ПрИб-181	Лабораторная работа №5	Зачёт
Кащенко В. А.	Распараллеливание циклов в OpenMP.	

Цель работы: Изучить методы распараллеливания циклов.

Теоретические сведения:
(Конспект теоретических сведений написан в тетради)

Задание 6, программа «Сумма чисел»:
Для распараллеливания циклов используется директива #omp parallel for
Исходный код (k — количество нитей, n — количество чисел ряда 1,2,3...∞):

```
#include <stdio.h>
#include <omp.h>
#include <iostream>
#include <stdlib.h>
#include <iomanip>

using namespace std;

// Хранение результатов сложения
int sequental_res, parallel_res;

double time(bool parallel, int k, int N)
{
	// Для замеров скорости
	double start_time, end_time;

	// объявим переменные результатов сложения
	int res = 0;
	int res_omp = 0;
	// последовательно сложим
	start_time = omp_get_wtime();

	for (int i = 1; i <= N; ++i)
		res += i;
	end_time = omp_get_wtime();

	sequental_res = res;

	// параллельно умножим
	start_time = omp_get_wtime();
	// используем parallel for
	#pragma omp parallel for num_threads(k) if(parallel)
	for (int i = 1; i <= N; ++i)
		res_omp += i;
	end_time = omp_get_wtime();
	parallel_res = res_omp;

	return (end_time-start_time) * 1'000'000 ; // мкс
```

```
}

// вывод на экран времени
void print_time(int n, int k)
{
	// func format is (bool parallel, int k, int N)
	// Time table
	cout << left << fixed << setprecision(3)
	<< setw(10) << n << setw(17) << time(0, k, n) << setw(18)
	<< time(0, k, n) << setw(18) << sequental_res << setw(15) << parallel_res <<
	endl;
}

// точка входа
int main(int argc, char* argv[])
{
	cout << "Кащенко В. А. ПрИб-181\n\"Сложение чисел\" (параллельная версия)\
	n\n4 потока:\n" <<
	setw(10) << "Кол-во " << setw(20) << "Послед-но, мкс " << setw(20) <<
	"Паралл-но, мкс " << setw(22) <<  " Р-ат послед-но " << setw(15) << " Р-ат
	паралл-но " << endl;

	// func format is (int k, int n)
	print_time(50, 4);
	print_time(100, 4);
	print_time(500, 4);
	print_time(1000, 4);
	print_time(5000, 4);
	print_time(15000, 4);
	print_time(30000, 4);
	print_time(50000, 4);
	print_time(100000, 4);
	print_time(1000000, 4);

	cout << "\n6 потоков:\n" <<  setw(10) << "Кол-во " << setw(20) << "Послед-но,
	мкс " << setw(20) << "Паралл-но, мкс " << setw(22) << " Р-ат послед-но " <<
	setw(15) << " Р-ат паралл-но " << endl;

	print_time(50, 6);
	print_time(100, 6);
	print_time(500, 6);
	print_time(1000, 6);
	print_time(5000, 6);
	print_time(15000, 6);
	print_time(30000, 6);
	print_time(50000, 6);
	print_time(100000, 6);
	print_time(1000000, 6);
```

```
    cout << "\n12 потоков:\n" << setw(10) << "Кол-во " << setw(20) << "Послед-
    но, мкс " << setw(20) << "Паралл-но, мкс " << setw(22) << " Р-ат послед-но "
    << setw(15) << " Р-ат паралл-но " << endl;

    print_time(50, 12);
    print_time(100, 12);
    print_time(500, 12);
    print_time(1000, 12);
    print_time(5000, 12);
    print_time(15000, 12);
    print_time(30000, 12);
    print_time(50000, 12);
    print_time(100000, 12);
    print_time(1000000, 12);
}
```

