

**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 Худяков А. А. Р4115  
(Фамилия И.О.) (номер группы)

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

Санкт-Петербург

2023 г.

## Содержание

1	Описание решаемой задачи .....	3
2	Характеристика используемого оборудования: .....	4
3	Текст программы.....	4
4	Эксперименты .....	8
	Заключение .....	18

# 1 Описание решаемой задачи

В исходном коде программы, полученной в результате выполнения лабораторной работы №1 нужно применить технологию распараллеливания OpenMP. Для этого нужно перед каждым циклом for вставить следующую директиву OpenMP:

```
"#pragma omp parallel for default(none) private(...) shared(...)"
```

Наличие параметра default(none) является обязательным.

Все циклы необходимо проверить на наличие зависимостей между итерациями и при их наличии использовать специальные директивы OpenMP. В ряде случаев стоит вообще отказаться от распараллеливания цикла, но это необходимо обосновать.

Вариант решаемой задачи:

Этап Map

Массив M1

Номер варианта	Операция
6	Кубический корень после деления на число e

Массив M2

Номер варианта	Операция
2	Модуль косинуса

Этап Merge

Номер варианта	Операция
4	Выбор большего (т.е. $M2[i] = \max(M1[i], M2[i])$ )

Этап Sort

Номер варианта	Операция
6	Сортировка вставками (Insertion sort)

## 2 Характеристика используемого оборудования:

Процессор AMD Ryzen 5 5600H with Radeon Graphics 3.3 GHz (AMD64 Family 25 Model 80 Stepping 0 AuthenticAMD ~3301 МГц)

Операционная система Ubuntu 22.04.2 LTS

Количество физических ядер: 6

Количество логических ядер: 12

Версия GCC: 11.3.0

Оперативная память 16 Гб

L2 Cache 3MB

L3 Cache 16MB

Вся работа проводилась на виртуальной машине VMware Workstation Player 16, ресурсы, выделенные виртуальной машине:

Количество процессоров: 6

Оперативная память 6 Гб

## 3 Текст программы

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

// #define SCHEDULE dynamic
// #define CHUNK 10

double max_el(double* restrict a, double* restrict b)
{
    return (*a) > (*b) ? (*a) : (*b);
}

void insert_sort(double* M, int n)
{

```

```

int key = 0;
double temp = 0.0;
for (int k = 0; k < n - 1; k++)
{
    key = k + 1;
    temp = M[key];
    for (int j = k + 1; j > 0; j--)
    {
        if (temp < M[j - 1])
        {
            M[j] = M[j - 1];
            key = j - 1;
        }
    }
    M[key] = temp;
}
}

int main(int argc, char* argv[])
{
    int i, N;
    struct timeval T1, T2;
    long delta_ms;
    double Abeg = 1.0;
    double A = 315.0;
    double Aend = A * 10.0;
    unsigned int seed;
    N = atoi(argv[1]);          /* N равен первому параметру
командной строки */
    gettimeofday(&T1, NULL); /* запомнить текущее время T1 */
    int N_2 = N / 2;
    double* restrict M1 = malloc(N * sizeof(double));
    double* restrict M2 = malloc(N_2 * sizeof(double));
    double* restrict M2_old = malloc(N_2 * sizeof(double));

    const int num_threads = atoi(argv[2]); /* amount of threads */
    #if defined(_OPENMP)
    omp_set_dynamic(0);
    omp_set_num_threads(num_threads);
    #endif

    double expon = exp(1);
    unsigned int seed1[num_threads];
    unsigned int seed2[num_threads];

    for (i = 0; i < 100; i++) /* 100 экспериментов */
    {
        double X = 0.0;
        seed = i;
        #pragma omp parallel default(none) shared(N, N_2, M1, M2,
M2_old, X, expon, seed1, seed2, A, Abeg, Aend)

```

```

{

#ifdef _OPENMP
    int tid = omp_get_thread_num();
    seed1[tid] = rand();
    seed2[tid] = rand();

    // #pragma omp for schedule(SCHEDULE, CHUNK)
#pragma omp for
    for (int j = 0; j < N; j++) {
        int tid = omp_get_thread_num();
        unsigned int local_seed = seed1[tid] + j;
        M1[j] = ((double)rand_r(&local_seed) / (RAND_MAX))
* (A - Abeg) + Abeg;
    }
    // #pragma omp for schedule(SCHEDULE, CHUNK)
#pragma omp for
    for (int k = 0; k < N_2; ++k) {
        int tid = omp_get_thread_num();
        unsigned int local_seed1 = seed2[tid] + k;
        M2[k] = ((double)rand_r(&local_seed1) /
(RAND_MAX)) * (Aend - A) + A;
    }
#else
    /* Заполнить массив исходных данных размером N */
    // GENERATE
    for (int j = 0; j < N; j++)
    {
        M1[j] = ((double)rand_r(seedp) / (RAND_MAX)) * (A
- Abeg) + Abeg;
    }
    /* Заполнить массив исходных данных размером N/2 */
    for (int k = 0; k < N_2; k++)
    {
        M2[k] = ((double)rand_r(seedp) / (RAND_MAX)) *
(Aend - A) + A;
    }
#endif

    // MAP
#ifdef CHUNK && defined(SCHEDULE)
#pragma omp for schedule(SCHEDULE, CHUNK)
#else
#pragma omp for
#endif
    for (int j = 0; j < N; j++)
    {
        M1[j] = cbrt(M1[j] / expon);
    }
}

```

