Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

“Суперкомпьютерное моделирование и технологии”

Решение трёхмерного гиперболического уравнения в прямоугольном параллелепипеде

ОТЧЕТ

о выполненном задании

студентка 620 группы

Руденко Ксения Рустамовна

Вариант 3

Москва, 2023

Оглавление

1. Постановка задачи

Решается задача для трехмерного гиперболического уравнения в области прямоугольного параллелепипеда. Данное уравнение часто применяется в теории тепло- и массопереноса, гидро- и аэромеханике, электростатике. Таким образом, данная задача является актуальной в рамках современного суперкомпьютерного моделирования.

Постановка задачи:
В трёхмерной замкнутой области

$$\Omega = [0 \le x \le Lx] \times [0 \le y \le Ly] \times [0 \le z \le Lz]$$

для $(0 < t \le T]$ 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

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

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

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

$$u(0, y, z, t) = 0, \qquad u(L_x, y, z, t) = 0, \quad (4)$$
$$u(x, 0, z, t) = 0, \qquad u(x, L_y, z, t) = 0, \quad (5)$$
$$u(x, y, 0, t) = 0, \qquad u(x, y, L_z, t) = 0, \quad (6)$$

либо периодические граничные условия

$$u(0, y, z, t) = u(L_x, y, z, t), \qquad u_x(0, y, z, t) = u_x(L_x, y, z, t), \quad (7)$$
$$u(x, 0, z, t) = u(x, L_y, z, t), \qquad u_y(x, 0, z, t) = u_y(x, L_y, z, t), \quad (8)$$
$$u(x, y, 0, t) = u(x, y, L_z, t), \qquad u_z(x, y, 0, t) = u_z(x, y, L_z, t), \quad (9)$$

В варианте 3 граничные условия представлены в (10) – (12):

$$u(0, y, z, t) = 0, \qquad u(L_x, y, z, t) = 0, \quad (10)$$
$$u(x, y, 0, t) = 0, \qquad u(x, y, L_z, t) = 0, \quad (11)$$
$$u(x, 0, z, t) = u(x, L_y, z, t), \qquad u_y(x, 0, z, t) = u_y(x, L_y, z, t), \quad (12)$$

Аналитическое уравнение функции u:

$$u(x,y,z,t) = \sin\left(\frac{\pi}{L_x}x\right)\cdot\sin\left(\frac{2\pi}{L_y}y\right)\cdot\sin\left(\frac{3\pi}{L_z}z\right)\cdot\cos(a_t\cdot t),$$

$$a_t = \frac{\pi}{2}\sqrt{\frac{1}{L_x^2}+\frac{4}{L_y^2}+\frac{9}{L_z^2}},$$

$$a^2 = {}^1/_4$$

2. Численный метод решения задачи

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

$T = T_0$,
$L_x = L_{x_0}$, $L_y = L_{y_0}$, $L_z = L_{z_0}$,
$\bar{\omega}_h = \{(x_i = ih_x, y_j = jh_y, z_k = kh_z), \mathrm{i,j,k} = 0,1,\dots,\mathrm{N}, h_x\mathrm{N} = L_x, h_y\mathrm{N} = L_y, h_z\mathrm{N} = L_z,\}$
$\omega_\tau = \{t_n = n\tau, n = 0,\dots,K, \tau K = T\}$.

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

Для аппроксимации исходного уравнения (1) однородными граничными условиями (4) – (6) и начальными условиями (2) – (3) воспользуемся следующей системой уравнений:

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

Здесь Δ_h– семиточечный разностный аналог оператора Лапласа:

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

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 (те для 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

$$u_{ijk}^0 = \varphi(x_i, y_j, z_k), \quad (x_i, y_j, z_k) \in \omega_h. \tag{10}$$

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

$$\frac{u_{ijk}^1 - u_{ijk}^0}{\tau} = \frac{\tau}{2}\Delta_h\varphi(x_{i,}y_j, z_k), \quad (x_{i,}y_j, z_k) \in \omega_h. \tag{11}$$

$$u_{ijk}^1 = u_{ijk}^0 + \frac{\tau^2}{2}\Delta_h\varphi(x_{i,}y_j, z_k), \tag{12}$$

