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

**Факультет программной инженерии и компьютерной техники**

**Параллельные вычисления**

**Лабораторная работа №2**

**Исследование эффективности параллельных библиотек для С - программ**

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

**Преподаватель: Жданов А. Д.**

**Санкт-Петербург**

**2023**

Александров Юрий Витальевич

11 4 10

Этапы берутся согласно 1 лабораторной работе.

Для данной лабораторной работы необходимо скачать библиотеку AMD Framewave. Далее, используя данную библиотеку, необходимо изменить функции этапов Map и Merge, воспользовавшись Multi-Threaded функциями, и провести исследование на эффективность параллельной библиотеки.

**Характеристики компьютера:**

Процессор: Intel® Celeron N5095A, 2GHz

Количество ядер: 4

ОЗУ: 16 Gb, 2933 МГц

**Особенность установки библиотеки AMD Framewave:**

Сначала скачаем архив с библиотекой AMD Framewave.

curl -L <https://sourceforge.net/projects/framewave/files/framewave-releases/Framewave%201.3.1/FW_1.3.1_Lin64.tar.gz/download> > AMD\_Framewave\_Lin64.tar.gz

Далее разархивируем библиотеку в папку с проектом ко 2 лабораторной

tar xvzf AMD\_Framewave\_Lin64.tar.gz

Далее необходимо пройтись по всем файлам и сделать so, so.1 файлы, которые необходимы для вычислений

for FILE in \*; do

for EXT in "so.1" "so"; do

LIB\_FILE=$(echo $FILE | awk -F '.' '{print $1}')

ln -sf $FILE "$LIB\_FILE.$EXT"

done

done

В конце происходит экспорт полного пути к каталогу, который содержит библиотеку, которая будет задействована.

export LD\_LIBRARY\_PATH=/mnt/e/ITMO/Parallel\_Programming/LR\_2\_Aleksandrov/Parallel\_Aleksandrov\_lr2/Parallel\_Aleksandrov\_lr2/FW\_1.3.1\_Lin64/lib:$LD\_LIBRARY\_PATH

**Компиляция кода:**

gcc -O3 -Wall -Werror -o lab2.o lab2.c -I FW\_1.3.1\_Lin64 -c && gcc -O3 -L FW\_1.3.1\_Lin64/lib lab2.o -o lab2 -lm -lfwImage -lfwBase -lfwSignal

