ПрИб-181	Лабораторная работа №4	Зачёт
Кащенко В. А.	Распределение работы	

Цель работы: Изучить распределение работы между имеющимися нитями.

Теоретические сведения:
(Конспект теоретических данных написан в тетради)

Задания (контрольные вопросы):

1. Вопрос: «Могут ли функции omp_get_thread_num() и omp_get_num_threads() вернуть одинаковые значения на нескольких нитях одной параллельной области?»
Ответ: ..._num() не может, так как возвращает номер текущей нити, а ..._threads() только одинаковые значения и могут вывести (общее количество потоков, задействованных в области).

2. Вопрос: «Можно ли распределить между нитями итерации цикла без использования директивы for?»
Ответ: Нет, иначе нити выполнят весь цикл каждая.

3. Вопрос: «Можно ли одной директивой распределить между нитями итерации сразу нескольких циклов?»
Ответ: Нет, на нити распределяется один цикл, но можно сразу запустить выполнение второго с распределением без задержек.

4. Вопрос: «Возможно ли, что при статическом распределении итераций цикла нитям достанется разное количество итераций?»
Ответ: Нет. При статическом распределении итерации распределяются равномерно.

5. Вопрос: «Могут ли при повторном запуске программы итерации распределяемого цикла достаться другим нитям? Если да, то при каких способах распределения итераций?»
Ответ: Да, могут, порядок нитей не определен, при всех способах распределения.

6. Вопрос: «Для чего может быть полезно указывать параметр chunk при способе распределения итераций guided?»
Ответ: Это может быть полезно, если итерации имеет смысл выполнять одной нити несколько раз.

7. Вопрос: «Можно ли реализовать параллельные секции без использования директив sections и section?»
Ответ: Эти директивы специально созданы для этого. Если их не использовать, то и параллельных секций не будет.

8. Вопрос: «Как при выходе из параллельных секций разослать значение некоторой локальной переменной всем нитям, выполняющим данную параллельную область?»
Ответ: Данное действие смысла не имеет. При выходе из параллельных областей значения в нитях не будут использованы.

9. Вопрос: «В каких случаях может пригодиться механизм задач?»
Ответ: Для выделения отдельной независимой задачи. (task) Задача может выполняться немедленно после создания или быть отложенной.

10. Вопрос(задание): «Напишите параллельную программу, реализующую скалярное произведение двух векторов.»
Ответ:

```
#include <stdio.h>
#include <omp.h>
#include <iostream>
#include <vector>
#include <random>
#include <algorithm>
#include <iterator>

using namespace std;

vector<int> randVec(size_t size)
{
	vector<int> v(size);
	// генератор true-random-number
	random_device r;
	// используем лямда-функцию generate()
	// & фиксирует ссылку на локальный объект, чтобы видеть актуальное значение
	generate(v.begin(), v.end(), [&] {return r();});
	return v;
}

int main(int argc, char* argv[])
{
	double start_time, end_time;
	int input = 0;
	cout << "введите размер вектора: ";
	cin >> input;
	// создание векторов
	vector<int> v(randVec(input));
	vector<int> v1(randVec(input));
	// объявим переменные результатов произведения
	int res = 0;
	int res_omp = 0;
	// последовательно умножим
	start_time = omp_get_wtime();
	for (int i = 0; i < input; i++)
	{
		res += v[i] * v1[i];
	}
	end_time = omp_get_wtime();
	// *** Out
	printf("\nПоследовательная версия:\n");
	printf("Время на скалярное произведение векторов = %f\n", end_time - start_time);
	printf("Скалярное произведение (сумма произведений) = %d\n", res);
	// ***
```

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

        // используем parallel for с опцией reduction (суммирование в res_omp)
        #pragma omp parallel for reduction(+:res_omp)
        for (int i = 0; i < input; i++)
        {
                res_omp += v[i] * v1[i];
        }
        end_time = omp_get_wtime();
        // *** Out
        printf("\nПараллельная версия:\n");
        printf("Время на скалярное произведение векторов = %f\n", end_time -
start_time);
        printf("Скалярное произведение (сумма произведений) = %d\n", res_omp);
        // ***
}
```

Работа программы:

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ g++ task1.cpp -
fopenmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
введите размер вектора: 300000

Последовательная версия:
Время на скалярное произведение векторов = 0.001212
Скалярное произведение (сумма произведений) = 1657501034

Параллельная версия:
Время на скалярное произведение векторов = 0.000483
Скалярное произведение (сумма произведений) = 1657501034
```

Как можно заметить, параллельное выполнение операций намного быстрее последовательного.

