**ФГАОУ ВО «Национальный исследовательский университет ИТМО»**

**Факультет программной инженерии и компьютерной техники**

**Параллельные вычисления**

**Лабораторная работа №3**

**Распараллеливание циклов с помощью технологии OpenMP**

**Студент: Александров Ю.В. гр. Р4114**

**Преподаватель: Жданов А. Д.**

**Санкт-Петербург**

**2023**

Этапы берутся согласно 1 лабораторной работе.

Для данной работы необходимо все for-циклы распараллелить при помощи технологии OpenMP.

Воспользуемся директивой: #pragma omp parallel for default(none) private(…) shared(…)

**Характеристики компьютера:**

Процессор: Intel® Celeron N5095A, 2GHz

Количество ядер: 4

ОЗУ: 16 Gb, 2933 МГц

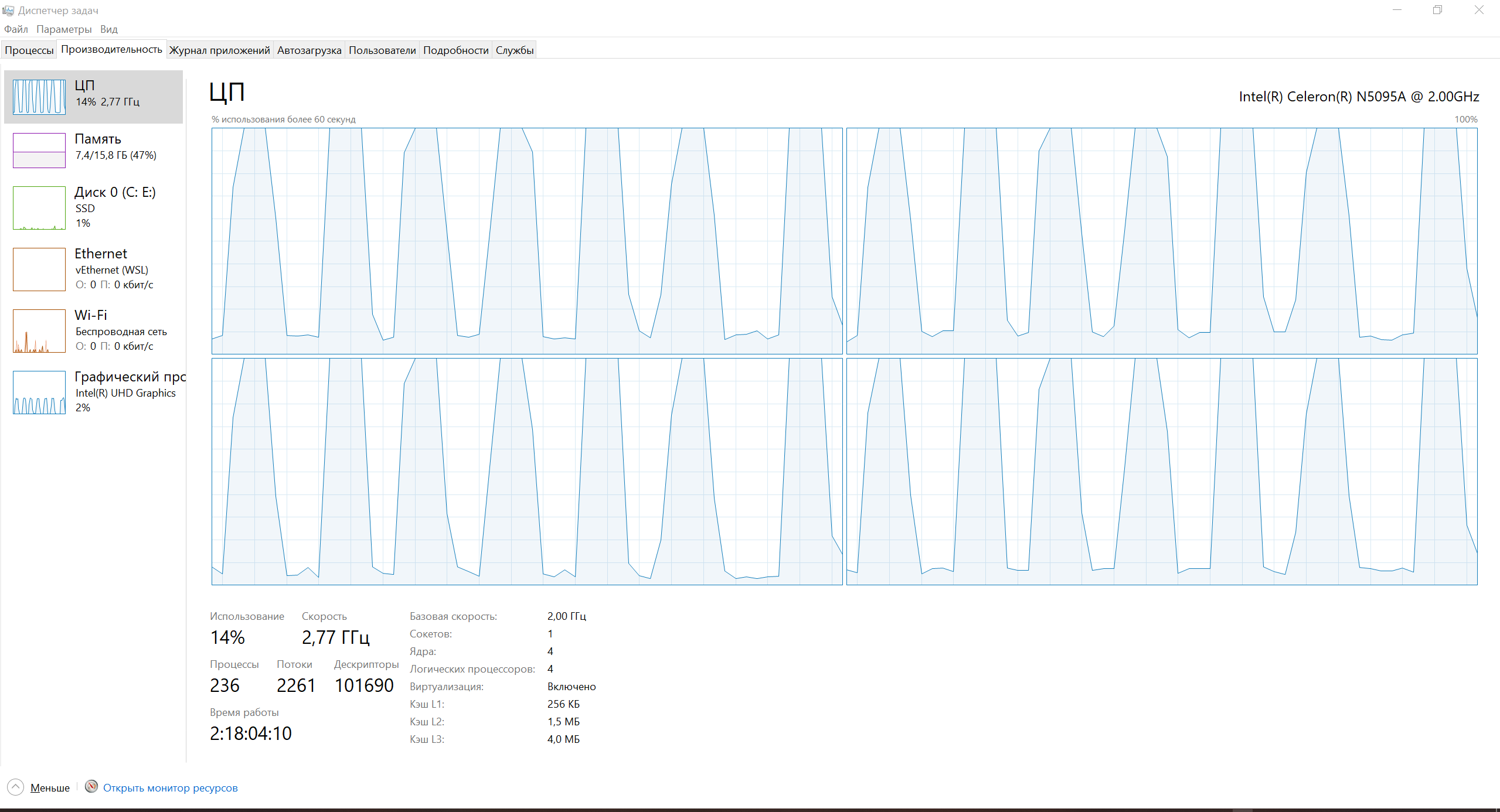
В ходе практического анализа было выявлено, что более быстрым по выполнению типом параметра является guided. А default тип имеет схожесть со всеми типами параметров с размеров check\_size = 8.

