**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра теории вероятностей и кибербезопасности**

**ОТЧЁТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

*дисциплина: Параллельное программирование*

Студент: Добычин Дмитрий Романович

Студ. билет № 1032217059

Группа: НПИбд-01-21

**МОСКВА**

2024 г.

# Общая информация:

# Язык программирования: Fortran

**Задание №1:**

Используя средства OpenMP для редукции действий с массивами, реализуйте параллельные функции по нахождению суммы, минимума и максимума некоторого массива. Код заготовки модуля reduction находится в файле reduction.f90 в директории src. Вам необходимо дополнить функции omp\_sum, omp\_max и omp\_min.

После того, как функции будут реализованы, их надо протестировать с помощью программы test\_reduction.f90 из каталога test.

Ознакомьтесь с кодом программы и выясните как её следует запускать.

Данная программа для нахождения среднего времени работы функции использует алгоритм реального времени (on-line average), который реализован в функции online\_average из модуля stats из каталога src. Сам алгоритм нахождения среднего основывается на постоянном обновлении среднего значения выборки по мере поступления новых значений. Его преимущество заключается в том, что не требуется накапливать результаты замеров в массив, что экономит память в случае большого количества замеров.

Протестируйте быстродействие параллельных функций с помощью скрипта для построения графиков plot.py.

Как им пользоваться показано в лабораторной №0. Вы должны получить по три графика на каждую из созданных функций.

**Реализация на Fortran задания №1:**

**module** reduction

**use** iso\_fortran\_env**,** **only:** int32**,** int64**,** real32**,** real64

**implicit** **none**

**private**

! Доступные вовне функции и подпрограммы

**public** **::** omp\_sum**,** omp\_max**,** omp\_min

**public** **::** omp\_reduction

abstract **interface**

**function** omp\_reduction**(**A**,** ths\_num**)**

**double** **precision,** **dimension(**1**:),** **intent(in)** **::** A

**integer,** **intent(in)** **::** ths\_num

**double** **precision** **::** omp\_reduction

**end** **function** omp\_reduction

**end** **interface**

**contains**

**function** omp\_sum**(**A**,** ths\_num**)** **result(**S**)**

**implicit** **none**

**real(**real64**),** **dimension(**1**:),** **intent(in)** **::** A

**integer(**int32**),** **intent(in)** **::** ths\_num

**real(**real64**)** **::** S

**integer(**int64**)** **::** i

**integer(**int64**)** **::** length

length **=** **size(**A**)**

S **=** 0.0d0

! Параллельный блок с редукцией по сумме

!$omp parallel do num\_threads(ths\_num) reduction(+:S)

**do** i **=** 1**,** length

S **=** S **+** A**(**i**)**

**end** **do**

!$omp end parallel do

**end** **function** omp\_sum

**function** omp\_max**(**A**,** ths\_num**)** **result(**M**)**

**implicit** **none**

**real(**real64**),** **dimension(**1**:),** **intent(in)** **::** A

**integer(**int32**),** **intent(in)** **::** ths\_num

**real(**real64**)** **::** M

**integer(**int64**)** **::** i

**integer(**int64**)** **::** length

length **=** **size(**A**)**

M **=** A**(**1**)**

! Параллельный блок с редукцией по максимуму

!$omp parallel do num\_threads(ths\_num) reduction(max:M)

**do** i **=** 1**,** length

**if** **(**A**(**i**)** **>** M**)** M **=** A**(**i**)**

**end** **do**

!$omp end parallel do

**end** **function** omp\_max

**function** omp\_min**(**A**,** ths\_num**)** **result(**M**)**

**implicit** **none**

**real(**real64**),** **dimension(**1**:),** **intent(in)** **::** A

**integer(**int32**),** **intent(in)** **::** ths\_num

**real(**real64**)** **::** M

**integer(**int64**)** **::** i

**integer(**int64**)** **::** length

length **=** **size(**A**)**

M **=** A**(**1**)**

! Параллельный блок с редукцией по минимуму

!$omp parallel do num\_threads(ths\_num) reduction(min:M)

