Отчёт по практическому заданию (3) в рамках курса «Суперкомпьютерное моделирование и технологии»

Вариант 1

Никифоров Никита Игоревич, гр. 621

nickiforov.nik@gmail.com

Октябрь 2022

1 Задача

Необходимо реализовать численный метод аппроксимации для трёхмерного гиперболического уравнения в области, представляющей из себя прямоугольный параллелепипед. Для реализации метода предлагается использовать языки программирования C/C++, с использованием библиотеки параллельного вычисления MPI и OpenMP.

Необходимо провести исследование реализованного численного метода для заданного интеграла, области и точности на параллельных вычислительных системах ВМК МГУ: IBM Polus

2 Математическая постановка задачи

В трёхмерной замкнутой области

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

Для $(0 < t \le T]$ найти решение $u(x, y, z, t)$ уравнения в частных производных

$$\frac{\partial^2 u}{\partial t^2} = \Delta u \tag{1}$$

С начальными условиями

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

$$\left.\frac{\partial u}{\partial t}\right|_{t=0} = 0 \tag{3}$$

Граничные условия (вариант 1):

$$u(0, y, z, t) = 0 \qquad u(L_x, y, z, t) = 0 \tag{4}$$

$$u(x, 0, z, t) = 0 \qquad u(x, L_y, z, t) = 0 \tag{5}$$

$$u(x, y, 0, t) = 0 \qquad u(x, y, L_z, t) = 0 \tag{6}$$

Аналитическое решение:

$$u_{analytical}(x, y, z, t) = sin(\frac{\pi}{L_x}x) \cdot sin(\frac{2\pi}{L_y}y) \cdot sin(\frac{3\pi}{L_z}z) \cdot cos(a_y \cdot t), \tag{7}$$

$$a_t = \sqrt{\frac{1}{L_x^2} + \frac{4}{L_y^2} + \frac{9}{L_z^2}} \tag{8}$$

Численный метод решения задачи

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

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

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

Для аппроксимации уравнения воспользуемся равенством:

$$\frac{u_{ijk}^{n+1} - 2u_{ijk}^n + u_{ijk}^{n-1}}{\tau^2} = \Delta_h u^n \tag{9}$$

$$\Delta_h u^n = \frac{u_{i-1,j,k}^n - 2u_{i,j,k}^n + u_{i+1,j,k}^n}{h^2} + \frac{u_{i,j-1,k}^n - 2u_{i,j,k}^n + u_{i,j+1,k}^n}{h^2} + \tag{10}$$

$$+ \frac{u_{i,j,k-1}^n - 2u_{i,j,k}^n + u_{i,j,k+1}^n}{h^2} \tag{11}$$

(Если $L_x = L_y = L_z$, то $h_x = h_y = h_z = h$).

Для начала счёта находим u^0. Из условия (2) получаем:

$$u^0_{ijk} = \varphi(x_i, y_j, z_k), \quad (x_i, y_j, z_k) \in \omega_h. \tag{12}$$

Следующий шаг:

$$\frac{u^1_{ijk} - u^0_{ijk}}{\tau} = \frac{\tau}{2}\Delta_h\varphi(x_i, y_j, z_k) \quad (x_i, y_j, z_k) \in \omega_h \tag{13}$$

$$u^1_{ijk} = u^0_{ijk} + \frac{\tau^2}{2}\Delta_h\varphi(x_i, y_j, z_k) \tag{14}$$

3 Аналитическое решение задачи

$$\varphi(x, y, z, t) = sin(\frac{\pi * x}{L_x})sin(\frac{\pi * y}{L_y})sin(\frac{\pi * z}{L_z})cos(a_t * t) \tag{15}$$

4 Программная реализация

4.1 MPI

Для программной реализации предложенного метода используется язык C++, а также библиотека параллельного программирования MPI и OpenMP.

Для эффективной программной реализации предложенного численного метода предлагается разбить вычисления на трёхмерную сетку, каждый блок из которых вычисляется на своём MPI процессе. Для создания топологии процессов используются функции библиотеки MPI:

- MPI_Dims_create — для автоматического получения размерностей сетки процессов,
- MPI_Cart_create — для создания сетки процессов MPI.

