Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра «Теоретическая информатика и компьютерные технологии»

Отчёт о лабораторной работе № 1 по курсу

«Разработка параллельных и

распределенных программ»

Распараллеливание алгоритма вычисления произведения двух матриц.

Студент: Белецкий В. С.

Группа: ИУ9-51Б

Преподаватель: Царев А.С.

Москва, 2022

Содержание

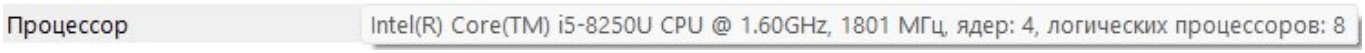
Постановка задачи

Две квадратные матрицы A и B размерности n сначала перемножить стандартным алгоритмом для получения матрицы C той же размерности. Замерить время вычисления, сравнить с временем при вычислении элементов матрицы C не по строкам, а по столбцам.

Затем конечную матрицу C условно разделить на примерно равные прямоугольные подматрицы и распараллелить программу таким образом, чтобы каждый поток занимался вычислением своей подматрицы. Матрицы A и B для этого разделить на примерно равные группы строк и столбцов соответственно. Сделать для разного количества потоков (разных разбиений), также замерить время вычисления, сравнить с вычислениями стандартным алгоритмом. Также по окончании вычислений сравнивать получившуюся матрицу с той, что была вычислена стандартным алгоритмом, для проверки правильности вычислений Уделить внимание проблеме синхронизации потоков.
При наличии в языке программирования и архитектуре возможностей управления приоритетами процессов и потоков оценить также их влияние на время работы программы.

Практическая реализация

Лабораторная работа выполнялась на ноутбуке с установленной ОС Windows 11, процессор имеет характеристики, указанные на скриншоте:



Код программы :

```
import random
import time
import multiprocessing
```

```
import numpy
import ctypes


class matrix(object):
    def __init__(self):
        self.n = 0 # количество строк и столбцов
        self.lines = []

    def set_matrix(self, file):
        new_file = file.readlines()
        self.n = int(new_file[0])
        for i in range(0, self.n):
            line = []
            for j in range(0, self.n):
                line.append(float(new_file[j + i * self.n + 2]))
            self.lines.append(line)

    def print(self):
        for line in self.lines:
            print(line)

    def set_empty(self, n):
        self.n = n
        for i in range(0, self.n):
            line = []
            for j in range(0, self.n):
                line.append(0)
            self.lines.append(line)
```

```
def make_answer(shared_array, shape, x, y, start, end):
    var_answer = to_numpy_array(shared_array, shape)
    for i in range(start, end):
        for j in range(0, x.n):
            cage = 0
            for k in range(0, x.n):
                cage = cage + x.lines[i][k] * y.lines[k][j]
            var_answer[i][j] = cage

def make_answer_1():
    for i in range(0, answer.n):
        for j in range(0, answer.n):
            cage = 0
            for k in range(0, answer.n):
                cage = cage + x.lines[i][k] * y.lines[k][j]
            answer.lines[i][j] = cage

def to_shared_array(arr, ctype):
    shared_array = multiprocessing.Array(ctype, arr.size, lock=False)
    temp = numpy.frombuffer(shared_array, dtype=arr.dtype)
    temp[:] = arr.flatten(order='C')
    return shared_array

def to_numpy_array(shared_array, shape):
    arr = numpy.ctypeslib.as_array(shared_array)
    return arr.reshape(shape)
```

```
x = matrix()
y = matrix()
answer = matrix()
if __name__ == '__main__':
    file = open('matrix.txt', 'w')
    file.write("400\n")
    for i in range(0, 160000):
        file.write("\n" + str(random.uniform(-10.0, 10.0)))
    file.close()

    file = open('matrix 2.txt', 'w')
    file.write("400\n")
    for i in range(0, 160000):
        file.write("\n" + str(random.uniform(-10.0, 10.0)))
    file.close()

    file1 = open('matrix.txt', 'r')
    file2 = open('matrix 2.txt', 'r')

    x.set_matrix(file1)
    y.set_matrix(file2)

    file1.close()
    file2.close()

    answer.set_empty(x.n)

    # без многопоточности
```

```
    time_start = time.perf_counter()


    make_answer_1()


    time_end = time.perf_counter()
    print("Время без многопоточности : " + str(time_end - time_start))
    print("Ячейка 135 277 равна : " + str(round(answer.lines[134][276], 4)))
    print()


    # 2 потока
    time_start = time.perf_counter()


    mp_array = numpy.zeros((400, 400), dtype=numpy.float32)
    shared_array = to_shared_array(mp_array, ctypes.c_float)
    answer_mp = to_numpy_array(shared_array, mp_array.shape)


    th1 = multiprocessing.Process(target=make_answer, args=(shared_array, mp_array.shape,
x, y, 0, 200,))
    th2 = multiprocessing.Process(target=make_answer, args=(shared_array, mp_array.shape,
x, y, 200, 400,))


    th1.start()
    th2.start()


    th1.join()
    th2.join()


    time_end = time.perf_counter()
    print("Время с 2 потоками : " + str(time_end - time_start))
```

```
    print("Ячейка 135 277 равна : " + str(round(answer_mp[134][276], 4)))

    print()


    # 4 потока

    time_start = time.perf_counter()


    mp_array = numpy.zeros((400, 400), dtype=numpy.float32)

    shared_array = to_shared_array(mp_array, ctypes.c_float)

    answer_mp = to_numpy_array(shared_array, mp_array.shape)


    th1 = multiprocessing.Process(target=make_answer, args=(shared_array, mp_array.shape,
x, y, 0, 100, ))

    th2 = multiprocessing.Process(target=make_answer, args=(shared_array, mp_array.shape,
x, y, 100, 200, ))

    th3 = multiprocessing.Process(target=make_answer, args=(shared_array, mp_array.shape,
x, y, 200, 300, ))

    th4 = multiprocessing.Process(target=make_answer, args=(shared_array, mp_array.shape,
x, y, 300, 400, ))


    th1.start()

    th2.start()

    th3.start()

    th4.start()


    th1.join()

    th2.join()

    th3.join()

    th4.join()


    time_end = time.perf_counter()
```

```
print("Время с 4 потоками : " + str(time_end - time_start))
print("Ячейка 135 277 равна : " + str(round(answer_mp[134][276], 4)))
```