**do** i **=** 1**,** length

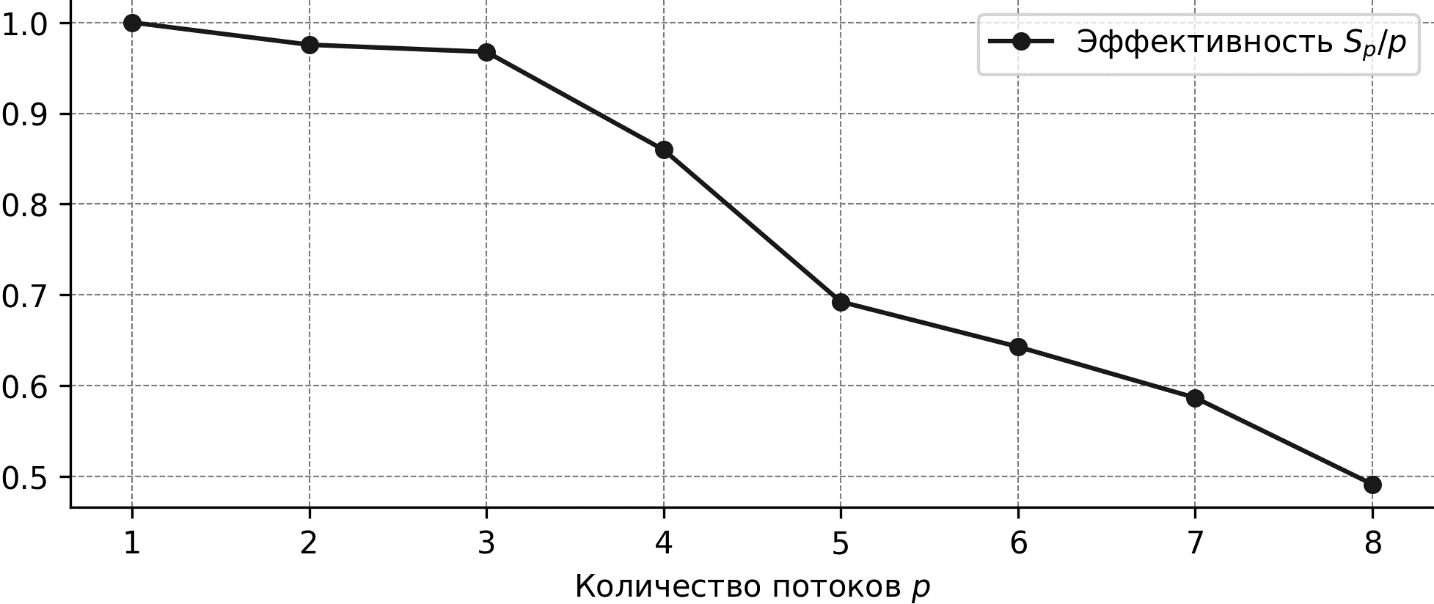
**if** **(**A**(**i**)** **<** M**)** M **=** A**(**i**)**