11. Вопрос(задание): «Напишите параллельную программу, реализующую поиск максимального значения вектора.»

Ответ:

```
#include <stdio.h>
#include <omp.h>
#include <vector>
#include <random>
#include <algorithm>
#include <iterator>
#include <iostream>

using namespace std;

vector<int> randVec(size_t size)
{
	vector<int> v(size);
	// генератор true-random-number
	random_device r;
	// используем лямда-функцию generate()
	// & фиксирует ссылку на локальный объект, чтобы видеть актуальное
значение
	generate(v.begin(), v.end(), [&] {return r();});
	return v;
}

int main(int argc, char* argv[])
{
	double start_time, end_time;
	// фиксируем ввод (подходящее для замера скорости значение)
	int input = 10000000;
	// создание векторов
	vector<int> v(randVec(input));

	//начальные значения векторов
	int count = v[0];
	int count1 = v[0];
	// n - поток, i - итератор
	int n,i;
	// последовательный поиск
	start_time = omp_get_wtime();
	for (i = 1; i < input; i++)
	{
		if (v[i] > count)
		count = v[i];
	}
	end_time = omp_get_wtime();

	// *** Out
	printf("\nПоследовательная версия:\n");
	printf("Время на поиск = %f\n", end_time - start_time);
	printf("Максимальное значение = %d\n", count);
	// ***
```

```
        start_time = omp_get_wtime();
        // общее - v; локальные - i, n
        #pragma omp parallel shared(v) private(i, n)
        {
                /* номер текущей нити */
                n = omp_get_thread_num();
                // параллельный поиск
                #pragma omp for
                for (i = 1; i < input; i++)
                {
                        if (v[i] > count1)
                        count1 = v[i];
                }
        }
        end_time = omp_get_wtime();

        // *** Out (параллельный)
        printf("\n\nПараллельная версия:\n");
        printf("Время на поиск = %f\n", end_time - start_time);
        printf("Максимальное значение = %d\n", count1);
        // ***
]
```

Работа программы:

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ g++ task2.cpp -f
openmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out

Последовательная версия:
Время на поиск = 0.026349
Максимальное значение = 2147483481


Параллельная версия:
Время на поиск = 0.004929
Максимальное значение = 2147483481
```

Как можно заметить, параллельное выполнение операций намного быстрее последовательного.

Практика (примеры из методички):

Пример 1 (в методичке обозначается как пример 17) демонстрация работы функций omp_get_num_threads() и omp_get_thread_num():

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

int main()
{
    int count, num;
    double start_time, end_time, time;
    start_time = omp_get_wtime();
    #pragma omp parallel
    {
        count=omp_get_num_threads();
        num=omp_get_thread_num();
        if (num == 0) printf("Всего нитей: %d\n", count);
        else printf("Нить номер %d\n", num);
    }
    // время окончания работы параллельной секции
    end_time = omp_get_wtime();
    // время работы параллельной секции
    time = end_time-start_time;
    printf("Время работы параллельной секции: %f\n", time);
}
```

Предполагаемое поведение программы: Master-нить (0) выведет количество нитей, а остальные нити выведут свой номер. (порядок не определен) Можно заметить это в консоли:

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ g++ ex1_17.cpp -fopenmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
Нить номер 9
Нить номер 7
Нить номер 6
Нить номер 4
Нить номер 1
Нить номер 11
Всего нитей: 12
Нить номер 10
Нить номер 8
Нить номер 2
Нить номер 3
Нить номер 5
Время работы параллельной секции: 0.003922
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ export OMP_NUM_THREADS=1
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
Всего нитей: 1
Время работы параллельной секции: 0.000073
```

Мы видим время работы 12-ти нитей. Затем количество потоков снижается до одной нити и мы видим совершенно логичный результат — последовательная версия программы отработала быстрее. Это легко объясняется тем, что работа 12-ти потоков с printf() занимает больше времени, чем работа одной единственной нити-Master.

Пример 2 (в методичке обозначается как пример 18) Директива for:

```cpp
int main()
{
    #ifdef _OPENMP
        int i, n;
        int *A = static_cast<int *>(malloc(10000000 * sizeof(int)));
        int *B = static_cast<int *>(malloc(10000000 * sizeof(int)));
        int *C = static_cast<int *>(malloc(10000000 * sizeof(int)));


