Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Учебный курс

“Суперкомпьютерное моделирование и технологии”

Решение трёхмерного гиперболического уравнения в прямоугольном параллелепипеде

ОТЧЕТ

о выполненном задании

студента 621 учебной группы факультета ВМК МГУ

Морквина Андрея Андреевича

Вариант 2

Москва, 2022

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 имеет следующий вид:

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

$$\frac{\partial^2 u}{\partial t^2} = \Delta u \qquad (1)$$

с начальными условиями

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

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

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

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

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

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

либо периодические граничные условия

$$u(0, y, z, t) = u(L_x, y, z, t), \qquad u_x(0, y, z, t) = u_x(L_x, y, z, t), \qquad (7)$$

$$u(x, 0, z, t) = u(x, L_y, z, t), \qquad u_y(x, 0, z, t) = u_y(x, L_y, z, t), \qquad (8)$$

$$u(x, y, 0, t) = u(x, y, L_z, t), \qquad u_z(x, y, 0, t) = u_z(x, y, L_z, t), \qquad (9)$$

В варианте 2 граничные условия представлены в (10) – (12):

$$u(0, y, z, t) = 0, \qquad u(L_x, y, z, t) = 0, \qquad (10)$$

$$u(x, 0, z, t) = 0, \qquad u(x, L_y, z, t) = 0, \qquad (11)$$

$$u(x, y, 0, t) = u(x, y, L_z, t), \qquad u_z(x, y, 0, t) = u_z(x, y, L_z, t), \qquad (12)$$

Аналитическое уравнение функции u:

$$u(x,y,z,t) = \sin\left(\frac{\pi}{L_x}\,\mathrm{x}\right) * \sin\left(\frac{\pi}{L_y}\,\mathrm{y}\right) * \sin\left(\frac{2\pi}{L_z}\,\mathrm{z}\right) * \cos(\mathrm{a}_t * \mathrm{t} + 2\pi),$$

$$\mathrm{a}_t = \pi\sqrt{\frac{1}{L_x^2} + \frac{1}{L_y^2} + \frac{4}{L_z^2}}$$

2. Численный метод решения задачи

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

$T = T_0$,
$L_x = L_{x_0}$, $L_y = L_{y_0}, L_z = L_{z_0}$,
$\omega_h = \{(x_i = ih_x, y_j = jh_y, z_k = kh_z), i,j,k = 0,\dots,N, h_xN = L_x, h_yN = L_y, h_zN = L_z\}$,
$\omega_\tau = \{t_n = n\tau, n = 0,\dots,K, \tau K = T\}$.

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:

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

Здесь Δ_h– семиточечный разностный аналог оператора Лапласа:

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

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 (т. е. 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:

$$u_{ijk}^0 = \phi(x_i, y_j, z_k), \ (x_i, y_j, z_k) \in \omega_h. \qquad (13)$$

Простейшая замена 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:

$$\frac{u_{ijk}^1 - u_{ijk}^0}{\tau} = \frac{\tau}{2}\,\Delta_h\phi(x_i, y_j, z_k), \qquad (x_i, y_j, z_k) \in \omega_h. \tag{13}$$

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

Разностная аппроксимация для периодических граничных условий выглядит следующим образом:

$$u_{0jk}^{n+1} = u_{Njk}^{n+1} \qquad u_{1jk}^{n+1} = u_{N+1jk}^{n+1}$$
$$u_{iok}^{n+1} = u_{iNk}^{n+1} \qquad u_{i1k}^{n+1} = u_{iN+1k}^{n+1}$$
$$u_{ij0}^{n+1} = u_{ijN}^{n+1} \qquad u_{ij1}^{n+1} = u_{ijN+1}^{n+1}$$

$i, j, k = 0, \ldots, N$.

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.

Разработка MPI версии была самой простой. Единственной проблема - это низкоуровневая модель на языке C, что вызвало ошибки при работе с памятью и дополнительные итерации с отладчиком и профилировщиком.

Разработка MPI/OpenMP версии была самой трудной. При распараллеливании компилятор IBM XL не хотел/плохо мог распараллеливать через стандарт OpenMP. После чего в модулях на Polus нашлись другие компиляторы, на которых удалось получить адекватные результаты.

Для версии MPI/OpenACC пришлось довольно много задублировать код, так как компилятор довольно упорно не хотел воспринимать прагмы routine. Так же из-за этого пришлось оптимизировать граничные условия. Далее были проведены эксперименты с параметрами gang и vector и оптимизированы пересылки данных.

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.

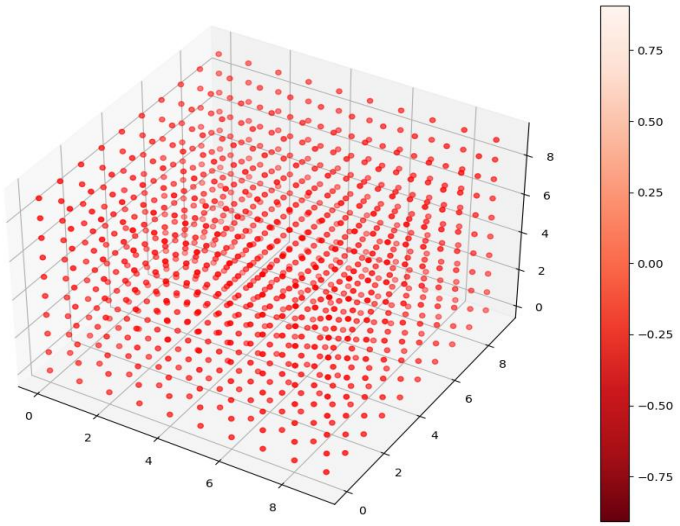


Рисунок 1. График аналитической функции при $L = 1$ на 20 шаге по времени на сетке 10^3.

