Московский государственный университет имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики

Отчет

По курсу: «Суперкомпьютерное моделирование и технологии»

«MPI»

Цирунов Леонид Александрович

группа 628

27 ноября 2024 г.

Математическая постановка задачи

В трехмерной замкнутой области

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

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

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

С начальными условиями

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

$$\frac{\partial u}{\partial t}|_{t=0} = 0$$

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

$$\begin{aligned} u(0, y,\ z,\ t) &= u(L_x,\ y,\ z,\ t), & u_x(0, y,\ z, t) &= u_x(L_x,\ y,\ z,\ t), \\ u(x,\ 0, z,\ t) &= u(x,\ L_y,\ z,\ t), & u_y(x,\ 0, z, t) &= u_y(x,\ L_y,\ z,\ t), \\ u(x,\ y,\ 0, t) &= u(x,\ y,\ L_z,\ t), & u_z(x,\ y,\ 0, t) &= u_z(x,\ y,\ L_z,\ t), \end{aligned}$$

Численный метод решения задачи

Введем на Ω сетку: $\omega_{h\tau} = \bar{\omega}_h \times \omega_\tau$

$T = T_0$

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

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

$\omega_\tau = \left\{t_n = n\tau,\ n = 0, 1, \ldots, K,\ \tau K = T\right\}$

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

Для аппроксимации исходного уравнения с начальными условиями воспользуемся следующей системой уравнений:

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

Где $\triangle_h$ - семиточечный разностный аналог оператора Лапласа:

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

Для начала счета должны быть заданы значения $u_{i,j,k}^{0}$ и $u_{i,j,k}^{1}$, $(x_i, y_j, z_k) \in \omega_h$.

$$u_{i,j,k}^{0} = \varphi(x_i, y_j,\ z_k), \qquad (x_i, y_j, z_k) \in \omega_h.$$

$$u_{i,j,k}^{1} = u_{i,j,k}^{0} + \frac{\tau^2}{2} \triangle_h \varphi(x_i, y_j,\ z_k)$$

Для вычисления значений u^0 , $u^1 \in \gamma_h$ допускается использование аналитического значения , которое задается в программе еще для вычисления погрешности решения задачи.

Из варианта №8 следуют следующие формулы:

$$u_{analytical} = \sin(\frac{2\pi}{L_x}x)*\sin\left(\frac{4\pi}{L_y}y\right)*\sin\left(\frac{6\pi}{L_z}z\right)*\cos(a_t*t),$$

$$a_t = \pi\sqrt{\left(\frac{4}{{L_x}^2} + \frac{16}{{L_y}^2} + \frac{36}{{L_z}^2}\right)}$$

Граничные условия:

$$\text{П: } u(0, y, z, t) = u(L_x, y, z, t), \quad u_x(0, y, z, t) = u_x(L_x, y, z, t),$$
$$\text{П: } u(x, 0, z, t) = u(x, L_y, z, t), \quad u_y(x, 0, z, t) = u_y(x, L_y, z, t),$$
$$\text{П: } u(x, y, 0, t) = u(x, y, L_z, t), \quad u_z(x, y, 0, t) = u_z(x, y, L_z, t)$$

Алгоритм численного решения:

1. Вычисление граничных значений u^0 и u^1.
2. Вычисление u^0 внутри области: $u^0_{i,j,k} = \varphi(x_i, y_j, z_k)$
3. Вычисление u^1 внутри области: $u^1_{i,j,k} = u^0_{i,j,k} + \frac{\tau^2}{2} \triangle_h \varphi(x_i, y_j, z_k)$
4. Вычисление K – 1 раз u^{n+1}:
   $u^{n+1}_{ijk} = 2u^n_{ijk} - u^{n-1}_{ijk} + \tau^2 \triangle_h u^n$

Программный метод решения задачи

Программная реализация состоит из 4 файлов: ***main.cpp***, ***equation_solution.cpp***, ***equation.h*** и ***Makefile***.

Файл ***main.cpp*** содержит основную функцию, получающую входные значения, производящую инициализацию MPI с созданием декартовой топологии процессов, и запускающую решение задачи, а также две функции для сохранения результатов:

- ***dump_block_to_CSV*** - сохранение погрешности или сетки (для каждого блока), полученной численным или аналитическим способом, в файл в формате CSV (для дальнейшей визуализации);
- ***save_statistics*** - вывод в файл информации о времени работы решения и максимальной погрешности вычисления.

На вход программа получает 3 или 6 значений в зависимости от переданного ***L_type***. Описание параметров в порядке передачи:

- ***N*** - размер сетки по одной координате (конечный размер N^3);
- ***L_type*** - тип значений L по разным координатам, принимает значения: *1*, *pi*, *custom.*

В случае *1* и *pi*, значения L по всем координатам равны числу соответственно. Если ***L_type*** передано значение *custom* , в этом случае необходимо передать значения L по каждой координате: $L_x,\ L_y,\ L_z$.

Файл ***equation.h*** содержит объявления типов: класса сетки с функцией получения индекса элемента в линейном массиве описывающем сетку, класса блока с функцией получения индекса элемента в линейном массиве описывающем блок, а также функцию $u_{analytical}$ для вычисления аналитических значений точек сетки.

