



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

СУПЕРКОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ И ТЕХНОЛОГИИ

Отчет по заданию №3 «Задача для трёхмерного гиперболического уравнения в прямоугольном параллелепипеде»

ВАРИАНТ №3

*Богатенкова Анастасия Олеговна
628 группа*

22 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
6 Выводы	16

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 можноенным 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 две программы: последовательная и параллельная с использованием MPI. 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.

Разбиение сетки на блоки происходит рекурсивно по следующему алгоритму:

- разбиение начинается с оси X и целой сетки, изначальный *size* равен числу процессов;
- если текущее количество областей *size* = 1, возвращается текущий параллелепипед;
- если *size* нечётно, то по текущей оси область делится на две части: одна со стороной, делённой на *size*, и оставшаяся часть. Первая часть достаётся 1 процессу, для оставшейся выполняется следующий пункт;
- по выбранной оси область делится пополам, алгоритм запускается рекурсивно для подобластей переходя на следующую ось ($X \rightarrow Y$; $Y \rightarrow Z$; $Z \rightarrow X$).

4 Графики решений и погрешности

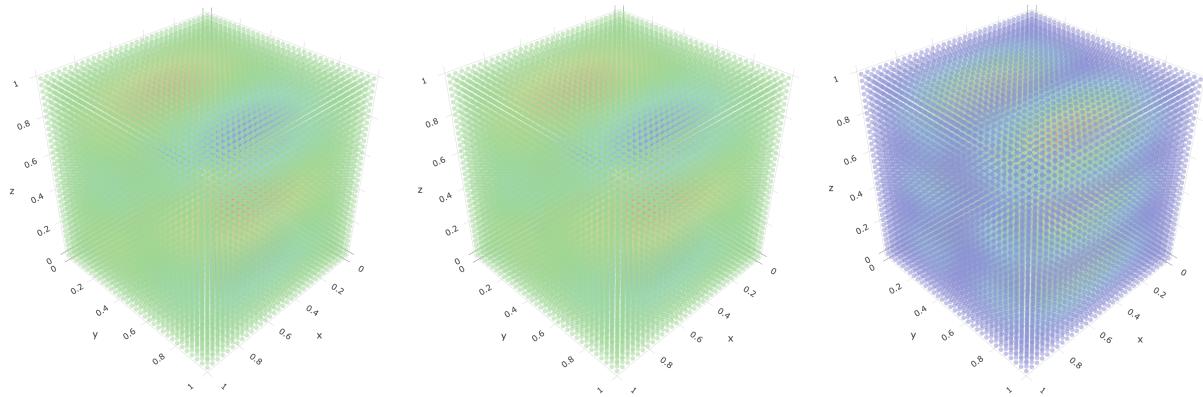


Рис. 1: Графики аналитической функции, вычисленной функции и погрешности вычисления для $L_x = L_y = L_z = 1$

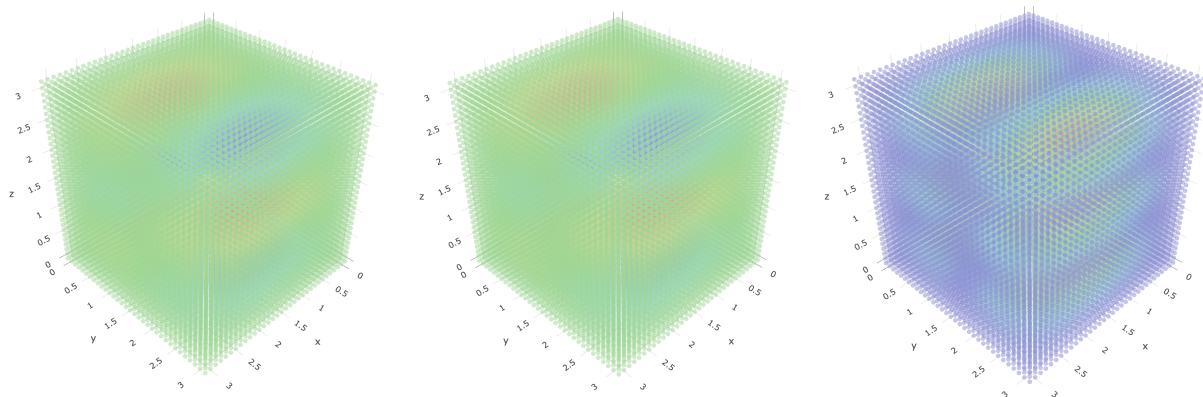


Рис. 2: Графики аналитической функции, вычисленной функции и погрешности вычисления для $L_x = L_y = L_z = \pi$