**end** **do**

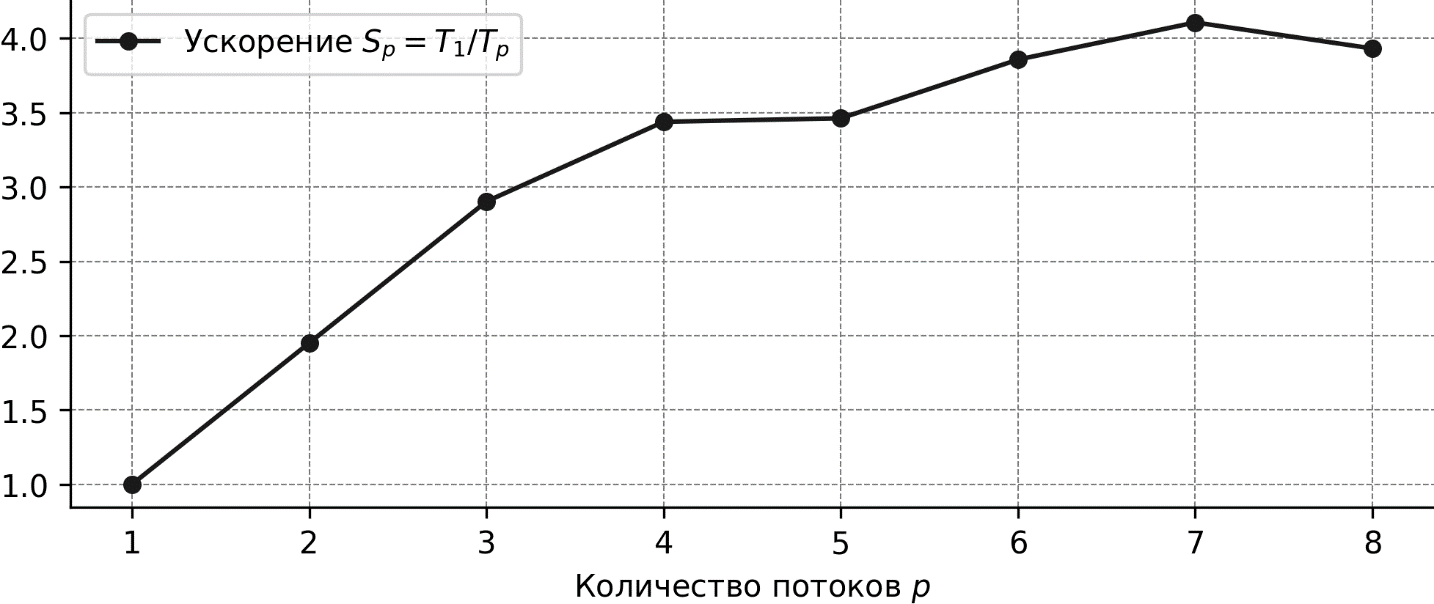
!$omp end parallel do

**end** **function** omp\_min

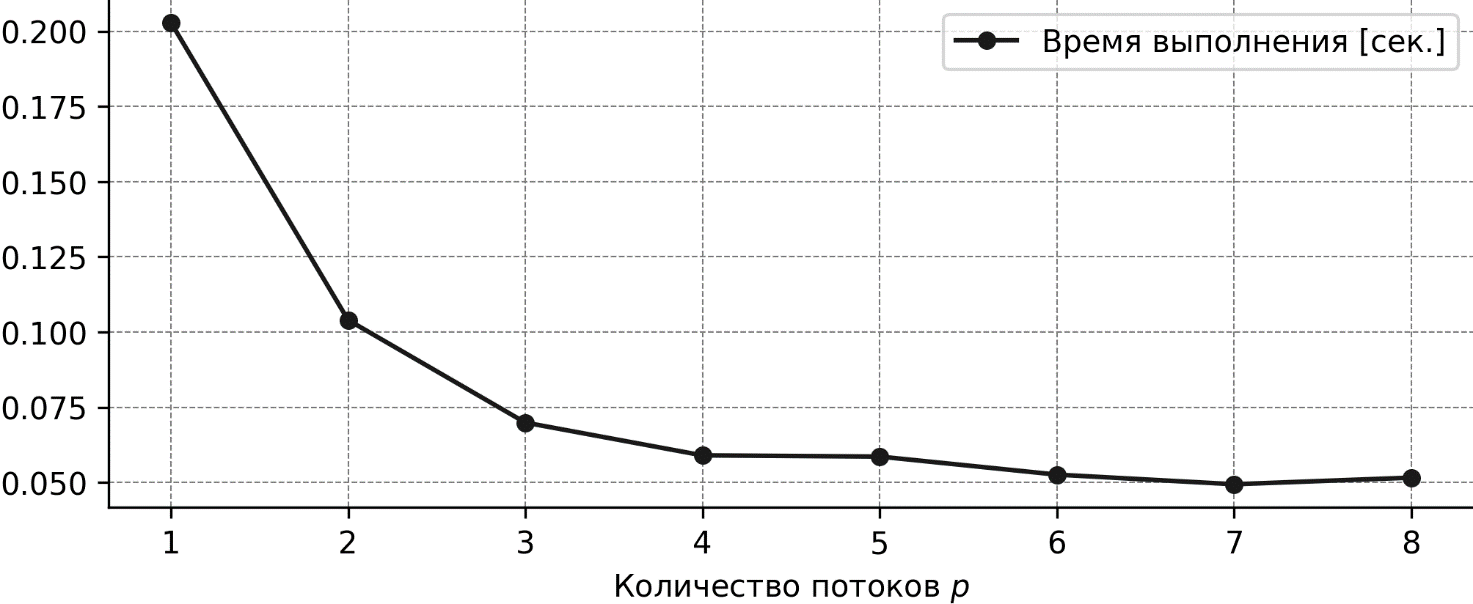
**end** **module** reduction

****

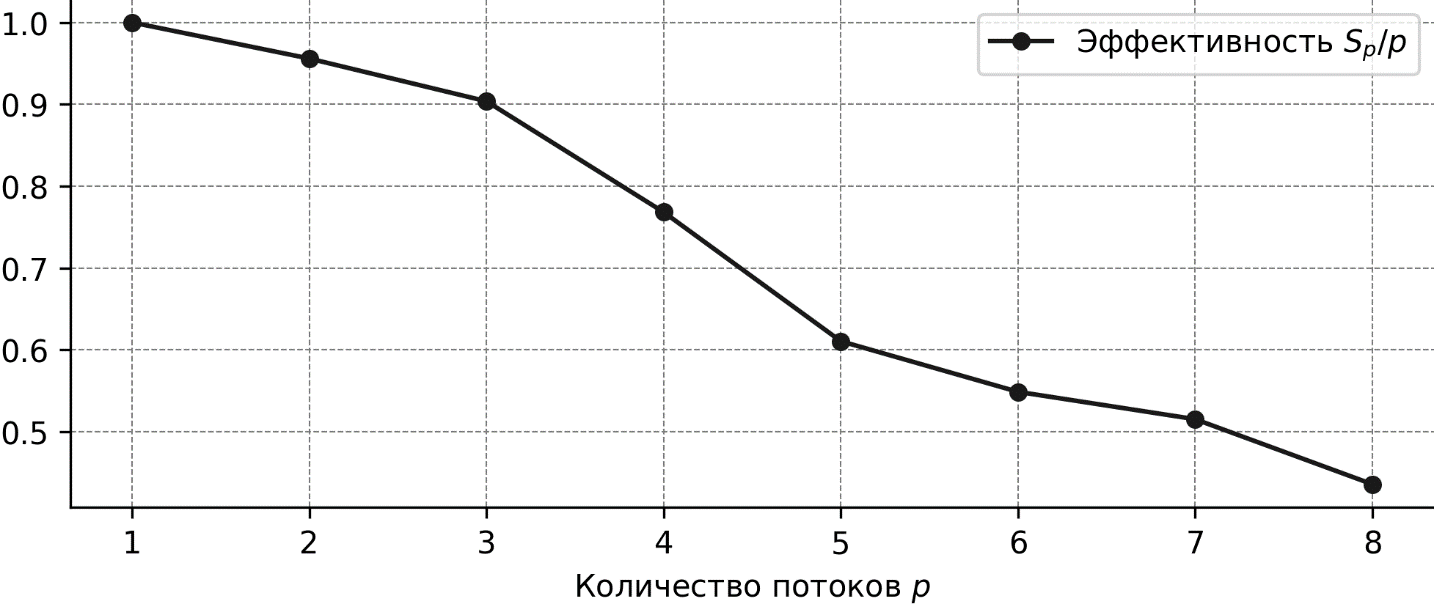
**Рис. 1.1.** Reduction\_max (эффективность).

