САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе № 5

«OpenMP»

Выполнил: Лымарь Павел Игоревич

студ. гр. М313Д

Санкт-Петербург

2020

Цель работы: Знакомство со стандартом распараллеливания команд OpenMP.

Инструментарий и требования к работе: C++, Python, Java.

Теоретическая часть

1. OpenMP – стандарт для распараллеливания программ, преимущественно на языках C, C++. Дает описание совокупности команд компилятора, библиотек и переменных окружения, предназначенных для написания многопоточных приложений.

   Параллельные вычисления в OpenMP реализованы с помощью многопоточности, в которой основной поток вызывает вспомогательные.

   Разберемся в устройстве многопоточного программирования. Изначально для ускорения программ использовались многоядерные процессоры, при этом в программе исполнялось несколько независимых процессов. Но со временем требовалось выполнить один процесс, но максимально быстро, с данной задачей множество ядер не в силах быстро справится, т.к. только один исполнитель будет заниматься процессом. Тут нам на помощь и приходят треды. Для каждого процесса выделено несколько тредов, при этом если треды относятся к одному процессу, то память у них общая. Допустим, нам требуется посчитать сумму чисел в массиве. Разумеется, мы можем пройтись по всем элементам массива и добавлять их к счетчику $counter$. Как мы можем такое ускорить? Создадим еще один поток и заставим его пробегаться по второй половине массива, а первый будет смотреть элементы первой половины массива. Таким образом мы ускорим выполнение в 2 раза. Очевидно, что можно создавать и больше потоков, деля массив пропорционально их количеству, тем самым ускоряя работу программы.

Практическая часть

Вариант 7. Определитель матрицы

2. Для того, чтобы найти детерминант матрицы будем приводить ее к треугольному виду и находить произведение элементов на главной диагонали (Метод Гаусса). Данный алгоритм может быть реализован за $O(n^3)$.

   Реализовать алгоритм весьма тривиально: будем перебирать переменную $i = 0..n-1$, на каждой итерации будем работать с i-ым столбцом справа. После будем занулять все элементы под номерами i, строк, расположенных ниже строки i. Перед этим, разумеется, выберем, какой элемент поставим на i-ую строку. Просто пробежавшись по всем номерам строк $j > i$, и выбрав из них ту, на которой стоит наибольший по модулю элемент на позиции i.

   Если мы меняем строку i местами с какой-либо наш текущий результат требуется умножить на -1 (по свойствам детерминанта матрицы).

   После того как мы выбрали i-ую строку так же надо проверить, не равен ли ее i элемент 0 (в таком случае выведем 0, т.к. элемент стоит на диагонали и участвует в произведении).

   В конечном итоге, после всех необходимых проверок обнуляем ранее оговоренные строки.

   Программа некорректно работает при больших входных данных, т.к. язык программирования C++ не в полной мере подходит для решения подобных задач.

   Условия тестирования:

   Скомпилированная программа выполнялась на ноутбуке Xiaomi Mi Notebook Pro, с процессором intel core i7 10510U (4 ядра, 8 потоков, базовая тактовая частота 1.80GHz).

Тесты проводились с использованием файла исходных данных input.txt (находится во вложении).