Работа программы:

```
4 потока:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        3.121            0.904             1275              1275
100       0.971            0.913             5050              5050
500       1.979            1.903             125250            125250
1000      3.412            3.313             500500            500500
5000      16.270           15.060            12502500          12502500
15000     47.208           41.384            112507500         112507500
30000     73.819           84.564            450015000         450015000
50000     126.400          138.152           1250025000        1250025000
100000    247.034          279.071           705082704         705082704
1000000   1762.593         2623.887          1784293664        1784293664

6 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.949            0.567             1275              1275
100       0.660            0.617             5050              5050
500       1.534            22.815            125250            125250
1000      2.433            2.551             500500            500500
5000      9.474            9.457             12502500          12502500
15000     48.582           27.454            112507500         112507500
30000     73.553           53.228            450015000         450015000
50000     87.788           88.944            1250025000        1250025000
100000    180.551          174.335           705082704         705082704
1000000   1714.875         1759.825          1784293664        1784293664

12 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.548            0.533             1275              1275
100       0.671            0.634             5050              5050
500       1.376            1.422             125250            125250
1000      2.342            2.470             500500            500500
5000      10.007           10.289            12502500          12502500
15000     32.915           28.600            112507500         112507500
30000     69.572           52.146            450015000         450015000
50000     101.424          86.666            1250025000        1250025000
100000    172.666          175.831           705082704         705082704
1000000   1961.953         1767.654          1784293664        1784293664
```

В общем случае получилось, что самая быстрая реализация оказалась на 6 нитях (меньше не хватило быстродействия нитей, а с большим количеством нити просто теряли свое быстродействие на организацию параллелизма). А заметна разница, по сравнению с последовательной версией, на двенадцати потоках.
Выигрыш по сравнению с последовательным выполнением виден в основном на бОльшем количестве нитей и на бОльших суммах.

Задание 7, параметр schedule:
Для распараллеливания циклов используется директива #omp parallel for
Для сравнения работы параметр schedule изменялся для замеров.
Блокировка k и N не в вводилась для получения более полной картины.
Исходный код (k — количество нитей, n — количество чисел ряда 1,2,3...∞):

```
#include <stdio.h>
#include <omp.h>
#include <iostream>
#include <stdlib.h>
#include <iomanip>

using namespace std;

// Хранение результатов сложения
int sequental_res = 0, parallel_res = 0;

double time(bool parallel, int k, int n)
{
// Для замеров скорости
double start_time, end_time;

// объявим переменные результатов сложения
int res = 0;
int res_omp = 0;
// последовательно сложим
start_time = omp_get_wtime();

for (int i = 1; i <= n; ++i)
res += i;
end_time = omp_get_wtime();

sequental_res = res;

// параллельно умножим
start_time = omp_get_wtime();
// используем parallel for
#pragma omp parallel for num_threads(k) reduction(+: res_omp) if(parallel)
//#pragma omp parallel for schedule (static, 1) num_threads(k) reduction(+: res_omp)
if(parallel)
```

```
//#pragma omp parallel for schedule (static, 2) num_threads(k) reduction(+: res_omp)
if(parallel)
//#pragma omp parallel for schedule (dynamic) num_threads(k) reduction(+: res_omp)
if(parallel)
//#pragma omp parallel for schedule (dynamic, 2) num_threads(k) reduction(+: res_omp)
if(parallel)
//#pragma omp parallel for schedule (guided) num_threads(k) reduction(+: res_omp)
if(parallel)
//#pragma omp parallel for schedule (guided, 2) num_threads(k) reduction(+: res_omp)
if(parallel)
//#pragma omp parallel for schedule (static) num_threads(k) reduction(+: res_omp)
if(parallel)
for (int i = 1; i <= n; ++i)
res_omp += i;
end_time = omp_get_wtime();
parallel_res = res_omp;

return (end_time-start_time) * 1'000'000 ; // мкс
}

// вывод на экран времени
void print_time(int n, int k)
{
// func format is (bool parallel, int k, int N)
// Time table
cout << left << fixed << setprecision(3)
<< setw(10) << n << setw(17) << time(0, k, n) << setw(18)
<< time(1, k, n) << setw(18) << sequental_res << setw(15) << parallel_res << endl;
}

// точка входа
int main(int argc, char* argv[])
{
cout << "Кащенко В. А. ПрИб-181\n\"Сложение чисел\" (параллельная версия)\n\n4
потока:\n" <<
setw(10) << "Кол-во " << setw(20) << "Послед-но, мкс " << setw(20) << "Паралл-но,
мкс " << setw(22) <<
" Р-ат послед-но " << setw(15) << " Р-ат паралл-но " << endl;
int threads = 4;
print_time(50, threads);
print_time(100, threads);
print_time(500, threads);
print_time(1000, threads);
print_time(5000, threads);
print_time(15000, threads);
print_time(30000, threads);
print_time(50000, threads);
print_time(100000, threads);
print_time(1000000, threads);

cout << "\n6 потоков:\n" <<
```

```
setw(10) << "Кол-во " << setw(20) << "Послед-но, мкс " << setw(20) << "Паралл-но,
мкс " << setw(22) <<
" Р-ат послед-но " << setw(15) << " Р-ат паралл-но " << endl;
threads = 6;
print_time(50, threads);
print_time(100, threads);
print_time(500, threads);
print_time(1000, threads);
print_time(5000, threads);
print_time(15000, threads);
print_time(30000, threads);
print_time(50000, threads);
print_time(100000, threads);
print_time(1000000, threads);

cout << "\n12 потоков:\n" <<
setw(10) << "Кол-во " << setw(20) << "Послед-но, мкс " << setw(20) << "Паралл-но,
мкс " << setw(22) <<
" Р-ат послед-но " << setw(15) << " Р-ат паралл-но " << endl;

threads = 12;
print_time(50, threads);
print_time(100, threads);
print_time(500, threads);
print_time(1000, threads);
print_time(5000, threads);
print_time(15000, threads);
print_time(30000, threads);
print_time(50000, threads);
print_time(100000, threads);
print_time(1000000, threads);
}
```

