

**Московский государственный
университет имени М. В. Ломоносова**

**Факультет вычислительной математики
и кибернетики**

Итоговый отчет

По курсу: «Суперкомпьютерное моделирование и
технологии»

Цирунов Леонид Александрович
группа 628
30 ноября 2024 г.

Математическая постановка задачи

В трехмерной 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

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

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: $\omega_{ht} = \bar{\omega}_h \times \omega_\tau$

$$T = T_0$$

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

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

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

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:

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

Где Δ_h - семиточечный разностный аналог оператора Лапласа:

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

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

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

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

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:

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

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

Границные условия:

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

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

$$\Pi: u(x, y, 0, t) = u(x, y, L_z, t), \quad u_z(x, y, 0, t) = u_z(x, y, L_z, t)$$

Алгоритм численного решения:

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

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, задающую количество потоков, отключающую динамическое управление количеством потоков, 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: *I*, *pi*, *custom*.

В случае *I* 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.

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

Распараллеливание производится с использованием технологии MPI и OpenMP.

OpenMP

Для распараллеливания используются прагмы:

- **#pragma omp parallel for collapse(N)** , где N - количество вложенных циклов
- **#pragma omp parallel for collapse(3) reduction(max : error)** - для корректного сбора максимальной погрешности между процессами. Переменная error будет в этом случае private для каждого потока, а в конце будет произведена редукционная операция Max по всем потокам

MPI

Для реализации решения использовались встроенные функции библиотеки MPI. Изначально производится инициализация среды, затем создается декартова топология процессов и процессы соседи по каждому измерению. В зависимости от топологии определяются блоки сетки, которые обсчитываются процессом. На каждом шаге после 0 производится обмен граничными элементами между процессами и запись значений точек в гало (дополнительный слой вокруг блока для хранения данных соседей). Для отправки и получения данных используются неблокирующие асинхронные функции *Irecv* и *Isend*.

- *MPI_Init(&argc, &argv)* — инициализация среды MPI.
- *MPI_Comm_rank(MPI_COMM_WORLD, &rank)* — определение уникального идентификатора процесса (ранга) в заданном коммуникаторе.
- *MPI_Comm_size(MPI_COMM_WORLD, &proc_num)* — определение общего количества процессов в заданном коммуникаторе.
- *MPI_Abort(MPI_COMM_WORLD, 1)* — прерывание выполнения всех процессов в коммуникаторе в случае ошибки.
- *MPI_Dims_create(proc_num, 3, dims)* — распределение процессов по сетке, с определением размерности в каждом измерении.
- *MPI_Cart_create(MPI_COMM_WORLD, 3, dims, periods, 1, &comm_cart)* — автоматическое создание декартовой топологии процессов.
- *MPI_Cart_coords(comm_cart, rank, 3, coords)* — определение декартовых координат процесса в сетке по его рангу.
- *MPI_Cart_shift(comm_cart, 0, 1, &neighbors[0], &neighbors[1])* — нахождение соседей процесса вдоль измерения dim0.
- *MPI_Cart_shift(comm_cart, 1, 1, &neighbors[2], &neighbors[3])* — нахождение соседей процесса вдоль измерения dim1.
- *MPI_Cart_shift(comm_cart, 2, 1, &neighbors[4], &neighbors[5])* — нахождение соседей процесса вдоль измерения dim2.

- *MPI_Isend(b.left_send.data(), data_size, MPI_DOUBLE, send_neighbor, b.rank + 0, comm_cart, &reqs[req_count++])* — инициация асинхронной отправки данных. (Отправка производится по 6 направлениям, для каждого из которых вызывается данная функция с указанием нужного тега сообщения и ранга процесса - получателя)
- *MPI_Irecv(b.right_receive.data(), data_size, MPI_DOUBLE, recv_neighbor, recv_neighbor + 0, comm_cart, &reqs[req_count++])* — инициация асинхронного принятия данных. (Принятие производится по 6 направлениям, для каждого из которых вызывается данная функция с указанием нужного тега сообщения и ранга процесса - отправителя)
- *MPI_Waitall(req_count, reqs, MPI_STATUSES_IGNORE)* — Ожидание завершения всех асинхронных оперций, для дальнейшей обработки полученных данных.
- *MPI_Reduce(&error, &step_max_error, 1, MPI_DOUBLE, MPI_MAX, 0, comm_cart)* — собирает данные со всех процессов о значении переменной *error* и выполняет операцию получения максимального значения, после чего передает его в переменную *step_max_error* процесса с рангом 0.
- *MPI_Reduce(&local_time, &time, 1, MPI_DOUBLE, MPI_MAX, 0, comm_cart)* — собирает данные со всех процессов о значении переменной *local_time* и выполняет операцию получения максимального значения, после чего передает его в переменную *time* процесса с рангом 0.
- *MPI_Barrier(MPI_COMM_WORLD)* — синхронизация всех процессов в коммуникаторе, ожидает, что все процессы достигнут данной точки перед продолжением работы.
- *MPI_Finalize()* — завершение среды MPI. Освобождение ресурсов и корректное завершение работы MPI.

Результаты расчетов

OpenMP

Таблица 1: Результаты при $L = 1$

Число OpenMP нитей	Число точек сетки N по одной оси	Время решения	Ускорение	Погрешность
0-Sequential	128	8,77943	1,000	1.4004e-06
1	128	8,69085	1,0102	1.4004e-06
2	128	4,52765	1,9391	1.4004e-06
4	128	2,31081	3,7993	1.4004e-06
8	128	1,32673	6,6173	1.4004e-06
16	128	0,944555	9,2948	1.4004e-06
32	128	0,749173	11,7188	1.4004e-06
0-Sequential	256	62,994	1,000	3.49927e-07
1	256	63,2269	0,9963	3.49927e-07
2	256	32,6417	1,9299	3.49927e-07
4	256	16,4166	3,8372	3.49927e-07
8	256	9,30982	6,7664	3.49927e-07
16	256	6,73194	9,3575	3.49927e-07
32	256	5,48987	11,4746	3.49927e-07
0-Sequential	512	484,384	1,000	8.71479e-08
1	512	483,014	1,0028	8.71479e-08
2	512	251,639	1,9249	8.71479e-08
4	512	126,323	3,8345	8.71479e-08
8	512	70,0937	6,9105	8.71479e-08
16	512	44,9736	10,7704	8.71479e-08
32	512	36,6006	13,2343	8.71479e-08

Таблица 2: Результаты при $L = \pi$

Число OpenMP нитей	Число точек сетки N по одной оси	Время решения	Ускорение	Погрешность
0-Sequential	128	8,47777	1,000	1.41974e-07
1	128	8,57208	0,9890	1.41974e-07
2	128	4,43321	1,9123	1.41974e-07
4	128	2,28357	3,7125	1.41974e-07
8	128	1,21966	6,9509	1.41974e-07
16	128	0,966537	8,7713	1.41974e-07
32	128	0,740379	11,4506	1.41974e-07
0-Sequential	256	62,5438	1,000	3.55074e-08
1	256	63,1029	0,9911	3.55074e-08
2	256	32,5949	1,9188	3.55074e-08
4	256	16,5531	3,7784	3.55074e-08
8	256	8,86288	7,0568	3.55074e-08
16	256	6,70346	9,3301	3.55074e-08
32	256	5,50078	11,3700	3.55074e-08
0-Sequential	512	480,591	1,000	8.8744e-09
1	512	481,711	0,9977	8.8744e-09
2	512	248,499	1,9340	8.8744e-09
4	512	125,171	3,8395	8.8744e-09
8	512	66,1442	7,2658	8.8744e-09
16	512	45,7946	10,4945	8.8744e-09
32	512	37,1287	12,9439	8.8744e-09

MPI

Таблица 1: Результаты при L = 1

Число MPI процессов	Число точек сетки N по одной оси	Время решения	Ускорение	Погрешность
0-Sequential	128	8,77943	1,000	1.4004e-06
1	128	9,20052	0,9542	1.4004e-06
2	128	4,59858	1,9092	1.4004e-06
4	128	2,61351	3,3592	1.4004e-06
8	128	1,57326	5,5804	1.4004e-06
16	128	0,850441	10,3234	1.4004e-06
32	128	0,590741	14,8617	1.4004e-06
0-Sequential	256	62,994	1,000	3.49927e-07
1	256	68,1183	0,9248	3.49927e-07
2	256	34,5796	1,8217	3.49927e-07
4	256	19,7141	3,1954	3.49927e-07
8	256	9,58322	6,5734	3.49927e-07
16	256	5,64634	11,1566	3.49927e-07
32	256	3,54626	17,7635	3.49927e-07
0-Sequential	512	484,384	1,000	8.71479e-08
1	512	508,865	0,9519	8.71479e-08
2	512	266,067	1,8205	8.71479e-08
4	512	130,923	3,6998	8.71479e-08
8	512	70,7735	6,8441	8.71479e-08
16	512	38,0006	12,7467	8.71479e-08
32	512	24,9555	19,4099	8.71479e-08

Таблица 2: Результаты при $L = \pi$

Число MPI процессов	Число точек сетки N по одной оси	Время решения	Ускорение	Погрешность
0-Sequential	128	8,47777	1,000	1.41974e-07
1	128	9,07345	0,9343	1.41974e-07
2	128	4,55414	1,8616	1.41974e-07
4	128	2,49134	3,4029	1.41974e-07
8	128	1,52365	5,5641	1.41974e-07
16	128	0,857452	9,8872	1.41974e-07
32	128	0,590185	14,3646	1.41974e-07
0-Sequential	256	62,5438	1,000	3.55074e-08
1	256	67,1549	0,9313	3.55074e-08
2	256	35,4423	1,7647	3.55074e-08
4	256	17,3933	3,5959	3.55074e-08
8	256	9,54596	6,5519	3.55074e-08
16	256	6,15678	10,1585	3.55074e-08
32	256	3,44088	18,1767	3.55074e-08
0-Sequential	512	480,591	1,000	8.8744e-09
1	512	512,18	0,9383	8.8744e-09
2	512	265,867	1,8076	8.8744e-09
4	512	137,52	3,4947	8.8744e-09
8	512	70,6636	6,8011	8.8744e-09
16	512	35,8618	13,4012	8.8744e-09
32	512	24,9462	19,2651	8.8744e-09

MPI + OpenMP

1. Количество OpenMP нитей на процесс = 1:

Таблица 1.1: Результаты при L = 1

Число MPI процессов	Число точек сетки N по одной оси	Время решения	Ускорение	Погрешность
0-Sequential	128	8,77943	1,000	1.4004e-06
1	128	8,78846	0,9990	1.4004e-06
2	128	5,84676	1,5016	1.4004e-06
4	128	2,49951	3,5125	1.4004e-06
8	128	1,74339	5,0358	1.4004e-06
16	128	1,16079	7,5633	1.4004e-06
32	128	0,590567	14,8661	1.4004e-06
0-Sequential	256	62,994	1,000	3.49927e-07
1	256	64,9288	0,9702	3.49927e-07
2	256	34,2605	1,8387	3.49927e-07
4	256	21,9	2,8764	3.49927e-07
8	256	8,97345	7,0200	3.49927e-07
16	256	6,44694	9,7711	3.49927e-07
32	256	3,13201	20,1130	3.49927e-07
0-Sequential	512	484,384	1,000	8.71479e-08
1	512	499,143	0,9704	8.71479e-08
2	512	252,266	1,9201	8.71479e-08
4	512	134,291	3,6070	8.71479e-08
8	512	79,4925	6,0935	8.71479e-08
16	512	54,0828	8,9563	8.71479e-08
32	512	22,3816	21,6421	8.71479e-08

Таблица 1.2: Результаты при $L = \pi$

Число MPI процессов	Число точек сетки N по одной оси	Время решения	Ускорение	Погрешность
0-Sequential	128	8,47777	1,000	1.41974e-07
1	128	8,73889	0,9701	1.41974e-07
2	128	4,72145	1,7956	1.41974e-07
4	128	2,4073	3,5217	1.41974e-07
8	128	1,51146	5,6090	1.41974e-07
16	128	0,817303	10,3729	1.41974e-07
32	128	0,55731	15,2119	1.41974e-07
0-Sequential	256	62,5438	1,000	3.55074e-08
1	256	64,8378	0,9646	3.55074e-08
2	256	33,4614	1,8691	3.55074e-08
4	256	18,3309	3,4119	3.55074e-08
8	256	9,3261	6,7063	3.55074e-08
16	256	6,05766	10,3247	3.55074e-08
32	256	3,8675	16,1716	3.55074e-08
0-Sequential	512	480,591	1,000	8.8744e-09
1	512	501,871	0,9576	8.8744e-09
2	512	264,851	1,8146	8.8744e-09
4	512	143,562	3,3476	8.8744e-09
8	512	87,9549	5,4641	8.8744e-09
16	512	39,0933	12,2934	8.8744e-09
32	512	20,6163	23,3112	8.8744e-09

2) Количество OpenMP нитей на процесс = 2:

Таблица 2.1: Результаты при L = 1

Число MPI процессов	Число точек сетки N по одной оси	Время решения	Ускорение	Погрешность
0-Sequential	128	8,77943	1,000	1.4004e-06
1	128	4,75797	1,8452	1.4004e-06
2	128	3,26669	2,6876	1.4004e-06
4	128	1,76244	4,9814	1.4004e-06
8	128	1,16557	7,5323	1.4004e-06
16	128	0,820489	10,7002	1.4004e-06
32	128	0,513303	17,1038	1.4004e-06
0-Sequential	256	62,994	1,000	3.49927e-07
1	256	33,9802	1,8538	3.49927e-07
2	256	21,4607	2,9353	3.49927e-07
4	256	11,073	5,6890	3.49927e-07
8	256	5,19309	12,1304	3.49927e-07
16	256	4,33959	14,5161	3.49927e-07
32	256	2,78488	22,6200	3.49927e-07
0-Sequential	512	484,384	1,000	8.71479e-08
1	512	264,1	1,8341	8.71479e-08
2	512	142,708	3,3942	8.71479e-08
4	512	68,9858	7,0215	8.71479e-08
8	512	45,2255	10,7104	8.71479e-08
16	512	26,8418	18,0459	8.71479e-08
32	512	17,6212	27,4887	8.71479e-08

Таблица 2.2: Результаты при $L = \pi$

Число MPI процессов	Число точек сетки N по одной оси	Время решения	Ускорение	Погрешность
0-Sequential	128	8,47777	1,000	1.41974e-07
1	128	4,56087	1,8588	1.41974e-07
2	128	3,26108	2,5997	1.41974e-07
4	128	1,30556	6,4936	1.41974e-07
8	128	0,999156	8,4849	1.41974e-07
16	128	0,663293	12,7813	1.41974e-07
32	128	0,466975	18,1547	1.41974e-07
0-Sequential	256	62,5438	1,000	3.55074e-08
1	256	33,6383	1,8593	3.55074e-08
2	256	17,9906	3,4765	3.55074e-08
4	256	9,85455	6,3467	3.55074e-08
8	256	7,98199	7,8356	3.55074e-08
16	256	3,15786	19,8058	3.55074e-08
32	256	2,73497	22,8682	3.55074e-08
0-Sequential	512	480,591	1,000	8.8744e-09
1	512	254,271	1,8901	8.8744e-09
2	512	139,785	3,4381	8.8744e-09
4	512	85,3684	5,6296	8.8744e-09
8	512	48,5356	9,9018	8.8744e-09
16	512	32,6227	14,7318	8.8744e-09
32	512	14,7935	32,4866	8.8744e-09

3) Количество OpenMP нитей на процесс = 4 :

Таблица 3.1: Результаты при L = 1

Число MPI процессов	Число точек сетки N по одной оси	Время решения	Ускорение	Погрешность
0-Sequential	128	8,77943	1,000	1.4004e-06
1	128	2,89351	3,0342	1.4004e-06
2	128	1,96257	4,4734	1.4004e-06
4	128	1,09617	8,0092	1.4004e-06
8	128	0,945275	9,2877	1.4004e-06
16	128	0,732593	11,9840	1.4004e-06
32	128	0,325534	26,9693	1.4004e-06
0-Sequential	256	62,994	1,000	3.49927e-07
1	256	20,3292	3,0987	3.49927e-07
2	256	13,4621	4,6794	3.49927e-07
4	256	7,30345	8,6252	3.49927e-07
8	256	3,43487	18,3396	3.49927e-07
16	256	2,66733	23,6169	3.49927e-07
32	256	1,81473	34,7126	3.49927e-07
0-Sequential	512	484,384	1,000	8.71479e-08
1	512	153,466	3,1563	8.71479e-08
2	512	82,864	5,8455	8.71479e-08
4	512	48,7568	9,9347	8.71479e-08
8	512	29,0539	16,6719	8.71479e-08
16	512	21,6885	22,3337	8.71479e-08
32	512	11,6228	41,6753	8.71479e-08

Таблица 3.2: Результаты при $L = \pi$

Число MPI процессов	Число точек сетки N по одной оси	Время решения	Ускорение	Погрешность
0-Sequential	128	8,47777	1,000	1.41974e-07
1	128	2,3831	3,5575	1.41974e-07
2	128	2,02599	4,1845	1.41974e-07
4	128	1,13927	7,4414	1.41974e-07
8	128	0,860866	9,8480	1.41974e-07
16	128	0,38323	22,1219	1.41974e-07
32	128	0,337238	25,1388	1.41974e-07
0-Sequential	256	62,5438	1,000	3.55074e-08
1	256	17,0477	3,6688	3.55074e-08
2	256	12,0244	5,2014	3.55074e-08
4	256	6,94968	8,9995	3.55074e-08
8	256	5,36409	11,6597	3.55074e-08
16	256	2,68311	23,3102	3.55074e-08
32	256	1,93354	32,3468	3.55074e-08
0-Sequential	512	480,591	1,000	8.8744e-09
1	512	128,187	3,7491	8.8744e-09
2	512	91,6757	5,2423	8.8744e-09
4	512	54,3328	8,8453	8.8744e-09
8	512	31,1446	15,4310	8.8744e-09
16	512	18,3425	26,2010	8.8744e-09
32	512	11,8672	40,4974	8.8744e-09

4) Количество OpenMP нитей на процесс = 8:

Таблица 4.1: Результаты при L = 1

Число MPI процессов	Число точек сетки N по одной оси	Время решения	Ускорение	Погрешность
0-Sequential	128	8,77943	1,000	1.4004e-06
1	128	1,88086	4,6678	1.4004e-06
2	128	1,26382	6,9467	1.4004e-06
4	128	0,807667	10,8701	1.4004e-06
8	128	0,558438	15,7214	1.4004e-06
16	128	0,326319	26,9044	1.4004e-06
32	128	0,239759	36,6177	1.4004e-06
0-Sequential	256	62,994	1,000	3.49927e-07
1	256	12,8527	4,9012	3.49927e-07
2	256	8,85137	7,1169	3.49927e-07
4	256	4,19413	15,0196	3.49927e-07
8	256	3,44527	18,2842	3.49927e-07
16	256	1,9664	32,0352	3.49927e-07
32	256	1,3886	45,3651	3.49927e-07
0-Sequential	512	484,384	1,000	8.71479e-08
1	512	93,8672	5,1603	8.71479e-08
2	512	46,0996	10,5073	8.71479e-08
4	512	32,7387	14,7955	8.71479e-08
8	512	24,4265	19,8303	8.71479e-08
16	512	16,0972	30,0912	8.71479e-08
32	512	9,77882	49,5340	8.71479e-08

Таблица 4.2: Результаты при $L = \pi$

Число MPI процессов	Число точек сетки N по одной оси	Время решения	Ускорение	Погрешность
0-Sequential	128	8,47777	1,000	1.41974e-07
1	128	1,3541	6,2608	1.41974e-07
2	128	1,2967	6,5380	1.41974e-07
4	128	0,734378	11,5442	1.41974e-07
8	128	0,641668	13,2121	1.41974e-07
16	128	0,295744	28,6659	1.41974e-07
32	128	0,23282	36,4134	1.41974e-07
0-Sequential	256	62,5438	1,000	3.55074e-08
1	256	10,6586	5,8679	3.55074e-08
2	256	6,59082	9,4895	3.55074e-08
4	256	4,87802	12,8216	3.55074e-08
8	256	3,38477	18,4780	3.55074e-08
16	256	2,46825	25,3393	3.55074e-08
32	256	1,49754	41,7644	3.55074e-08
0-Sequential	512	480,591	1,000	8.8744e-09
1	512	79,9815	6,0088	8.8744e-09
2	512	63,1985	7,6045	8.8744e-09
4	512	25,1734	19,0912	8.8744e-09
8	512	22,1404	21,7065	8.8744e-09
16	512	13,8749	34,6374	8.8744e-09
32	512	9,62267	49,9436	8.8744e-09

