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 № 3

“Распараллеливание циклов с помощью технологии OpenMP”

По дисциплине “Параллельные вычисления”

Студент группы P4114

Трофимова Полина Владимировна

Преподаватель:

Жданов Андрей Дмитриевич

Санкт-Петербург, 2024 г.

Оглавление

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.

13. Необязательное задание №1 (для получения оценки «4» и «5»). Построить график параллельного ускорения для точек $N < N1$ и найти значения N, при которых накладные расходы на распараллеливание превышают выигрыш от распараллеливания (независимо для различных типов расписания).

14. Необязательное задание №2 (для получения оценки «5»). Для лучшего результата по итогам всех экспериментов сделать еще минимум три эксперимента, заменив флаг «-O3» на другие флаги оптимизации. Построить график времени выполнения от N.

Краткая характеристика системы

Операционная система: Windows 10 Домашняя

Тип системы: 64-разрядная операционная система

Процессор: AMD Ryzen 7 5700U

Оперативная память: 8ГБ

Количество физических ядер: 8

Количество логических ядер: 16

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

Программа lab1.c

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

//void combSort(double arr[], int n) {
//    int gap = n;
//    bool swapped = true;
//    int temp;
//    while (gap != 1 || swapped) {
//        gap *= 10 / 13;
//        if (gap < 1) {
//            gap = 1;
//        }
//        swapped = false;
//        for (int i = 0; i < n - gap; i++) {
//            if (arr[i] > arr[i + gap]) {
//                temp = arr[i];
//                arr[i] = arr[i + gap];
//                arr[i + gap] = temp;
//                swapped = true;
//            }
//        }
//    }
//}

int main(int argc, char* argv[])
{
    int i, j, N;
    struct timeval T1, T2;
    //double e = exp(1.0);
    double A = 2;
    long delta_ms;
    double sum_sin = 0.0;
    double min_nonzero = 0;

    // N равен первому параметру командной строки
    N = atoi(argv[1]);

    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));

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

    unsigned int seed = 1;

        // запомнить текущее время T1
    gettimeofday(&T1, NULL);


    for (i = 0; i < 100; i++) // 100 экспериментов
    {
        seed = i; // инициализировать начальное значение ГСЧ
        sum_sin = 0.0;
```

```
        for (j = 0; j < N; j++) // Заполнить массив исходных данных размером N
        {
            M1[j] = (((double)rand_r(&seed) / (RAND_MAX)) * A); //2 + 0.1);
        }


        for (j = 0; j < N / 2; j++)
        {
            M2[j] = ((double)rand_r(&seed) / (RAND_MAX)) * A*10;
            M2_copy[j] = M2[j];
        }

#pragma omp parallel default(none) shared(N, A, M1, M2, M2_copy, i, sum_sin,
min_nonzero)
        {

        //map
#pragma omp for //schedule(auto)
        for (j = 0; j < N; j++)
        {
            // Гиперболический тангенс с последующим уменьшением на 1
            M1[j] = ((tanh(M1[j])) - 1);
        }

#pragma omp for //schedule(auto)
        for (j = 1; j < N / 2; j++)
        {
            // Десятичный логарифм, возведенный в степень e
            M2[j] = pow((M2[j] + M2_copy[j - 1]), M_E);
        }

#pragma omp single
        M2[0] = pow(M2[0], M_E);


#pragma omp for //schedule(auto)
        for (j = 0; j < N / 2; j++) // Решить поставленную задачу, заполнить массив
с результатами
        {
            //Деление (т.е. M2[i] = M1[i]/M2[i])
            M2[j] = M1[j] / M2[j];
        }

        //combSort(M2, N / 2);
        // Сортировка расчёской (Comb sort) результатов

        //минимальные не нулевой на два фора
        //double min_nonzero = M2[0];

#pragma omp for reduction(min:min_nonzero) //schedule(dynamic)
        for (j = 0; j < N / 2; j++)
        {
            if ((M2[j] != 0) && (M2[j] < min_nonzero))
            {
                min_nonzero = M2[j];
            }
        }

#pragma omp parallel for reduction (+:sum_sin) //schedule(dynamic)
        for (j = 0; j < N / 2; j++)
        {
            if ((int)(M2[j] / min_nonzero) % 2 == 0)
            {
                sum_sin += sin(M2[j]);
```

```
            }
        }
#pragma omp single
        printf("sum_sin = %.10lf\n", sum_sin);
        }
    }

