УНИВЕРСИТЕТ ИТМО

Лабораторная работа №3 "Распараллеливание циклов с помощью технологии OpenMP"

Дисциплина

Параллельные вычисления

Автор

Дмитрий Рачковский

22 декабря 2020 г.

Содержание

1 Введение

1.1 Цель работы

В данной работе необходимо исследовать эффективность распараллеливания программ на языке C с помощью технологии OpenMP. Для этого одна и та же программа, производящая достаточное количество вычислений с использованием канонических циклов for, компилируется сначала без использования OpenMP, а затем с использованием распараллеливания циклов for при различных типах расписания. Время выполенения полученных выполняемых файлов сравнивается. Также производится контроль результата вычислений, который не должен меняться (в пределах допустимой погрешности) при распараллеливании, гарантируя правильность выполнения.

1.2 Использованное оборудование

Для проведения экспериментов использовалась виртуальная машина с OS Debian. Виртуальная машина получила доступ к 6 ядрам (используется 64-битный процессор Intel Core i7-8750H @ 2.20 GHz с 6 физическими и 12 логическими ядрами) и 6 ГБ оперативной памяти (из 16, доступных в системе). Во время выполнения экспериментов система была подключена к источнику питания.

Версия использованного компилятора: gcc (Debian 6.3.0-18+deb9u1) 6.3.0 20170516.

Версия OpenMP: 4.5 (201511)

2 Ход работы

2.1 Используемая программа

Для проведения экспериментов программа, использованная в лабораторной работе №1, была модифицирована следующим образом:

- Перед всеми циклами for была добавлена директива *#pragma omp parallel for default(none) private(...) shared(...)*
- Все циклы были проверены на внутренние зависимости чтоб избежать ошибок при распараллеливании.
- Программа была проверена на прямую совместимость с компиляторами без поддержки OpenMP.
- Программа была распараллелена с использованием разных типов расписания (*static, dynamic, guided*) и разного chunk_size. Для этого параметра были искользованы как фиксированные значения $(1, 2, 3, 4, 5, 10, 50, 100)$, так и значения, зависящие от размера массива $(N/100, N/20, N/10, N/5, N/3, N/2)$.

Полный текст программы может быть найден в приложении 4.1.

2.2 Получение данных

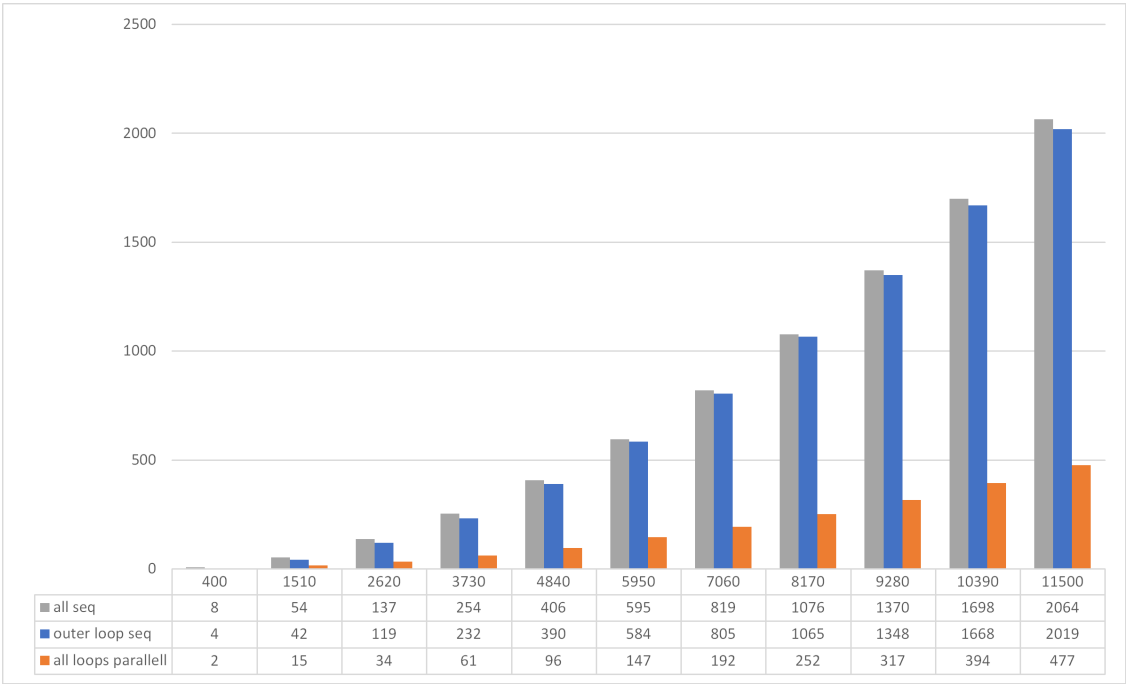
Полученная программа компилируется без подключения OpenMP для проверки прямой совместимости. Затем эта же программа распараллеливания с использованием флага -openmp. Команды для компиляции программы могут быть найдены в приложении 4.2.

Полученная программа запускается со значениями $N = N1, N1 + \Delta, N1 + 2\Delta, N1 + 3\Delta ... N2$, найденными лабораторной работе №1.

2.3 Анализ результатов

2.3.1 Эксперимент без указания расписания

По выполнению первого эксперимента без указания расписания я заметил, что время выполнения распараллеленных программ очень слабо отличается от линейной программы, и улучшение во времени работы очень незначительное (около двух процентов для больших значений. Тогда я попробовал также распараллелить внешний цикл, и получил среднее улучшение времени работы на 75% независимо от значения N. Время работы видно на следущем графике:



	400	1510	2620	3730	4840	5950	7060	8170	9280	10390	11500
all seq	8	54	137	254	406	595	819	1076	1370	1698	2064
outer loop seq	4	42	119	232	390	584	805	1065	1348	1668	2019
all loops parallell	2	15	34	61	96	147	192	252	317	394	477

Рис. 1: Производительность распараллеленой программы без указания расписания

Соответственно, параллельное ускорение для нераспараллеленного внешнего цикла сходит к единице с увеличением параметра N, тогда как для распараллеленного внешнего цикла параллельное ускорение всегда на уровне 50%.

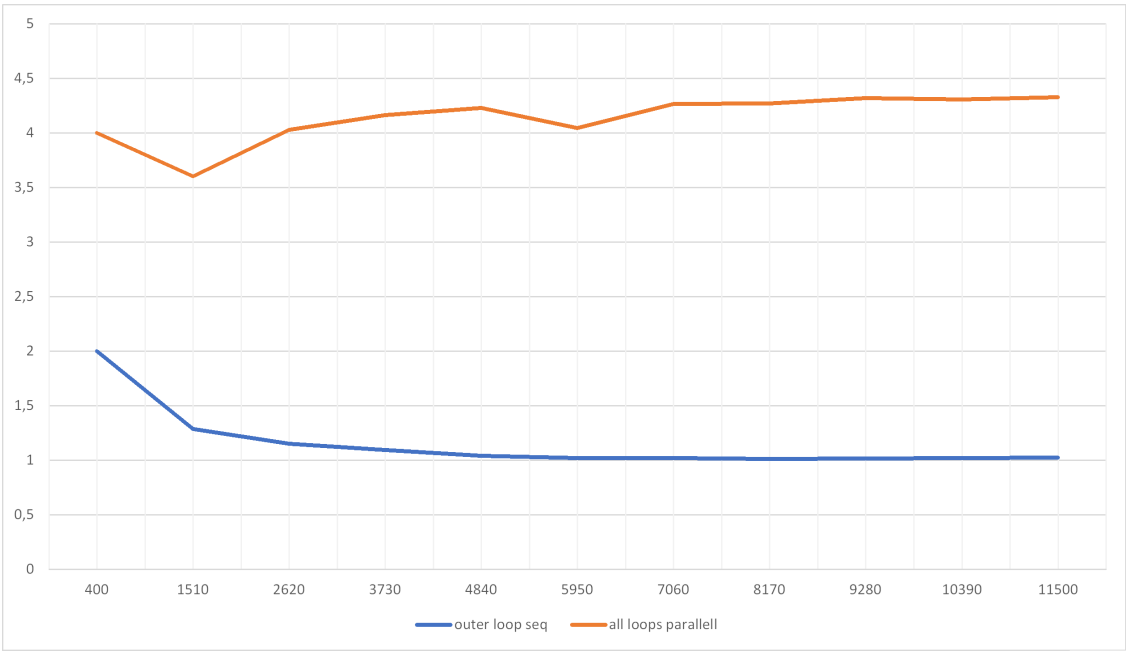


Рис. 2: Параллельное ускорение распараллеленой программы без указания расписания

2.3.2 Эксперимент с указанием расписания и линейным внешним циклом

После первого эксперимента был проведён повторный с указанием различных типов расписания (static, dynamic, guided) и разных значений параметра chunk_size. Для этого параметра были искользованы как фиксированные значения $(1, 2, 3, 4, 5, 10, 50, 100)$, так и значения, зависящие от размера массива $(N/100, N/20, N/10, N/5, N/3, N/2)$. Результат можно увидеть в следующей таблице (красным подсвечены минимальные значения для каждого N):