5 Результаты расчётов

В таблицах 1, 2, 3, 4, 5, 6, 7, 8 и на графиках 3, 4, 5, 6, 7, 8, 9, 10 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.

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 4 нитями на процесс.

Результаты ускорения были получены с использованием запусков как MPI-программы, запущенной на 1 процессе, так и с использованием последовательной программы, не использующей MPI. Результаты сравнения с последовательной программой хуже результатов сравнения с MPI-программой, однако они улучшаются по мере увеличения размера сетки.

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.890	1.000	$9.3 \cdot 10^{-6}$
4	128^3	0.556	3.397	$9.3 \cdot 10^{-6}$
8	128^3	0.442	4.276	$9.3 \cdot 10^{-6}$
16	128^3	0.277	6.821	$9.3 \cdot 10^{-6}$
32	128^3	0.242	7.805	$9.3 \cdot 10^{-6}$
1	256^3	14.314	1.000	$2.1 \cdot 10^{-6}$
4	256^3	3.808	3.759	$2.1 \cdot 10^{-6}$
8	256^3	2.073	6.905	$2.1 \cdot 10^{-6}$
16	256^3	1.288	11.112	$2.1 \cdot 10^{-6}$
32	256^3	0.854	16.766	$2.1 \cdot 10^{-6}$
1	512^3	110.868	1.000	$2.9 \cdot 10^{-7}$
4	512^3	29.622	3.743	$2.9 \cdot 10^{-7}$
8	512^3	16.497	6.720	$2.9 \cdot 10^{-7}$
16	512^3	8.942	12.398	$2.9 \cdot 10^{-7}$
32	512^3	5.429	20.421	$2.9 \cdot 10^{-7}$

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

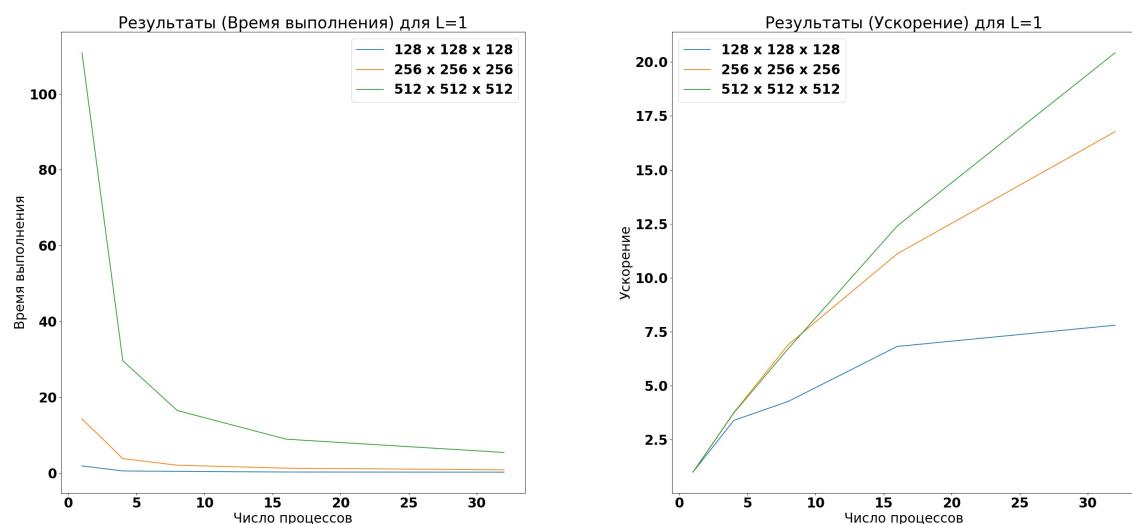


Рис. 3: Графики времени работы и ускорения для Polus при $L = 1$ (сравнение с 1-процессной MPI программой)