Разностная аппроксимация для периодических граничных условий выглядит следующим образом:

$$u_{0jk}^{n+1} = u_{Njk}^{n+1}, \quad u_{1jk}^{n+1} = u_{N+1jk}^{n+1},$$

$$u_{i0k}^{n+1} = u_{iNk}^{n+1}, \quad u_{i1k}^{n+1} = u_{iN+1k}^{n+1},$$

$$u_{ij0}^{n+1} = u_{ijN}^{n+1}, \quad u_{ij1}^{n+1} = u_{ijN+1}^{n+1},$$

$i, j, k = 0,1, \dots, N.$

Для вычисления $u^0, u^1 \in \gamma_h$ допускается использование аналитического решения.

3. Программная реализация

Алгоритм численного решения задачи выглядит следующим образом:

1. Вычислить граничные значения u^0, u^1 используя граничные условия (для периодического условия – значение аналитического решения).
2. Вычислить внутренние значения u^0, u^1.
3. Далее K шагов:
    a. Вычисляем значение u^{n+1} во внутренних узлах сетки.
    b. Вычисляем граничные значения u^{n+1} используя граничные условия (для периодического условия пользуемся разностной аппроксимацией и считаем значение u_{ijN}^{n+1}), зная $u_{ijN+1}^{n+1} = u_{ij1}^{n+1}$.

Общая идея параллельных версий следующая. Сетка разбивается на блоки, затем каждый процесс считает значения в своем блоке и обменивается данными с соседями.

Для параллельных версий используется блочное 3d разбиение (2xN или 4xN), так как оно позволяет уменьшить число коммуникаций между процессорами. Разбиение осуществляется за счет создания декартовой топологии через MPI_Cart_create.

Оценка корректности осуществлялась за счет сравнения максимальной погрешности сетки, так как алгоритм не рандомизированный, то эта погрешность должна быть постоянной и равной значению на последовательной версии.

4. Исследование масштабируемости программы

Запуски проводились на Polus для $L_x = L_y = L_z = L$, $T = 2$, $K = 10000$ на 20 шагах по времени. Результаты исследования и графики решений представлены ниже.

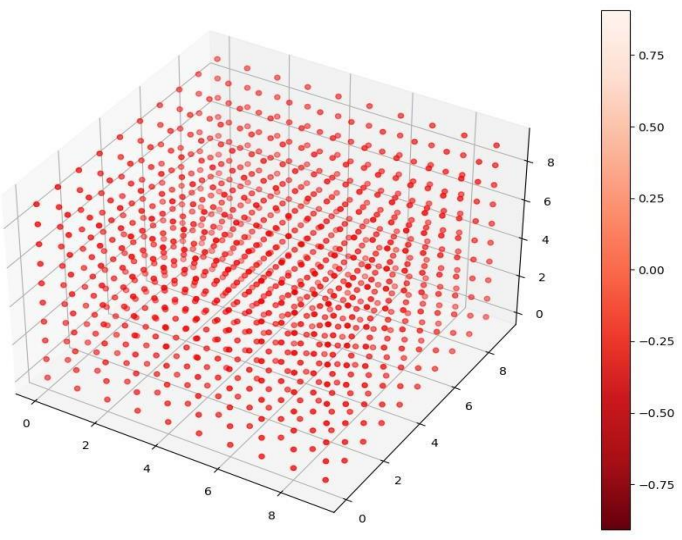

Рисунок 1. График аналитической функции при $L = 1$ на 20 шаге по времени на сетке 10^3.

