

Национальный исследовательский университет
Высшая школа экономики
Московский институт электроники и математики

Департамент прикладной математики
кафедра компьютерной безопасности

Лабораторная работа №1
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 выполнения задания	4
3.1 Исходная программа без оптимизации	4
3.2 Автоматическая оптимизация	4
3.3 Полуавтоматическая оптимизация	5
3.4 Сравнительная таблица	6
4 Вывод по результатам выполнения задания	7
5 Репозиторий с кодом	8

1 Постановка задачи

1. Реализовать на языке C++ программу, выполняющую умножение плотных матриц больших размерностей.
2. Оптимизировать данную программу следующими способами:
 - Автоматически (только с помощью ключей компилятора)
 - Полуавтоматически (с помощью OpenMP и ключей компилятора)
3. Сравнить время работы следующих вариантов программы:
 - программа без оптимизации
 - автоматически оптимизированная программа
 - полуавтоматически оптимизированная программа

2 Описание используемых ресурсов

2.1 Вычислительная система

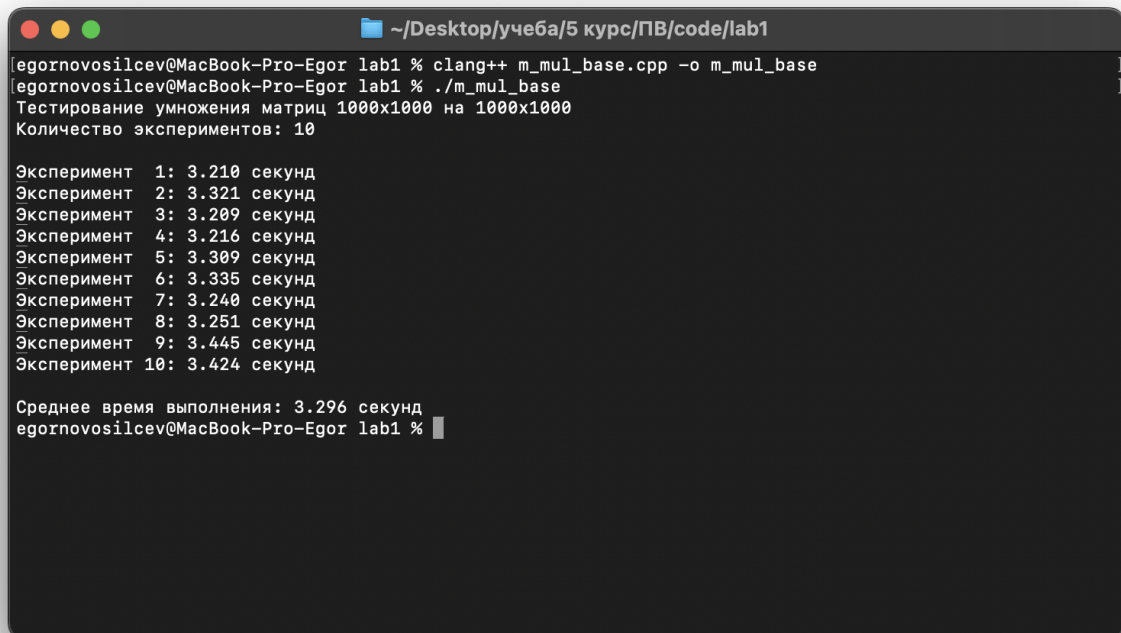
- Процессор - Apple M4 Pro
- Количество ядер CPU - 12
- Количество ядер GPU - 16
- Тип оперативной памяти - LPDDR5X
- Объем оперативной памяти - 24 ГБ
- Операционная система - macOS Sequoia 15.6.1

2.2 Компилятор

Apple clang version 17.0.0 (clang-1700.0.13.5)

3 Результаты выполнения задания

3.1 Исходная программа без оптимизации



```
egornovosilcev@MacBook-Pro-Egor lab1 % clang++ m_mul_base.cpp -o m_mul_base
egornovosilcev@MacBook-Pro-Egor lab1 % ./m_mul_base
Тестирование умножения матриц 1000x1000 на 1000x1000
Количество экспериментов: 10

Эксперимент 1: 3.210 секунд
Эксперимент 2: 3.321 секунд
Эксперимент 3: 3.209 секунд
Эксперимент 4: 3.216 секунд
Эксперимент 5: 3.309 секунд
Эксперимент 6: 3.335 секунд
Эксперимент 7: 3.240 секунд
Эксперимент 8: 3.251 секунд
Эксперимент 9: 3.445 секунд
Эксперимент 10: 3.424 секунд

Среднее время выполнения: 3.296 секунд
egornovosilcev@MacBook-Pro-Egor lab1 %
```

