МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

“Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики”

(НИУ ИТМО)

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.04.04 Программная инженерия

Лабораторная работа №

“Распараллеливание циклов с помощью технологии OpenMP”

По дисциплине “Параллельные вычисления”

Студент группы P4114

Шитов Григорий

Семенович

Преподаватель:

Жданов Андрей Дмитриевич

Санкт-Петербург, 2024 г.

Оглавление

1. Описание решаемой задачи

1. Добавить во все for-циклы (кроме цикла в функции main, указывающего количество экспериментов) в программе из ЛР №1 директиву OpenMP:

2. Проверить все for-циклы на внутренние зависимости по данным между итерациями.

3. Убедиться, что получившаяся программа обладает свойством прямой совместимости с компиляторами, не поддерживающими OpenMP

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

5. Провести эксперименты, замеряя параллельное ускорение

6. Провести эксперименты, добавив параметр «schedule» и варьируя в экспериментах тип расписания. Исследование нужно провести для всех возможных расписаний: static, dynamic, guided. Следующей «степенью свободы», которую необходимо использовать, является chunck_size, которому необходимо задать четыре различных варианта: единице, меньше чем число потоков, равному числу потоков и больше чем число потоков. Привести сравнение параллельного ускорения при различных расписаниях с результатами п. 5.

7. Определить, какой тип расписания на вычислительной машине при использовании «schedule default».

8. Выбрать из рассмотренных в п. 5 и п. 6 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.

2. Краткая характеристика системы

Операционная система: Ubuntu 22.04

Процессор: AMD® Ryzen 3 2200u with radeon vega mobile gfx × 4

Оперативная память: 8ГБ

Количество физических ядер: 2

Количество логических ядер: 4

gcc version 11.4.0 (Ubuntu 11.4.0-1ubuntu1~22.04)

3. Программа lab2.c

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

#define M1SIZE (N)
#define M2SIZE (N / 2)

void gnomeSort(double *array, int size);

int main(int argc, char *argv[])
{
    int i, N, j;
    unsigned int seed;
    double sum = 0;
    struct timeval T1, T2;
    long delta_ms;
    N = atoi(argv[1]); // N равен первому параметру командной строки
    int threadCount = atoi(argv[2]);
    double *restrict M1 = (double *)calloc(M1SIZE, sizeof(double));
    double *restrict M2 = (double *)calloc(M2SIZE, sizeof(double));
    double *restrict M2temp = (double *)calloc(M2SIZE, sizeof(double));
    FILE *resultOfTest = fopen("Results/Results_without_shedule.txt", "a");
    gettimeofday(&T1, NULL); // запомнить текущее время T1
    for (i = 0; i < 100; i++)
    { // 100 экспериментов
        seed = i;
        srand(seed); // инициализировать начальное значение ГСЧ
                     // Заполнить массив исходных данных размером N
        for (j = 0; j < M1SIZE; j++)
        {
            M1[j] = 1 + rand_r(&seed) % 360;
        }
        for (j = 0; j < M2SIZE; j++)
        {
            M2[j] = 360 + rand_r(&seed) % (10 * 360 - 361);
        }
// Решить поставленную задачу, заполнить массив с результатами
#pragma omp parallel for default(none) private(j) shared(M1, N)
num_threads(threadCount)
        for (j = 0; j < M1SIZE; j++)
        {
            M1[j] = 1.0 / tanh(sqrt(M1[j]));
        }

// copy m2 array
```

```
#pragma omp parallel for default(none) private(j) shared(M2, M2temp, N)
num_threads(threadCount)
        for (j = 0; j < M2SIZE; j++)
        {
            M2temp[j] = M2[j];
        }

// new value in M2mak
#pragma omp parallel for default(none) private(j) shared(M2, M2temp, N)
num_threads(threadCount)
        for (j = 1; j < M2SIZE; j++)
        {
            M2[j] += M2temp[j - 1];
            M2[j] = sqrt(M2[j] * M_E);
        }

// Merge
#pragma omp parallel for default(none) private(j) shared(M2, M1, N)
num_threads(threadCount)
        for (j = 0; j < M2SIZE; j++)
        {
            M2[j] = M1[j] / M2[j];
        }

        // Отсортировать массив с результатами указанным методомared(M1, N)
        // gnomeSort(M2, M2SIZE);

        // REDUCE
        double min = 10 * 360 + 1;
#pragma omp parallel for default(none) reduction(min : min) private(j) shared(M2,
N) num_threads(threadCount)
        for (j = 0; j < M2SIZE; j++)
        {
            if (M2[j] < min)
            {
                min = M2[j];
            }
        }

#pragma omp parallel for default(none) reduction(+ : sum) private(j) shared(M2,
min, N) num_threads(threadCount)
        for (j = 0; j < M2SIZE; j++)
        {
            if ((int)(M2[j] / min) % 2 == 0)
            {
                sum += sin(M2[j]);
            }
        }
    }
    free(M1);
```

```
    free(M2);
    free(M2temp);
    printf("X: %lf\n", sum);
    gettimeofday(&T2, NULL); // запомнить текущее время T2
    delta_ms = (T2.tv_sec - T1.tv_sec) * 1000 +
               (T2.tv_usec - T1.tv_usec) / 1000;
    printf("\nN=%d. Milliseconds passed: %ld\n", N, delta_ms);
    fprintf(resultOfTest, "%ld\n", delta_ms);
    fclose(resultOfTest);
    return 0;
}

void gnomeSort(double *array, int size)
{
    int index = 0;

