**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”**

**Факультет** Программной инженерии и компьютерной техники

**Направление подготовки (специальность)** Системное и прикладное ПО

# ОТЧЕТ

Лабораторная работа №3
по предмету «Параллельные вычисления»

Тема проекта: «Распараллеливание циклов с помощью технологии OpenMP».

Обучающийся Ткаченко В.В. (**Фамилия И.О.**) Р4114 (**номер группы**)

Преподаватель Жданов А. Д. (**Фамилия И.О.**)

Санкт-Петербург
2023 г.

**Содержание**

## Описание решаемой задачи

Вариант: (576)

Map: 6 | 5

Merge: 1

Sort: 6

1. Добавить во все for-циклы (кроме цикла в функции main, указывающего количество экспериментов) в программе из ЛР №1 следующую директиву OpenMP: ”#pragma omp parallel for default(none) private(...) shared(...)”. Наличие параметра default(none) является обязательным.

2. Проверить все for-циклы на внутренние зависимости по данным между итерациями. Если зависимости обнаружились, использовать для защиты критических секций директиву ”#pragma omp critical” или ”#pragma omp atomic” (если операция атомарна), или параметр reduction (предпочтительнее) или вообще отказаться от распараллеливания цикла (свой выбор необходимо обосновать).

3. Убедиться, что получившаяся программа обладает свойством прямой совместимости с компиляторами, не поддерживающими OpenMP (для проверки этого можно скомпилировать программу без опции ”–fopenmp”, в результате не должно быть сообщений об ошибках, а программа должна корректно работать).

4. Использовать функцию SetNumThreads для изменения числа потоков. В отчете указать максимальное количество потоков.

5. Провести эксперименты, замеряя параллельное ускорение. Привести сравнение графиков параллельного ускорения с ЛР №1 и ЛР №2.

6. Провести эксперименты, добавив параметр ”schedule” и варьируя в экспериментах тип расписания. Исследование нужно провести для всех

возможных расписаний: static, dynamic, guided. С 4 вариантами chunck_size равными: единице, меньше чем число потоков, равному числу потоков и больше чем число потоков. Привести сравнение параллельного ускорения при различных расписаниях с результатами п.4.

7. Определить какой тип расписания на машине при использовании ”schedule” ”default”. 91

8. Выбрать из рассмотренных в п.4 и п.5 наилучший вариант при различных N. Сформулировать условия, при которых наилучшие результаты получились бы при использовании других типов расписания.

9. Найти вычислительную сложность алгоритма до и после распараллеливания, сравнить полученные результаты.

10. Для иллюстрации того, что программа действительно распараллелилась, привести график загрузки процессора (ядер) от времени при выполнении программы при N = N1 для лучшего варианта распараллеливания. Для получения графика можно как написать скрипт так и просто сделать скриншот диспетчера задач, указав на скриншоте моменты начала и окончания эксперимента (в отчёте нужно привести текст скрипта или название использованного диспетчера). Недостаточно привести однократное моментальное измерение загрузки утилитой htop, т.к. требуется привести график изменения загрузки за всё время выполнения программы.

11. Написать отчёт о проделанной работе.

12. Подготовиться к устным вопросам на защите.

**Краткая характеристика «железа»**

| | |
|---|---|
| Имя ОС: | Майкрософт Windows 10 Pro |
| Версия: | 10.0.19045 Сборка 19045 |
| Изготовитель: | LENOVO |

Модель: 20BE009ART

Тип: Компьютер на базе x64

SKU системы: LENOVO_MT_20BE

Процессор: Intel(R) Core(TM) i7-4710MQ

Версия BIOS: LENOVO GMET85WW (2.33 ), 30.05.2018

Версия SMBIOS: 2.7

Версия встроенного контроллера: 1.14

Режим BIOS: Устаревший

gcc version: 11.3.0 (Ubuntu 11.3.0-1ubuntu1~22.04)