****

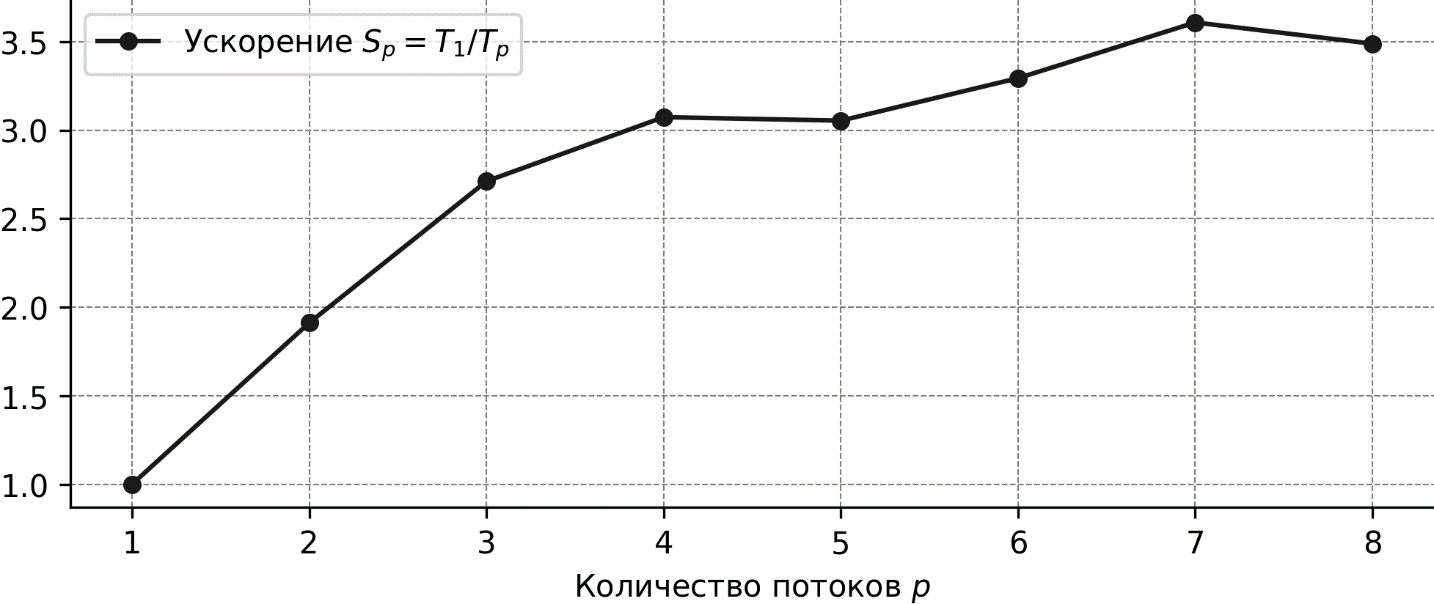
**Рис. 1.2.** Reduction\_max (ускорение).

****

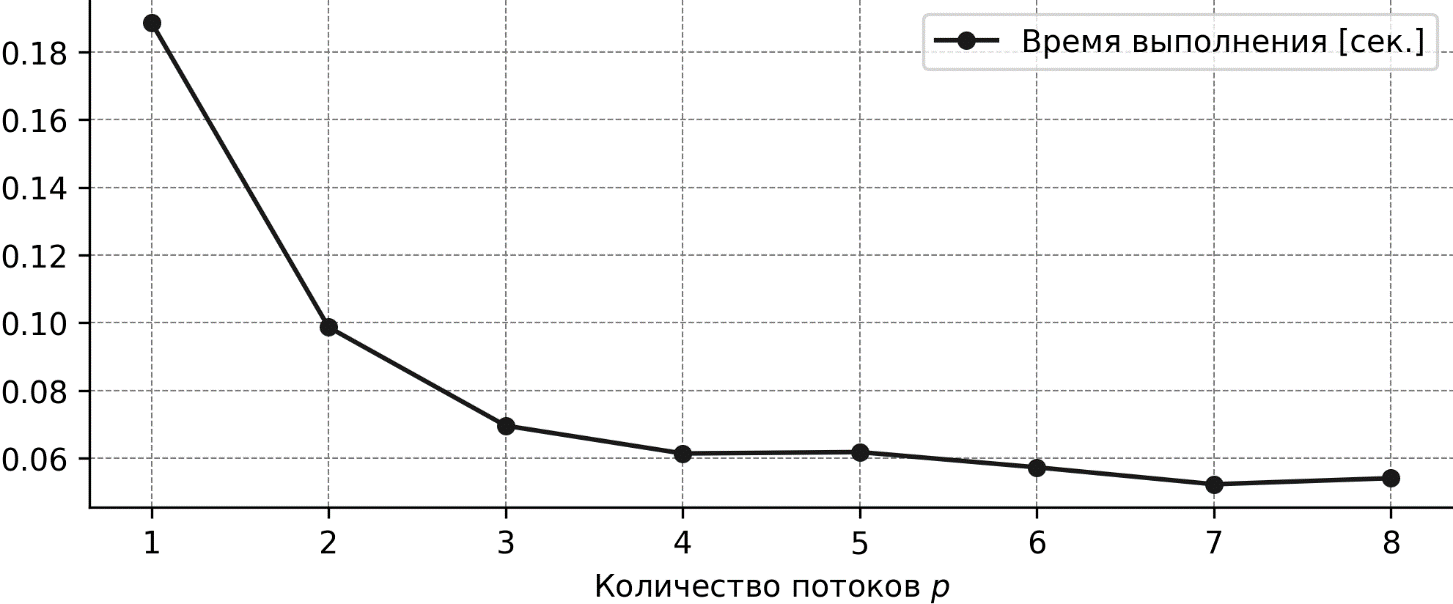
**Рис. 1.3.** Reduction\_max (время выполнения).