Файл ***equation_solution.cpp*** содержит основной алгоритм решения задачи. Содержит функции:

- ***laplace_operator*** - вычисление значения разностного аналога оператора Лапласа;

- ***fill_send_buffers*** - функция, используемая для заполнения буферов для отправки данных соседним процессам.
- ***exchange_ghost_layers*** - функция реализующую пересылку данных между процессами, а также последующее заполнение дополнительного слоя, содержащего данные других процессов(гало).
- ***init*** - вычисление внутренних u^0 , u^1 и граничных начальных точек, необходимых для запуска алгоритма;
- ***run_algo*** - итерация по K – 1 оставшимся временным шагам алгоритма с вычислением значений граничных и внутренних точек на новом шаге алгоритма. На каждом шаге производится подсчет максимальной погрешности численного решения от аналитического с выводом информации в консоль. Также в этом цикле производится заполнение буферов для отправки данных соседним процессам.
- ***solve_equation*** - инициализируются переменные, выполняется запуск алгоритма и ведется подсчет времени алгоритма.

Распараллеливание производится с использованием технологии MPI.

Файл ***Makefile*** содержит цели для компиляции и запуска задач. Для запуска на Polus используются команды «***make compile_polus***» и «***make run_all_polus***».

Результаты расчетов

Таблица 1: Результаты при L = 1

Число MPI процессов	Число точек сетки N по одной оси	Время решения Polus	Время решения локально M3 arm	Ускорение Polus	Ускорение локально	Погреш-ность
0-Sequential	128	8,77943	0,264147	1,000	1,000	1.4004e-06
1	128	9,20052	0,327221	0,9542	0,8072	1.4004e-06
2	128	4,59858	0,182044	1,9092	1,451	1.4004e-06
4	128	2,61351	0,112038	3,3592	2,358	1.4004e-06
8	128	1,57326	0,069210	5,5804	3,817	1.4004e-06
16	128	0,850441	0,086133	10,3234	3,067	1.4004e-06
32	128	0,590741	0,057794	14,8617	4,570	1.4004e-06
0-Sequential	256	62,994	2,175710	1,000	1,000	3.49927e-07
1	256	68,1183	2,622760	0,9248	0,8295	3.49927e-07
2	256	34,5796	1,452760	1,8217	1,4976	3.49927e-07
4	256	19,7141	0,797872	3,1954	2,7269	3.49927e-07
8	256	9,58322	0,440487	6,5734	4,9393	3.49927e-07
16	256	5,64634	0,460547	11,1566	4,7242	3.49927e-07
32	256	3,54626	0,415943	17,7635	5,2308	3.49927e-07
0-Sequential	512	484,384	16,5496	1,000	1,000	8.71479e-08
1	512	508,865	20,6374	0,9519	0,8019	8.71479e-08
2	512	266,067	11,4375	1,8205	1,4470	8.71479e-08
4	512	130,923	6,0540	3,6998	2,7337	8.71479e-08
8	512	70,7735	3,2353	6,8441	5,1153	8.71479e-08
16	512	38,0006	2,2250	12,7467	7,4379	8.71479e-08
32	512	24,9555	1,9539	19,4099	8,4702	8.71479e-08

Таблица 2: Результаты при L = π

Число MPI процессов	Число точек сетки N по одной оси	Время решения Polus	Время решения локально M3 arm	Ускорение Polus	Ускорение локально	Погреш-ность
0-Sequential	128	8,47777	0,264720	1,000	1,000	1.41974e-07
1	128	9,07345	0,321982	0,9343	0,8222	1.41974e-07
2	128	4,55414	0,182116	1,8616	1,4536	1.41974e-07
4	128	2,49134	0,114308	3,4029	2,3158	1.41974e-07
8	128	1,52365	0,072591	5,5641	3,6467	1.41974e-07
16	128	0,857452	0,100738	9,8872	2,6278	1.41974e-07
32	128	0,590185	0,053644	14,3646	4,9348	1.41974e-07
0-Sequential	256	62,5438	2,1346	1,000	1,000	3.55074e-08
1	256	67,1549	2,6469	0,9313	0,8065	3.55074e-08
2	256	35,4423	1,4467	1,7647	1,4755	3.55074e-08
4	256	17,3933	0,7946	3,5959	2,6865	3.55074e-08
8	256	9,54596	0,4305	6,5519	4,9589	3.55074e-08
16	256	6,15678	0,4677	10,1585	4,5639	3.55074e-08
32	256	3,44088	0,4015	18,1767	5,3166	3.55074e-08
0-Sequential	512	480,591	16,8837	1,000	1,000	8.8744e-09
1	512	512,18	21,0584	0,9383	0,8018	8.8744e-09
2	512	265,867	11,4206	1,8076	1,4784	8.8744e-09
4	512	137,52	6,0383	3,4947	2,7961	8.8744e-09
8	512	70,6636	3,2270	6,8011	5,2320	8.8744e-09
16	512	35,8618	2,1158	13,4012	7,9800	8.8744e-09
32	512	24,9462	1,9288	19,2651	8,7535	8.8744e-09

График 1: зависимость ускорения на Polus от количества процессов при L = 1:

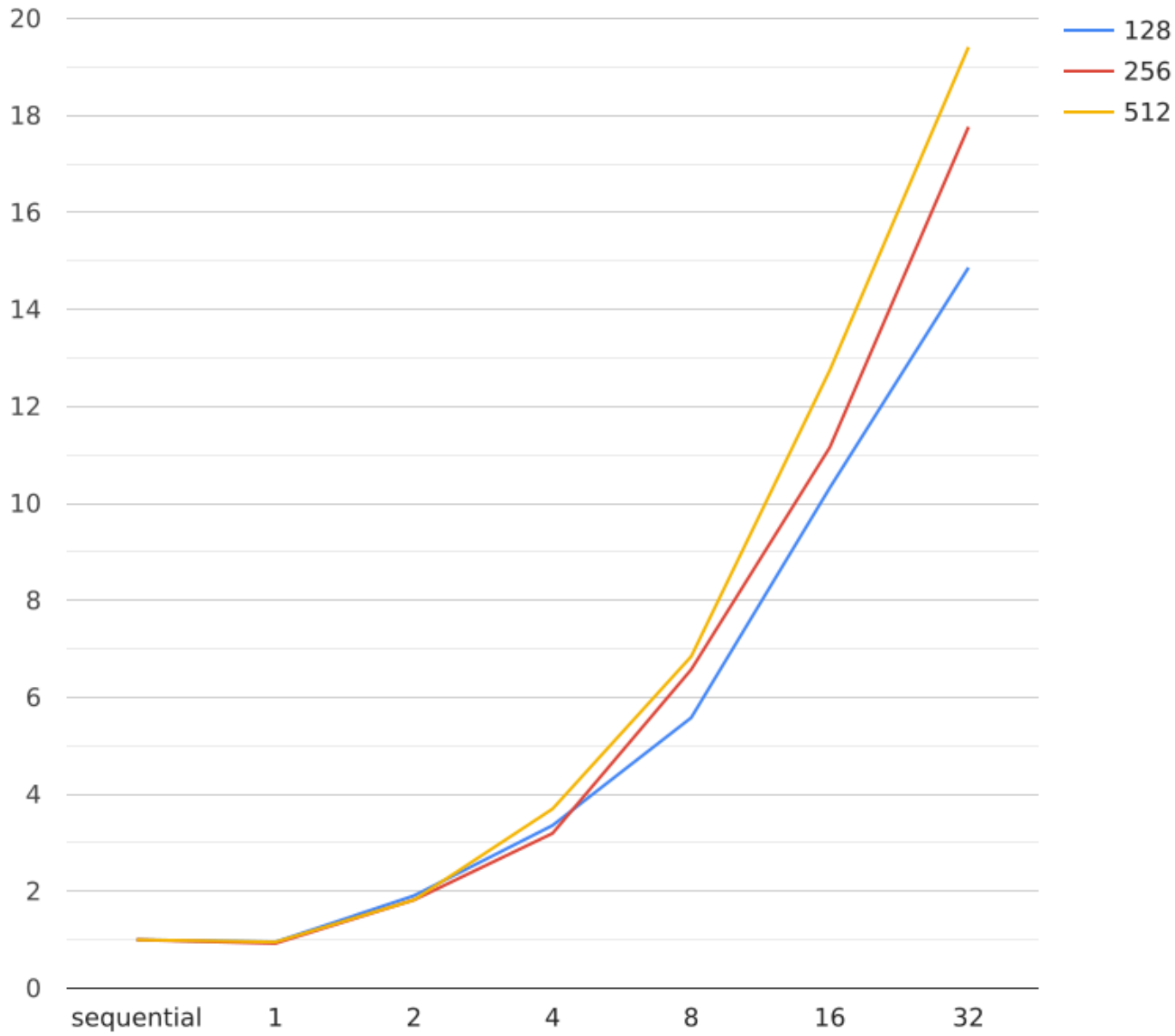


График 2: зависимость ускорения на Polus от количества процессов при L = π:

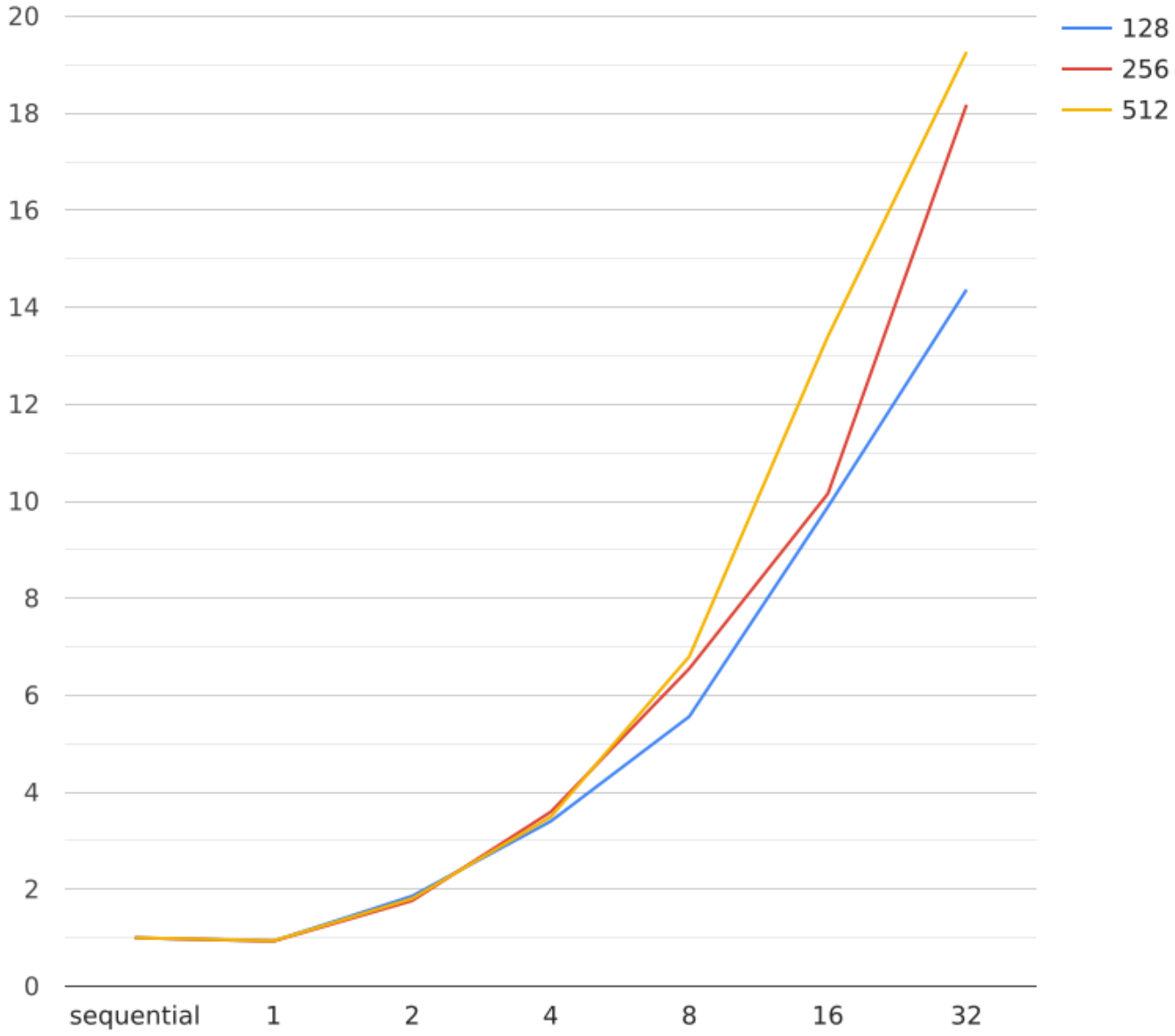


График 3: зависимость ускорения локально на M3 arm от количества процессов при L = 1:

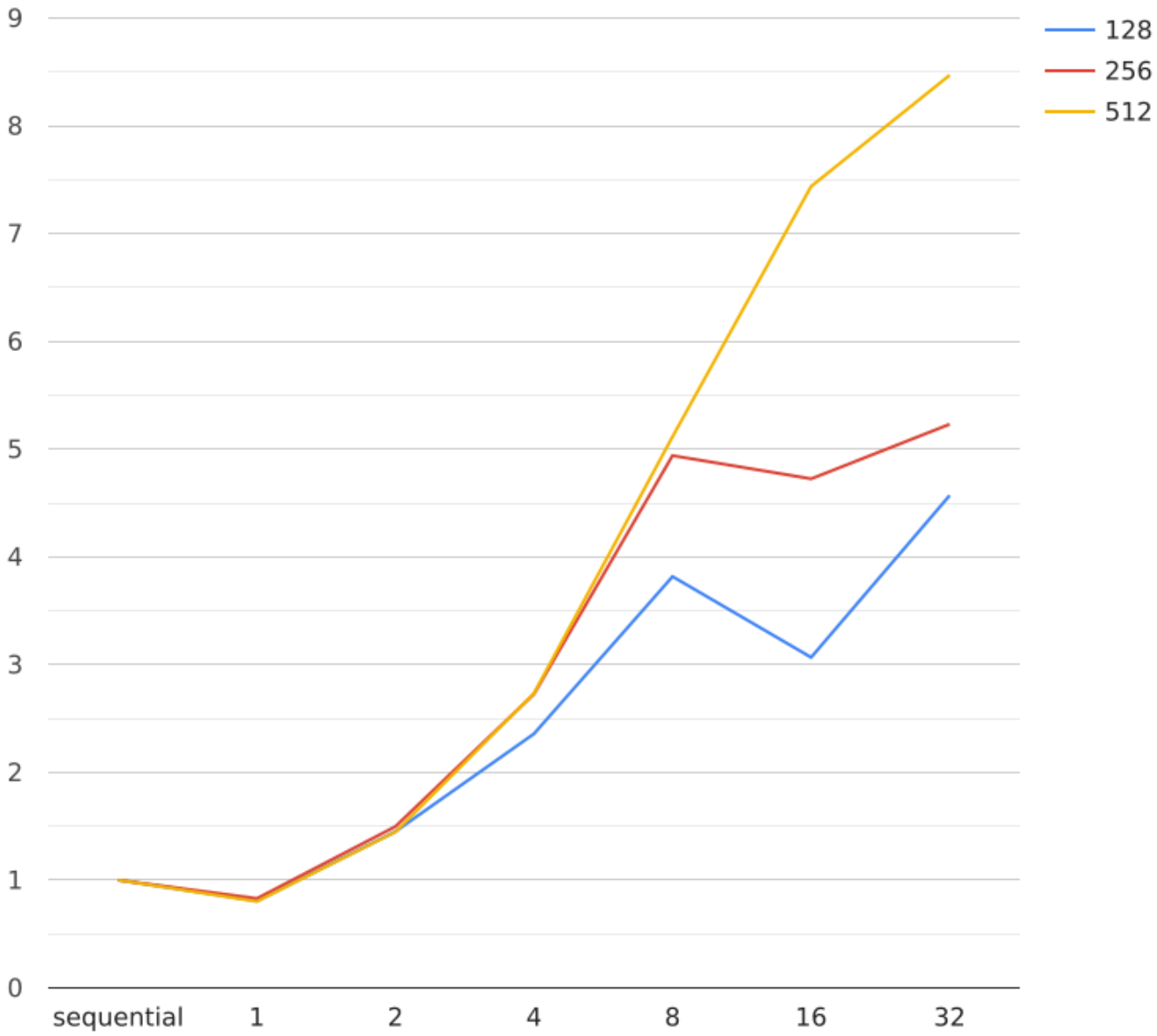
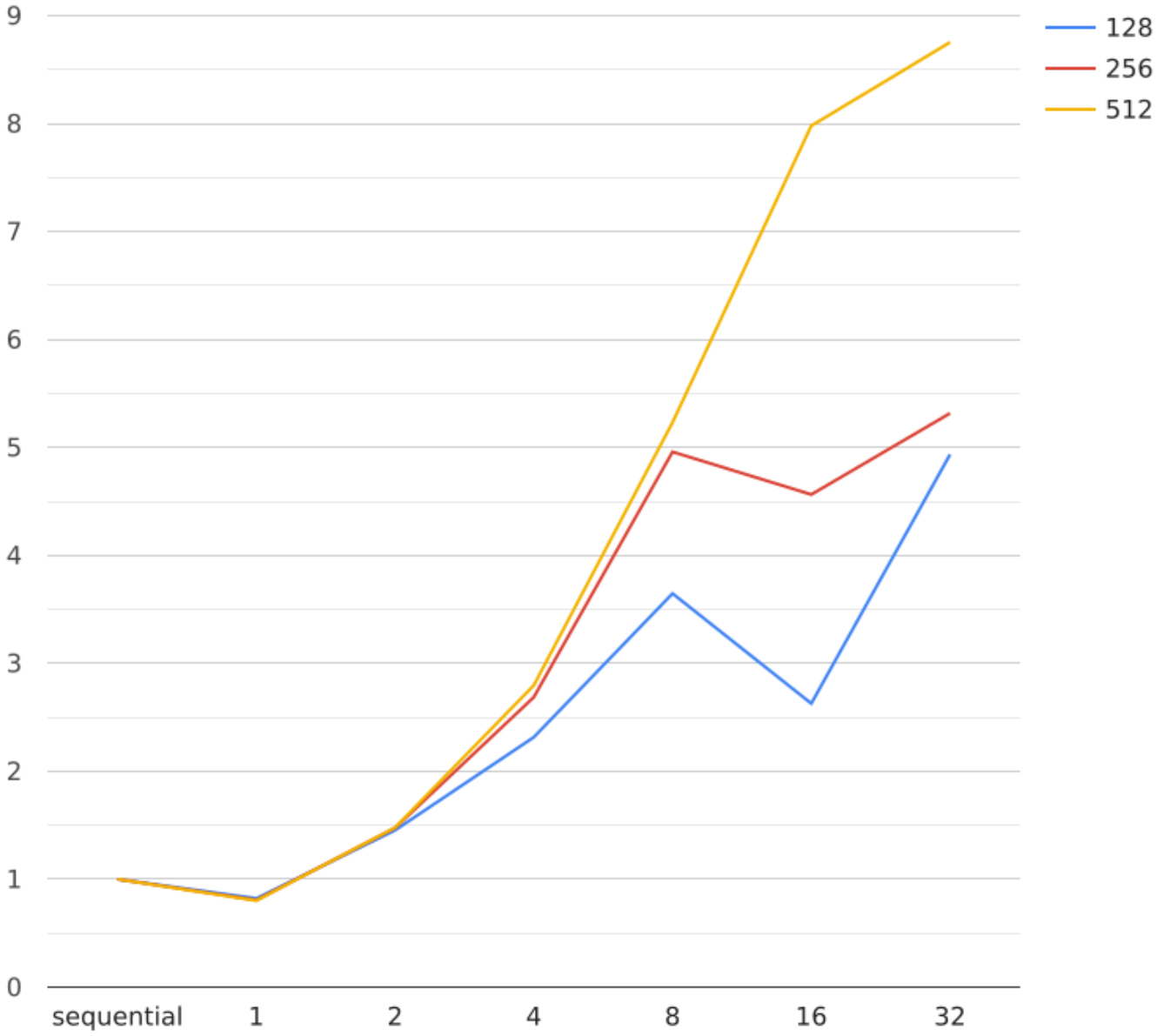