WSL2

## Листинг программы lab3.c

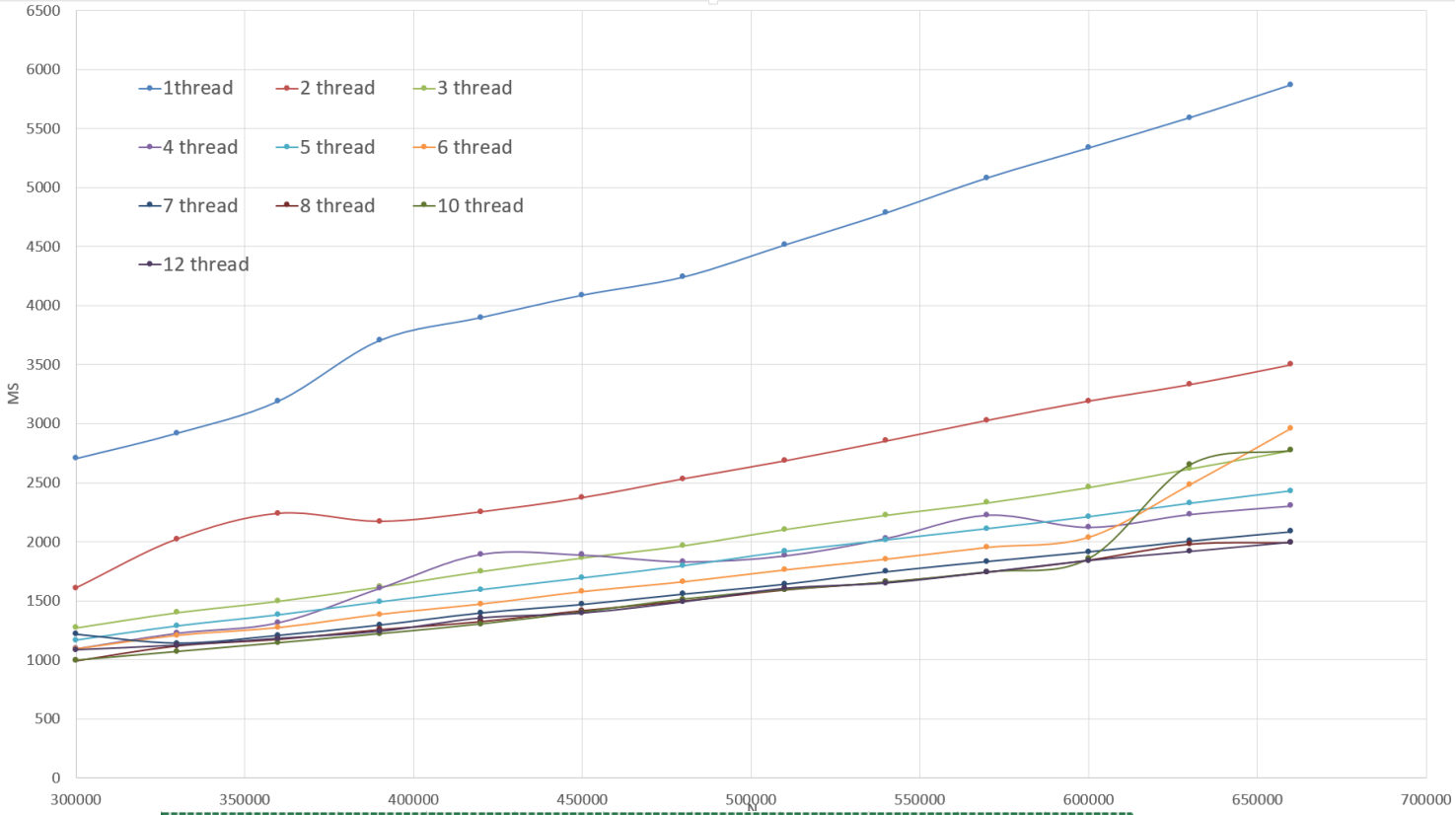
```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#include <omp.h>
#include <string.h>

int main(int argc, char *argv[])
{
    int i, j, N, M;
    // int CHUNKSIZE;
    // const char *SCHEDULE;
    double e = exp(1.0);
    struct timeval T1, T2;
    long delta_ms, extra2;
    double min = 1;
    double max = 576;

    N = atoi(argv[1]);
    M = atoi(argv[2]);
    // SCHEDULE = argv[3];

    // enum omp_sched_t schedule_type;
    // if (strcmp(SCHEDULE, "static") == 0)
    // {
    //     schedule_type = omp_sched_static;
    // }
    // else if (strcmp(SCHEDULE, "dynamic") == 0)
    // {
    //     schedule_type = omp_sched_dynamic;
    // }
    // else if (strcmp(SCHEDULE, "guided") == 0)
    // {
    //     schedule_type = omp_sched_guided;
    // }
    // else if (strcmp(SCHEDULE, "auto") == 0)
    // {
    //     schedule_type = omp_sched_auto;
    // }
    // CHUNKSIZE = atoi(argv[4]);

#if defined(_OPENMP)
    omp_set_dynamic(0);
    omp_set_num_threads(M);
    // omp_set_schedule(schedule_type, CHUNKSIZE);
#endif

    double *restrict M1 = (double *)malloc(N * sizeof(double));
```

```
    double *restrict M2 = (double *)malloc(N / 2 * sizeof(double));
    double *restrict M2_copy = (double *)malloc(N / 2 * sizeof(double));

    unsigned int seed;
    unsigned int *restrict seed1 = &seed;
    unsigned int *restrict seed2 = &seed;

    gettimeofday(&T1, NULL);

    for (i = 0; i < 100; i++)
    {
        double min_nonzero = INFINITY;
        double sum_sin = 0.0;
        seed = i;

        for (j = 0; j < N; j++)
        {
            // делим каждый член на e и считаем кубический корень
            M1[j] = ((double)rand_r(seed1) / (RAND_MAX)) * (max - min) + min;
        }
        for (j = 0; j < N / 2; j++)
        {
            M2[j] = ((double)rand_r(seed2) / (RAND_MAX)) * (max * 10 - max) +
max;
        }

#pragma omp parallel default(none) shared(M1, M2, M2_copy, i, min, max, e, N,
sum_sin, min_nonzero)
        {
#pragma omp for
            for (j = 0; j < N; j++)
            {
                M1[j] = cbrt(M1[j] / e);
            }

#pragma omp for
            for (j = 0; j < N / 2; j++)
            {
                M2_copy[j] = M2[j];
            }

#pragma omp for
            for (j = 1; j < N / 2; j++)
            {
                if (j == 0)
                {
                    M2[0] = log(fabs(tan(M2[0])));
                }
                else
                {
```

```
                    M2[j] = log(fabs(tan(M2[j] + M2_copy[j - 1])));
                }
            }

#pragma omp for
            for (j = 0; j < N / 2; j++)
            {
                M2[j] = pow(M1[j], M2[j]);
            }

#pragma omp single
            {
                int j;
                for (j = 0; j < N / 2 - 1 && M2[j] == 0; j++)
                    ;
                min_nonzero = M2[j];
            }

#pragma omp for reduction(+ : sum_sin)
            for (j = 0; j < N / 2; j++)
            {
                if ((int)(M2[j] / min_nonzero) % 2 == 0)
                {
                    sum_sin += sin(M2[j]);
                }
            }
        }
    }
    gettimeofday(&T2, NULL);
    delta_ms = 1000 * (T2.tv_sec - T1.tv_sec) + (T2.tv_usec - T1.tv_usec) / 1000;
    printf("%ld\n", delta_ms);
    free(M1);
    free(M2);
    free(M2_copy);

    return 0;
}
```