****

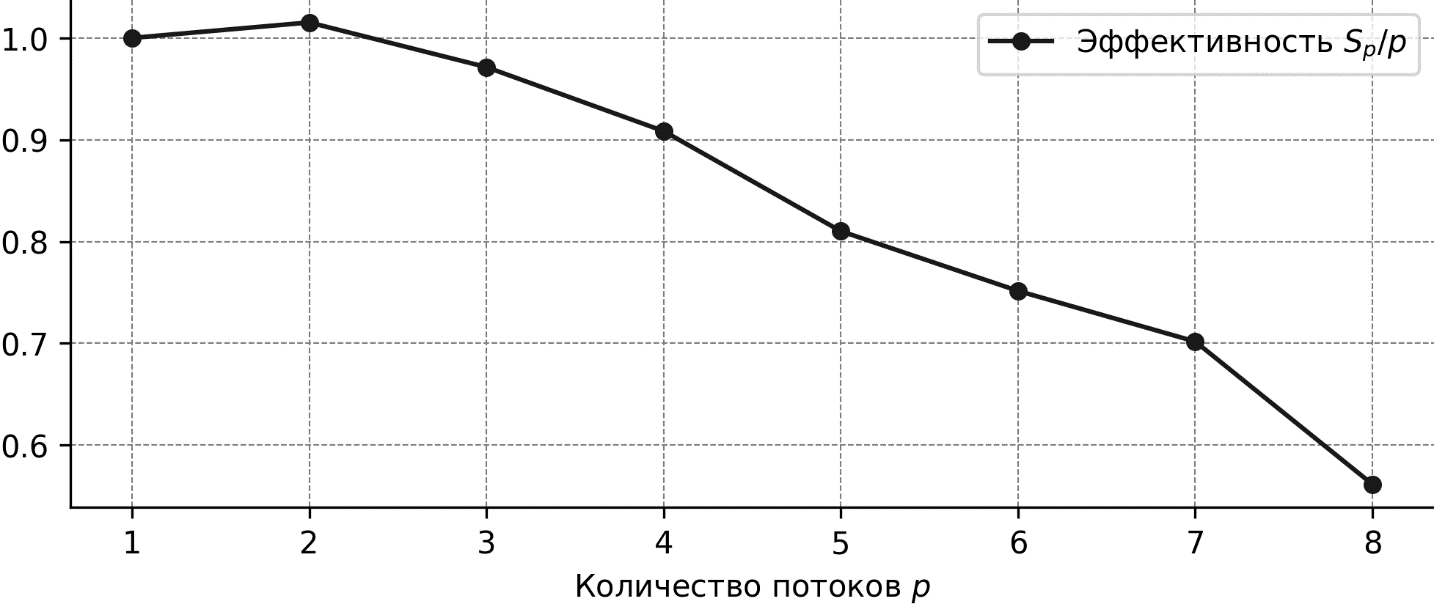
**Рис. 1.4.** Reduction\_min (эффективность).

****

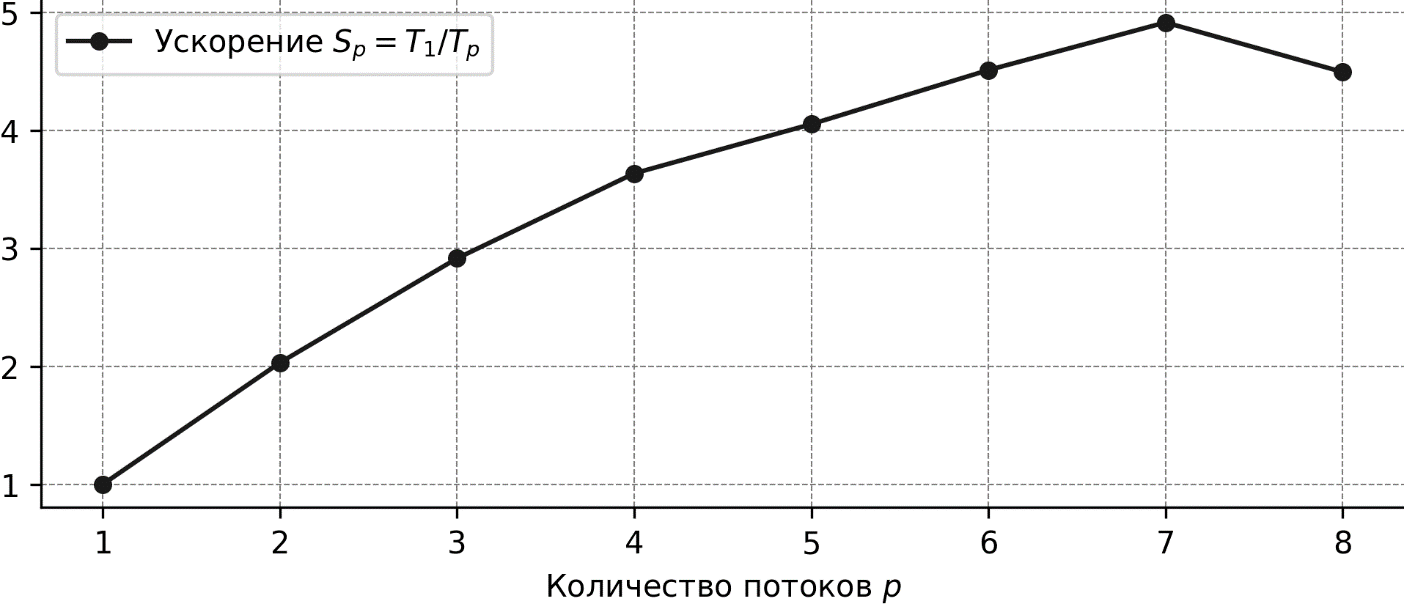
**Рис. 1.5.** Reduction\_min (ускорение).