    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);

    free(M1);
    free(M2);
    free(M2_copy);

    return 0;
}
```

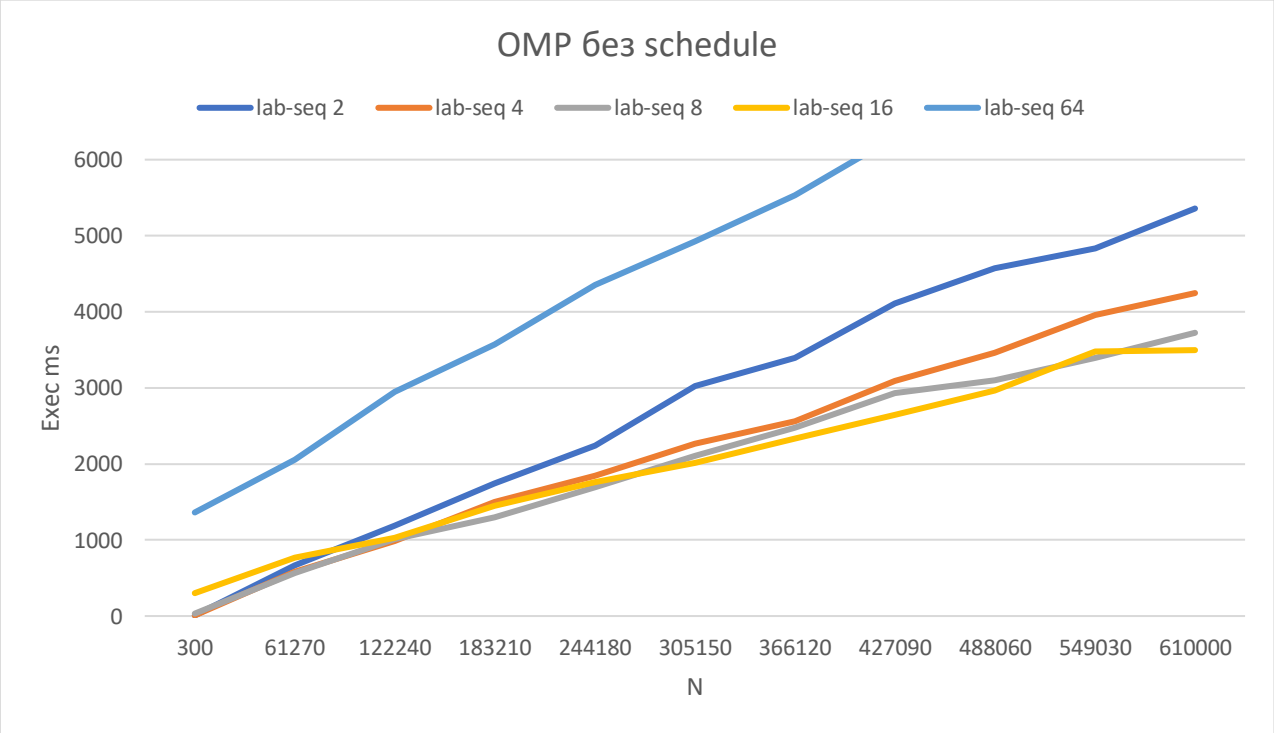
Результаты

Прямая совместимость достигается:

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

OMP без параметра schedule

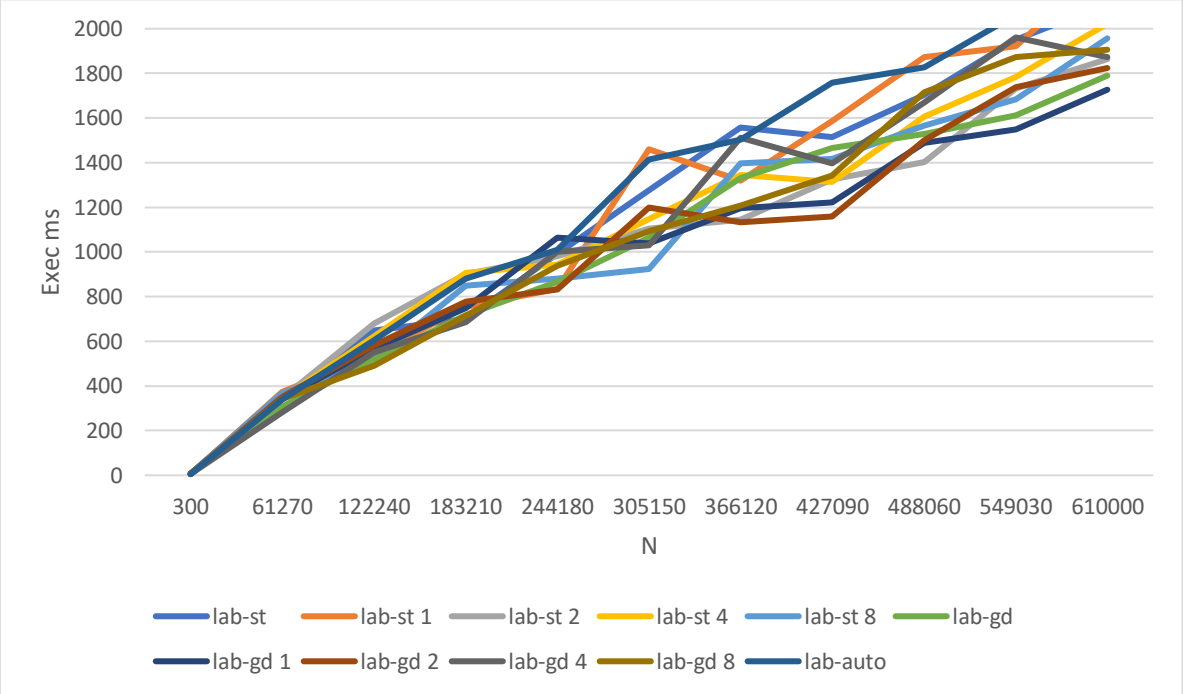
		N	lab-seq 2	lab-seq 4	lab-seq 8	lab-seq 16	lab-seq 64
1	N1	300	8	9	32	301	1362
2	N1+delta	61270	670	580	566	765	2053
3	N1+2delta	122240	1186	987	1008	1025	2951
4	N1+3delta	183210	1741	1503	1298	1448	3567
5	N1+4delta	244180	2238	1844	1693	1758	4351
6	N1+5delta	305150	3020	2263	2108	2015	4926
7	N1+6delta	366120	3390	2558	2473	2335	5533
8	N1+7delta	427090	4107	3087	2927	2644	6294
9	N1+8delta	488060	4570	3464	3096	2963	6935
10	N1+9delta	549030	4830	3955	3397	3479	7404
11	N2	610000	5358	4246	3724	3496	8291