В качестве аргументов функция MPI_Dims_create принимает количество процессов MPI, а возвращает сбалансированную размерность сетки процессов.

В качестве аргументов функция MPI_Cart_create принимает размерность стеки, периодичность граней, а также необходимость размещать близкие по сетке на близких физических ядрах.

В каждом MPI процессе выполняется следующий алгоритм:

1. Вычислить в блоке значения в точках сетки по формулам (8) и (9).
2. Вычислить ошибку аппроксимации как $max_{i,j,k}(|u^n_{xyz} - analitical_solution(x, y, z, t)|)$.
3. Скопировать грани вычисленного блока в непрерывные буферы.
4. Провести обмены с соседними блоками непрерывными буферами.

```
1 void calc_next() {
2     for (int i = start[0]; i < bsize[0] - end[0]; i++) {
3         for (int j = start[1]; j < bsize[1] - end[1]; j++) {
4             for (int k = start[2]; k < bsize[2] - end[2]; k++) {
5                 double lap = func_lap(i, j, k, *data[1]);
6                 (*data[2])(i, j, k) = tau*tau*lap + 2 * (*data[1])(i, j, k) -
7                                       (*data[0])(i, j, k);
```

```
            }
        }
    }
}
```

Из особенностей реализации можно указать необходимость отслеживать глобальные координаты в каждом блоке, так как внутри блока адресация идёт в относительных координатах, поэтому при вызове аналитической функции идёт смешение относительных координат, для получения глобальных координат в сетке.

4.2 OpenMP

Для использования библиотеки `OpenMP` перед каждым циклом прописывается директива `#pragma omp parallel for collapse(N) num_threads(THREADS)`, где:

- `#pragma omp parallel for` — основная часть директивы, которая говорит компилятору, где находится цикл, вычисление которого необходимо исполнить на нескольких потоках.
- `collapse(N)` — указывает вложенность циклов, где `N`, уровень вложенности цикла.
- `num_threads(THREADS)` — указывает количество потоков, которое необходимо использовать. Здесь `THREADS = 4`.

При подсчёте ошибки используется дополнительно директива редукции `reduction(max:error)`

5 Исследование программной реализации

Для исследования использовалась система Polus и домашний компьютер. Характеристики узла системы параллельного вычисления Polus (на данный момент имеет в работе 3 узла):

- 2 десятиядерных процессора IBM POWER8 (каждое ядро имеет 8 потоков) всего 160 потоков
- Общая оперативная память 256 Гбайт (в узле 5 оперативная память 1024 Гбайт) с ECC контролем
- 2 x 1 ТБ 2.5” 7K RPM SATA HDD
- 2 x NVIDIA Tesla P100 GPU, 16Gb, NVLink
- 1 порт 100 ГБ/сек

Необходимо провести экспериментальное исследование для $L_x = L_y = L_z = 1$ и $L_x = L_y = L_z = \pi$. И для разного количества процессов для MPI версии: $cpus = 1, 4, 8, 16, 32$, для OpenMP+MPI: $cpus = 1, 2, 4, 8$. И для различного числа точек в сетке $128^3, 256^3, 512^3$.

Ускорение считалось как отношение общего времени выполнения на одном MPI-процессе ко времени вычисления на заданном количестве MPI-процессов. Составим таблицу для системы Polus:

