Федеральное государственное автономное образовательное учреждение высшего образования

«**Университет ИТМО»**

**Факультет ПИиКТ**

**Дисциплина: Параллельные вычисления**

**Лабораторная работа 4**

OpenMP сортировка

Выполнил: Гурин Евгений Иванович

Преподаватель: Жданов Андрей Дмитриевич

Группа: Р4116

Санкт-Петербург 2023г.

## Задача

## Конфигурация

Host Name: EGURIN-PC

OS Name: Microsoft Windows 11 Pro

OS Version: 10.0.22000 N/A Build 22000

OS Manufacturer: Microsoft Corporation

OS Configuration: Standalone Workstation

OS Build Type: Multiprocessor Free

Registered Owner: user

Registered Organization: N/A

Product ID: 00331-10000-00001-AA539

Original Install Date: 02.10.2022, 21:59:41

System Boot Time: 20.03.2023, 2:46:00

System Manufacturer: ASUS

System Model: System Product Name

System Type: x64-based PC

Processor(s): 1 Processor(s) Installed.

[01]: AMD64 Family 23 Model 113 Stepping 0 AuthenticAMD ~3600 Mhz

BIOS Version: American Megatrends Inc. 2803, 27.04.2022

Windows Directory: C:\Windows

System Directory: C:\Windows\system32

Boot Device: \Device\HarddiskVolume2

System Locale: en-us;English (United States)

Input Locale: en-us;English (United States)

Time Zone: (UTC+03:00) Moscow, St. Petersburg

Total Physical Memory: 32 679 MB

Available Physical Memory: 20 506 MB

Virtual Memory: Max Size: 87 975 MB

Virtual Memory: Available: 19 470 MB

Virtual Memory: In Use: 68 505 MB

Page File Location(s): D:\pagefile.sys

Domain: WORKGROUP

Logon Server: \\EGURIN-PC

Hotfix(s): 5 Hotfix(s) Installed.

[01]: KB5022505

[02]: KB5012170

[03]: KB5023698

[04]: KB5022369

[05]: KB5022925

Network Card(s): 4 NIC(s) Installed.

[01]: Realtek PCIe 2.5GbE Family Controller

Connection Name: Ethernet

Status: Media disconnected

[02]: Intel(R) Wi-Fi 6 AX200 160MHz

Connection Name: Wi-Fi

DHCP Enabled: Yes

DHCP Server: 192.168.1.1

IP address(es)

[01]: 192.168.1.47

[02]: fe80::933b:210e:a9a7:2c6e

[03]: Bluetooth Device (Personal Area Network)

Connection Name: Bluetooth Network Connection

Status: Media disconnected

[04]: VirtualBox Host-Only Ethernet Adapter

Connection Name: Ethernet 2

DHCP Enabled: No

IP address(es)

[01]: 192.168.56.1

[02]: fe80::527e:5766:393d:acc6

Hyper-V Requirements: A hypervisor has been detected. Features required for Hyper-V will not be displayed.

## Результаты работы

Для экспериментов был выбран компилятор clang

**Прямая совместимость** была достигнута с помощью проверки

#ifdef \_OPENMP

// here omp\_get\_num\_procs implementation exists

#else

int omp\_get\_num\_procs() { return 1; }

double omp\_get\_wtime() {

struct timeval t;

gettimeofday(&t, NULL);

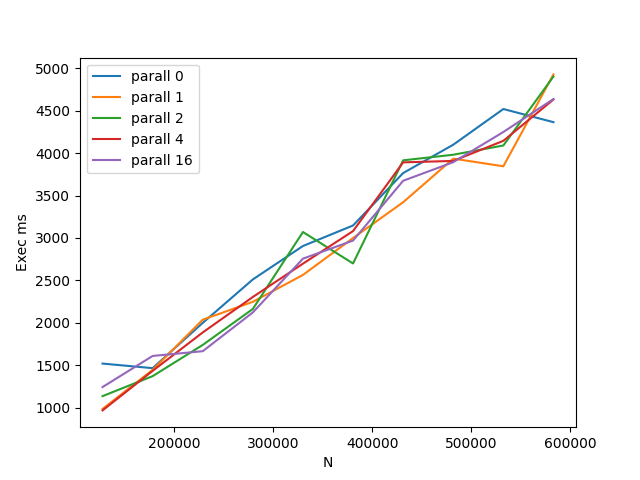
return t.tv\_sec + t.tv\_usec / 1000000.0;

