

Национальный исследовательский университет
Высшая школа экономики
Московский институт электроники и математики

Департамент прикладной математики
кафедра компьютерной безопасности

Лабораторная работа №2
ПО
Параллельным вычислениям

Освоение векторизации
Вариант №1

Выполнил
Новосильцев Е.Д.

Проверил
Байдин Г.С.

Москва 2025

Содержание

Содержание	1
1 Постановка задачи	2
2 Описание используемых ресурсов	3
2.1 Вычислительная система	3
2.2 Компилятор	3
3 Результаты ЛР1	4
4 Результаты выполнения задания	5
4.1 Сравнительная таблица	5
5 Вывод по результатам выполнения задания	6
6 Репозиторий с кодом	7

1 Постановка задачи

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

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 ЛР1

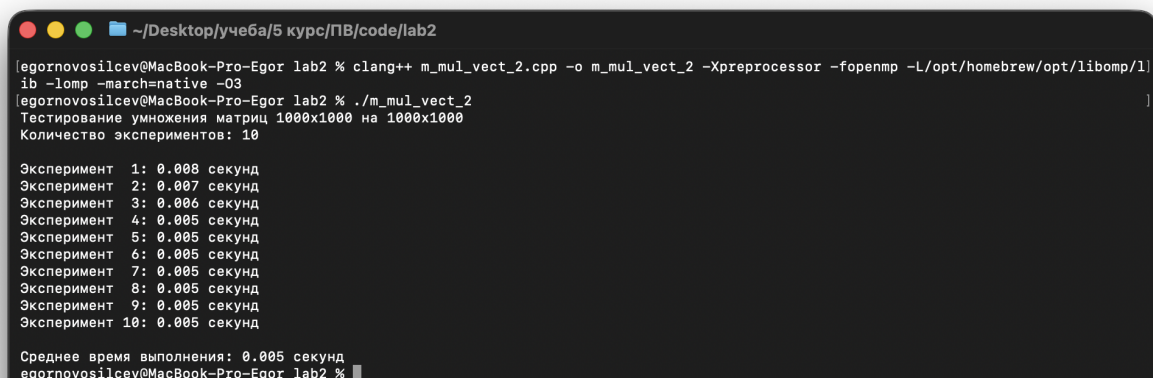
Отчет с описанием результатов ЛР1 доступен по [ссылке](#).

4 Результаты выполнения задания

OpenMP оптимизация в полуавтоматической версии ЛР1 ([m_mul_vect.cpp](#)) с директивами `collapse(2)` + `schedule(guided)` + `simd reduction` оказалась оптимальной для стандартного алгоритма умножения матриц. Попытки применить дополнительные техники, такие как `taskloop`, явный контроль области видимости через `default(none)` и ручное управление количеством потоков, только ухудшили производительность. Причина в том, что для регулярных матричных вычислений с предсказуемой нагрузкой накладные расходы на реализацию данных техник превышают потенциальный выигрыш.

В связи с этим была выполнена оптимизация самого алгоритма умножения: применено блочное умножение с размером блока 64×64 элемента. Данные каждого блока полностью помещаются в L1-кэш процессора и многократно переиспользуются внутри вычислений, что радикально снижает количество `cache misses` и медленных обращений к оперативной памяти. Обновленный блочный алгоритм создает четыре уровня вложенных циклов, что позволило заменить `collapse(2)` на `collapse(4)`, распараллеливая все уровни одновременно, тем самым обеспечивая идеальную загрузку всех потоков.

Полученные результаты:



```
~/Desktop/учеба/5 курс/ПВ/code/lab2
egornovosilcev@MacBook-Pro-Egor lab2 % clang++ m_mul_vect_2.cpp -o m_mul_vect_2 -Xpreprocessor -fopenmp -L/opt/homebrew/opt/libomp/lib -lomp -march=native -O3
egornovosilcev@MacBook-Pro-Egor lab2 % ./m_mul_vect_2
Тестирование умножения матриц 1000x1000 на 1000x1000
Количество экспериментов: 10

Эксперимент 1: 0.008 секунд
Эксперимент 2: 0.007 секунд
Эксперимент 3: 0.006 секунд
Эксперимент 4: 0.005 секунд
Эксперимент 5: 0.005 секунд
Эксперимент 6: 0.005 секунд
Эксперимент 7: 0.005 секунд
Эксперимент 8: 0.005 секунд
Эксперимент 9: 0.005 секунд
Эксперимент 10: 0.005 секунд

Среднее время выполнения: 0.005 секунд
egornovosilcev@MacBook-Pro-Egor lab2 %
```

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

5 Вывод по результатам выполнения задания

В ходе выполнения лабораторной работы полуавтоматическая реализация из ЛР1 была улучшена за счет оптимизации алгоритма через блочное умножение и соответствующего распараллеливания вычислений с использованием директивы `collapse(4)`. Такая комбинация алгоритмической и параллельной оптимизации позволила эффективно задействовать все доступные вычислительные ресурсы.

Проведенные эксперименты показали значительное ускорение оптимизированной версии программы по сравнению с предыдущей реализацией.

6 Репозиторий с кодом

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