Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

# ЛАБОРАТОРНАЯ РАБОТА №5

**Выбор оптимальной опции оптимизации**
по дисциплине: «Основы разработки программного обеспечения»

Выполнил
студент гр.30030/2х В.Ю.Сподынейко

Руководитель
доцент, к.т.н А.В.Петров

Санкт-Петербург
2023

# Содержание

# 1 Задание

Цель работы – выбор опции оптимизации, оптимальные для вашего приложения.

## 1.1 Выбор подходящего уровня оптимизации

На основе примера, демонстрирующего различные уровни оптимизации, написать первый сценарий, выполняющие следующие действия в *цикле*:

- Компиляцию **вашего приложения, не интерактивно обрабатывающего данные,** на языке C/C++/Fortran/Objective/Objective C++/Ada с ключами оптимизации:
  - -O0
  - -Os
  - -O1
  - -O2
  - -O3
  - -O2 -march=native
  - -O3 -march=native
  - -O2 -march=native -funroll-loops
  - -O3 -march=native -funroll-loops
- Вычисление времени выполнения программы (time). Приложение без оптимизации должно работать по меньшей мере 20 с.
- Вычисление занимаемого исполняемым файлом дискового пространства (в байтах) (du).
- Сценарий должен принимать только имя исходного файла программы. Вывод сценария должен содержать следующую информацию:
- Текущие опции оптимизации.
- Время затраченное программой на выполнение.
- Занимаемое программой дисковое пространство.

## 1.2 Выбор системного метода оптимизации

Выберите вариант оптимизации, дающий наибольшую производительность для вашего приложения (оптимальная опция).

Проведите оптимизацию с оптимальной опцией, межпроцедурной оптимизацией[1] (см. серию опций -fipa-*) и оптимизацией времени компоновки[2] (-flto). Определите время работы приложения.

Проведите оптимизацию с оптимальной опцией и с оптимизацией с обратной связью (-fprofile-generate/-fprofile-use). Определите время работы приложения.

Проведите оптимизацию с оптимальной опцией, межпроцедурной оптимизацией, оптимизацией времени компоновки и с оптимизацией с обратной связью. Определите время работы приложения.

[1] interprocedural analysis
[2] link-time optimization

### 1.3 Отчётность

Подготовить в электронной форме документ (не отчёт), содержащий таблицу со всеми используемыми вариантами оптимизации и временем работы при них. Сделать вывод по оптимальной стратегии оптимизации вашего приложения.

## 2 Основная часть

### 2.1 Особенности реализации

Основная программа перемножает две прямоугольные матрицы заданного размера: $A(100, 50)$ и $B(50, 20)$.
Элементы результирующей матрицы $C = A \times B$ формируются по правилу:

$$C_{ij} = \sum_{k=1}^{50} a_{ik}\, b_{kj},\ i = 1, 2, \ldots, 100,\ j = 1, 2, \ldots, 20. \tag{1}$$

### 2.2 Описание алгоритма

Исходный код программы размещён в **Приложении А**.
Алгоритм перемножения двух прямоугольных матриц реализован с помощью технологии FORTRAN. Основная логика программы использует вложенные итерирующие циклы (оператор *do*), в которых изменяются индексы *i, j, k* и поочерёдно находятся члены результирующей матрицы $C$, согласно формуле (1). Перебор элементов с индексом $i$ производится с помощью доступа к элементам массивов посредством сечений массивов (array sections)[3].

