

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

**Параллельный алгоритм решения
трёхмерного волнового уравнения с
использованием графических ускорителей**

Выполнила:
Яркова Юлия Сергеевна
группа 622

Москва
2025

Математическая постановка дифференциальной 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

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

с начальными условиями

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

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

и граничными условиями варианта 5:

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

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

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

$$h_x N = L_x, \ h_y N = L_y, \ h_z N = L_z,$$

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

Аппроксимируем уравнение (1) явной разностной схемой:

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

где Δ_h — семиточечный разностный аналог оператора Лапласа:

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

Начальные условия аппроксимируем следующим образом:

$$u_{ijk}^0 = \varphi(x_i, y_j, z_k), \quad (x_i, y_j, z_k) \in \omega_h, \quad (5)$$

$$u_{ijk}^1 = u_{ijk}^0 + a^2 \frac{\tau^2}{2} \Delta_h \varphi(x_i, y_j, z_k), \quad (x_i, y_j, z_k) \in \omega_h. \quad (6)$$

Граничные условия варианта 5 аппроксимируются на сетке:

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

для $i, j, k = 0, 1, \dots, N$.

Постановка задачи для варианта 5

Исходные данные

- Граничные условия:
 - По x : периодические
 - По y : первого рода (однородные)
 - По z : первого рода (однородные)
- Аналитическое решение:

$$u_{\text{analytical}} = \sin\left(\frac{2\pi}{L_x}x\right) \cdot \sin\left(\frac{\pi}{L_y}y\right) \cdot \sin\left(\frac{\pi}{L_z}z\right) \cdot \cos(a_t t + 2\pi),$$

где

$$a_t = \frac{1}{2} \sqrt{\frac{4}{L_x^2} + \frac{1}{L_y^2} + \frac{1}{L_z^2}}, \quad a^2 = \frac{1}{4\pi^2}.$$

- Начальная функция:

$$\varphi(x, y, z) = u_{\text{analytical}}(x, y, z, 0) = \sin\left(\frac{2\pi}{L_x}x\right) \cdot \sin\left(\frac{\pi}{L_y}y\right) \cdot \sin\left(\frac{\pi}{L_z}z\right).$$

- Параметры:
 - L_x, L_y, L_z — размеры области (варианты: все равны 1 или π)
 - T — конечное время
 - N — число узлов по каждому направлению
 - K — число шагов по времени (около 20 шагов)

-

Требуется:

Расширить имеющуюся программу в последнем задании с использованием MPI+CUDA, основываясь на реализованных ранее параллельных версиях программы.

Реализация задания 4 с использованием CUDA

В данной программе реализовано численное решение трёхмерного волнового уравнения в кубической области с использованием гибридной параллельной технологии MPI + CUDA. Расчётная область по координатам x , y и z разбивается между MPI-процессами с помощью декартовой топологии, при этом каждому процессу соответствует свой локальный поддомен с добавлением призрачных слоёв для обмена граничными значениями. Внутри каждого MPI-процесса вычисления выполняются на GPU.