}

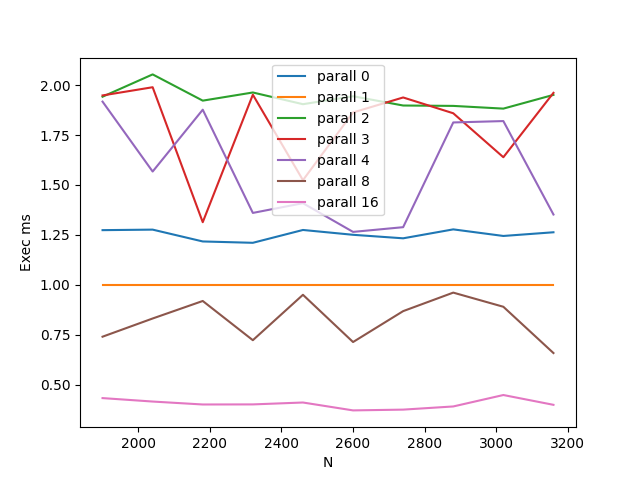
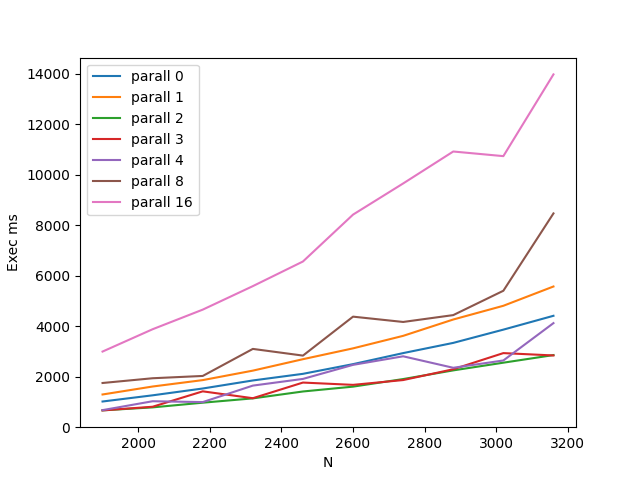
#endif

