МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

ОТЧЕТ

по практической работе

«Методологию многопоточного программирования»

по дисциплине

Параллельные вычисления

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_Мартынов Д. С.\_\_\_

(подпись) (фамилия, и.,о.)

СТУДЕНТЫ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ Кабальцев Н.Д.\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ Сухоруков В.А.\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ Мосташов В.С. \_\_

(подпись) (фамилия, и.,о.)

\_\_\_\_\_\_\_19-В-2\_\_\_\_\_\_\_\_

(шифр группы)

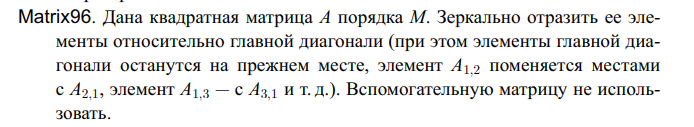
Работа защищена «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2021

Цель работы: Разработать программы, реализующие методологию многопоточного программирования.

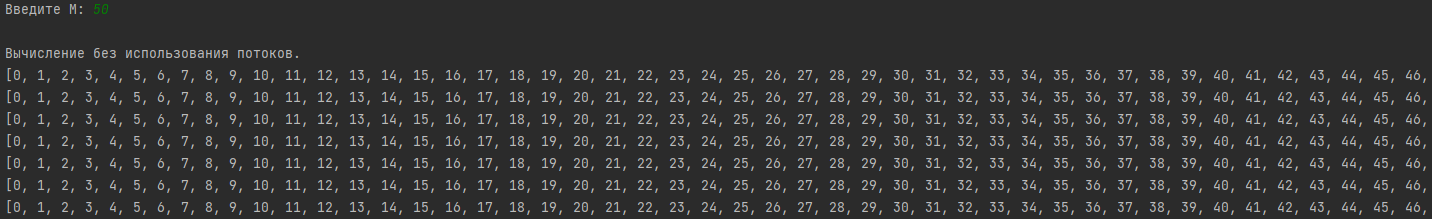
Задание №1



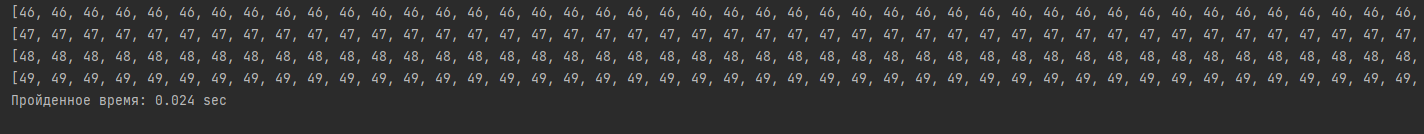
**Код программы:**

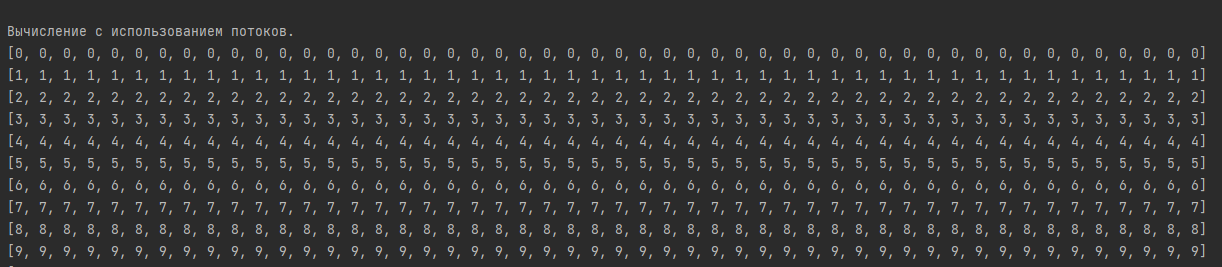
import threading, time  
from multiprocessing import cpu\_count  
  
# определение количества потоков  
CPU = cpu\_count()  
  
# Вывод матрицы   
def printMatrix(M):  
 global A  
 for i in range(M):  
 print(A[i])  
  
# обмен значениями между элементами в указаном интервале  
def func(a, b):  
 global A  
 for i in range(a, b):  
 for j in range(0, i):  
 A[i][j], A[j][i] = A[j][i], A[i][j]  
  
# вычисление затраченного времени  
class Profiler(object):  
 def \_\_enter\_\_(self):  
 self.\_startTime=time.time()  
 def \_\_exit\_\_(self, type, value, traceback):  
 print("Пройденное время: {:.3f} sec".format(  
 time.time() - self.\_startTime))  
   
