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: P4116

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

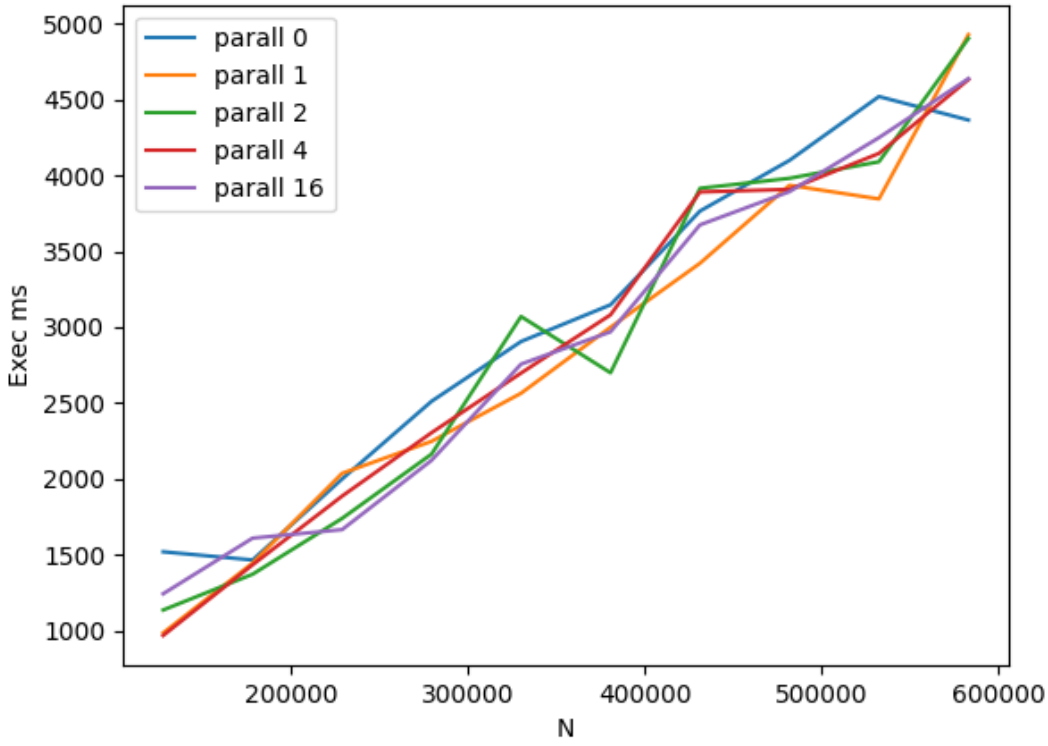
Для экспериментов был выбран компилятор clang

Прямая совместимость была достигнута с помощью проверки

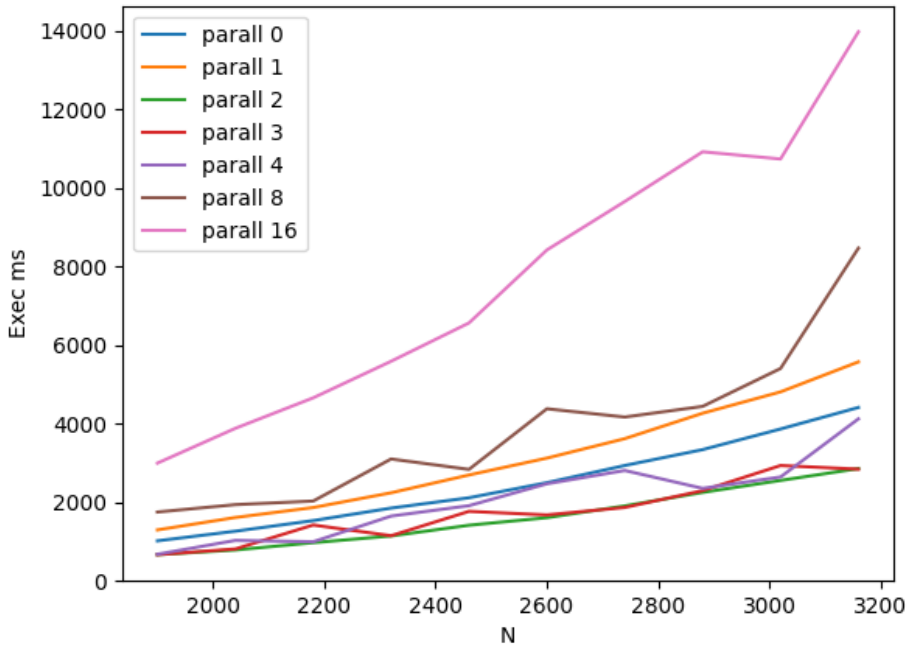
```
#ifdef _OPENMP
    // here omp_get_num_procs implementation exists
#else
    int omp_get_num_procs() { return 1; }
    double omp_get_wtime() {
        struct timeval t;
        gettimeofday(&t, NULL);
        return t.tv_sec + t.tv_usec / 1000000.0;
    }
#endif
```