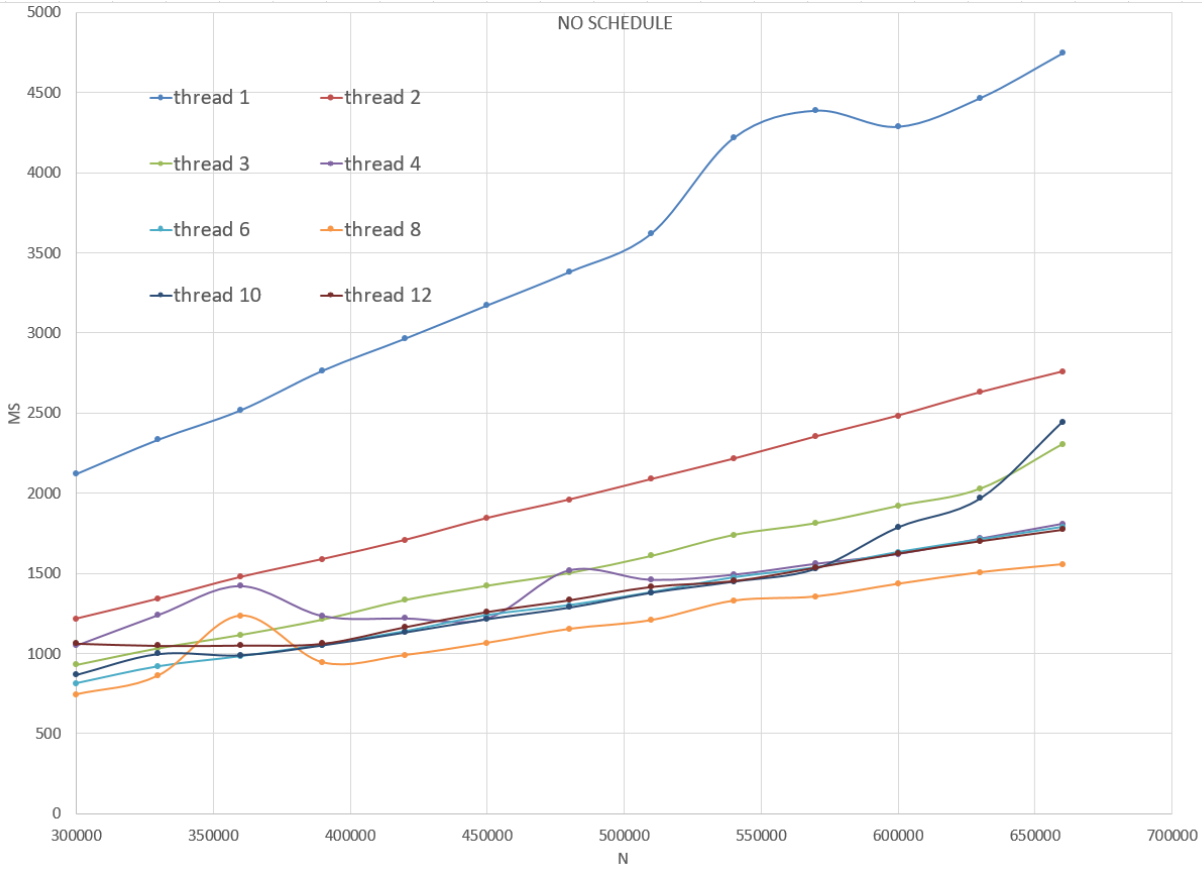
## Результаты экспериментов

Ниже приведены результаты ЛР2, компилятор gcc (AMD Framevawe)




| | MS | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| N | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 10 | 12 |
| 300000 | 2706 | 1608 | 1271 | 1095 | 1167 | 1091 | 1216 | 993 | 995 | 1085 |
| 330000 | 2921 | 2024 | 1400 | 1227 | 1287 | 1206 | 1140 | 1120 | 1069 | 1124 |
| 360000 | 3193 | 2241 | 1497 | 1316 | 1381 | 1272 | 1206 | 1175 | 1143 | 1181 |
| 390000 | 3706 | 2173 | 1617 | 1607 | 1491 | 1384 | 1293 | 1257 | 1220 | 1241 |
| 420000 | 3900 | 2254 | 1749 | 1895 | 1594 | 1471 | 1396 | 1325 | 1303 | 1354 |
| 450000 | 4089 | 2374 | 1864 | 1889 | 1694 | 1576 | 1468 | 1418 | 1405 | 1395 |
| 480000 | 4243 | 2534 | 1968 | 1832 | 1797 | 1660 | 1556 | 1499 | 1514 | 1490 |
| 510000 | 4514 | 2686 | 2105 | 1882 | 1917 | 1760 | 1640 | 1593 | 1595 | 1604 |
| 540000 | 4785 | 2854 | 2226 | 2030 | 2015 | 1851 | 1746 | 1658 | 1658 | 1649 |
| 570000 | 5082 | 3030 | 2331 | 2227 | 2111 | 1951 | 1833 | 1742 | 1745 | 1741 |
| 600000 | 5335 | 3193 | 2461 | 2124 | 2212 | 2037 | 1915 | 1845 | 1858 | 1839 |
| 630000 | 5592 | 3333 | 2619 | 2233 | 2326 | 2483 | 2006 | 1979 | 2655 | 1916 |
| 660000 | 5870 | 3501 | 2774 | 2305 | 2431 | 2961 | 2086 | 1992 | 2776 | 1994 |

