$5.8 \cdot 10^{-8}$
6	512^3	11.657	2.805	$5.8 \cdot 10^{-8}$

Таблица 6: Результаты для Polus MPI+OpenMP с 8 нитями при $L = \pi$ (сравнение с 1-процессной MPI программой)

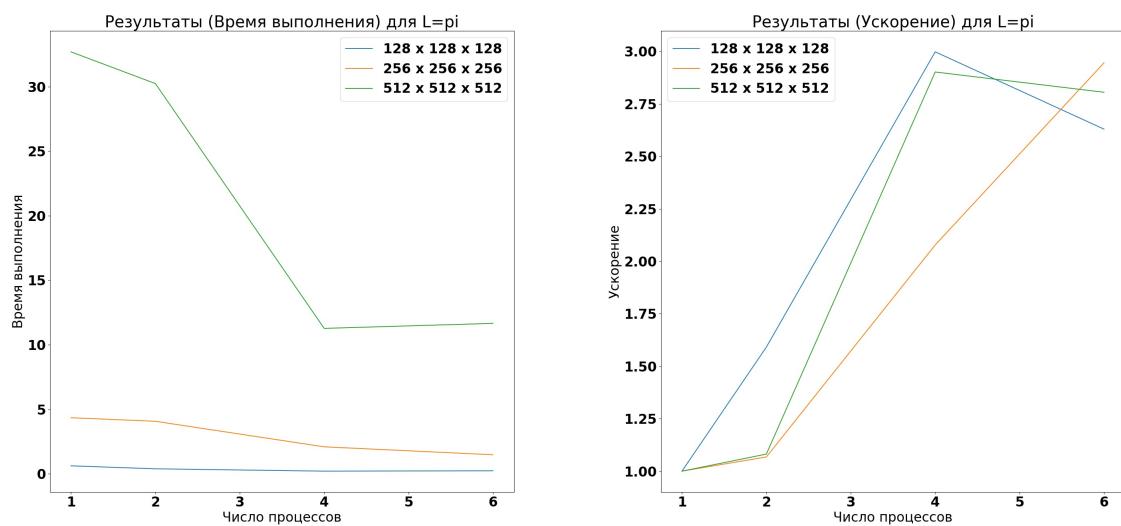


Рис. 8: Графики времени работы и ускорения для Polus MPI+OpenMP с 8 нитями при $L = \pi$ (сравнение с 1-процессной MPI программой)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.213	1.000	$9.3 \cdot 10^{-6}$
2	128^3	0.361	3.363	$9.3 \cdot 10^{-6}$
4	128^3	0.185	6.566	$9.3 \cdot 10^{-6}$
6	128^3	0.172	7.057	$9.3 \cdot 10^{-6}$
1	256^3	8.761	1.000	$2.1 \cdot 10^{-6}$
2	256^3	4.159	2.106	$2.1 \cdot 10^{-6}$
4	256^3	2.179	4.021	$2.1 \cdot 10^{-6}$
6	256^3	1.592	5.503	$2.1 \cdot 10^{-6}$
1	512^3	66.157	1.000	$2.9 \cdot 10^{-7}$
2	512^3	29.229	2.263	$2.9 \cdot 10^{-7}$
4	512^3	12.273	5.390	$2.9 \cdot 10^{-7}$
6	512^3	11.529	5.738	$2.9 \cdot 10^{-7}$

Таблица 7: Результаты для Polus MPI+OpenMP с 8 нитями при $L = 1$ (сравнение с последовательной программой без MPI)

