

**Национальный исследовательский университет  
Высшая школа экономики  
Московский институт электроники и математики**

Департамент прикладной математики  
кафедра компьютерной безопасности

Лабораторная работа №3  
по  
*Параллельным вычислениям*

Освоение векторизации и распараллеливания программ  
Вариант №1

Выполнил  
Новосильцев Е.Д.

Проверил  
Байдин Г.С.

Москва 2026

# Содержание

<b>Содержание</b>	<b>1</b>
<b>1 Постановка задачи</b>	<b>2</b>
<b>2 Описание используемых ресурсов</b>	<b>3</b>
2.1 Вычислительная система . . . . .	3
2.2 Компилятор . . . . .	3
<b>3 Результаты предыдущих ЛР</b>	<b>4</b>
<b>4 Результаты выполнения задания</b>	<b>5</b>
4.1 Сравнительная таблица . . . . .	6
<b>5 Вывод по результатам выполнения задания</b>	<b>7</b>
<b>6 Репозиторий с кодом</b>	<b>8</b>

# 1 Постановка задачи

1. Оптимизировать программу из ЛР2, выполняющую умножение плотных матриц больших размерностей, с помощью MPI.
2. Сравнить время работы данной оптимизации с вариантами программы из ЛР1 и ЛР2.

## **2 Описание используемых ресурсов**

### **2.1 Вычислительная система**

- Процессор - Apple M4 Pro
- Количество ядер CPU - 12
- Количество ядер GPU - 16
- Тип оперативной памяти - LPDDR5X
- Объем оперативной памяти - 24 ГБ
- Операционная система - macOS Tahoe 26.1

### **2.2 Компилятор**

Apple clang version 17.0.0 (clang-1700.0.13.5)

### **3 Результаты предыдущих ЛР**

- Отчет с описанием результатов ЛР1 доступен по [ссылке](#).
- Отчет с описанием результатов ЛР2 доступен по [ссылке](#).

## 4 Результаты выполнения задания

Для перехода на MPI модель параллелизма была изменена с многопоточной (общая память) на многопроцессную (распределённая память). Хранение матрицы заменено с  $vector<vector<int>>$  на одномерный  $vector<int>$  в row-major порядке, поскольку MPI-функции передачи данных требуют непрерывного блока памяти. Блочный алгоритм умножения сохранён, однако вместо OpenMP-директив распараллеливание реализовано через распределение строк матрицы А между процессами:

1. Процесс 0 генерирует матрицы, рассыпает В целиком всем процессам через  $MPI\_Bcast$ , раздаёт блоки строк А через  $MPI\_Scatterv$ ;
2. Каждый процесс независимо выполняет блочное умножение своей части;
3. Результаты собираются обратно через  $MPI\_Gatherv$ .

Наименьшее время выполнения программы достигается при использовании 8 процессов (`-pr 8`) несмотря на то, что у моего процессора 12 ядер. Это связано с тем, что чип Apple M4 Pro содержит 8 P-ядер (~4.5 ГГц) и 4 E-ядра (~2.5 ГГц). При `-pr 12` четыре процесса попадают на медленные E-ядра и получают тот же объём работы, что и процессы на P-ядрах, однако выполняют его значительно дольше. Поскольку  $MPI\_Gatherv$  является коллективной операцией, все процессы ожидают завершения самого медленного, и общее время определяется именно E-ядрами.

Необходимо отметить, что MPI-версия программы работает медленнее, чем OpenMP-версия с оптимизацией алгоритма из ЛР2 ([`m\_mul\_vect\_2.cpp`](#)). Это обусловлено природой MPI: каждый процесс имеет собственное адресное пространство, поэтому данные необходимо явно копировать между процессами: в каждом эксперименте пересыпается вся матрица В и соответствующие блоки матриц А и С, тогда как OpenMP-потоки работают с общей памятью без какого-либо копирования. Кроме того, создание отдельных процессов и межпроцессная синхронизация существенно дороже, чем создание лёгких потоков внутри одного процесса. MPI предназначен для распределённых систем, где общая память недоступна и накладные расходы на коммуникацию компенсируются масштабированием на десятки и сотни узлов.

Полученные результаты:

```

~/Desktop/учеба/5 курс/ПВ/code/lab3

[egorovosilcev@MacBook-Pro-Egor lab3 % mpicxx m_mul_mpi.cpp -O3 -march=native -o m_mul_mpi
[egorovosilcev@MacBook-Pro-Egor lab3 % mpirun -np 8 ./m_mul_mpi
Тестирование умножения матриц 1000x1000 на 1000x1000
Количество MPI-процессов: 8
Количество экспериментов: 10

Эксперимент 1: 0.018 секунд
Эксперимент 2: 0.010 секунд
Эксперимент 3: 0.010 секунд
Эксперимент 4: 0.011 секунд
Эксперимент 5: 0.010 секунд
Эксперимент 6: 0.010 секунд
Эксперимент 7: 0.010 секунд
Эксперимент 8: 0.010 секунд
Эксперимент 9: 0.010 секунд
Эксперимент 10: 0.010 секунд

Среднее время выполнения: 0.011 секунд
egorovosilcev@MacBook-Pro-Egor lab3 %

```

## 4.1 Сравнительная таблица

ЛР, №	Оптимизация	Среднее время, с	Отношение ко времени исходной версии, %	Отношение ко времени предыдущей версии, %
1	-	3.296	-	-
1	Автоматическая	0.770	25	25
1	Полуавтоматическая	0.114	3	15
2	Полуавтоматическая + оптимизация алгоритма	0.005	0.15	4
3	MPI	0.011	0.33	220

## 5 Вывод по результатам выполнения задания

В ходе выполнения лабораторной работы программа умножения матриц была адаптирована для работы с использованием технологии MPI. Реализовано распределение данных между процессами через коллективные операции *MPI\_Bcast*, *MPI\_Scatterv* и *MPI\_Gatherv*, что позволило распараллелить вычисления на уровне процессов с раздельной памятью.

Проведенные эксперименты показали, что на одной машине MPI-версия уступает OpenMP-версии из-за накладных расходов на межпроцессное копирование данных. Однако полученная реализация является основой для масштабирования на распределённые вычислительные системы, где MPI способен обеспечить ускорение, недоступное для OpenMP.

## 6 Репозиторий с кодом

GitHub репозиторий со всеми материалами доступен по [ссылке](#).