Число OpenMP нитей	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
2	128^3	0,48	1,9	$7*10^{-8}$
4	128^3	0,25	3,72	$7*10^{-8}$
8	128^3	0,19	4,89	$7*10^{-8}$
16	128^3	0,18	5,1	$7*10^{-8}$
4	256^3	2	3,8	$2*10^{-8}$
8	256^3	1	7,5	$2*10^{-8}$
16	256^3	0,84	8,9	$2*10^{-8}$
32	256^3	0,62	12	$2*10^{-8}$

Таблица 1. Результаты исследования OpenMP программы при $L = 1$

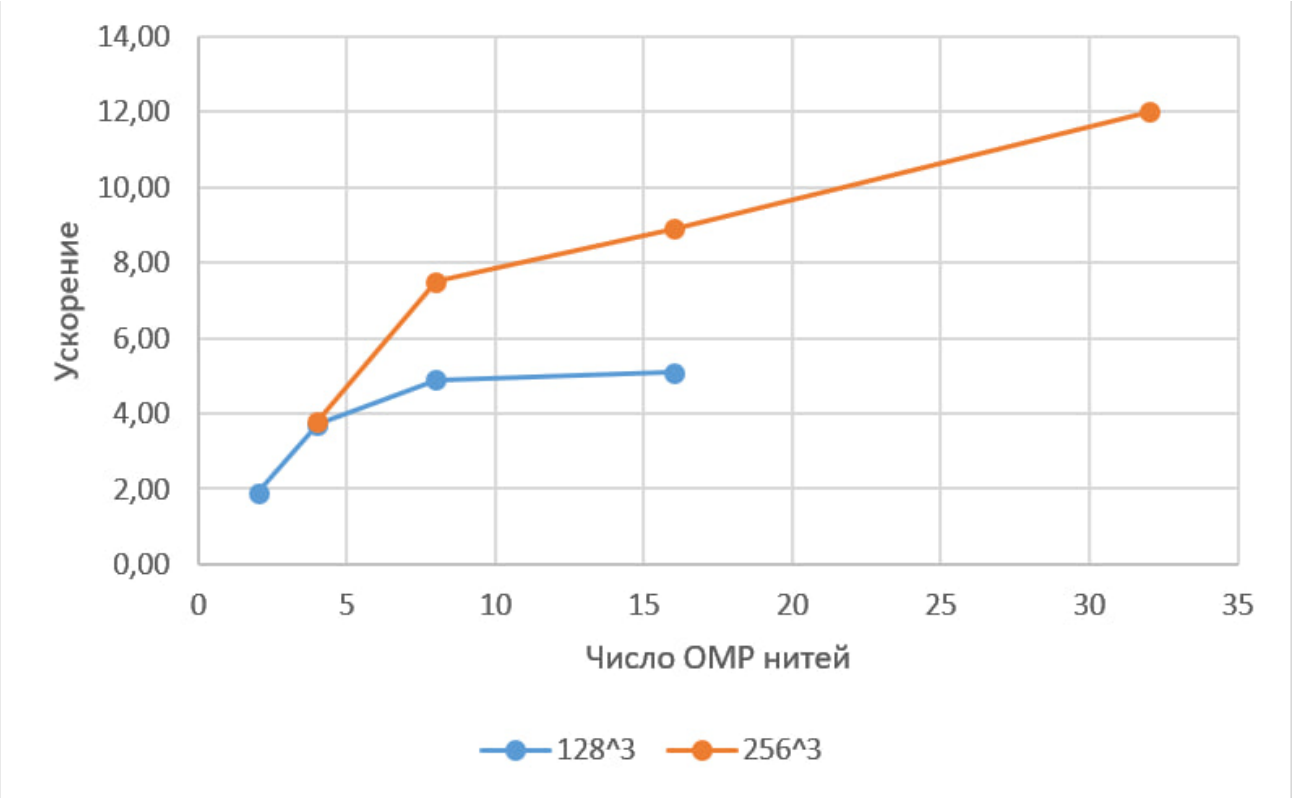

Рисунок 2. График ускорения OpenMP программы при $L = 1$.