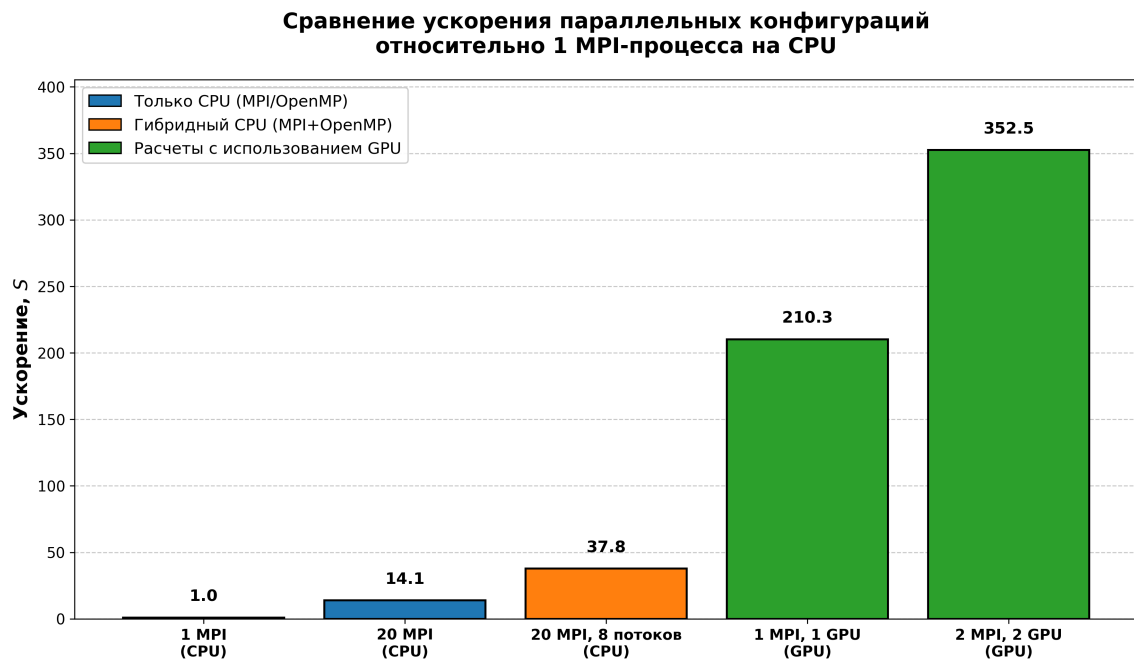


Рис. 1: Сравнение ускорений между различными вариантами программы по полному времени работы.

Начальные условия и граничные условия задаются на основе известного аналитического решения, что позволяет на каждом временном шаге оценивать максимальную погрешность численного решения. Первый и второй временные слои вычисляются отдельными CUDA-ядрами, после чего используется стандартная конечно-разностная схема второго порядка по времени. На каждом шаге времени выполняется обмен граничными слоями между соседними процессами по всем трём направлениям с помощью неблокирующих операций MPI. В процессе моделирования программа выводит максимальную ошибку относительно аналитического решения для каждого временного шага, а по завершении вычислений измеряется и выводится общее время выполнения (максимальное по всем MPI-процессам).

Результаты вычислительных экспериментов

Для написанной параллельной программы были проведены эксперименты на задаче в единичном кубе с $N = 256$. Во всех экспериментах было рассчитано 50 итераций по времени.

В таблице 1 представлено сравнение производительности для различных реализаций параллельной программы по чистому времени вычислений, т.е. без учета инициализации буферов и коммуникаций между процессами и устройствами. Видно, что распараллеливая программу только на CPU, можно добиться ускорения лишь в 38 раз. Параллельная GPU-программа уже дает ускорение в 10 раз больше; при использовании сразу 2 графических карт ускорение так же возрастает почти в 2 раза, достигая более 700x прироста по скорости работы по сравнению с последовательной версией программы. Сравнение по полному времени работы (включая инициализацию) представлено на Рис. 1. Уменьшение абсолютных значений ускорений для CUDA программ вызвано тем, что в данном случае инициализация и межпроцессные

коммуникации занимают около 40-50% времени работы программы; в CPU программах вычисления производятся гораздо медленней, поэтому добавление времени на инициализацию не так заметно. Но даже по чистому времени работы программа на CUDA выполняется в 6-10 раз быстрее самой жфффективной параллельной имплементации на CPU. Результаты для отдельных параллельных реализаций программы на CPU можно увидеть в таблицах 3, 4, 5. Соответствующие графики ускорений представлены на рис. 2, 3 и рис. 4.

В таблице 2 представлена подробная статистика по времени выполнения различных этапов MPI+CUDA версии программы. Можно заметить, что время выполнения CUDA-ядер занимает всего около 50% от общего времени выполнения программы, т.к. размер рассматриваемой задачи сравнительно небольшой и времени на накладные расходы уходит столько же, сколько и на сами вычисления. На более массивных задачах время вычислений все-таки будет превышать время на накладные расходы.

Таблица 1: Сравнение производительности последовательной программы, MPI, MPI + OpenMP и GPU по чистому времени вычислений

Число MPI процессов	Число нитей	Число GPU	Время, сек	Ускорение	Погрешность
1	–	–	119.86	1.0	2.69e-7
20	–	–	8.52	14.1	2.69e-7
20	8	–	3.17	37.8	2.69e-7
1	–	1	0.32	374.6	2.69e-7
2	–	2	0.17	705.1	2.69e-7

Таблица 2: Подробная статистика по различным этапам MPI+CUDA-версии программы с 2 процессами.

Описание этапа	Время, сек
Общее время выполнения	0.339
Время инициализации	0.087
Время копирования с устройства на хост	0.012
Время копирования с хоста на устройство	0.028
Время обмена граничными условиями	0.006
Время основных вычислений	0.174

Таблица 3: Результаты работы OpenMP программы.

