Московский Государственный Университет имени М.В. Ломоносова Факультет вычислительной математики и кибернетики

Отчёт по заданию курса «Суперкомпьютерное моделирование и технологии»

Шастун Екатерина Алексеевна, 628 группа

Москва, 2022

Постановка задачи

В рамках задания предлагалось произвести вычисление численного решения трехмерного гиперболического дифференциального уравнения в области трехмерного параллелепипеда.

В трехмерной замкнутой области:

$$\Omega = [0 \le x \le L_x] \times [0 \le y \le L_y] \times [0 \le z \le L_z],$$

для $(0 \le t \le T]$ требуется найти решение $u(z, y, z, t)$ уравнения в частных производных

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

с начальными условиями

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

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

с граничными условиями

$u(0, y, z, t) = u(L_x, y, z, t),\ u_x(0, y, z, t) = u_x(L_x, y, z, t),$

$u(x, 0, z, t) = u(x, L_y, z, t),\ u_y(x, 0, z, t) = u_y(x, L_y, z, t),$

$u(x, y, 0, t) = 0.$

Аналитическое решение определялось вариантом задания:

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

$$a_t = \pi \sqrt{\frac{4}{L_x} + \frac{4}{L_y} + \frac{1}{L_z}}$$

Численное решение

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

$T = T_0$

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

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

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

В программной реализации использованы значения $T_0 = 1, L_{x_0} = L_{y_0} = L_{z_0}$ и равны 1 или $\pi, K = 1000$, но n изменяется от 0 до 20.

Для аппроксимации исходного уравнения используем следующую формулу:

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

Где Δ_h-- семиточечный оператор Лапласа:

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

Эта схема является явной и значения u^{n+1}_{ijk} можно выразить через значения, полученные на предыдущих шагах.

Из начальных условий следует, что

$$u^0_{ijk} = \varphi(x_i, y_j, z_k)$$

и

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

После нахождения u^1_{ijk} и u^0_{ijk} можно найти все остальные значения u^n_{ijk}. Для нахождения значений на границах, используется заданая аналитическая функция.

Программная реализация

Программа использует MPI для параллельного выполнения. В командной строке задаются три значения $L_{x_0}, L_{y_0}, L_{z_0}$.

В программе используются три основных массива:

- значения аналитической функции,
- массивы для хранения u^{n-1} и u^n.

Сетка топологически разбивается между процессами на блоки, и для обеспечения обмена данными между процессами задаются еще 6 массивов, каждый для соответствующей «стороны», где у процесса в топологии есть сосед.

На нулевом временном слое массивы заполняются значениями, равными значению аналитической функции про t=0.

Далее начинается цикл по t. На каждом шаге цикла сначала процессы обмениваются данными, затем в цикле считаются значения u^n_{ijk}: если (x_i, y_j, z_k) лежит на границе, используется аналитическая функция, иначе используется разностная формула.

Каждый процесс считает максимальное значение погрешности и 0-й процесс выводит максимальную из них.

Основные использованные функции MPI:

- MPI_Dims_create, MPI_Cart_create, MPI_Cart_coords – для создания блочной топологии из процессов,
- MPI_Cart_shift – для нахождения соседей процесса в топологии,

- MPI_Send, MPI_Recv, MPI_Sendrecv_replace – для обмена данными между процессами,
- MPI_Reduce – для нахождения максимальных погрешности и времени выполнения.

Также в программе используется технология OpenMP для распараллеливания вложенных циклов, которые проходят по элементам сетки.

Результаты расчетов для системы Polus

Ниже приведены результаты запуска программы на суперкомпьютере на разных числах процессов и с разными размерами сетки с OpenMP и без.

Число MPI-процессов	Число точек сетки	Время выполнения	Ускорение	Погрешность на 20-м временном слое
1	128^3	16.600358	1	0.000879
4	128^3	3.2174885	5.2	0.000879
8	128^3	1.3237355	12.5	0.000879
16	128^3	0.4920925	33.7	0.000879
32	128^3	0.4146275	40	0.000879
1	256^3	180.16595	1	0.000882
4	256^3	41.736934	4.3	0.000882
8	256^3	19.396145	9.3	0.000882
16	256^3	9.4026870	19.2	0.000882
32	256^3	4.5074815	40	0.000882
1	512^3	1544.9943	1	0.000883
4	512^3	358.41677	4.3	0.000883
8	512^3	187.97306	8.2	0.000883
16	512^3	90.555377	17	0.000883
32	512^3	47.386006	32.6	0.000883