Число OpenMP нитей	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
2	128^3	0,47	2	$7*10^{-9}$
4	128^3	0,23	4,1	$7*10^{-9}$
8	128^3	0,18	5,2	$7*10^{-9}$
16	128^3	0,17	5,5	$7*10^{-9}$
4	256^3	1,9	3,6	$2*10^{-9}$
8	256^3	1,1	6,2	$2*10^{-9}$
16	256^3	0,89	7,8	$2*10^{-9}$
32	256^3	0,64	11	$2*10^{-9}$

Таблица 1. Результаты исследования OpenMP программы при $L = \pi$

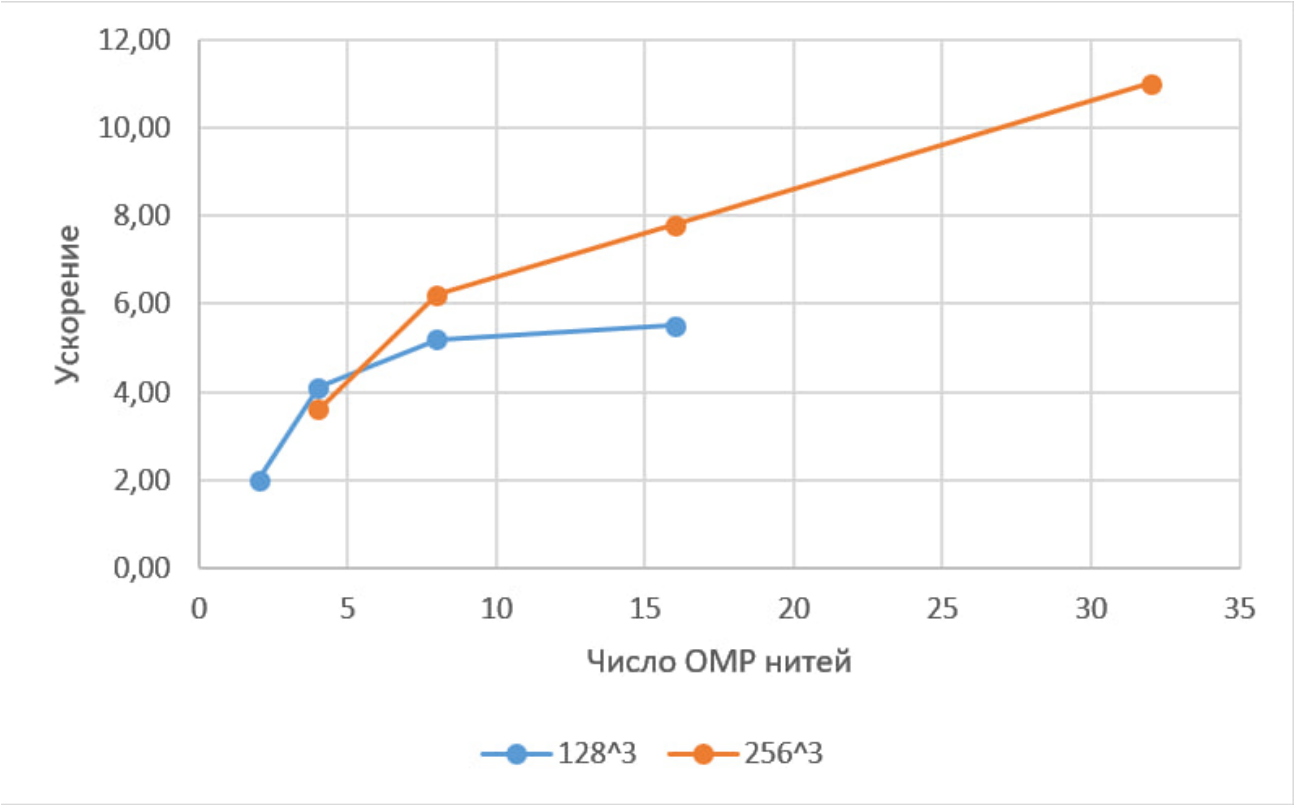

Рисунок 5. График ускорения OpenMP программы при $L = \pi$.