Оценка эффективности

Таблица 5: эффективность распараллеливания с использованием технологии OpenMP

Число OpenMP нитей	128	256	512
0-Sequential	100,00 %	100,00 %	100,00 %
1	98,90 %	99,63 %	100,28 %
2	95,62 %	96,50 %	96,25 %
4	92,81 %	95,93 %	95,86 %
8	86,89 %	84,58 %	86,38 %
16	54,82 %	58,48 %	67,32 %
32	35,78 %	35,86 %	41,36 %

Таблица 6: эффективность распараллеливания с использованием технологии MPI

Число MPI процессов	128	256	512
0-Sequential	100,00 %	1,000	100,00 %
1	95,42 %	92,48 %	95,19 %
2	95,46 %	91,09 %	91,03 %
4	83,98 %	79,88 %	92,50 %
8	69,76 %	82,17 %	85,55 %
16	64,52 %	69,73 %	79,67 %
32	46,44 %	55,51 %	60,66 %

Таблица 7: эффективность распараллеливания с использованием MPI + OpenMP

Число MPI процессов	Число OpenMP нитей	128	256	512
0-Sequential	0	100,00 %	100,00 %	100,00 %
1	1	99,90 %	97,02 %	97,04 %
2	1	75,08 %	91,94 %	96,01 %
4	1	87,81 %	71,91 %	90,18 %
8	1	62,95 %	87,75 %	76,17 %
16	1	47,27 %	61,07 %	55,98 %
32	1	46,46 %	62,85 %	67,63 %
1	2	92,26 %	92,69 %	91,71 %
2	2	67,19 %	73,38 %	84,86 %
4	2	62,27 %	71,11 %	87,77 %
8	2	47,08 %	75,82 %	66,94 %
16	2	33,44 %	45,36 %	56,39 %
32	2	26,72 %	35,34 %	42,95 %
1	4	75,85 %	77,47 %	78,91 %
2	4	55,92 %	58,49 %	73,07 %
4	4	50,06 %	53,91 %	62,09 %
8	4	29,02 %	57,31 %	52,10 %
16	4	18,73 %	36,90 %	34,90 %
32	4	21,07 %	27,12 %	32,56 %
1	8	58,35 %	61,27 %	64,50 %
2	8	43,42 %	44,48 %	65,67 %
4	8	33,97 %	46,94 %	46,24 %
8	8	24,56 %	28,57 %	30,98 %
16	8	21,02 %	25,03 %	27,06 %
32	8	14,30 %	17,72 %	19,35 %

Графики для сравнения ускорения

График 1: зависимость ускорения на Polus от числа процессов(MPI)/ потоков(OpenMP) для сетки 128 при L = 1:

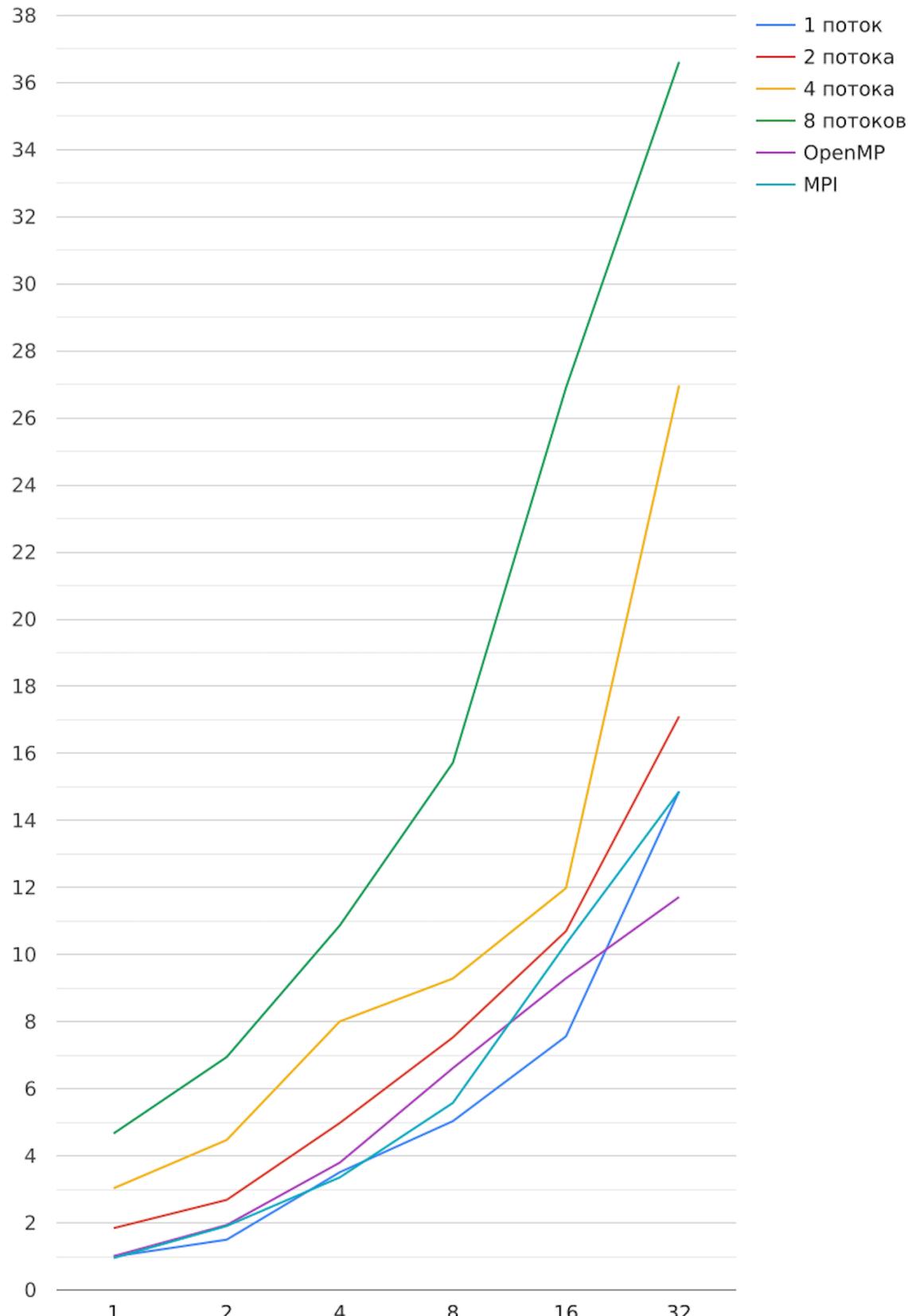


График 2: зависимость ускорения на Polus от числа процессов(MPI)/ потоков(OpenMP) для сетки 128 при $L = \pi$:

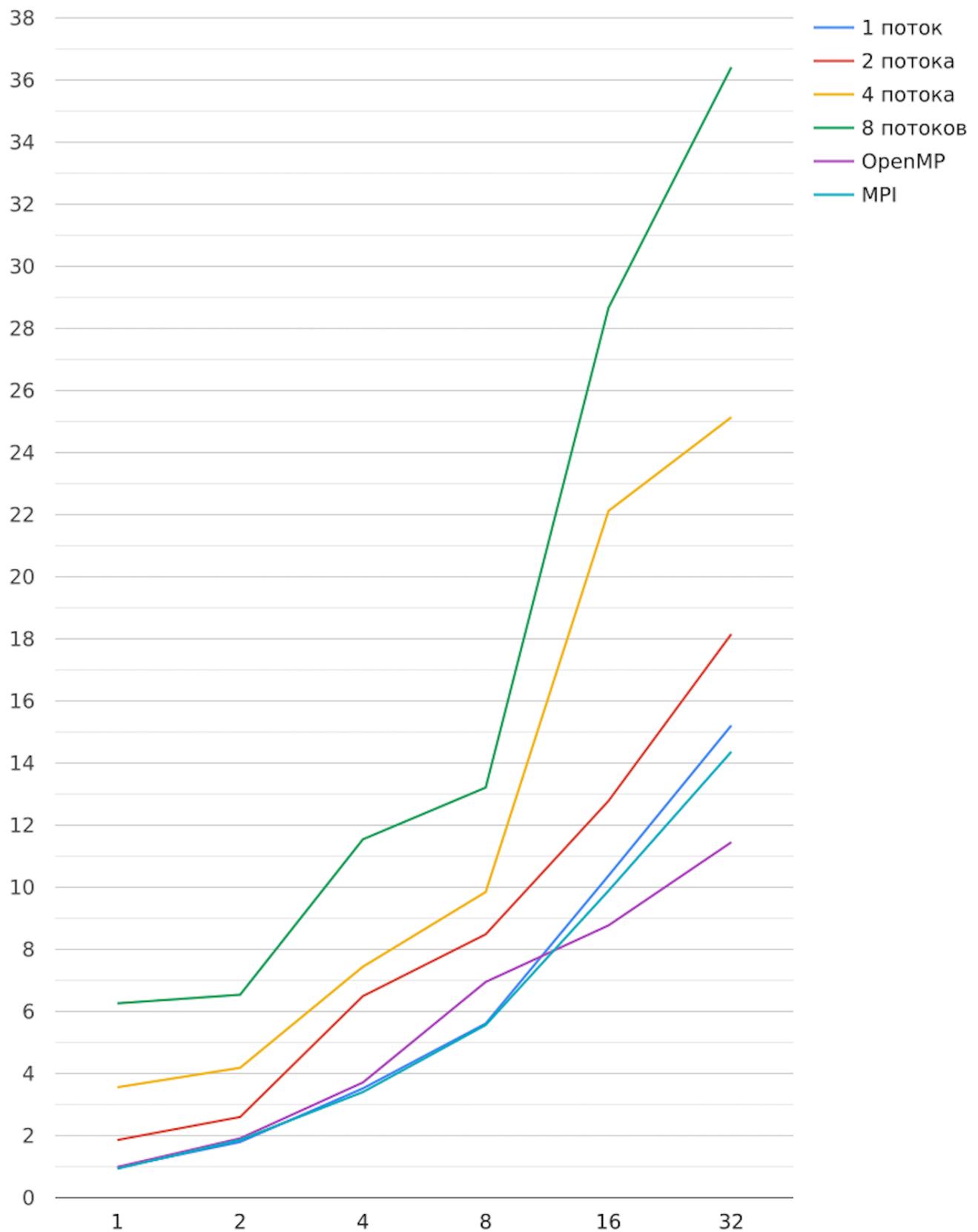


График 3: зависимость ускорения на Polus от числа процессов(MPI)/потоков(OpenMP) для сетки 256 при L = 1:

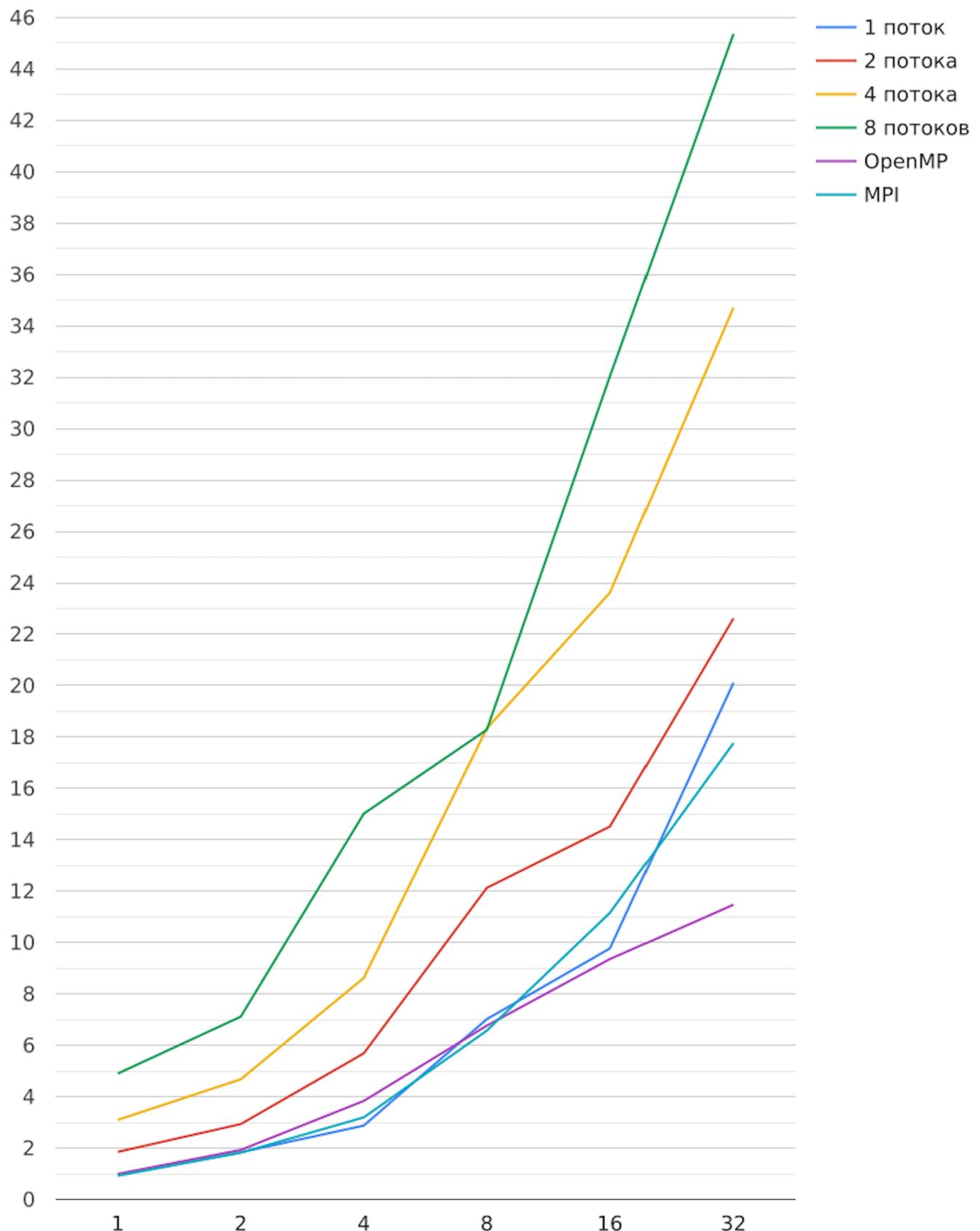


График 4: зависимость ускорения на Polus от числа процессов(MPI)/потоков(OpenMP) для сетки 256 при $L = \pi$:

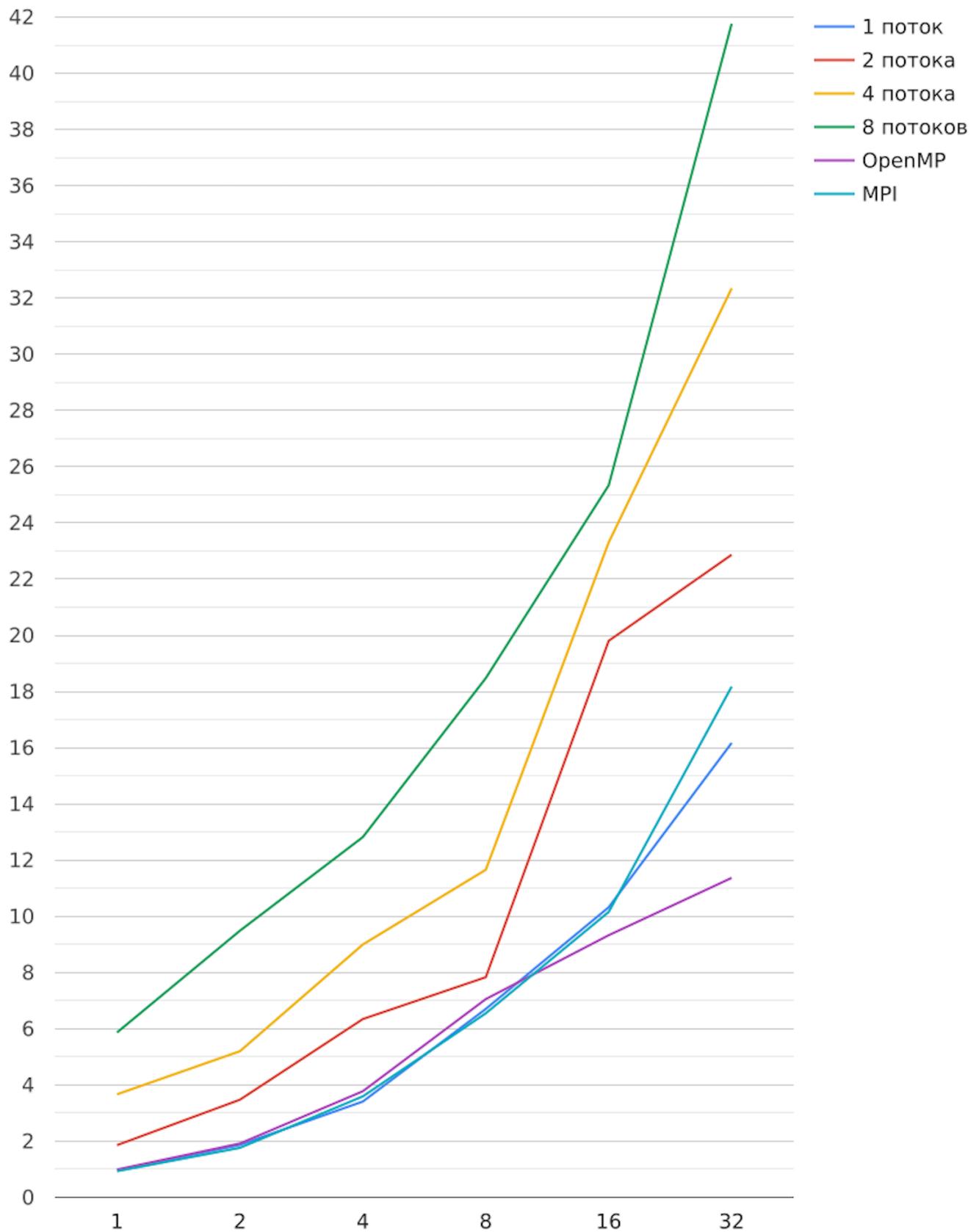


График 5: зависимость ускорения на Polus от числа процессов(MPI)/потоков(OpenMP) для сетки 512 при L = 1:

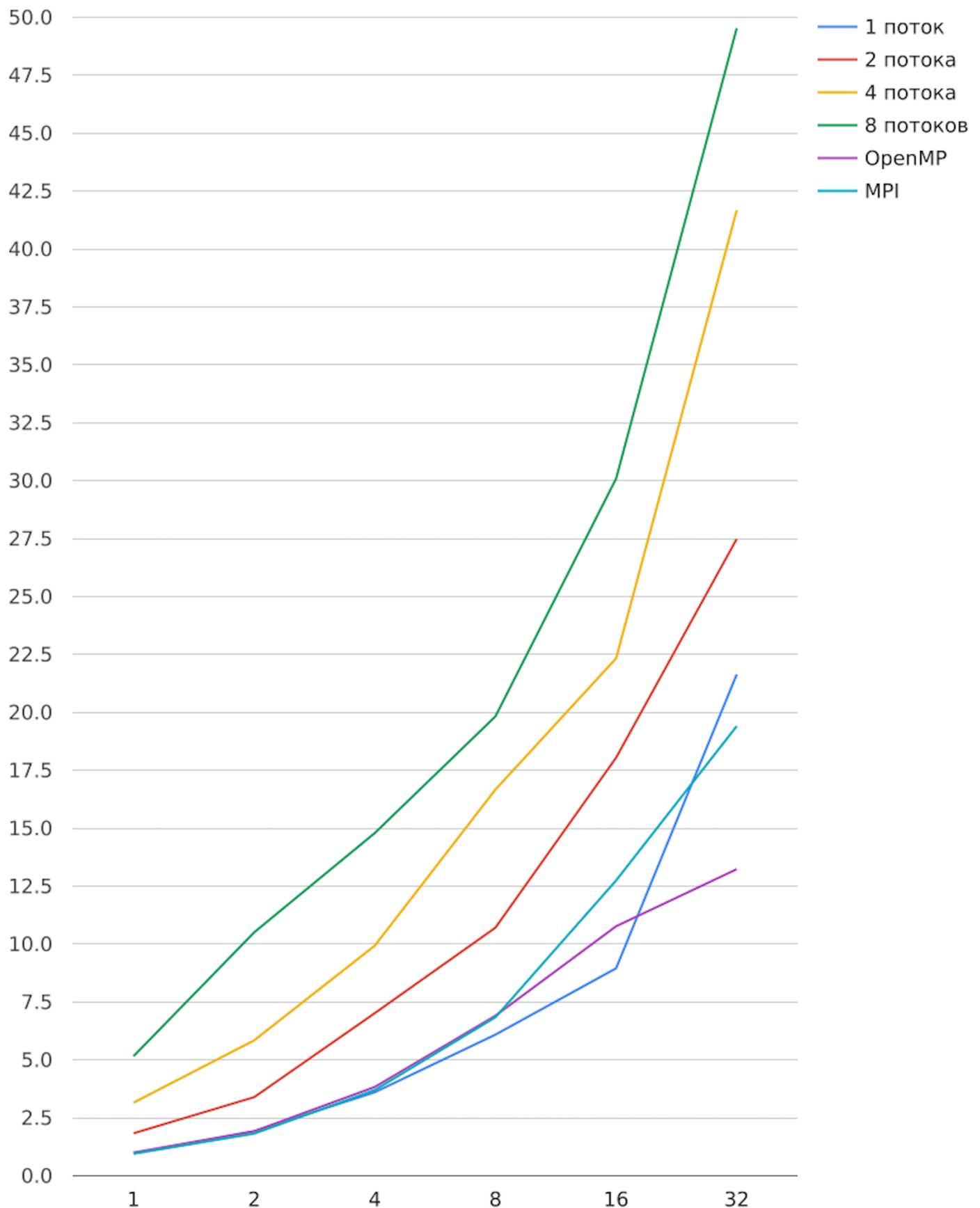


График 6: зависимость ускорения на Polus от числа процессов(MPI)/ потоков(OpenMP) для сетки 512 при $L = \pi$:

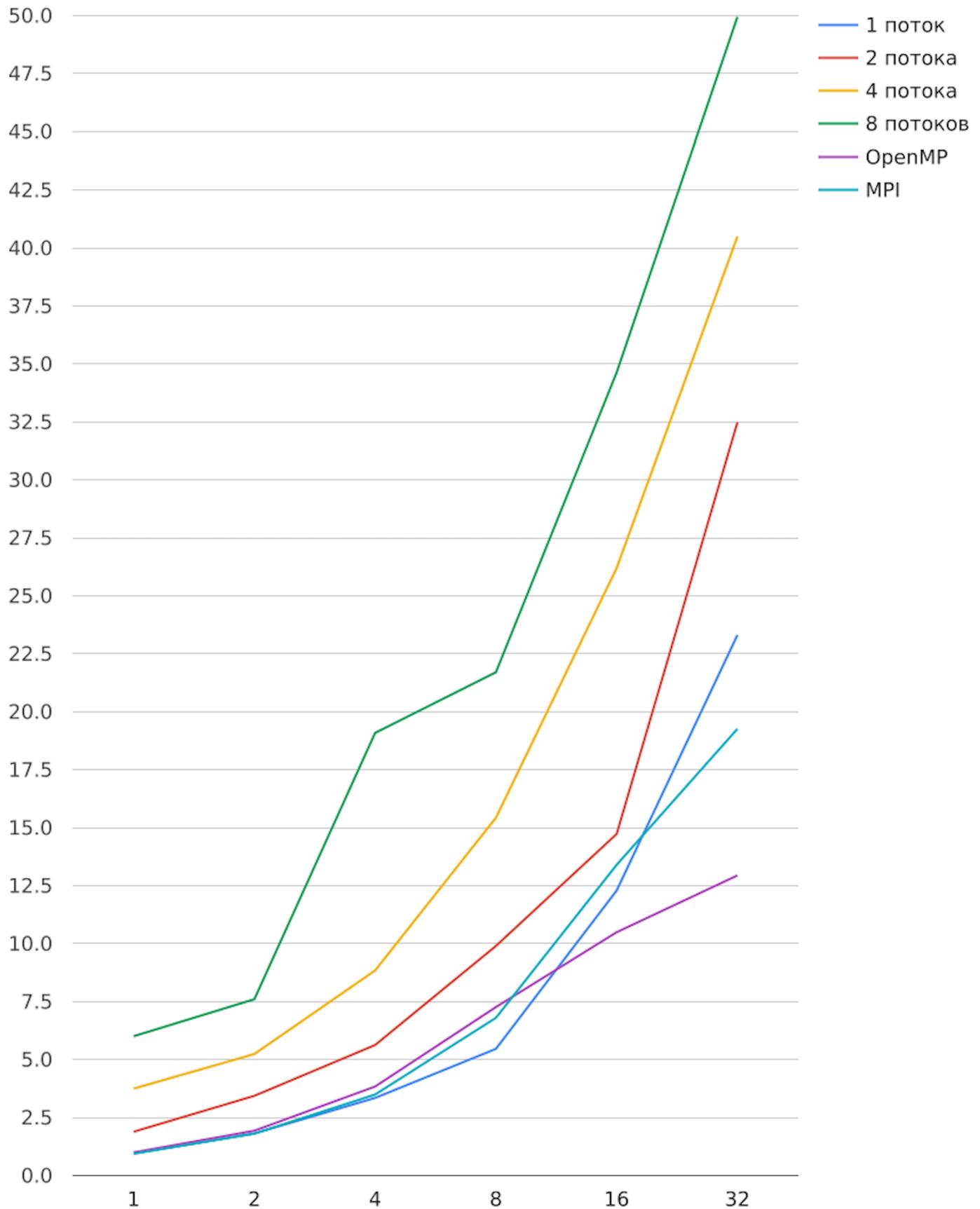


График 7: Максимальное ускорение для каждого вида сетки:

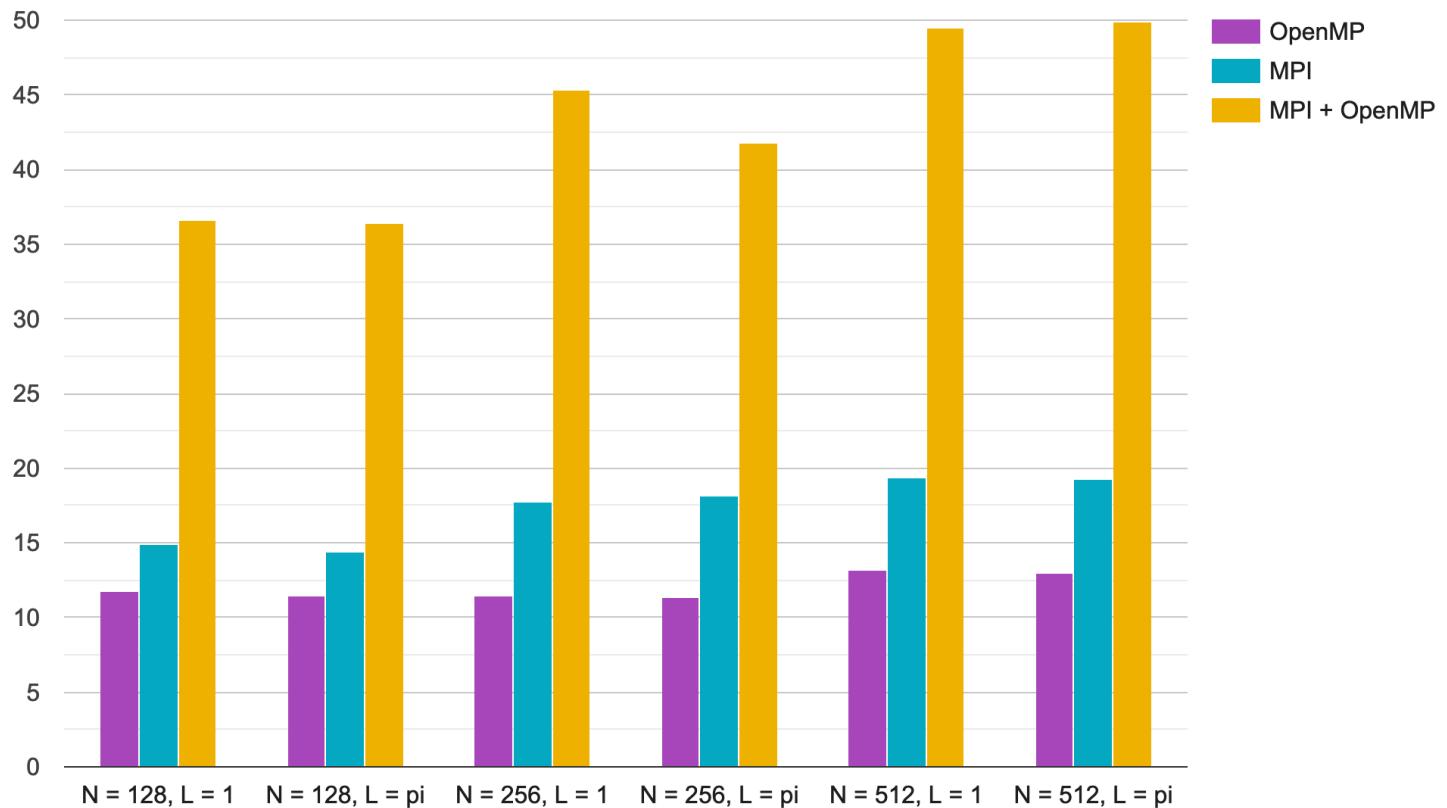
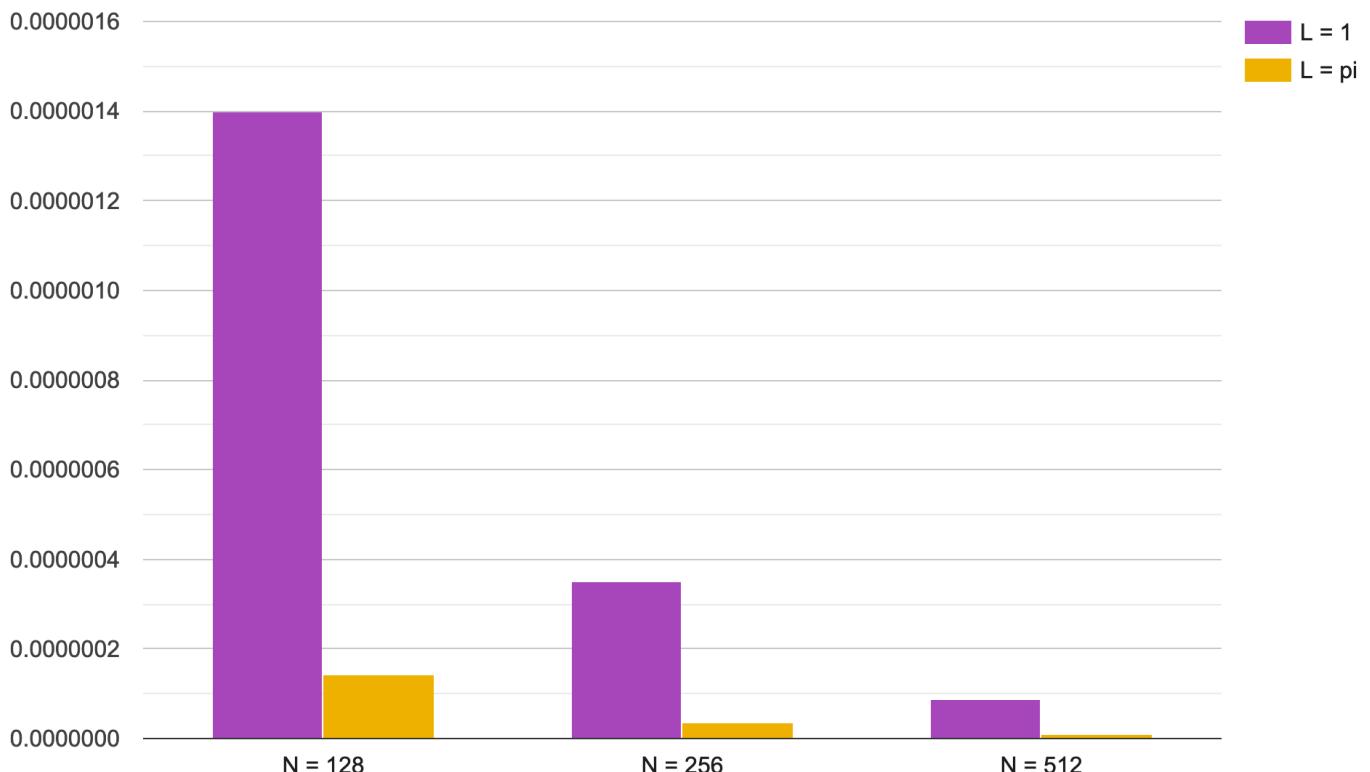