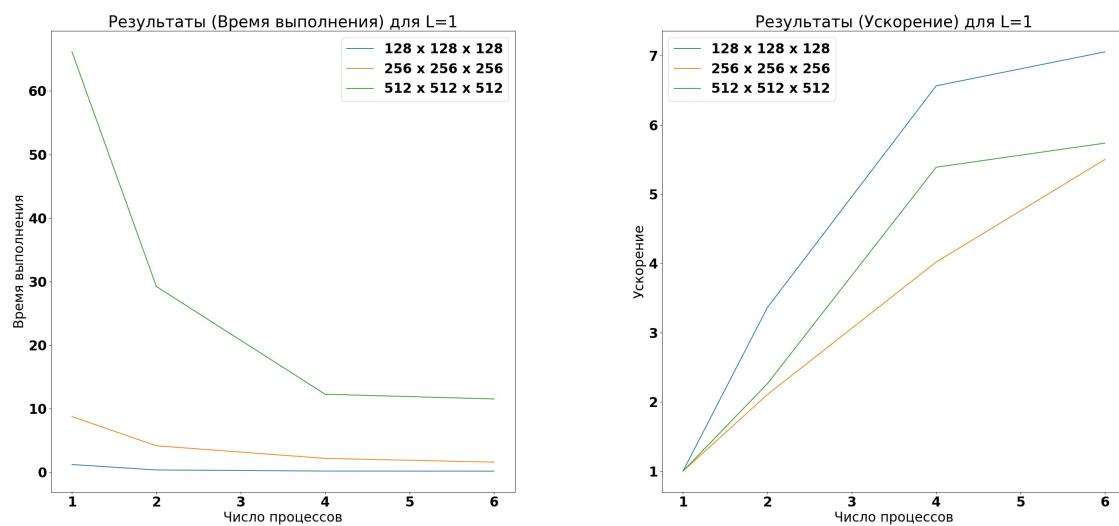


Рис. 9: Графики времени работы и ускорения для Polus MPI+OpenMP с 8 нитями при $L = 1$ (сравнение с последовательной программой без MPI)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.214	1.000	$9.8 \cdot 10^{-7}$
2	128^3	0.388	3.130	$9.8 \cdot 10^{-7}$
4	128^3	0.206	5.897	$9.8 \cdot 10^{-7}$
6	128^3	0.235	5.173	$9.8 \cdot 10^{-7}$
1	256^3	8.762	1.000	$2.4 \cdot 10^{-7}$
2	256^3	4.069	2.153	$2.4 \cdot 10^{-7}$
4	256^3	2.091	4.191	$2.4 \cdot 10^{-7}$
6	256^3	1.475	5.941	$2.4 \cdot 10^{-7}$
1	512^3	66.396	1.000	$5.8 \cdot 10^{-8}$
2	512^3	30.240	2.196	$5.8 \cdot 10^{-8}$
4	512^3	11.269	5.892	$5.8 \cdot 10^{-8}$
6	512^3	11.657	5.696	$5.8 \cdot 10^{-8}$

Таблица 8: Результаты для Polus MPI+OpenMP с 8 нитями при $L = \pi$ (сравнение с последовательной программой без MPI)

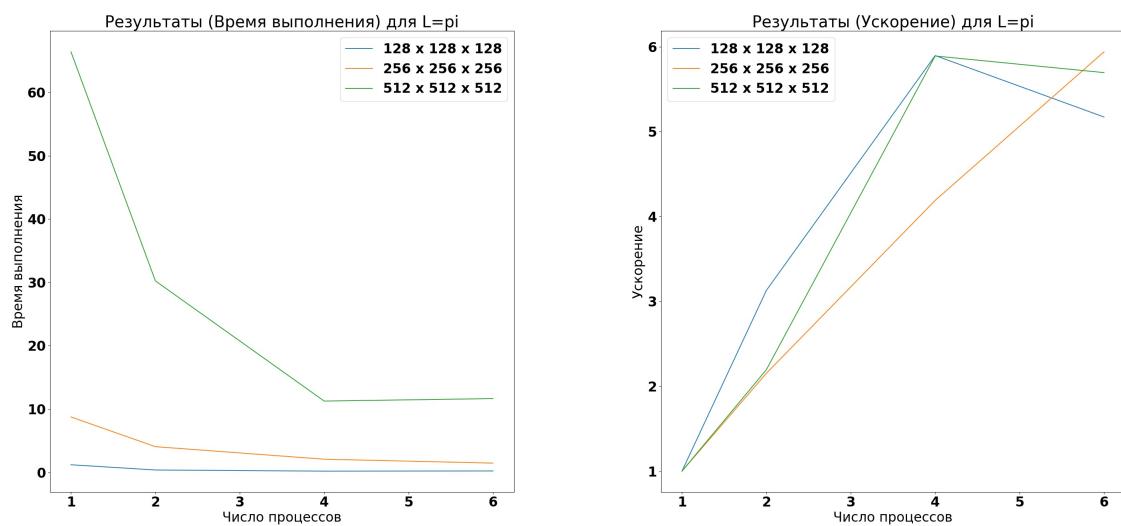


Рис. 10: Графики времени работы и ускорения для Polus MPI+OpenMP с 8 нитями при $L = \pi$ (сравнение с последовательной программой без MPI)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	0.947	1.000	$9.3 \cdot 10^{-6}$
2	128^3	0.628	1.508	$9.3 \cdot 10^{-6}$
4	128^3	0.543	1.745	$9.3 \cdot 10^{-6}$
6	128^3	0.555	1.705	$9.3 \cdot 10^{-6}$
1	256^3	3.588	1.000	$2.1 \cdot 10^{-6}$
2	256^3	3.347	1.072	$2.1 \cdot 10^{-6}$
4	256^3	2.077	1.727	$2.1 \cdot 10^{-6}$
6	256^3	1.703	2.106	$2.1 \cdot 10^{-6}$
1	512^3	27.885	1.000	$2.9 \cdot 10^{-7}$
2	512^3	19.185	1.454	$2.9 \cdot 10^{-7}$
4	512^3	12.294	2.268	$2.9 \cdot 10^{-7}$
6	512^3	11.568	2.411	$2.9 \cdot 10^{-7}$