Таблица 1. Запуск на Polus без OpenMP для $L_{x_0}, L_{y_0}, L_{z_0}$=1

Число MPI-процессов	Число точек сетки	Время выполнения	Ускорение	Погрешность на 20-м временном слое
1	128^3	16.628236	1	0.000090
4	128^3	2.8737580	5.8	0.000090
8	128^3	1.0955685	15.2	0.000090
16	128^3	0.6635145	25.1	0.000090
32	128^3	0.4428205	37.6	0.000090
1	256^3	176.37192	1	0.000090
4	256^3	42.208623	4.2	0.000090
8	256^3	18.803151	9.4	0.000090

16	256^3	9.4260665	18.7	0.000090
32	256^3	4.7468645	37.2	0.000090
1	512^3	1614.8141	1	0.000090
4	512^3	357.30113	4.5	0.000090
8	512^3	190.65331	8.5	0.000090
16	512^3	91.708120	17.6	0.000090
32	512^3	47.623713	33.9	0.000090

Таблица 2. Запуск на Polus без OpenMP для $L_{x_0}, L_{y_0}, L_{z_0}$=$\pi$

Число MPI-процессов	Число OpenMP нитей в процессе	Число точек сетки	Время выполнения	Ускорение	Погрешность на 20-м временном слое
1	4	128^3	4.7389105	1	0.000879
2	4	128^3	2.8252485	1.7	0.000879
4	4	128^3	1.3836700	3.4	0.000879
8	4	128^3	1.0714195	4.4	0.000879
1	4	256^3	54.900592	1	0.000882
2	4	256^3	29.268371	1.9	0.000882
4	4	256^3	12.711699	4.3	0.000882
8	4	256^3	8.1082370	6.8	0.000882
1	4	512^3	441.22480	1	0.000883
2	4	512^3	222.78346	2	0.000883
4	4	512^3	115.60375	3.8	0.000883
8	4	512^3	59.929599	7.4	0.000883

Таблица 3. Запуск на Polus с OpenMP для $L_{x_0}, L_{y_0}, L_{z_0}$=1

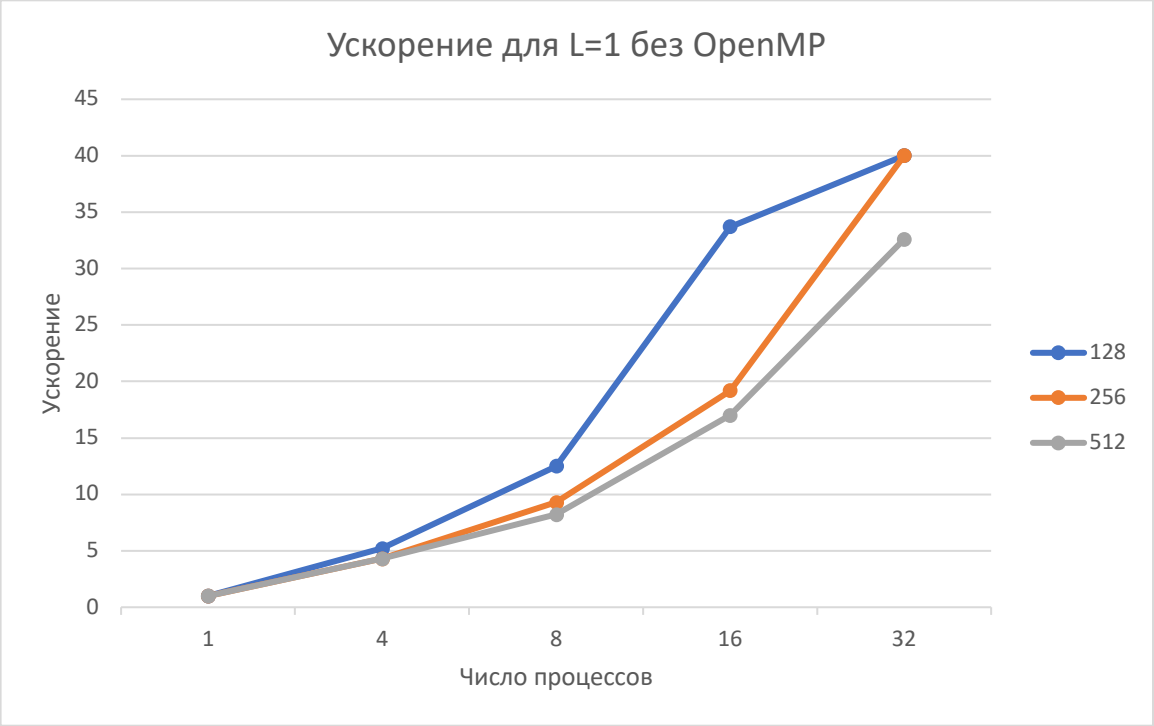
Число MPI-процессов	Число OpenMP нитей в процессе	Число точек сетки	Время выполнения	Ускорение	Погрешность на 20-м временном слое
1	4	128^3	5.3220050	1	0.000879
2	4	128^3	2.7031195	2	0.000879
4	4	128^3	1.0408560	5.1	0.000879
8	4	128^3	1.0893865	4.9	0.000879