```

#if defined(CHUNK) && defined(SCHEDULE)
#pragma omp for schedule(SCHEDULE, CHUNK)
#else
#pragma omp for
#endif
    for (int k = 0; k < N_2; k++)
    {
        M2_old[k] = M2[k];
    }
#pragma omp single
    M2[0] = fabs(cos(M2[0]));
#if defined(CHUNK) && defined(SCHEDULE)
#pragma omp for schedule(SCHEDULE, CHUNK)
#else
#pragma omp for
#endif
    for (int k = 1; k < N_2; k++)
    {
        M2[k] = M2[k] + M2_old[k - 1];
    }
#if defined(CHUNK) && defined(SCHEDULE)
#pragma omp for schedule(SCHEDULE, CHUNK)
#else
#pragma omp for
#endif
    for (int k = 1; k < N_2; ++k) {
        M2[k] = fabs(cos(M2[k]));
    }

    // MERGE
#if defined(CHUNK) && defined(SCHEDULE)
#pragma omp for schedule(SCHEDULE, CHUNK)
#else
#pragma omp for
#endif
    for (int k = 0; k < N_2; k++)
    {
        M2[k] = max_el(&M1[k], &M2[k]);
    }

    /* Отсортировать массив с результатами указанным
методом */
    // SORT
    // insert_sort(&M2, N_2);

    // REDUCE

    int k = 0;
    while (M2[k] == 0 && k < N_2 - 1) k++;
    double minelem = M2[k];

```