Приводится сравнение с параллельной программой с помощью Open MP без явно указанного расписания:




| БЕЗ SCHEDULE | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ms | N | 1 | 2 | 3 | 4 | 6 | 8 | 10 | 12 |
| 500 | 300000 | 2120 | 1216 | 928 | 1048 | 811 | 744 | 867 | 1062 |
| 1000 | 330000 | 2335 | 1341 | 1032 | 1239 | 917 | 861 | 998 | 1048 |
| 1500 | 360000 | 2518 | 1476 | 1115 | 1420 | 980 | 1235 | 990 | 1050 |
| 2000 | 390000 | 2766 | 1588 | 1211 | 1234 | 1053 | 944 | 1052 | 1061 |
| 2500 | 420000 | 2965 | 1707 | 1333 | 1219 | 1136 | 989 | 1132 | 1164 |
| 3000 | 450000 | 3173 | 1844 | 1422 | 1218 | 1237 | 1065 | 1215 | 1259 |
| 3500 | 480000 | 3381 | 1959 | 1501 | 1518 | 1299 | 1152 | 1287 | 1332 |
| 4000 | 510000 | 3622 | 2087 | 1608 | 1459 | 1380 | 1207 | 1380 | 1416 |
| 4500 | 540000 | 4218 | 2214 | 1739 | 1491 | 1473 | 1328 | 1449 | 1452 |
| 5000 | 570000 | 4389 | 2353 | 1812 | 1559 | 1536 | 1354 | 1529 | 1536 |
| 5500 | 600000 | 4288 | 2483 | 1920 | 1620 | 1631 | 1433 | 1787 | 1624 |
| 6000 | 630000 | 4467 | 2629 | 2027 | 1716 | 1710 | 1505 | 1967 | 1701 |
| 6500 | 660000 | 4746 | 2758 | 2305 | 1808 | 1788 | 1555 | 2444 | 1772 |

Наблюдается ускорение времени работы программы в среднем на 500мс во всех рассмотренных случаях, наилучший результат при использовании OpenMP достигается при указании 8 потоков (процессор 4 ядерный). Дальнейшие эксперименты с разными расписаниями будут проводиться именно на 8 потоках.

## Сравнение schedule

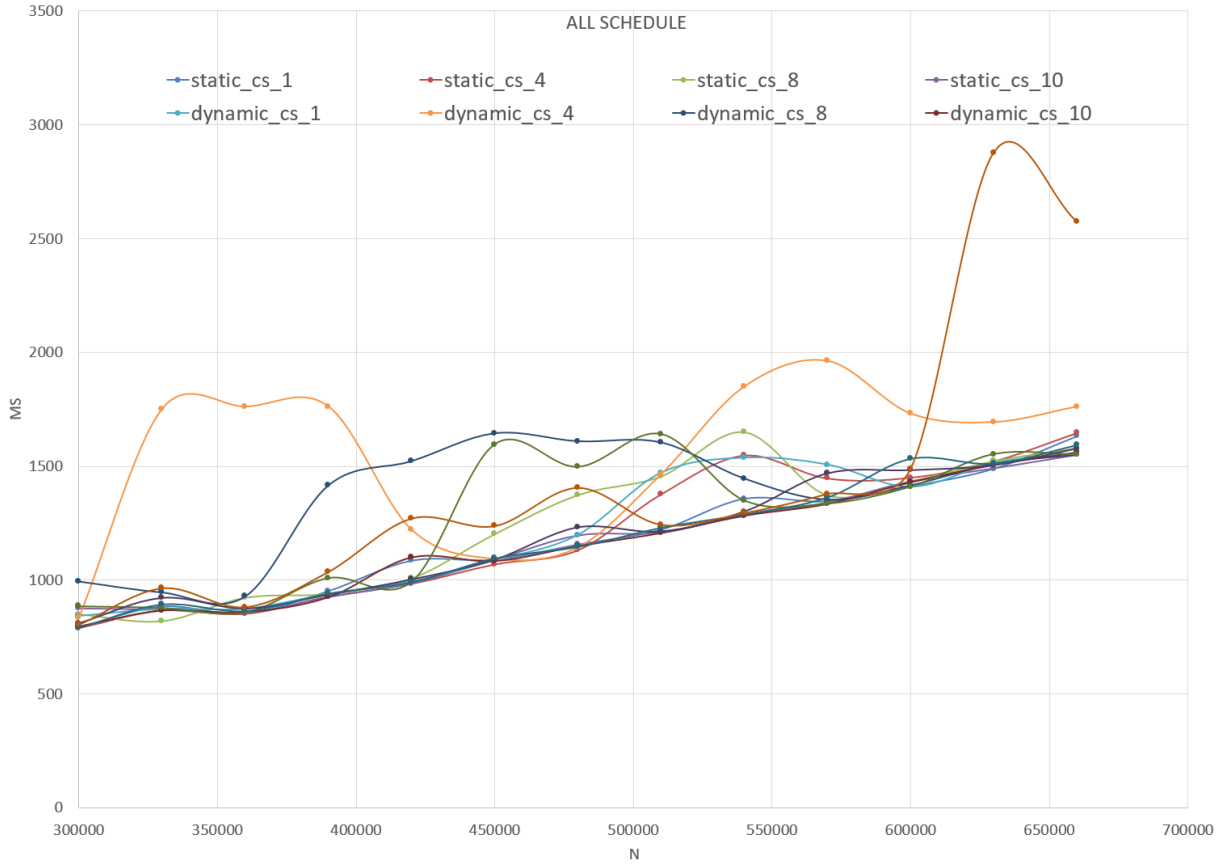
Тип расписания на машине по умолчанию определялся с помощью кода:

```c
#include <stdio.h>
#include <omp.h>

int main()
{
    int chunk_size, schedule_type;
    enum omp_sched_t kind;

#pragma omp parallel
    {
#pragma omp single
        {
            // Get the current schedule type and chunk size
            omp_get_schedule(&kind, &chunk_size);

            switch (kind)
            {
            case omp_sched_static:
                printf("Schedule type: static\n");
                break;
            case omp_sched_dynamic:
                printf("Schedule type: dynamic\n");
                break;
            case omp_sched_guided:
                printf("Schedule type: guided\n");
                break;
            case omp_sched_auto:
                printf("Schedule type: auto\n");
                break;
            default:
                printf("Unknown schedule type\n");
            }
            printf("Chunk size: %d\n", chunk_size);
        }
    }

    return 0;
}
```

Были получены следующие результаты:

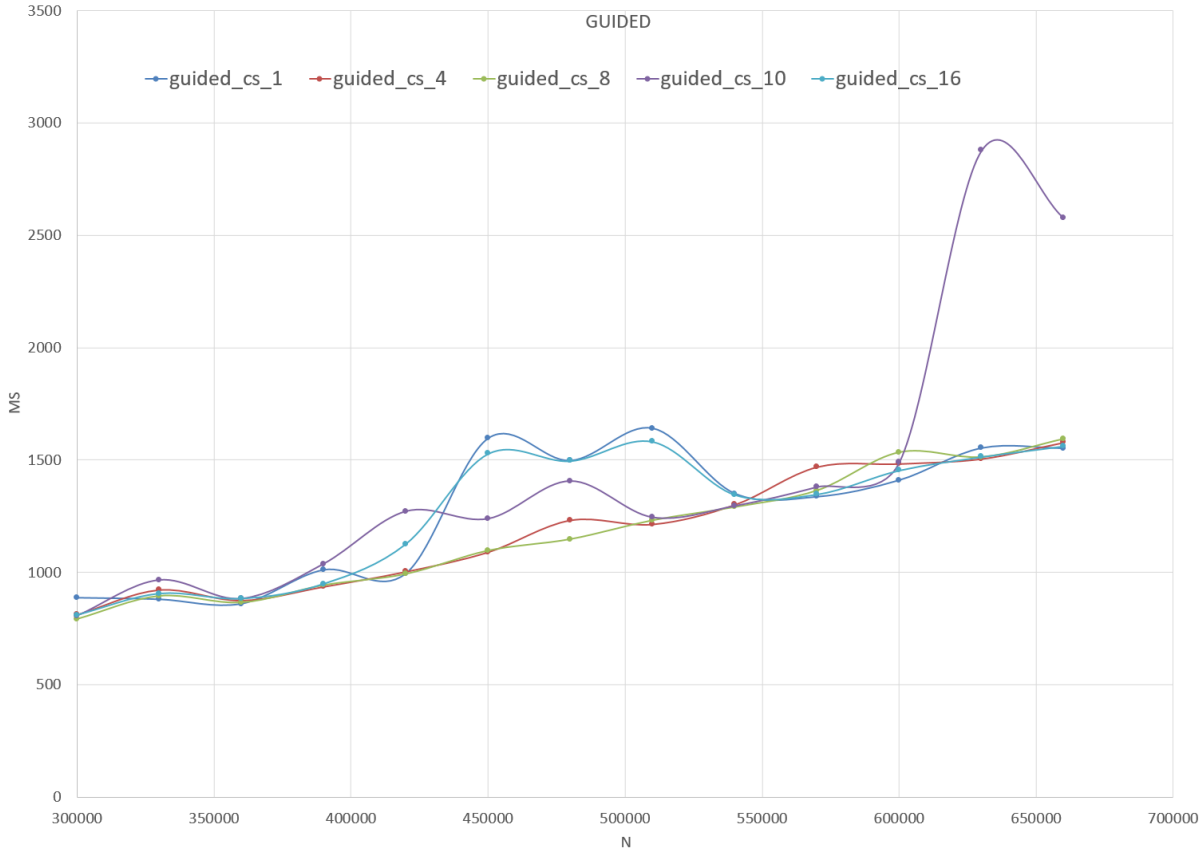
```
Schedule type: dynamic
Chunk size: 1
```

Сравнение static, guided, dynamic на одном графике:

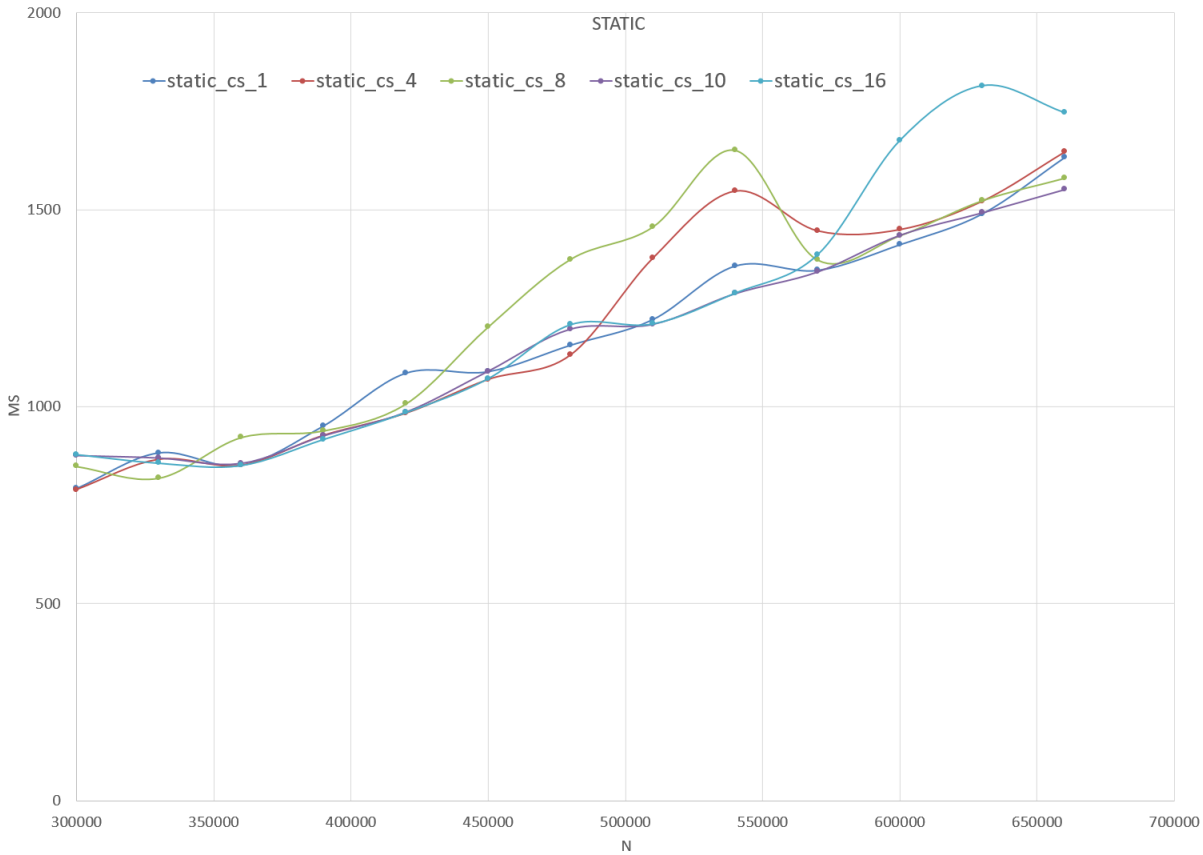



Согласно графику, наилучшие результаты показывает расписание guided с chunk size = 8