    while (index < size)
    {
        if (index == 0)
            index++;
        if (array[index] >= array[index - 1])
            index++;
        else
        {
            int temp = array[index];
            array[index] = array[index - 1];
            array[index - 1] = temp;
            index--;
        }
    }
}
```

Листинг 1- Код программы lab2.c

4. Результаты

4.1. OMP без параметра schedule

	N	K=1	K=2	K=4	K=6
1	1500	13	12	76	60
2	136350	886	606	613	527
3	271200	1693	1284	1089	1105
4	406050	2531	1716	1658	1467
5	540900	3336	2499	2071	1922
6	675750	4147	2809	2549	2368
7	810600	5077	3474	3237	2928
8	645450	5805	4111	3563	3311
9	1080300	6688	4500	3973	3750
10	1215150	7467	5165	4501	4241
11	1350000	8377	5705	4958	4668

Таблица 1 - Результаты без параметра schedule

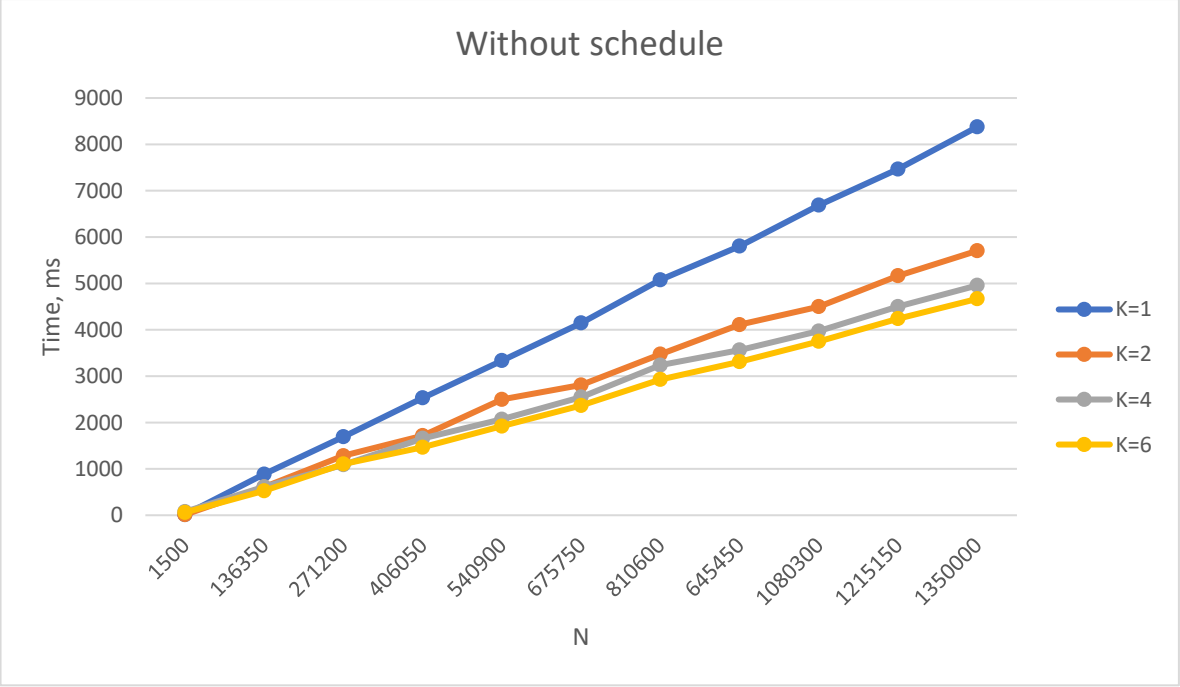

Рисунок 1- Результаты без параметра schedule

Результаты заметно улучшаются, при увеличении потоков 6, далее ускорение не насколько значительно.

4.2 Сравнение различных вариантов schedule

Исследование различных вариаций расписания проводилось с наилучшим результатом количества потоков (4) на аналогичных значениях параметра N (размерности массива)

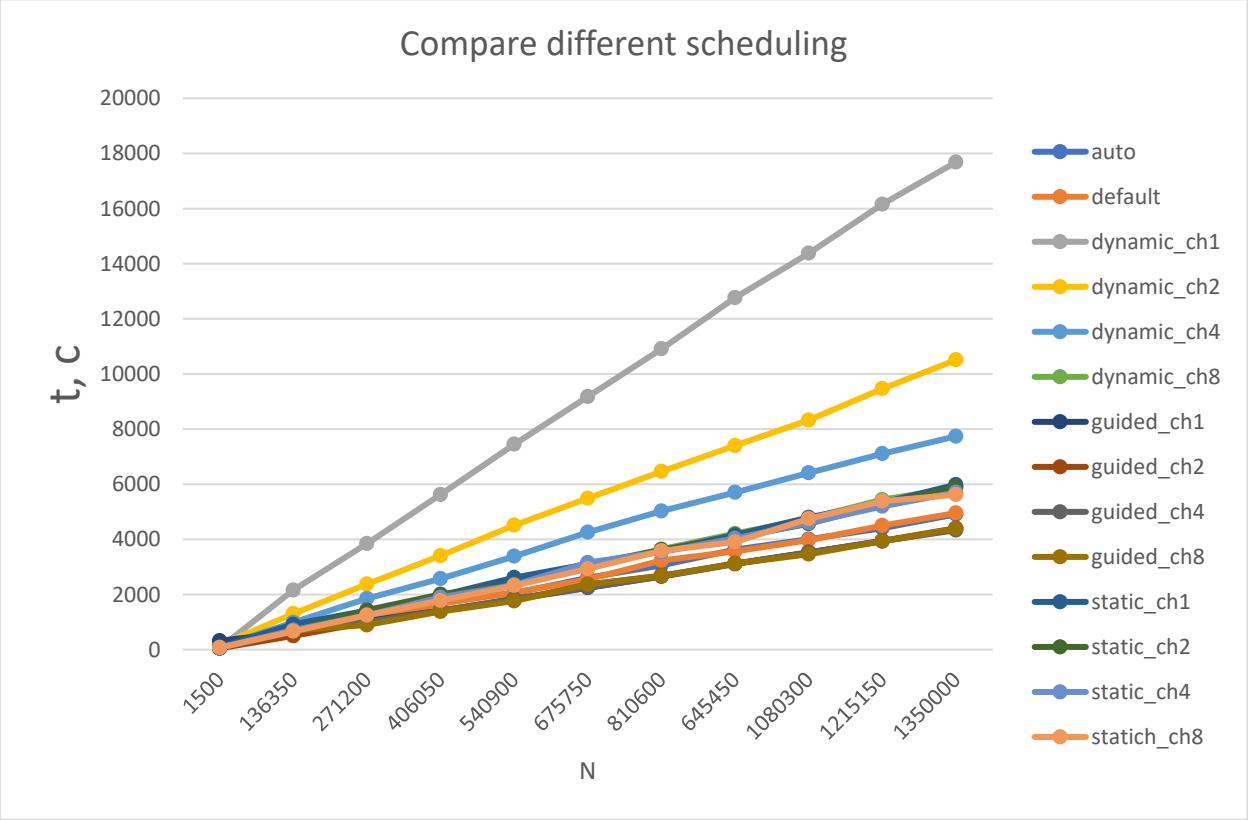

Рисунок 2 – Сравнение различные вариантов schedule