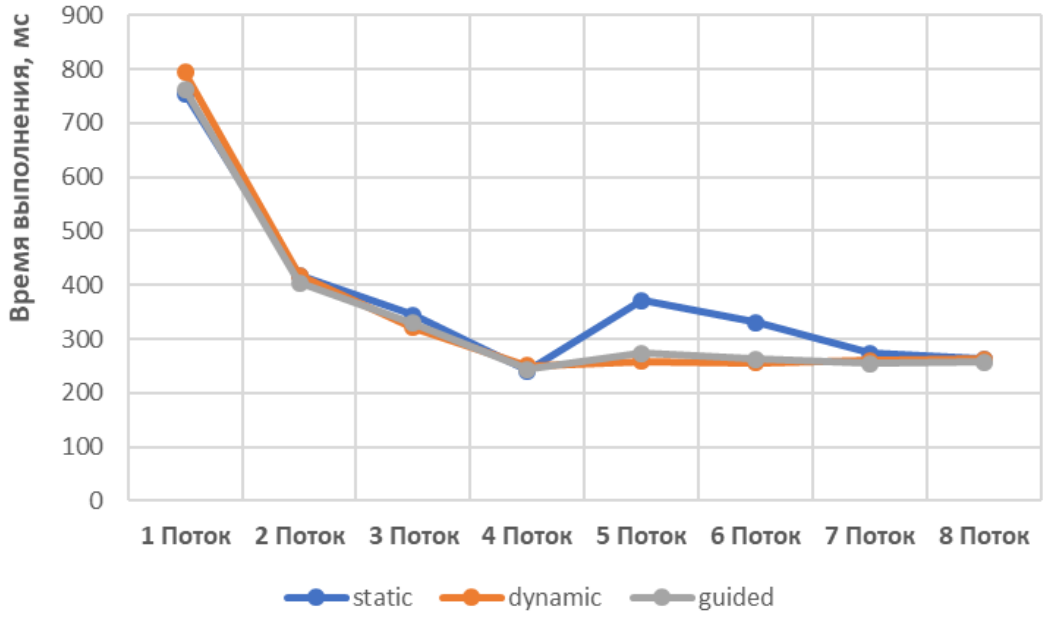

Рисунок 1 – График времени работы программы при различных значениях числа потоков при одинаковом параметре schedule

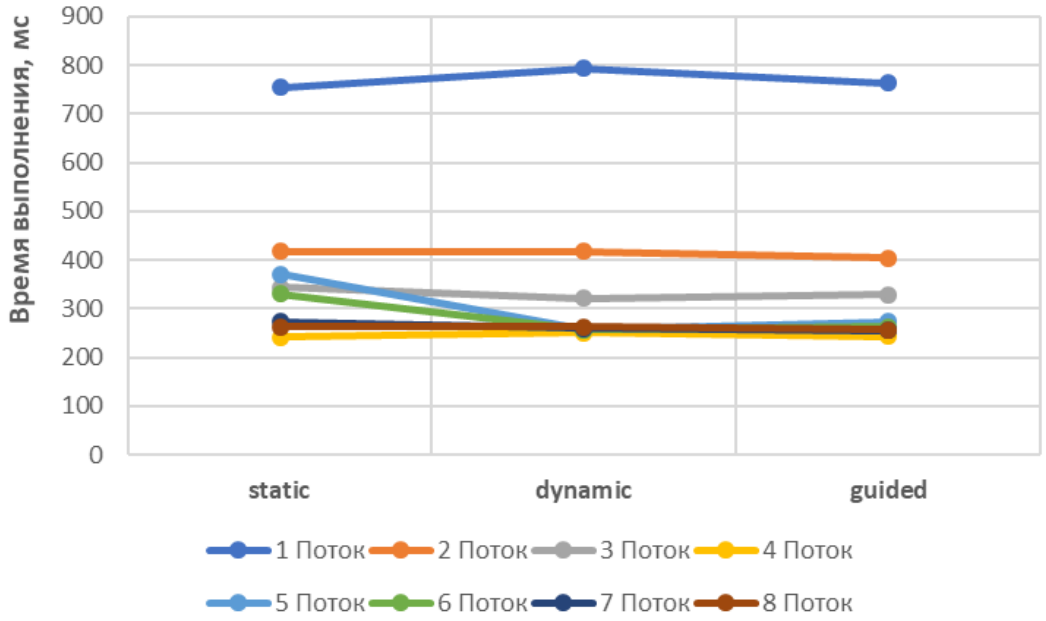

Рисунок 2 – График времени работы программы при одинаковом значении числа потоков при различных параметрах schedule