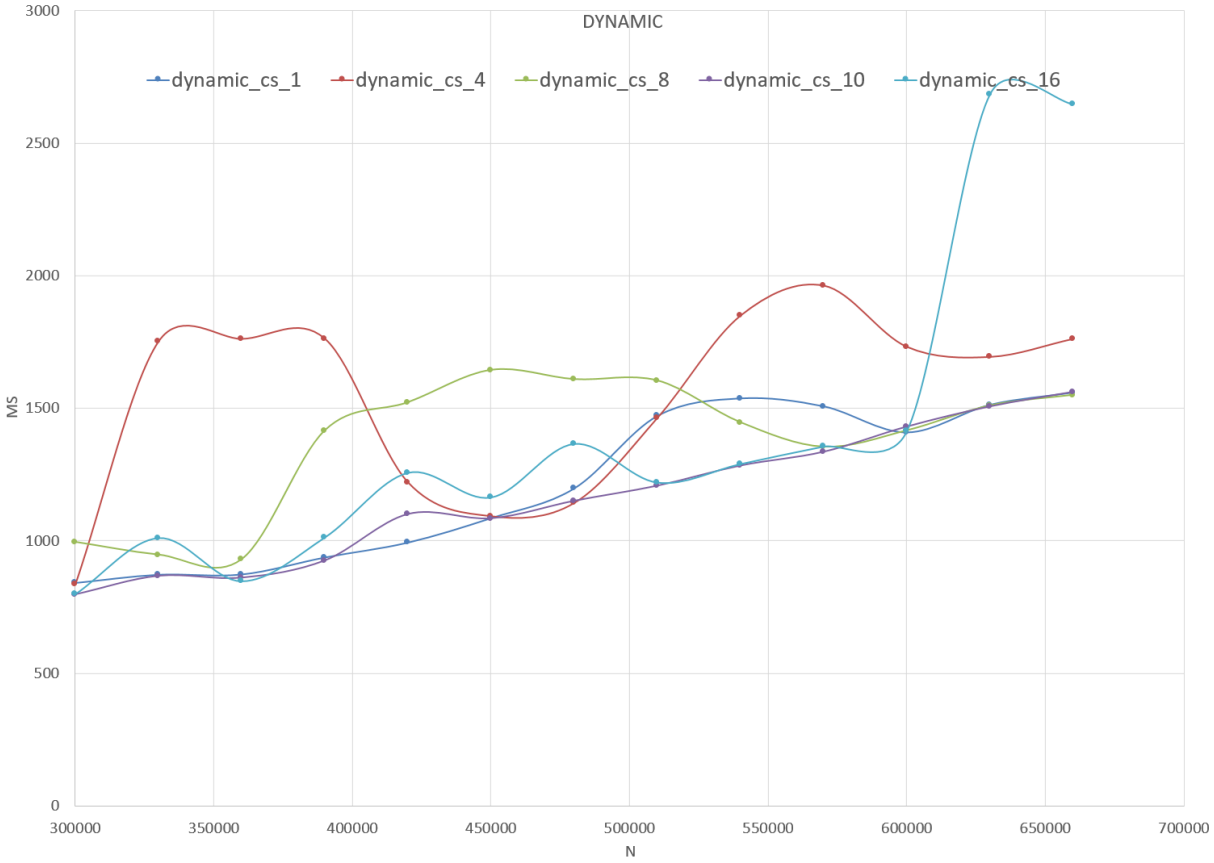
Сравнение guided с различными chunk size:



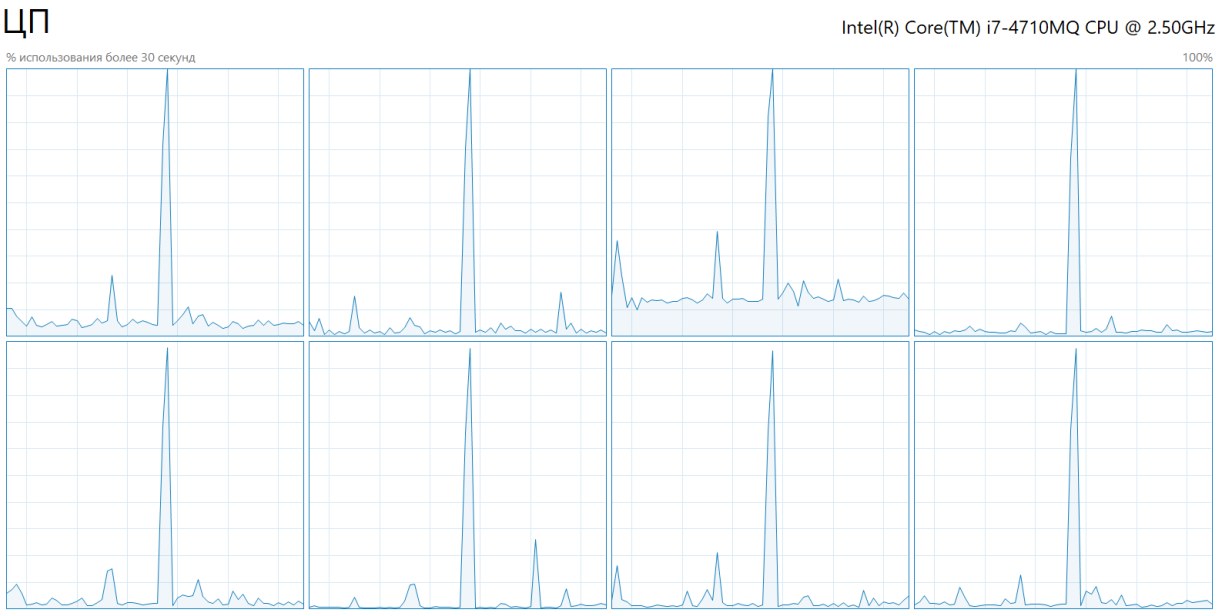

Сравнение static с различными chunk size:



Сравнение dynamic с различными chunk size:



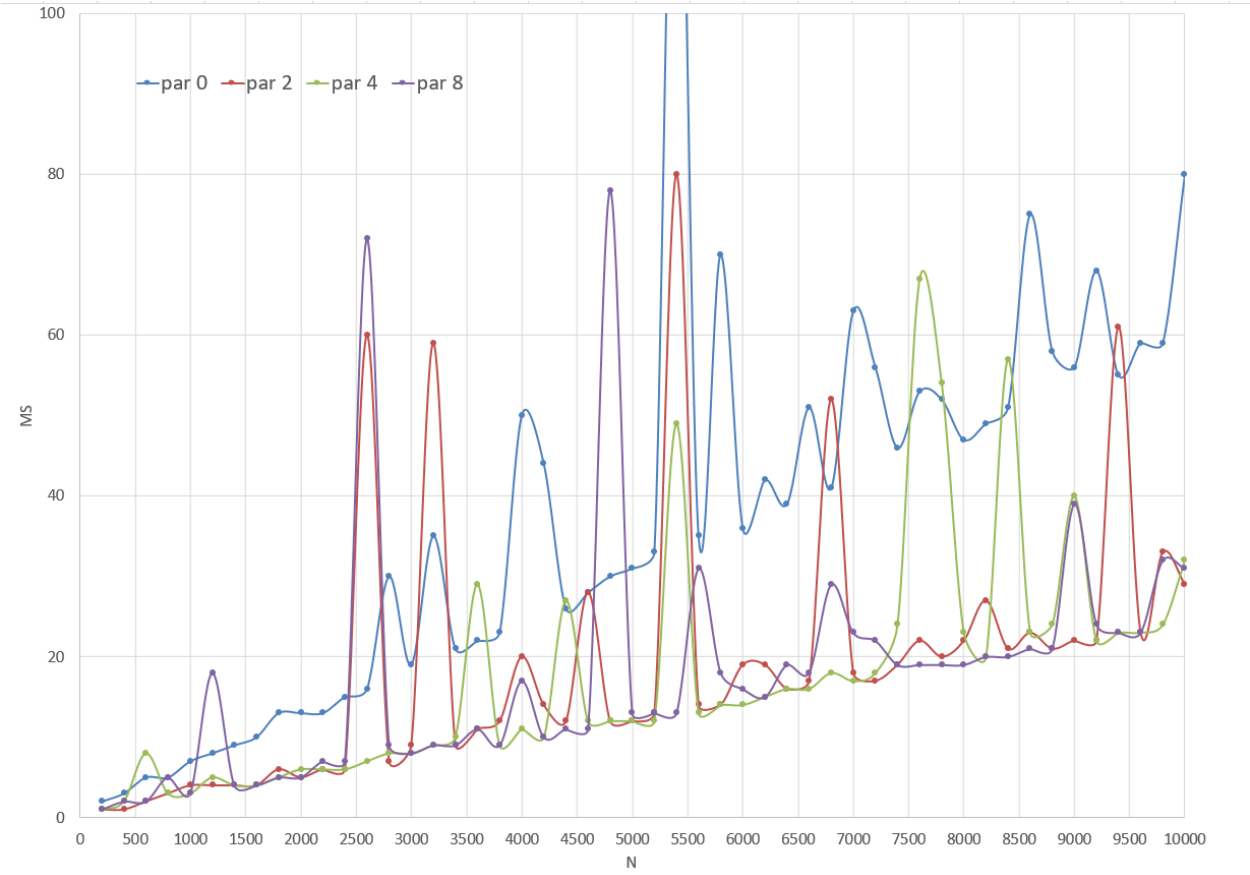

Скриншот диспетчера задач:



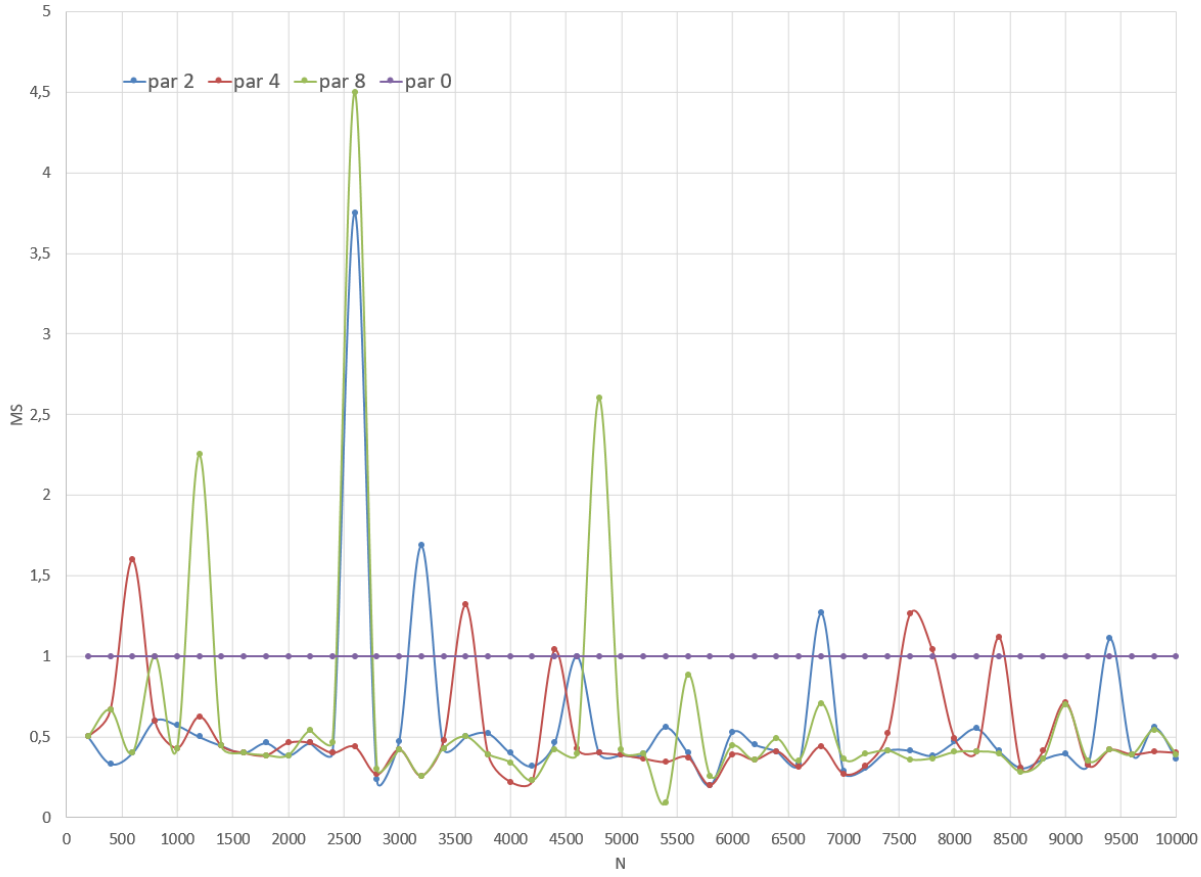
## Дополнительное задание 1

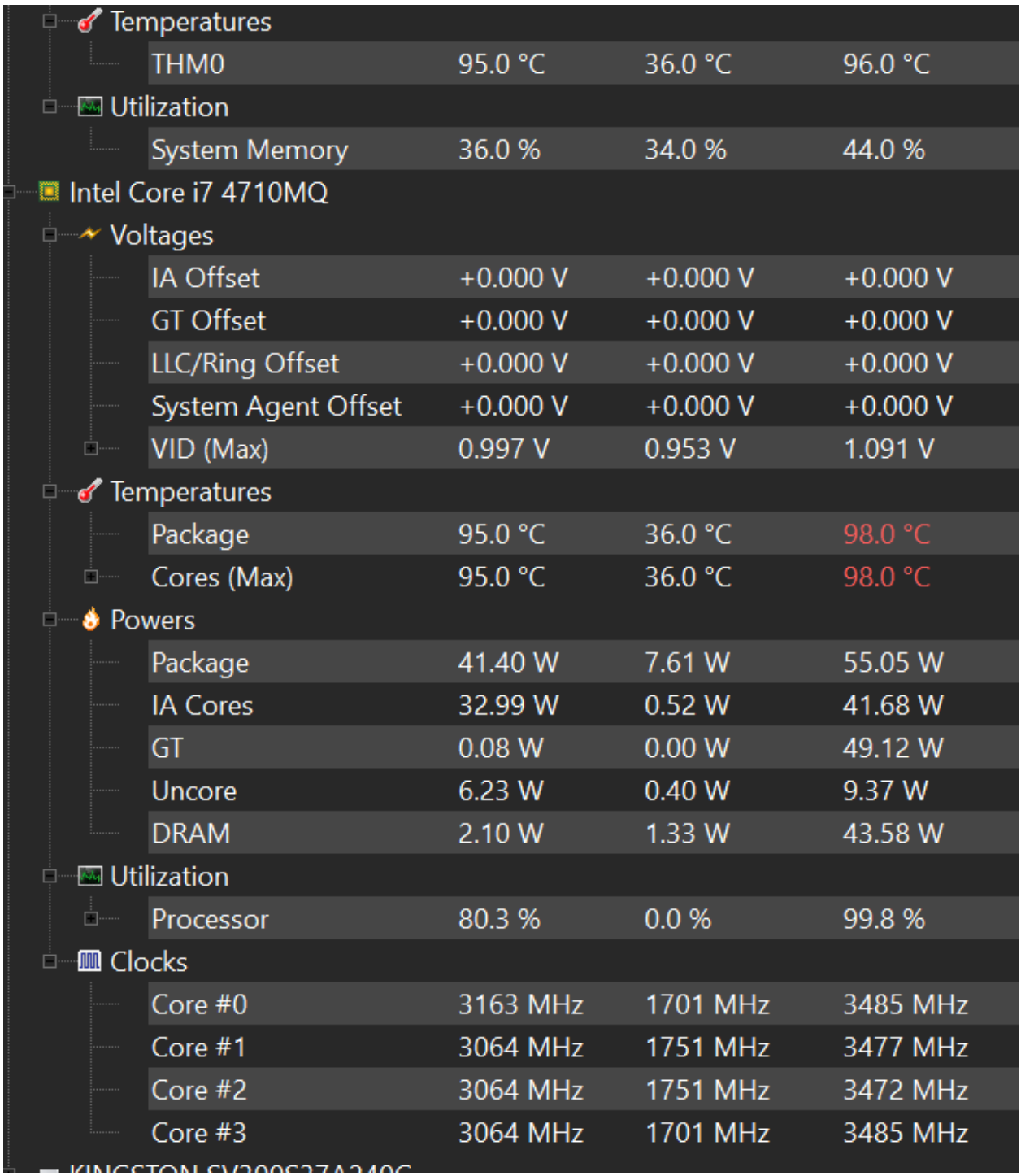
Приведены графики времени выполнения и параллельного ускорения для значений N от 200 до 10000 с шагом 200

Время работы:



Параллельное ускорение:






**Вывод**

По сравнению с автоматическим распараллеливанием и распараллеливанием с помощью AMD Framewave наблюдается прирост производительности.

Наилучшим показало себя расписание guided с размером chunk size равным количеству потоков 8. Количество потоков было определено при выполнении программы без явно заданного расписания.