3.2 Автоматическая оптимизация

На данном этапе для увеличения быстродействия программы были использованы следующие ключи оптимизации компилятора:

- *-march=native* - оптимизация кода под конкретный процессор, архитектура определяется автоматически
- *-O3* - максимальный (агрессивный) уровень оптимизации

Полученные результаты:

```
~/Desktop/учеба/5 курс/ПВ/code/lab1
egornovosilcev@MacBook-Pro-Egor lab1 % clang++ m_mul_base.cpp -o m_mul_comp_opt -march=native -O3
egornovosilcev@MacBook-Pro-Egor lab1 % ./m_mul_comp_opt
Тестирование умножения матриц 1000x1000 на 1000x1000
Количество экспериментов: 10

Эксперимент 1: 0.792 секунд
Эксперимент 2: 0.768 секунд
Эксперимент 3: 0.766 секунд
Эксперимент 4: 0.771 секунд
Эксперимент 5: 0.779 секунд
Эксперимент 6: 0.772 секунд
Эксперимент 7: 0.763 секунд
Эксперимент 8: 0.762 секунд
Эксперимент 9: 0.754 секунд
Эксперимент 10: 0.769 секунд

Среднее время выполнения: 0.770 секунд
egornovosilcev@MacBook-Pro-Egor lab1 %
```

3.3 Полуавтоматическая оптимизация

На данном этапе для увеличения быстродействия программы были использованы следующие директивы OpenMP:

- `#pragma omp parallel for collapse(2) schedule(guided)`
 - Задается перед началом общего цикла в методе *multiply*
 - *parallel for* - распараллеливание итераций цикла *for* между потоками
 - *collapse(2)* - объединение двух вложенных циклов *for* в один параллельный регион, что улучшает балансировку
 - *schedule(guided)* - определение способа распределения итераций между потоками: размер блока итераций уменьшается экспоненциально
- `#pragma omp simd reduction(+:sum)`
 - Задается перед началом третьего вложенного цикла в методе *multiply*
 - *simd* - включение векторизации цикла (использование SIMD-инструкций процессора)
 - *reduction(+:sum)* - указание на то, что переменная *sum* участвует в операции редукции (гарантия корректности параллельных вычислений)

Кроме того, были использованы следующие ключи компилятора:

- `-Xpreprocessor -fopenmp -L/opt/homebrew/opt/libomp/lib -lomp` - включение поддержки OpenMP (аналог `-fopenmp` в компиляторе GCC)
- `-march=native -O3` - аналогично пункту 3.2

Полученные результаты:

```

egornovosilcev@MacBook-Pro-Egor lab1 % clang++ m_mul_vect.cpp -o m_mul_vect -Xpreprocessor -fopenmp -L/
opt/homebrew/opt/libomp/lib -lomp -march=native -O3
egornovosilcev@MacBook-Pro-Egor lab1 % ./m_mul_vect
Тестирование умножения матриц 1000x1000 на 1000x1000
Количество экспериментов: 10

Эксперимент 1: 0.126 секунд
Эксперимент 2: 0.116 секунд
Эксперимент 3: 0.111 секунд
Эксперимент 4: 0.111 секунд
Эксперимент 5: 0.115 секунд
Эксперимент 6: 0.115 секунд
Эксперимент 7: 0.109 секунд
Эксперимент 8: 0.111 секунд
Эксперимент 9: 0.111 секунд
Эксперимент 10: 0.116 секунд

Среднее время выполнения: 0.114 секунд
egornovosilcev@MacBook-Pro-Egor lab1 %
  
```

3.4 Сравнительная таблица

Оптимизация	Среднее время, с	Отношение ко времени исходной версии, %	Отношение ко времени предыдущей версии, %
-	3.296	-	-
Автоматическая	0.770	25	25
Полуавтоматическая	0.114	3	15

4 Вывод по результатам выполнения задания

В ходе выполнения лабораторной работы была разработана программа умножения матриц и проведена её оптимизация. Базовая реализация была улучшена за счет применения двух основных подходов: распараллеливания вычислений на несколько потоков и использования векторных операций процессора. Такая комбинация методов оптимизации позволила эффективно задействовать все доступные вычислительные ресурсы процессора.

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

5 Репозиторий с кодом

GitHub репозиторий со всеми материалами доступен по [ссылке](#). Краткое описание содержимого:

- *m_mul_base.cpp* - исходная программа без оптимизации
- *m_mul_vect.cpp* - полуавтоматически оптимизированная программа
- *m_mul_vect.asm* - ассемблерный листинг файла *m_mul_vect.cpp*
- *vect_multiply.asm* - фрагмент *m_mul_vect.asm*, содержащий листинг метода *multiply*