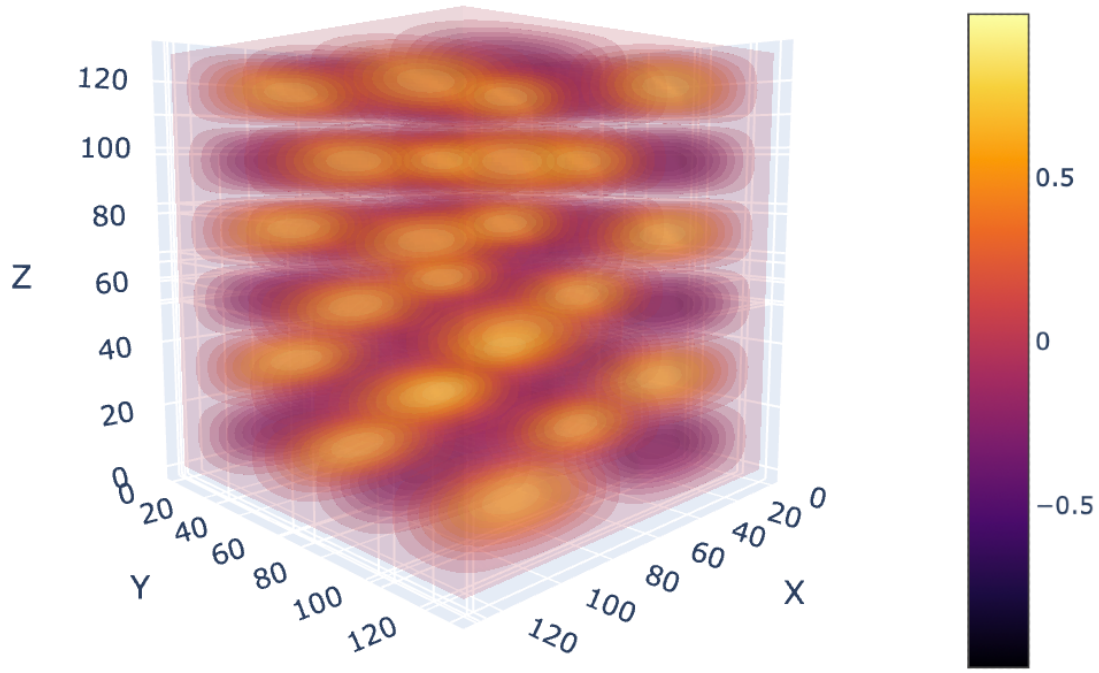
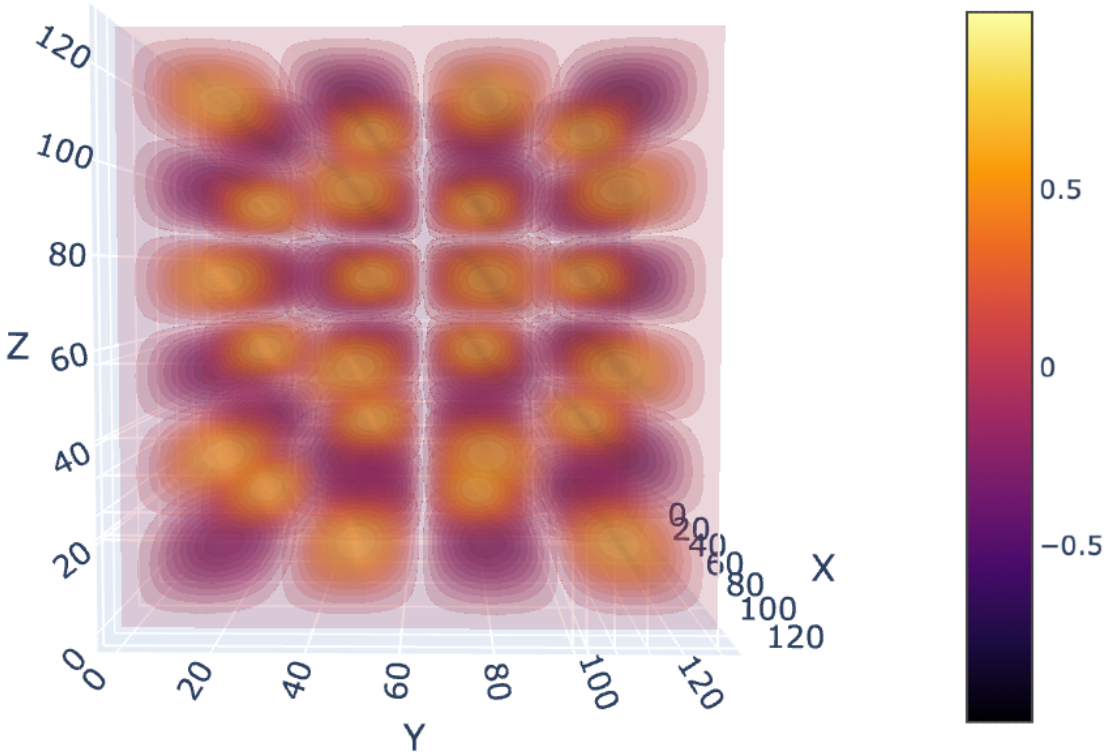