Работа программы (без планирования):

```
4 потока:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        9.119            319.267           1275              1275
100       6.014            7.696             5050              5050
500       8.652            7.571             125250            125250
1000      13.920           8.279             500500            500500
5000      59.922           24.758            12502500          12502500
15000     54.195           10.687            112507500         112507500
30000     90.462           19.721            450015000         450015000
50000     123.960          32.313            1250025000        1250025000
100000    250.308          63.744            705082704         705082704
1000000   2758.031         701.940           1784293664        1784293664

6 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.790            110.537           1275              1275
100       1.314            1.496             5050              5050
500       1.648            1.483             125250            125250
1000      2.482            1.755             500500            500500
5000      9.620            3.117             12502500          12502500
15000     28.516           7.214             112507500         112507500
30000     54.735           13.595            450015000         450015000
50000     91.734           22.188            1250025000        1250025000
100000    186.865          43.814            705082704         705082704
1000000   1814.598         429.007           1784293664        1784293664

12 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        1.007            117.047           1275              1275
100       1.143            2.724             5050              5050
500       1.629            1.978             125250            125250
1000      2.520            1.744             500500            500500
5000      9.539            2.772             12502500          12502500
15000     27.297           4.556             112507500         112507500
30000     54.065           7.902             450015000         450015000
50000     90.928           12.204            1250025000        1250025000
100000    184.589          23.144            705082704         705082704
1000000   1805.201         215.529           1784293664        1784293664
```

Данные результаты будем считать «отправной точкой исследования», с ним будем сравнивать директивы планирования. Здесь видны стабильные «просадки» скорости на маленьком числе элементов.

Работа программы (schedule (static)):

```
4 потока:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        9.479            3.463             1275              1275
100       3.412            3.211             5050              5050
500       24.640           7.995             125250            125250
1000      12.551           12.261            500500            500500
5000      50.240           48.986            12502500          12502500
15000     135.862          147.833           112507500         112507500
30000     57.814           61.973            450015000         450015000
50000     90.471           99.380            1250025000        1250025000
100000    171.989          186.521           705082704         705082704
1000000   2521.786         1714.966          1784293664        1784293664

6 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.570            0.554             1275              1275
100       0.645            0.630             5050              5050
500       1.511            1.409             125250            125250
1000      2.451            2.326             500500            500500
5000      10.053           11.387            12502500          12502500
15000     61.253           73.765            112507500         112507500
30000     79.077           57.783            450015000         450015000
50000     87.362           90.162            1250025000        1250025000
100000    175.305          175.859           705082704         705082704
1000000   1806.613         1743.449          1784293664        1784293664

12 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.566            0.539             1275              1275
100       0.630            0.644             5050              5050
500       1.522            1.470             125250            125250
1000      2.746            2.295             500500            500500
5000      10.931           9.576             12502500          12502500
15000     72.809           29.941            112507500         112507500
30000     77.943           60.388            450015000         450015000
50000     97.822           102.593           1250025000        1250025000
100000    195.234          287.050           705082704         705082704
1000000   2017.476         1971.241          1784293664        1784293664
```