Число MPI-процессов p	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.895	1.000	$9.8 \cdot 10^{-7}$
4	128^3	0.543	3.492	$9.8 \cdot 10^{-7}$
8	128^3	0.422	4.492	$9.8 \cdot 10^{-7}$
16	128^3	0.275	6.886	$9.8 \cdot 10^{-7}$
32	128^3	0.273	6.932	$9.8 \cdot 10^{-7}$
1	256^3	14.354	1.000	$2.4 \cdot 10^{-7}$
4	256^3	3.876	3.703	$2.4 \cdot 10^{-7}$
8	256^3	2.114	6.789	$2.4 \cdot 10^{-7}$
16	256^3	1.362	10.543	$2.4 \cdot 10^{-7}$
32	256^3	0.908	15.809	$2.4 \cdot 10^{-7}$
1	512^3	110.897	1.000	$5.8 \cdot 10^{-8}$
4	512^3	29.485	3.761	$5.8 \cdot 10^{-8}$
8	512^3	16.643	6.663	$5.8 \cdot 10^{-8}$
16	512^3	8.924	12.427	$5.8 \cdot 10^{-8}$
32	512^3	5.466	20.289	$5.8 \cdot 10^{-8}$

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

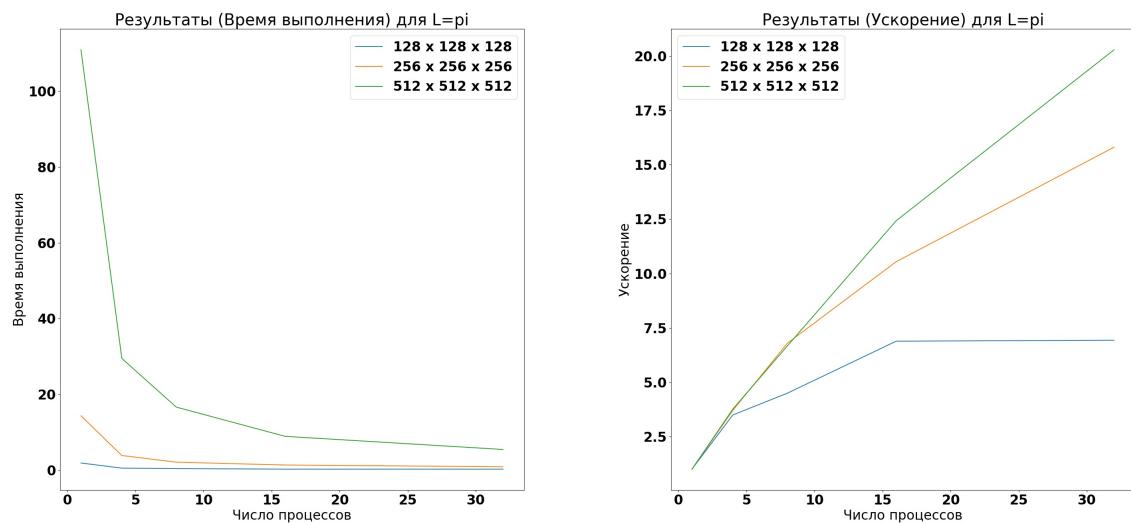


Рис. 4: Графики времени работы и ускорения для Polus при $L = \pi$ (сравнение с 1-процессной MPI программой)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.213	1.000	$9.3 \cdot 10^{-6}$
4	128^3	0.556	2.181	$9.3 \cdot 10^{-6}$
8	128^3	0.442	2.745	$9.3 \cdot 10^{-6}$
16	128^3	0.277	4.379	$9.3 \cdot 10^{-6}$
32	128^3	0.242	5.011	$9.3 \cdot 10^{-6}$
1	256^3	8.761	1.000	$2.1 \cdot 10^{-6}$
4	256^3	3.808	2.300	$2.1 \cdot 10^{-6}$
8	256^3	2.073	4.226	$2.1 \cdot 10^{-6}$
16	256^3	1.288	6.801	$2.1 \cdot 10^{-6}$
32	256^3	0.854	10.261	$2.1 \cdot 10^{-6}$
1	512^3	66.157	1.000	$2.9 \cdot 10^{-7}$
4	512^3	29.622	2.233	$2.9 \cdot 10^{-7}$
8	512^3	16.497	4.010	$2.9 \cdot 10^{-7}$
16	512^3	8.942	7.398	$2.9 \cdot 10^{-7}$
32	512^3	5.429	12.185	$2.9 \cdot 10^{-7}$