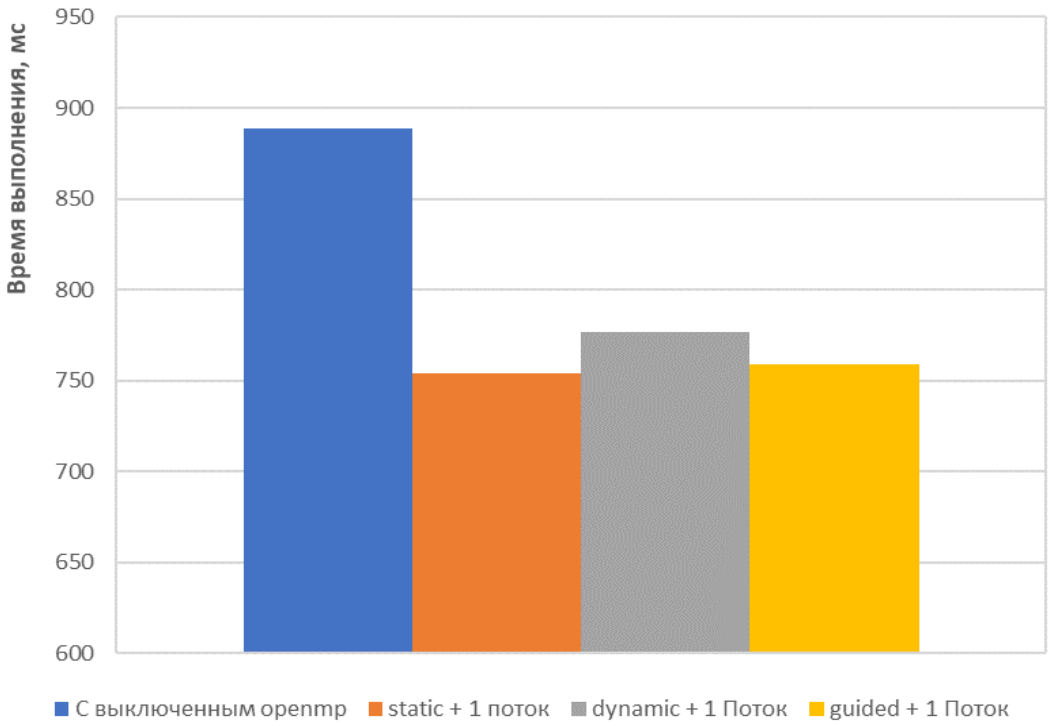

Рисунок 3 – График времени работы программы с выключенным openmp и с включенным 1 потоком

Листинг

Компилятор: gpp.

Файл компилируются с помощью script.bat, необходимые команды, для компилирования вручную можно посмотреть внутри него.

Main.cpp

```
#include <iostream>
#include <omp.h>
#include <sstream>

using namespace std;

float determinant(float **matrix, int n) {
    float result = 1;
    for (int i = 0; i < n; i++) {
        int temp = i;
        for (int j = i + 1; j < n; j++) {
            if (abs(matrix[j][i]) >= abs(matrix[temp][i])) {
                temp = j;
```

```
                break;
            }
        }
        if (temp != i){
            result *= -1;
            swap(matrix[i], matrix[temp]);
        }
        if (matrix[i][i] == 0) {
            return 0;
        }
        result *= matrix[i][i];
        //#pragma omp parallel for schedule(static)
        //#pragma omp parallel for schedule(dynamic)
        #pragma omp parallel for schedule(guided)
        for (int j = i + 1; j < n; j++) {
            float b = matrix[j][i] / matrix[i][i];
            for (int k = i + 1; k < n; k++) {
                matrix[j][k] -= matrix[i][k] * b;
            }
        }
        delete matrix[i];
    }
    return result;
}

int main(int numOfArgs, char *args[]) {
    if (numOfArgs <= 2) {
        printf("Usage: hw5.exe <number of threads> <input file> [<output
file>].\n");
        return 0;
    }
    int threads;
    stringstream(args[1]) >> threads;
    int maxNumberOfThreads = omp_get_max_threads();
    if (threads < 0 or maxNumberOfThreads < threads) {
        printf("Illegal number of threads.\n");
        return 0;
    }
    if (threads < maxNumberOfThreads) {
        omp_set_num_threads(threads);
```

```
} else {
    threads = maxNumberOfThreads;
}

FILE *file;
file = fopen(args[2], "r");
if (file == NULL) {
    printf("File isn't found.\n");
    return 0;
}
fclose(file);

freopen(args[2], "r", stdin);
bool isOutputFileExists = false;
if (numOfArgs == 4) {
    file = fopen(args[3], "w");
    if (file == NULL) {
        printf("File isn't found.\n");
        return 0;
    } else {
        isOutputFileExists = true;
    }
}

int n;
cin >> n;
float **matrix = new float *[n];
for (int i = 0; i < n; i++) {
    matrix[i] = new float[n];
    for (int j = 0; j < n; j++) {
        cin >> matrix[i][j];
    }
}

auto time = omp_get_wtime();
float result = determinant(matrix, n);
time = omp_get_wtime() - time;

delete matrix;
if (isOutputFileExists) {
```

```
        fprintf(file, "Determinant: %g\n", result);
        fclose(file);
    }
    printf("Determinant: %g\nTime: %g sec\n", result, time);

    return 0;
}
```