Таблица 9: Результаты для Polus MPI+OpenMP со 128 нитями при $L = 1$ (сравнение с 1-процессной MPI программой)

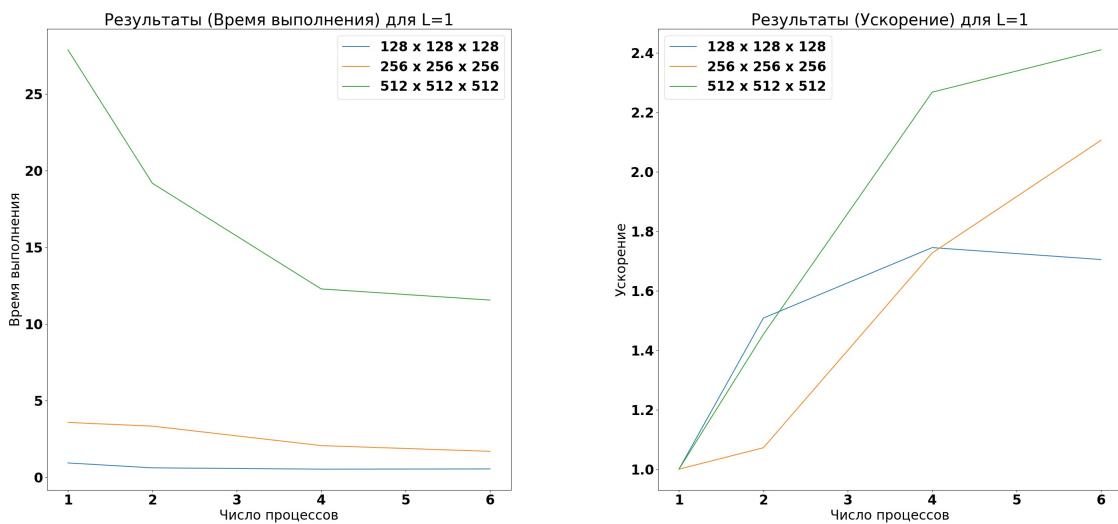


Рис. 11: Графики времени работы и ускорения для Polus MPI+OpenMP со 128 нитями при $L = 1$ (сравнение с 1-процессной MPI программой)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	0.946	1.000	$9.8 \cdot 10^{-7}$
2	128^3	0.619	1.528	$9.8 \cdot 10^{-7}$
4	128^3	0.545	1.736	$9.8 \cdot 10^{-7}$
6	128^3	0.632	1.498	$9.8 \cdot 10^{-7}$
1	256^3	4.713	1.000	$2.4 \cdot 10^{-7}$
2	256^3	3.782	1.246	$2.4 \cdot 10^{-7}$
4	256^3	1.853	2.544	$2.4 \cdot 10^{-7}$
6	256^3	2.887	1.633	$2.4 \cdot 10^{-7}$
1	512^3	27.659	1.000	$5.8 \cdot 10^{-8}$
2	512^3	18.347	1.508	$5.8 \cdot 10^{-8}$
4	512^3	14.241	1.942	$5.8 \cdot 10^{-8}$
6	512^3	10.028	2.758	$5.8 \cdot 10^{-8}$

Таблица 10: Результаты для Polus MPI+OpenMP со 128 нитями при $L = \pi$ (сравнение с 1-процессной MPI программой)

