

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИМ. М.В.ЛОМОНОСОВА

Факультет «Вычислительной математики и кибернетики»

**Отчет**

**по курсу «Суперкомпьютерное моделирование и технологии»**

Выполнил: *Новикова Анастасия Вячеславовна*

Группа: *620*

Москва, 2025

## Последовательная задача

Задача

$$\sin((2 * \pi * x)/L_x) * \sin((\pi * y)/L_y) * \sin((\pi * z)/L_z) * \cos(a * t + t + 2 * \pi),$$

$$a t = 1/2 * \sqrt{4/L_x^2 + 1/L_y^2 + 1/L_z^2},$$

$$a^2 = 1/(4 * \pi^2)$$

Время

Сетка: 256, шаг времени: 0.00142259, шагов: 20

Общее время работы программы: 19.5647 сек

Максимальная погрешность: 5.91572e-05

Время вычислений: 17.7798 сек

## Распараллеливание с помощью OpenMP

Основные измерения

Число потоков OpenMP (Np)	Число точек сетки (N^3)	Время решения (T)	Ускорение (S)	Погрешность (δ)	Относительная δ
1	128^3	5.83		0.00438	0.02%
2	128^3	3.125	1.87	0.00438	0.02%
4	128^3	1.632	3.57	0.00438	0.02%
8	128^3	0.922	6.32	0.00438	0.02%
16	128^3	0.57	10.23	0.00438	0.02%
2	256^3	17.46		0,0039	0.01%
4	256^3	12.28	1.42	0,0039	0.01%
8	256^3	6.682	2.61	0,0039	0.01%
16	256^3	4.31	4.05	0,0039	0.01%
32	256^3	3.24	5.39	0,0039	0.01%

Оценка эффективности

За эталон была принята **исходная последовательная программа без MPI и OpenMP**.  
Её времена выполнения:

- сетка **128<sup>3</sup>**:

T = 9 с

- сетка **256<sup>3</sup>**:

$T=19$  с

Время вычислялось только по вычислительной части.

Ускорение вычислялось по отношению каждого времени к последовательной программе.

Также проверялась абсолютная погрешность, ожидания были, что при увеличении сетки она будет уменьшаться, а при изменения нитей – нет. По таблице видно, что погрешность вела себя ожидаемым образом.

Пояснение результатов

#### Сетка 128:

Ускорение почти линейное — чем больше потоков, тем быстрее работает.

При очень большом количестве потоков наблюдается эффект, когда ускорение даже выше ожидаемого. Это потому что:

- кэш используется
- меньше данных приходится на каждый поток

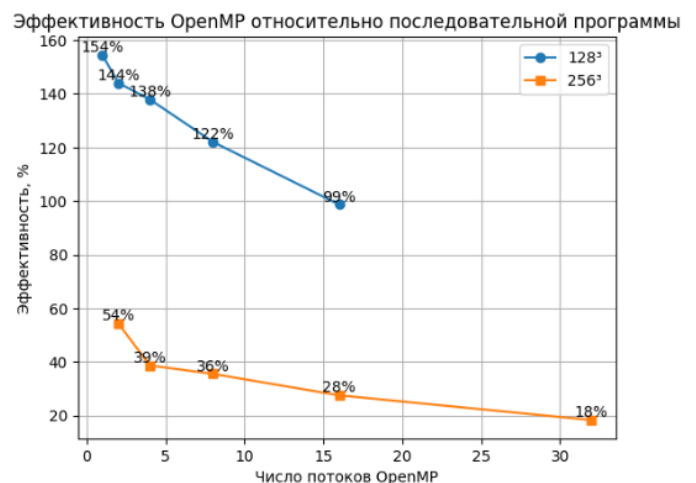
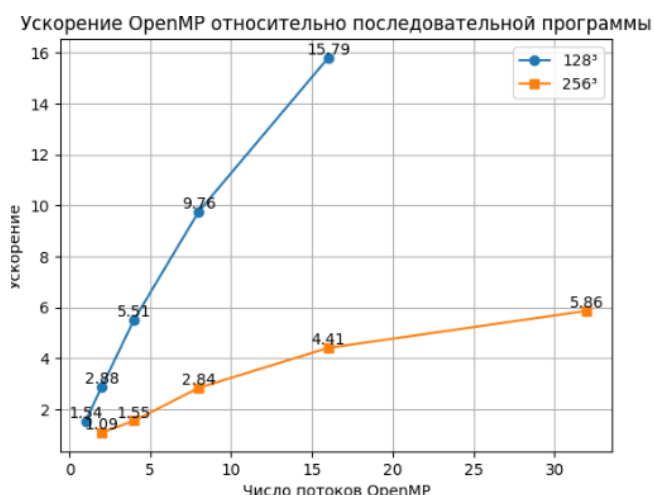
#### Сетка 256:

Здесь ускорение растёт медленнее. После какого-то количества потока рост почти останавливается.