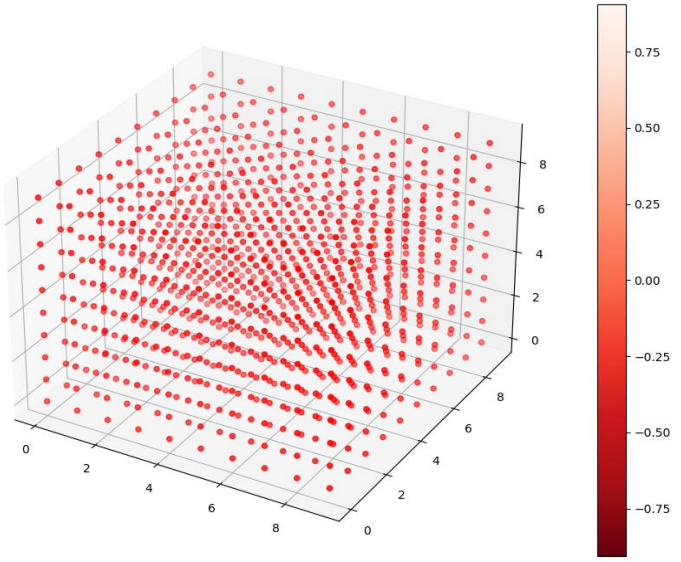


Рисунок 2. График вычисленной функции при $L = 1$ на 20 шаге по времени на сетке 10^3.

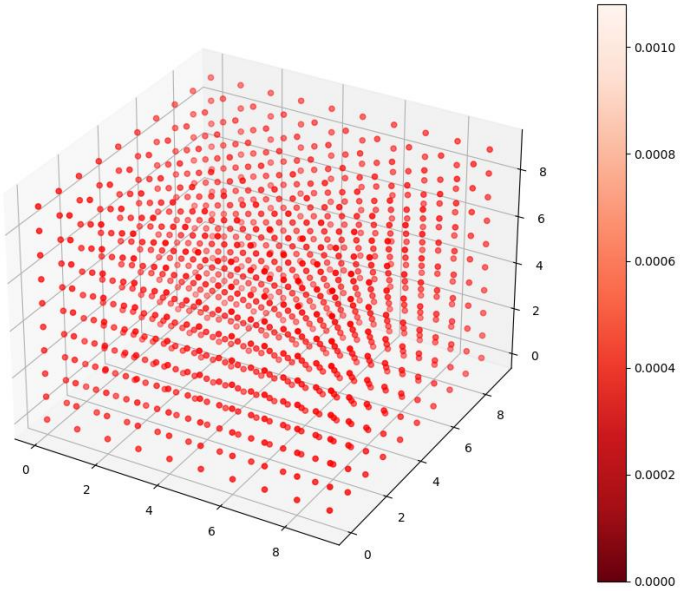


Рисунок 3. График погрешности при $L = 1$ на 20 шаге по времени на сетке 10^3.

Сетка	MPI-процессы	Время работы (с)	Ускорение	Ошибка
128^3	1	0,93	1	$7 * 10^{-8}$
	4	0,25	3,72	$7 * 10^{-8}$
	8	0,19	4,89	$7 * 10^{-8}$
	16	0,18	5,1	$7 * 10^{-8}$
	32	0,17	5,5	$7 * 10^{-8}$
256^3	1	7,5	1	$2 * 10^{-8}$
	4	2	3,8	$2 * 10^{-8}$
	8	1	7,5	$2 * 10^{-8}$
	16	0,84	8,9	$2 * 10^{-8}$
	32	0,62	12	$2 * 10^{-8}$
512^3	1	61	1	$4 * 10^{-9}$
	4	15,8	3,9	$4 * 10^{-9}$
	8	8	7,6	$4 * 10^{-9}$
	16	4,2	14,5	$4 * 10^{-9}$
	32	3,8	16	$4 * 10^{-9}$