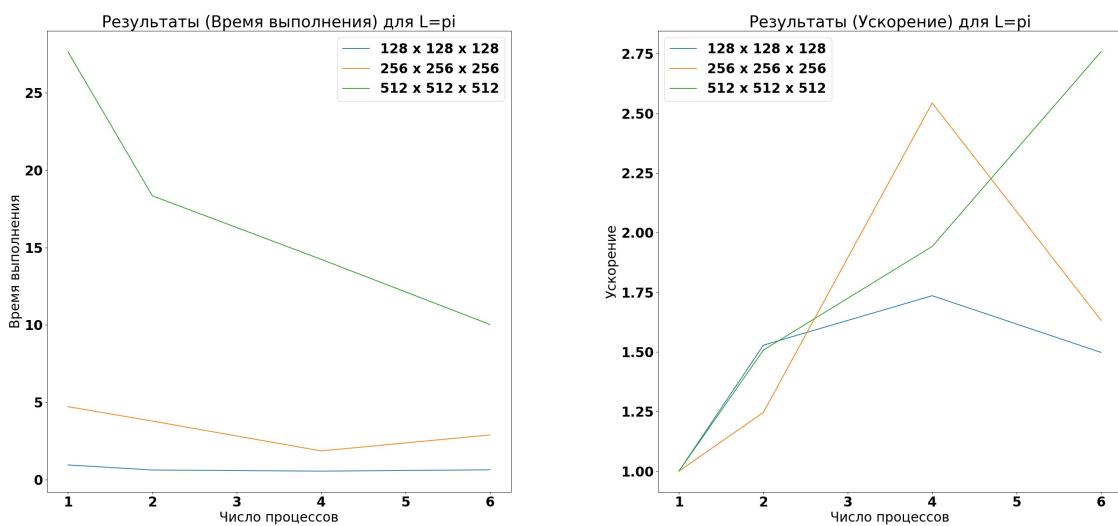


Рис. 12: Графики времени работы и ускорения для Polus MPI+OpenMP со 128 нитями при $L = \pi$ (сравнение с 1-процессной MPI программой)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.213	1.000	$9.3 \cdot 10^{-6}$
2	128^3	0.628	1.932	$9.3 \cdot 10^{-6}$
4	128^3	0.543	2.236	$9.3 \cdot 10^{-6}$
6	128^3	0.555	2.185	$9.3 \cdot 10^{-6}$
1	256^3	8.761	1.000	$2.1 \cdot 10^{-6}$
2	256^3	3.347	2.617	$2.1 \cdot 10^{-6}$
4	256^3	2.077	4.217	$2.1 \cdot 10^{-6}$
6	256^3	1.703	5.143	$2.1 \cdot 10^{-6}$
1	512^3	66.157	1.000	$2.9 \cdot 10^{-7}$
2	512^3	19.185	3.448	$2.9 \cdot 10^{-7}$
4	512^3	12.294	5.381	$2.9 \cdot 10^{-7}$
6	512^3	11.568	5.719	$2.9 \cdot 10^{-7}$

Таблица 11: Результаты для Polus MPI+OpenMP со 128 нитями при $L = 1$ (сравнение с последовательной программой без MPI)

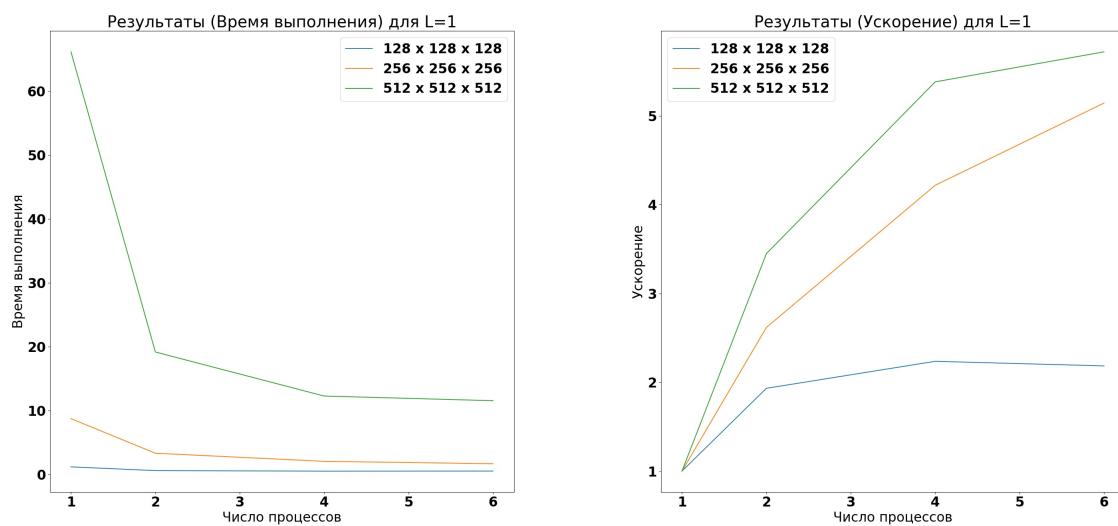


Рис. 13: Графики времени работы и ускорения для Polus MPI+OpenMP со 128 нитями при $L = 1$ (сравнение с последовательной программой без MPI)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.214	1.000	$9.8 \cdot 10^{-7}$
2	128^3	0.619	1.961	$9.8 \cdot 10^{-7}$
4	128^3	0.545	2.227	$9.8 \cdot 10^{-7}$
6	128^3	0.632	1.922	$9.8 \cdot 10^{-7}$
1	256^3	8.762	1.000	$2.4 \cdot 10^{-7}$
2	256^3	3.782	2.317	$2.4 \cdot 10^{-7}$
4	256^3	1.853	4.728	$2.4 \cdot 10^{-7}$
6	256^3	2.887	3.035	$2.4 \cdot 10^{-7}$
1	512^3	66.396	1.000	$5.8 \cdot 10^{-8}$
2	512^3	18.347	3.619	$5.8 \cdot 10^{-8}$
4	512^3	14.241	4.662	$5.8 \cdot 10^{-8}$
6	512^3	10.028	6.621	$5.8 \cdot 10^{-8}$

Таблица 12: Результаты для Polus MPI+OpenMP со 128 нитями при $L = \pi$ (сравнение с последовательной программой без MPI)