В ходе реализации сделал для матриц 400 на 400, на старте программа создает 2 текстовых файла, куда записывает 2 матрицы, после чего заносит их в объекты класса. Сделал тесты для стандарта, 2х потоков и 4х, с выводом случайной ячейки для проверки правильности решения, чтобы не выводить и не сравнивать 3 большие матрицы. Матрицы по потокам разбиваются на равное количество линий и считаются, соответственно, одним потоком первые 200 линий итоговой матрицы, а вторым 200-400е линии, в 4х потоках, соответственно, первые 100, 100-200е, 200-300е и 300-400е.

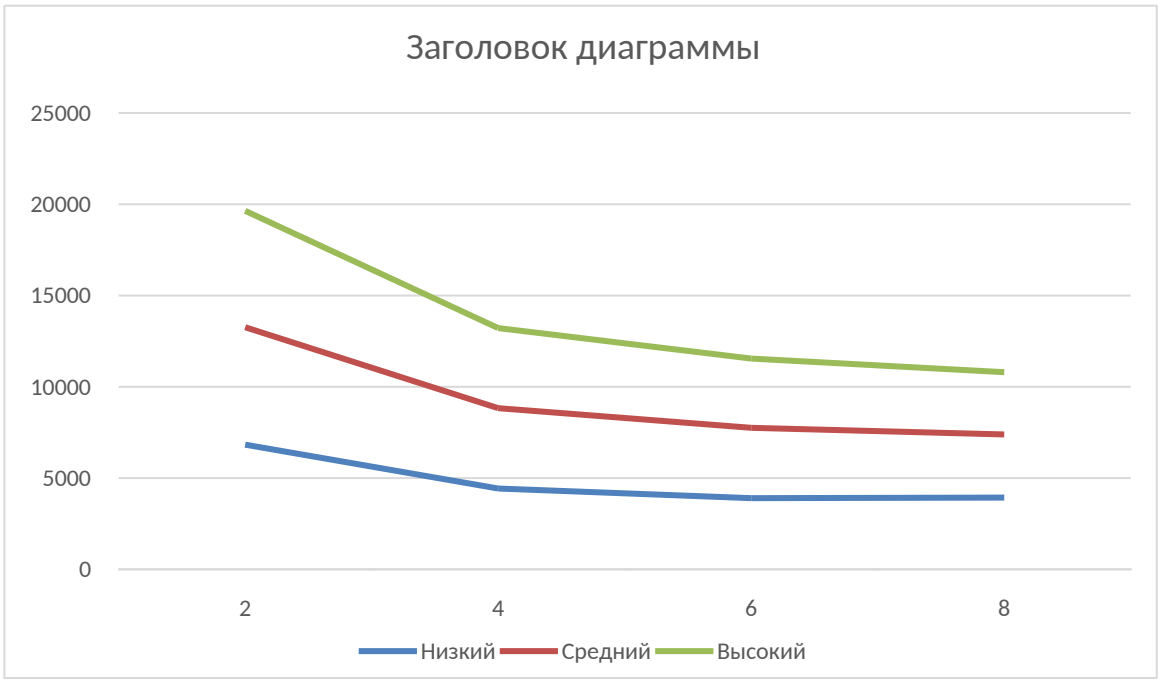
Результаты

В ходе выполнения лабораторной работы были изучены способы распараллеливания вычислений с помощью потоков. Для каждой конфигурации (матрицы 400х400 элементов и выполнение вычислений с 2, 4, 6, 8 потоками) исходных данных проводилось измерение затраченного время на вычисление. Результаты измерений прилагаются в виде таблицы и графика (указывается время в миллисекундах).

Время выполнения задачи стандартным алгоритмом: 38219 миллисекунд;

Время выполнения задачи алгоритмом по столбцам: 31298 миллисекунд.

Количество потоков	С низким приоритетом	С средним приоритетом	С высоким приоритетом
2	6832	6433	6378
4	4429	4402	4388
6	3901	3854	3794
8	3932	3459	3411



Выводы

Из приведённых данных видно, что использование потоков для вычисления произведения матриц позволило сократить время, затрачиваемое на вычисление. Оптимальное ускорение — в 1.74 раза. Также наглядно демонстрируется незначительное влияние во многих случаях выбора приоритета запускаемых потоков: в каждой тестовой конфигурации все создаваемые в ходе работы потоки были одного типа приоритета, что означает, что они все оказываются в «очереди» планировщика на схожих позициях, а в силу реализованного разбиения в большинстве случаев нет принципиальной разницы, какая из квадратных подматриц одинакового размера будет вычислена раньше.