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

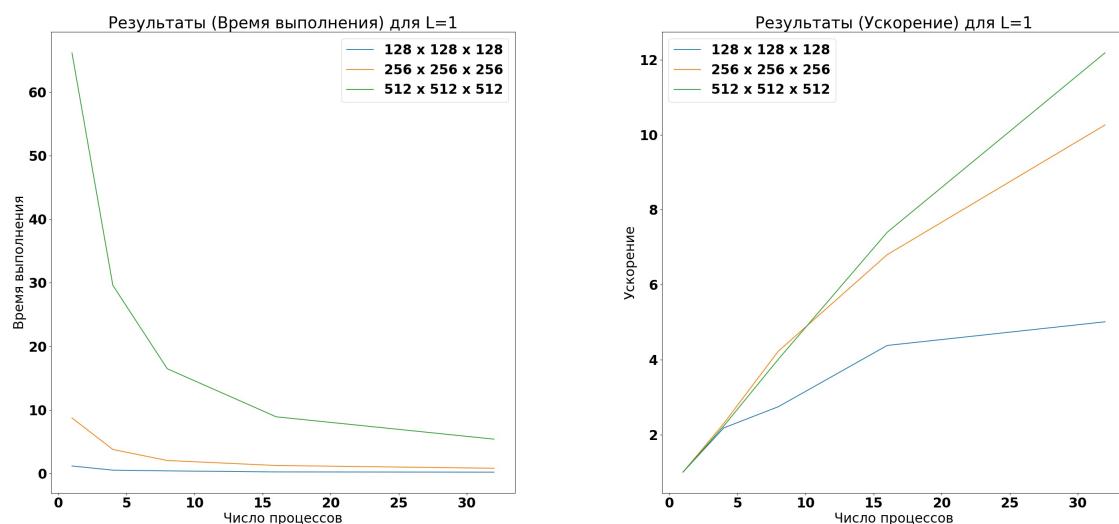


Рис. 5: Графики времени работы и ускорения для Polus при $L = 1$ (сравнение с последовательной программой без MPI)

Число MPI-процессов p	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.214	1.000	$9.8 \cdot 10^{-7}$
4	128^3	0.543	2.236	$9.8 \cdot 10^{-7}$
8	128^3	0.422	2.877	$9.8 \cdot 10^{-7}$
16	128^3	0.275	4.410	$9.8 \cdot 10^{-7}$
32	128^3	0.273	4.439	$9.8 \cdot 10^{-7}$
1	256^3	8.762	1.000	$2.4 \cdot 10^{-7}$
4	256^3	3.876	2.260	$2.4 \cdot 10^{-7}$
8	256^3	2.114	4.144	$2.4 \cdot 10^{-7}$
16	256^3	1.362	6.435	$2.4 \cdot 10^{-7}$
32	256^3	0.908	9.649	$2.4 \cdot 10^{-7}$
1	512^3	66.396	1.000	$5.8 \cdot 10^{-8}$
4	512^3	29.485	2.252	$5.8 \cdot 10^{-8}$
8	512^3	16.643	3.990	$5.8 \cdot 10^{-8}$
16	512^3	8.924	7.440	$5.8 \cdot 10^{-8}$
32	512^3	5.466	12.147	$5.8 \cdot 10^{-8}$