	400	1510	2620	3730	4840	5950	7060	8170	9280	10390	11500
seq	8	55	138	254	408	717	818	1077	1371	1700	2063
static,1	5	45	124	296	445	736	1658	1227	1501	1772	2043
static,2	5	44	124	300	499	750	1014	1298	1559	1874	2217
static,3	5	43	124	298	500	752	1011	1261	1553	1841	2217
static,4	5	43	125	300	495	748	1010	1261	1557	1868	2186
static,5	5	43	124	298	496	745	1010	1295	1551	1864	2207
static,10	5	43	122	295	497	746	1012	1287	1552	1866	2211
static,50	5	43	122	300	494	745	1006	1284	1547	1866	2188
static,100	6	44	123	301	496	743	1011	1257	1551	1831	2204
static,N/100	5	43	130	296	494	749	1011	1286	1529	1863	2206
static,N/20	5	43	151	306	497	747	990	1290	1554	1844	2212
static,N/10	5	44	157	303	494	752	1016	1296	1529	1873	2215
static,N/5	6	47	166	312	509	767	1034	1318	1557	1901	2251
static,N/3	7	52	175	326	536	792	1067	1308	1632	1941	2270
static,N/2	9	60	188	349	571	827	1110	1366	1712	1997	2359
dynamic,1	6	50	165	320	524	788	1054	1334	1592	1926	2275
dynamic,2	5	46	160	335	503	756	1003	1307	1580	1865	2214
dynamic,3	5	45	157	327	498	758	1023	1301	1565	1879	2233
dynamic,4	5	44	154	323	506	752	1020	1272	1536	1873	2222
dynamic,5	5	44	151	321	499	752	1020	1261	1543	1870	2198
dynamic,10	4	43	154	318	502	746	1013	1287	1556	1864	2206
dynamic,50	5	43	152	321	493	746	1012	1282	1532	1860	2210
dynamic,100	6	44	156	325	494	748	1006	1286	1539	1861	2201
dynamic,N/100	5	43	150	312	493	749	1012	1284	1549	1831	2204
dynamic,N/20	4	43	153	304	495	747	1015	1262	1716	1838	2213
dynamic,N/10	5	44	153	297	493	748	1017	1289	1555	1844	2188
dynamic,N/5	6	48	161	309	516	769	1040	1319	1589	1904	2256
dynamic,N/3	7	52	172	323	535	798	1043	1351	1629	1940	2292
dynamic,N/2	9	59	189	349	561	823	1107	1366	1691	1995	2357
guided,1	5	43	150	299	491	746	1010	1278	1551	1858	2204
guided,2	5	43	121	293	495	740	1011	1281	1545	1836	2205
guided,3	5	43	124	269	491	741	1012	1279	1544	1833	2202
guided,4	5	43	127	294	498	751	1011	1282	1544	1856	2201
guided,5	4	43	130	294	493	743	1010	1280	1546	1855	2177
guided,10	5	43	131	294	496	742	1013	1282	1543	1856	2201
guided,50	5	43	149	295	495	744	1069	1285	1542	1858	2201
guided,100	6	44	154	296	497	748	1102	1282	1549	1860	2175
guided,N/100	5	43	152	294	496	746	1016	1283	1545	1833	2203
guided,N/20	5	44	155	296	496	752	1018	1259	1521	1864	2206
guided,N/10	5	44	156	298	497	746	1017	1292	1530	1867	2213
guided,N/5	6	48	132	310	450	772	1016	1316	1592	1901	2259
guided,N/3	7	52	184	328	532	789	1063	1346	1632	1934	2344
guided,N/2	9	60	193	349	560	834	1108	1395	1668	2000	2355

Рис. 3: Время работы распараллеленой программы при линейном внешнем цикле

Можно заметить, что для небольших значений N параллелизация приносит улучшение времени выполнения, при этом небольшие значения параметра chink_size предпочтительнее:

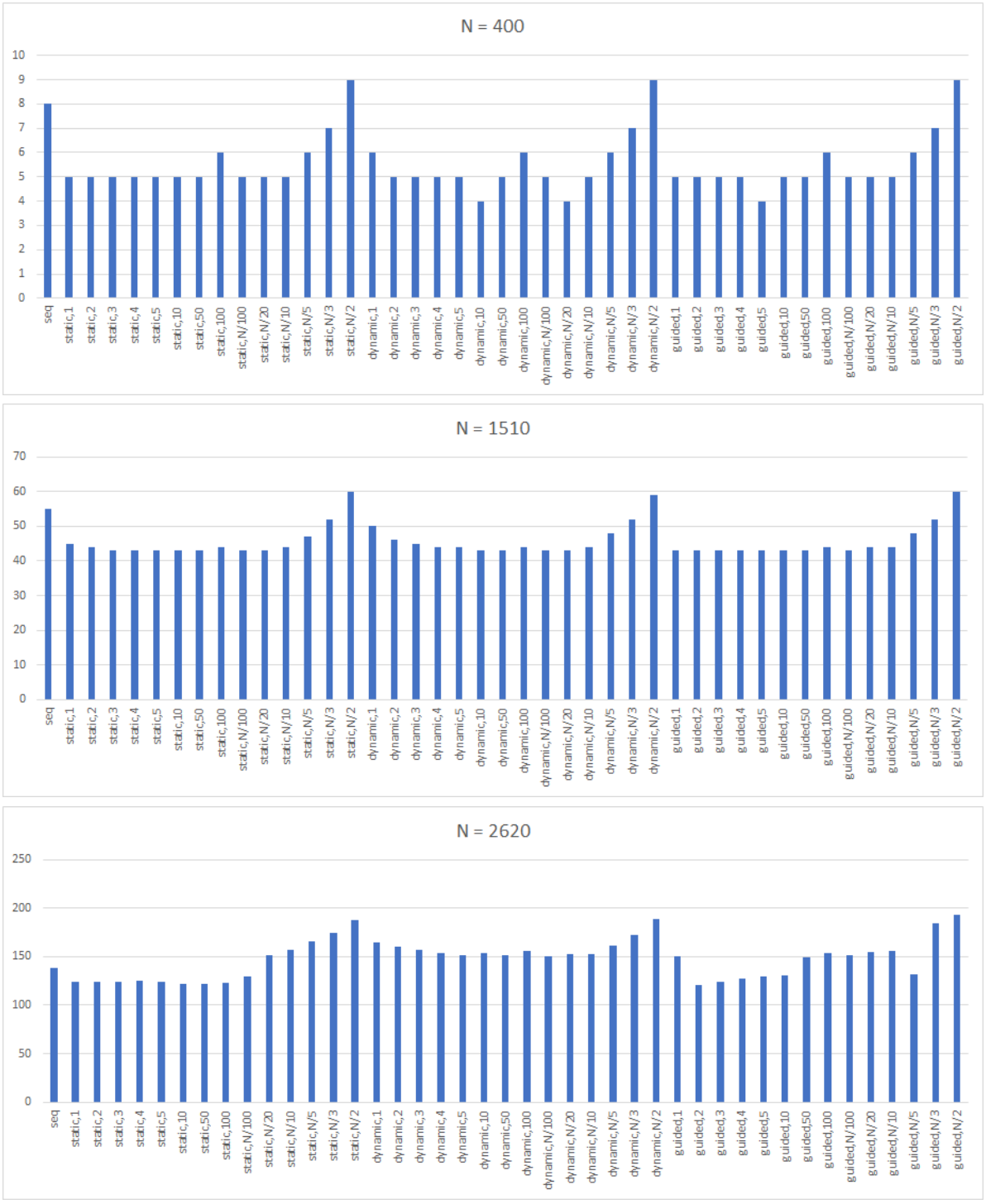


Рис. 4: Разные виды расписания при небольших значениях N

При этом с увеличением числа N улучшение времени работы при использовании параллелизации всё меньше, после чего совсем сходит на нет:

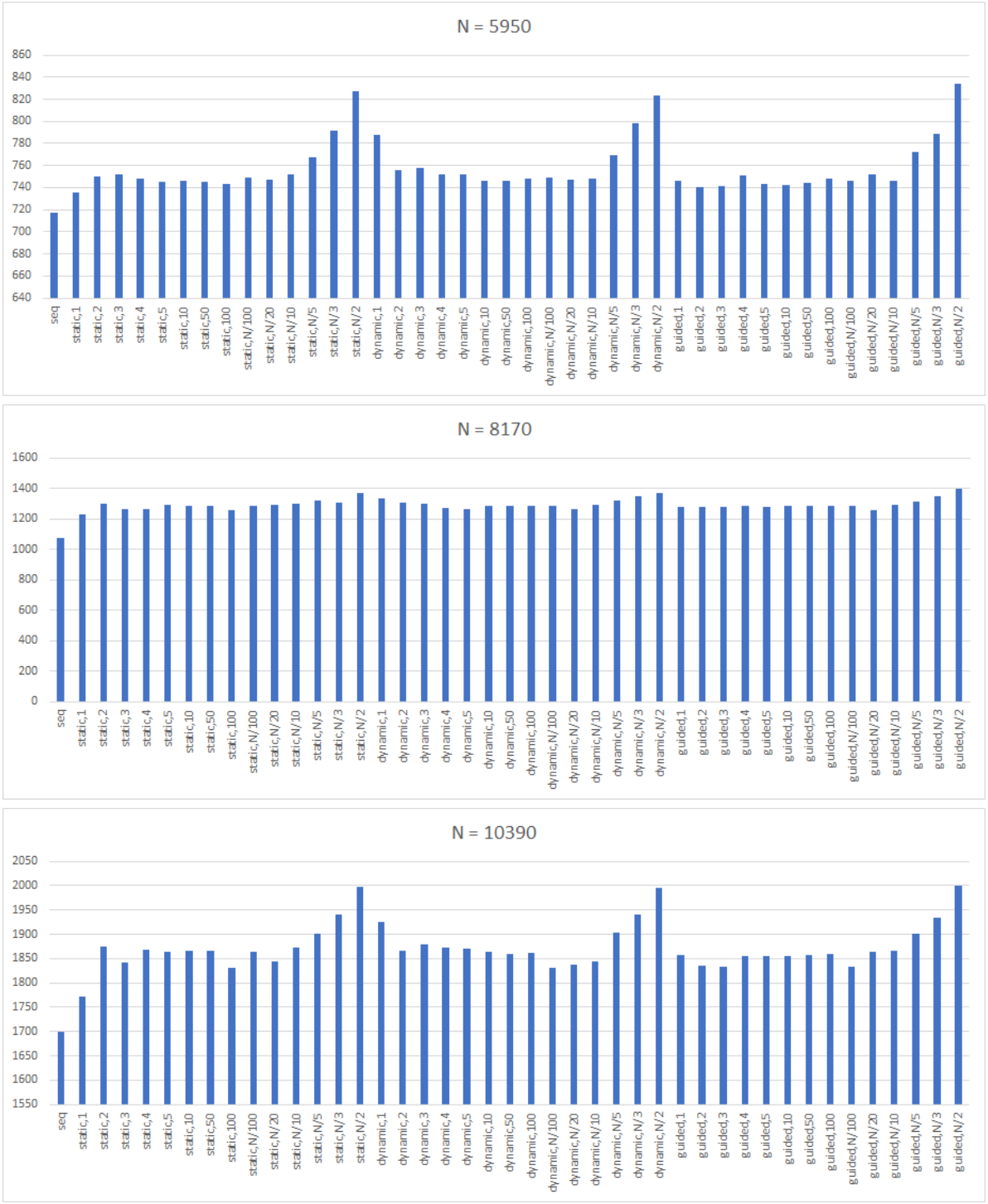


Рис. 5: Разные виды расписания при больших значениях N

Соответственно, параллельное ускорение держится немного ниже нуля для практически всех значений N, кроме самых маленьких:

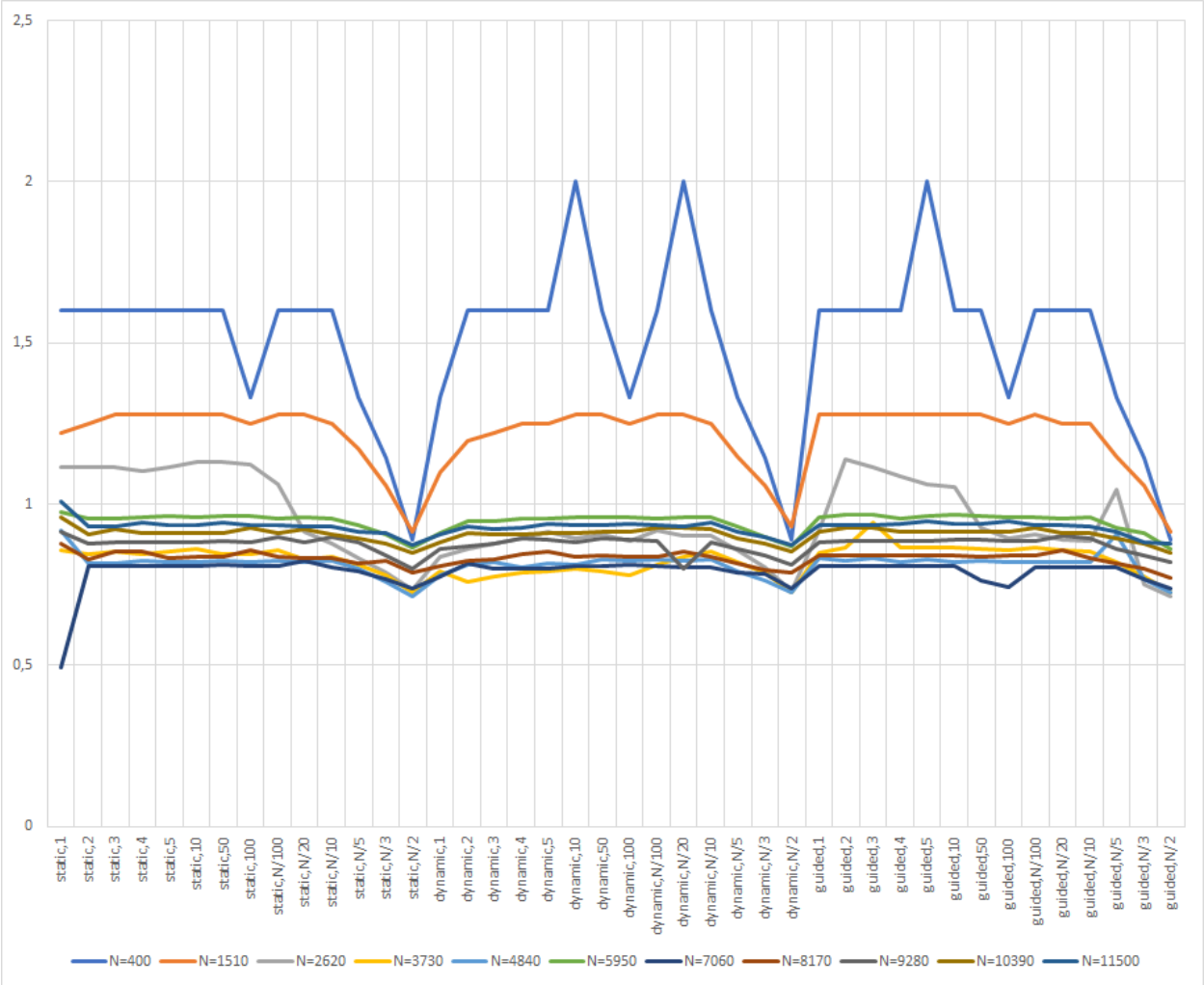


Рис. 6: Параллельное распараллеленой программы без указания расписания

Отсутствие положительного эффекта от параллелизации обусловлено, на мой взгляд, тем, что большую часть программы составляет сортировка массива, котороя не была распараллелена; и при возрастании длины массива распараллеливание остальных частей не даёт большого преимущества.