```

        // sum of matching array elements
#ifdef CHUNK && defined(SCHEDULE)
#pragma omp for schedule(SCHEDULE, CHUNK)
#else
#pragma omp for
#endif
    for (int k = 0; k < N_2; k++)
    {
        M2_old[k] = 0.0;
        if ((int)(M2[k] / minelem) % 2 == 0)
        {
            M2_old[k] = sin(M2[k]);
        }
    }
#ifdef CHUNK && defined(SCHEDULE)
#pragma omp for reduction(+ : X) schedule(SCHEDULE, CHUNK)
#else
#pragma omp for reduction(+ : X)
#endif
    for (int j = 0; j < N_2; ++j)
    {
        X += M2_old[j];
    }
    // printf("X = %f ", X);
    // printf("%f", X);
    // printf("\n\n");
}
gettimeofday(&T2, NULL); /* запомнить текущее время T2 */
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); /*
T2 - T1 */
return 0;
}

```

## 4 Эксперименты

1. Сделаем сравнение с результатами распараллеливания из ЛР1 и ЛР2.

Рассмотрим графики параллельного ускорения для каждой из работ. Результаты ЛР1 и автоматического распараллеливания с помощью компилятора GCC:



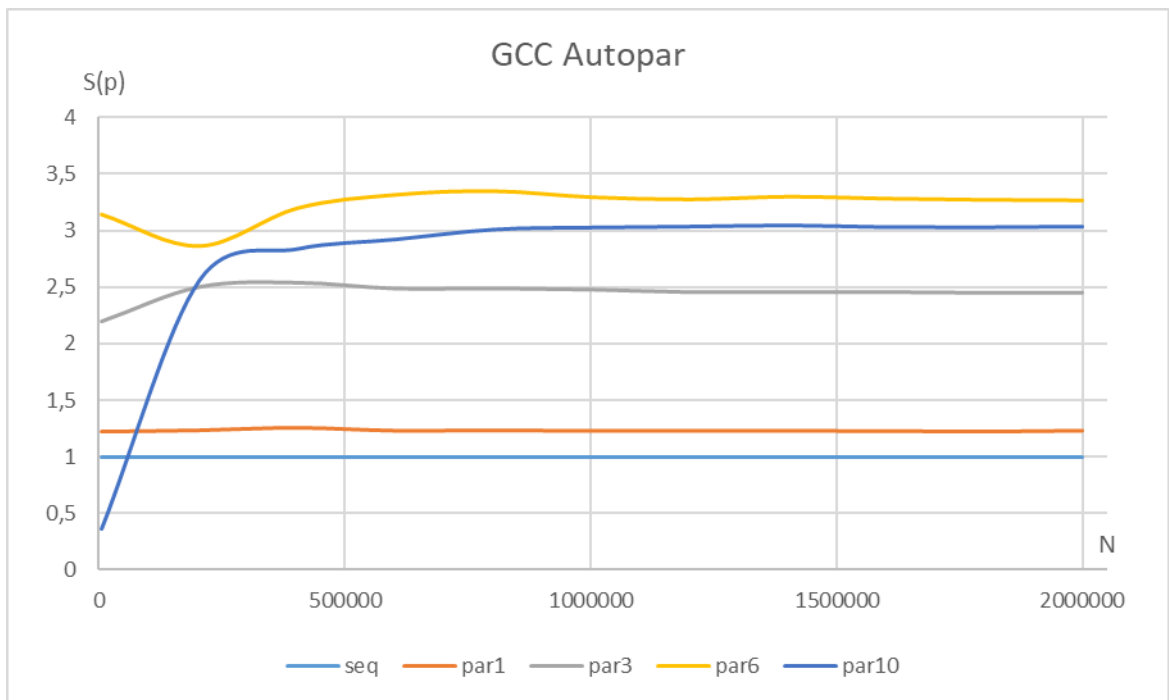


Рисунок 1 - График параллельного ускорения GCC Autopar