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

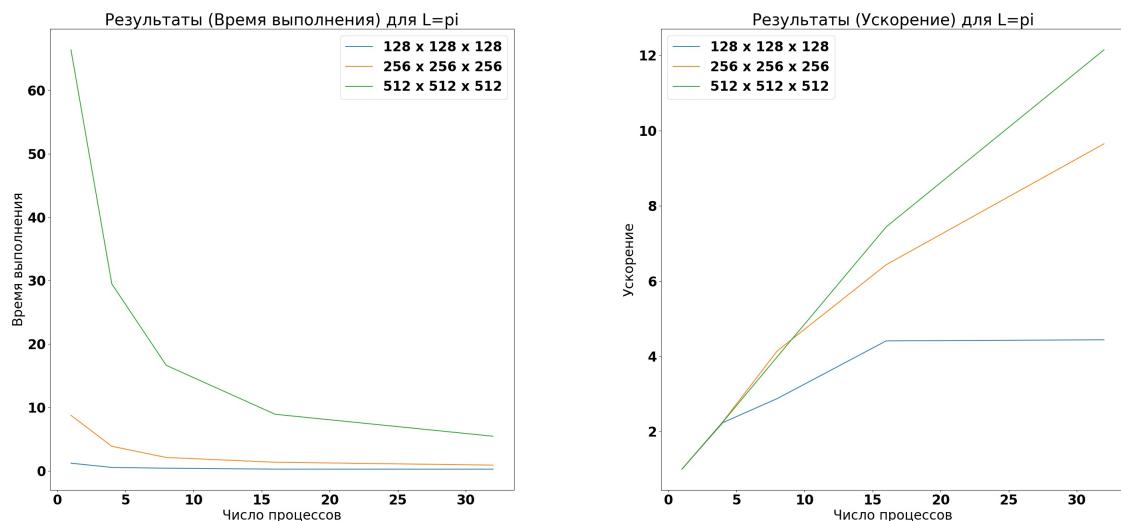


Рис. 6: Графики времени работы и ускорения для Polus при $L = \pi$ (сравнение с последовательной программой без MPI)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	0.837	1.000	$9.3 \cdot 10^{-6}$
4	128^3	0.271	3.087	$9.3 \cdot 10^{-6}$
8	128^3	0.254	3.301	$9.3 \cdot 10^{-6}$
16	128^3	0.143	5.853	$9.3 \cdot 10^{-6}$
32	128^3	0.105	7.949	$9.3 \cdot 10^{-6}$
1	256^3	5.053	1.000	$2.1 \cdot 10^{-6}$
4	256^3	1.434	3.523	$2.1 \cdot 10^{-6}$
8	256^3	1.076	4.694	$2.1 \cdot 10^{-6}$
16	256^3	0.923	5.472	$2.1 \cdot 10^{-6}$
32	256^3	0.666	7.589	$2.1 \cdot 10^{-6}$
1	512^3	39.389	1.000	$2.9 \cdot 10^{-7}$
4	512^3	9.967	3.952	$2.9 \cdot 10^{-7}$
8	512^3	9.898	3.980	$2.9 \cdot 10^{-7}$
16	512^3	5.888	6.690	$2.9 \cdot 10^{-7}$
32	512^3	3.438	11.457	$2.9 \cdot 10^{-7}$

Таблица 5: Результаты для Polus MPI+OpenMP с 4 нитями при $L = 1$ (сравнение с 1-процессной MPI программой)