График 4: зависимость ускорения локально на M3 arm от количества процессов при L = π:

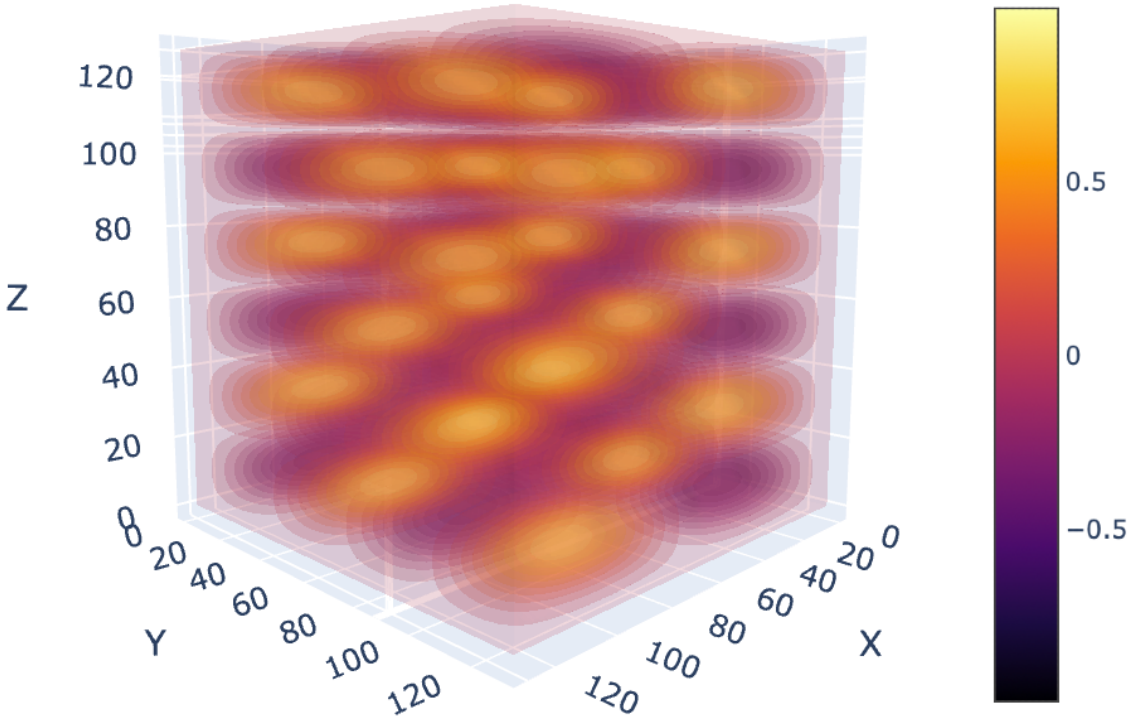


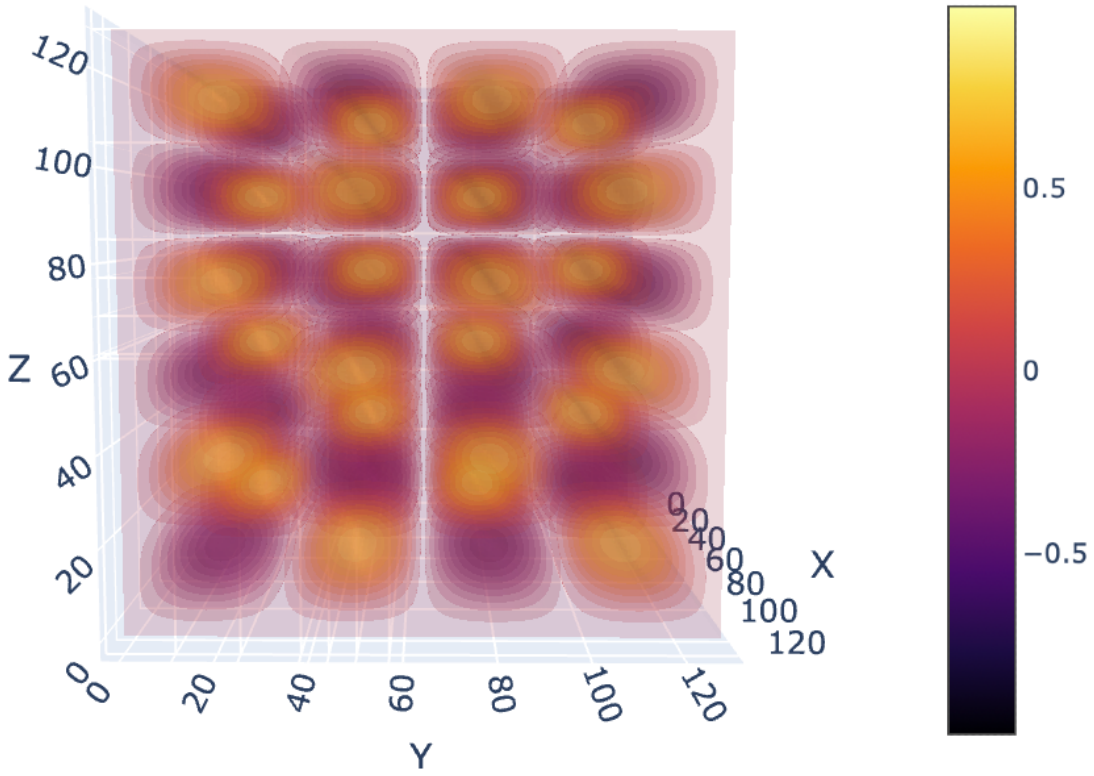
Визуализация сетки, полученной аналитическим решением:



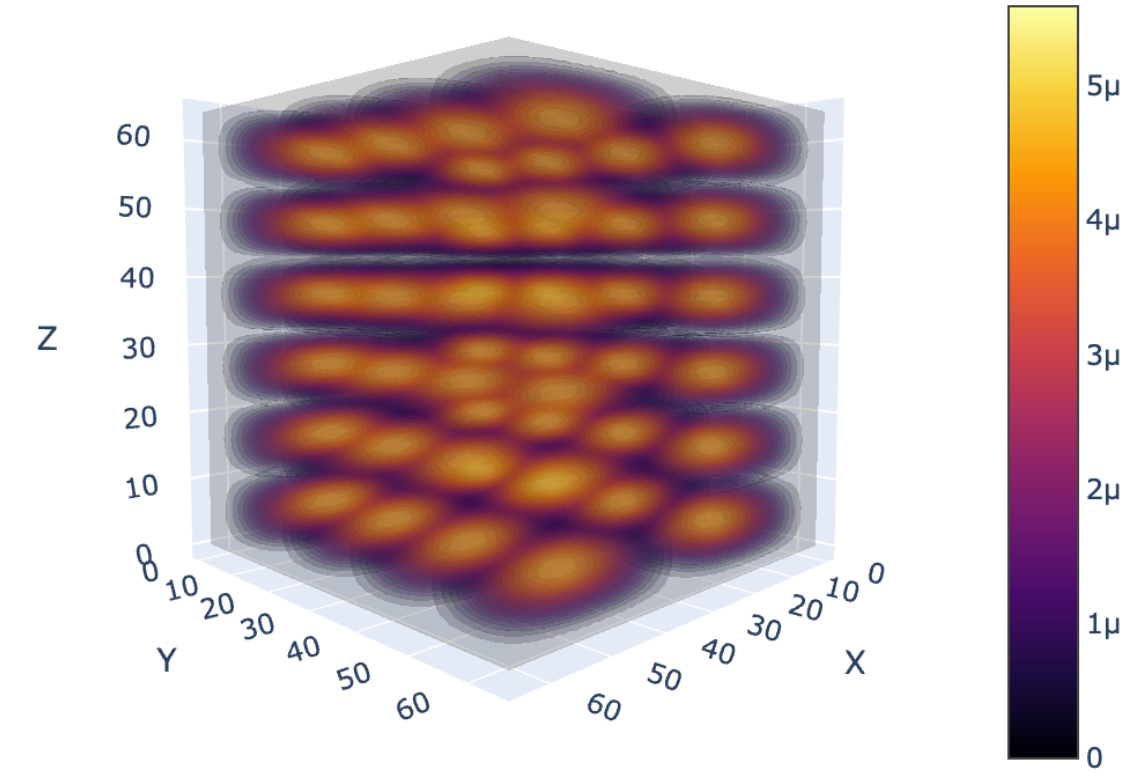


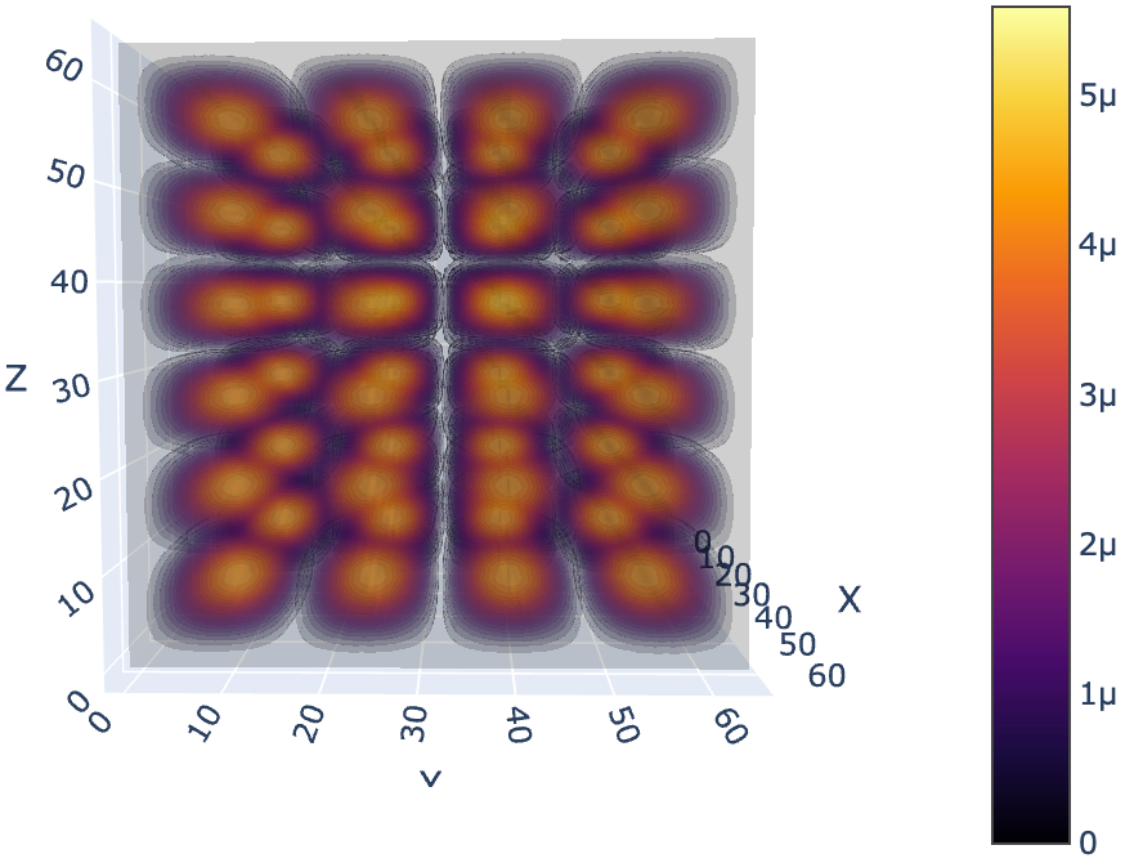
Визуализация сетки, полученной численным решением:





Визуализация сетки погрешности:





Вывод

Задача для трехмерного гиперболического уравнения в области, представляющей из себя прямоугольный параллелепипед, подходит для распараллеливания с помощью технологии MPI, позволяя получить ускорение вплоть до 19 раз. Причем при большем размере сетки ниже погрешность, а распараллеливание дает лучшие результаты по сравнению с мелкими сетками.

Также было замечено, что последовательный код работает немного быстрее, чем код на одном MPI процессе, это связано с накладными расходами на инициализацию, а также с действиями, необходимыми для подготовки разделения данных между процессами.

В сравнении с OpenMP решением мы получили такую же точность, но больший прирост в скорости работы для большего числа процессов относительно такого же числа нитей.