Число OpenMP нитей	N^3	Время	Ускорение	Погрешность
1	256 ³	120.08	1.000	2.69×10^{-7}
2	256 ³	62.26	1.928	2.69×10^{-7}
4	256 ³	32.45	3.700	2.69×10^{-7}
8	256 ³	17.43	6.889	2.69×10^{-7}
16	256 ³	10.32	11.635	2.69×10^{-7}
32	256 ³	9.28	12.939	2.69×10^{-7}

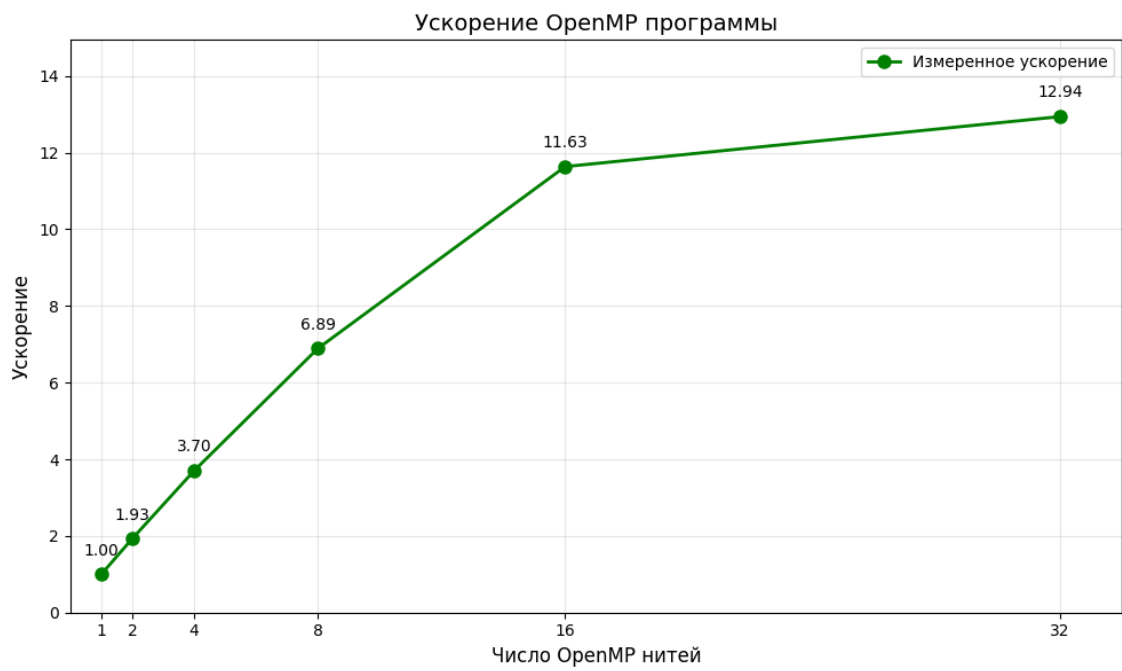


Рис. 2: График ускорений в зависимости от числа потоков для OpenMP программы.