**Результаты эксперементов**

**CLANG (автоматизированное распараллеливание)**

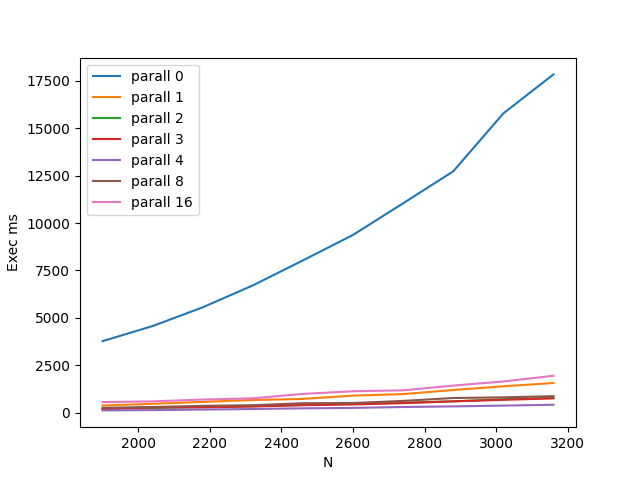


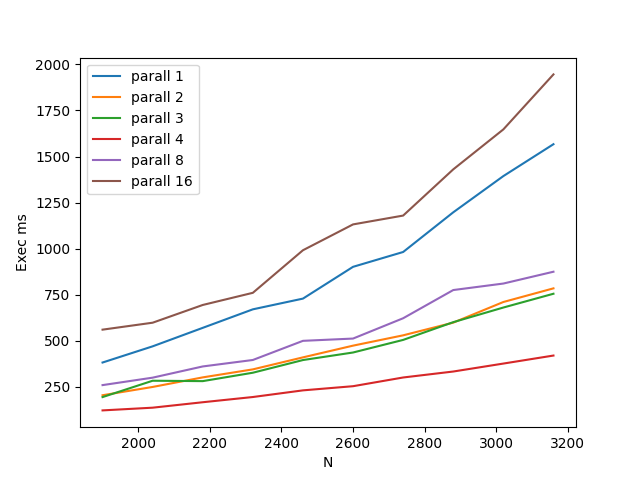
**Разделение сортировки на 2 секции**

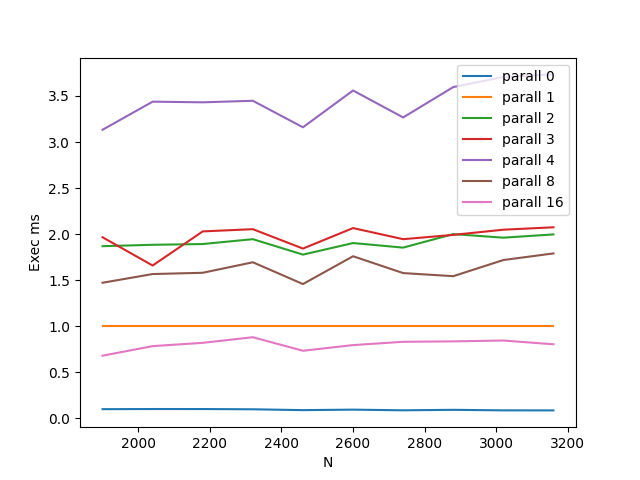
****

Можно заметить, что разделение сортировки на 2 потока даёт большой прирост (сравнивая 1 и 2 потока). Далее прироста нет, так как для сортировки потока используется 2 секции в любом случае и количество общего числа потоков не влияет существенно.

**Разделение сортировки на 4 секции**

****

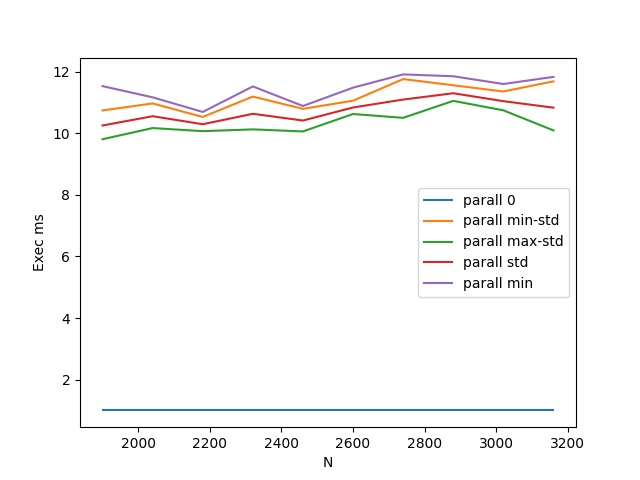
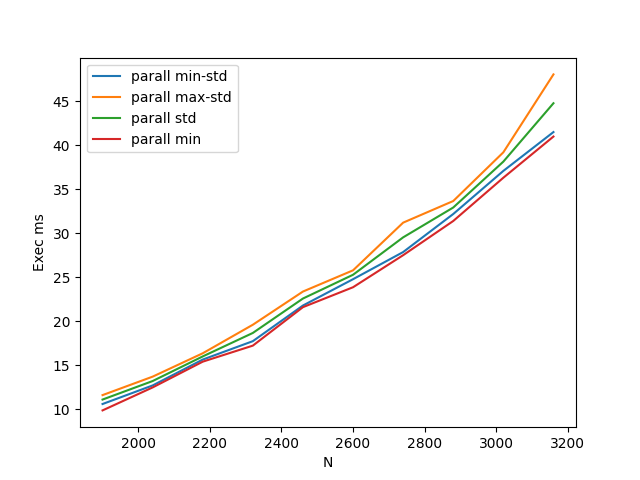




Разделение сортировки на большее количество секций даёт отличный прирост и в данном случае уже имеет влияние общее число потоков и виден рост производительности до 4-х потоков, далее наблюдается ухудшение.