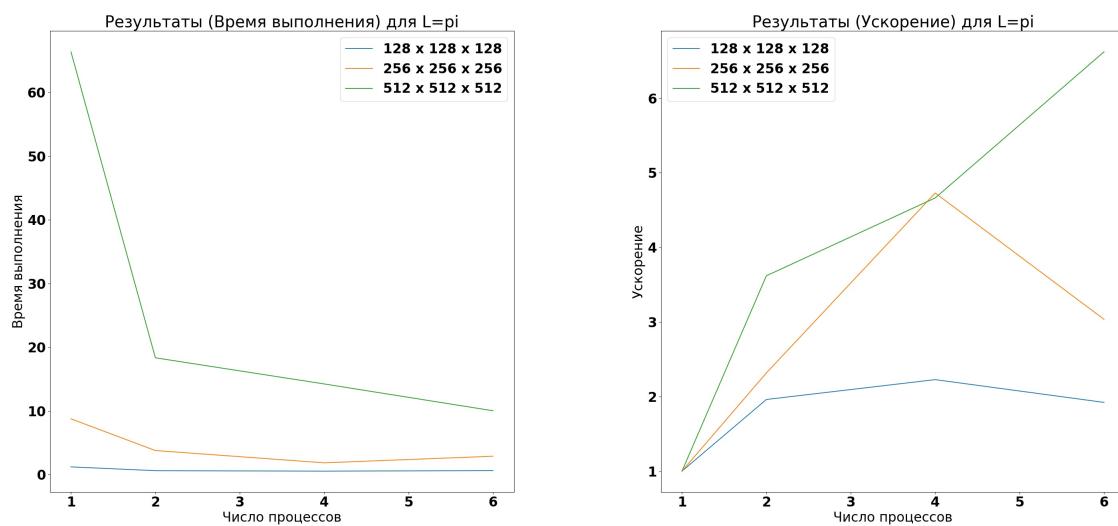


Рис. 14: Графики времени работы и ускорения для Polus MPI+OpenMP со 128 нитями при $L = \pi$ (сравнение с последовательной программой без MPI)

5.3 Результаты запусков MPI+CUDA

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	0.487	1.000	$9.3 \cdot 10^{-6}$
2	128^3	0.338	1.440	$9.3 \cdot 10^{-6}$
4	128^3	0.306	1.589	$9.3 \cdot 10^{-6}$
6	128^3	0.141	3.461	$9.3 \cdot 10^{-6}$
1	256^3	1.243	1.000	$2.1 \cdot 10^{-6}$
2	256^3	0.583	2.132	$2.1 \cdot 10^{-6}$
4	256^3	0.291	4.274	$2.1 \cdot 10^{-6}$
6	256^3	0.217	5.726	$2.1 \cdot 10^{-6}$
1	512^3	9.537	1.000	$2.9 \cdot 10^{-7}$
2	512^3	5.173	1.844	$2.9 \cdot 10^{-7}$
4	512^3	2.369	4.025	$2.9 \cdot 10^{-7}$
6	512^3	1.366	6.980	$2.9 \cdot 10^{-7}$

Таблица 13: Результаты для Polus MPI+CUDA со 128 нитями при $L = 1$ (сравнение с 1-процессной MPI программой)