****

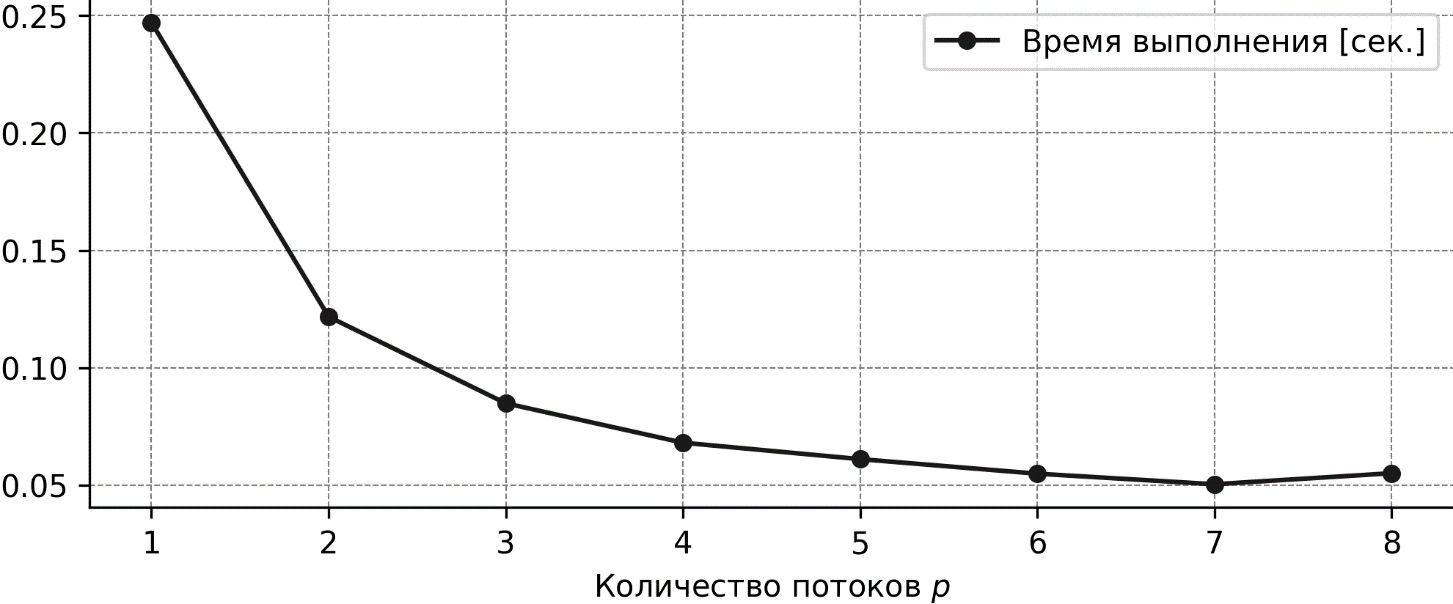
**Рис. 1.6.** Reduction\_min (время выполнения).

****

**Рис. 1.7.** Reduction\_sum (эффективность).

****

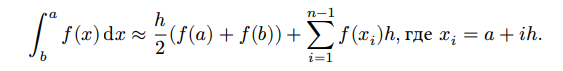
**Рис. 1.8.** Reduction\_sum (ускорение).

****

**Рис. 1.9.** Reduction\_sum (время выполнения).

**Задание №2:**

Используйте формулу трапеции для реализации функции trapezoidal, которая аппроксимирует значение интеграла от скалярной функции 𝑓(𝑥).



Код заготовки модуля trapezoidal\_rule вам необходимо дополнить, он находится в одноименном файле все в той же директории src.

Функция должна уметь распараллеливать свою работу на заданное число потоков с помощью технологии OpenMP. После реализации функции проверьте корректность ее работы и быстродействие с помощью тестирующей программы test\_trapezoidal.f90 из каталога test.

Исходный код модуля integrands, где находятся подынтегральная функция смотрите в файле integrands.f90 в каталоге src.

Для нахождения среднего также используется модуль stats, кратко описанный в предыдущем задании. Протестировать быстродействие параллельного вычисления в зависимости от числа потоков можно с помощью скрипта для построения графиков plot.py.