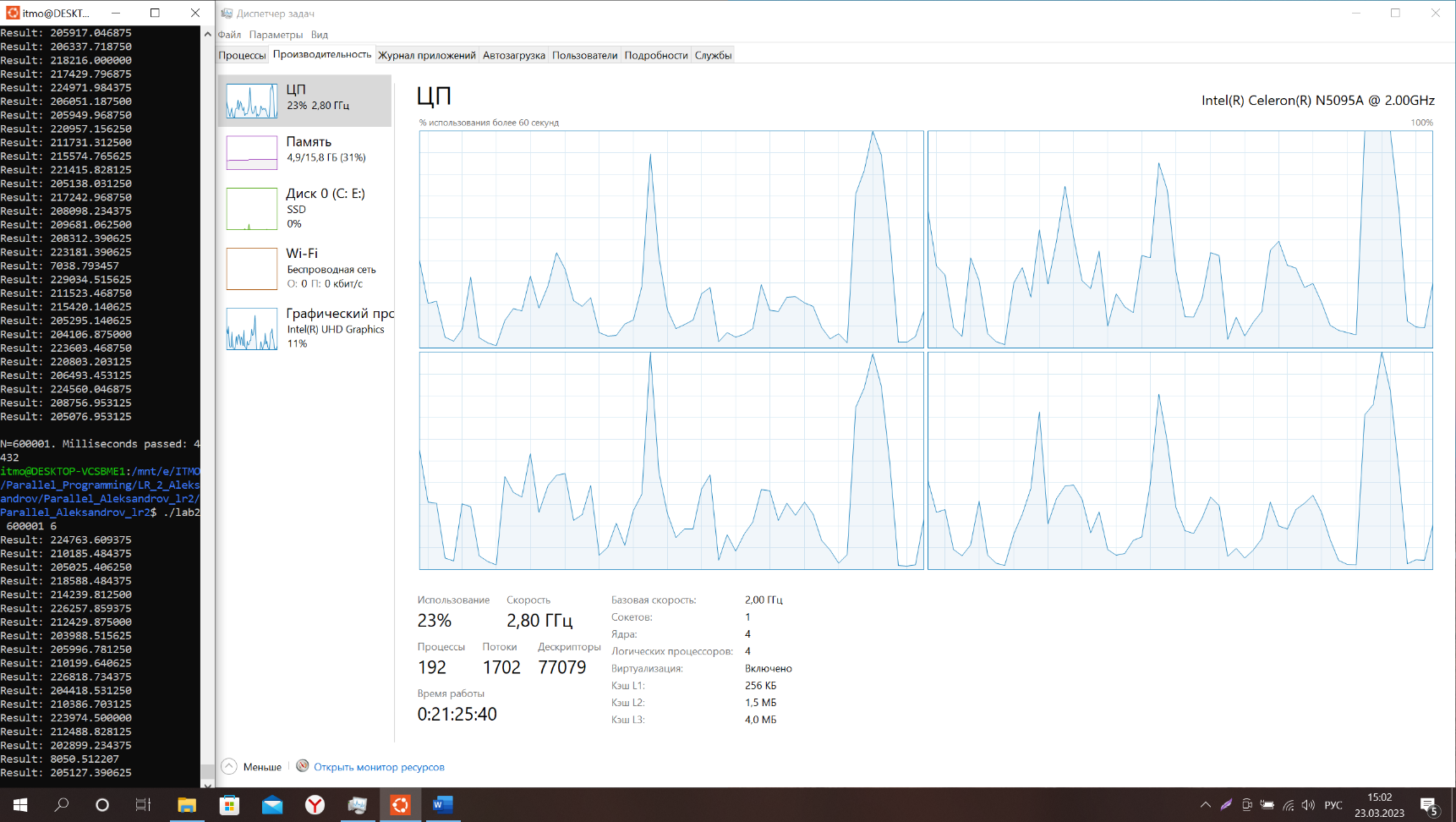
**Результаты исследований:**

Таблица 1 – Результаты исследований

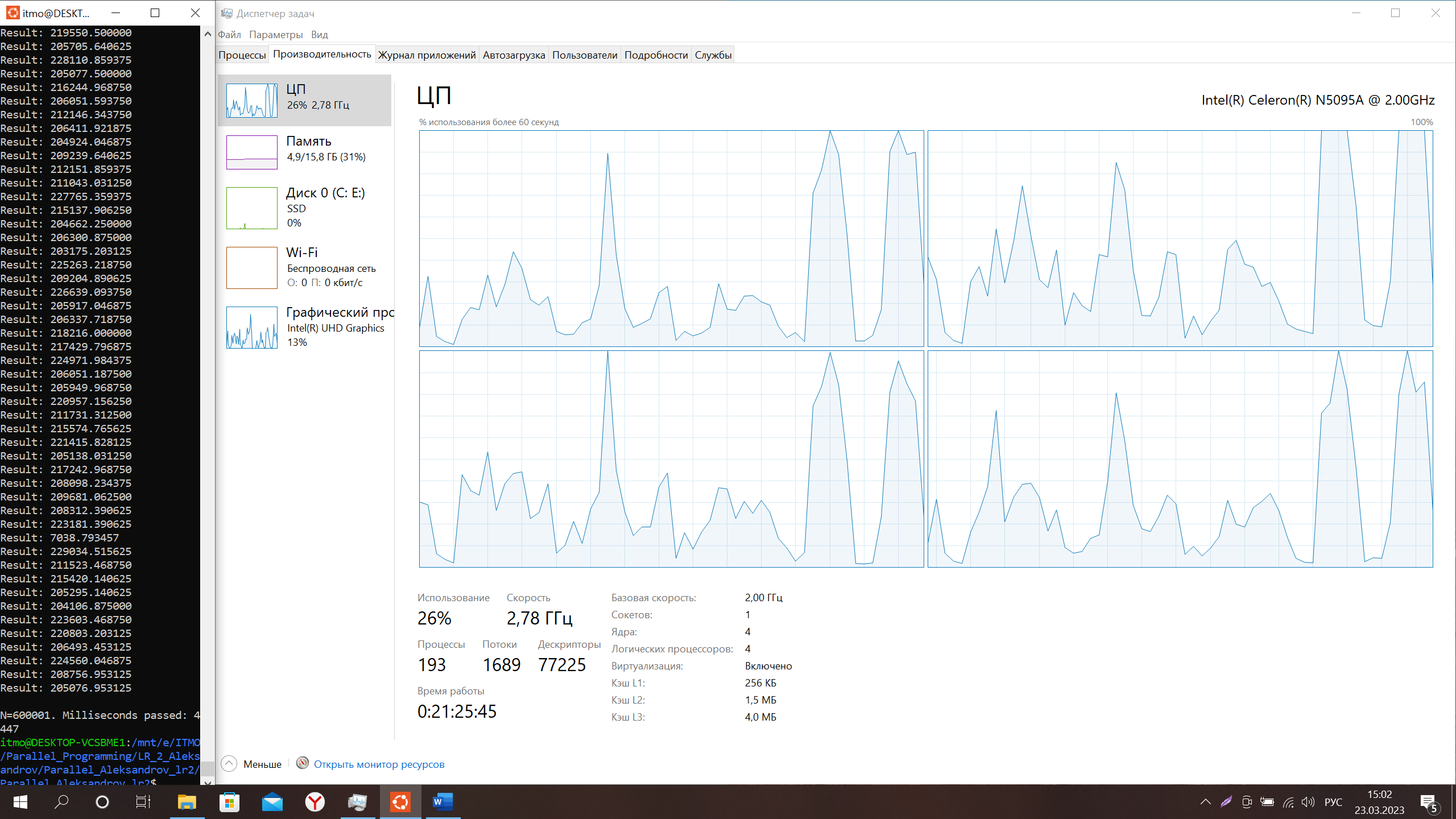
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **N** | **K = 1** | **K = 2** | **K = 3** | **K = 4** | **K = 6** | **K = 8** |
| 1 | 0,898 | 0,814 | 0,857 | 0,836 | 0,893 | 0,984 |
| 60001 | 1,517 | 1,336 | 1,324 | 1,309 | 1,297 | 1,097 |
| 120001 | 2,188 | 1,686 | 1,471 | 1,439 | 1,328 | 1,266 |
| 180001 | 2,939 | 2,127 | 1,924 | 1,678 | 1,643 | 1,648 |
| 240001 | 3,739 | 2,65 | 2,416 | 2,07 | 2,095 | 2,062 |
| 300001 | 4,527 | 3,1 | 2,741 | 2,479 | 2,457 | 2,454 |
| 360001 | 5,348 | 3,603 | 3,14 | 2,867 | 2,86 | 2,826 |
| 420001 | 6,143 | 4,1 | 3,555 | 3,229 | 3,203 | 3,17 |
| 480001 | 6,893 | 4,587 | 3,941 | 3,568 | 3,572 | 3,535 |
| 540001 | 7,699 | 5,073 | 4,348 | 3,987 | 3,926 | 3,928 |
| 600001 | 8,452 | 5,719 | 4,687 | 4,286 | 4,285 | 4,26 |

**K = 6**

Начало:

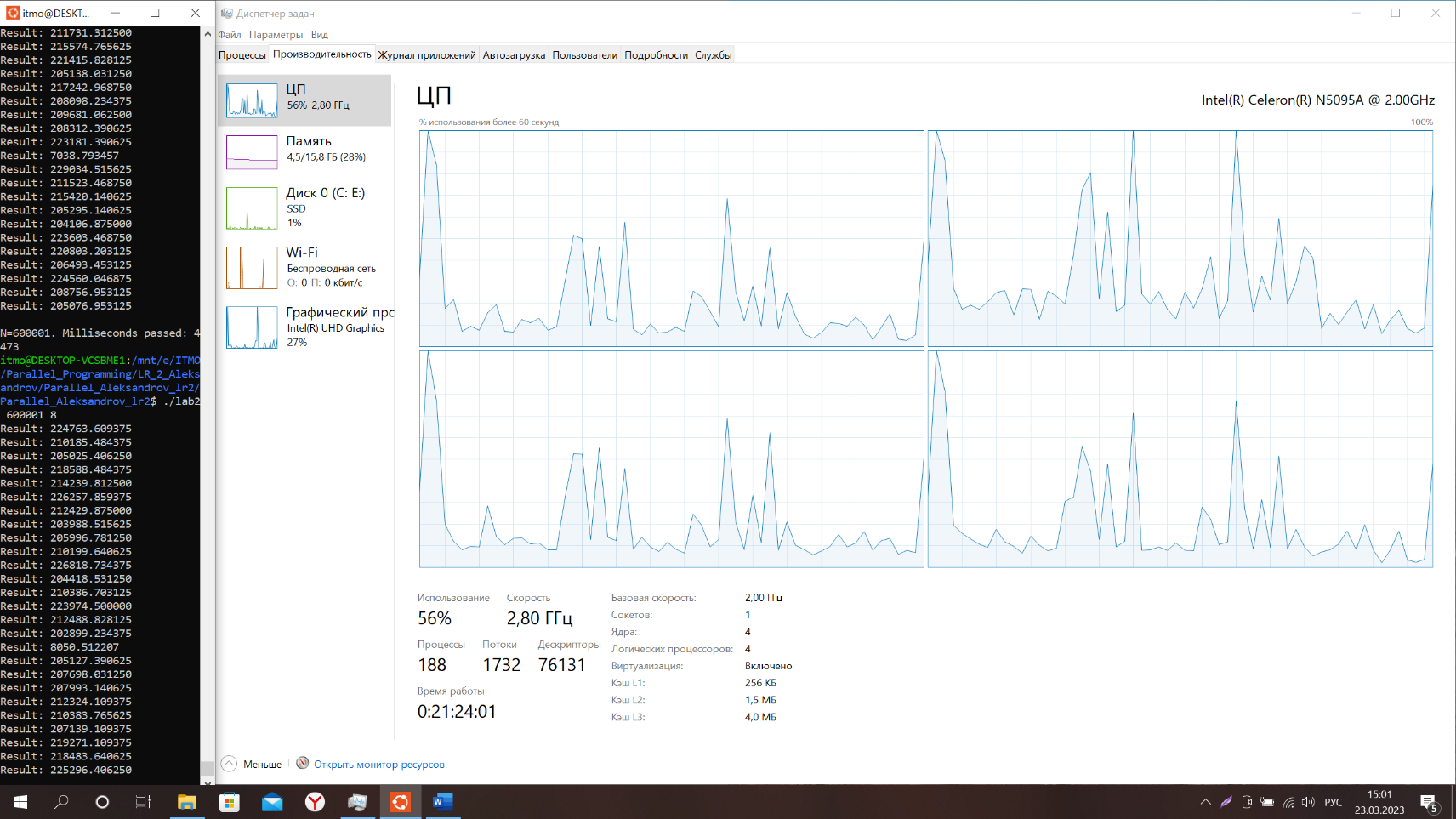


Конец:

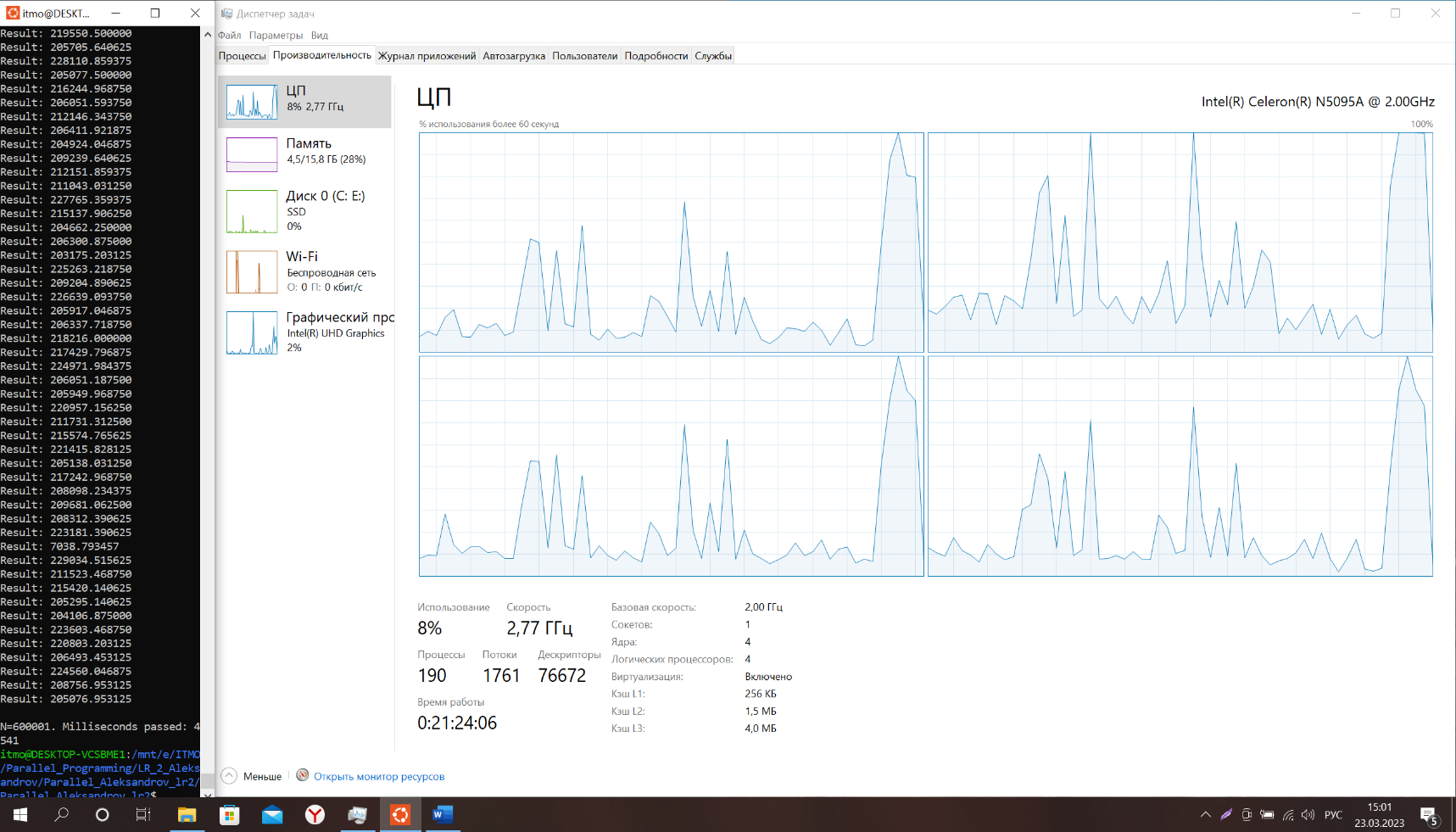


**K = 8**

Начало:



Конец:



# Оценка коэффициента распараллеливания

Рассчитаем коэффициент распараллеливания, он будет равен:

Где, время выполнения на p потоках, время выполнения на 1 потоке, кол-во потоков.

Важное ограничение – данная формула не учитывает накладные расходы, t(p) не может быть больше t(1). Это связано с допущением закона Амдала: распараллеленный код будет давать линейный прирост скорости работы при изменении количества вычислителей p от 0 до .

Получены следующие коэффициенты параллельного ускорения:

Таблица 2 – Коэффициенты распараллеливания

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

Теперь подсчитаем получившуюся параллельную эффективность:

Таблица 3 – Параллельная эффективность

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

**Вывод:** в ходе выполнения работы было проведено распраллеливание процесса при помощи библиотеки AMD FrameWave.В результате исследования, проанализировав результаты из прошлой работы, сравнив с компиляторами CLANG, а также GCC, библиотека AMD FrameWave производит более эффективное распараллеливание чем при автоматическом распралаллеливании.