        /* Заполним исходные массивы */
        for (i = 0; i < 10000000; i++)
        {
            A[i] = i;
            B[i] = 2 * i;
            C[i] = 0;
        }
        double start_time, end_time, time;
        start_time = omp_get_wtime();
        #pragma omp parallel shared(A, B, C) private(i, n)
        {
            /* Получим номер текущей нити */
            n = omp_get_thread_num();
            #pragma omp for
            for (i = 0; i < 10000000; i++)
            {
                C[i] = A[i] + B[i];
                //printf("Нить %d сложила элементы с номером %d\n", n, i);
            }
        }
        // время окончания работы параллельной секции
        end_time = omp_get_wtime();
        free(A);
        free(B);
        free(C);
        // время работы параллельной секции
        time = end_time-start_time;
        printf("Время работы параллельной секции: %f\n", time);
    #else
        printf ("Последовательная версия, демонстрация параллелизма невозможна\n");
    #endif
```

Предполагаемое поведение программы:
Инициализация трех массивов A, B, C (общие для параллельной области). Вспомогательные переменные i и n - локальные. Каждая нить присвоит переменной n номер. Далее цикл for с распределением итераций. На каждой i-ой итерации сложение i-ых элементов массивов A и B в i-ый элемент массива C.
Для работы с большим количеством элементов (для замеров времени) использовался malloc. Вывод информации о сложении слишком сильно отражается на скорости работы, printf() убран. Заметно главное отличие последовательной и параллельной версии. Выигрыш параллельной работы, как и ожидалось.

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ export OMP_NUM_
THREADS=12
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
Время работы параллельной секции: 0.006322
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ export OMP_NUM_
THREADS=1
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
Время работы параллельной секции: 0.022255
```

Пример 3 (в методичке обозначается как пример 19) Опция schedule

Обзор работы директив:

#pragma omp for schedule (static)

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ export OMP_NUM_
THREADS=12
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ g++ ex3_19.cpp
-fopenmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
1
Нить 5 выполнила итерацию 5
Нить 6 выполнила итерацию 6
Нить 1 выполнила итерацию 1
Нить 8 выполнила итерацию 8
Нить 2 выполнила итерацию 2
Нить 7 выполнила итерацию 7
Нить 9 выполнила итерацию 9
Нить 4 выполнила итерацию 4
Нить 0 выполнила итерацию 0
Нить 3 выполнила итерацию 3
Время работы параллельной секции: 1.011282
```

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
0
Нить 0 выполнила итерацию 0
Нить 0 выполнила итерацию 1
Нить 0 выполнила итерацию 2
Нить 0 выполнила итерацию 3
Нить 0 выполнила итерацию 4
Нить 0 выполнила итерацию 5
Нить 0 выполнила итерацию 6
Нить 0 выполнила итерацию 7
Нить 0 выполнила итерацию 8
Нить 0 выполнила итерацию 9
Время работы параллельной секции: 10.002077
```

#pragma omp for schedule (static, 1)

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
0
Нить 0 выполнила итерацию 0
Нить 0 выполнила итерацию 1
Нить 0 выполнила итерацию 2
Нить 0 выполнила итерацию 3
Нить 0 выполнила итерацию 4
Нить 0 выполнила итерацию 5
Нить 0 выполнила итерацию 6
Нить 0 выполнила итерацию 7
Нить 0 выполнила итерацию 8
Нить 0 выполнила итерацию 9
Время работы параллельной секции: 10.001981
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
1
Нить 0 выполнила итерацию 0
Нить 4 выполнила итерацию 4
Нить 5 выполнила итерацию 5
Нить 6 выполнила итерацию 6
Нить 9 выполнила итерацию 9
Нить 7 выполнила итерацию 7
Нить 1 выполнила итерацию 1
Нить 8 выполнила итерацию 8
Нить 3 выполнила итерацию 3
Нить 2 выполнила итерацию 2
Время работы параллельной секции: 1.014493
```

#pragma omp for schedule(static, 2)

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
0
Нить 0 выполнила итерацию 0
Нить 0 выполнила итерацию 1
Нить 0 выполнила итерацию 2
Нить 0 выполнила итерацию 3
Нить 0 выполнила итерацию 4
Нить 0 выполнила итерацию 5
Нить 0 выполнила итерацию 6
Нить 0 выполнила итерацию 7
Нить 0 выполнила итерацию 8
Нить 0 выполнила итерацию 9
Время работы параллельной секции: 10.002023
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
1
Нить 0 выполнила итерацию 0
Нить 1 выполнила итерацию 2
Нить 2 выполнила итерацию 4
Нить 3 выполнила итерацию 6
Нить 4 выполнила итерацию 8
Нить 3 выполнила итерацию 7
Нить 1 выполнила итерацию 3
Нить 0 выполнила итерацию 1
Нить 2 выполнила итерацию 5
Нить 4 выполнила итерацию 9
Время работы параллельной секции: 2.005306
```