Результаты заметно улучшаются, при увеличении потоков до 8/16, далее идет ухудшение.

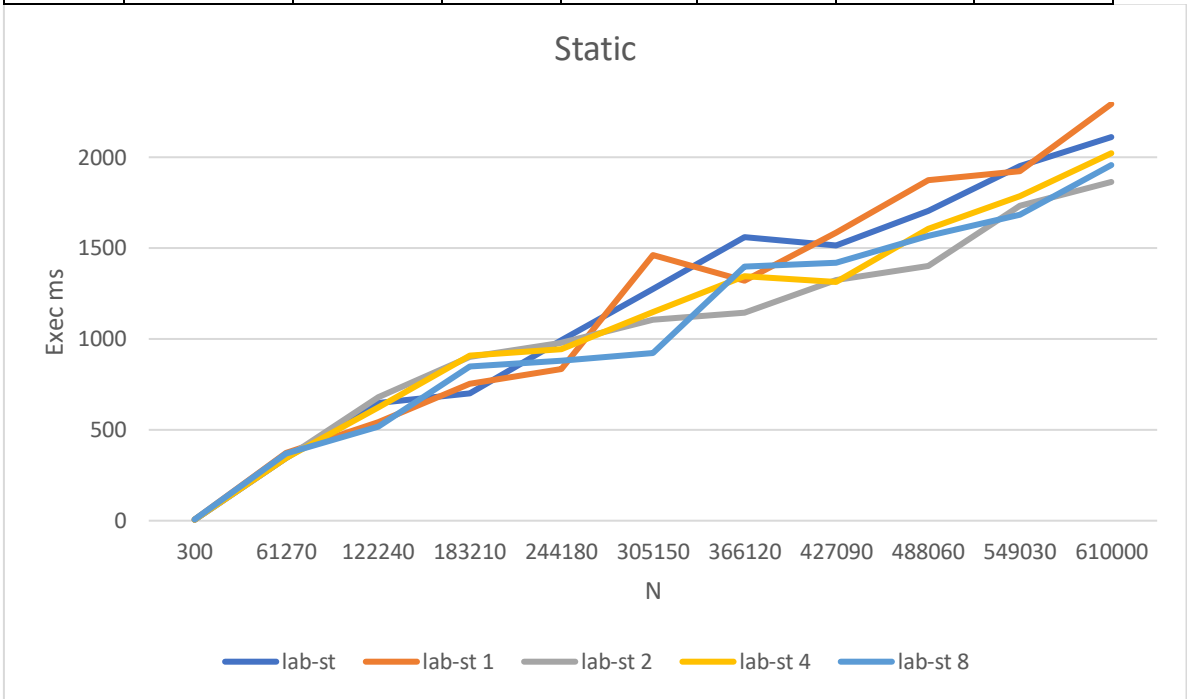
Сравнение различных вариантов schedule

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



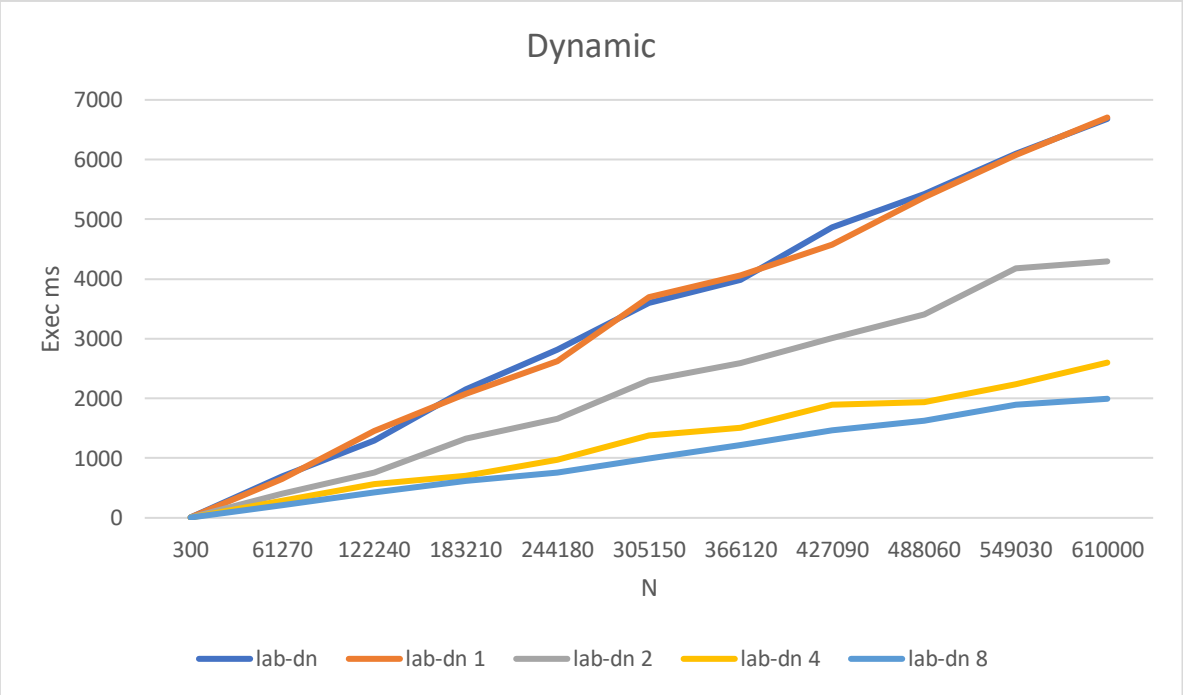
Сравнение static с различными chunk_size

		N	lab-st	lab-st 1	lab-st 2	lab-st 4	lab-st 8
1	N1	300	5	6	5	6	7
2	N1+delta	61270	345	374	345	344	369
3	N1+2delta	122240	649	542	681	622	516
4	N1+3delta	183210	699	755	901	907	848
5	N1+4delta	244180	993	835	980	943	880
6	N1+5delta	305150	1276	1461	1106	1147	923
7	N1+6delta	366120	1559	1320	1145	1345	1397
8	N1+7delta	427090	1514	1586	1324	1315	1418
9	N1+8delta	488060	1705	1874	1403	1606	1566
10	N1+9delta	549030	1951	1923	1732	1785	1683
11	N2	610000	2111	2293	1864	2022	1957



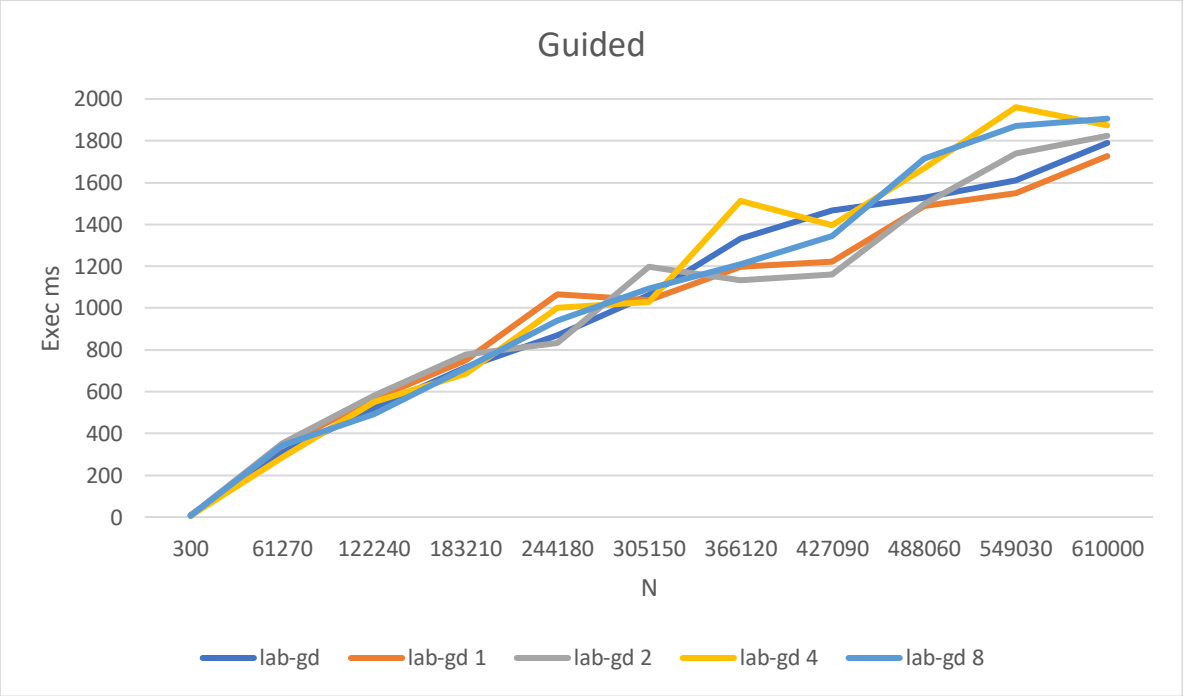