Как видно из Рис. 2 самым медленным способом расписания является dynamic со значением chunksize=1. В свою очередь, лучший результат показывает расписание guided со значениями chunksize = 4 или chunksize = 8 (разница незначительна, объясняется это тем, что на моей машине максимум 4 потока)

Далее приводятся таблицы и графики для всех видом расписания и chunksize.

4.3 Сравнение static с различными chunk_size

	N	CS=1	CS=2	CS=4	CS=8
1	1500	99	73	73	73
2	136350	915	690	707	664
3	271200	1414	1434	1245	1249
4	406050	1961	2006	1896	1782
5	540900	2614	2377	2346	2324
6	675750	3105	3003	3158	2930
7	810600	3594	3626	3545	3584
8	645450	4158	4063	4035	3906
9	1080300	4795	4568	4601	4750
10	1215150	5331	5277	5198	5385
11	1350000	5984	5893	5705	5623

Таблица 2 - Сравнение static с различными chunk_size

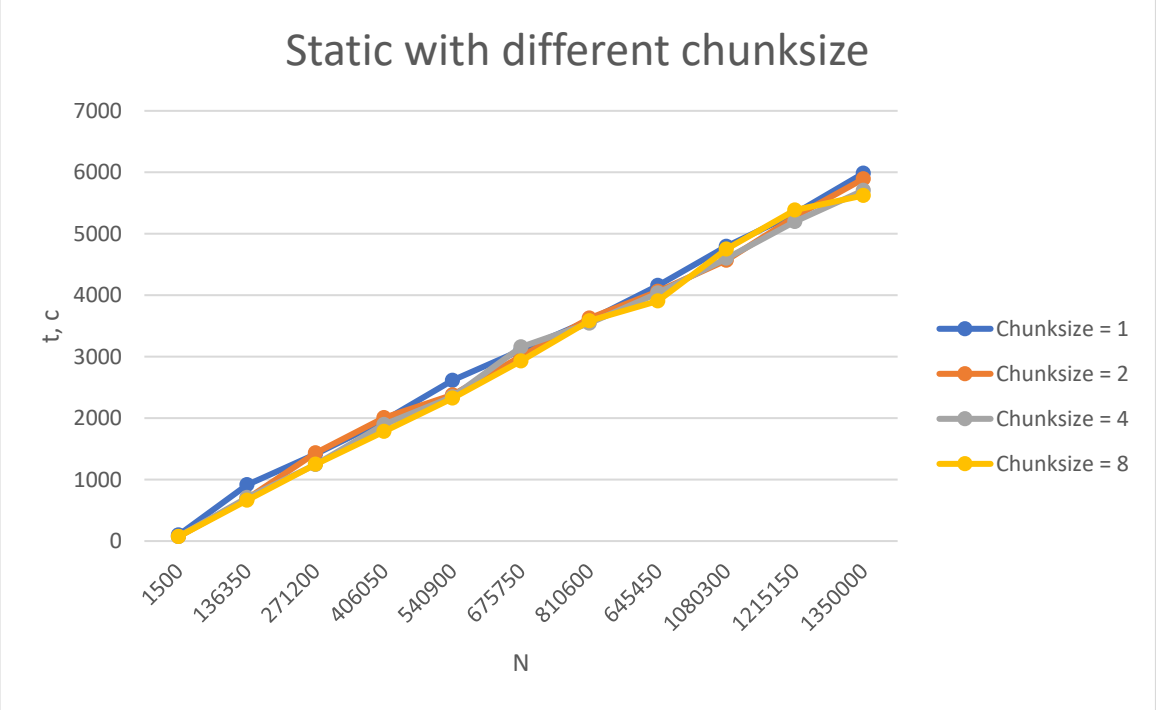

Рисунок 3 – Сравнение static с различными chunk_size

4.4 Сравнение dynamic с различными chunk_size

	N	CS=1	CS=2	CS=4	CS=8
1	1500	61	166	111	44
2	136350	2159	1301	985	687
3	271200	3845	2376	1855	1391
4	406050	5626	3408	2575	1846
5	540900	7451	4513	3388	2452
6	675750	9180	5491	4258	3069
7	810600	10912	6465	5026	3640
8	645450	12768	7401	5703	4208
9	1080300	14378	8325	6411	4753
10	1215150	16158	9465	7108	5439