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

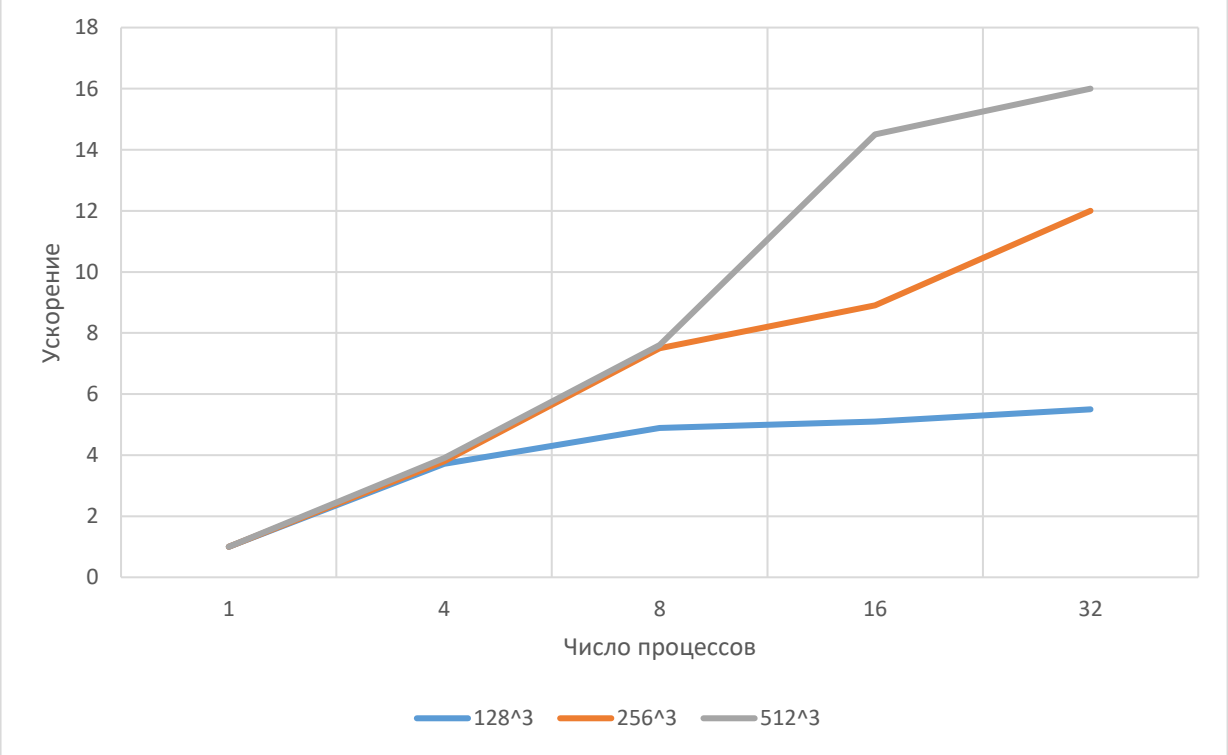


Рисунок 4. График ускорения MPI программы при $L = 1$.

Сетка	MPI-процессы	Время работы (с)	Ускорение	Ошибка
128^3	1	0,94	1	$7 * 10^{-9}$
	4	0,23	4,1	$7 * 10^{-9}$
	8	0,18	5,2	$7 * 10^{-9}$
	16	0,17	5,5	$7 * 10^{-9}$
	32	0,16	5,9	$7 * 10^{-9}$
256^3	1	6,9	1	$2 * 10^{-9}$
	4	1,9	3,6	$2 * 10^{-9}$
	8	1,1	6,2	$2 * 10^{-9}$
	16	0,89	7,8	$2 * 10^{-9}$
	32	0,64	11	$2 * 10^{-9}$
512^3	1	58	1	$4 * 10^{-10}$
	4	15,1	3,8	$4 * 10^{-10}$
	8	10,4	5,6	$4 * 10^{-10}$
	16	3,9	15	$4 * 10^{-10}$
	32	3,3	17,5	$4 * 10^{-10}$

Таблица 2. Результаты исследования MPI программы при $L = \pi$.