Для статического планирования свойственно, что во время выполнения OpenMP гарантирует, что если есть два отдельных цикла с одинаковым количеством итераций и нужно выполнить их с тем же количеством потоков, то каждый поток получит точно такой же диапазон итераций в обеих параллельных областях, используя статическое планирование.
При сравнении с результатом без планирования видно, что общем случае программа стала медленнее, а значит, данная директива не подходит для этой задачи, однако, «просадок» скорости нет, как в случае без планирования.

Работа программы (schedule (static, 1)):

```
4 потока:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        10.027           3.719             1275              1275
100       3.259            2.991             5050              5050
500       5.765            5.616             125250            125250
1000      8.988            9.161             500500            500500
5000      36.037           35.631            12502500          12502500
15000     102.716          102.367           112507500         112507500
30000     170.785          39.406            450015000         450015000
50000     65.346           65.402            1250025000        1250025000
100000    130.189          129.909           705082704         705082704
1000000   1291.516         1295.518          1784293664        1784293664

6 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.573            0.508             1275              1275
100       0.622            0.575             5050              5050
500       1.131            1.130             125250            125250
1000      1.794            1.784             500500            500500
5000      6.976            6.973             12502500          12502500
15000     20.620           20.550            112507500         112507500
30000     40.477           40.387            450015000         450015000
50000     66.994           66.854            1250025000        1250025000
100000    134.162          136.418           705082704         705082704
1000000   1375.369         1326.830          1784293664        1784293664

12 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.584            0.524             1275              1275
100       0.621            0.589             5050              5050
500       1.193            1.168             125250            125250
1000      1.821            1.827             500500            500500
5000      7.209            11.813            12502500          12502500
15000     20.668           20.572            112507500         112507500
30000     40.431           40.413            450015000         450015000
50000     67.150           66.932            1250025000        1250025000
100000    133.437          133.260           705082704         705082704
1000000   1468.155         1397.260          1784293664        1784293664
```

Данный код работает быстрее на маленьком числе итераций, но медленнее на большом. Его целесообразно использовать при небольшом числе итераций.

«Просадок» скорости нет, как в случае без планирования.

Работа программы (schedule (static, 2)):

```
4 потока:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        9.183            3.282             1275              1275
100       3.179            3.033             5050              5050
500       5.766            5.701             125250            125250
1000      9.135            8.912             500500            500500
5000      35.890           35.598            12502500          12502500
15000     102.390          102.108           112507500         112507500
30000     201.521          39.402            450015000         450015000
50000     65.467           65.333            1250025000        1250025000
100000    130.059          130.155           705082704         705082704
1000000   1295.133         1296.211          1784293664        1784293664

6 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.614            0.512             1275              1275
100       0.627            0.572             5050              5050
500       1.150            1.154             125250            125250
1000      1.829            20.233            500500            500500
5000      7.162            7.194             12502500          12502500
15000     20.494           20.505            112507500         112507500
30000     40.499           40.483            450015000         450015000
50000     67.005           80.259            1250025000        1250025000
100000    133.505          133.117           705082704         705082704
1000000   1372.790         1330.686          1784293664        1784293664

12 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.615            0.524             1275              1275
100       0.609            0.583             5050              5050
500       1.179            1.176             125250            125250
1000      1.863            1.831             500500            500500
5000      7.191            7.163             12502500          12502500
15000     20.632           20.573            112507500         112507500
30000     40.489           40.559            450015000         450015000
50000     66.848           66.898            1250025000        1250025000
100000    133.230          133.694           705082704         705082704
1000000   1334.540         1340.975          1784293664        1784293664
```

Данный код показывает выигрыш по сравнению со (static, 1), но, аналогично, его целесообразно использовать при небольшом числе итераций. Опять таки, «просадок» скорости на малом числе итераций нет.