11	1350000	17687	10511	7739	5866

Таблица 3 - Сравнение dynamic с различными chunk_size

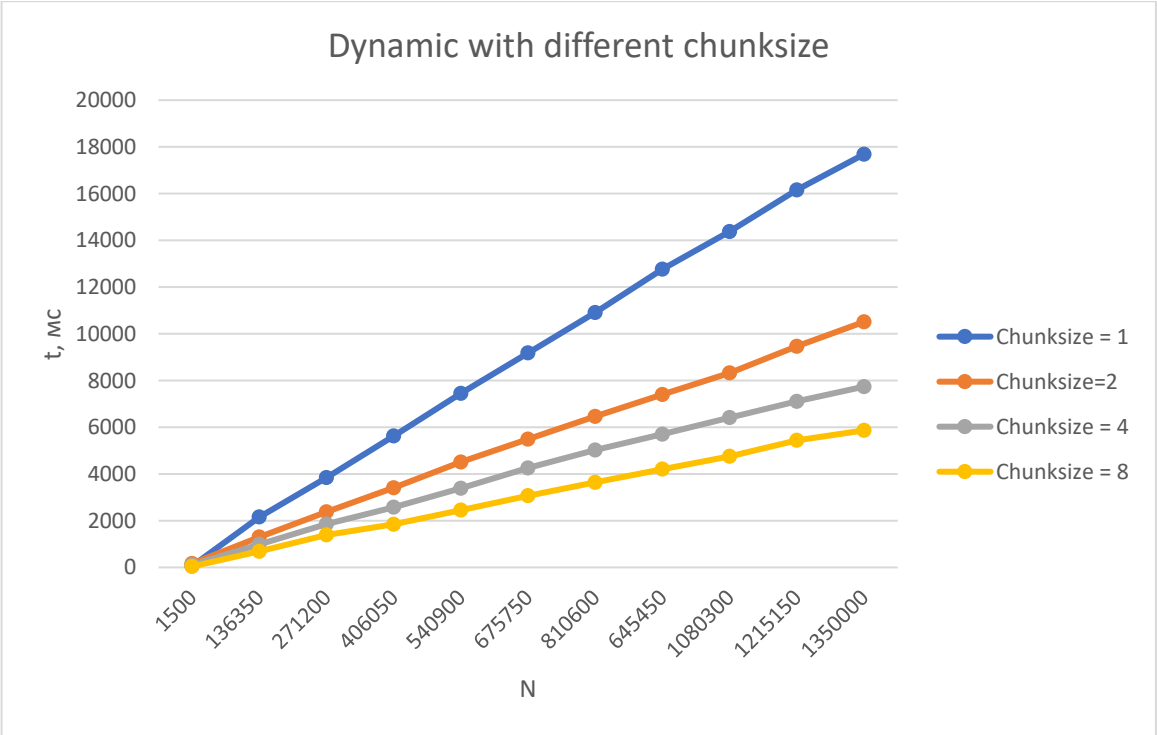

Рисунок 4 – Сравнение dynamic с различными chunk_size

4.5 Сравнение guided с различными chunk_size

	N	CS=1	CS=2	CS=4	CS=8
1	1500	323	49	48	73
2	136350	632	497	668	702
3	271200	1038	986	991	893
4	406050	1418	1406	1413	1381
5	540900	1823	1797	1852	1765
6	675750	2243	2361	2253	2365
7	810600	2661	2685	2648	2674
8	645450	3123	3120	3104	3123
9	1080300	3522	3521	3531	3461
10	1215150	3932	3948	3940	3937
11	1350000	4374	4404	4333	4395

Таблица 4 - Сравнение guided с различными chunk_size

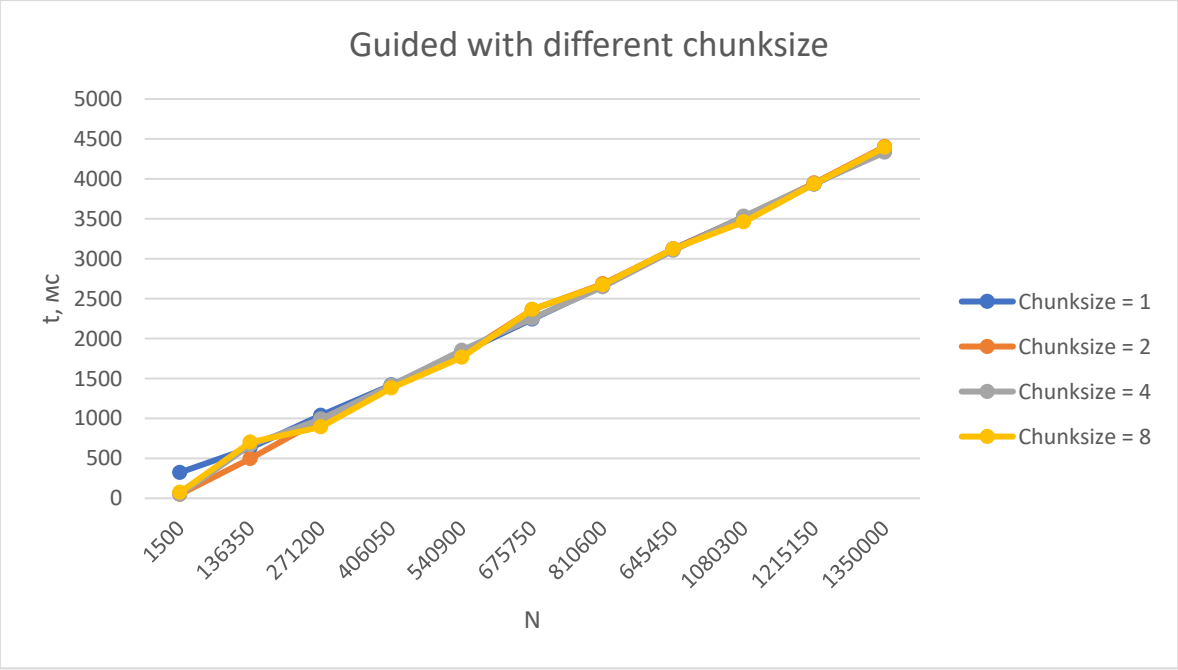

Рисунок 5 – Сравнение guided с различными chunk_size

4.6 Наилучший результат:

Количество потоков – 4

Расписание (schedule) – guided (с параметром chunk size 4 или 8)

Сложность:

1. Заполнение массивов M1 и M2: O(N)

2. Параллельный цикл для вычисления новых значений M1 и M2: O(N) (так как каждый элемент обрабатывается независимо)

3. Параллельный цикл для нахождения минимального ненулевого значения в M2: O(N/2)

4. Параллельный цикл для вычисления суммы sin значений из M2: O(N/2)

5. График параллельного ускорения для N < 10000

	N	Without OMP	K=1	K=2	K=4	K=6
1	100	1	1	18	20	33
2	1090	8	8	8	20	36