#pragma omp for schedule (dynamic)

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
0
Нить 0 выполнила итерацию 0
Нить 0 выполнила итерацию 1
Нить 0 выполнила итерацию 2
Нить 0 выполнила итерацию 3
Нить 0 выполнила итерацию 4
Нить 0 выполнила итерацию 5
Нить 0 выполнила итерацию 6
Нить 0 выполнила итерацию 7
Нить 0 выполнила итерацию 8
Нить 0 выполнила итерацию 9
Время работы параллельной секции: 10.002089
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
1
Нить 0 выполнила итерацию 6
Нить 10 выполнила итерацию 8
Нить 4 выполнила итерацию 5
Нить 3 выполнила итерацию 0
Нить 7 выполнила итерацию 7
Нить 1 выполнила итерацию 2
Нить 9 выполнила итерацию 1
Нить 5 выполнила итерацию 9
Нить 8 выполнила итерацию 4
Нить 2 выполнила итерацию 3
Время работы параллельной секции: 1.006048
```

#pragma omp for schedule (dynamic, 2)

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ g++ ex3_19.cpp
-fopenmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
0
Нить 0 выполнила итерацию 0
Нить 0 выполнила итерацию 1
Нить 0 выполнила итерацию 2
Нить 0 выполнила итерацию 3
Нить 0 выполнила итерацию 4
Нить 0 выполнила итерацию 5
Нить 0 выполнила итерацию 6
Нить 0 выполнила итерацию 7
Нить 0 выполнила итерацию 8
Нить 0 выполнила итерацию 9
Время работы параллельной секции: 10.001842
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
1
Нить 4 выполнила итерацию 0
Нить 5 выполнила итерацию 2
Нить 8 выполнила итерацию 4
Нить 6 выполнила итерацию 6
Нить 2 выполнила итерацию 8
Нить 4 выполнила итерацию 1
Нить 5 выполнила итерацию 3
Нить 2 выполнила итерацию 9
Нить 6 выполнила итерацию 7
Нить 8 выполнила итерацию 5
Время работы параллельной секции: 2.015585
```

#pragma omp for schedule (guided)

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ g++ ex3_19.cpp
-fopenmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
0
Нить 0 выполнила итерацию 0
Нить 0 выполнила итерацию 1
Нить 0 выполнила итерацию 2
Нить 0 выполнила итерацию 3
Нить 0 выполнила итерацию 4
Нить 0 выполнила итерацию 5
Нить 0 выполнила итерацию 6
Нить 0 выполнила итерацию 7
Нить 0 выполнила итерацию 8
Нить 0 выполнила итерацию 9
Время работы параллельной секции: 10.002007
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
1
Нить 4 выполнила итерацию 0
Нить 1 выполнила итерацию 4
Нить 7 выполнила итерацию 1
Нить 0 выполнила итерацию 3
Нить 6 выполнила итерацию 2
Нить 9 выполнила итерацию 5
Нить 8 выполнила итерацию 6
Нить 2 выполнила итерацию 7
Нить 5 выполнила итерацию 9
Нить 11 выполнила итерацию 8
Время работы параллельной секции: 1.001568
```

#pragma omp for schedule (guided, 2)

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ g++ ex3_19.cpp
-fopenmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
0
Нить 0 выполнила итерацию 0
Нить 0 выполнила итерацию 1
Нить 0 выполнила итерацию 2
Нить 0 выполнила итерацию 3
Нить 0 выполнила итерацию 4
Нить 0 выполнила итерацию 5
Нить 0 выполнила итерацию 6
Нить 0 выполнила итерацию 7
Нить 0 выполнила итерацию 8
Нить 0 выполнила итерацию 9
Время работы параллельной секции: 10.002144
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
1
Нить 8 выполнила итерацию 0
Нить 5 выполнила итерацию 4
Нить 1 выполнила итерацию 2
Нить 3 выполнила итерацию 6
Нить 9 выполнила итерацию 8
Нить 8 выполнила итерацию 1
Нить 5 выполнила итерацию 5
Нить 1 выполнила итерацию 3
Нить 3 выполнила итерацию 7
Нить 9 выполнила итерацию 9
Время работы параллельной секции: 2.004573
```