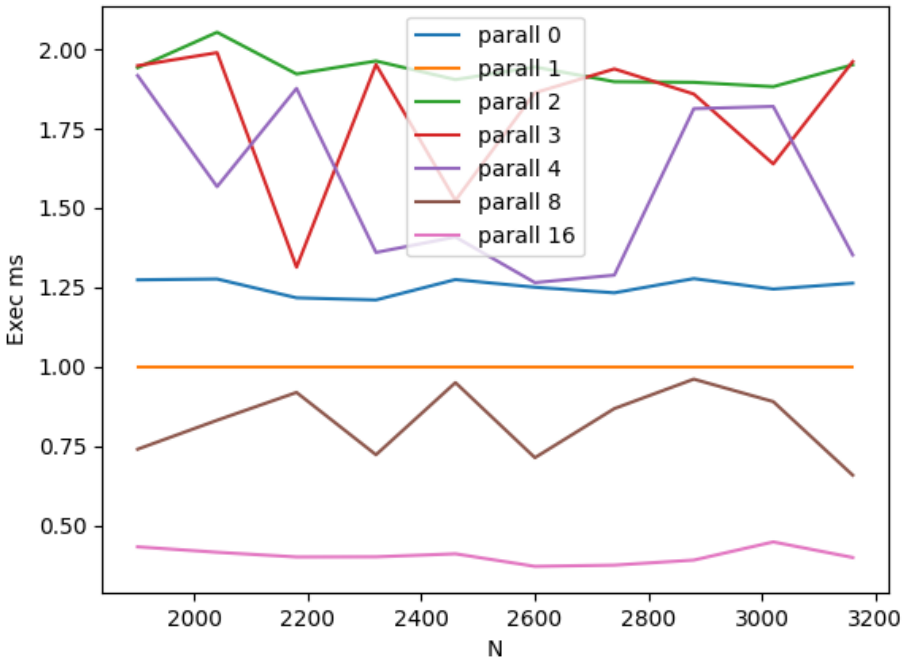
Результаты эксперементов

CLANG (автоматизированное распараллеливание)