Сравнение dynamic с различными chunk_size

		N	lab-dn	lab-dn 1	lab-dn 2	lab-dn 4	lab-dn 8
1	N1	300	8	8	7	5	6
2	N1+delta	61270	690	654	404	284	213
3	N1+2delta	122240	1297	1457	762	567	421
4	N1+3delta	183210	2149	2077	1322	705	624
5	N1+4delta	244180	2818	2629	1659	968	762
6	N1+5delta	305150	3601	3696	2303	1375	999
7	N1+6delta	366120	3982	4058	2588	1514	1219
8	N1+7delta	427090	4861	4577	3007	1891	1466
9	N1+8delta	488060	5425	5364	3408	1933	1631
10	N1+9delta	549030	6101	6075	4178	2238	1894
11	N2	610000	6682	6705	4295	2600	1994



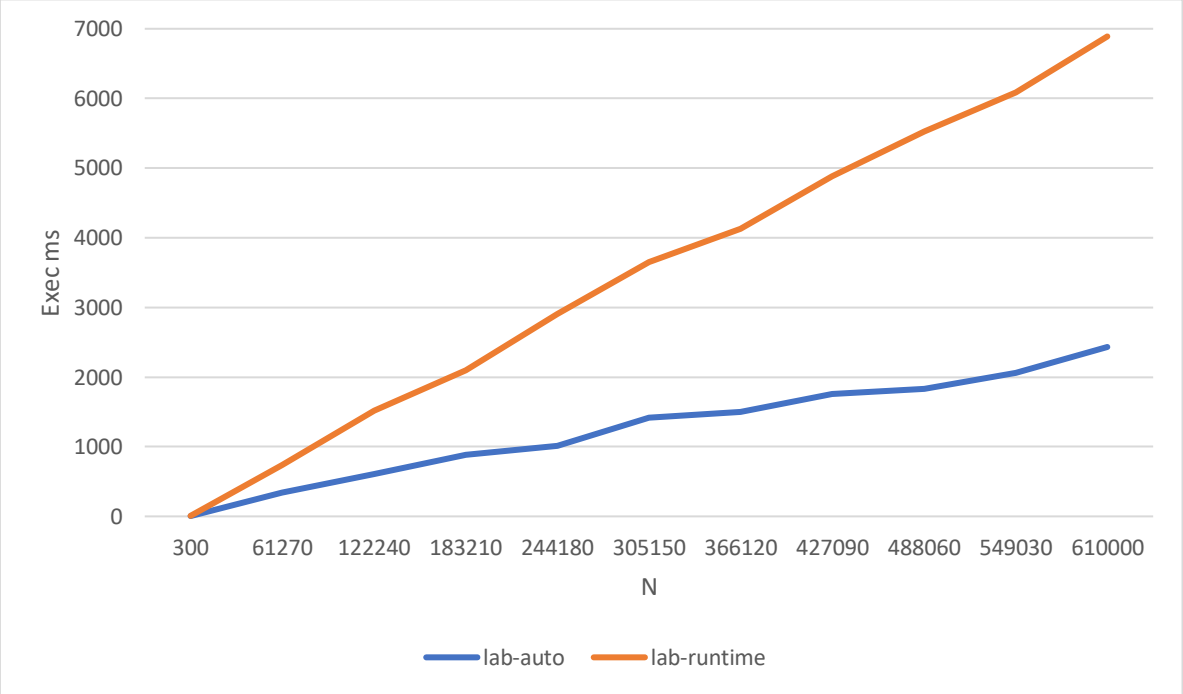
Сравнение guided с различными chunk_size

		N	lab-gd	lab-gd 1	lab-gd 2	lab-gd 4	lab-gd 8
1	N1	300	9	6	7	6	6
2	N1+delta	61270	308	342	351	283	342
3	N1+2delta	122240	521	567	583	550	492
4	N1+3delta	183210	717	749	779	685	714
5	N1+4delta	244180	868	1065	833	1001	939