В параллельной области выполняется цикл, итерации которого распределяются между существующими нитями. На каждой итерации будет напечатано, какая нить выполнила данную итерацию. В тело цикла вставлена также задержка, имитирующая некоторые вычисления. Код можно посмотреть ниже. Здесь последовательные версии ожидаемо медленнее, чем параллельные, и скорость их всех в районе 10 секунд. Можно сказать, что директивы schedule не влияют на последовательные вычисления.
Параллельные же версии показали различный результат.
Расписание определяет, как итерации цикла распределяются между потоками. Выбор правильного расписания может сильно повлиять на скорость работы приложения.

Static — в начале цикла решается, какой поток будет работать со значениями на конкретной итерации.
Dynamic — нить для вычислений следующей итерации выбирается «на лету», что может быть полезно, если вычисления занимают разное количество нитей.
Static – итерации делятся на блоки по размер итераций и статически разделяются между потоками (в начале цикла);
Dynamic – распределение итерационных блоков осуществляется динамически (по умолчанию размер=1) Нить для вычислений следующей итерации выбирается «на лету», что может быть полезно, если вычисления занимают разное количество нитей.
Guided – размер итерационного блока уменьшается экспоненциально при каждом распределении; размер определяет минимальный размер блока (по умолчанию размер (chunk) =1)

Отлия #pragma omp for schedule (static), #pragma omp for schedule (static, 1), #pragma omp for schedule (static, 2)

Тут всё просто, числа(chunk) — это количество итераций, получаемое конкретной нитью.
Если значение chunk не указано, то всё множество итераций делится на непрерывные куски примерно одинакового размера (конкретный способ зависит от реализации).
Видно, что чем больше итераций выполняет одна нить,тем дольше выполняется программа.
Без указания chunk на текущей машине побеждает по скорости.

#pragma omp for schedule (dynamic) и #pragma omp for schedule (dynamic, 2)

Динамическое распределение итераций, но при указании chunk одна нить выполняет chunk итераций.
Здесь видно логичное уменьшение времени работы при chunk равном 2.

#pragma omp for schedule (guided) и #pragma omp for schedule (guided, 2)

Здесь порции уменьшаются до величины chunk. При значении chunk = 2 программа работает медленнее.

Пример 4 (в методичке обозначается как пример 20) Опция schedule

#pragma omp for schedule (static, 6)

Последовательная версия

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
0
Нить 0 выполнила итерацию 0
Нить 0 выполнила итерацию 1
Нить 0 выполнила итерацию 2
Нить 0 выполнила итерацию 3
Нить 0 выполнила итерацию 4
Нить 0 выполнила итерацию 5
Нить 0 выполнила итерацию 6
Нить 0 выполнила итерацию 7
Нить 0 выполнила итерацию 8
Нить 0 выполнила итерацию 9
Нить 0 выполнила итерацию 10
```

(***)

```
Нить 0 выполнила итерацию 190
Нить 0 выполнила итерацию 191
Нить 0 выполнила итерацию 192
Нить 0 выполнила итерацию 193
Нить 0 выполнила итерацию 194
Нить 0 выполнила итерацию 195
Нить 0 выполнила итерацию 196
Нить 0 выполнила итерацию 197
Нить 0 выполнила итерацию 198
Нить 0 выполнила итерацию 199
Время работы параллельной секции: 200.041247
```

Параллельная версия

```
0 - последовательная версия | !0 - параллельная
1
Нить 5 выполнила итерацию 30
Нить 9 выполнила итерацию 54
Нить 2 выполнила итерацию 12
Нить 8 выполнила итерацию 48
Нить 3 выполнила итерацию 18
Нить 10 выполнила итерацию 60
Нить 11 выполнила итерацию 66
Нить 7 выполнила итерацию 42
Нить 1 выполнила итерацию 6
Нить 6 выполнила итерацию 36
Нить 0 выполнила итерацию 0
Нить 4 выполнила итерацию 24
Нить 9 выполнила итерацию 55
Нить 5 выполнила итерацию 31
```

(***)