3	2080	16	15	11	84	44
4	3070	21	22	19	85	46
5	4060	29	28	40	55	45
6	5050	34	37	42	49	54
7	6040	42	43	31	55	55
8	7030	49	46	33	49	58
9	8020	53	52	36	106	61
10	9010	74	61	42	66	63
11	10000	144	79	46	69	65

Таблица 5 – Сравнение результатов с и без распараллеливания

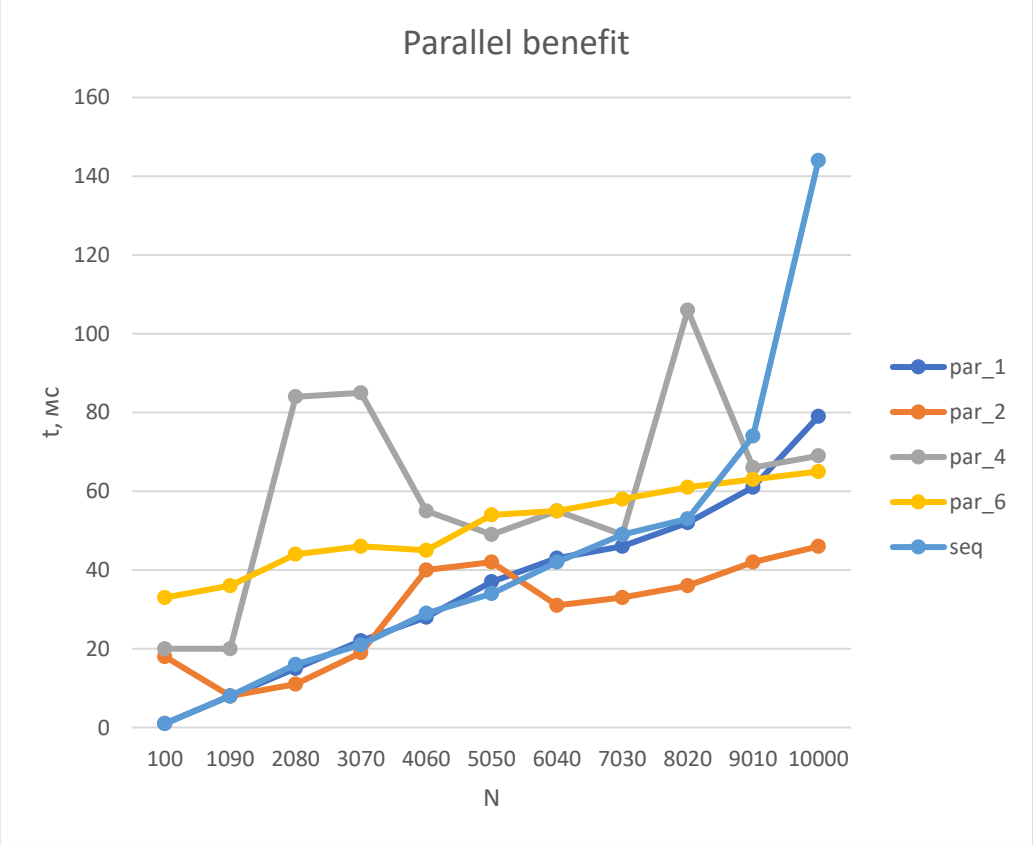

Рисунок 6 – Сравнение результатов с и без распараллеливания

До значения N=9010, большинство результатов с распараллеливанием медленнее чем без распараллеливания. Исходя из этого, можно сказать что накладные расходы на создание потоков сильно влияют на эксперименты со значением N<9010.

6. Применение различных флагов оптимизации

Далее приводятся таблицы и графики результатов, полученных при компиляции с различными флагами

6.1 Применение флага –O3

	N	auto	d_1	d_2	d_4	d_8	g_1	g_2	g_4	g_8	st_1	st_2	st_4	st_8	def
1	1500	123	61	166	111	44	323	49	48	73	99	73	73	73	76
2	136350	746	2159	1301	985	687	632	497	668	702	915	690	707	664	613
3	271200	1218	3845	2376	1855	1391	1038	986	991	893	1414	1434	1245	1249	1089
4	406050	1788	5626	3408	2575	1846	1418	1406	1413	1381	1961	2006	1896	1782	1658
5	540900	2047	7451	4513	3388	2452	1823	1797	1852	1765	2614	2377	2346	2324	2071
6	675750	2608	9180	5491	4258	3069	2243	2361	2253	2365	3105	3003	3158	2930	2549