Работа программы (schedule (dynamic)):

```
4 потока:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        13.869           5.861             1275              1275
100       7.739            7.402             5050              5050
500       25.973           26.075            125250            125250
1000      49.444           49.287            500500            500500
5000      235.743          235.522           12502500          12502500
15000     211.094          136.970           112507500         112507500
30000     273.350          273.304           450015000         450015000
50000     464.092          464.115           1250025000        1250025000
100000    909.819          909.378           705082704         705082704
1000000   10147.246        10141.673         1784293664        1784293664

6 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        2.278            1.052             1275              1275
100       1.613            1.614             5050              5050
500       5.706            5.672             125250            125250
1000      10.784           10.763            500500            500500
5000      51.342           51.227            12502500          12502500
15000     152.975          158.831           112507500         112507500
30000     304.454          302.031           450015000         450015000
50000     455.164          455.045           1250025000        1250025000
100000    909.364          967.845           705082704         705082704
1000000   9282.162         9396.763          1784293664        1784293664

12 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        2.171            0.970             1275              1275
100       1.470            1.453             5050              5050
500       5.137            5.082             125250            125250
1000      9.643            9.757             500500            500500
5000      68.725           108.209           12502500          12502500
15000     176.441          137.241           112507500         112507500
30000     274.088          276.689           450015000         450015000
50000     541.224          617.996           1250025000        1250025000
100000    916.851          911.406           705082704         705082704
1000000   9687.267         9356.350          1784293664        1784293664
```

Здесь меняется метод планирования на динамическое.

Документы Intel описывают планирование `dynamic` :

> Используйте внутреннюю рабочую очередь, чтобы дать каждому потоку блок итераций цикла размером с кусок. Когда поток завершен, он извлекает следующий блок итераций цикла из верхней части рабочей очереди. По умолчанию размер блока равен 1. Будьте осторожны при использовании этого типа планирования из-за дополнительных накладных расходов.

На небольшом числе итераций можно заметить прирост скорости, но на большем числе, получается, видны эти дополнительные накладные расходы.

Работа программы (schedule (dynamic, 2)):

```
4 потока:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        12.056           4.931             1275              1275
100       5.542            5.417             5050              5050
500       16.817           17.018            125250            125250
1000      31.248           31.370            500500            500500
5000      146.599          146.845           12502500          12502500
15000     433.685          193.885           112507500         112507500
30000     165.282          165.665           450015000         450015000
50000     281.247          281.299           1250025000        1250025000
100000    564.854          561.756           705082704         705082704
1000000   5952.103         5754.719          1784293664        1784293664

6 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.910            0.793             1275              1275
100       1.080            1.068             5050              5050
500       3.388            3.410             125250            125250
1000      6.311            6.260             500500            500500
5000      29.455           29.421            12502500          12502500
15000     87.001           87.233            112507500         112507500
30000     173.625          173.598           450015000         450015000
50000     288.864          290.269           1250025000        1250025000
100000    585.262          583.412           705082704         705082704
1000000   5815.776         5707.165          1784293664        1784293664

12 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.799            0.764             1275              1275
100       1.042            1.035             5050              5050
500       3.271            3.255             125250            125250
1000      6.061            6.023             500500            500500
5000      49.329           28.113            12502500          12502500
15000     122.167          83.098            112507500         112507500
30000     165.545          165.472           450015000         450015000
50000     275.527          275.351           1250025000        1250025000
100000    550.316          561.661           705082704         705082704
1000000   5636.654         5679.040          1784293664        1784293664
```

Эта версия dynamic работает немного лучше, но совершенно аналогично видны накладные расходы с увеличением числа итераций.