# создание матрицы  
M = int(input("Введите M: "))  
A = []  
for i in range(M):  
 A.append([0]\*M)  
   
for i in range(M):  
 for j in range(M):  
 A[i][j] = j  
   
with Profiler() as p:  
 print("\nВычисление без использования потоков.")  
 printMatrix(M)  
 for i in range(1, M):  
 for j in range(0, i):  
 A[i][j], A[j][i] = A[j][i], A[i][j]  
 print()  
 printMatrix(M)  
  
with Profiler() as p:  
 print("\nВычисление с использованием потоков.")  
 printMatrix(M)  
   
 # определение границ в которых будут работать потоки  
 args = []  
 for i in range(CPU+1):  
 args.append(int((M/CPU)\*i))  
   
 #Создание потоков  
 threads = []  
 for i in range(CPU):  
 t = threading.Thread(target=func, args=(args[i],args[i+1],))  
 threads.append(t)  
 for thread in threads:  
 thread.start()  
 for thread in threads:  
 thread.join()  
 print()  
 printMatrix(M)

**Результат работы:**

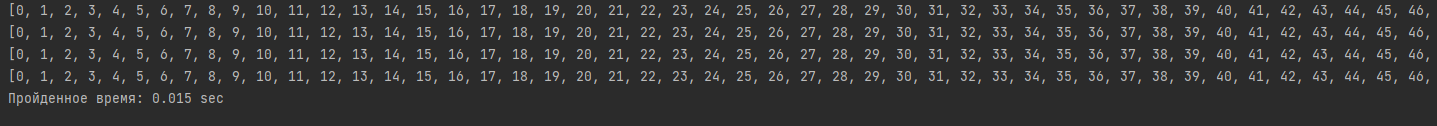
****

**…**

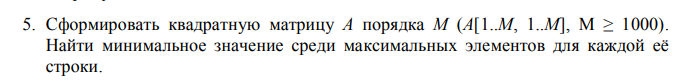
****

****

**…**

****

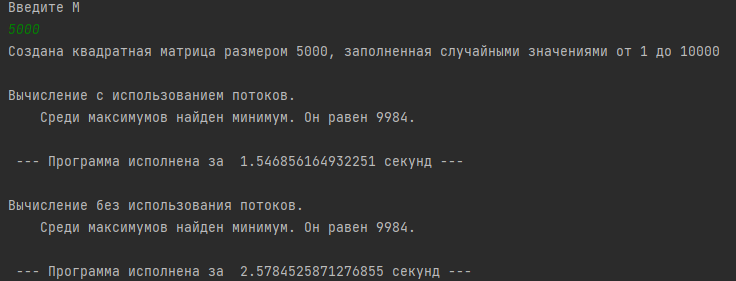
**Задние № 2**

****

**Код программы:**

import threading   
import time  
from multiprocessing import cpu\_count  
from random import randint  
  
# определение количества потоков  
CPU = cpu\_count()  
  
# Создание матрицы   
def create\_matrix(M):  
 A=[]  
 for i in range(M):  
 A.append([0]\*M)  
 for i in range(M):  
 for j in range(M):  
 A[i][j] = randint(1,10000)  
 print("Создана квадратная матрица размером %d, заполненная случайными значениями от 1 до 10000" %(M))  
 return(A)   
# END create\_matrix -----------------------------------------------  
  
# нахождение в строках максимума в указаном интервале  
def func(a, b, max\_in\_string):  
 for i in range(a, b):  
 temp = A[i][0]  
 for j in range(M-1):  
 if temp < A[i][j+1]:  
 temp = A[i][j+1]  
 max\_in\_string.append(temp)  
# END func --------------------------------------------------------  
   
def minimum(m):  
 minimum=m[0]  
 for i in range(0,len(m)-1):  
 if(m[i]<minimum):  
 minimum=m[i]  
 return minimum   
# END minimum ------------------------------------------------   
  
#Запрос M  
print("Введите M ")  
M=input()  
M=int(M)  
  
#Создание матрицы   
A = create\_matrix(M)  
  
#Замер времени исполнения программы  
start\_time = time.time()  
print("\nВычисление с использованием потоков.")  
  