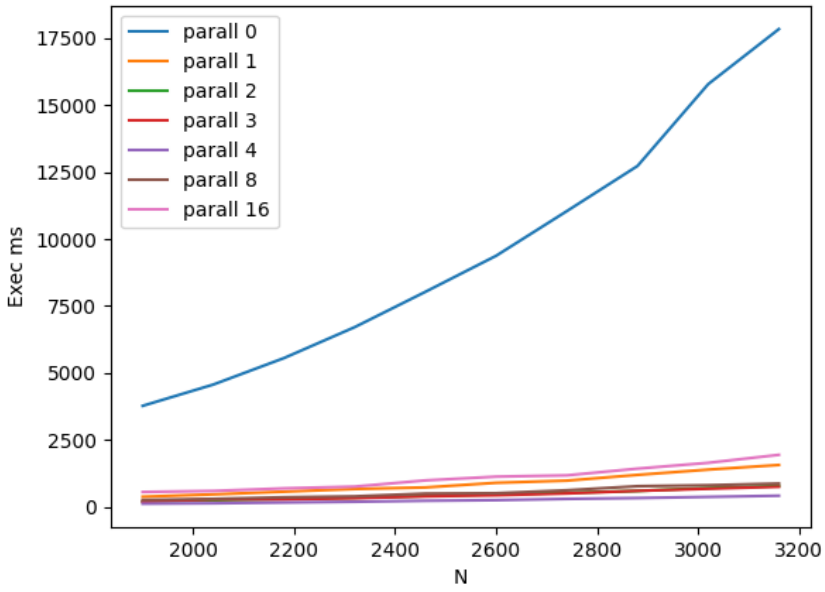
Разделение сортировки на 2 секции

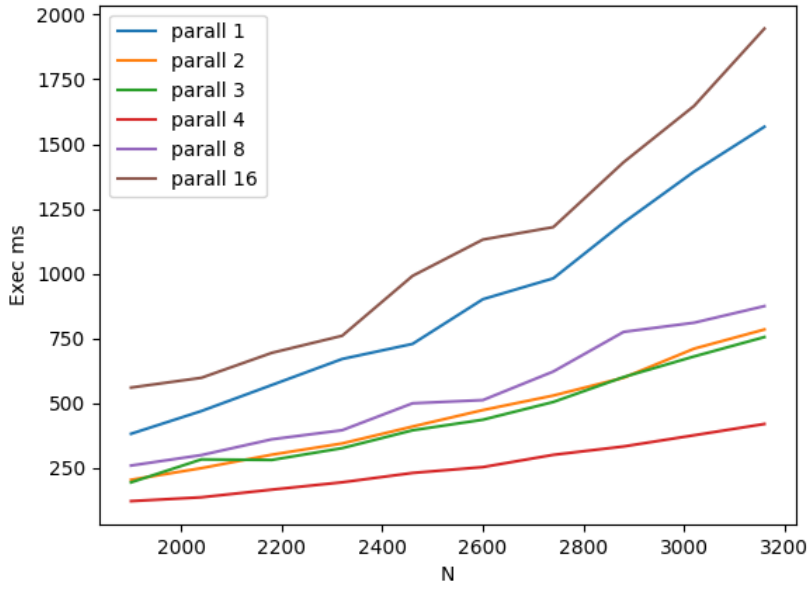


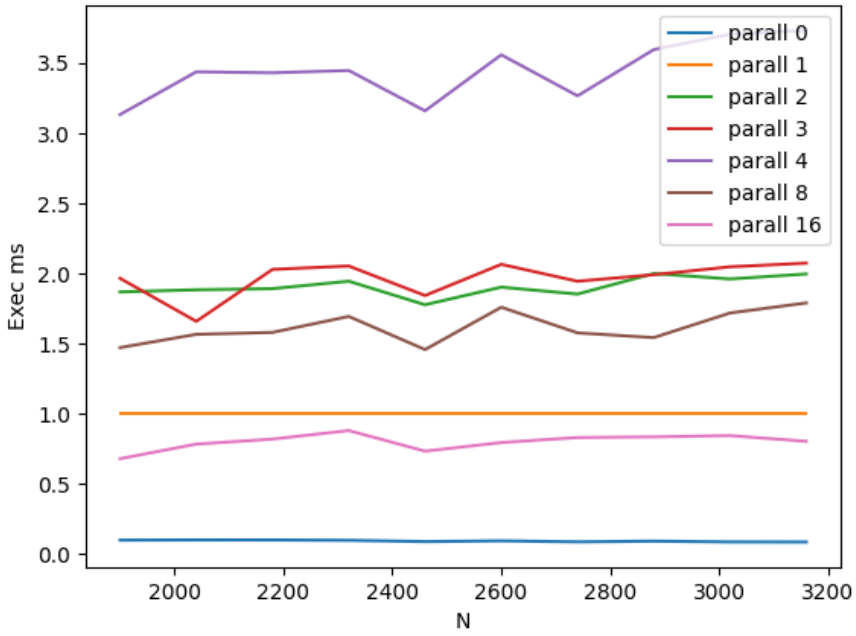


Можно заметить, что разделение сортировки на 2 потока даёт большой прирост (сравнивая 1 и 2 потока). Далее прироста нет, так как для сортировки потока используется 2 секции в любом случае и количество общего числа потоков не влияет существенно.