```
Нить 1 выполнила итерацию 154
Нить 2 выполнила итерацию 160
Нить 4 выполнила итерацию 172
Нить 8 выполнила итерацию 197
Нить 7 выполнила итерацию 191
Нить 5 выполнила итерацию 179
Нить 3 выполнила итерацию 167
Нить 6 выполнила итерацию 185
Нить 0 выполнила итерацию 149
Нить 1 выполнила итерацию 155
Нить 2 выполнила итерацию 161
Нить 4 выполнила итерацию 173
Время работы параллельной секции: 18.005807
```

#pragma omp for schedule (dynamic, 6)

Последовательная версия

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
0
Нить 0 выполнила итерацию 0
Нить 0 выполнила итерацию 1
Нить 0 выполнила итерацию 2
Нить 0 выполнила итерацию 3
Нить 0 выполнила итерацию 4
Нить 0 выполнила итерацию 5
Нить 0 выполнила итерацию 6
Нить 0 выполнила итерацию 7
Нить 0 выполнила итерацию 8
Нить 0 выполнила итерацию 9
Нить 0 выполнила итерацию 10
```

(***)

```
Нить 0 выполнила итерацию 190
Нить 0 выполнила итерацию 191
Нить 0 выполнила итерацию 192
Нить 0 выполнила итерацию 193
Нить 0 выполнила итерацию 194
Нить 0 выполнила итерацию 195
Нить 0 выполнила итерацию 196
Нить 0 выполнила итерацию 197
Нить 0 выполнила итерацию 198
Нить 0 выполнила итерацию 199
Время работы параллельной секции: 200.041437
```

Параллельная версия

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
1
Нить 0 выполнила итерацию 54
Нить 6 выполнила итерацию 42
Нить 7 выполнила итерацию 24
Нить 9 выполнила итерацию 6
Нить 4 выполнила итерацию 30
Нить 8 выполнила итерацию 48
Нить 2 выполнила итерацию 36
Нить 10 выполнила итерацию 18
Нить 3 выполнила итерацию 12
Нить 5 выполнила итерацию 0
Нить 1 выполнила итерацию 60
Нить 11 выполнила итерацию 66
Нить 6 выполнила итерацию 43
Нить 9 выполнила итерацию 7
Нить 2 выполнила итерацию 37
Нить 0 выполнила итерацию 55
```

(***)

```
Нить 6 выполнила итерацию 166
Нить 8 выполнила итерацию 184
Нить 1 выполнила итерацию 190
Нить 0 выполнила итерацию 149
Нить 5 выполнила итерацию 197
Нить 6 выполнила итерацию 167
Нить 7 выполнила итерацию 161
Нить 9 выполнила итерацию 155
Нить 1 выполнила итерацию 191
Нить 2 выполнила итерацию 173
Нить 8 выполнила итерацию 185
Нить 3 выполнила итерацию 179
Время работы параллельной секции: 18.015429
```

#pragma omp for schedule (guided, 6)

Последовательная версия

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ g++ ex4_20.cpp
-fopenmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
0
Нить 0 выполнила итерацию 0
Нить 0 выполнила итерацию 1
Нить 0 выполнила итерацию 2
Нить 0 выполнила итерацию 3
Нить 0 выполнила итерацию 4
Нить 0 выполнила итерацию 5
Нить 0 выполнила итерацию 6
Нить 0 выполнила итерацию 7
Нить 0 выполнила итерацию 8
Нить 0 выполнила итерацию 9
Нить 0 выполнила итерацию 10
```

(***)

```
Нить 0 выполнила итерацию 186
Нить 0 выполнила итерацию 187
Нить 0 выполнила итерацию 188
Нить 0 выполнила итерацию 189
Нить 0 выполнила итерацию 190
Нить 0 выполнила итерацию 191
Нить 0 выполнила итерацию 192
Нить 0 выполнила итерацию 193
Нить 0 выполнила итерацию 194
Нить 0 выполнила итерацию 195
Нить 0 выполнила итерацию 196
Нить 0 выполнила итерацию 197
Нить 0 выполнила итерацию 198
Нить 0 выполнила итерацию 199
Время работы параллельной секции: 200.041219
```