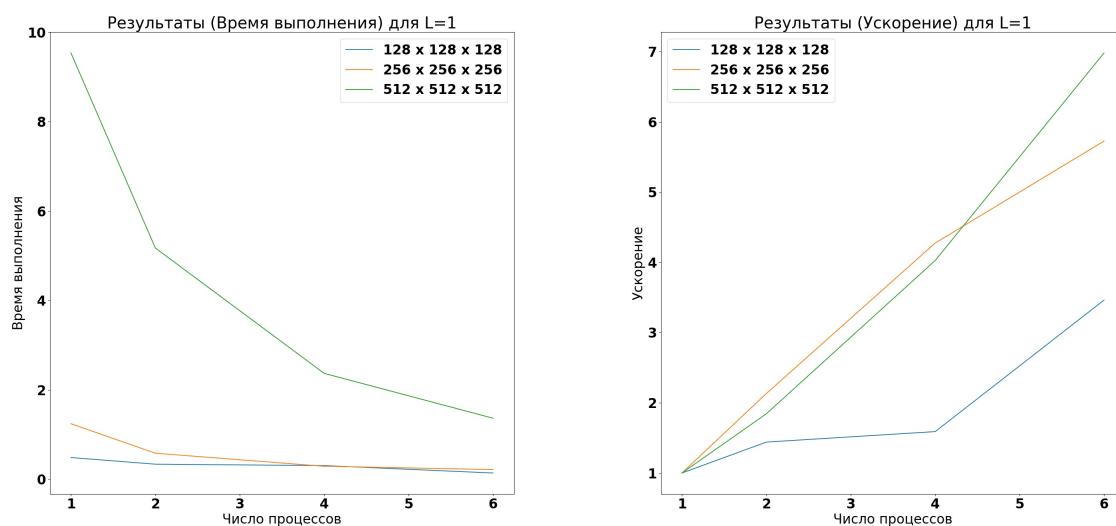


Рис. 15: Графики времени работы и ускорения для Polus MPI+CUDA со 128 нитями при $L = 1$ (сравнение с 1-процессной MPI программой)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	0.507	1.000	$9.8 \cdot 10^{-7}$
2	128^3	0.436	1.163	$9.8 \cdot 10^{-7}$
4	128^3	0.331	1.530	$9.8 \cdot 10^{-7}$
6	128^3	0.162	3.131	$9.8 \cdot 10^{-7}$
1	256^3	1.169	1.000	$2.4 \cdot 10^{-7}$
2	256^3	0.568	2.059	$2.4 \cdot 10^{-7}$
4	256^3	0.292	4.001	$2.4 \cdot 10^{-7}$
6	256^3	0.218	5.371	$2.4 \cdot 10^{-7}$
1	512^3	9.511	1.000	$5.8 \cdot 10^{-8}$
2	512^3	5.389	1.765	$5.8 \cdot 10^{-8}$
4	512^3	2.451	3.881	$5.8 \cdot 10^{-8}$
6	512^3	1.375	6.919	$5.8 \cdot 10^{-8}$

Таблица 14: Результаты для Polus MPI+CUDA со 128 нитями при $L = \pi$ (сравнение с 1-процессной MPI программой)

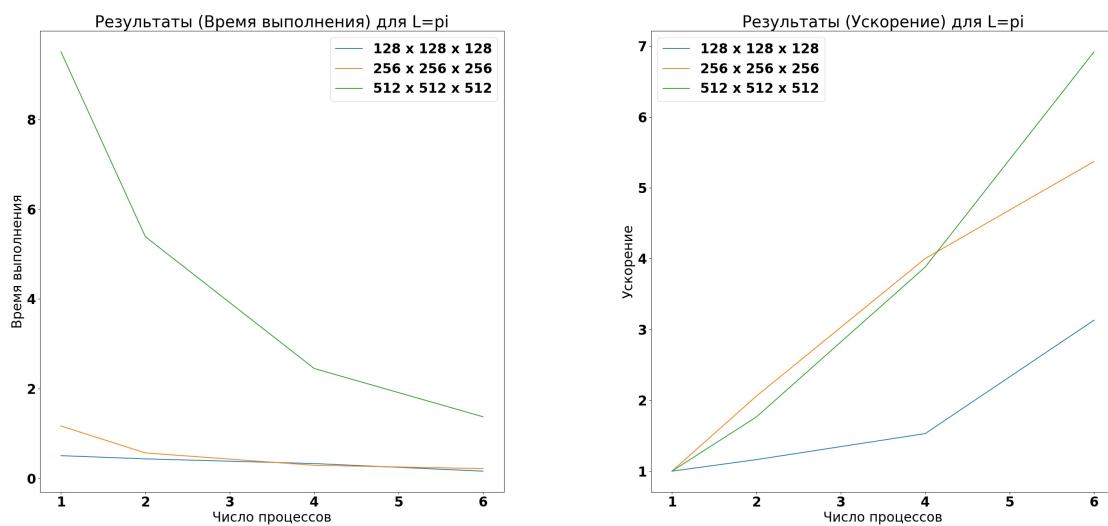


Рис. 16: Графики времени работы и ускорения для Polus MPI+CUDA со 128 нитями при $L = \pi$ (сравнение с 1-процессной MPI программой)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.213	1.000	$9.3 \cdot 10^{-6}$
2	128^3	0.338	3.592	$9.3 \cdot 10^{-6}$
4	128^3	0.306	3.964	$9.3 \cdot 10^{-6}$
6	128^3	0.141	8.633	$9.3 \cdot 10^{-6}$
1	256^3	8.761	1.000	$2.1 \cdot 10^{-6}$
2	256^3	0.583	15.024	$2.1 \cdot 10^{-6}$
4	256^3	0.291	30.121	$2.1 \cdot 10^{-6}$
6	256^3	0.217	40.352	$2.1 \cdot 10^{-6}$
1	512^3	66.157	1.000	$2.9 \cdot 10^{-7}$
2	512^3	5.173	12.789	$2.9 \cdot 10^{-7}$
4	512^3	2.369	27.923	$2.9 \cdot 10^{-7}$
6	512^3	1.366	48.418	$2.9 \cdot 10^{-7}$

Таблица 15: Результаты для Polus MPI+CUDA со 128 нитями при $L = 1$ (сравнение с последовательной программой без MPI)