Число MPI процессов N_p	Число OpenMP нитей в процессе	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
2	1	128^3	1,9	1	$7*10^{-9}$
2	2	128^3	0,63	3,01	$7*10^{-9}$
2	4	128^3	0,45	4,2	$7*10^{-9}$
2	8	128^3	0,26	7,3	$7*10^{-9}$
4	1	256^3	2,32	3,23	$7*10^{-8}$
4	2	256^3	1,26	6	$7*10^{-8}$
4	4	256^3	0,7	10,7	$7*10^{-8}$
4	8	256^3	0,69	10,9	$7*10^{-8}$

Таблица 3. Результаты исследования MPI/OpenMP программы при $L = 1$.

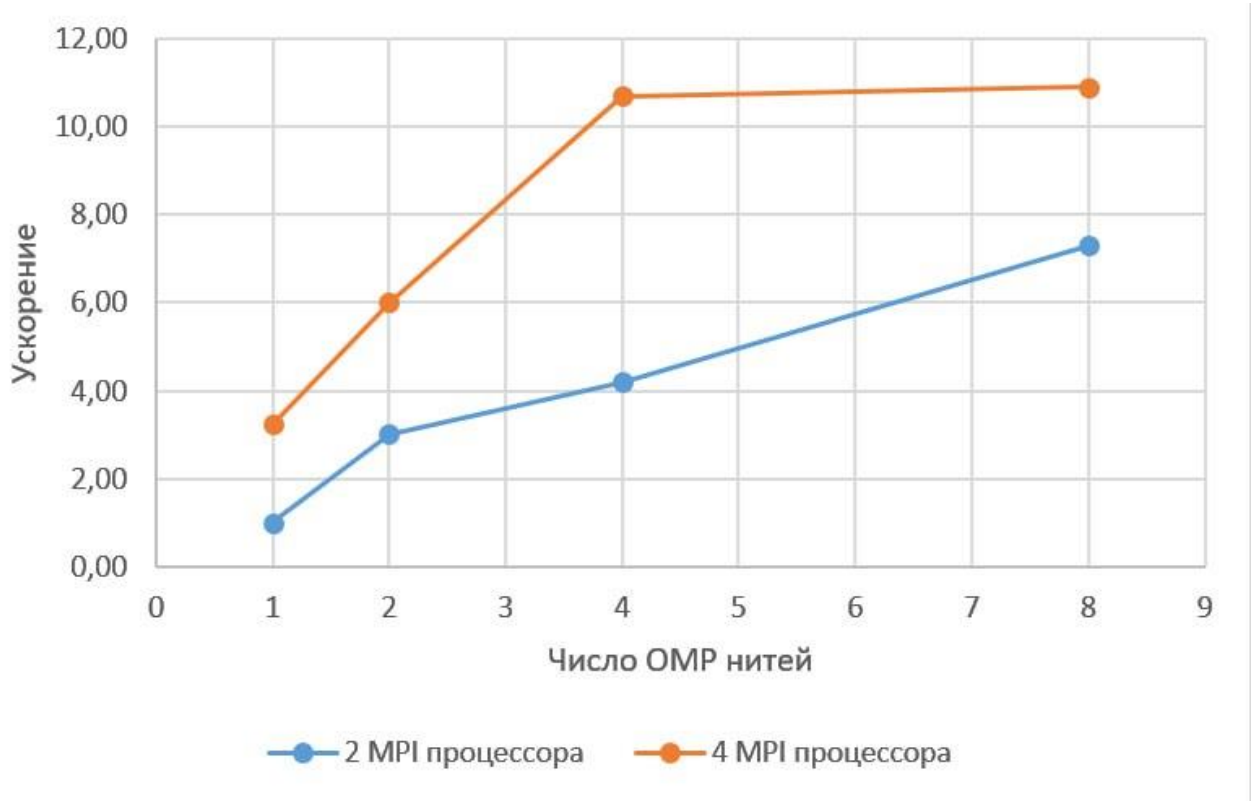

Рисунок 6. График ускорения MPI/OpenMP программы при $L = 1$.