Работа программы (schedule (guided)):

```
4 потока:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        10.843           3.927             1275              1275
100       3.828            20.549            5050              5050
500       7.531            6.912             125250            125250
1000      11.775           13.048            500500            500500
5000      46.692           46.879            12502500          12502500
15000     144.569          149.805           112507500         112507500
30000     187.708          56.651            450015000         450015000
50000     87.817           88.606            1250025000        1250025000
100000    177.999          175.563           705082704         705082704
1000000   1948.015         1715.972          1784293664        1784293664

6 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.653            0.569             1275              1275
100       0.697            0.670             5050              5050
500       1.496            1.490             125250            125250
1000      2.422            2.390             500500            500500
5000      11.515           29.623            12502500          12502500
15000     65.458           28.732            112507500         112507500
30000     52.259           56.284            450015000         450015000
50000     143.293          97.242            1250025000        1250025000
100000    189.396          188.713           705082704         705082704
1000000   1795.415         2031.703          1784293664        1784293664

12 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.623            0.586             1275              1275
100       0.670            0.697             5050              5050
500       1.481            1.536             125250            125250
1000      2.547            2.471             500500            500500
5000      10.440           10.074            12502500          12502500
15000     29.862           30.853            112507500         112507500
30000     60.164           52.089            450015000         450015000
50000     96.865           96.879            1250025000        1250025000
100000    207.402          195.211           705082704         705082704
1000000   2188.785         1991.472          1784293664        1784293664
```

Здесь меняется метод планирования на управляемое.

Он также описывает планирование `guided` :

> Аналогично динамическому планированию, но размер блока начинается с большого и уменьшается, чтобы лучше справляться с дисбалансом нагрузки между итерациями. Необязательный параметр chunk указывает минимальный размер используемого фрагмента. По умолчанию размер блока составляет приблизительно loop_count/number_of_threads.

Получается, размер блока при 12 потоках и самом большом количестве итераций:

$$\sim 83\,333,\,333$$

По сравнению с результатами без планирования «просадок» нет, но в общем случае выполнение оказалось медленнее.

Работа программы (schedule (guided, 2)):

```
4 потока:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        10.943           3.694             1275              1275
100       3.567            3.495             5050              5050
500       7.183            7.415             125250            125250
1000      13.061           12.407            500500            500500
5000      51.749           47.883            12502500          12502500
15000     163.829          136.729           112507500         112507500
30000     267.480          52.085            450015000         450015000
50000     90.036           94.374            1250025000        1250025000
100000    172.542          178.607           705082704         705082704
1000000   2036.831         1719.451          1784293664        1784293664

6 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.631            0.596             1275              1275
100       0.722            0.672             5050              5050
500       1.404            1.437             125250            125250
1000      2.405            2.263             500500            500500
5000      10.936           10.955            12502500          12502500
15000     26.806           26.521            112507500         112507500
30000     52.607           54.820            450015000         450015000
50000     91.599           87.053            1250025000        1250025000
100000    199.096          192.551           705082704         705082704
1000000   1919.694         1908.524          1784293664        1784293664

12 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        0.778            0.595             1275              1275
100       0.672            0.688             5050              5050
500       1.473            1.540             125250            125250
1000      2.699            23.717            500500            500500
5000      9.354            10.417            12502500          12502500
15000     46.646           27.419            112507500         112507500
30000     72.037           58.875            450015000         450015000
50000     88.992           141.074           1250025000        1250025000
100000    200.888          198.753           705082704         705082704
1000000   2144.036         1944.633          1784293664        1784293664
```

Эта версия guided работает быстрее, но в общем случае она всё равно оказалась медленнее случая без планирования, что говорит о неподходящем варианте использования данного планирования.

Задание 8, программа «Число π»:

Здесь количество итераций фиксированное (иначе не дойдём до числа π в процессе вычислений), посмотрим на изменение скорости относительно числа активных потоков.