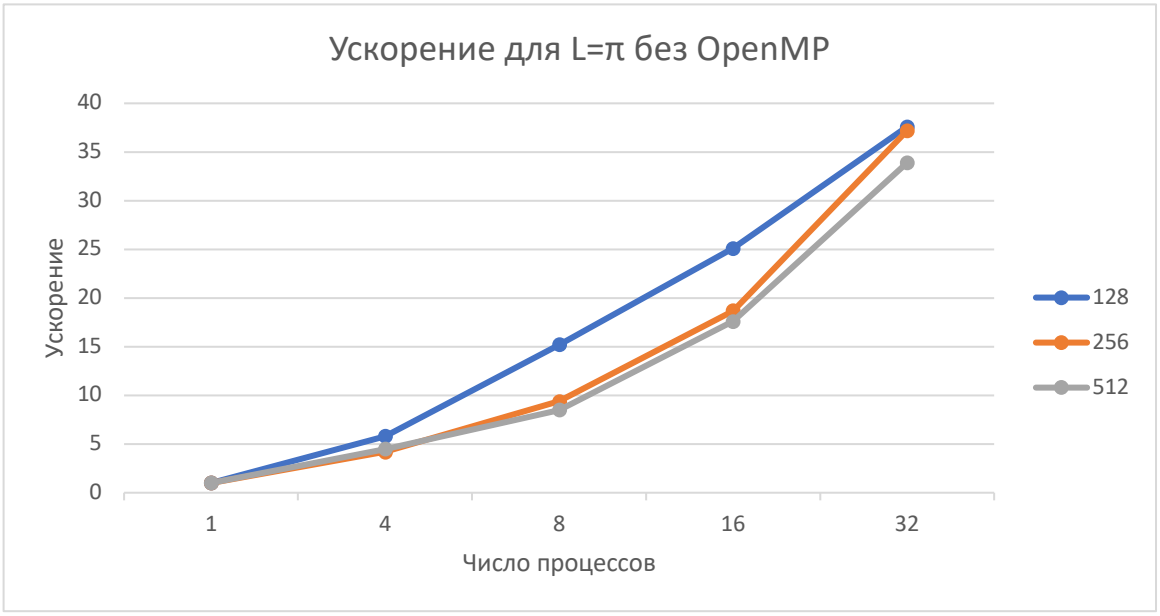
1	4	256^3	54.425716	1	0.000882
2	4	256^3	29.729265	1.8	0.000882
4	4	256^3	12.917617	4.2	0.000882
8	4	256^3	7.9654410	6.8	0.000882
1	4	512^3	401.28209	1	0.000883
2	4	512^3	266.87632	1.5	0.000883
4	4	512^3	119.96107	3.3	0.000883
8	4	512^3	59.513777	6.7	0.000883