**Static**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **Static K = 1** | **Static K = 2** | **Static K = 4** | **Static K = 8** |
| 1 | 0,031 | 0,033 | 0,034 | 0,035 |
| 60001 | 0,652 | 0,573 | 0,638 | 0,612 |
| 120001 | 0,956 | 0,957 | 0,908 | 0,963 |
| 180001 | 1,367 | 1,463 | 1,393 | 1,421 |
| 240001 | 1,776 | 1,871 | 1,783 | 1,83 |
| 300001 | 2,158 | 2,11 | 2,056 | 2,052 |
| 360001 | 2,367 | 2,359 | 2,34 | 2,415 |
| 420001 | 2,765 | 2,799 | 2,663 | 2,67 |
| 480001 | 3,077 | 3,096 | 3,092 | 3,05 |
| 540001 | 3,355 | 3,501 | 3,386 | 3,371 |
| 600001 | 3,714 | 3,862 | 3,731 | 3,647 |
| **Dynamic** |  |  |  |  |
| **N** | **Dynamic K = 1** | **Dynamic K = 2** | **Dynamic K = 4** | **Dynamic K = 8** |
| 1 | 0,052 | 0,045 | 0,031 | 0,029 |
| 60001 | 1,439 | 0,866 | 0,728 | 0,593 |
| 120001 | 2,476 | 1,681 | 1,298 | 1,062 |
| 180001 | 3,33 | 2,247 | 1,73 | 1,476 |
| 240001 | 4,399 | 2,891 | 2,25 | 1,818 |
| 300001 | 5,537 | 3,455 | 2,733 | 2,241 |
| 360001 | 6,32 | 4,143 | 3,217 | 2,564 |
| 420001 | 7,395 | 4,74 | 3,639 | 3,118 |
| 480001 | 8,322 | 5,41 | 4,149 | 3,372 |
| 540001 | 9,403 | 6,233 | 4,551 | 3,727 |
| 600001 | 10,28 | 6,933 | 5,218 | 4,078 |

**Guided**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **Guided K = 1** | **Guided K = 2** | **Guided K = 4** | **Guided K = 8** |
| 1 | 0,047 | 0,032 | 0,033 | 0,03 |
| 60001 | 0,56 | 0,531 | 0,558 | 0,536 |
| 120001 | 0,97 | 0,916 | 0,907 | 0,918 |
| 180001 | 1,3 | 1,34 | 1,271 | 1,301 |
| 240001 | 1,61 | 1,672 | 1,615 | 1,683 |
| 300001 | 1,887 | 1,894 | 1,879 | 1,953 |
| 360001 | 2,272 | 2,27 | 2,256 | 2,278 |
| 420001 | 2,613 | 2,526 | 2,513 | 2,653 |
| 480001 | 2,969 | 2,939 | 2,89 | 3,037 |
| 540001 | 3,184 | 3,104 | 3,176 | 3,288 |
| 600001 | 3,478 | 3,44 | 3,494 | 3,544 |



**Расчёт параллельного ускорения:**

Путём исследования было отчётливо выявлено, что при типе static и маленьких значениях элементах массива, накладные расходы на распараллеливание превышают выигрыш от распараллеливания. Количество

Элементов массива 10 000.

Проведем расчёт времени на разные флаги

**Вывод:** в ходе выполнения лабораторной работы я научился работать с OpenMP. Было получено значительное ускорение при некоторых конфигурациях. Также были видны конфигурации при которых происходили значительное замедление.