# определение границ в которых будут работать потоки  
args = []  
for i in range(CPU+1):  
 args.append(int((M/CPU)\*i))  
  
max\_in\_string=[]  
  
#Создание потоков  
threads=[]  
for i in range(CPU):  
 tr = threading.Thread(target=func, args=(args[i],args[i+1],max\_in\_string,))  
 threads.append(tr)  
for thread in threads:  
 thread.start()  
for thread in threads:  
 thread.join()  
  
#Поиск минимума  
mi=minimum(max\_in\_string)  
print("\tСреди максимумов найден минимум. Он равен %d." %(mi))  
print("\n --- Программа исполнена за %s секунд ---" % (time.time() - start\_time))  
# END ---------------------------------------------------------------   
  
#Замер времени исполнения программы  
start\_time = time.time()  
print("\nВычисление без использования потоков.")  
  
max\_in\_string=[]  
  
for i in range(M):  
 temp = A[i][0]  
 for j in range(M-1):  
 if temp < A[i][j+1]:  
 temp = A[i][j+1]  
 max\_in\_string.append(temp)  
  
#Поиск минимума  
mi=minimum(max\_in\_string)  
print("\tСреди максимумов найден минимум. Он равен %d." %(mi))  
print("\n --- Программа исполнена за %s секунд ---" % (time.time() - start\_time))  
# END ---------------------------------------------------------------

**Результат работы:**

****

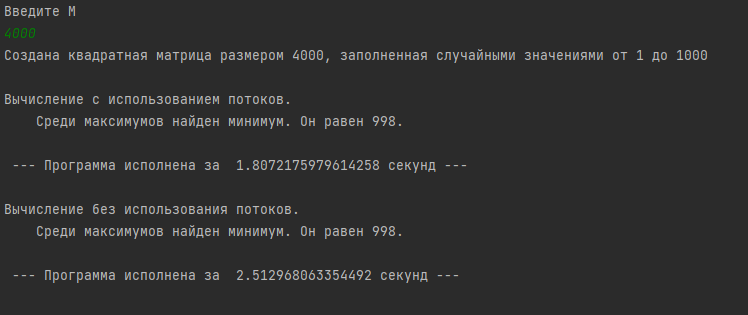
**Задание №3**

****

**Код программы:**

import threading   
import time  
from multiprocessing import cpu\_count  
from random import randint  
  
# определение количества потоков  
CPU = cpu\_count()  
  
# Создание матрицы   
def create\_matrix(M):  
 A=[]  
 for i in range(M):  
 A.append([0]\*M)  
 for i in range(M):  
 for j in range(M):  
 A[i][j] = randint(1,1000)  
 print("Создана квадратная матрица размером %d, заполненная случайными значениями от 1 до 1000" %(M))  
 return(A)   
# END create\_matrix -----------------------------------------------  
  
# нахождение в столбцах максимума в указаном интервале  
def func(a, b, max\_in\_column):  
 for j in range(a, b):  
 temp = A[0][j]  
 for i in range(M-1):  
 if temp < A[i+1][j]:  
 temp = A[i+1][j]  
 max\_in\_column.append(temp)  
# END func --------------------------------------------------------  
   
def minimum(m):  
 minimum=m[0]  
 for i in range(0,len(m)-1):  
 if(m[i]<minimum):  
 minimum=m[i]  
 return minimum   
# END minimum ------------------------------------------------   
  
#Запрос M  
print("Введите M ")  
M=input()  
M=int(M)  
  
#Создание матрицы   
A = create\_matrix(M)  
  
#Замер времени исполнения программы  
start\_time = time.time()  
print("\nВычисление с использованием потоков.")  
  
# определение границ в которых будут работать потоки  
args = []  
for i in range(CPU+1):  
 args.append(int((M/CPU)\*i))  
  
max\_in\_column=[]  
  
#Создание потоков  
threads=[]  
for i in range(CPU):  
 tr = threading.Thread(target=func, args=(args[i],args[i+1],max\_in\_column,))  
 threads.append(tr)  
for thread in threads:  
 thread.start()  
for thread in threads:  
 thread.join()  
  
#Поиск минимума  
mi=minimum(max\_in\_column)  
print("\tСреди максимумов найден минимум. Он равен %d." %(mi))  
print("\n --- Программа исполнена за %s секунд ---" % (time.time() - start\_time))  
# END ---------------------------------------------------------------   
  
#Замер времени исполнения программы  
start\_time = time.time()  
print("\nВычисление без использования потоков.")  
  
max\_in\_column=[]  
  
for j in range(M):  
 temp = A[0][j]  
 for i in range(M-1):  
 if temp < A[i+1][j]:  
 temp = A[i+1][j]  
 max\_in\_column.append(temp)  
  
#Поиск минимума  
mi=minimum(max\_in\_column)  
print("\tСреди максимумов найден минимум. Он равен %d." %(mi))  
print("\n --- Программа исполнена за %s секунд ---" % (time.time() - start\_time))  
# END ---------------------------------------------------------------   
  
M=input()

**Результат работы:**

****

**Вывод: По результатам работ программ, в которых была реализована** методологию многопоточного программирования, и, проведя сравнения с программами без использования потоков, можно сделать вывод, что время затраченное на работу программы, где не была использована методологию многопоточного программирования, выше, чем в программах с применением потоков.