2.3.3 Эксперимент с указанием расписания и распараллеленным внешним циклом

Теперь проведём подобный эксперимент, но при этом также распараллелим внешний цикл. Время работы в данном случае видно в следующей таблице:

	N=400	N=1510	N=2620	N=3730	N=4840	N=5950	N=7060	N=8170	N=9280	N=10390	N=11500
seq	8	55	137	254	407	595	815	1076	1372	1697	2064
static,1	3	14	33	59	94	139	186	247	310	385	469
static,2	2	15	35	63	103	169	221	326	431	546	684
static,3	2	14	32	58	95	172	239	321	402	502	615
static,4	3	14	35	64	119	198	272	358	454	562	679
static,5	2	15	35	63	131	188	264	293	440	546	660
static,10	2	14	35	63	132	188	261	337	436	535	654
static,50	9	57	139	256	414	600	832	1084	1379	1710	2076
static,100	9	57	139	256	414	605	823	1078	1372	1701	2065
static,N/100	2	21	74	191	393	606	821	1078	1366	1700	2065
static,N/20	4	57	140	256	410	597	821	1078	1371	1702	2064
static,N/10	7	57	139	257	409	597	820	1077	1372	1701	2066
static,N/5	9	57	139	256	410	597	822	1078	1369	1703	2056
static,N/3	9	57	139	256	409	596	820	1077	1370	1696	2064
static,N/2	9	57	139	257	409	596	822	1075	1372	1703	2066
dynamic,1	2	15	34	63	99	139	191	250	320	431	477
dynamic,2	2	16	36	66	119	163	234	302	447	566	685
dynamic,3	2	14	32	59	116	177	240	324	411	501	614
dynamic,4	3	14	35	65	129	202	274	361	456	563	680
dynamic,5	3	15	35	64	135	195	267	351	442	543	664
dynamic,10	2	14	34	63	134	190	263	339	437	538	652
dynamic,50	9	58	139	256	413	602	832	1081	1376	1705	2069
dynamic,100	9	58	139	256	415	604	829	1080	1373	1697	2059
dynamic,N/100	2	21	74	191	395	597	819	1078	1369	1702	2057
dynamic,N/20	4	58	139	257	410	596	821	1078	1371	1702	2064
dynamic,N/10	7	57	139	257	411	597	820	1078	1371	1700	2064
dynamic,N/5	9	57	139	257	416	594	820	1079	1372	1700	2065
dynamic,N/3	9	57	139	256	410	596	822	1079	2030	1701	2064
dynamic,N/2	9	57	140	256	410	597	820	1078	1377	1694	2060
guided,1	2	13	33	61	96	137	186	245	325	387	468
guided,2	2	14	32	58	104	142	197	287	396	511	625
guided,3	2	15	35	64	125	201	275	392	455	564	682
guided,4	2	14	32	59	115	179	250	351	406	506	610
guided,5	2	15	35	64	131	199	274	394	453	562	685
guided,10	2	15	34	71	130	190	258	391	427	531	645
guided,50	9	58	139	265	415	600	821	1128	1379	1708	2062
guided,100	9	57	139	262	415	604	827	1089	1371	1697	2064
guided,N/100	2	20	74	198	395	598	822	1077	1367	1702	2058
guided,N/20	4	57	139	261	412	596	821	1078	1372	1702	2065
guided,N/10	7	57	139	258	417	596	820	1079	1371	1699	2059
guided,N/5	9	57	139	264	412	596	821	1076	1372	1710	2062
guided,N/3	9	57	139	255	409	596	820	1079	1367	1708	2056
guided,N/2	9	58	139	257	408	597	819	1075	1369	1701	2058

Рис. 7: Время работы программы при распараллеленном внешнем цикле

Это также показано на следующих графиках:

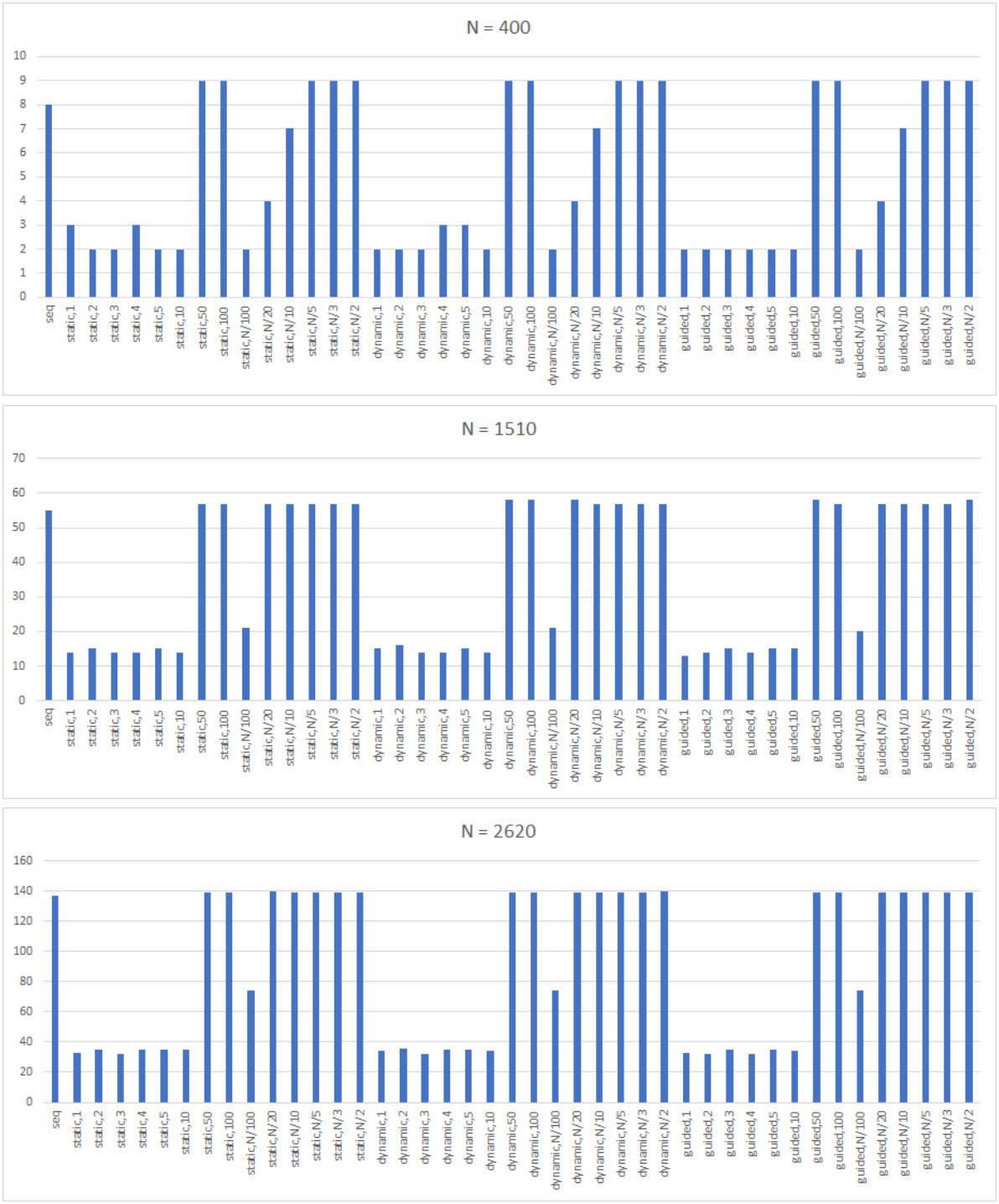


Рис. 8: Разные виды расписания при небольших значениях N

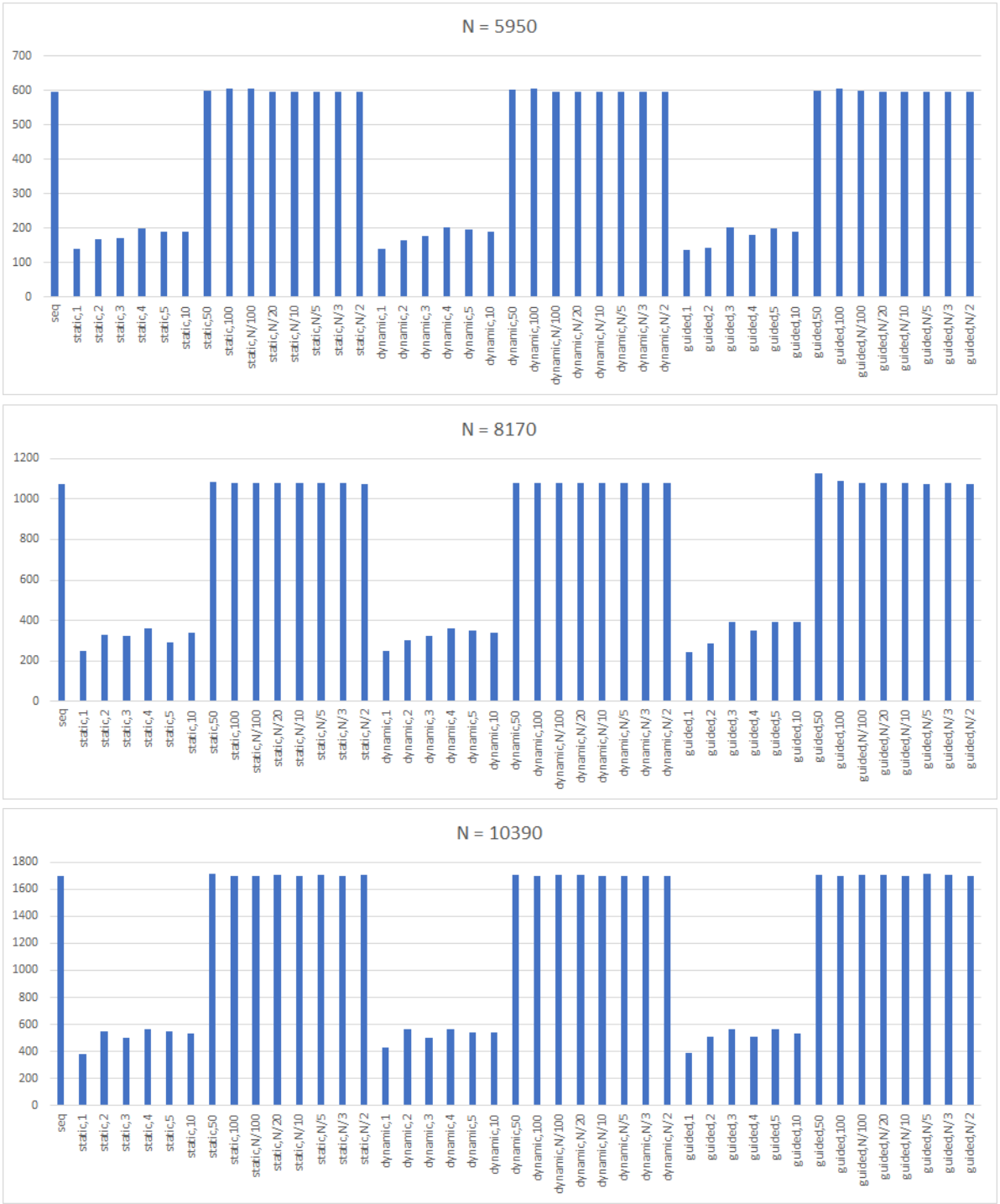


Рис. 9: Разные виды расписания при больших значениях N

Можно заметить значительное улучшение времени выполнения для всех значений N при небольшом значении параметра chunk_size. При значительном увеличении этого параметра параллелизация происходит неэффективно и время выполнения почти не уменьшается.

Параллельное ускорение в данном случае стабилизируется на значении 1 для самых неудачных типов расписания (с самыми большими значениями chunk_size), на значении 4.3 для самых удачных типов расписания (все три типа со значением chunk_size=1) и на значении 3 для всех остальных расписаний (при средних значениях chunk_size). Это видно на следующих графиках:

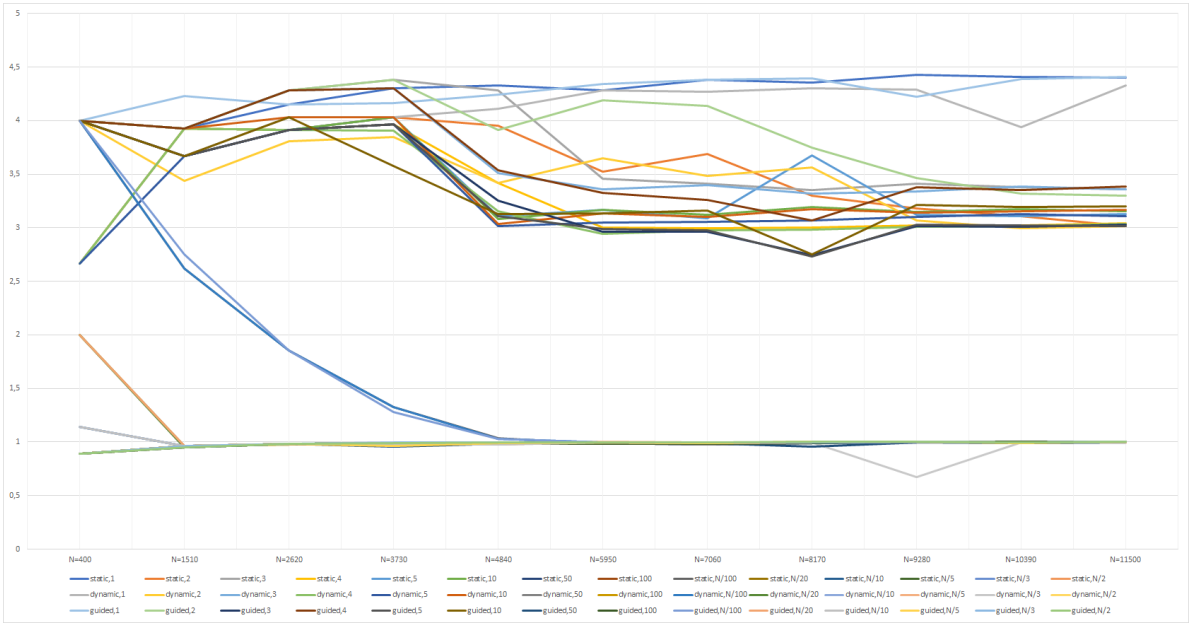


Рис. 10: Параллельное ускорение для каждого типа расписания при различных N

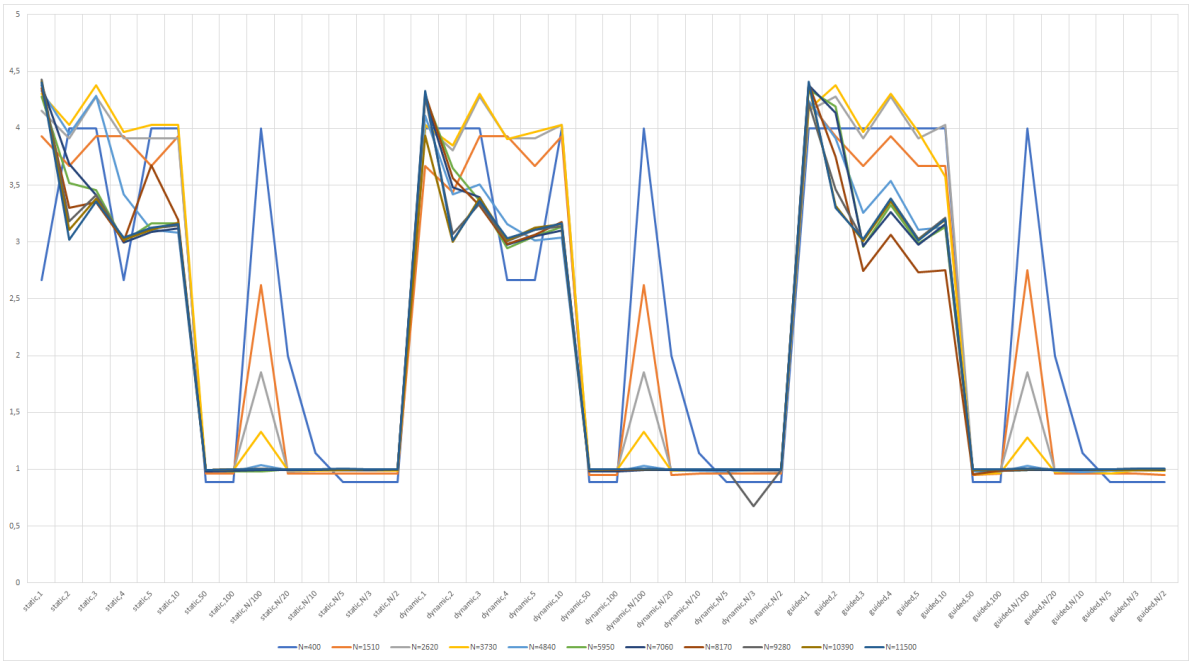


Рис. 11: Параллельное ускорение для каждого N при различных типах расписания

2.3.4 Загрузка процессора

Для подтверждения распараллеливания программы я замерил нагрузку ядер процессора при её выполнении. Из 12 ядер тестового стенда (6 физических + 6 логических) виртуальная машина, на которой проводились эксперименты, имела доступ только к 6 физическим. Их загрузка хорошо видна на следующем графике:

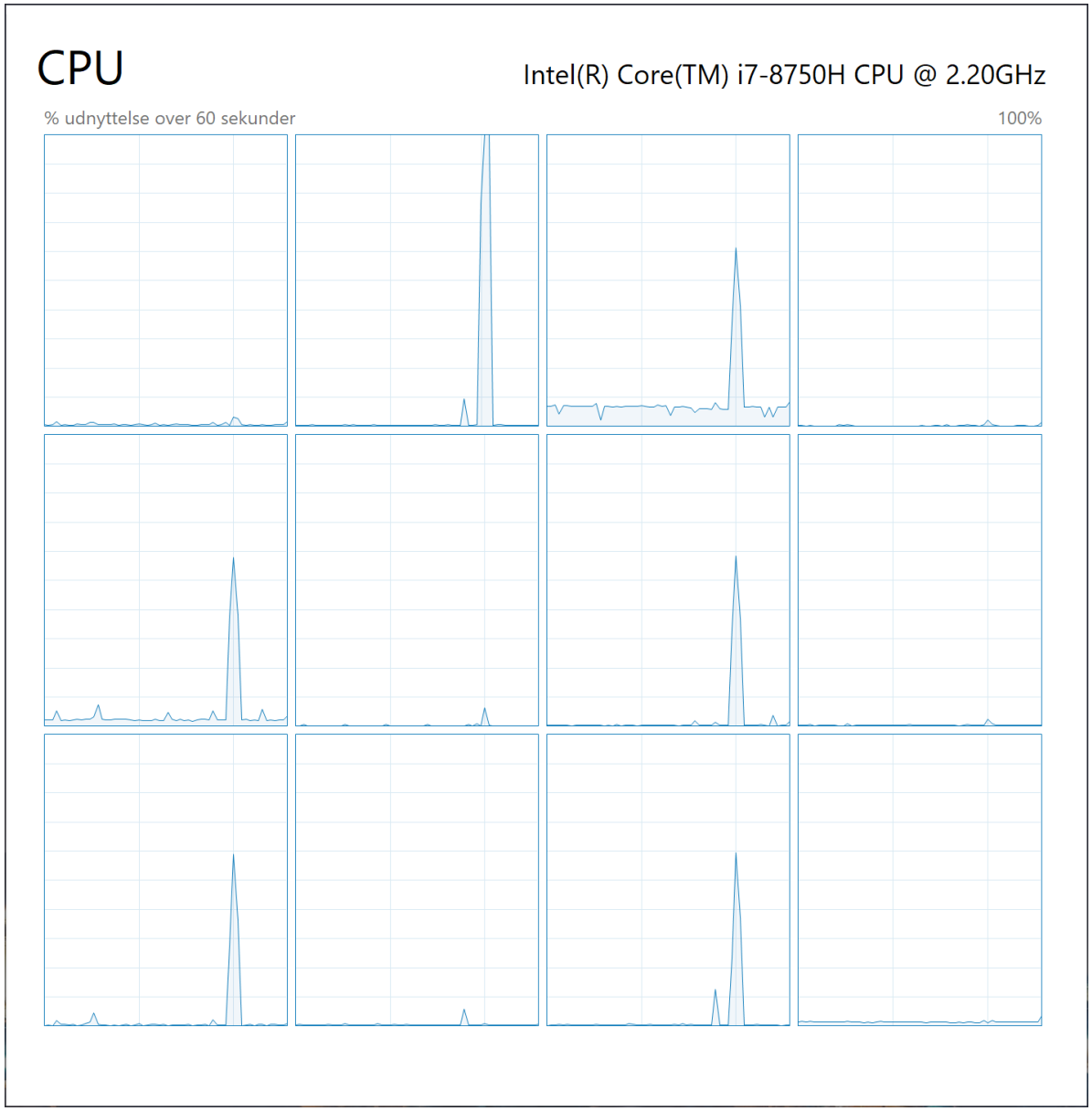


Рис. 12: График загрузки ядер процессора при N=11500

2.3.5 Накладные расходы

Для того, чтобы найти значения N, при которых накладные расходы на параллелизацию превышают выигрыш от распараллеливания, я постоил графики параллельного ускорения для точек $N < N_1$ при различных типах расписания. Прошлые эксперименты показали, что самыми эффективными являются маленькие значиния параметра chunk_size, поэтому для этого эксперимента были выбраны значения chunk_size = 1 и 2. Результат может быть виден на следующих графиках:

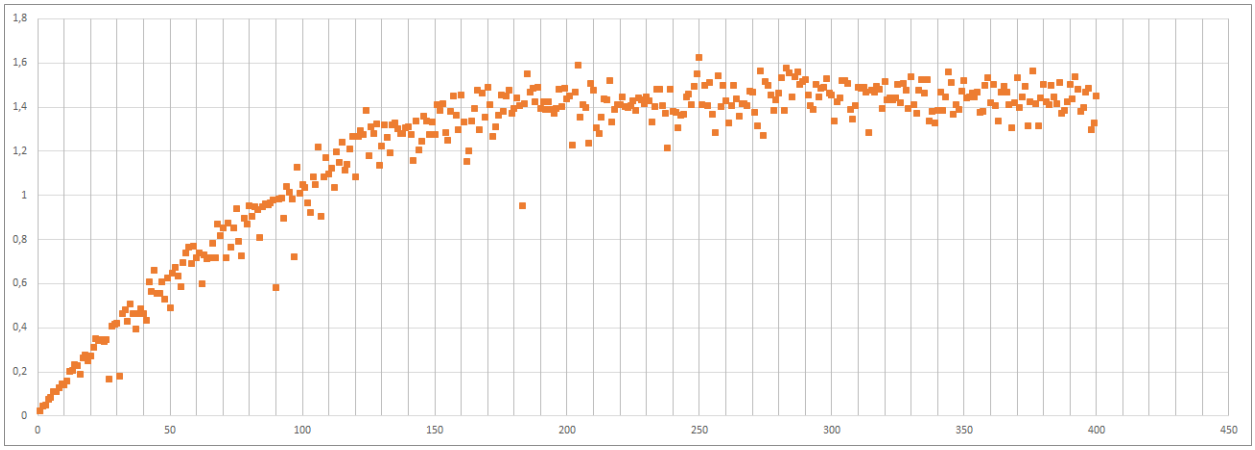


Рис. 13: Параллельное ускорение для расписания static, chink_size = 1

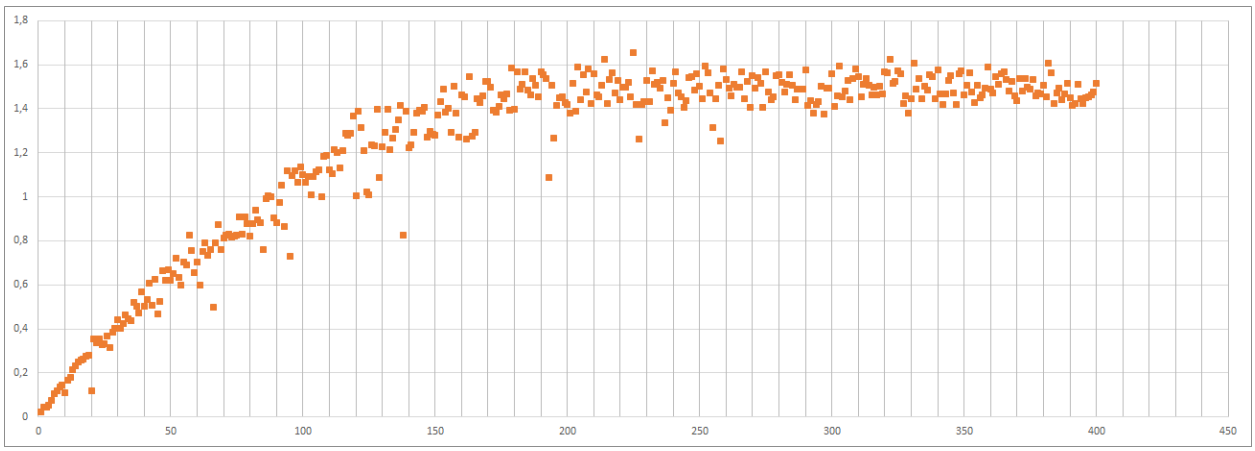


Рис. 14: Параллельное ускорение для расписания static, chink_size = 2

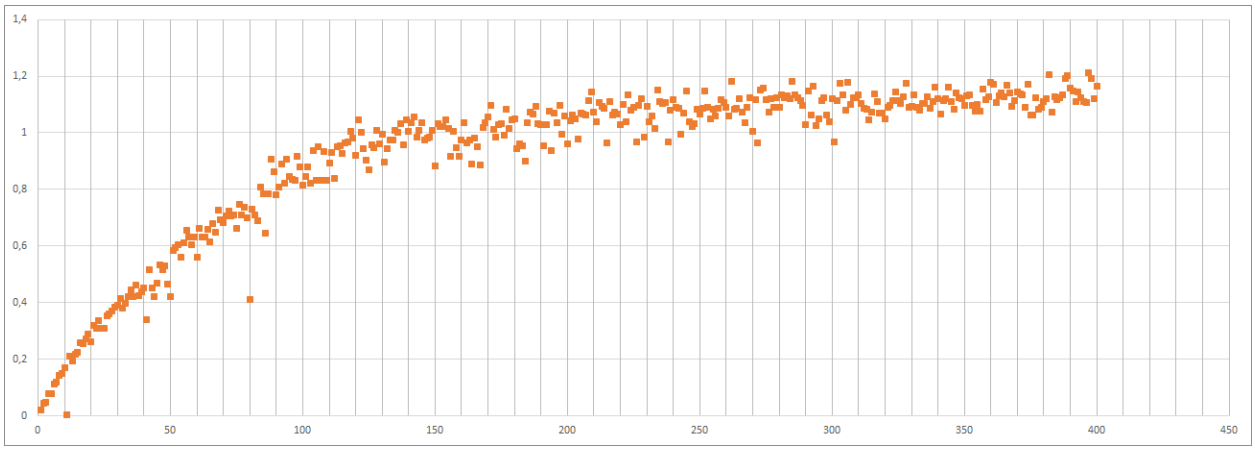


Рис. 15: Параллельное ускорение для расписания dynamic, chink_size = 1

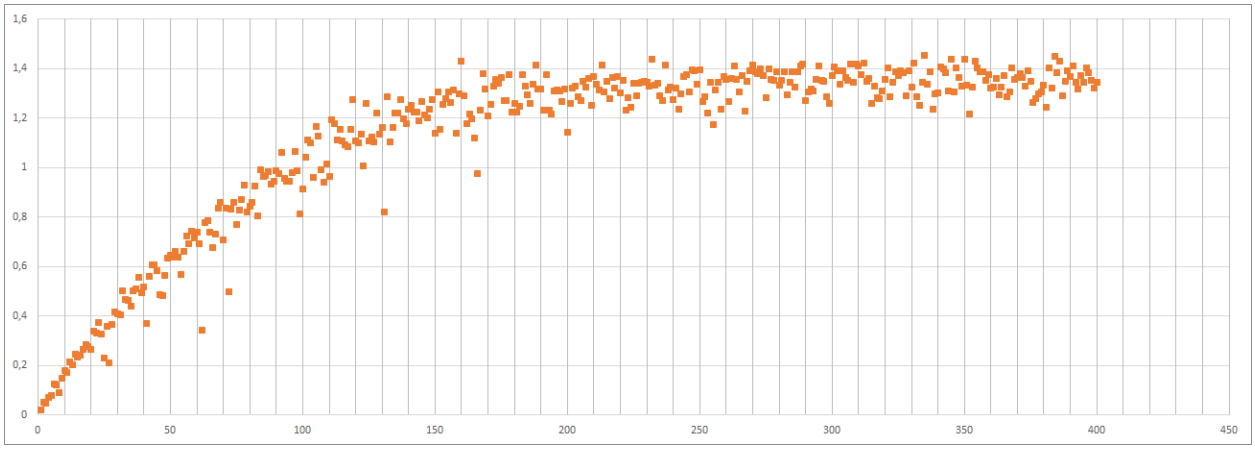


Рис. 16: Параллельное ускорение для расписания dynamic, chink_size = 2

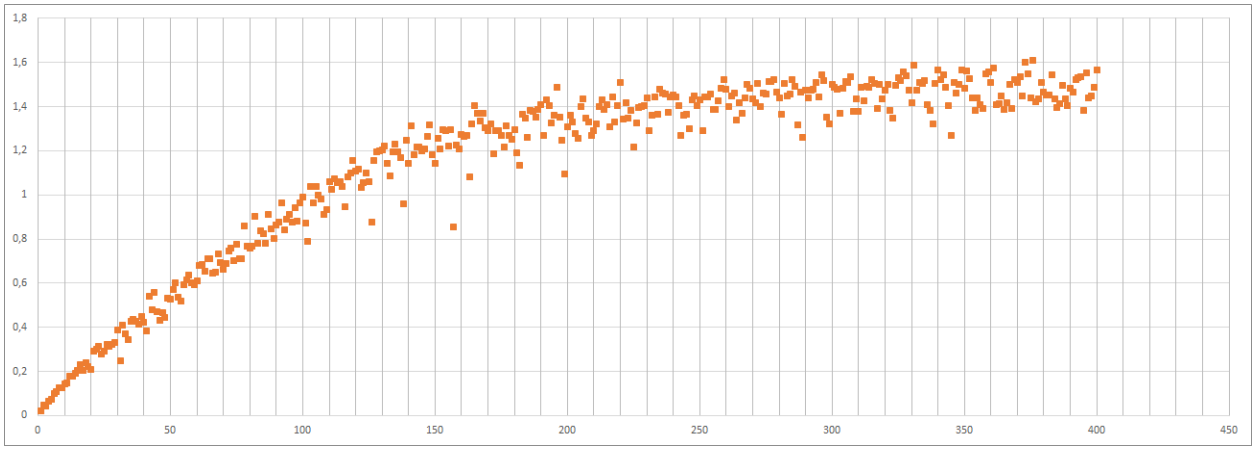


Рис. 17: Параллельное ускорение для расписания guided, chink_size = 1

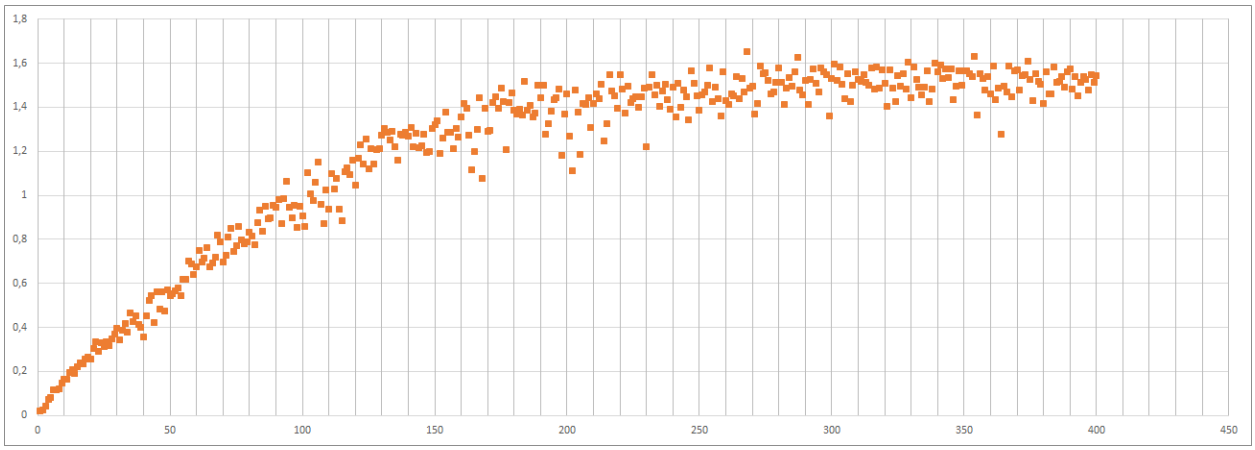


Рис. 18: Параллельное ускорение для расписания guided, chink_size = 2

Можно заметить, что для всех типов расписания при значениях N, где $N < 100$, накладные расходы на параллелизацию превышают выигрыш от распараллеливания, что происходит, когда значение параллельного ускорения меньше единицы.