Также можно сделать вывод, что при увеличении потоков больше чем количества физических ядер не происходит ускорения. Скорее всего система просто выделяет максимальное количество физических ядер, хотя указано большее количество потоков.

Параллельная эффективность выполнения программы уменьшается при увеличении передаваемого числа ядер.

**Приложение**

#include <stdio.h>

#include <stdlib.h>

#include <sys/time.h>

#include <math.h>

#include <time.h>

#include "FW\_1.3.1\_Lin64/fwBase.h"

#include "FW\_1.3.1\_Lin64/fwSignal.h"

// 2 Этап

// Применение операции соответсвенно 4 варианту к массиву М1

void func\_map\_m1(int sizeArrayM1, float M1[]) {

float\* arrayCosh = (float\*)malloc(sizeArrayM1 \* sizeof(float));

float\* arraySinh = (float\*)malloc(sizeArrayM1 \* sizeof(float));

fwsSqrt\_32f(M1, M1, sizeArrayM1);

fwsCosh\_32f\_A11(M1, arrayCosh, sizeArrayM1);

fwsSinh\_32f\_A11(M1, arraySinh, sizeArrayM1);

fwsDiv\_32f(arrayCosh, arraySinh, M1, sizeArrayM1);

free(arraySinh);

free(arrayCosh);

}

// Применение операции соответсвенно 2 варианту к массиву М2

void func\_map\_m2(int sizeArrayM2, float M2[], float M1[]) {

float\* M2\_tmp = (float\*)malloc(sizeArrayM2 \* sizeof(float));

fwsSqrt\_32f\_A11(M1, M1, sizeArrayM2 \* 2);

fwsCos\_32f\_A11(M1, M1, sizeArrayM2 \* 2);

fwsInv\_32f\_A11(M1, M1, sizeArrayM2 \* 2);

fwsAdd\_32f(M2\_tmp, &M2\_tmp[1], &M2[1], sizeArrayM2 - 1);

fwsCos\_32f\_A11(M2, M2, sizeArrayM2);

fwsAbs\_32f\_I(M2, sizeArrayM2);

free(M2\_tmp);

}

// 3 Этап

// Merge согласно 6 варинту

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

fwsSub\_32f(M1, M2, M2, sizeArrayM2);

fwsAbs\_32f(M2, M2, sizeArrayM2);

}

// 4 Этап

// Сортировка stupid sort согласно 4 варианту

int isSorted(float M2[], int sizeArrayM2)

{

while (--sizeArrayM2 > 1)

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

return 0;

return 1;

}

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

{

float tmp = \*x;

\*x = \*y;

\*y = tmp;

}

void shuffle(float M2[], int sizeArrayM2)

{

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

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

}

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

{

while (!isSorted(M2, sizeArrayM2))

shuffle(M2, sizeArrayM2);

}

// Этап reduce

// ПРИХОДИТ ОТСОРТИРОВАННЫЙ МАССИВ!!!

double reduce(float M2[], int sizeArrayM2) {

float amount = 0;

float min = 0;

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

if (M2[i] != 0) {

min = M2[i];

break;

}

}

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

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

amount += sin(M2[k]);

}

}

return amount;

}

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]);

fwSetNumThreads(numberFlow);

gettimeofday(&T1, NULL);

unsigned int first\_random = 0;

unsigned int second\_random = 0;

// 100 экспериментов

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

srand(i);

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

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

// 1 Этап

// Заполение массива M1

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

float 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++) {

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

M2[k] = value;

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

}

// 2 Этап

func\_map\_m1(N, M1);

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

printf("%.2f\n", M1[k]);

}\*/

func\_map\_m2(N / 2, M2, M1);

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

printf("%.2f\n", M2[k]);

}\*/

// 3 Этап

merge(M1, M2, N / 2);

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

printf("%.2f\n", M2[k]);

}\*/

// 4 Этап

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

// 5 Этап

float result = reduce(M2, N / 2);

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

free(M1);

free(M2);

}

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;

}