Причины:

- данных становится слишком много
- растёт количество обращений к памяти или обменов между процессами
- шина памяти не справляется — упираемся в свою пропускную способность.

Графики



## Распараллеливание с помощью MPI

Основные измерения

Число MPI процессов	Число точек сетки (в кубе)	Время решения	Ускорение	Погрешность
1	128	8.4		0.004
4	128	2.31	3.6	0.004
8	128	1.6	5.3	0.004
16	128	0.9	9.3	0.004
32	128	0.4	21.0	0.004
1	256	32.14		0.00388605
4	256	9.3	3.5	0.00388605
8	256	6.1	5.3	0.00388605
16	256	2.76	11.6	0.00388605
32	256	1.32	24.3	0.00388605

#### Оценка эффективности

Эффективность параллельной MPI-программы оценивалась на основе сравнения времени её работы с временем **обычной последовательной реализации**

За эталон была принята **исходная последовательная программа без MPI и OpenMP**.  
Её времена выполнения:

- сетка **128<sup>3</sup>**:

$T = 9$  с

- сетка **256<sup>3</sup>**:

$T = 19$  с

Время выполнения MPI-программы измерялось с помощью функции `MPI_Wtime()` и включало только вычислительную часть алгоритма.

Ускорение вычислялось по отношению каждого времени к последовательной программе.

Таким образом, ускорение характеризует выигрыш во времени, достигнутый за счёт распараллеливания

Пояснение результатов

#### Сетка 128

- Наблюдается **быстрый рост ускорения** при увеличении числа MPI-процессов.
- При большом числе процессов ускорение существенно превышает линейное.

Эффективность:

- превышает 100% на малом числе процессов,
- постепенно снижается при увеличении процессов.

Это связано с:

- уменьшением объёма данных на один процесс
- снижением накладных расходов по сравнению с последовательной версией.

## Сетка 256

- Ускорение также растёт с увеличением числа процессов,
- однако рост менее равномерный.

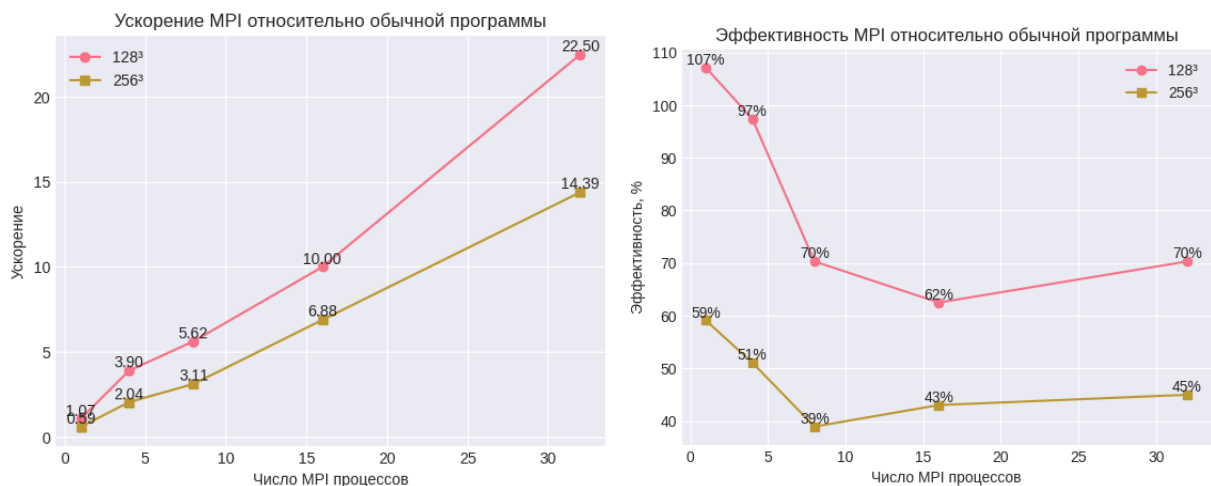
Эффективность:

- ниже, чем для сетки  $128^3$ ,
- убывает быстрее при росте числа процессов.