**Реализация на Fortran задания №2:**

**module** trapezoidal\_rule

**use** iso\_fortran\_env**,** **only** **:** int32**,** int64**,** real32**,** real64

**use** omp\_lib

**implicit** **none**

abstract **interface**

! Абстрактный интерфейс для подынтегральной функции

pure **function** f**(**x**)**

**double** **precision,** **intent(in)** **::** x

**double** **precision** **::** f

**end** **function** f

**end** **interface**

**private**

**public** **::** trapezoidal

**contains**

!----------------------------------------------------------------

! Функция вычисляющая значение интеграла по формуле трапеции

! func --- подынтегральная функция от одного аргумента

! a и b задают пределы интегрирования

! n --- число точек разбиения отрезка [a, b]

! threads\_num --- число потоков для распараллеливания

!----------------------------------------------------------------

**function** trapezoidal**(**func**,** a**,** b**,** n**,** threads\_num**)** **result** **(**res**)**

**implicit** **none**

**procedure(**f**)** **::** func

**real(**real64**),** **intent(in)** **::** a**,** b

**integer(**int64**),** **intent(in)** **::** n

**integer(**int64**),** **intent(in)** **::** threads\_num

**real(**real64**)** **::** res

**real(**real64**)** **::** h**,** x\_i**,** **sum**

**integer(**int64**)** **::** i

! Шаг интегрирования

h **=** **(**b **-** a**)** **/** **real(**n**,** **kind=**real64**)**

! Начальная сумма: трапециевидное приближение краев

res **=** **(**func**(**a**)** **+** func**(**b**))** **\*** h **/** 2.0

**sum** **=** 0.0

! Параллельный блок с редукцией для вычисления суммы значений f(x\_i)

!$omp parallel do private(x\_i) reduction(+:sum) num\_threads(threads\_num)