Разделение сортировки на 4 секции



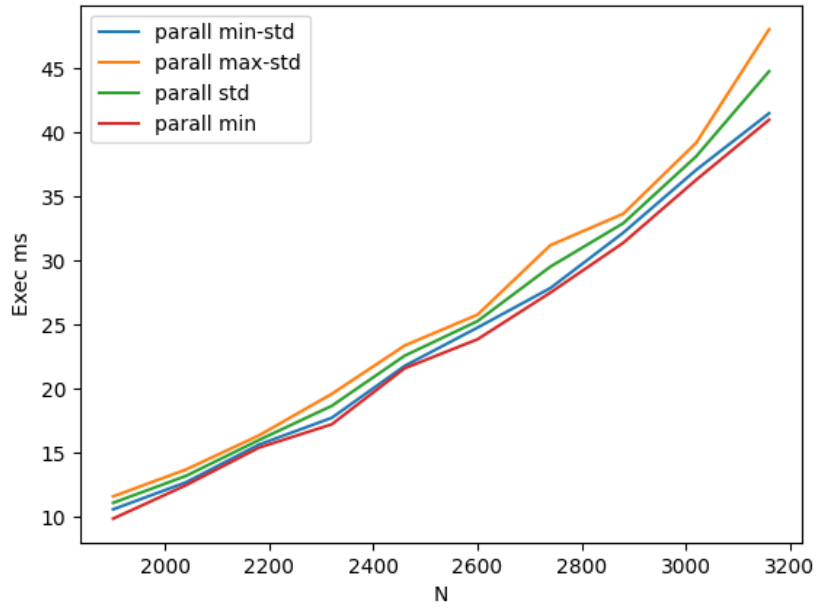


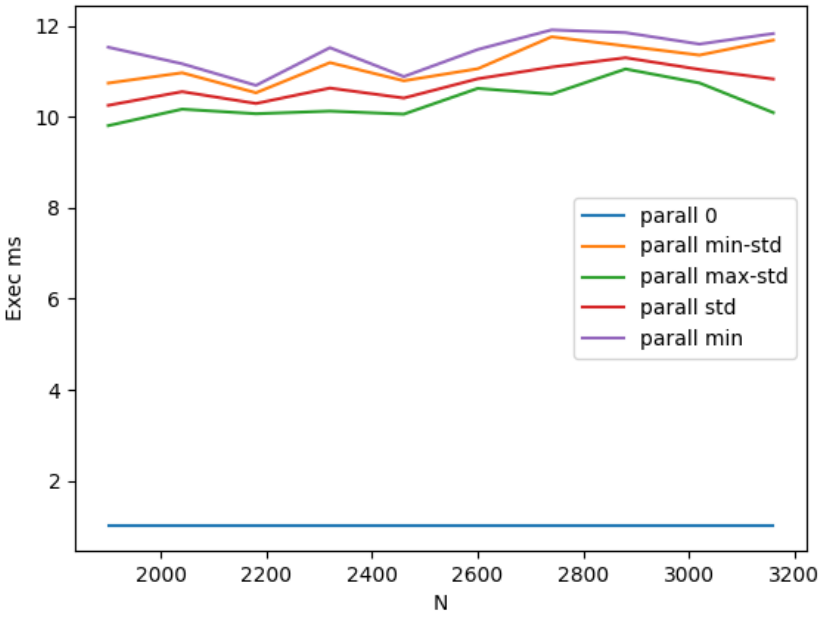


Разделение сортировки на большее количество секций даёт отличный прирост и в данном случае уже имеет влияние общее число потоков и виден рост производительности до 4-х потоков, далее наблюдается ухудшение.

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.