Далее представим график параллельного ускорения для результатов ЛР2, где использовалась библиотека AMD FrameWave:

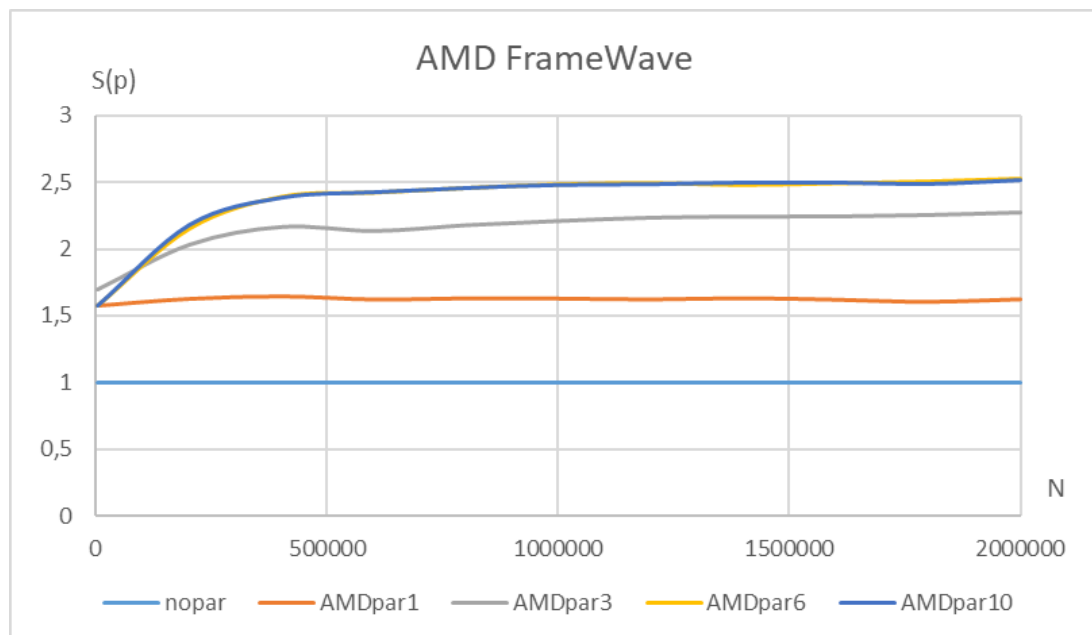


Рисунок 2 - График параллельного ускорения AMD FrameWave

После чего приведем графики параллельного ускорения для ЛР3, где используется библиотека OpenMP. Для повышения эффективности распараллеливания распараллелим также этап генерации массивов:

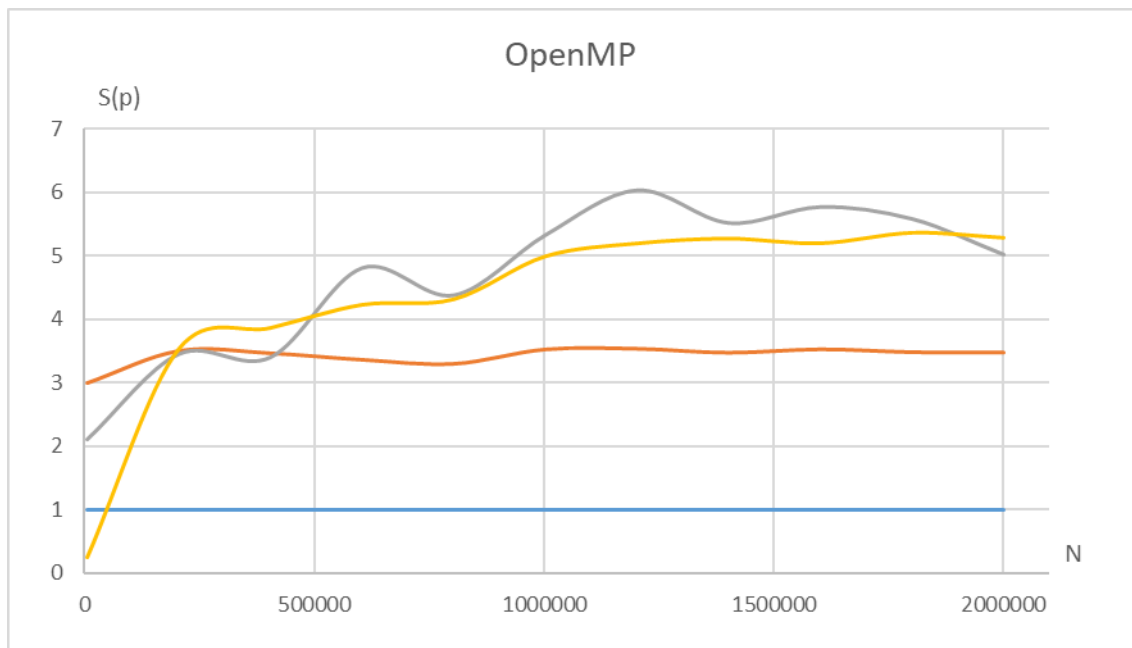


Рисунок 3 - График параллельного ускорения OpenMP

По результатам эксперимента можно заметить, что наиболее эффективные результаты показывает автоматическое распараллеливание от компилятора GCC, затем с близкими значениями параллельного ускорения показывает себя библиотека OpenMP.

## 2. Рассмотрим влияние параметра schedule и различных chunk size.

В данном случае будем рассматривать случай наилучшего распараллеливания, при  $p=6$ .

Представим графики параллельных ускорений.

Общий график

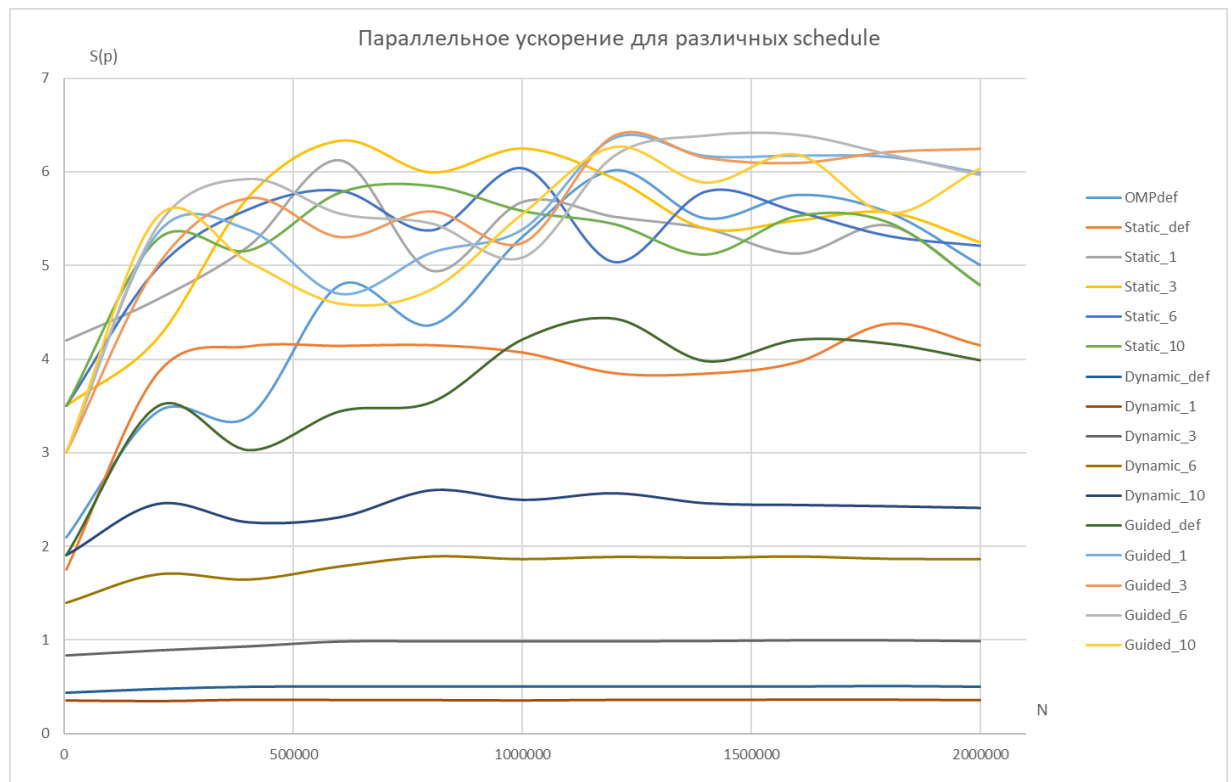


График для schedule=static

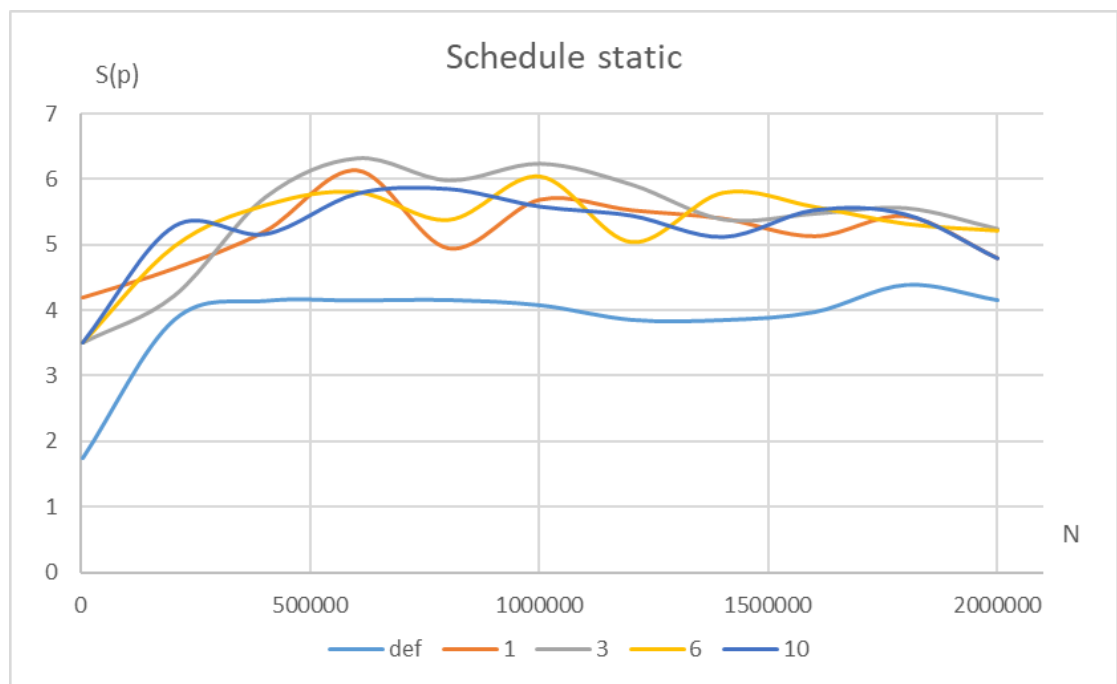


График для schedule = dynamic