6	N1+5delta	305150	1064	1038	1198	1030	1092
7	N1+6delta	366120	1331	1197	1134	1512	1209
8	N1+7delta	427090	1467	1222	1159	1397	1344
9	N1+8delta	488060	1529	1488	1499	1669	1714
10	N1+9delta	549030	1611	1549	1739	1961	1872
11	N2	610000	1790	1827	1824	1773	1906



Сравнение auto и runtime

		N	lab-auto	lab-runtime
1	N1	300	5	8
2	N1+delta	61270	338	736
3	N1+2delta	122240	607	1520
4	N1+3delta	183210	880	2096
5	N1+4delta	244180	1009	2907
6	N1+5delta	305150	1414	3649
7	N1+6delta	366120	1502	4128
8	N1+7delta	427090	1759	4881
9	N1+8delta	488060	1826	5530
10	N1+9delta	549030	2063	6083
11	N2	610000	2432	6890



Наилучший результат:

Количество потоков – 8

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

Сложность:

1. Заполнение массивов M1 и M2: O(N)

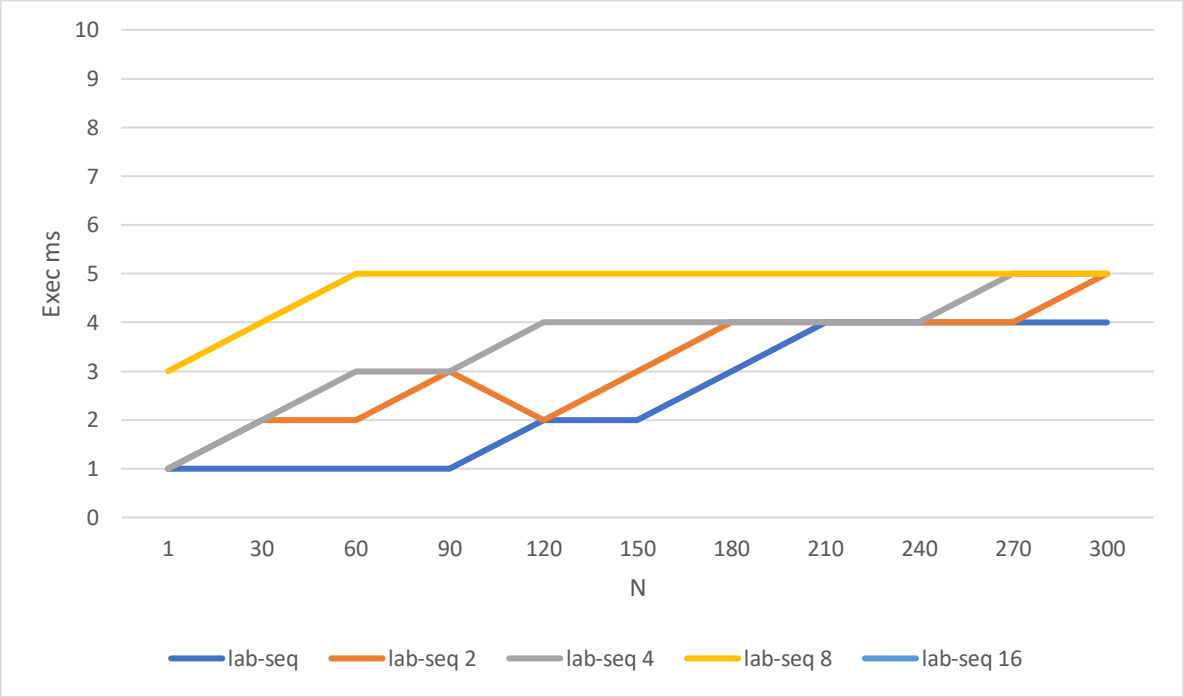