Таблица 1: Результаты исследования на машине Polus $L_x = L_y = L_z = 1$

Размер сетки	**число MPI процессов**	**Время работы (с)**	**Ускорение по времени**	**Ошибка**
$128 * 128 * 128$	1	2.356270	1	3.32669e-09
$128 * 128 * 128$	4	0.630888	3.73	3.32669e-09
$128 * 128 * 128$	8	0.457156	5.15	3.32669e-09
$128 * 128 * 128$	16	0.402237	5.85	3.32669e-09
$128 * 128 * 128$	32	0.194718	12.10	3.32669e-09
$256 * 256 * 256$	1	18.57330	1	8.24106e-10
$256 * 256 * 256$	4	4.704710	3.94	8.24106e-10
$256 * 256 * 256$	8	2.840980	6.53	8.24106e-10
$256 * 256 * 256$	16	1.571960	11.81	8.24106e-10
$256 * 256 * 256$	32	0.890774	20.85	8.24106e-10
$512 * 512 * 512$	1	150.018	1	2.04022e-10
$512 * 512 * 512$	4	37.8112	3.96	2.04022e-10
$512 * 512 * 512$	8	20.2954	7.39	2.04022e-10
$512 * 512 * 512$	16	10.7692	13.93	2.04022e-10
$512 * 512 * 512$	32	5.54698	27.04	2.04022e-10

Таблица 2: Результаты исследования на машине Polus $L_x = L_y = L_z = \pi$

Размер сетки	**число MPI процессов**	**Время работы (с)**	**Ускорение по времени**	**Ошибка**
$128 * 128 * 128$	1	2.097480	1	0.00423684
$128 * 128 * 128$	4	0.529625	3.96	0.00423684
$128 * 128 * 128$	8	0.319248	6.57	0.00423684
$128 * 128 * 128$	16	0.325514	6.44	0.00423684
$128 * 128 * 128$	32	0.259678	8.07	0.00423684
$256 * 256 * 256$	1	16.13620	1	0.0170563
$256 * 256 * 256$	4	4.044850	3.98	0.0170563
$256 * 256 * 256$	8	2.386630	6.76	0.0170563
$256 * 256 * 256$	16	1.240480	13.00	0.0170563
$256 * 256 * 256$	32	0.715582	22.54	0.0170563
$512 * 512 * 512$	1	130.179	1	0.0667194
$512 * 512 * 512$	4	32.7986	3.96	0.0667194
$512 * 512 * 512$	8	16.7369	7.77	0.0667194
$512 * 512 * 512$	16	8.91785	14.59	0.0667194
$512 * 512 * 512$	32	4.58811	28.37	0.0667194

Таблица 3: Результаты исследования на домашнем ПК $L_x = L_y = L_z = 1$

Размер сетки	**число MPI процессов**	**Время работы (с)**	**Ускорение по времени**	**Ошибка**
$128 * 128 * 128$	1	1.7638	1	3.32669e-09
$128 * 128 * 128$	2	0.284199	6.20	3.32669e-09
$128 * 128 * 128$	4	0.16448	10.72	3.32669e-09
$128 * 128 * 128$	8	0.099784	17.67	3.32669e-09
$256 * 256 * 256$	1	11.6942	1	8.24106e-10
$256 * 256 * 256$	2	2.20329	5.30	8.24106e-10
$256 * 256 * 256$	4	1.12874	10.36	8.24106e-10
$256 * 256 * 256$	8	0.59743	19.57	8.24106e-10
$512 * 512 * 512$	1	40.15	1	2.04022e-10
$512 * 512 * 512$	2	17.6784	2.27	2.04022e-10
$512 * 512 * 512$	4	9.06548	4.42	2.04022e-10
$512 * 512 * 512$	8	4.69384	8.55	2.04022e-10

Таблица 4: Результаты исследования на машине Polus $L_x = L_y = L_z = \pi$

Размер сетки	**число MPI процессов**	**Время работы (с)**	**Ускорение по времени**	**Ошибка**
$128 * 128 * 128$	1	1.85292	1	0.00423684
$128 * 128 * 128$	2	0.280265	6.61	0.00423684
$128 * 128 * 128$	4	0.159591	11.61	0.00423684
$128 * 128 * 128$	8	0.095655	19.37	0.00423684
$256 * 256 * 256$	1	10.8742	1	0.0170563
$256 * 256 * 256$	2	2.15846	5.03	0.0170563
$256 * 256 * 256$	4	1.09877	9.89	0.0170563
$256 * 256 * 256$	8	0.58233	18.67	0.0170563
$512 * 512 * 512$	1	33.6420	1	0.0667194
$512 * 512 * 512$	2	17.3213	1.94	0.0667194
$512 * 512 * 512$	4	8.68217	3.87	0.0667194
$512 * 512 * 512$	8	4.40099	7.64	0.0667194