3 Вывод

На основании данного эксперимента можно сделать несколько выводов:

- При отсутствии параллелизации внешнего цикла выгода от параллелизации падает с возрастанием числа N. Это происходит из-за того, что сортировка, обладающая сложностью $O(N^2)$, не была распараллелена, и параллелизация других частей программы со сложностью $O(N)$ не приносит видимых улучшений.
- При наличии параллелизации внешнего цикла наилучший результат показывает расписание static и guided с небольшими значениями chunk_size.
- При значении $N < 100$ накладные расходы на параллелизацию превышают выигрыш от параллелизации при всех типах расписания.

4 Приложения

4.1 Исходный код программы lab3.c

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

#define A 700 //
#define NUMBER_OF_ITERATIONS 50

long double random_on_interval(long double min, long double max, unsigned int *seed) {
    return (long double) ((rand_r(seed) % (int)(max + 1 - min)) + min);
}

unsigned int make_seed(int i, int j) {
    return 9572 + 234*i + 456*j;
}

int main(int argc, char* argv[]) {
    struct timeval T1, T2;
    int N = atoi(argv[1]);

    long double M1[N];
    long double M2[N/2];
    long double M2_shifted_copy[N/2];

    double X = 1;

    gettimeofday(&T1, NULL);

    int i, j = 0;
    //#pragma omp parallel for default(none) private(i, j, M1, M2, M2_shifted_copy) shared(N) schedule(runtime) reduction(+:X)
    for (i=0; i<NUMBER_OF_ITERATIONS; i++) {
        /******************** GENERATE ********************/
        unsigned int seed;

        #pragma omp parallel for default(none) private(j, seed) shared(M1, N, i) schedule(runtime)
        for (j = 0; j < N; j++) {
            seed = make_seed(i, j);
            M1[j] = random_on_interval(0, A, &seed);
        }

        #pragma omp parallel for default(none) private(j, seed) shared(M2, N, M2_shifted_copy, i) schedule(runtime)
        for (j = 0; j < N/2; j++) {
            seed = make_seed(i, j);
            M2[j] = random_on_interval(A, A*10, &seed);
        }
        M2_shifted_copy[0] = 0;
        memcpy(&M2_shifted_copy[1], M2, sizeof(long double) * ((N/2)-1));

        /******************** MAP ************************/
        #pragma omp parallel for default(none) private(j) shared(M1, N) schedule(runtime)
        for (j = 0; j < N; j++) {
```

```
53              // operation #1, remember to convert to radians
54              M1[j] = pow(sinhl((M1[j] * M_PI) / 180.0), 2);
55          }
56
57          #pragma omp parallel for default(none) private(j) shared(M2,
                M2_shifted_copy, N) schedule(runtime)
58          for (j = 0; j < N/2; j++) {
59              // operation #3
60              M2[j] = fabs(tanl(M2[j] + M2_shifted_copy[j]));
61          }
62
63          /******************** MERGE ************************/
64          #pragma omp parallel for default(none) private(j) shared(M1, M2, N
                ) schedule(runtime)
65          for (j = 0; j < N/2; j++) {
66              // operation #1
67              M2[j] = pow(M1[j], M2[j]);
68          }
69
70          /******************** SORT ************************/
71          int k, l, min_k;
72          for (k = 0; k < (N/2)-1; k++)
73          {
74              min_k = k;
75              for (l = k+1; l < N/2; l++)  {
76                  if (M2[l] < M2[min_k]) {
77                      min_k = l;
78                  }
79              }
80              if (min_k != k) {
81                  long double temp = M2[min_k];
82                  M2[min_k] = M2[k];
83                  M2[k] = temp;
84              }
85          }
86
87          /******************** REDUCE ************************/
88          int min_index;
89          for (min_index = 0; M2[min_index] <= 0; min_index++) {}
90          long double min = M2[min_index];
91
92          #pragma omp parallel for default(none) private(j) shared(M2, min,
                N) reduction(+:X) schedule(runtime)
93          for (j = 0; j < N/2; j++) {
94              if (isfinite(M2[j]) && (int)(M2[j] / min) % 2 == 0) {
95                  // remember to convert to radians
96                  X += sinl((M2[j] * M_PI) / 180.0);
97              }
98          }
99      }
100
101     gettimeofday(&T2, NULL);
102     long delta_ms = 1000 * (T2.tv_sec - T1.tv_sec) + (T2.tv_usec - T1.
            tv_usec) / 1000;
103     printf("%d\n", N);
104     printf("%ld\n", delta_ms);
105     printf("%.5f\n", X);
106
107     return 0;
108 }
```

4.2 Скрипт для компиляции программы lab3.c

```
gcc -O3 -Wall -lm -o lab3-seq lab3.c
gcc -O3 -Wall -Werror -fopenmp -lm -o lab3 lab3.c
```

4.3 Скрипт для запуска программы lab3.c

```
#!/bin/bash
set -eu

declare -a N_set=("400" "1510" "2620" "3730" "4840" "5950" "7060" "8170" "9280" "10390" "11500")
declare -a schedule_set=(static dynamic guided)
TEMPLATE="lab3_template.csv"
OUTPUT="lab3_output.csv"
N_LINE=1
TIME_LINE_NUM_START=2
X_LINE_NUM_START=46
EXECUTABLE_SEQ="lab3-seq"
EXECUTABLE_OMP="lab3"

NUM_OF_TRIES=3

function add_to_line() {
  FILE_NAME="${1}"
  LINE_NUM="${2}"
  TEXT="${3}"
  sed -e "${LINE_NUM}s/\$/${TEXT};/" -i "${FILE_NAME}"
}

function execute() {
    EXECUTABLE="${1}"
    N="${2}"
    TIME_LINE_NUM="${3}"
    X_LINE_NUM="${4}"

    X=$(./${EXECUTABLE} ${N})
    y=(${X//$'\n'/ })
    min_time=${y[1]}
    for (( meh=2; meh<=${NUM_OF_TRIES}; meh++ ))
    do
      X=$(./${EXECUTABLE} ${N})
      y=(${X//$'\n'/ })
      if [ "${y[1]}" -lt "${min_time}" ]
      then
        min_time=${y[1]}
      fi
    done
    add_to_line ${OUTPUT} ${time_line_num} ${min_time}
    add_to_line ${OUTPUT} ${x_line_num} $(sed 's/\./,/g' <<< ${y[2]})
}

echo -e ";;" > ${OUTPUT}
for (( meh=1; meh<=200; meh++ ))
do
    echo -e ";" >> ${OUTPUT}
done

for N in "${N_set[@]}"
do
```

```
53      add_to_line ${OUTPUT} ${N_LINE} ${N}
54      time_line_num=${TIME_LINE_NUM_START}
55      x_line_num=${X_LINE_NUM_START}
56      if [ "${N}" -eq "${N_set[0]}" ]
57      then
58          add_to_line ${OUTPUT} ${time_line_num} "seq"
59      fi
60      execute ${EXECUTABLE_SEQ} ${N} ${time_line_num} ${x_line_num}
61      ((time_line_num=time_line_num+1))
62      ((x_line_num=x_line_num+1))
63
64      for schedule in "${schedule_set[@]}"
65      do
66      for chunk in 1 2 3 4 5 10 50 100 $((${N}/100)) $((${N}/20)) $((${N
            }/10)) $((${N}/5)) $((${N}/3)) $((${N}/2))
67        do
68          export OMP_SCHEDULE="${schedule},${chunk}"
69          if [ "${N}" -eq "${N_set[0]}" ]
70          then
71              add_to_line ${OUTPUT} ${time_line_num} ${OMP_SCHEDULE}
72          fi
73          execute ${EXECUTABLE_OMP} ${N} ${time_line_num} ${x_line_num}
74          ((time_line_num=time_line_num+1))
75          ((x_line_num=x_line_num+1))
76        done
77      done
78  done
79
80  exit 0
```