[3] напр.: $A(:, k)$

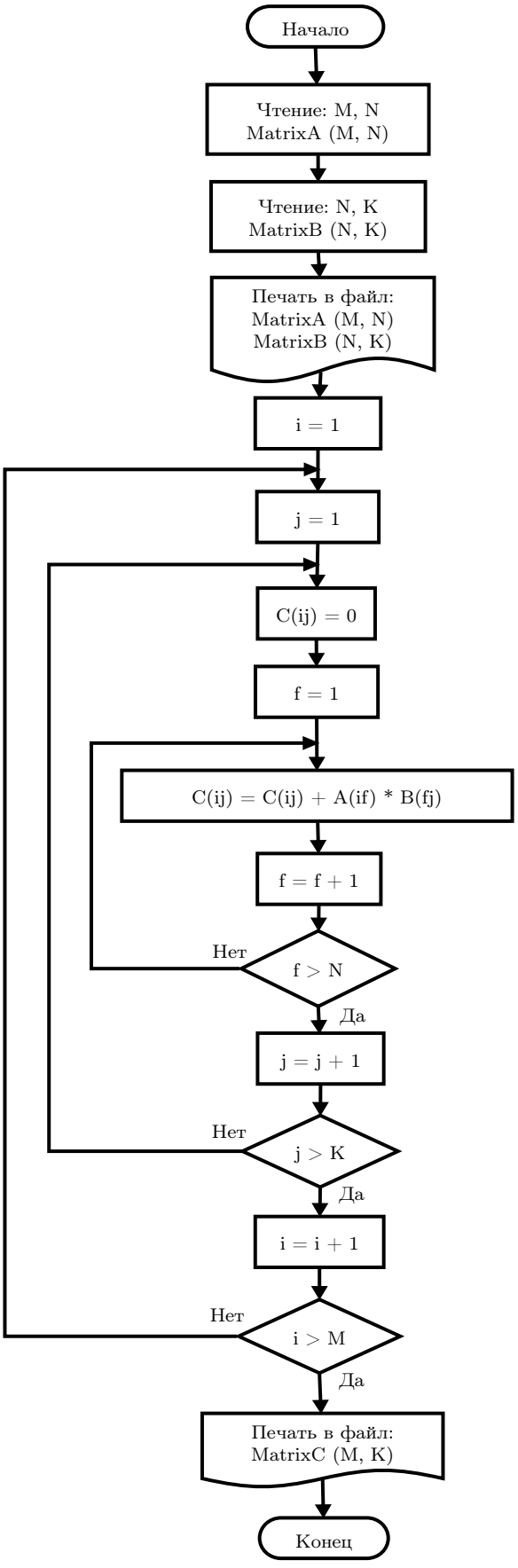


Рис. 1: Блок-схема алгоритма перемножения двух прямоугольных матриц

# 3 Заключение

Для выбора наиболее оптимального соотношения скорости выполнения программы и объёма конечного исполняемого файла необходимо выбрать уровень оптимизации и подобрать дополнительные системные опции оптимизации. В ходе выполнения лабораторной работы я поочерёдно компилировал исходный файл с различными значениями ключей оптимизации. После компиляции я замерял время выполнения программы и объём конечного исполняемого файла. Для автоматизации измерений был написан сценарий командной строки.

На основании полученных данных было принято решение о выборе уровня оптимизации -O3[4]. При этом уровне оптимизации включены следующие флаги межпроцедурной оптимизации: *-fipa-bit-cp, -fipa-cp, -fipa-icf, -fipa-ra, -fipa-sra, -fipa-vrp, -fipa-cp-clone*[1].

Оптимизация времени компоновки (флаг *-flto*) позволяет уменьшить дисковое пространство, занимаемое исполняемым файлом. Сокращения времени выполнения не наблюдается.

Оптимизация с оптимальной опцией и с обратной связью (-fprofile-generate / -fprofile-use) не дала видимых результатов.

**Вывод:** достаточной оптимизацией данного приложения будет выбор уровня оптимизации *-O3* с оптимизацией времени компоновки *-flto*. Время выполнения программы уменьшилось в 7.5 раз[5], объём конечного исполняемого файла уменьшился на 20%[6].

[4]Данный ключ в дополнение ко всем методам, применяемым на уровнях -O2 и -O1, включает в себя более дорогостоящие методы оптимизации, такие как подстановка функции и др.

[5]С 3.32 секунд до 0.43 секунд

[6]С 20952 байта до 16688 байта

# Список литературы


[1] Using the GNU Compiler Collection For gcc version 13.2.0: manual / Richard M. Stallman and the GCC Developer Community.— Boston: GNU Press, 2023. — p.173-246.

[2] Metcalf M. Modern Fortran Explained / M. Metcalf, J. Reid, M. Cohen.—$7^{th}$ Ed.— New York: Oxford University Press Inc., 2011. — 488 p.

[3] Основы программирования. Методические указания по составлению схем алгоритмов: метод. указание / И. А. Вернинов, В. А. Зимницкий, Л. К. Кириллова. — Ленинград: Изд-во ЛПИ им. М. И. Калинина, 1986. — 39 с.

[4] Львовский С. М. Набор и верстка в пакете LaTeX— 5-е изд., переработанное — М.: МЦНМО, 2014. — 400 с.

# A Приложение - Исходный код программы

```
1  program exercise_3_18_v2
2
3     implicit none
4     character(*), parameter     :: matrixA_file = "matrixA.txt",
        matrixB_file = "matrixB.txt", output_file = "output.txt"
5     integer, parameter          :: R_ = 8
6     integer                     :: In = 0, M, N, K
7     real(R_), allocatable       :: matrixA(:, :), matrixB(:, :),
        matrixC(:, :)
8
9     ! Read the MatrixA(M, N), M - rows, N - columns
10    open (file=matrixA_file, newunit=In)
11       read (In, *) M, N
12       allocate (matrixA(N, M)) ! matrixA(j, i): j - column
           number, i - row number
13       read (In, *) matrixA
14    close (In)
15
16    ! Read the MatrixB(N, K), N - rows, K - columns
17    open (file=matrixB_file, newunit=In)
18       read (In, *) N, K
19       allocate (matrixB(K, N)) ! matrixB(j, i): j - column
           number, i - row number
20       read (In, *) matrixB
21       allocate (matrixC(M, K))
22    close (In)
23
24    ! Find the matrixes Multiplication with use of matrixMult
        function
25    matrixC = matrixMult(matrixA, matrixB)
26
27  contains
28
29     pure function MatrixMult(A, B) result(C)
30       real(8)    A(N, M), B(K, N), C(M, K)
31       intent(in)  A, B
32       integer     jC, kC
33
34          C =  0.0
35             do jC = 1, K
36                do kC = 1, N
37                C(:, jC) = C(:, jC) + (A(:, kC) * B (kC, jC))
38             end do
39          end do
40    end function MatrixMult
41
42 end program exercise_3_18_v2
```