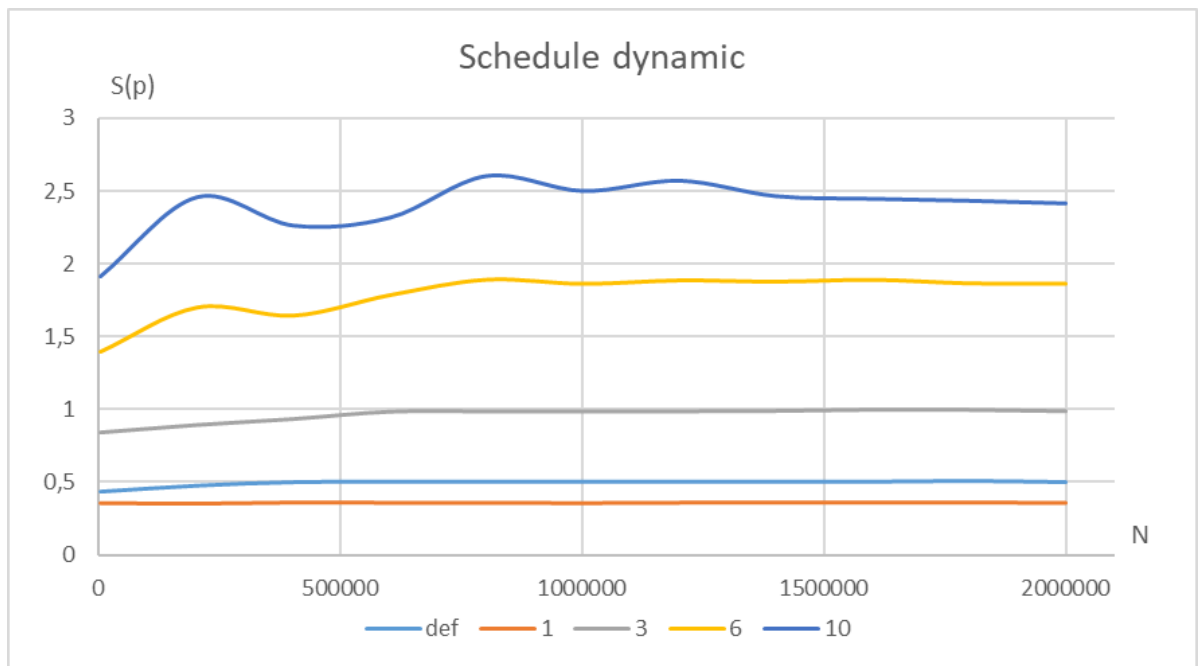
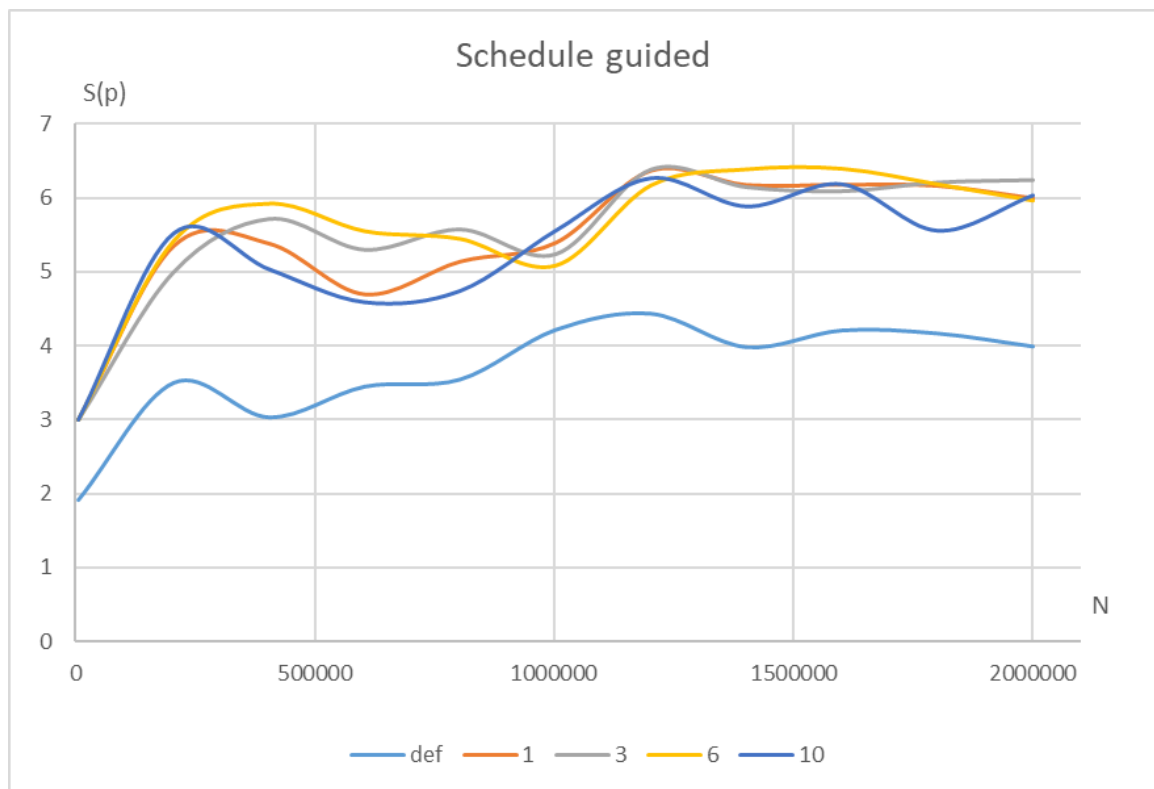


График для schedule = guided



По анализу графиков можно сделать вывод что наилучшим с точки зрения параллельного ускорения является расписания guided с chunk size = 6.

3. Определим тип расписания на машине при использовании schedule = default.

Для этого необходимо воспользоваться командой `omp_get_schedule`. В результате применения этой команды было выяснено, что расписание по умолчанию – `static` с параметром `chunk size 1`.

4. Наилучший вариант при различных  $N$ .

$M$  (количество потоков) – 6

Расписание (`schedule`) – `guided` (с параметром `chunk size 6`)

При запуске все вычислители (процессоры) равноценны и размеры массивов не меняются. Для других значений размерности массивов могут быть актуальны другие значения параметров.

5. Вычислительная сложность алгоритма до и после распараллеливания.

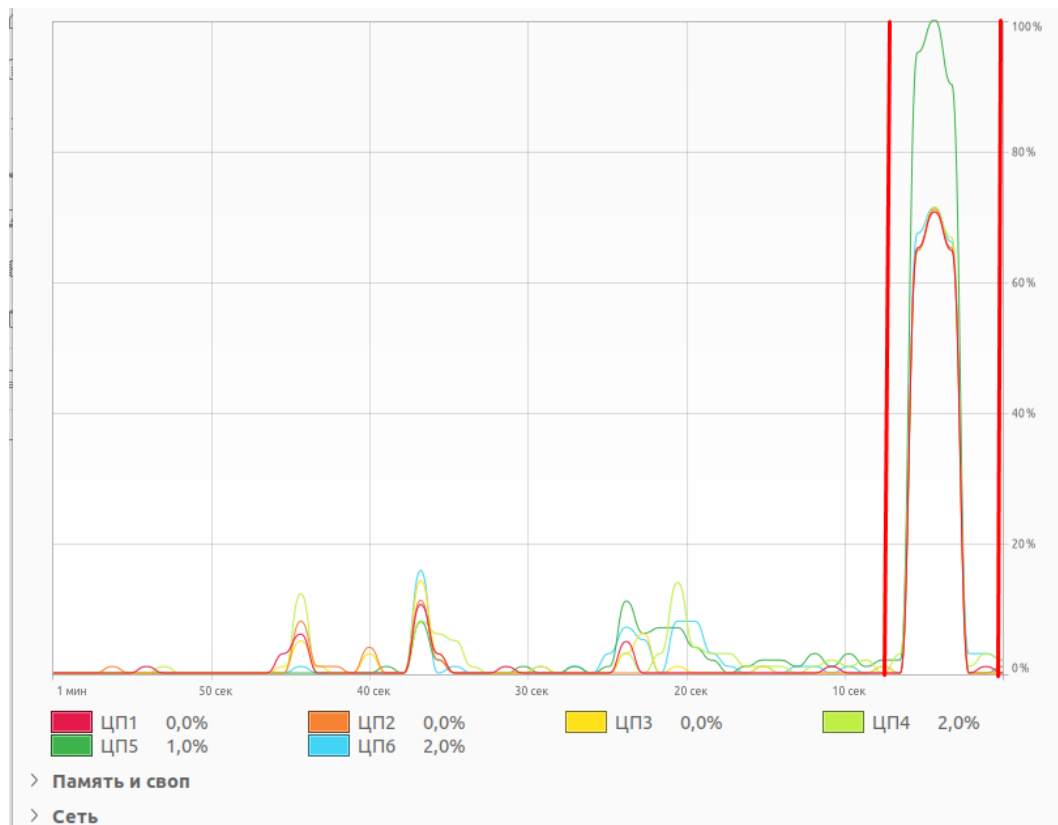
Без распараллеливания  $C1 * N$

С распараллеливанием  $C1 * N / M + C2 * N$ , где  $M$  – количество потоков.

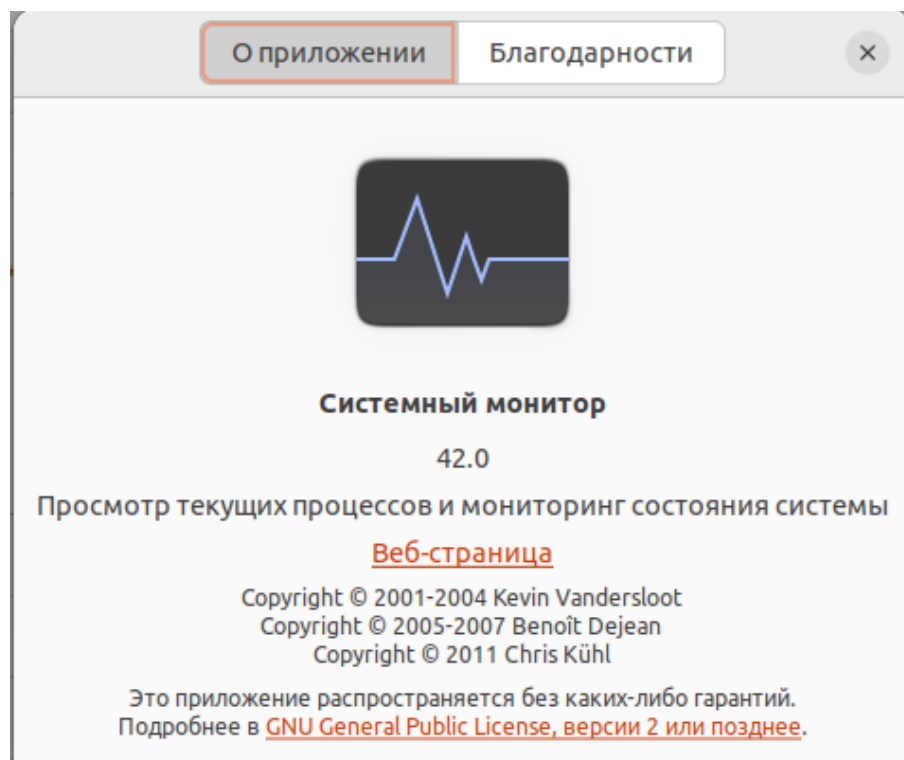