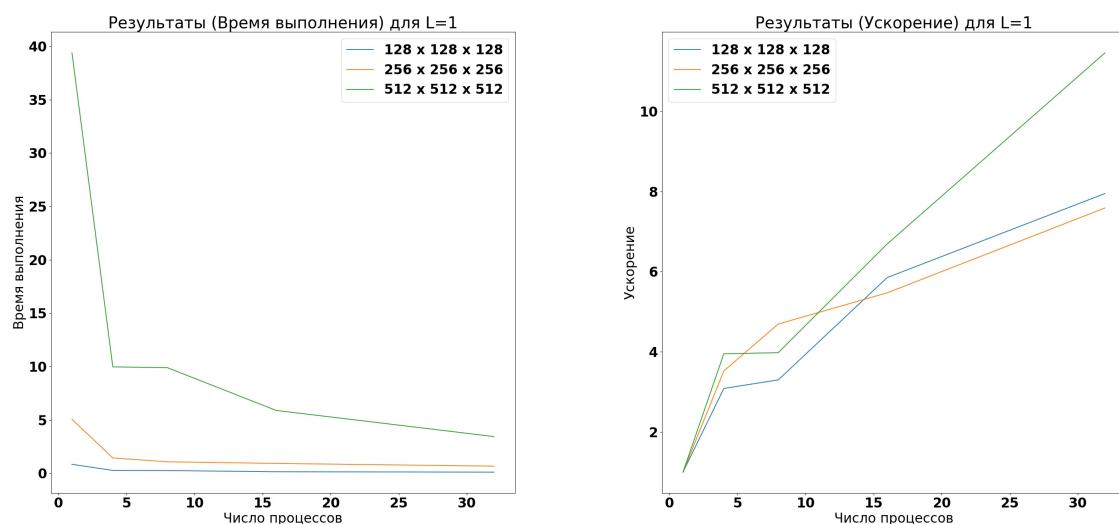


Рис. 7: Графики времени работы и ускорения для Polus MPI+OpenMP с 4 нитями при $L = 1$ (сравнение с 1-процессной MPI программой)

Число MPI-процессов p	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	1.895	1.000	$9.8 \cdot 10^{-7}$
4	128^3	0.543	3.492	$9.8 \cdot 10^{-7}$
8	128^3	0.422	4.492	$9.8 \cdot 10^{-7}$
16	128^3	0.275	6.886	$9.8 \cdot 10^{-7}$
32	128^3	0.273	6.932	$9.8 \cdot 10^{-7}$
1	256^3	14.354	1.000	$2.4 \cdot 10^{-7}$
4	256^3	3.876	3.703	$2.4 \cdot 10^{-7}$
8	256^3	2.114	6.789	$2.4 \cdot 10^{-7}$
16	256^3	1.362	10.543	$2.4 \cdot 10^{-7}$
32	256^3	0.908	15.809	$2.4 \cdot 10^{-7}$
1	512^3	110.897	1.000	$5.8 \cdot 10^{-8}$
4	512^3	29.485	3.761	$5.8 \cdot 10^{-8}$
8	512^3	16.643	6.663	$5.8 \cdot 10^{-8}$
16	512^3	8.924	12.427	$5.8 \cdot 10^{-8}$
32	512^3	5.466	20.289	$5.8 \cdot 10^{-8}$

Таблица 6: Результаты для Polus MPI+OpenMP с 4 нитями при $L = \pi$ (сравнение с 1-процессной MPI программой)