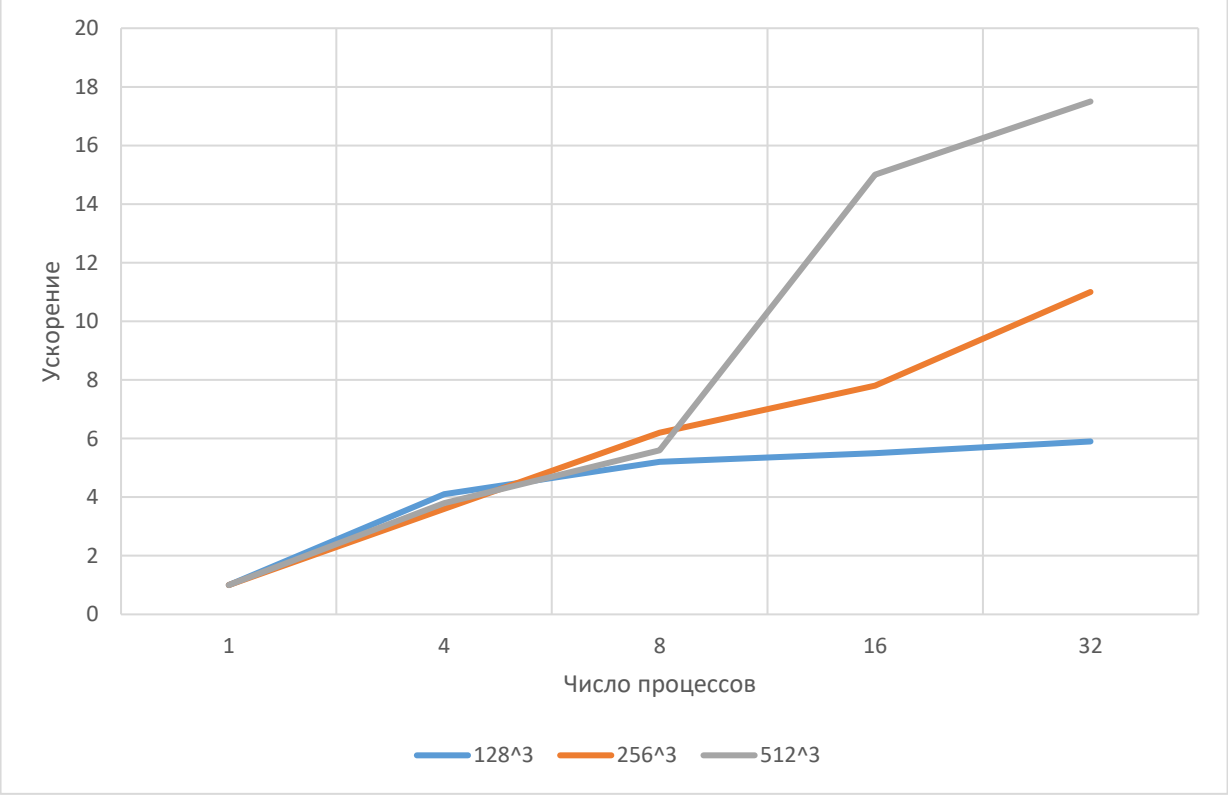


Рисунок 5. График ускорения MPI программы при $L = \pi$.

Сетка	MPI-процессы	Время итерации основного блока (с)	Время работы (с)	Ускорение	Ошибка
128^3	Послед.	-	0,93	1	$7 * 10^{-8}$
	1	0,02	0,37	2,5	$7 * 10^{-8}$
	2	0,009	0,23	4	$7 * 10^{-8}$
	4	0,005	0,14	6,6	$7 * 10^{-8}$
	8	0,002	0,11	8,45	$7 * 10^{-8}$
256^3	Послед.	-	7,5	1	$2 * 10^{-8}$
	1	0,07	2,32	3,23	$2 * 10^{-8}$
	2	0,03	1,26	6	$2 * 10^{-8}$
	4	0,02	0,7	10,7	$2 * 10^{-8}$
	8	0,02	0,69	10,9	$2 * 10^{-8}$
512^3	Послед.	-	61	1	$4 * 10^{-9}$
	1	0,7	19,5	3,1	$4 * 10^{-9}$
	2	0,4	10	6,1	$4 * 10^{-9}$
	4	0,25	5,35	11,4	$4 * 10^{-9}$
	8	0,2	4,34	14	$4 * 10^{-9}$

Таблица 3. Результаты исследования MPI/OpenMP (число нитей на ядро = 4) программы при $L = 1$.

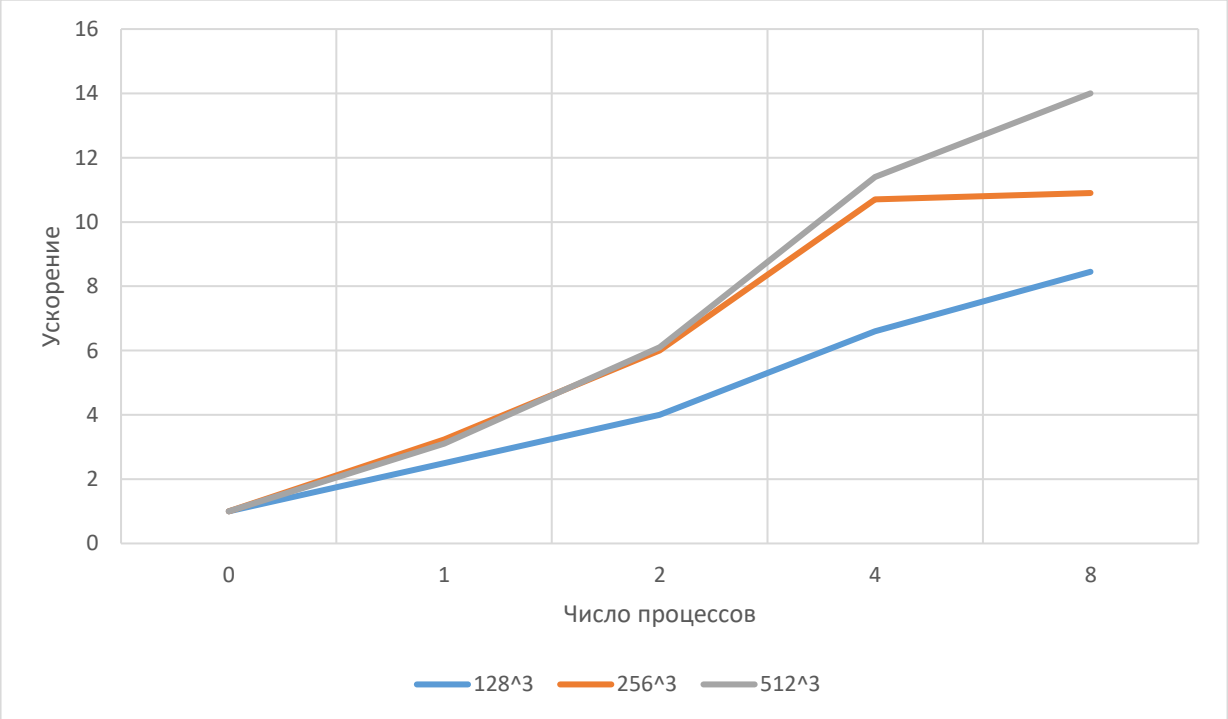


Рисунок 6. График ускорения MPI/OpenMP программы при $L = 1$.