7	810600	3061	10912	6465	5026	3640	2661	2685	2648	2674	3594	3626	3545	3584	3237
8	645450	3631	12768	7401	5703	4208	3123	3120	3104	3123	4158	4063	4035	3906	3563
9	1080300	4004	14378	8325	6411	4753	3522	3521	3531	3461	4795	4568	4601	4750	3973
10	1215150	4405	16158	9465	7108	5439	3932	3948	3940	3937	5331	5277	5198	5385	4501
11	1350000	4910	17687	10511	7739	5866	4374	4404	4333	4395	5984	5893	5705	5623	4958

Таблица 6 – Результаты выполнения программы при компиляции с флагом O3

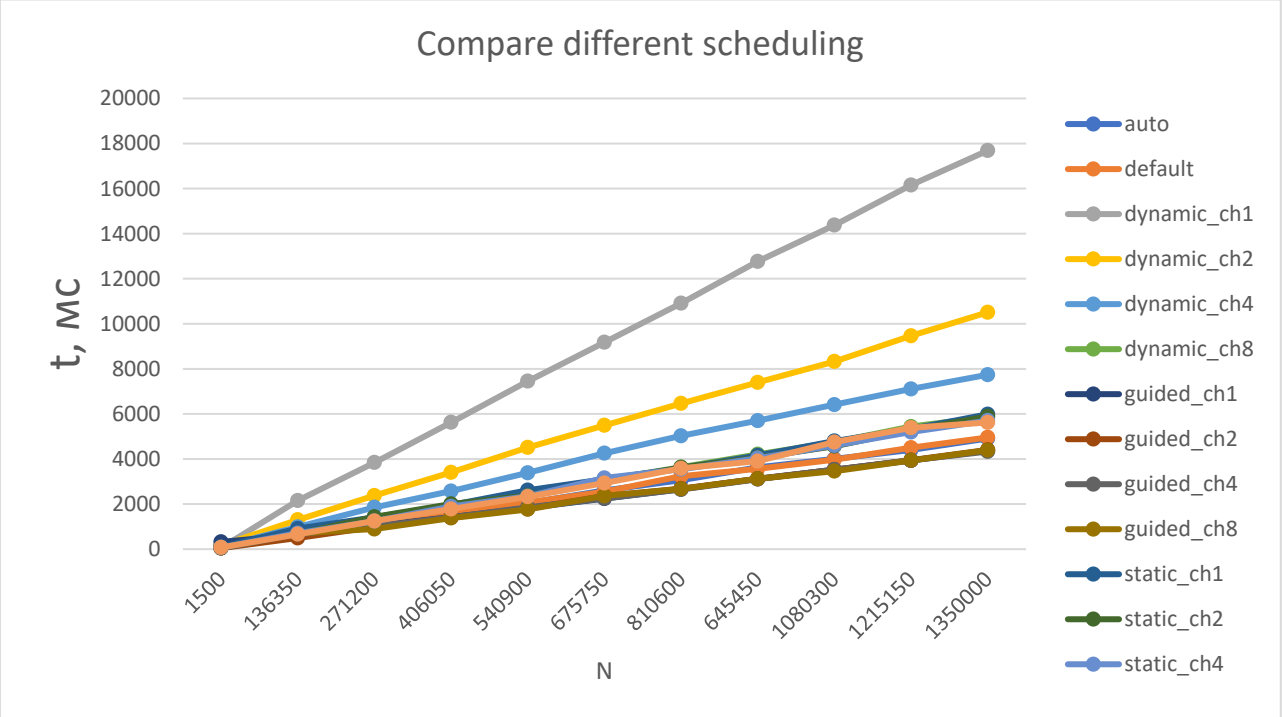

Рисунок 7 – Результаты выполнения программы при компиляции с флагом O3

6.2 Применение флага –O2

	N	auto	d_1	d_2	d_4	d_8	g_1	g_2	g_4	g_8	st_1	st_2	st_4	st_8	def
1	1500	143	60	74	41	47	49	26	26	42	39	7	38	44	29
2	136350	346	1849	1012	733	579	550	427	437	429	404	493	493	475	317
3	271200	802	3346	1949	1506	1193	826	874	837	819	807	965	960	1005	659
4	406050	951	4940	2828	2120	1700	1520	1220	1269	1219	1162	1386	1500	1412	973
5	540900	1314	6544	3793	2866	2225	1638	1624	1640	1657	1659	1940	1904	1860	1315
6	675750	1646	8115	4720	3523	2850	2033	2060	2015	2030	1941	2345	2423	2433	1633
7	810600	2020	9735	5662	4279	3384	2468	2446	2484	2445	2386	2873	3024	2794	1999
8	645450	2292	11351	6587	4997	3891	2881	2879	2856	2848	2698	3335	3394	3289	2278