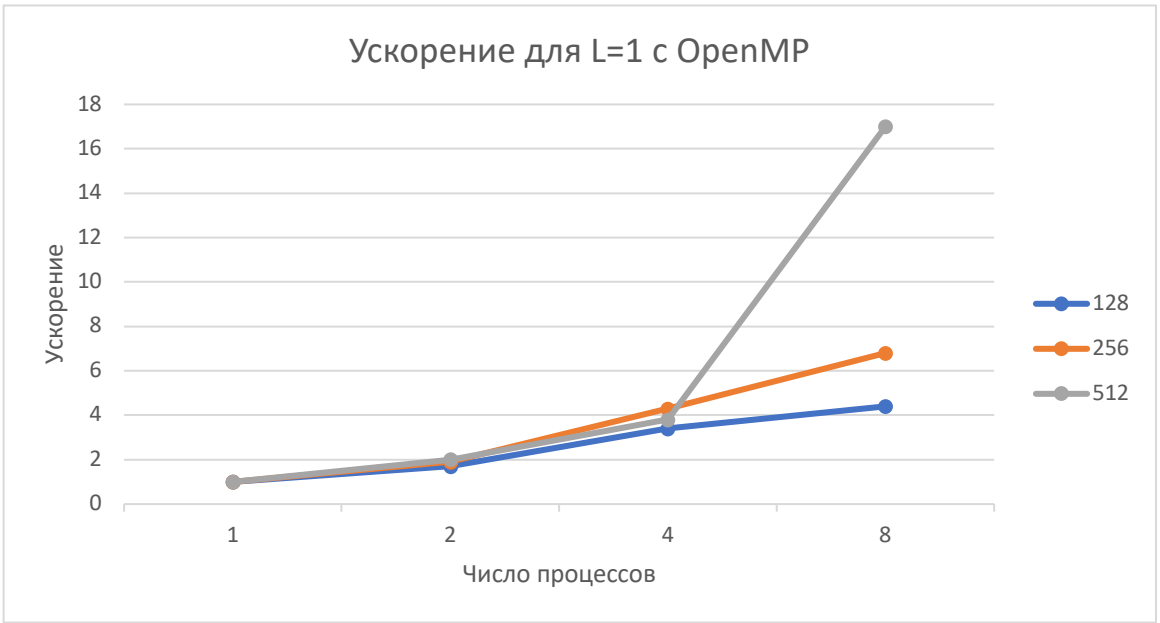
Таблица 4. Запуск на Polus с OpenMP для $L_{x_0}, L_{y_0}, L_{z_0}$=π

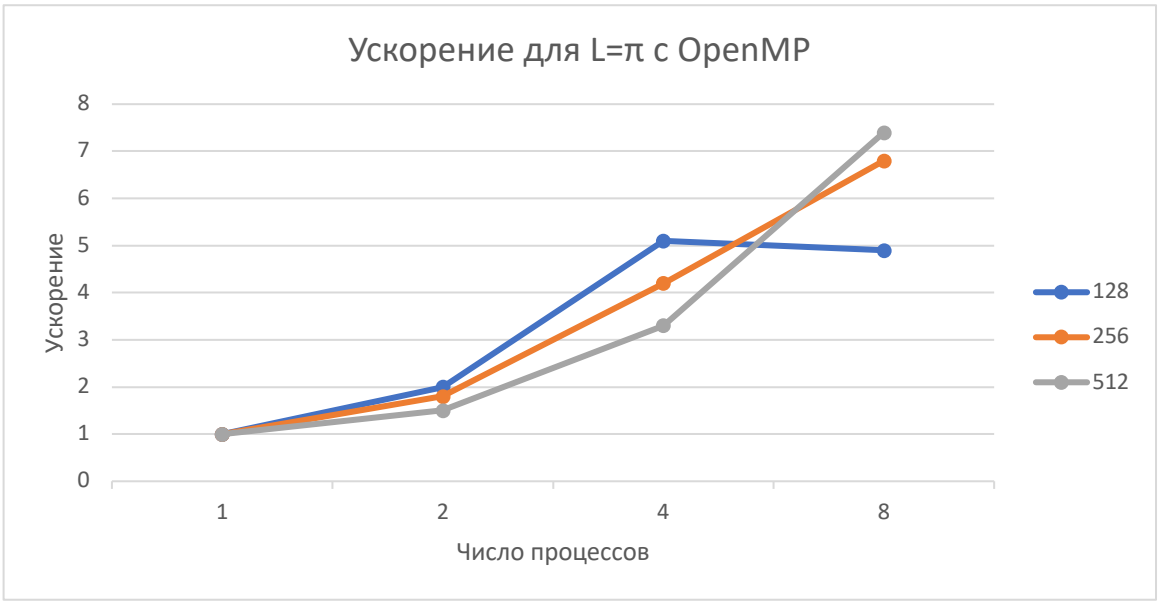
Графики решений и погрешностей можно найти в репозитории в формате .gif. Для простоты визуализации был использован размер сетки 32^3.