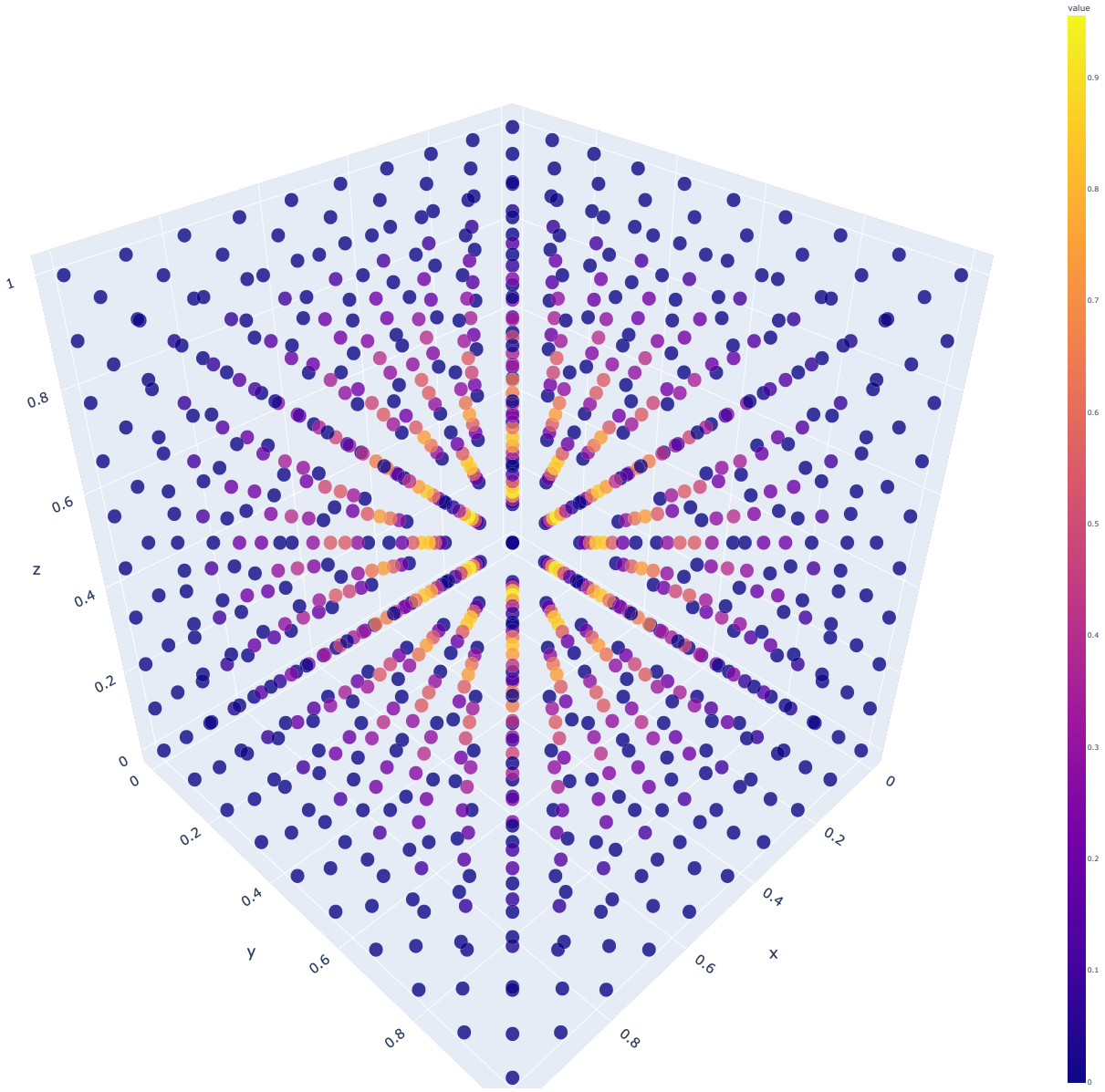


Рис. 1: График аналитического решения $time = 0$, $L_x = L_y = L_z = 1$, сетка 10^3.