**Приложение**

#include <stdio.h>

#include <stdlib.h>

#include <sys/time.h>

#include <math.h>

#include <time.h>

#include "omp.h"

// 3 Этап

// Merge согласно 6 варинту

void merge(double M1[], double M2[], int sizeArrayM2) {

for (int k = 0; k < sizeArrayM2; k++) {

M2[k] = fabsf(M1[k] - M2[k]);

}

}

// 4 Этап

// Сортировка stupid sort согласно 4 варианту

int isSorted(double M2[], int sizeArrayM2)

{

while (--sizeArrayM2 > 1)

if (M2[sizeArrayM2] < M2[sizeArrayM2 - 1])

return 0;

return 1;

}

void swap(double\* x, double\* y)

{

double tmp = \*x;

\*x = \*y;

\*y = tmp;

}

void shuffle(double M2[], int sizeArrayM2)

{

for (int i = 0; i < sizeArrayM2; i++)

swap(&M2[i], &M2[rand() % sizeArrayM2]);

}

void stupid\_sort(double M2[], int sizeArrayM2)

{

while (!isSorted(M2, sizeArrayM2))

shuffle(M2, sizeArrayM2);

}

int main(int argc, char\* argv[]) {

const int A = 440;

int numberFlow;

int N;

struct timeval T1, T2;

long delta\_ms;

if (argc < 3) {

printf("Need to enter arguments (sizeArray) and number flow ");

return -1;

}

N = atoi(argv[1]);

numberFlow = atoi(argv[2]);

#ifdef \_OPENMP

omp\_set\_num\_threads(numberFlow);

#endif

gettimeofday(&T1, NULL);

unsigned int first\_random = 0;

unsigned int second\_random = 0;

// 100 экспериментов

for (int i = 0; i < 100; i++) {

srand(i);

double\* M1 = (double\*)malloc(N \* sizeof(double));

double\* M2 = (double\*)malloc(N / 2 \* sizeof(double));

double\* M2\_Copy = (double\*)malloc(N / 2 \* sizeof(double));

// 1 Этап

// Заполение массива M1

for (int k = 0; k < N; k++) {

double value = 1 + rand\_r(&first\_random) % A;

M1[k] = value;

//printf("%.2f\n", value);

};

// Заполение массива M2

for (int k = 0; k < N / 2; k++) {

double value = 1 + rand\_r(&second\_random) % (A \* 10);

M2[k] = value;

M2\_Copy[k] = value;

//printf("%.2f\n", value);

}

#pragma omp parallel default(none) shared(N, M1, M2, M2\_Copy)

{

#pragma omp for schedule(runtime)

//2 Этап

//Применение операции соответсвенно 4 варианту к массиву М1

//for shared(N, M1) schedule(guided, 8)

for (int k = 0; k < N; k++) {

M1[k] = cosh(sqrt(M1[k])) / sinh(sqrt(M1[k]));

}

#pragma omp for schedule(runtime)

//Применение операции соответсвенно 2 варианту к массиву М2

for (int k = 0; k < N / 2; k++) {

if (k == 0) {

M2[k] = fabsf(cos(M2[k]));

continue;

}

M2[k] = fabsf(cos(M2[k] + M2\_Copy[k - 1]));

}

#pragma omp for schedule(runtime)

//3 Этап

//Merge согласно 6 варинту

for (int k = 0; k < N / 2; k++) {

M2[k] = fabsf(M1[k] - M2[k]);

}

}

// 4 Этап

//stupid\_sort(M2, N / 2);

// 5 Этап

//Приходит отсортированный массив

double result = 0;

double min = 0;

for (int i = 0; i < N / 2; i++) {

if (M2[i] != 0) {

min = M2[i];

break;

}

}

#pragma omp parallel for default(none) shared(N, M2, min) reduction(+:result) schedule(guided, 8)

for (int k = 0; k < N / 2; k++) {

if (((long)(M2[k] / min) % 2) == 0) {

result += sin(M2[k]);

}

}

printf("Result: %f\n", result);

free(M1);

free(M2);

free(M2\_Copy);

}

gettimeofday(&T2, NULL);

delta\_ms = 1000 \* (T2.tv\_sec - T1.tv\_sec) + (T2.tv\_usec - T1.tv\_usec) / 1000;

printf("\nN=%d. Milliseconds passed: %ld\n", N, delta\_ms);

return 0;

}