Графики ускорений

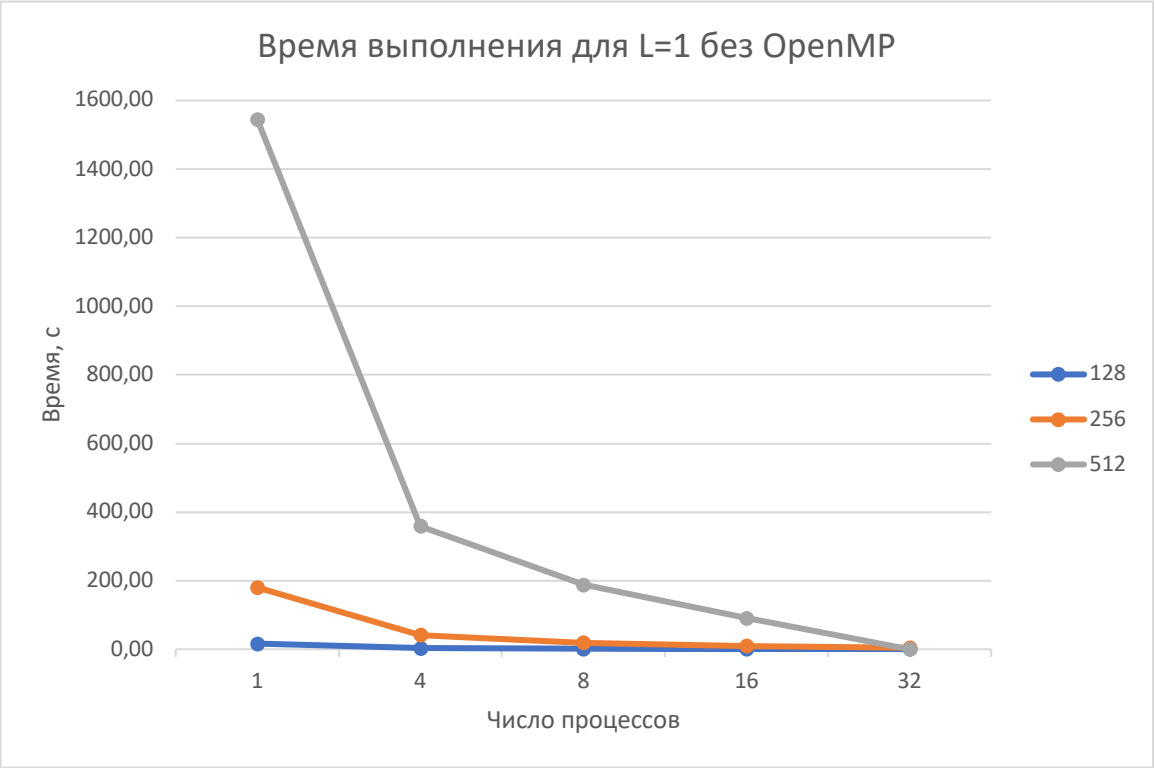


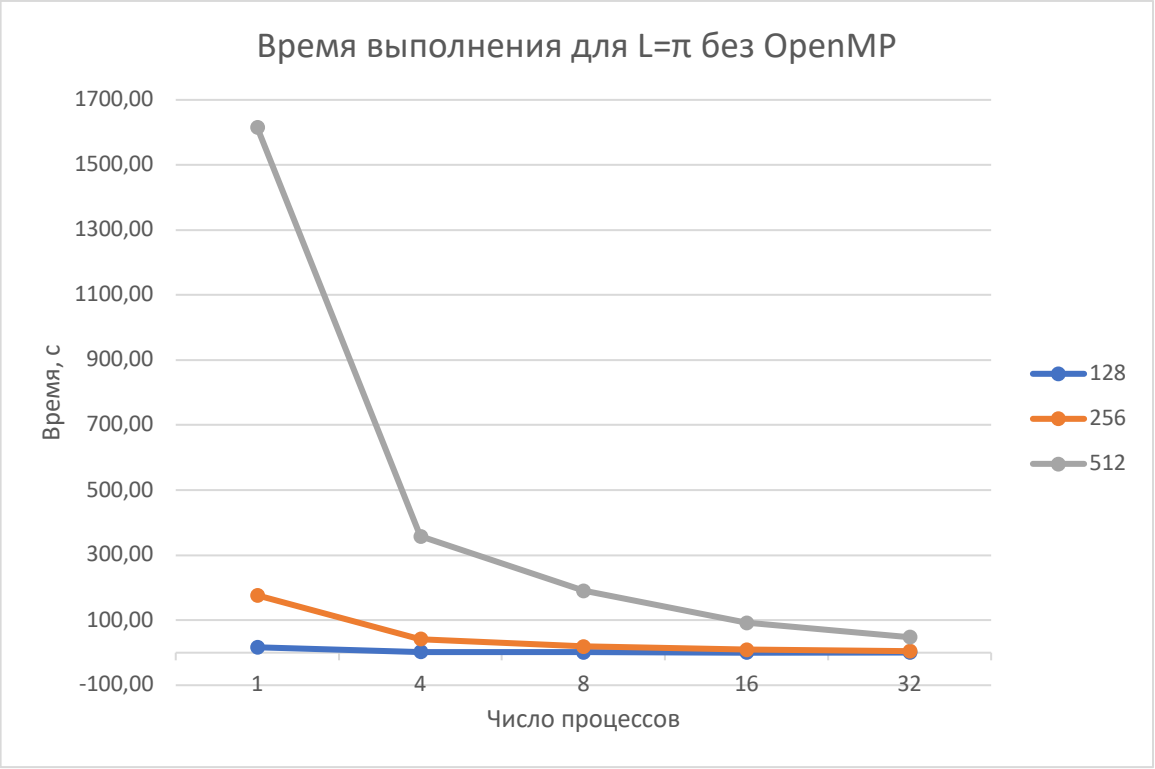

Ускорение для L=π без OpenMP
40
35
30
25
20
15
10
5
0
Ускорение
1
4
8