Сетка	MPI-процессы	Время итерации основного блока (с)	Время работы (с)	Ускорение	Ошибка
128^3	Послед.	-	0,94	1	$4 * 10^{-9}$
	1	0,02	0,41	2,3	$7 * 10^{-9}$
	2	0,006	0,24	4	$7 * 10^{-9}$
	4	0,007	0,19	4,9	$7 * 10^{-9}$
	8	0,004	0,1	9,4	$7 * 10^{-9}$
256^3	Послед.	-	6,9	1	$4 * 10^{-9}$
	1	0,08	2,37	2,9	$2 * 10^{-9}$
	2	0,04	1,35	5,1	$2 * 10^{-9}$
	4	0,02	0,67	10,3	$2 * 10^{-9}$
	8	0,03	0,6	11,5	$2 * 10^{-9}$
512^3	Послед.	-	58	1	$4 * 10^{-9}$
	1	0,7	19,3	3	$4 * 10^{-10}$
	2	0,4	10	5,8	$4 * 10^{-10}$
	4	0,28	5,4	10,7	$4 * 10^{-10}$
	8	0,15	4,5	12,9	$4 * 10^{-10}$

Таблица 4. Результаты исследования MPI/OpenMP (число нитей на ядро = 4) программы при $L = \pi$.

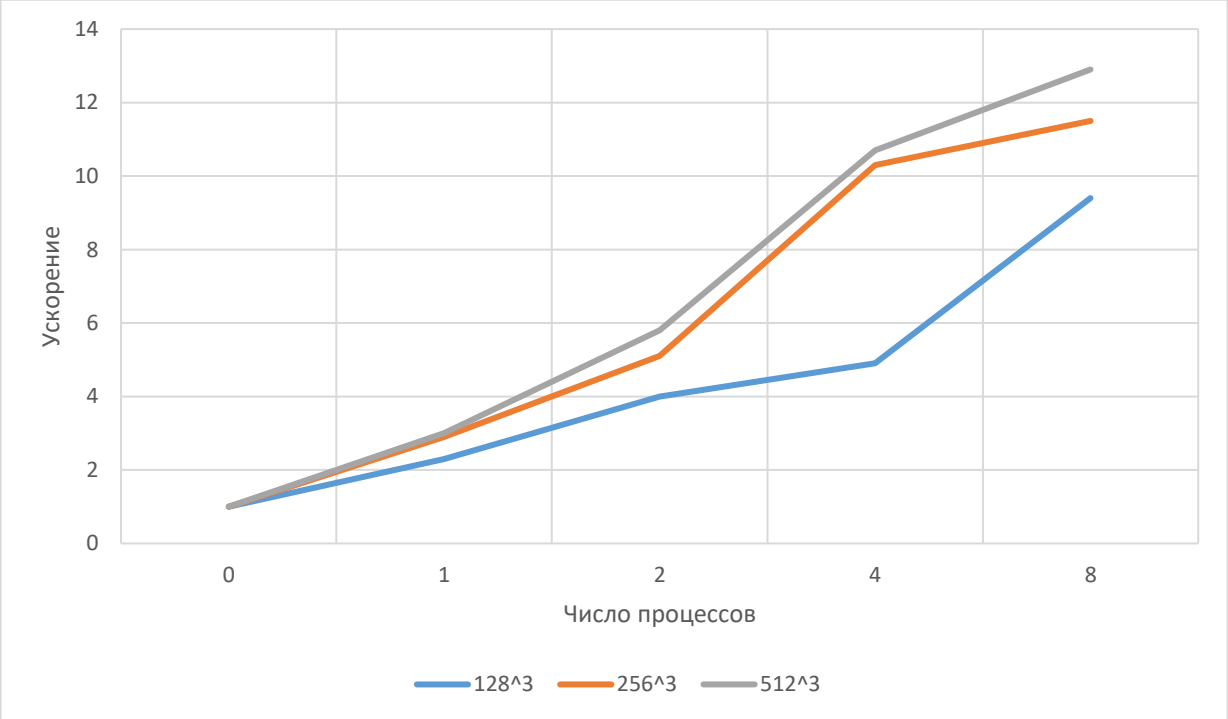


Рисунок 7. График ускорения MPI/OpenMP программы при $L = \pi$.

Сетка	MPI-процессы	Время передачи данных (с)	Время основного цикла for (с)	Время итерации основного блока (с)	Время работы (с)	Ускорение
128^3	Послед.	-	-	-	0,93	1
	1	0,006	0,0003	0,007	0,19	4,9
	2	0,004	0,0002	0,004	0,2	4,65
	4	0,003	0,0002	0,004	0,22	4,2
	6	0,002	0,0002	0,003	0,25	3,7
256^3	Послед.	-	-	-	7,5	1
	1	0,05	0,002	0,05	1	7,5
	2	0,03	0,001	0,03	0,7	10,7
	4	0,02	0,0008	0,02	0,58	12,9
	6	0,01	0,0006	0,02	0,61	12,3
512^3	Послед.	-	-	-	61	1
	1	0,4	0,01	0,4	8	7,6
	2	0,2	0,01	0,2	4,9	12,4
	4	0,1	0,009	0,16	3,27	18,7
	6	0,1	0,006	0,13	2,9	21

Таблица 5. Результаты исследования MPI/OpenACC программы (по 1 gpu на каждый процесс) при $L = 1$.