Исходный код программы:

```
#include <stdio.h>
#include <omp.h>
#include <iostream>
#include <stdlib.h>
#include <iomanip>

using namespace std;

// Хранение результатов
double sequental_res, parallel_res;

// Часть вычисления Pi
double f(double x)
{
	return (4.0 / (1.0 + x * x));
}

double time(bool parallel, int k, int N)
{
	double x = 0, pi = 0;

	// Для замеров скорости
	double start_time, end_time;

	// объявим переменные результатов
	double res = 0;
	double res_omp = 0;
	double dx = 1.0 / (double)N;


	// последовательно сложим
	start_time = omp_get_wtime();
	for (int i = 0; i < N; ++i)
	{
		x = dx * (i + 0.5);
		res += f(x);
	}
	end_time = omp_get_wtime();
	sequental_res = res * dx;

	// параллельно умножим
	if (parallel)
	start_time = omp_get_wtime();
```

```
	// используем parallel for с опцией reduction
	#pragma omp parallel for private(x) shared(dx) num_threads(k) reduction(+:
res_omp) if(parallel)
	for (int i = 0; i < N; ++i)
	{
		x = dx * (i + 0.5);
	res_omp += f(x);
	}

	if (parallel)
		end_time = omp_get_wtime();
	parallel_res = res_omp * dx;

	return end_time-start_time;
}

// вывод на экран времени
void print_time(int n, int k)
{
	// func format is (bool parallel, int k, int N)
	// Time table
	cout << left << fixed << setprecision(8)
	<< setw(11) << n << setw(12) << time(0, k, n) << setw(15)
	<< time(1, k, n) << setw(18) << sequental_res << setw(15) << parallel_res <<
	endl;
}

// точка входа
int main(int argc, char* argv[])
{
	cout << "Кащенко В. А. ПрИб-181\n\"Число Pi\" (параллельная версия)\n\n" <<
	setw(11) << "Кол-во " << setw(20) << " Послед-но " << setw(20) << "Паралл-
	но " << setw(22) <<
	" Р-ат послед-но " << setw(15) << " Р-ат паралл-но " << endl;

	cout << "2 потока:" << endl;
	print_time(1000000000, 2);
	cout << "4 потока:" << endl;
	print_time(1000000000, 4);
	cout << "6 потоков:" << endl;
	print_time(1000000000, 6);
	cout << "8 потоков:" << endl;
	print_time(1000000000, 8);
	cout << "10 потоков:" << endl;
	print_time(1000000000, 10);
	cout << "12 потоков:" << endl;
	print_time(1000000000, 12);
}
```

Работа программы:

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/5/last/true$ g++ task8.cpp -fopenmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/5/last/true$ ./a.out
Кащенко В. А. ПрИб-181
"Число Pi" (параллельная версия)

Кол-во     Послед-но   Паралл-но      Р-ат послед-но    Р-ат паралл-но
2 потока:
1000000000 3.21110720  1.61872415     3.14159265        3.14159265
4 потока:
1000000000 3.33796991  0.78843205     3.14159265        3.14159265
6 потоков:
1000000000 3.19821889  0.58846752     3.14159265        3.14159265
8 потоков:
1000000000 3.19868672  0.68803089     3.14159265        3.14159265
10 потоков:
1000000000 3.20756938  0.66084948     3.14159265        3.14159265
12 потоков:
1000000000 3.20455463  0.56412171     3.14159265        3.14159265
```

Исходя из результатов исследования, видно, что скорость растет при 2 $\rightarrow$ 6 потоках. Потом при 8 $\rightarrow$ 10 потоках она падает, а при 12 выдаёт самый быстрый результат.
Думаю, объяснить такое можно созданием накладных расходов при организации параллелизма на большом количестве потоков. При 12 результат должен быть более быстрым, просто накладные расходы делают своё дело.

Задание 9, программа «Матрица»:

Привычную таблицу не получается создать, так как нужно вручную вводить матрицы, ограничимся замером времени при каждом запуске программы и в качестве входных данных возьмём следующие:

$$\mathbf{C} = \mathbf{A}\cdot\mathbf{B} = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 1 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & 5 & 6 \\ 6 & 5 & 4 \\ 4 & 5 & 6 \end{pmatrix} = \begin{pmatrix} 28 & 30 & 32 \\ 28 & 30 & 32 \\ 28 & 30 & 32 \end{pmatrix}$$

При решении данной задачи нужно учитывать, что:

> Если выполнение любого связанного цикла изменяет какое-либо из значений, используемых для вычисления любого количества итераций, то поведение не определено.

поэтому будем использовать collapse. (в соответствии со спецификацией OpenMP)

Исходный код программы:

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

// матрица
double** new_matrix(int n)
{
	double** matrix;
	matrix = new double*[n];
	for(int i = 0; i < n; ++i)
		matrix[i] = new double[n];
	return matrix;
}

// чтение матрицы с клавиатуры
void read_matrix(double** matrix, int n)
{
	for(int i = 0; i < n; ++i)
		for(int j = 0; j < n; ++j)
			scanf("%lf", &matrix[i][j]);
}

// вывод матрицы
void print_matrix(double** matrix, int n)
{
```

```
        for(int i = 0; i < n; ++i)
        {
                for(int j = 0; j < n; ++j)
                        printf("%lf ", matrix[i][j]);
                printf("\n");
        }
}


// точка входа
int main()
{
        double t1, t2;

        int n, k, intBool;
        printf("0 - последовательно | !0 - параллельно\n");
        scanf("%d", &intBool);

        if (intBool)
        {
                printf("параллельная версия\n");
                printf("потоки\n");
                // количество потоков
                scanf("%d", &k);
        }
        printf("Размер матриц\n");
        // размер матриц
        scanf("%d", &n);

        double** A = new_matrix(n);
        double** B = new_matrix(n);
        double** C = new_matrix(n);

        read_matrix(A, n);
        read_matrix(B, n);

        t1 = omp_get_wtime();

        if (!intBool)
        {
                for (int i = 0; i < n; ++i)
                {
                        for (int j = 0; j < n; ++j)
                        {
                                C[i][j] = 0.0;
                                for (int k = 0; k < n; ++k)
                                        C[i][j] += A[i][k] * B[k][j];
                        }
                }
        }
```

```
	// основной вычислительный блок (с collapse(2) для работы со связанными циклами)
	#pragma omp parallel for shared(A, B, C) num_threads(k) collapse(2) if (intBool)
	for (int i = 0; i < n; ++i)
	{
		for (int j = 0; j < n; ++j)
		{
			C[i][j] = 0.0;
			for (int k = 0; k < n; ++k)
				C[i][j] += A[i][k] * B[k][j];
		}
	}
	t2 = omp_get_wtime();

	print_matrix(C, n);

	printf("Замер времени: %lf\n", t2 - t1);

}
```