9	1080300	2648	12974	7518	5693	4472	3275	3260	3279	3285	3130	3707	3780	3766	2740
10	1215150	2953	14616	8493	6399	5061	3673	3688	3666	3630	5013	4271	4274	4239	3067
11	1350000	3324	16389	9429	7173	5604	4202	4113	4101	4067	4862	4718	4767	4705	3513

Таблица 7 – Результаты выполнения программы при компиляции с флагом O2

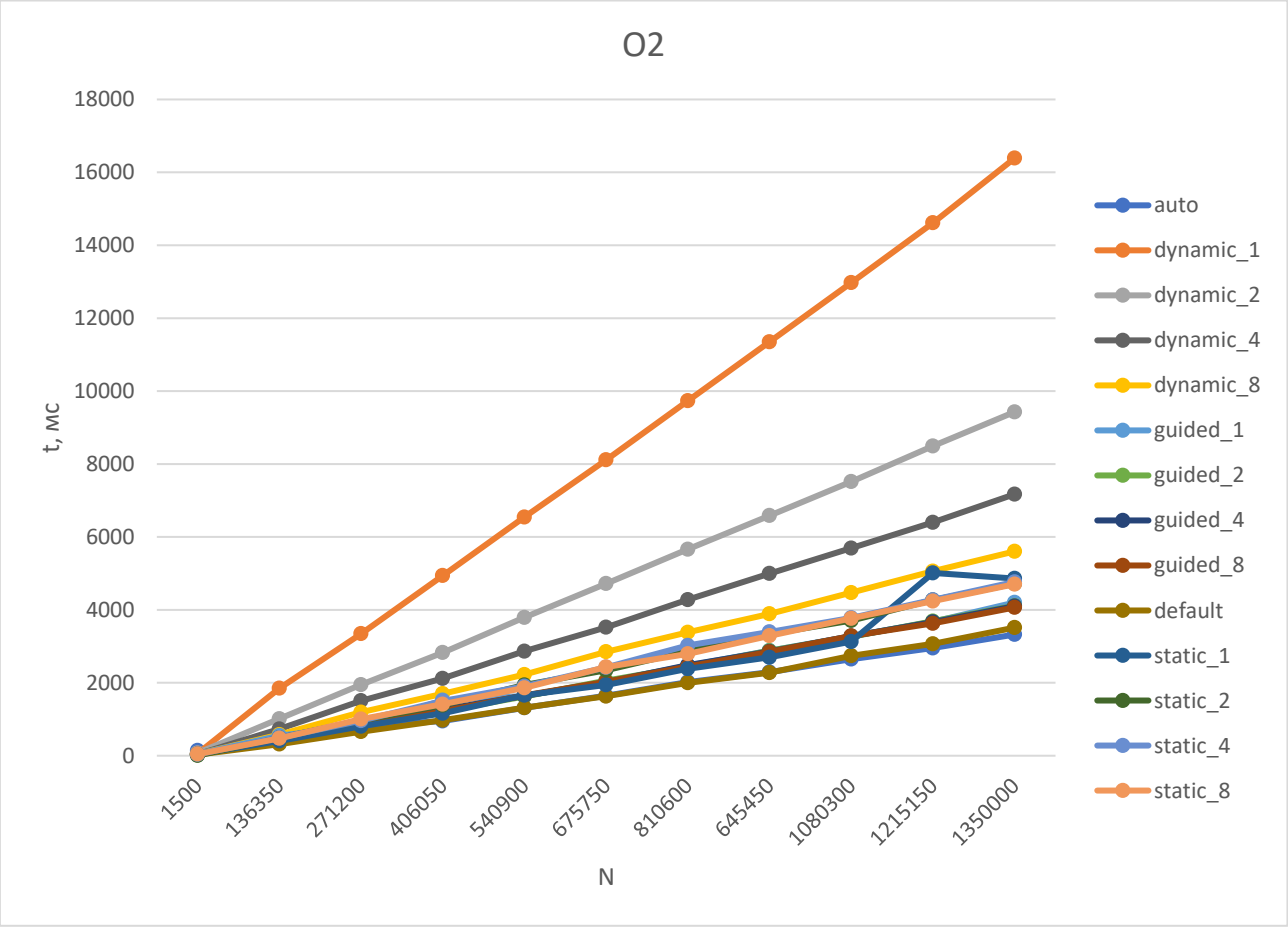

Рисунок 8 – Результаты выполнения программы при компиляции с флагом O2

6.3 Применение флага –O1

	N	auto	d_1	d_2	d_4	d_8	g_1	g_2	g_4	g_8	st_1	st_2	st_4	st_8	def
1	1500	147	69	43	27	25	35	21	33	30	135	34	43	7	41
2	136350	460	1789	1096	751	600	505	430	453	421	542	520	485	474	430
3	271200	839	3237	1945	1466	1163	885	822	819	822	1002	931	951	970	836
4	406050	1301	4857	2838	2101	1695	1265	1218	1279	1242	1489	1444	1502	1438	1260
5	540900	1660	6441	3770	2850	2213	1650	1666	1630	1675	2005	1899	1888	1828	2133
6	675750	2092	7995	4718	3478	2817	2060	2177	2058	2065	2473	2397	2339	2337	2050
7	810600	2567	9571	5643	4256	3344	2463	2511	2503	2466	2957	2863	2863	2798	2472
8	645450	2898	11142	6558	5003	3861	2850	2985	2900	2858	3420	3287	3334	3335	2900
9	1080300	3243	12723	7451	5607	4446	3283	3292	3261	3280	3878	3694	3705	3684	3289
10	1215150	3698	14295	8385	6309	4990	3706	3650	3684	3659	4443	4203	4265	4174	3756