**Доверительный интервал и замеры на основе минимального**

**Эксперименты проводились для 4 потоков и разделения сортировки на 4 секции**



# Листинг main.c

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <unistd.h>

#include <sys/time.h>

#include <omp.h>

#include <string.h>

#define min(a,b) (((a) < (b)) ? (a) : (b))

// #define DEBUG 1

#ifdef \_OPENMP

// here omp\_get\_num\_procs implementation exists

#else

int omp\_get\_num\_procs() { return 1; }

double omp\_get\_wtime() {

struct timeval t;

gettimeofday(&t, NULL);

return t.tv\_sec + t.tv\_usec / 1000000.0;

}

#endif

void swap(double \*a, double \*b) {

double t;

t = \*a, \*a = \*b, \*b = t;

}

void print\_arr(double \*array, int n) {

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

{

printf("%f ", array[i]);

}

printf("\n");

}

void sort\_stupid(double \*array, int n) {

#ifdef DEBUG

printf("sort stupid\n");

print\_arr(array, n);

#endif

int i = 0;

while (i < n - 1) {

if (array[i + 1] < array[i]) swap(array + i, array + i + 1), i = 0;

else i++;

}

#ifdef DEBUG

printf("sort stupid end\n");

print\_arr(array, n);

#endif

}

void merge\_sorted(double \*src1, int n1, double \*src2, int n2, double \*dst) {

#ifdef DEBUG

printf("merge\_sorted\n");

print\_arr(src1, n1);

print\_arr(src2, n2);

#endif

int i = 0, i1 = 0, i2 = 0;

while (i < n1 + n2) {

dst[i++] = src1[i1] > src2[i2] && i2 < n2 ? src2[i2++] : src1[i1++];

}

#ifdef DEBUG

printf("merge\_sorted end\n");

print\_arr(dst, n1 + n2);

#endif

}

void copy\_array(double \*dst, double \*src, int n) {

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

{

dst[i] = src[i];

}

}

void sort(double \*array, int n, double \*dst) {

#ifdef DEBUG

printf("sort\n");

print\_arr(array, n);

#endif

int n1 = n / 2;

int n2 = n - n1;

#pragma omp sections

{

#pragma omp section

{

sort\_stupid(array, n1);

}

#pragma omp section

{

sort\_stupid(array + n1, n2);

}

}

merge\_sorted(array, n1, array + n1, n2, dst);

#ifdef DEBUG

printf("sort end\n");

print\_arr(dst, n);

#endif

}

void sort\_dynamic(double \*array, int n, double \*dst, int n\_threads) {

#ifdef DEBUG

printf("sort\_dynamic\n");

print\_arr(array, n);

#endif

int n\_chunk = n\_threads < 2 ? n : ceil((double) n / n\_threads);

#pragma omp for

{

for (int k = 0; k < n\_threads; ++k)

{

int n\_done = n\_chunk \* k;

int n\_cur\_chunk = min((n - n\_done), n\_chunk);

// for debug

#ifdef DEBUG

printf("parallel for k: %d n\_chunk: %d n\_done: %d n\_cur\_chunk: %d\n", k, n\_chunk, n\_done, n\_cur\_chunk);

#endif

sort\_stupid(array + n\_done, n\_cur\_chunk);

}

}

double \* restrict cpy = malloc(n \* sizeof(double));

copy\_array(cpy, array, n);

copy\_array(dst, array, n);

for (int k = 1; k < n\_threads; ++k)

{

int n\_done = n\_chunk \* k;

int n\_cur\_chunk = min(n - n\_done, n\_chunk);

int n\_will\_done = n\_done + n\_cur\_chunk;

merge\_sorted(cpy, n\_done, array + n\_done, n\_cur\_chunk, dst);

copy\_array(cpy, dst, n\_will\_done);