График 8: Размер погрешности в зависимости от параметров N и L:



Графики для сравнения эффективности

График 9: зависимость эффективности в зависимости от числа процессов(MPI)/потоков(OpenMP) для сетки 128:

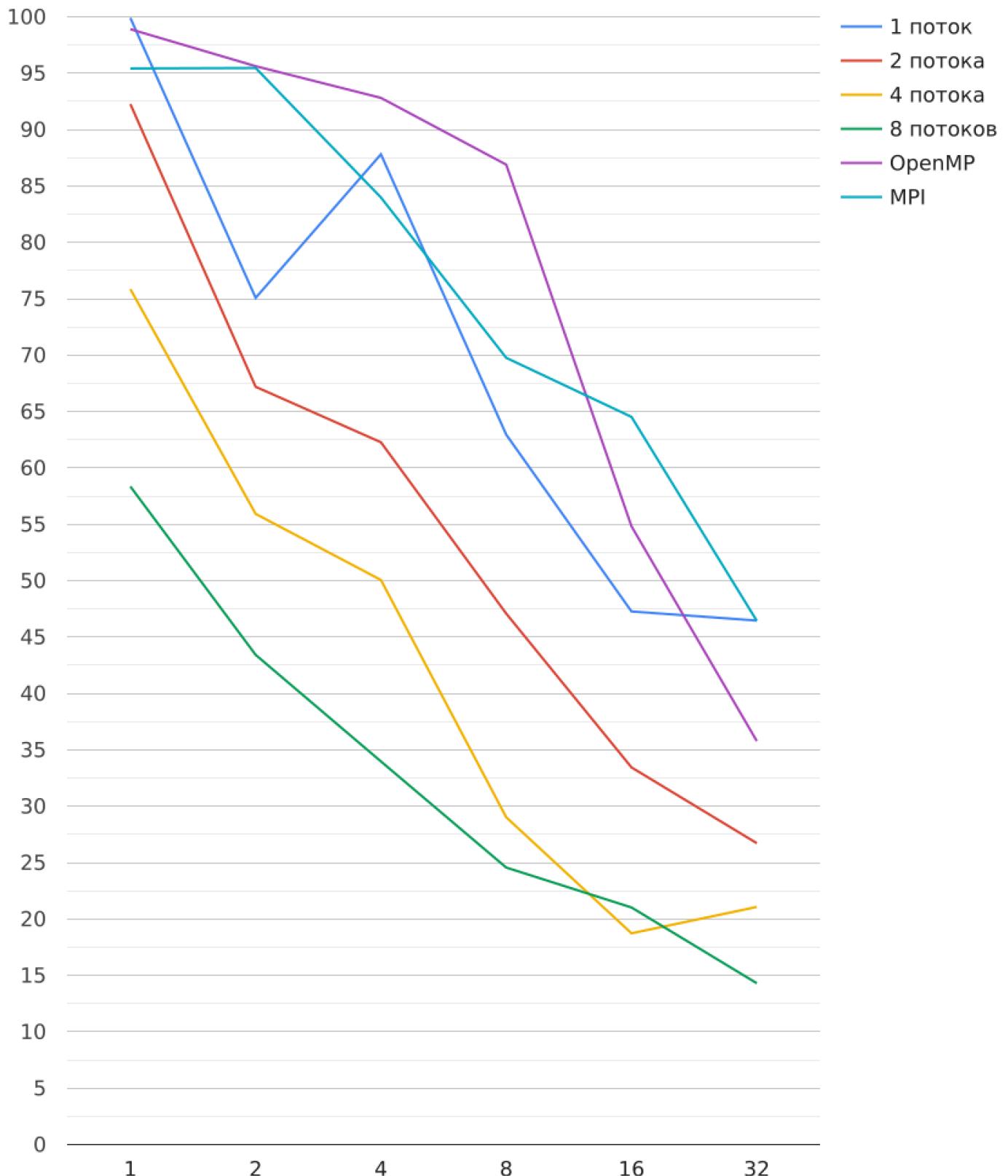


График 10: зависимость эффективности в зависимости от числа процессов(MPI)/потоков(OpenMP) для сетки 256:

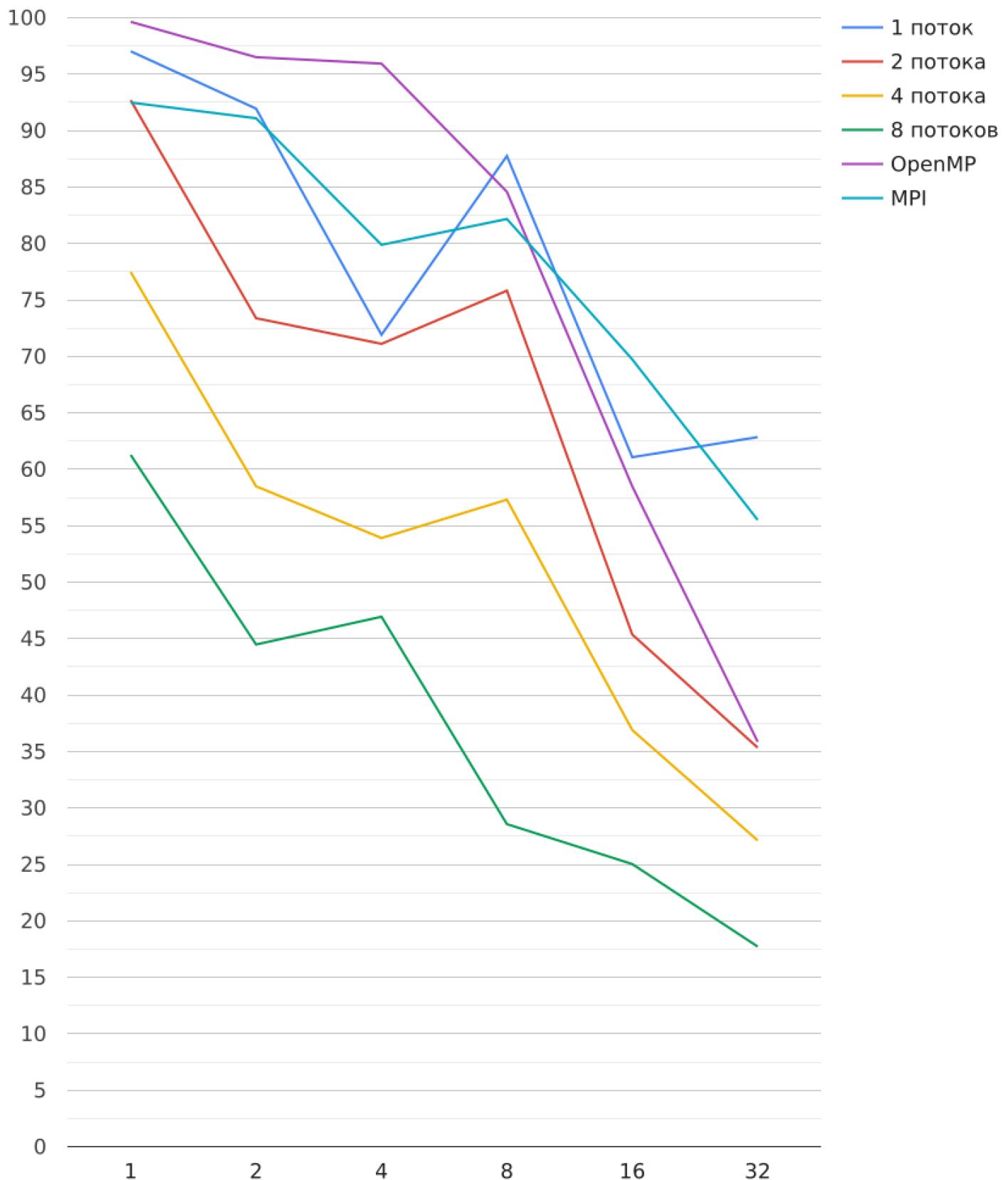
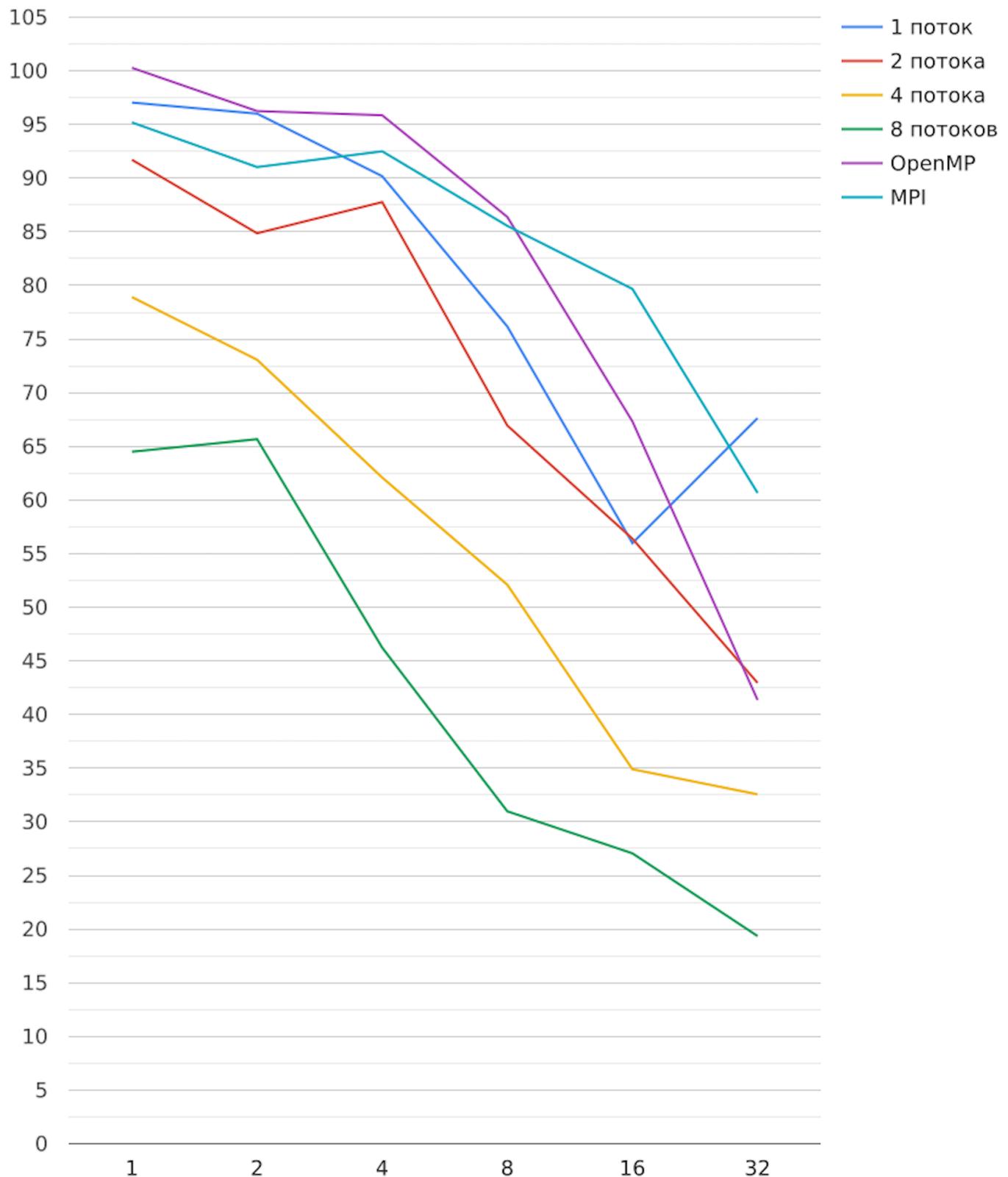
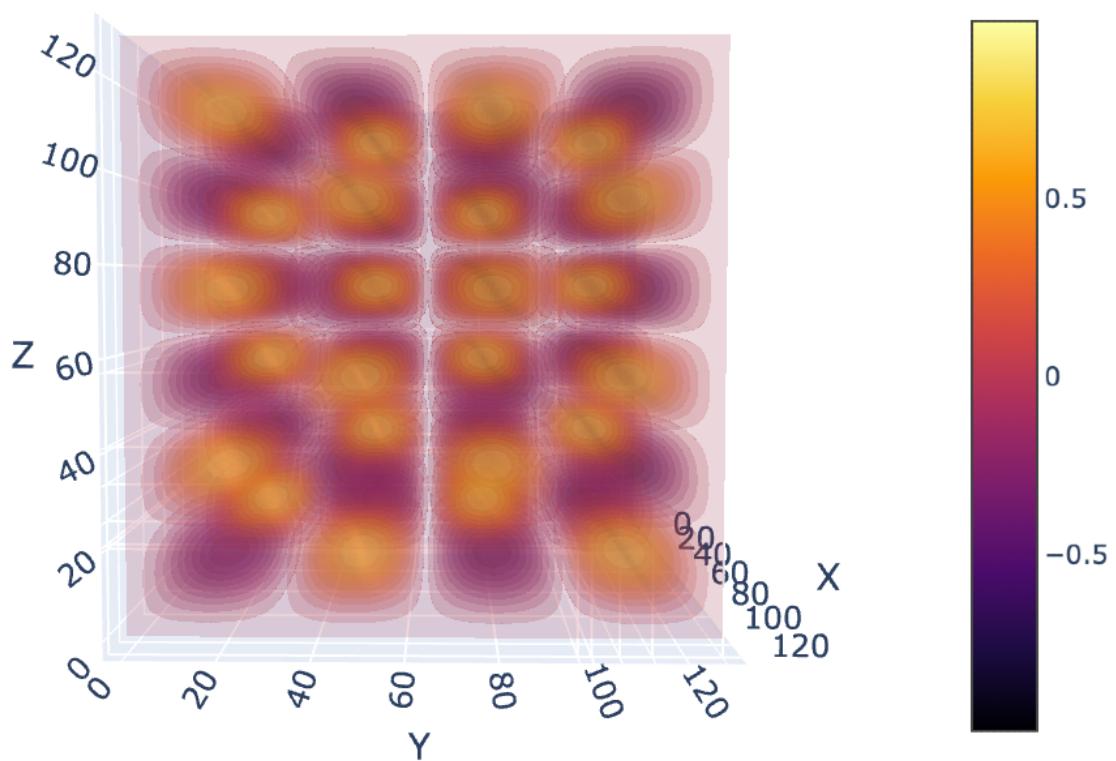
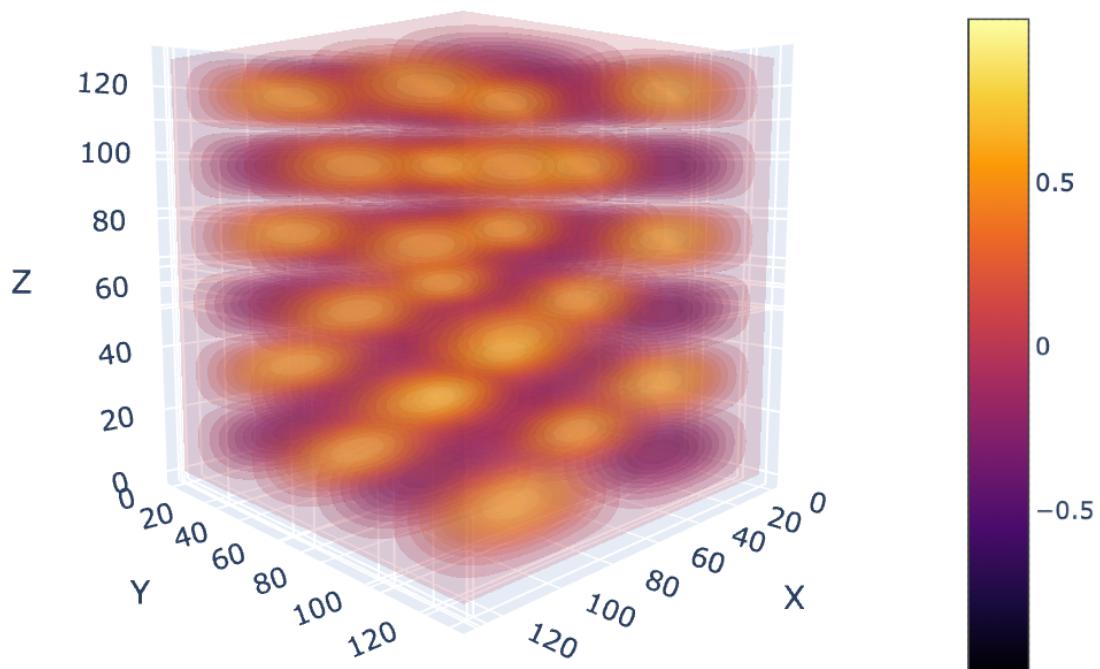