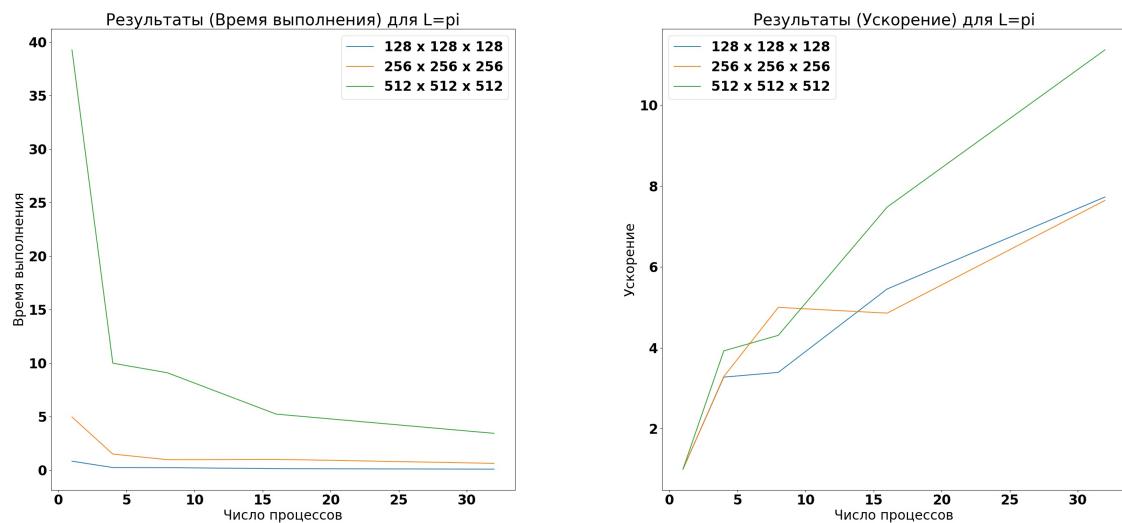


Рис. 8: Графики времени работы и ускорения для Polus MPI+OpenMP с 4 нитями при $L = \pi$ (сравнение с 1-процессной MPI программой)

Число MPI-процессов	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	0.601	1.000	$9.3 \cdot 10^{-6}$
4	128^3	0.271	2.215	$9.3 \cdot 10^{-6}$
8	128^3	0.254	2.369	$9.3 \cdot 10^{-6}$
16	128^3	0.143	4.201	$9.3 \cdot 10^{-6}$
32	128^3	0.105	5.705	$9.3 \cdot 10^{-6}$
1	256^3	2.927	1.000	$2.1 \cdot 10^{-6}$
4	256^3	1.434	2.041	$2.1 \cdot 10^{-6}$
8	256^3	1.076	2.719	$2.1 \cdot 10^{-6}$
16	256^3	0.923	3.170	$2.1 \cdot 10^{-6}$
32	256^3	0.666	4.397	$2.1 \cdot 10^{-6}$
1	512^3	19.470	1.000	$2.9 \cdot 10^{-7}$
4	512^3	9.967	1.953	$2.9 \cdot 10^{-7}$
8	512^3	9.898	1.967	$2.9 \cdot 10^{-7}$
16	512^3	5.888	3.307	$2.9 \cdot 10^{-7}$
32	512^3	3.438	5.663	$2.9 \cdot 10^{-7}$

Таблица 7: Результаты для Polus MPI+OpenMP с 4 нитями при $L = 1$ (сравнение с последовательной программой без MPI)

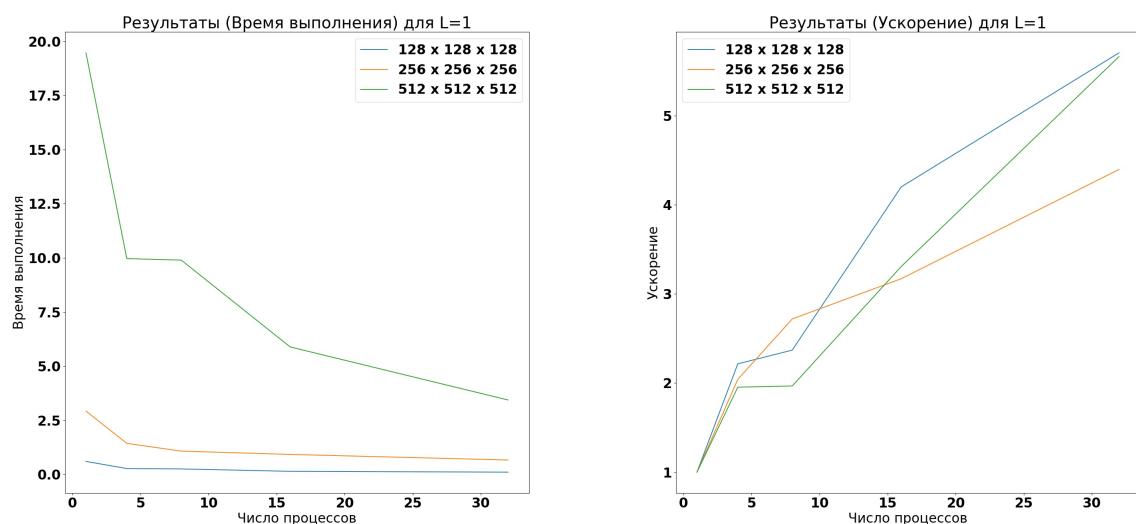


Рис. 9: Графики времени работы и ускорения для Polus MPI+OpenMP с 4 нитями при $L = 1$ (сравнение с последовательной программой без MPI)