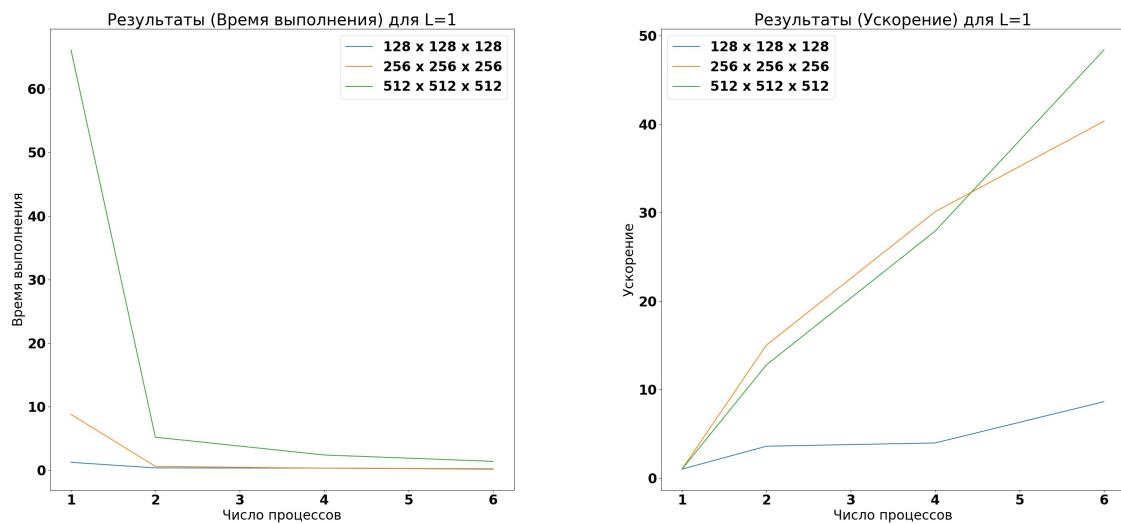


Рис. 17: Графики времени работы и ускорения для Polus MPI+CUDA со 128 нитями при $L = 1$ (сравнение с последовательной программой без MPI)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.214	1.000	$9.8 \cdot 10^{-7}$
2	128^3	0.436	2.785	$9.8 \cdot 10^{-7}$
4	128^3	0.331	3.665	$9.8 \cdot 10^{-7}$
6	128^3	0.162	7.500	$9.8 \cdot 10^{-7}$
1	256^3	8.762	1.000	$2.4 \cdot 10^{-7}$
2	256^3	0.568	15.433	$2.4 \cdot 10^{-7}$
4	256^3	0.292	29.992	$2.4 \cdot 10^{-7}$
6	256^3	0.218	40.262	$2.4 \cdot 10^{-7}$
1	512^3	66.396	1.000	$5.8 \cdot 10^{-8}$
2	512^3	5.389	12.321	$5.8 \cdot 10^{-8}$
4	512^3	2.451	27.091	$5.8 \cdot 10^{-8}$
6	512^3	1.375	48.300	$5.8 \cdot 10^{-8}$

Таблица 16: Результаты для Polus MPI+CUDA со 128 нитями при $L = \pi$ (сравнение с последовательной программой без MPI)

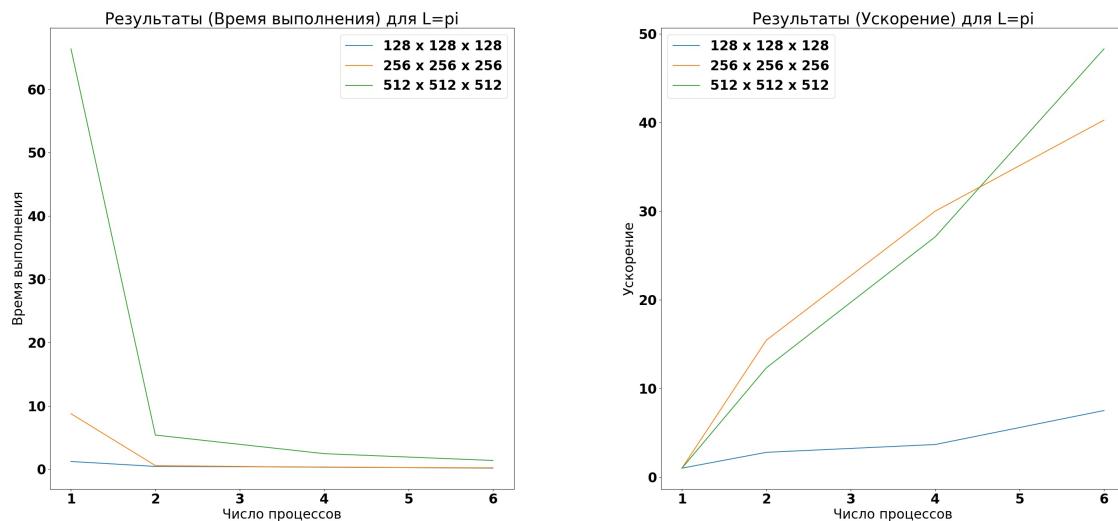


Рис. 18: Графики времени работы и ускорения для Polus MPI+CUDA со 128 нитями при $L = \pi$ (сравнение с последовательной программой без MPI)