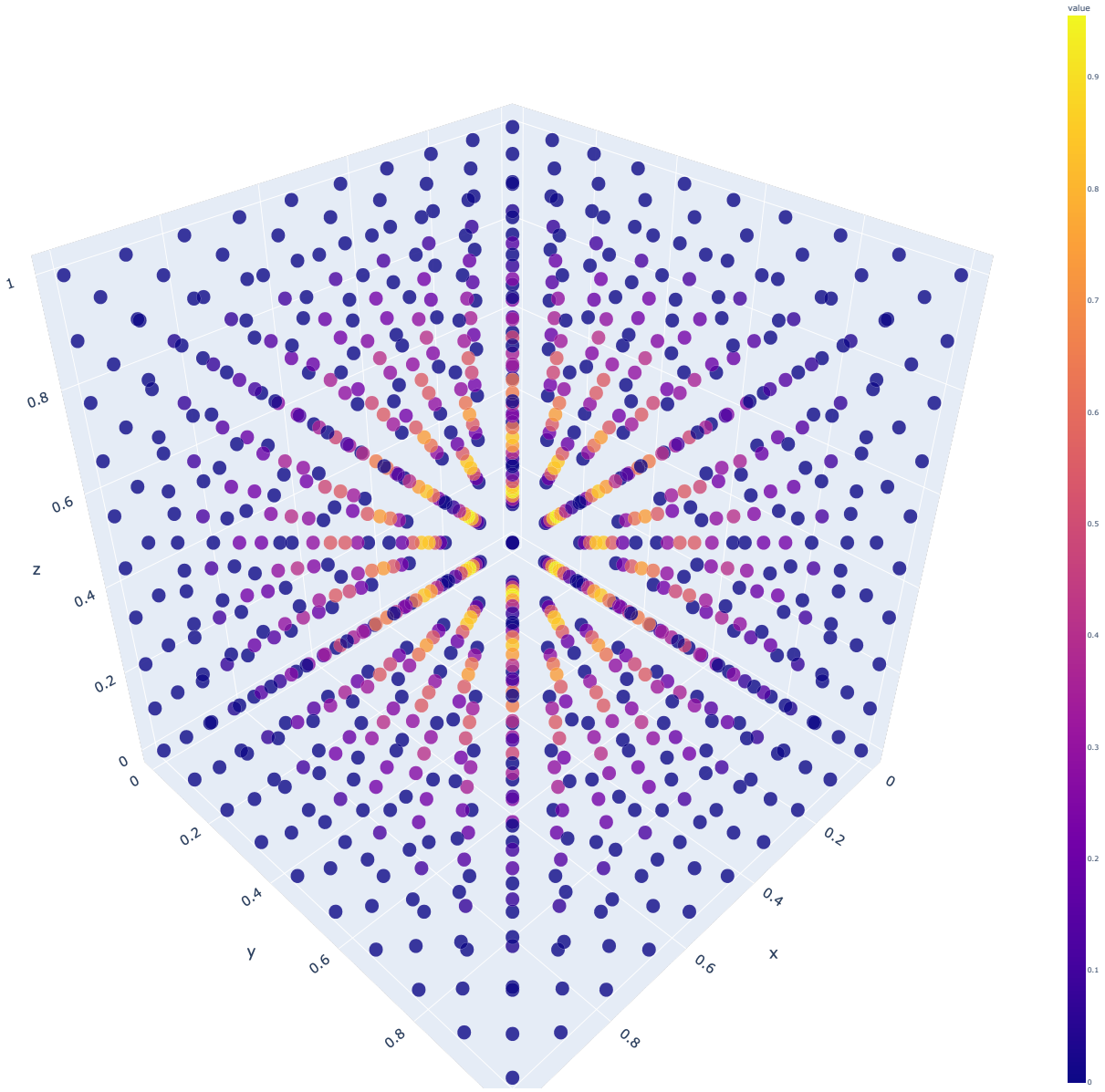


Рис. 2: График посчитанного решения $time = 21$, $L_x = L_y = L_z = 1$, сетка 10^3.