2. Параллельный цикл для вычисления новых значений M1 и M2: O(N) (так как каждый элемент обрабатывается независимо)

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)

График параллельного ускорения для N < 300

		N	lab-seq	lab-seq 2	lab-seq 4	lab-seq 8	lab-seq 16
1	N1	1	1	1	1	3	33
2	N1+delta	30	1	2	2	4	19
3	N1+2delta	60	1	2	3	5	24
4	N1+3delta	90	1	3	3	5	16
5	N1+4delta	120	2	2	4	5	76
6	N1+5delta	150	2	3	4	5	19
7	N1+6delta	180	2	4	4	5	49
8	N1+7delta	210	3	4	4	5	24
9	N1+8delta	240	4	4	4	5	24
10	N1+9delta	270	4	4	5	5	70
11	N2	300	4	5	5	5	23



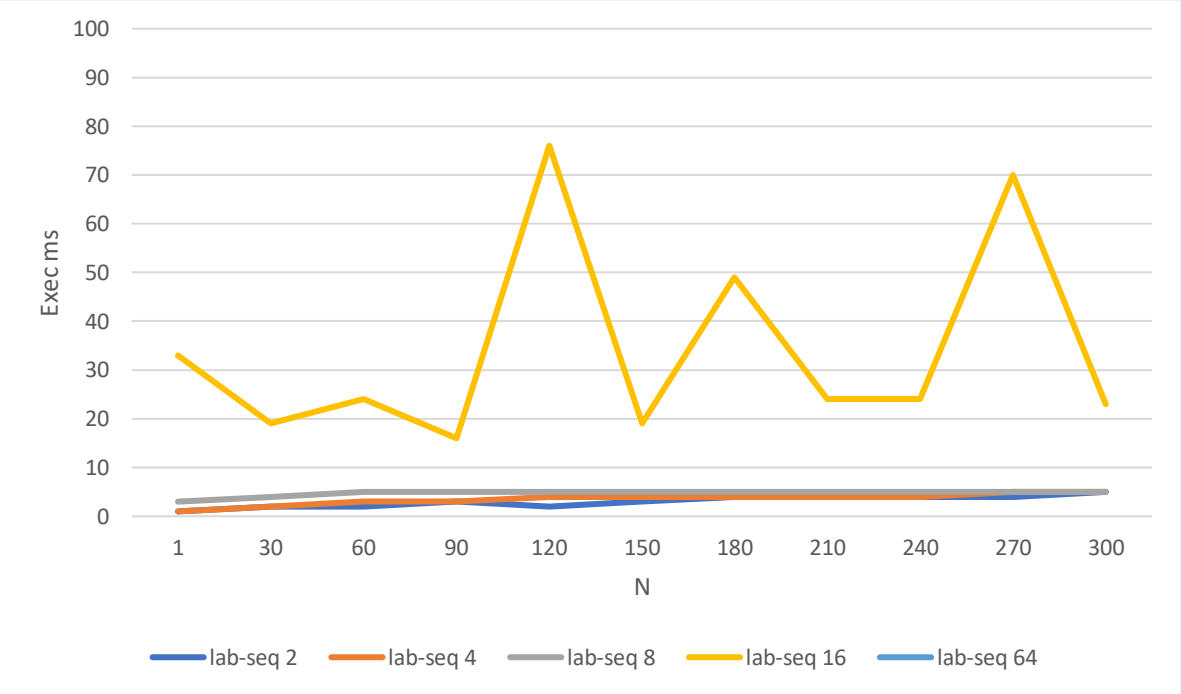
До значения 120, программа без распараллеливания работает быстрее.

Применение различных флагов оптимизации

Применение флага **–O3** приводится в самой первой таблице.

-O2 С параметром -O2, компилятор попытается увеличить производительность кода, не тратя много времени.