Работа программы (замеры):

(последовательная версия)

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/5/last/true$ g++ task9.cpp -fopenmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/5/last/true$ ./a.out
0 - последовательно | !0 - параллельно
0
Размер матриц
3
1
2
3
3
2
1
1
2
3
4
5
6
6
5
4
4
5
6
28.000000 30.000000 32.000000
28.000000 30.000000 32.000000
28.000000 30.000000 32.000000
Замер времени: 0.000014
```

(параллельная версия, 4 потока)

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/5/last/true$ ./a.out
0 - последовательно | !0 - параллельно
1
параллельная версия
потоки
4
Размер матриц
3
1
2
3
3
2
1
1
2
3
4
5
6
6
5
4
4
5
6
28.000000 30.000000 32.000000
28.000000 30.000000 32.000000
28.000000 30.000000 32.000000
Замер времени: 0.000340
```

(параллельная версия, 8 потоков)

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/5/last/true$ ./a.out
0 - последовательно | !0 - параллельно
1
параллельная версия
потоки
8
Размер матриц
3
1
2
3
3
2
1
1
2
3
4
5
6
6
5
4
4
5
6
28.000000 30.000000 32.000000
28.000000 30.000000 32.000000
28.000000 30.000000 32.000000
Замер времени: 0.000700
```

(параллельная версия, 12 потоков)

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/5/last/true$ ./a.out
0 - последовательно | !0 - параллельно
1
параллельная версия
потоки
12
Размер матриц
3
1
2
3
3
2
1
1
2
3
4
5
6
6
5
4
4
5
6
28.000000 30.000000 32.000000
28.000000 30.000000 32.000000
28.000000 30.000000 32.000000
Замер времени: 0.003741
```

Тут видно, что последовательная версия отработала быстрее любой параллельной реализации, причем видно заметное замедление при добавлении числа потоков. Объяснить это можно тем, что:

Во-первых, размер матриц мал для получения пользы от распараллеливания.

Во-вторых, при увеличении количества потоков увеличиваются и накладные расходы на организацию параллелизма.

Исходя из всего этого можно сделать промежуточный вывод: польза от параллельного выполнения будет в случае большого количества данных. Под каждое конкретное количество данных нужно использовать соответствующее число нитей.

Вывод: Изучены методы распараллеливания циклов.