}

#ifdef DEBUG

printf("sort\_dynamic end\n");

print\_arr(dst, n);

#endif

}

void print\_delta(double T1, double T2) {

printf("\n%f\n", (T2 - T1) \* 1000.0);

}

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

double T1, T2;

T1 = omp\_get\_wtime();

int finished = 0;

int i = 0;

#pragma omp parallel sections num\_threads(2) shared(i, finished)

{

#ifdef \_OPENMP

#pragma omp section

{

double time = 0;

while (finished < 1) {

double time\_temp = omp\_get\_wtime();

if (time\_temp - time < 1) {

usleep(100);

continue;

};

printf("\nPROGRESS: %d\n", i);

time = time\_temp;

}

}

#endif

#pragma omp section

{

const int N = atoi(argv[1]); /\* N - array size, equals first cmd param \*/

const int N\_sort\_threads = argc > 3 ? atoi(argv[3]) : 2;

const int N\_2 = N / 2;

const int A = 280;

double \* restrict m1 = malloc(N \* sizeof(double));

double \* restrict m2 = malloc(N\_2 \* sizeof(double));

double \* restrict m2\_cpy = malloc(N\_2 \* sizeof(double));

#if defined(\_OPENMP)

omp\_set\_dynamic(0);

const int M = atoi(argv[2]); /\* M - amount of threads \*/

omp\_set\_num\_threads(M);

#endif

for (i = 0; i < 100; i++) /\* 100 экспериментов \*/

{

double X = 0;

unsigned int seedp = i;

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

m1[j] = (rand\_r(&seedp) % (A \* 100)) / 100.0 + 1;

}

// generate 2

for (int j = 0; j < N\_2; ++j) {

m2[j] = A + rand\_r(&seedp) % (A \* 9);

}

#pragma omp parallel default(none) shared(N, N\_2, A, m1, m2, m2\_cpy, i, X, N\_sort\_threads)

{

#pragma omp for

for (int j = 0; j < N\_2; ++j) {

m2\_cpy[j] = m2[j];

}

// map

#pragma omp for

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

m1[j] = 1 / tanh(sqrt(m1[j]));

}

#pragma omp for

for (int j = 1; j < N\_2; ++j) {

m2[j] = m2[j] + m2\_cpy[j - 1];

}

#pragma omp for

for (int j = 1; j < N\_2; ++j) {

m2[j] = pow(log10(m2[j]), M\_E);

}

#pragma omp for

for (int j = 0; j < N\_2; ++j) {

m2\_cpy[j] = m2[j] > m1[j] ? m2[j] : m1[j] ;

}

if (N\_sort\_threads == 2) {

sort(m2\_cpy, N\_2, m2);

} else {

sort\_dynamic(m2\_cpy, N\_2, m2, omp\_get\_num\_procs());

}

int k = 0;

while (m2[k] == 0 && k < N\_2 - 1) k++;

double m2\_min = m2[k];

#ifdef DEBUG

print\_arr(m2, N\_2);

printf("min %f\n", m2\_min);

#endif

// reduce

#pragma omp for

for (int j = 0; j < N\_2; ++j) {

m2\_cpy[j] = 0;

if((int)(m2[j] / m2\_min) % 2 == 0) m2\_cpy[j] = sin(m2[j]);

}

#pragma omp for reduction(+ : X)

for (int j = 0; j < N\_2; ++j) {

X += m2\_cpy[j];

}

#pragma omp barrier

}

printf("%f ", X);

}

finished = 1;

}

}

T2 = omp\_get\_wtime();

print\_delta(T1, T2);

return 0;

}

# Вывод

Разделение сортировки на 2 секции привело к значительному увеличению производительности, однако при увеличении числа потоков более 4-х наблюдается ухудшение.

При использовании разделения сортировки на количество секции, соответствующее количеству вычислителей производительность увеличивается гораздо сильнее и увеличение количества потоков гораздо сильнее влияет на выполнение программы.