5.4 Сравнение результатов

N	p	Последова- тельная версия	MPI	MPI+ OpenMP (8 нитей)	MPI+ OpenMP (128 нитей)	MPI+ CUDA (128 нитей)
128	1	1.213	1.890	0.609	0.947	0.487
128	2	1.213	1.007	0.361	0.628	0.338
128	4	1.213	0.556	0.185	0.543	0.306
128	6	1.213	0.491	0.172	0.555	0.141
256	1	8.761	14.314	4.337	3.588	1.243
256	2	8.761	7.441	4.159	3.347	0.583
256	4	8.761	3.808	2.179	2.077	0.291
256	6	8.761	2.826	1.592	1.703	0.217
512	1	66.157	110.868	32.728	27.885	9.537
512	2	66.157	57.147	29.229	19.185	5.173
512	4	66.157	29.622	12.273	12.294	2.369
512	6	66.157	20.645	11.529	11.568	1.366

Таблица 17: Сравнение времени работы различных версий программы на p процессах при $L = 1$

N	p	MPI	MPI+ OpenMP (8 нитей)	MPI+ OpenMP (128 нитей)	MPI+ CUDA (128 нитей)
128	2	1.204	3.363	1.932	3.592
128	4	2.181	6.566	2.236	3.964
128	6	2.472	7.057	2.185	8.633
256	2	1.177	2.106	2.617	15.024
256	4	2.300	4.021	4.217	30.121
256	6	3.100	5.503	5.143	40.352
512	2	1.158	2.263	3.448	12.789
512	4	2.233	5.390	5.381	27.923
512	6	3.205	5.738	5.719	48.418

Таблица 18: Сравнение ускорения (сравнение с последовательной версией) различных версий программы при $L = 1$

5.5 Профилирование MPI+CUDA

Произведены замеры времени работы различных частей программы с помощью утилиты *nvprof*. Для этого программа запускалась на 1 MPI-процессе с различными размерами сетки N . Общее время, затраченное на выполнение некоторых функций, показано в таблицах 19, 20.

Исходя из полученных результатов, можно сделать вывод о том, что большая часть времени работы программы была потрачена на выделение памяти и копирование участков памяти с хоста на GPU. В параллельных циклах наиболее трудозатратными являются функции вычисления максимальной ошибки на слое и вычисления значений функции на очередном слое. В первом случае это может быть связано с применением операции редукции для вычисления максимума, во втором случае функция выполняет много вычислительно сложных операций.

N	Имя функции	Время выполнения
128	computeLayerErrorKernel	28.380ms
128	getNextUKernel	3.0294ms
	initZeroLayerKernel	201.89us
	fillFirstKindBoundaryKernel	201.22us
	fillPeriodicBoundaryKernel	166.18us
	initFirstLayerKernel	163.24us
256	computeLayerErrorKernel	197.49ms
256	getNextUKernel	24.357ms
	initZeroLayerKernel	1.6097ms
	initFirstLayerKernel	1.2881ms
	fillFirstKindBoundaryKernel	702.74us
	fillPeriodicBoundaryKernel	333.57us
512	computeLayerErrorKernel	1.51539s
512	getNextUKernel	224.93ms
	initZeroLayerKernel	12.826ms
	initFirstLayerKernel	10.817ms
	fillFirstKindBoundaryKernel	2.5307ms
	fillPeriodicBoundaryKernel	1.0409ms

Таблица 19: Времена выполнения параллельных циклов при $L = 1$, $p = 1$

<i>N</i>	Имя функции	Время выполнения
128	cudaMalloc	774.59ms
128	cudaMemcpyAsync	46.096ms
128	cudaDeviceSynchronize	29.777ms
128	cudaLaunchKernel	3.6855ms
128	cudaFree	955.52us
256	cudaMemcpyAsync	737.59ms
256	cudaDeviceSynchronize	228.19ms
256	cudaMalloc	227.72ms
256	cudaLaunchKernel	17.640ms
256	cudaFree	762.97us
512	cudaMemcpyAsync	4.70999s
512	cudaDeviceSynchronize	1.76144s
512	cudaMalloc	238.10ms
512	cudaLaunchKernel	16.218ms
512	cudaFree	3.5114ms

Таблица 20: Времена выполнения вспомогательных функций при $L = 1$, $p = 1$

6 Выводы

Задача для трёхмерного гиперболического уравнения в прямоугольном параллелепипеде отлично подходит для распараллеливания. В результате получены программные средства, решающие поставленную задачу средствами MPI, MPI+OpenMP, MPI+CUDA.

Несмотря на накладные расходы, неизбежно возникающие при исполнении параллельных программ, всё же можно сделать вывод о том, что параллельный алгоритм решения данной задачи может существенно снизить временные затраты на подсчет значений функции, а также позволяет вычислять значения с большей точностью.