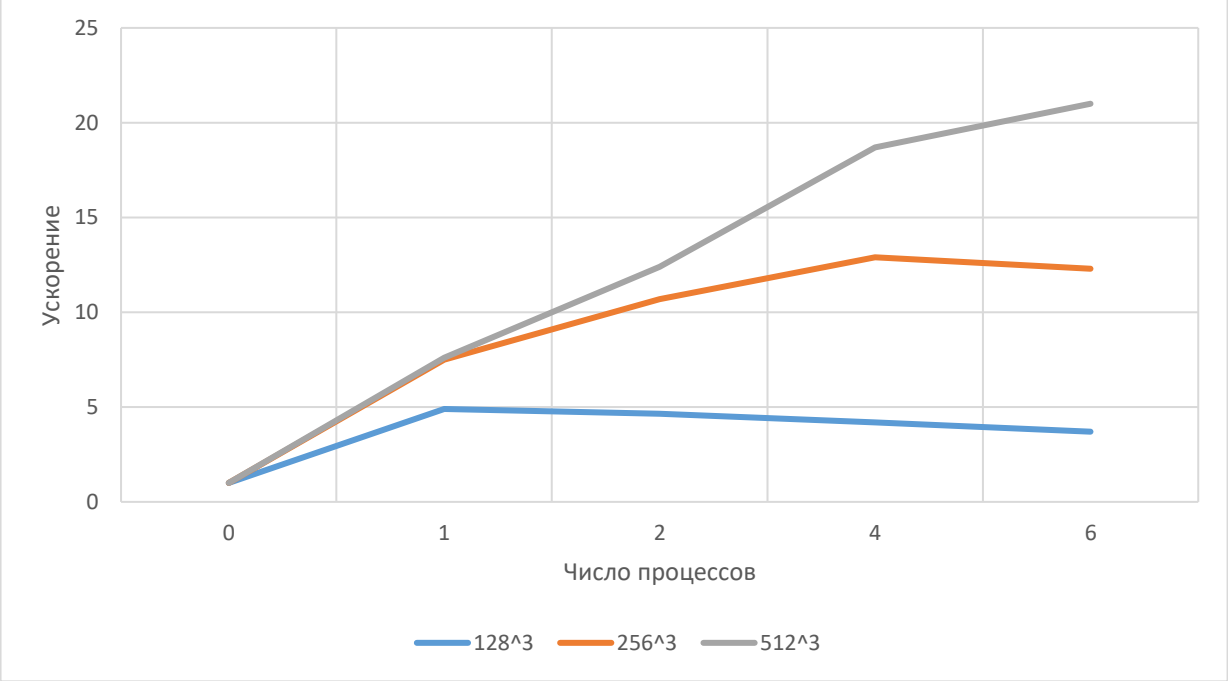


Рисунок 8. График ускорения MPI/OpenACC программы при $L = 1$.

Сетка	MPI-процессы	Время передачи данных (с)	Время основного цикла for (с)	Время итерации основного блока (с)	Время работы (с)	Ускорение
128^3	Послед.	-	-	-	0,94	1
	1	0,006	0,0003	0,007	0,19	4,9
	2	0,004	0,0002	0,004	0,23	4
	4	0,003	0,0002	0,004	0,2	4,7
	6	0,002	0,0002	0,003	0,25	3,7
256^3	Послед.	-	-	-	6,9	1
	1	0,05	0,002	0,05	1	6,9
	2	0,03	0,001	0,03	0,76	9,1
	4	0,02	0,0008	0,02	0,58	11,9
	6	0,01	0,0006	0,02	0,65	10,6
512^3	Послед.	-	-	-	58	1
	1	0,4	0,01	0,4	8,4	6,9
	2	0,2	0,01	0,2	4,9	11,8
	4	0,1	0,009	0,16	3,17	18,3
	6	0,1	0,006	0,13	2,9	21

Таблица 6. Результаты исследования MPI/OpenACC программы (по 1 gpu на каждый процесс) при $L = \pi$.