16
32
Число процессов
128
256
512

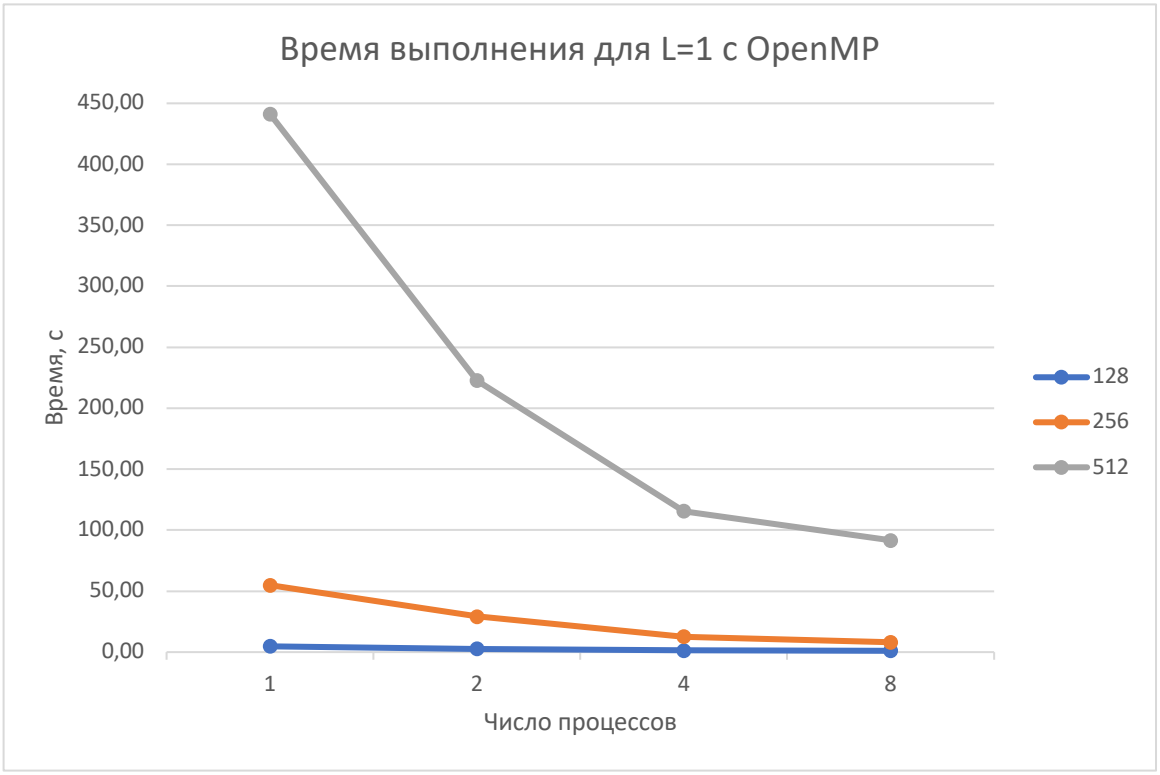

Ускорение для L=1 с OpenMP
18
16
14
12
10
8
6
4
2
0
Ускорение
1
2
4
8
Число процессов
128
256
512