Все циклы проходят по элементам массивов без вложений и часть программы не параллельна (заполнение)

6. Проиллюстрируем, что программа действительно распараллелилась.

Представим фрагмент диспетчера задач:

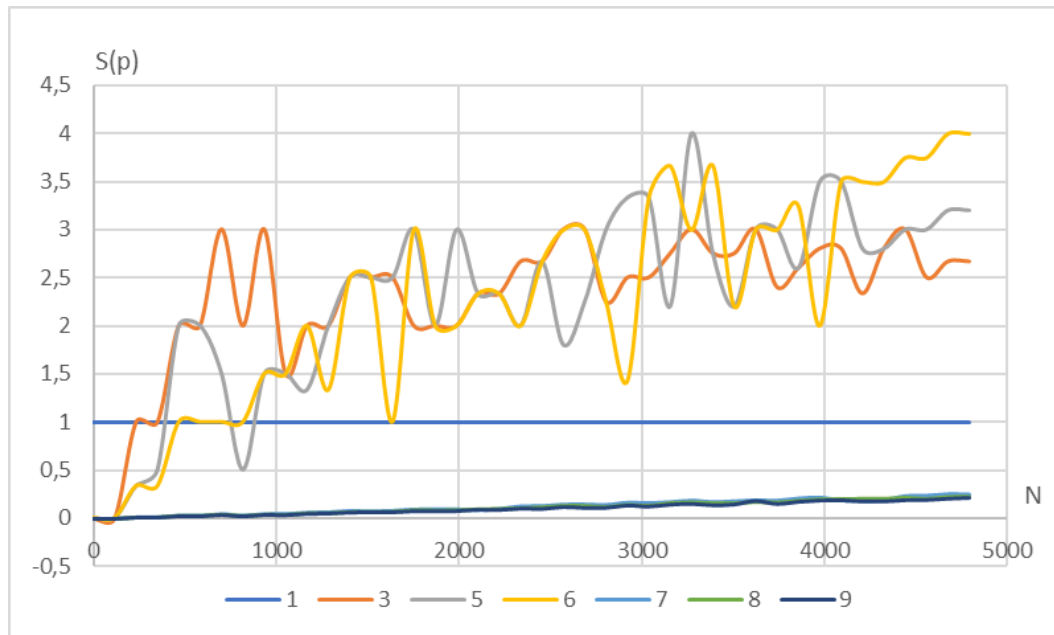


В качестве диспетчера использовался стандартный системный монитор Ubuntu:



7. Попробуем найти значения  $N$ , при которых накладные расходы на распараллеливание превышают выигрыш от распараллеливания

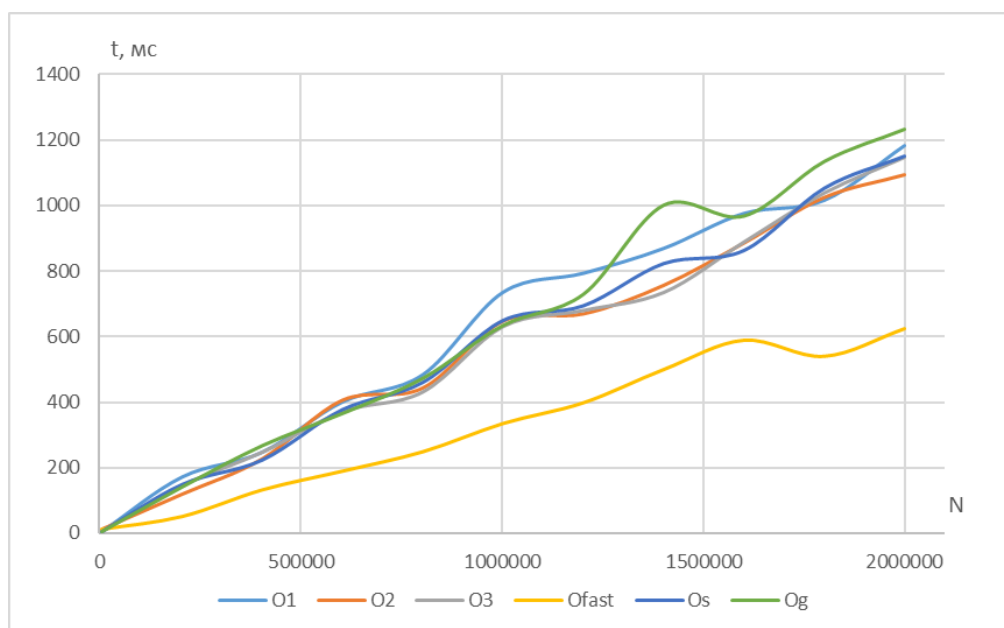
Приведем график параллельного ускорения



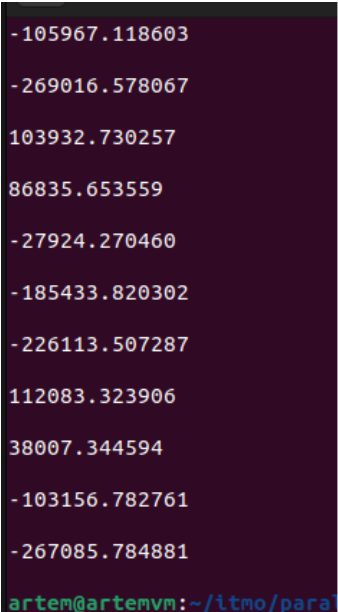
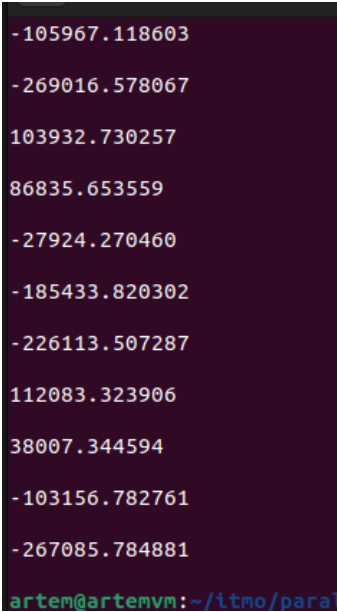
Исходя из отображенных на графике результатов можно сделать вывод, что распараллеливание показывает себя не эффективно на размерах массива меньше 500 элементов. Большая часть ресурсов тратится на накладные расходы, и не распараллеленная программа показывает себя в этом случае лучше.

8. Рассмотрим влияние флагов оптимизации на время выполнения программы

Построим график времени выполнения при различных оптимизаторах:



В данном случае из программы было исключено распараллеливание части, связанной с заполнением массивов M1 и M2 псевдослучайными числами. Проведем верификацию результатов, чтобы убедиться, что оптимизация не внесла искажений в результаты. Представим фрагменты результатов расчета для каждой из оптимизаций:

Без оптимизации	Оптимизация O1
	
Оптимизация O2	Оптимизация O3



<pre> -105967.118603 -269016.578067 103932.730257 86835.653559 -27924.270460 -185433.820302 -226113.507287 112083.323906 38007.344594 -103156.782761 -267085.784881 artem@artemvm:~/itmo/para7 </pre>	<pre> -105967.118603 -269016.578067 103932.730257 86835.653559 -27924.270460 -185433.820302 -226113.507287 112083.323906 38007.344594 -103156.782761 -267085.784881 artem@artemvm:~/itmo/para7 </pre>
<p>Оптимизация Ofast</p>	<p>Оптимизация Og</p>
<pre> -105967.118603 -269016.578067 103932.730257 86835.653559 -27924.270460 -185433.820302 -226113.507287 112083.323906 38007.344594 -103156.782761 -267085.784881 artem@artemvm:~/itmo/para7 </pre>	<pre> -105967.118603 -269016.578067 103932.730257 86835.653559 -27924.270460 -185433.820302 -226113.507287 112083.323906 38007.344594 -103156.782761 -267085.784881 artem@artemvm:~/itmo/para7 </pre>

## Заключение

Для получения максимального быстродействия была дополнительно распараллелена часть программы, которая занимается заполнением массивов псевдослучайными числами. В самом лучшем случае удалось достичь параллельного ускорения примерно равного 6, что совпадает с количеством потоков, выделенных виртуальной машине.

При сравнении различных параметров расписание наибольший прирост наблюдается со значением `guided`. Кроме того, применение расписания `dynamic` привело к существенному увеличению времени выполнения программы.

Для достаточно малых размеров массивов, не превышающих 500 элементов было выяснено, что распараллеливание неэффективно и накладные расходы нивелируют все преимущества распараллеливания, последовательная программа выполняется быстрее.

Были исследованы различные оптимизаторы и их влияние на время выполнения. Для оценки результатов расчетов было убрано распараллеливание генерации псевдослучайных чисел. Наилучшие результаты по распараллеливанию показал флаг оптимизации `Ofast`. Искажений результатов выявлено не было.