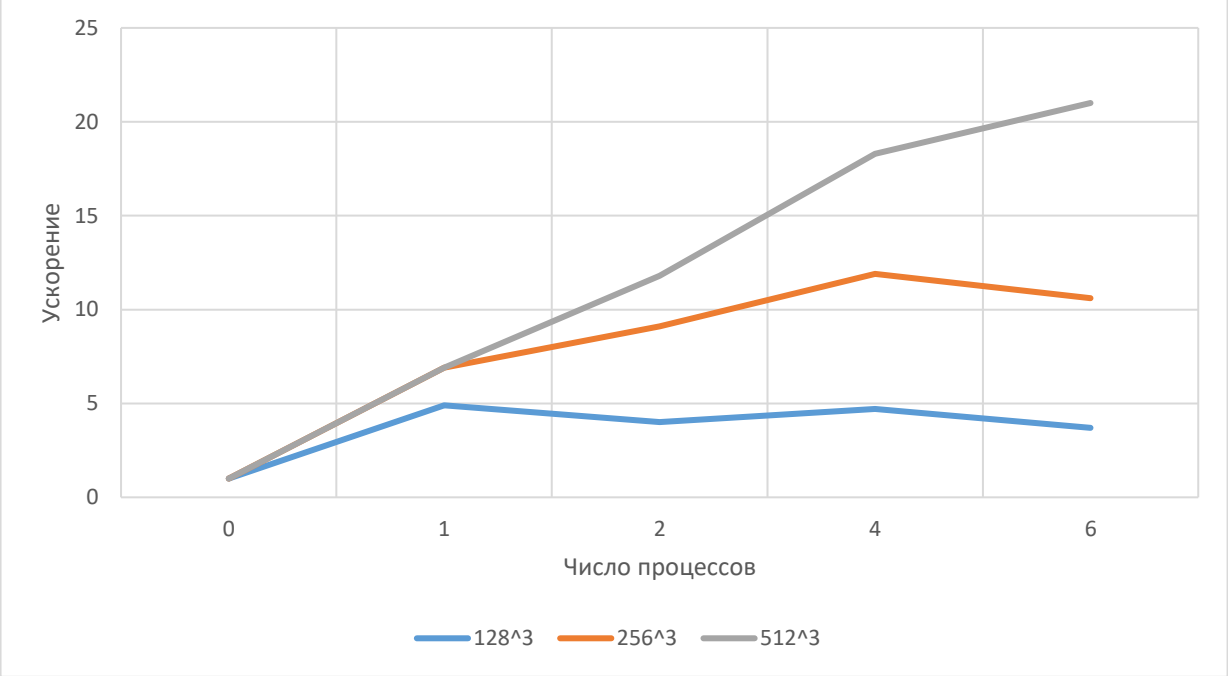
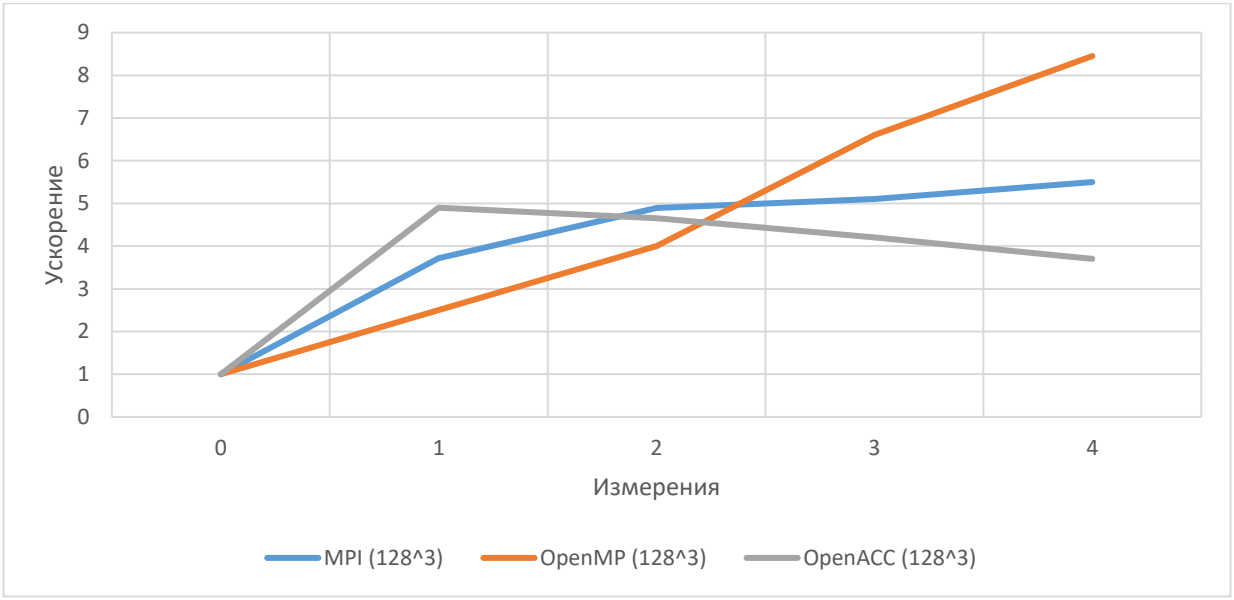
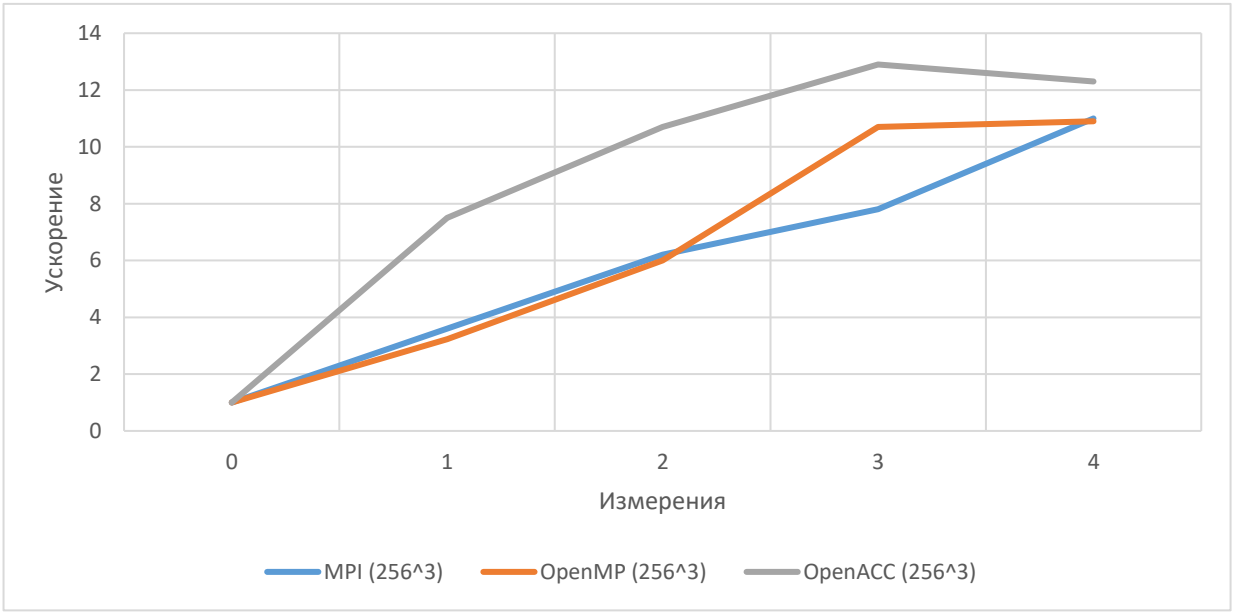