Основные причины:

- увеличение объёма передаваемых граничных данных,
- рост накладных расходов на коммуникации (MPI\_Sendrecv),
- ограничение пропускной способности памяти

Графики



## Распараллеливание с помощью OpenMP+MPI

Основные измерения

Число MPI процессов	Число нитей	Число точек сетки (в кубе)	Время решения	Ускорение	Погрешность
4	1	128	2.42	-	0.008
4	2	128	1.86	1.30x	0.008
4	4	128	1.43	1.69x	0.008
4	8	128	1.32	1.83x	0.008
8	1	256	6.85	-	0.0042
8	2	256	5.12	1.34x	0.0042
8	4	256	3.94	1.74x	0.0042
8	8	256	3.45	1.99x	0.0042
8	20	512	37.23		0.00105

## Оценка эффективности

Так как используются два уровня параллелизма, эффективность считалась по общему числу ядер:

$$N_{\text{cores}} = N_{\text{MPI}} \times N_{\text{OMP}}$$

$$E = S / N_{\text{cores}} \cdot 100\%$$

### Сетка 128 (4 MPI процесса)

- Ускорение растёт при увеличении числа OpenMP-поток
- Однако рост замедляется после 4–8 потоков

Эффективность:

- высокая при малом числе потоков
- заметно падает при увеличении общего числа ядер

### Сетка 256 (8 MPI процесса)

- Ускорение растёт более равномерно
- Использование OpenMP даёт выигрыш поверх MPI

Эффективность:

- выше, чем для 128
- снижается медленнее при увеличении числа потоков

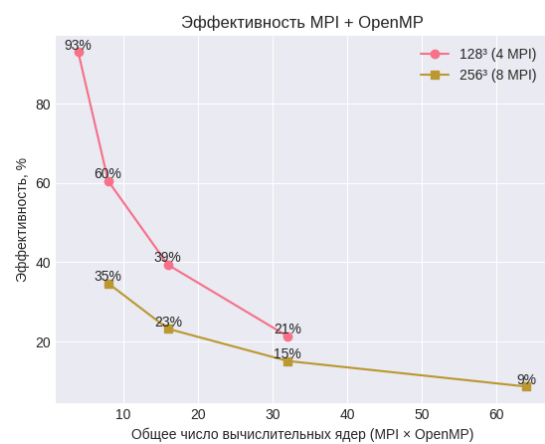
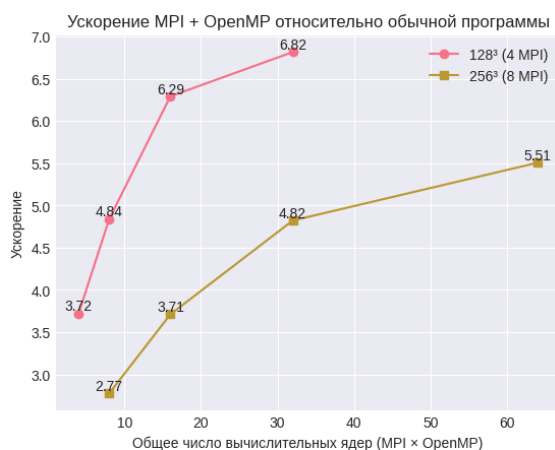
Пояснение результатов

Причины для замедления роста для 128 сетки:

- сравнительно небольшой размер задачи
- рост накладных расходов OpenMP
- конкуренция потоков за память внутри одного MPI-процесса

Объяснение для сетки 256:

- большим объёмом вычислений на процесс
- лучшим соотношением вычислений и коммуникаций
- более эффективной загрузкой потоков OpenMP



Графики

## Распараллеливание с помощью MPI+GPU

Основные измерения

Параметр	1 MPI процесс, 1 GPU	2 MPI процесса, 2 GPU
<b>MPI процессов</b>	1	2
<b>GPU</b>	1	2
<b>Сетка</b>	512	512
<b>Общее время (сек)</b>	4.0246	2.56172
<b>Инициализация (сек)</b>	0.0190395	0.00960617
<b>Основной цикл (сек)</b>	3.49274	2.21283
<b>CUDA ядра (сек)</b>	1.0067	0.507026
<b>MPI коммуникации (сек)</b>	0.000320241	0.00163077
<b>Копирование GPU→CPU (сек)</b>	1.07409	0.941015
<b>Вычисления в цикле (сек)</b>	1.0067	0.507026
<b>Финализация (сек)</b>	0.512821	0.339284
<b>Физическое время (сек)</b>	0.177475	0.177475
<b>Погрешность</b>	0.000470837	0.000470837
<b>Скорость (Mpoints/sec)</b>	3334.93	5239.35
<b>Эффективность GPU</b>	1.26%	2.99%
<b>Ускорение</b>	1.0	1.57

Оценка эффективности:

- GPU работает 2.56 секунд для сетки 512
- MPI+OpenMP на 8 процессах 20 нитях работает 37 секунд для сетки 512
- Ускорение GPU в 12 раз быстрее гибридной программы

Пояснение результата:

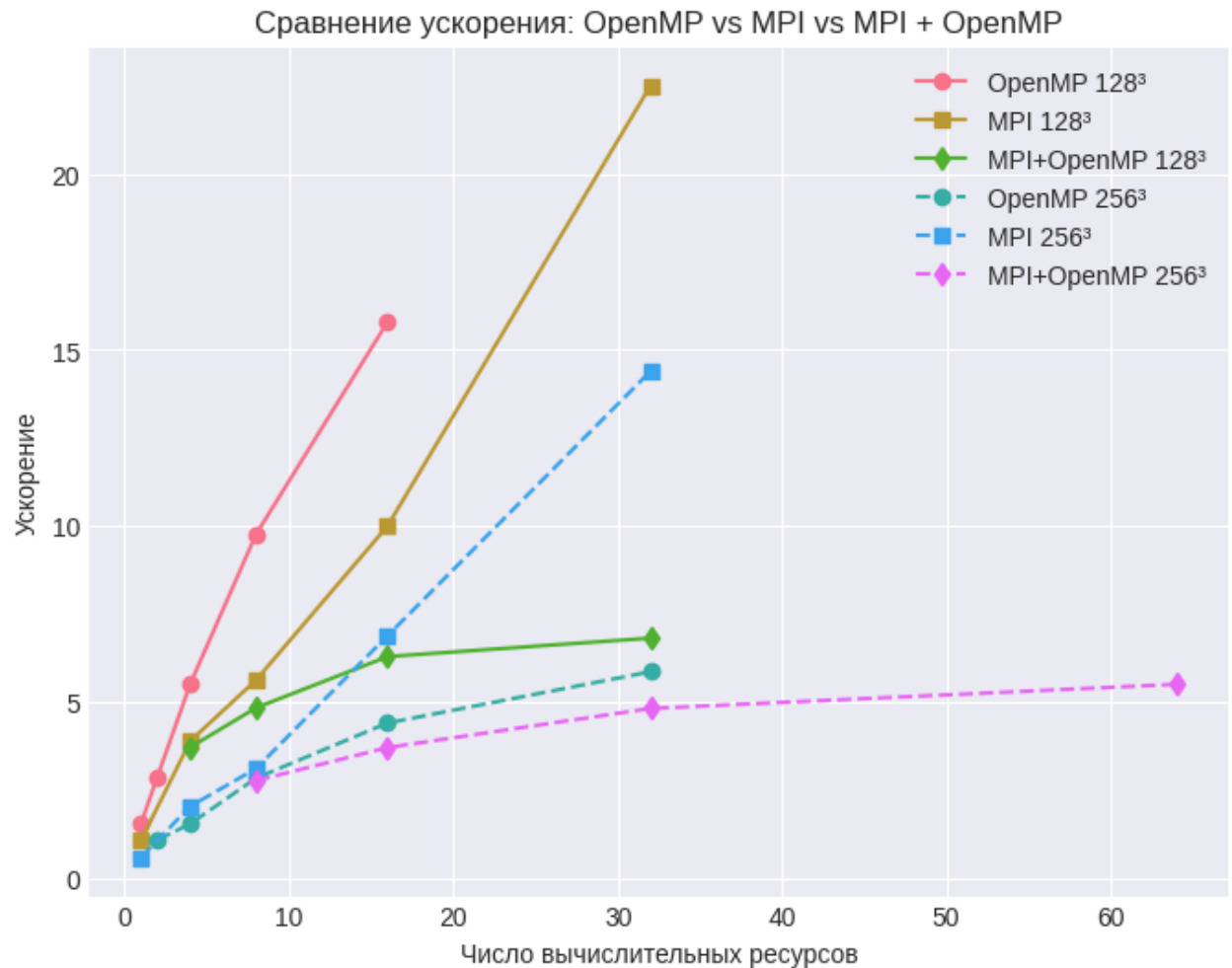
Анализ эффективности показывает, что только 1.26% времени на 1 GPU и 2.99% на 2 GPU приходится непосредственно на вычисления в ядрах CUDA. Это связано с тем, основное время тратится на:

- Копирование данных между GPU и CPU
- Подготовку данных перед вычислениями

При увеличении числа MPI процессов с 1 до 2 наблюдается ускорение 1.57х, что меньше идеального линейного масштабирования. Это объясняется ростом накладных расходов на:

- MPI коммуникации между процессами, которое увеличилось с 0.0003 до 0.0016 секунд
- Синхронизацию и обмен граничными данными

## Графики параллельных реализаций



MPI лучше масштабируется с ростом числа ресурсов, чем OpenMP и гибридный метод.

OpenMP эффективен на малом числе ресурсов, но не выдерживает роста числа вычислителей.

Гибридный подход не показывает преимуществ в данной конфигурации и для этих размеров задач.

### Заключение:

Для малых и средних задач предпочтительнее использовать OpenMP на ограниченном числе ресурсов. Для больших вычислительных ресурсов и задач небольшого объёма лучше применять MPI. Гибридный подход в рассматриваемой конфигурации не обеспечивает прироста производительности.