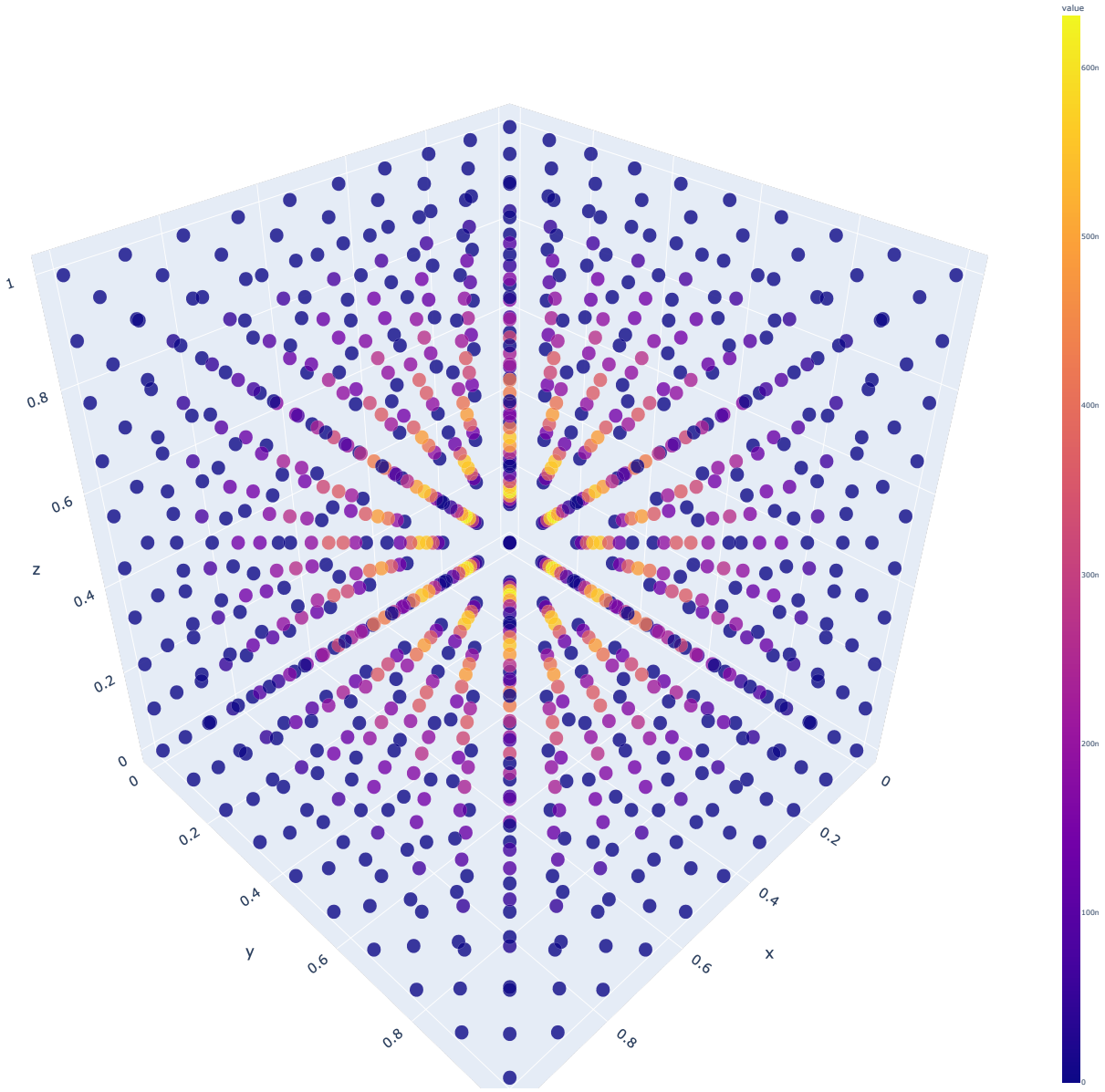
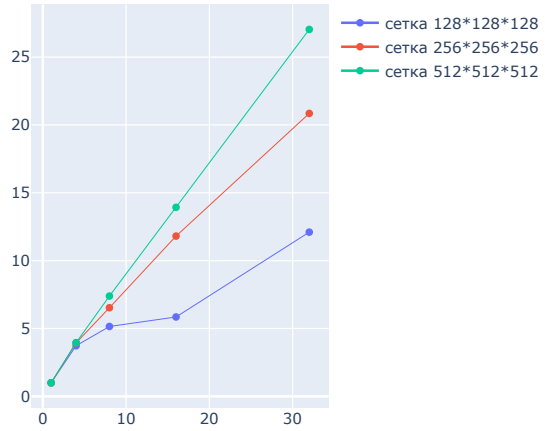
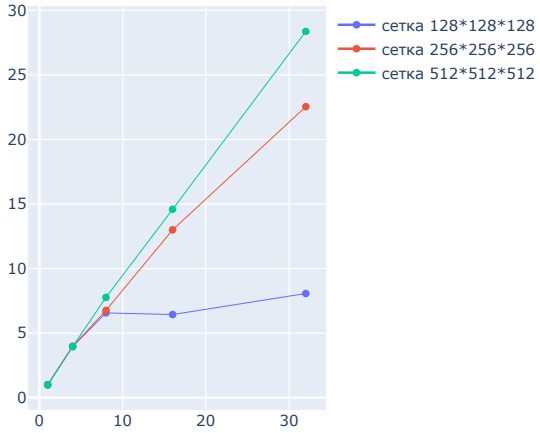