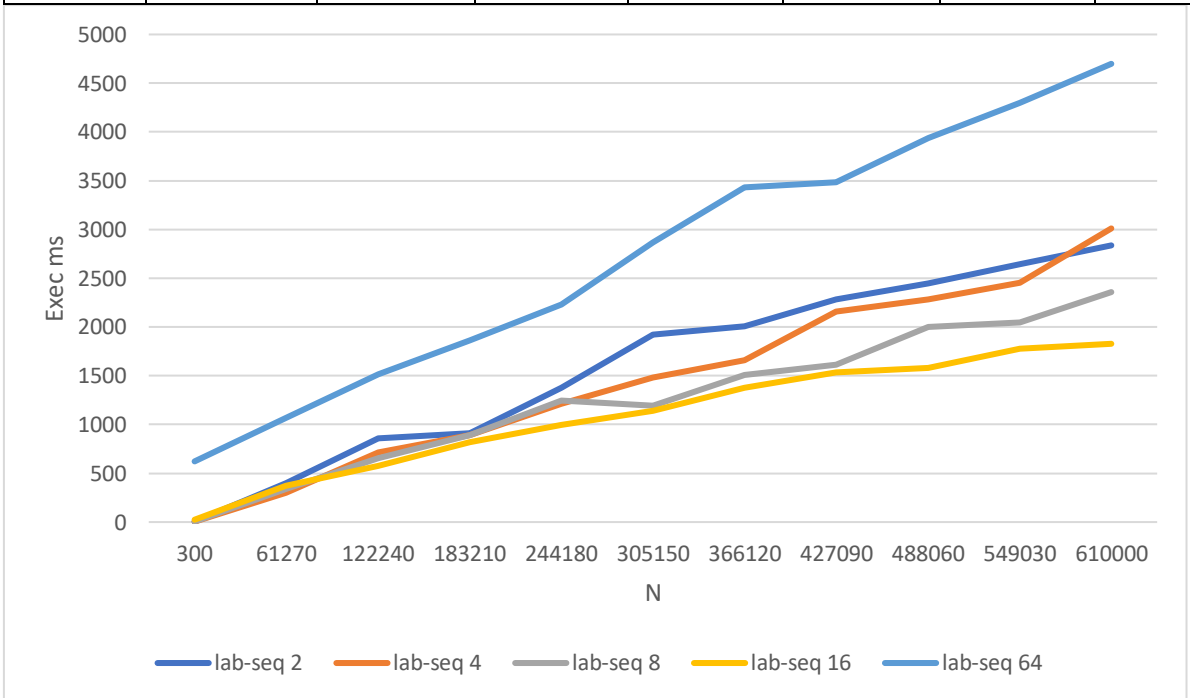
		N	lab-seq 2	lab-seq 4	lab-seq 8	lab-seq 16	lab-seq 64
1	N1	300	5	4	5	18	694
2	N1+delta	61270	473	398	347	326	937
3	N1+2delta	122240	766	751	670	627	1528
4	N1+3delta	183210	1105	1021	631	785	1893
5	N1+4delta	244180	1402	1162	1135	925	2218
6	N1+5delta	305150	1718	1468	1339	1158	2605
7	N1+6delta	366120	1850	1779	1510	1365	3248
8	N1+7delta	427090	2166	2131	1889	1507	3603
9	N1+8delta	488060	2590	2249	2050	1642	4048
10	N1+9delta	549030	2905	2259	1965	1809	4295
11	N2	610000	3066	2451	2342	1954	4616



.

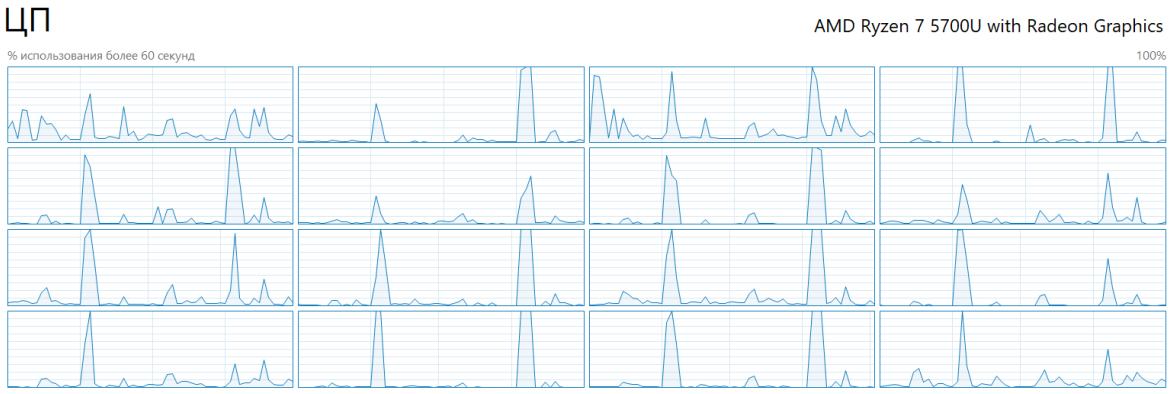
-O1

		N	lab-seq 2	lab-seq 4	lab-seq 8	lab-seq 16	lab-seq 64
1	N1	300	5	4	5	24	622
2	N1+delta	61270	396	298	338	374	1067
3	N1+2delta	122240	859	715	656	573	1518
4	N1+3delta	183210	914	890	891	821	1863
5	N1+4delta	244180	1376	1214	1244	996	2233
6	N1+5delta	305150	1922	1485	1193	1139	2867
7	N1+6delta	366120	2010	1657	1507	1380	3435
8	N1+7delta	427090	2284	2156	1616	1535	3482
9	N1+8delta	488060	2449	2281	2000	1578	3939
10	N1+9delta	549030	2642	2452	2047	1778	4297
11	N2	610000	2837	3011	2358	1828	4699



Лучше всего проявляет себя параметр –O3.

Выводы



По сравнению с использованием автоматического распараллеливания максимальное параллельное ускорение составляет около 8. При сравнении различных параметров расписание наибольший прирост наблюдается со значением guided. При использовании параметра dynamic время выполнения программы существенно увеличивалось.

Для N < N1 было найдено значение, когда использование распараллеливания увеличивает время выполнения. Для 100 экспериментов такое значение было около 120 элементов.

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