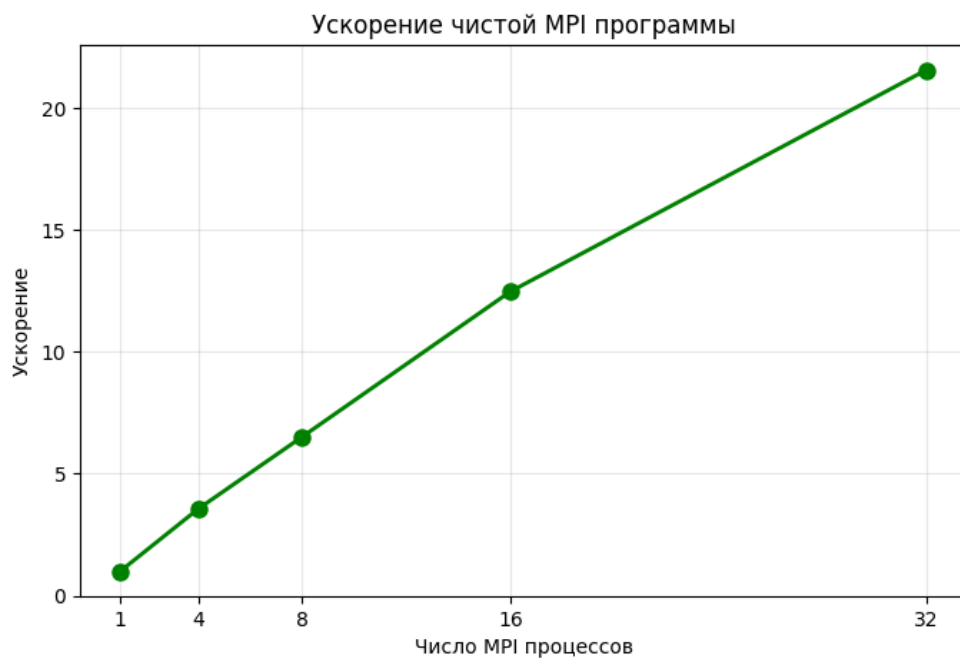


Рис. 3: График ускорений в зависимости от числа процессов для MPI программы.

Таблица 4: Результаты работы MPI программы.

MPI процессы	N^3	Время	Ускорение	Погрешность
1	256^3	118.88	1.00	2.69×10^{-7}
4	256^3	32.26	3.56	2.69×10^{-7}
8	256^3	17.67	6.52	2.69×10^{-7}
16	256^3	9.23	12.48	2.69×10^{-7}
32	256^3	5.34	21.58	2.69×10^{-7}

Таблица 5: Результаты работы OpenMP+MPI программы

MPI процессы	# нитей	N^3	Время	Ускорение	Погрешность
8	1	256^3	22.92	1.00	2.69×10^{-7}
8	2	256^3	11.79	1.88	2.69×10^{-7}
8	4	256^3	8.49	2.63	2.69×10^{-7}
8	8	256^3	6.17	3.59	2.69×10^{-7}

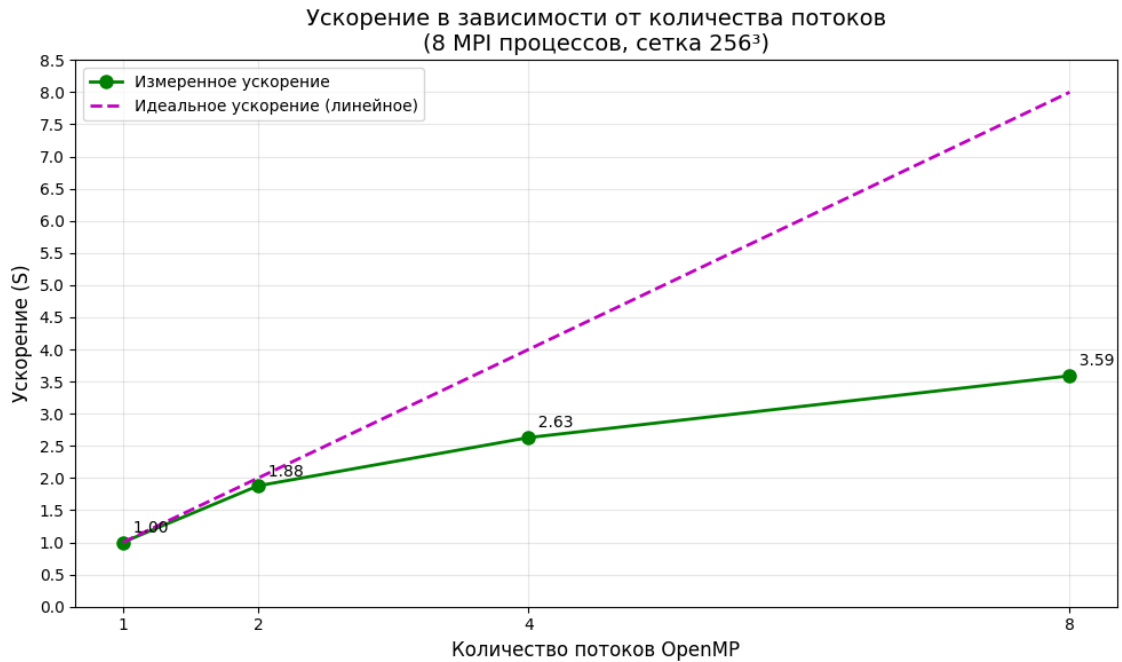


Рис. 4: График ускорений в зависимости от числа потоков для MPI+OpenMP программы с 8 процессами.