Число MPI процессов N_p	Число OpenMP нитей в процессе	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
2	1	128^3	0,82	2,3	$9*10^{-9}$
2	2	128^3	0,48	4	$9*10^{-9}$
2	4	128^3	0,38	4,9	$9*10^{-9}$
2	8	128^3	0,2	9,4	$9*10^{-9}$
4	1	256^3	2,37	2,9	$2*10^{-9}$
4	2	256^3	1,35	5,1	$2*10^{-9}$
4	4	256^3	0,67	10,3	$2*10^{-9}$
4	8	256^3	0,6	11,5	$2*10^{-9}$

Таблица 3. Результаты исследования MPI/OpenMP программы при $L = \pi$.

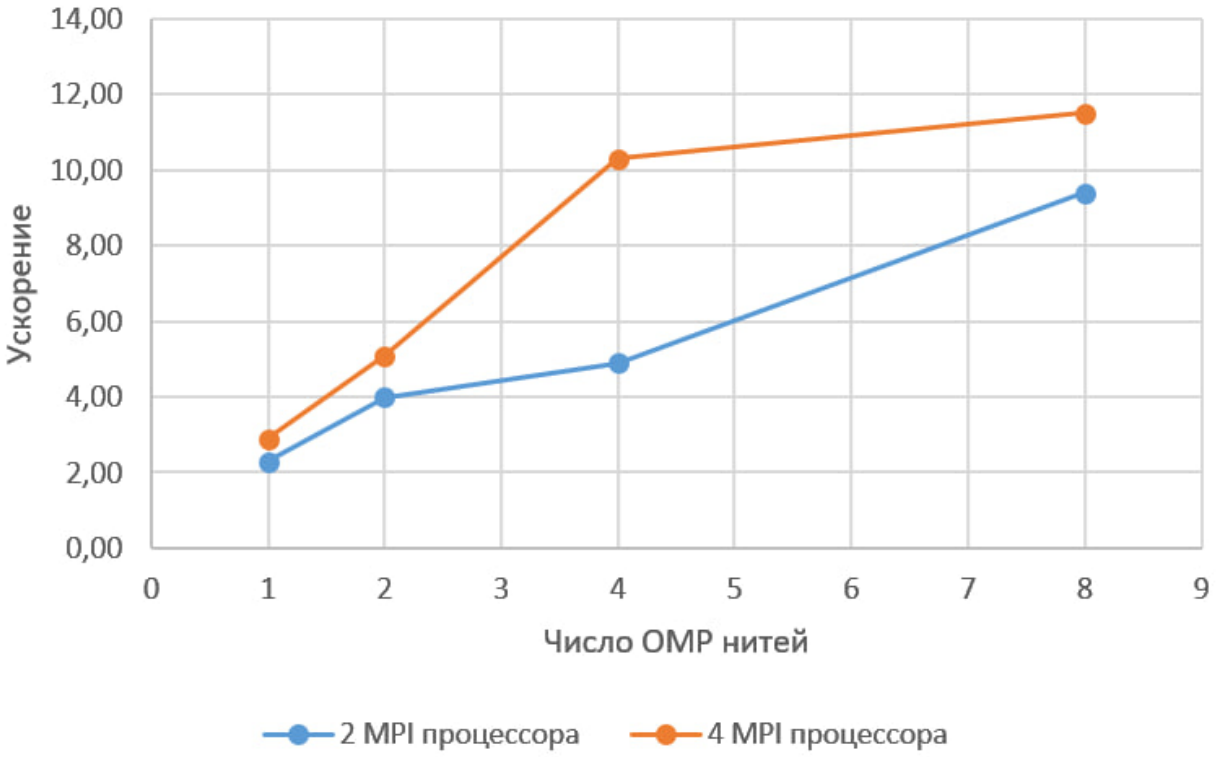

Рисунок 7. График ускорения MPI/OpenMP программы при $L = \pi$.

5. Выводы

В ходе исследования были полученные результаты:

1) Все 2 варианта программы решают задачу примерно за одинаковое время. Различия в программах – в количестве используемых ресурсов.
2) OpenMP довольно просто и быстро позволяет ускорить программу.
3) MPI/OpenMP показывает отличные результаты, однако быстро упирается в пропускную способность кеша и начинает часто ловить cachemiss, от чего эффективность ускорения падает.