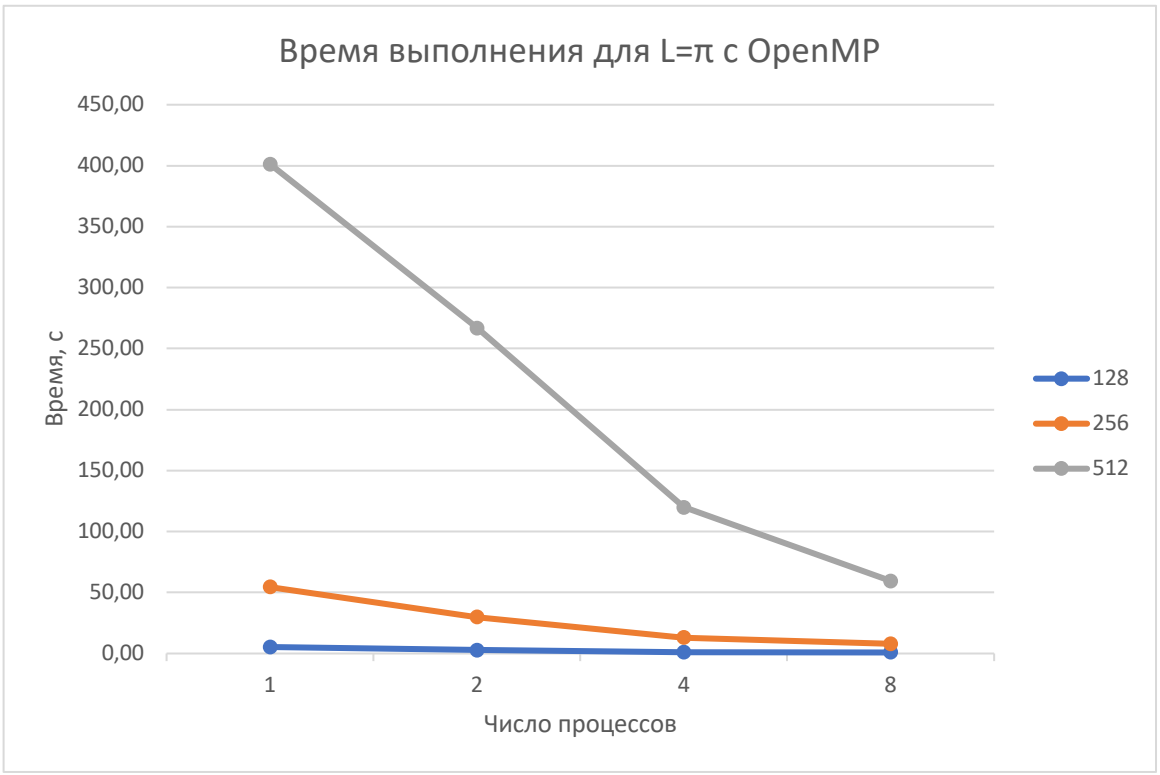

Ускорение для L=π с OpenMP
8
7
6
5
4
3
2
1
0
Ускорение
1
2
4
8
Число процессов
128
256
512

Графики времени выполения






Время выполнения для L=1 с OpenMP
450,00
400,00
350,00
300,00
250,00
200,00
150,00
100,00
50,00
0,00
Время, с
1
2
4
8
Число процессов
128
256
512


Время выполнения для L=π с OpenMP
450,00
400,00
350,00
300,00
250,00
200,00
150,00
100,00
50,00
0,00
Время, с
1
2
4
8
Число процессов
128
256
512

Итог

В результате проделанной работы был программно реализован численный метод нахождения решения дифференциального уравнения и провелось распараллеливание программы с помощью блочной топологии процессов.

Полученные результаты показывают, что с ростом числа процессов, уменьшается время вычисления. При добавлении в MPI-программу распараллеливания циклов с OpenMP, время выполнения уменьшается по сравнению с MPI без OpenMP, особенно на больших размерах сетки. Это говорит о хорошей масштабируемости задачи.