**do** i **=** 1**,** n**-**1

x\_i **=** a **+** i **\*** h

**sum** **=** **sum** **+** func**(**x\_i**)**

**end** **do**

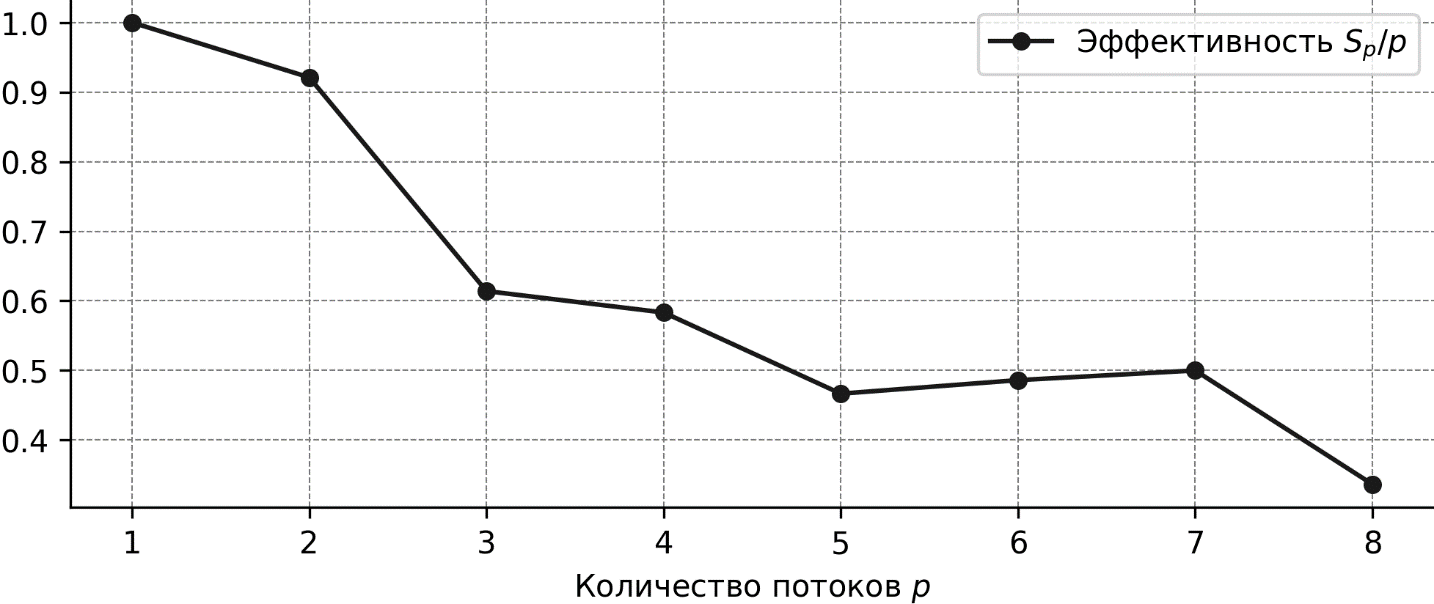
!$omp end parallel do

! Добавляем основную сумму

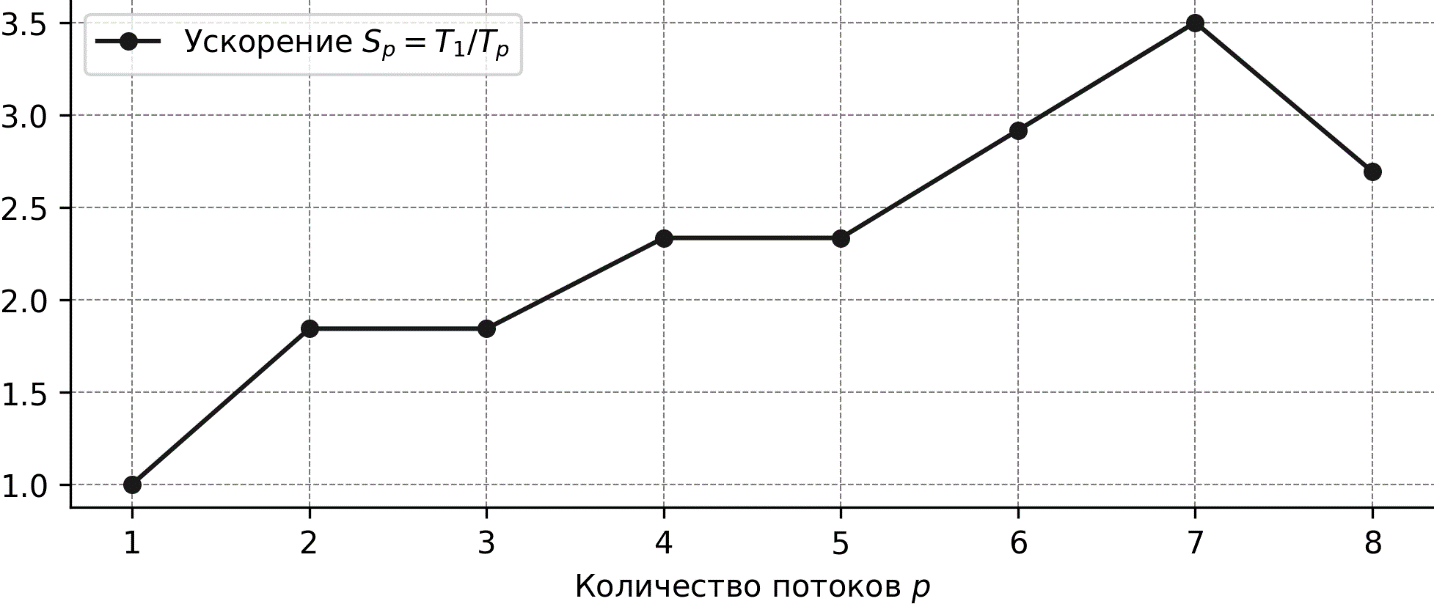
res **=** res **+** **sum** **\*** h

**end** **function** trapezoidal

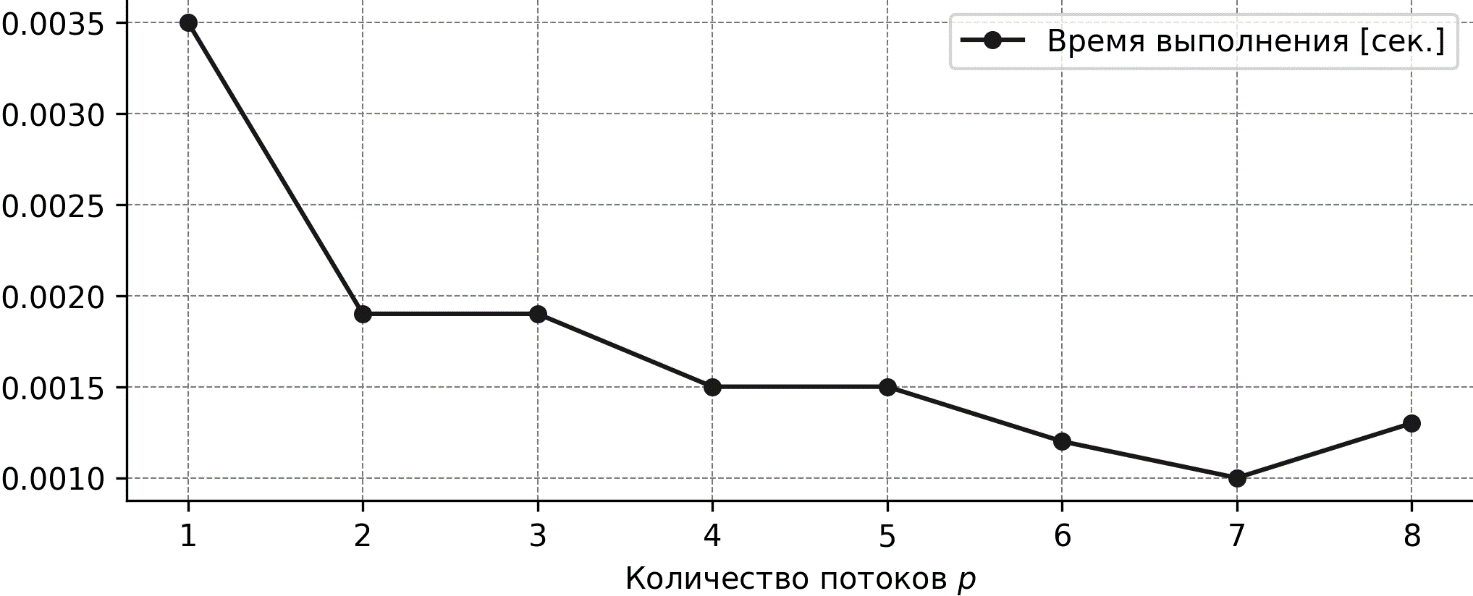
**end** **module** trapezoidal\_rule

****

**Рис. 2.1.** Trapezoidal (эффективность).

****

**Рис. 2.2.** Trapezoidal (ускорение).

****

**Рис. 2.3.** Trapezoidal (время выполнения).