Рис. 3: График погрешности решения $time = 21$, $L_x = L_y = L_z = 1$, сетка 10^3.

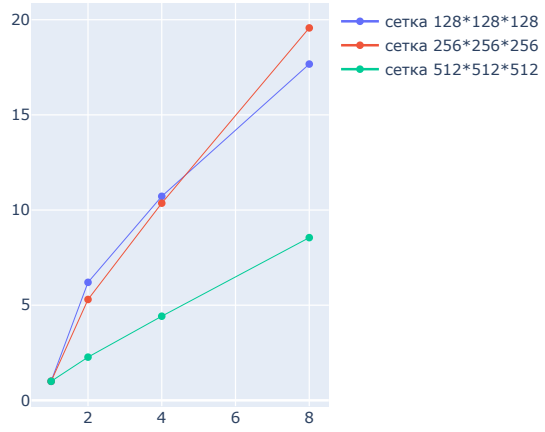


(a) $L_x = L_y = L_z = 1$

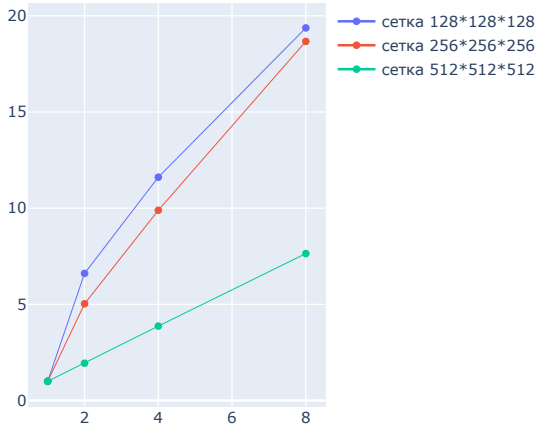


(b) $L_x = L_y = L_z = \pi$

Рис. 4: График зависимости ускорения от числа процессов MPI, на различных сетках



(a) $L_x = L_y = L_z = 1$



(b) $L_x = L_y = L_z = \pi$

Рис. 5: График зависимости ускорения от числа процессов MPI, реализация MPI+OpenMP, на различных сетках