Рисунок 9. График ускорения MPI/OpenACC программы при $L = \pi$.





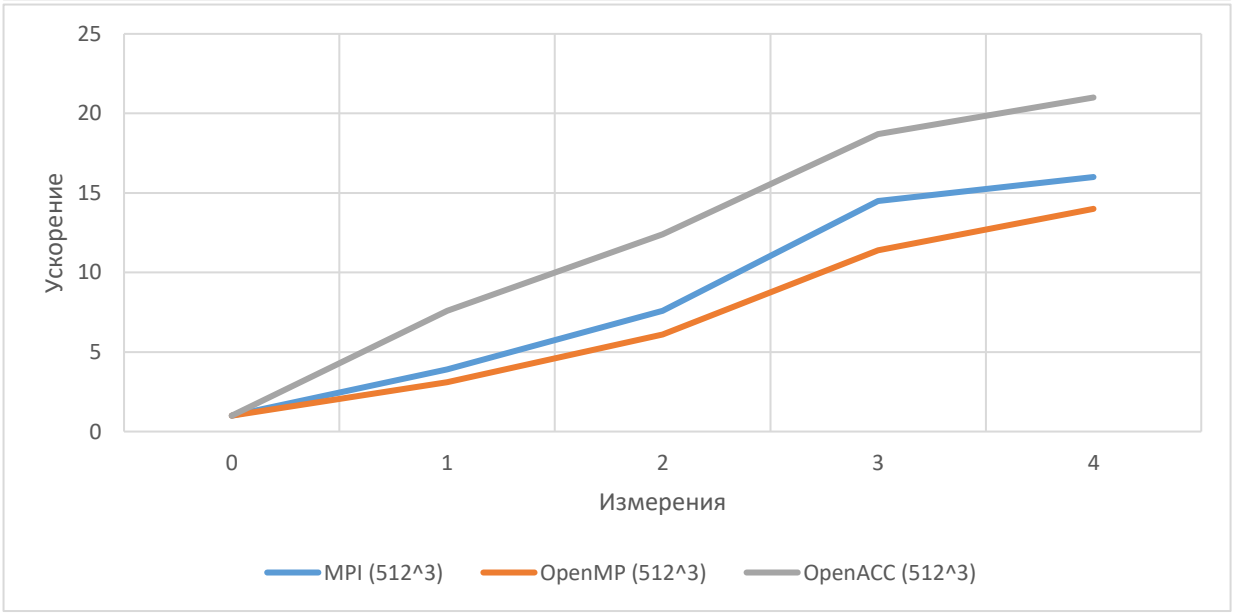


Рисунок 10-12. Ускорения всех программ на всех сетках при $L = 1$.

5. Выводы

В ходе исследования были полученные ожидаемые результаты:

1) Все 3 варианта программы решают задачу примерно за одинаковое время. Различия в программах – в количестве используемых ресурсов.
2) MPI довольно просто и быстро позволяет ускорить программу. Все упирается в достаточно большое количество ядер и хорошую топологию между ними.
3) MPI/OpenMP показывает отличные результаты, однако быстро упирается в пропускную способность кеша и начинает часто ловить cachemiss, от чего эффективность ускорения падает.
4) MPI/OpenACC имеет самую быструю скорость вычисления итерации, однако пересылки данных между CPU и GPU на маленьких объемах данных занимают очень много времени, от чего эффективно обрабатывать маленькие сетки трудно (особенно в данной задаче, так как на каждой итерации нужно обмениваться данными).