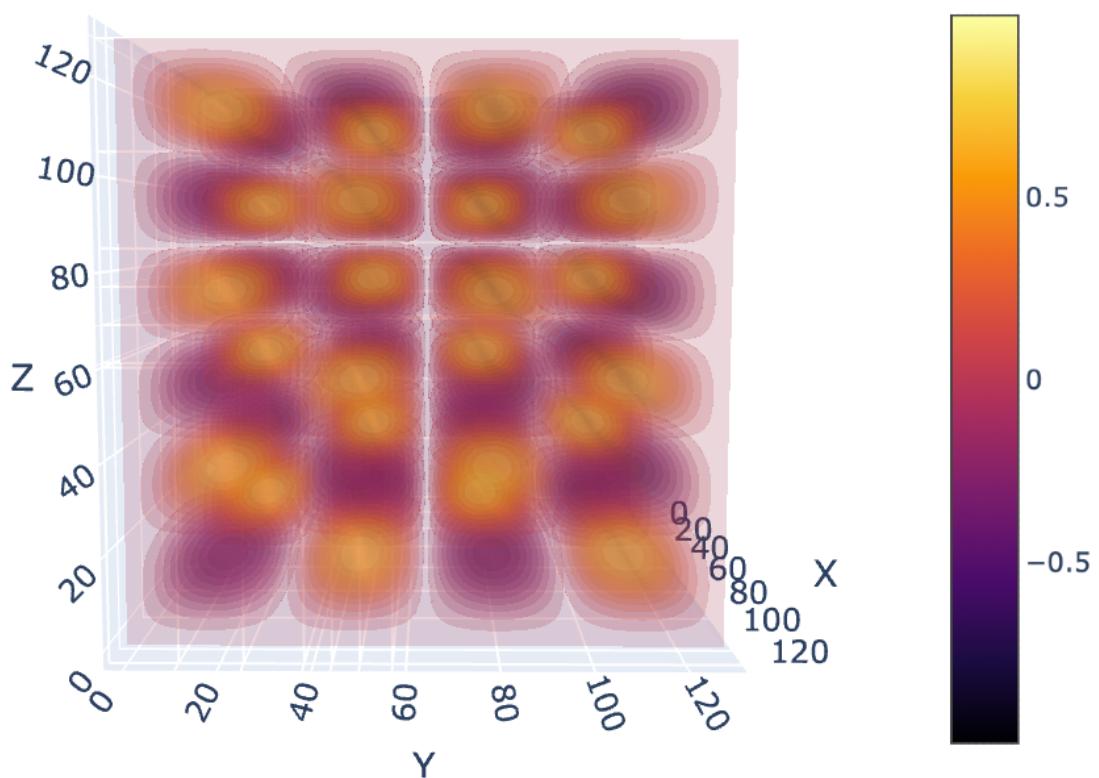
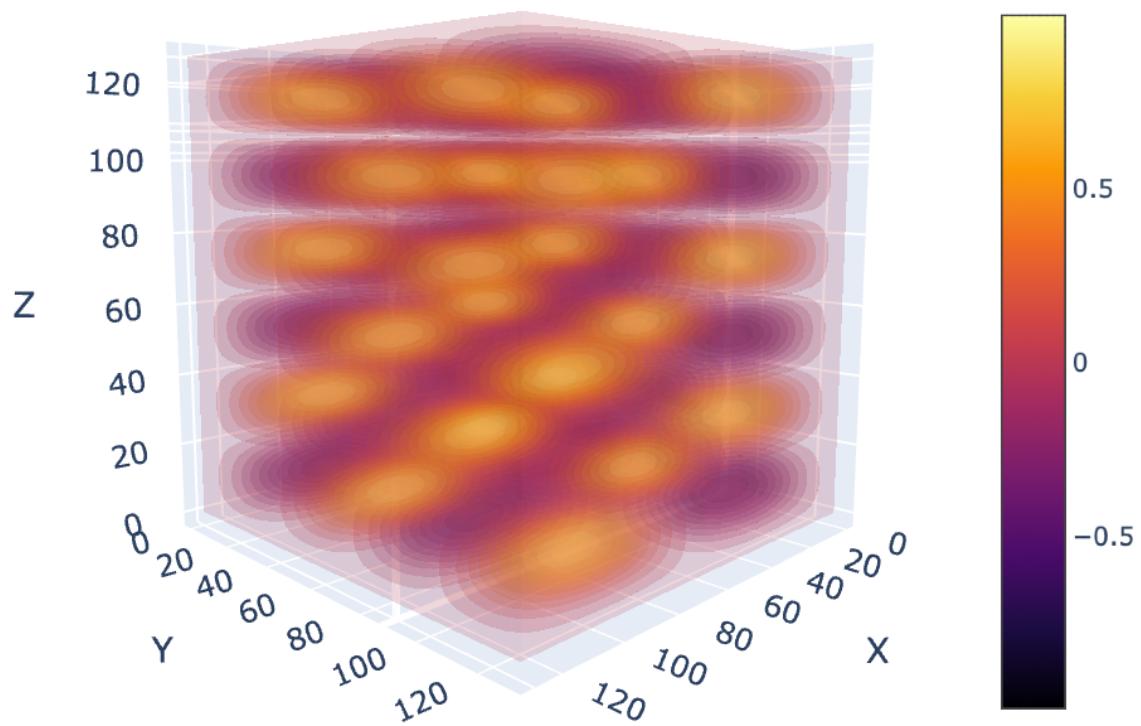
График 11: зависимость эффективности в зависимости от числа процессов(MPI)/потоков(OpenMP) для сетки 512:



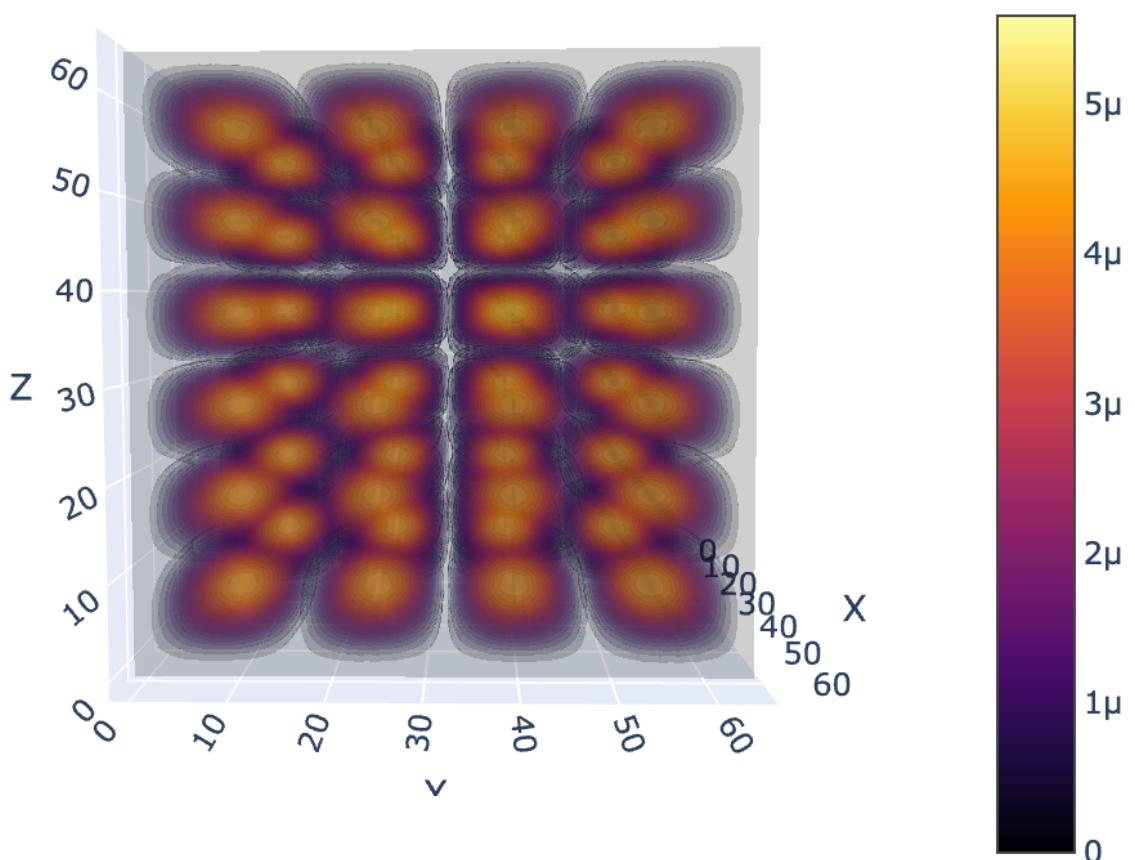
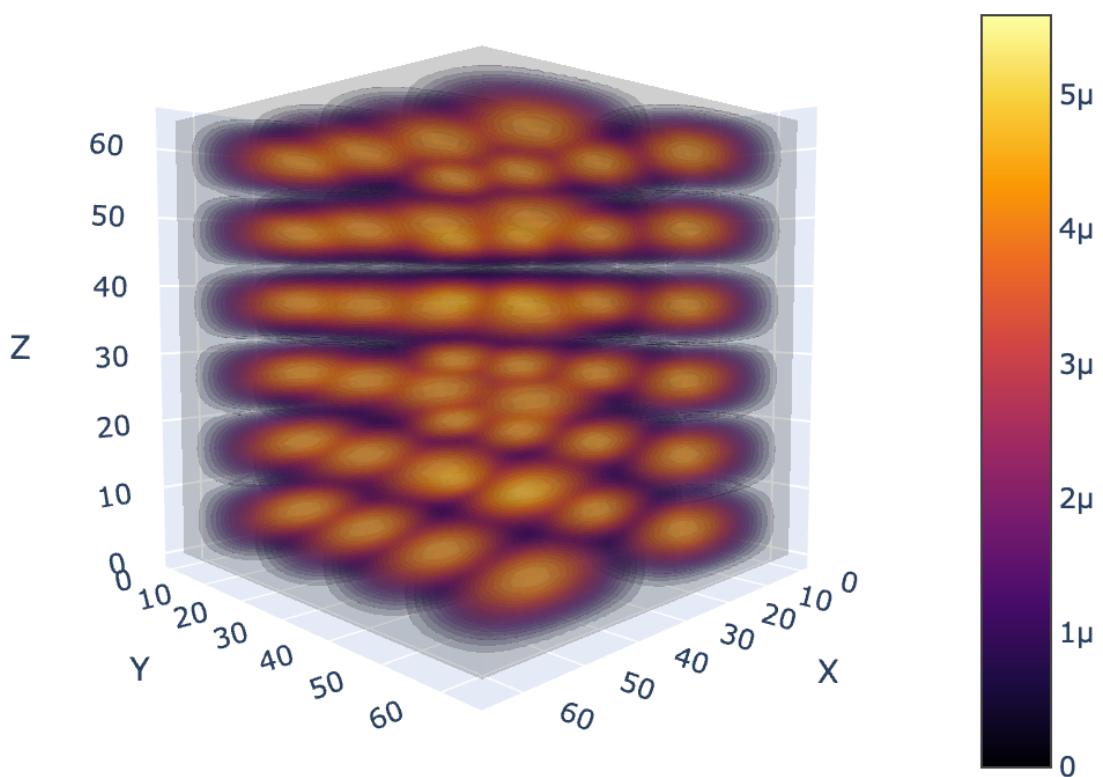
Визуализация сетки, полученной аналитическим решением:



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 и OpenMP, а также смешанной реализации. Причем при большем размере сетки ниже погрешность, а распараллеливание дает лучшие результаты по сравнению с мелкими сетками.

Из анализа графиков видно, что ускорение с помощью OpenMP дает почти линейный прирост в скорости работы программы при увеличении числа нитей. Решение с MPI в чистом виде и смешанное близится к экспоненциальному ускорению при увеличении числа процессов. Если же сравнивать прирост скорости для одинакового числа процессов в смешанной программе, то прирост при увеличении числа нитей схож с линейным. Наилучший результат работы показала смешанная программа, позволив получить ускорение вплоть до 45-49 раз.

При сравнении чистого MPI и OpenMP, можно сделать вывод, что до определенного числа потоков, в моем случае 8, распараллеливание с помощью нитей дает лучший результат, однако с увеличением их числа лучше перейти к MPI реализации. Это связано с необходимостью пересылки данных между процессами в MPI, соответственно чем меньше нужно пересылать данных (размер буфера), тем быстрее будет происходить обмен, и для количества процессов меньше 8, потери на пересылку снижают скорость работы в сравнении с OpenMP.