Параллельная версия

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
0 - последовательная версия | !0 - параллельная
1
Нить 0 выполнила итерацию 111
Нить 7 выполнила итерацию 33
Нить 4 выполнила итерацию 119
Нить 6 выполнила итерацию 17
Нить 2 выполнила итерацию 93
Нить 1 выполнила итерацию 47
Нить 10 выполнила итерацию 83
Нить 8 выполнила итерацию 102
Нить 5 выполнила итерацию 60
Нить 9 выполнила итерацию 0
Нить 3 выполнила итерацию 72
Нить 11 выполнила итерацию 126
Нить 0 выполнила итерацию 112
```

(***)

```
Нить 11 выполнила итерацию 184
Нить 3 выполнила итерацию 174
Нить 4 выполнила итерацию 196
Нить 5 выполнила итерацию 180
Нить 1 выполнила итерацию 191
Нить 11 выполнила итерацию 185
Нить 4 выполнила итерацию 197
Нить 1 выполнила итерацию 192
Нить 4 выполнила итерацию 198
Нить 11 выполнила итерацию 186
Время работы параллельной секции: 19.006473
```

Здесь аналогично прошлому примеру, только одной нитью выполняется по 6 итераций, что отражается на скорости работы. Последовательные версии рассматривать нет смысла, они все одинаково а районе двухсот секунд.
Время параллельных версий:
Static — ~18 секунд
Dynamic — ~18 секунд
Guided — ~19 секунд, оказался дольше остальных

Пример 5 (в методичке обозначается как пример 21) Директива sections
Цель данного примера — продемонстрировать работу sections

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

int main()
{
    int n;
    #pragma omp parallel private(n)
    {
        n=omp_get_thread_num();
        #pragma omp sections
        {
            #pragma omp section
            {
                printf("Первая секция, процесс %d\n", n);
            }
            #pragma omp section
            {
                printf("Вторая секция, процесс %d\n", n);
            }
            #pragma omp section
            {
                printf("Третья секция, процесс %d\n", n);
            }
        }
        printf("Параллельная область, процесс %d\n", n);
    }
}
```

Предположительно, программа распределит секции на 3 нити, они выведут сообщение со своим номером, далее все нити выведут одинаковое сообщение со своим номером.
Результат:

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ g++ ex5_21.cpp
-fopenmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
Третья секция, процесс 7
Вторая секция, процесс 4
Первая секция, процесс 10
Параллельная область, процесс 7
Параллельная область, процесс 1
Параллельная область, процесс 4
Параллельная область, процесс 11
Параллельная область, процесс 8
Параллельная область, процесс 3
Параллельная область, процесс 9
Параллельная область, процесс 10
Параллельная область, процесс 5
Параллельная область, процесс 0
Параллельная область, процесс 2
Параллельная область, процесс 6
```

В результате секции взяли на выполнение 3 нити, 7-4-10, далее все нити вывели свой номер (в том числе и 7-4-10). Ожидания оправдались.

Пример 6 (в методичке обозначается как пример 22) Опция lastprivate.

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

int main(int argc, char *argv[])
{
    int n=0;
    #pragma omp parallel
    {
        #pragma omp sections lastprivate(n)
        {
            #pragma omp section
            {
                n=1;
            }
            #pragma omp section
            {
                n=2;
            }
            #pragma omp section
            {
                n=3;
            }
        }
        printf("Значение n на нити %d: %d\n",
        omp_get_thread_num(), n);
    }
    printf("Значение n в последовательной области: %d\n", n);
}
```

Продемонстрируем использование опции lastprivate. Опция lastprivate используется вместе с директивой sections. Переменная n объявлена как lastprivate переменная. Три нити, выполняющие секции присваивают своей локальной копии n разные значения.
На выходе из области sections значение n из последней секции присваивается локальным копиям во всех нитях, поэтому все нити напечатают число 3. Это же значение должно сохраниться для переменной n и в последовательной области.

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ g++ ex6_22.c -f
openmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4/actual$ ./a.out
Значение n на нити 3: 3
Значение n на нити 0: 3
Значение n на нити 2: 3
Значение n на нити 9: 3
Значение n на нити 6: 3
Значение n на нити 8: 3
Значение n на нити 5: 3
Значение n на нити 11: 3
Значение n на нити 7: 3
Значение n на нити 1: 3
Значение n на нити 10: 3
Значение n на нити 4: 3
Значение n в последовательной области: 3
```

Вывод: Изучено распределение работы между имеющимися нитями.