11	1350000	4139	15916	9290	6997	5525	4100	4076	4094	4138	4835	4669	4665	4614	4098

Таблица 8– Результаты выполнения программы при компиляции с флагом O1

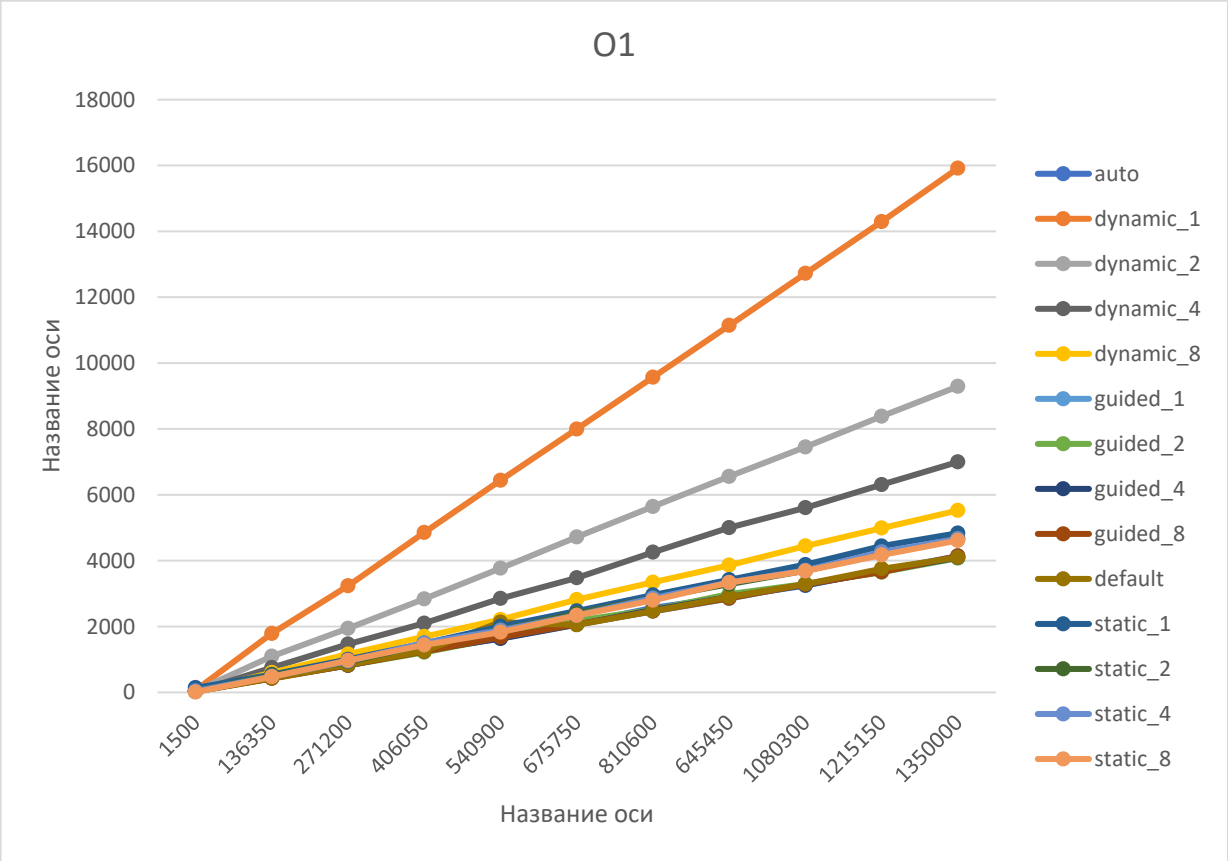

Рисунок 9 – Результаты выполнения программы при компиляции с флагом O1

Хоть результаты и сложно отличить на графике, но в целом лучше всего себя показал эксперимент, при котором компиляция проходила с флагом O3.

Выводы

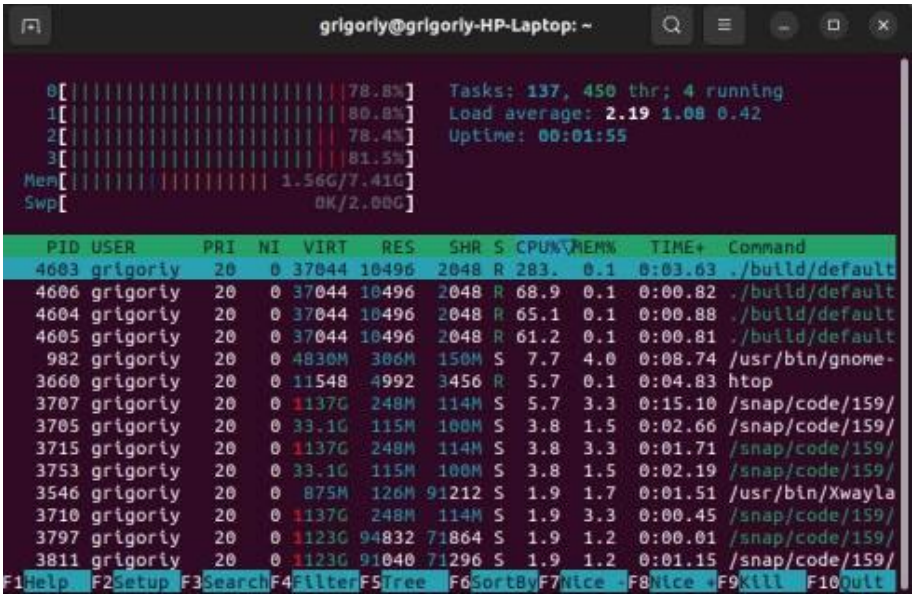

Рисунок 10 – Доказательство распараллеливания в начале

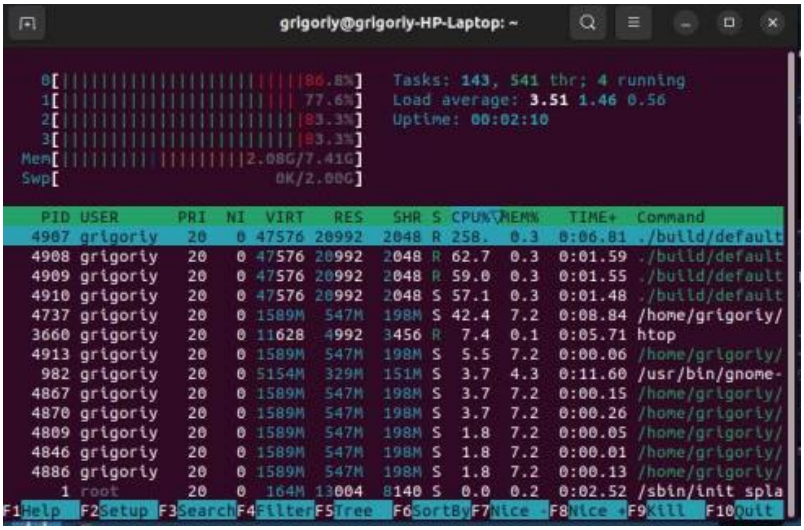

Рисунок 11 – Доказательство распараллеливания в конце

По сравнению с использованием автоматического распараллеливания максимальное параллельное ускорение составляет около 4. При сравнении различных параметров расписание наибольший прирост наблюдается со значением guided. При использовании параметра dynamic время выполнения программы существенно увеличивалось.

Было найдено значение, когда на использование распараллеливания сильно влияют накладные расходы, вследствие чего увеличивается время выполнения. Для 100 экспериментов такое значение было около 9010 элементов.

Было проведено 3 дополнительных эксперимента для различных параметров оптимизации, наилучшие результаты достигаются при использовании –O3.