Число MPI-процессов p	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	128^3	0.553	1.000	$9.8 \cdot 10^{-7}$
4	128^3	0.261	2.123	$9.8 \cdot 10^{-7}$
8	128^3	0.252	2.198	$9.8 \cdot 10^{-7}$
16	128^3	0.157	3.535	$9.8 \cdot 10^{-7}$
32	128^3	0.110	5.010	$9.8 \cdot 10^{-7}$
1	256^3	3.127	1.000	$2.4 \cdot 10^{-7}$
4	256^3	1.515	2.064	$2.4 \cdot 10^{-7}$
8	256^3	0.997	3.137	$2.4 \cdot 10^{-7}$
16	256^3	1.026	3.048	$2.4 \cdot 10^{-7}$
32	256^3	0.652	4.796	$2.4 \cdot 10^{-7}$
1	512^3	19.472	1.000	$5.8 \cdot 10^{-8}$
4	512^3	10.003	1.947	$5.8 \cdot 10^{-8}$
8	512^3	9.113	2.137	$5.8 \cdot 10^{-8}$
16	512^3	5.248	3.710	$5.8 \cdot 10^{-8}$
32	512^3	3.454	5.638	$5.8 \cdot 10^{-8}$

Таблица 8: Результаты для Polus MPI+OpenMP с 4 нитями при $L = \pi$ (сравнение с последовательной программой без MPI)

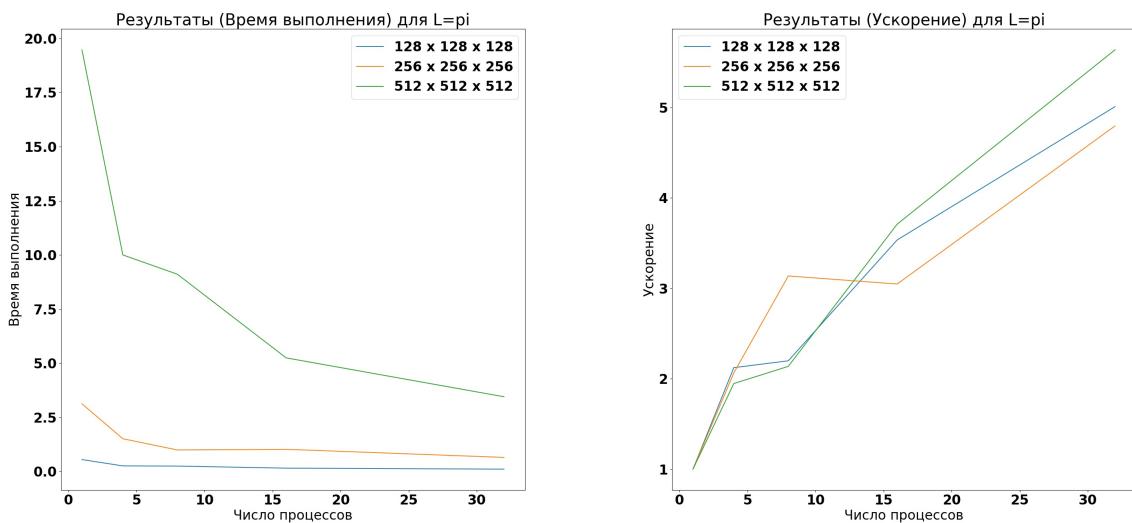


Рис. 10: Графики времени работы и ускорения для Polus MPI+OpenMP с 4 нитями при $L = \pi$ (сравнение с последовательной программой без MPI)

6 Выводы

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

Результаты расчётов показали, что чем выше число точек сетки, тем лучше показатели распараллеливания программы, так как накладные расходы, связанные с пересылкой данных между процессами становятся незначительными по сравнению со сложностью задачи, решаемой каждым из процессов. При этом, если сравнивать результаты распараллеливания программы с последовательной версией, они становятся хуже в силу того, что в последовательной версии нет многочисленных затратных «лишних» инструкций, замедляющих её работу.

Несмотря на накладные расходы, 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.