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: Существуют способы и помимо for. Для распределения работ можно использовать параллельные секции sections и конструкцию single.
В цикле for по умолчанию барьером для потоков является конец цикла. Все потоки достигнув конца цикла дожидаются тех, кто еще не завершился, после чего основная нить продолжает выполняться дальше. Используя условие nowait для цикла можно разрешить основной нити не дожидаться завершения дочерних нитей.
Если нужно сделать действия, которые не являются итерациями цикла, то применяется секции sections. Для разрешения не ждать синхронизации можно использовать nowait.
Конструкция singngle используется, если действия должна выполнить одна нить.

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: Такое возможно. Чтобы избежать подобного, можно использовать `#pragma omp for schedule для распределения`

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: _Помимо параллельных секций есть ещё parallel construct (параллельный фрагмент, блок программы, управляемый директивой parallel. Именно параллельные фрагменты,

совместно с параллельными областями, представляют параллельно-выполняемую часть программы) и параллельная область patallel region (параллельно выполняемые участки программного кода, динамически-возникающие в результате вызова функций из параллельных фрагментов).

8. Вопрос: «Как при выходе из параллельных секций разослать значение некоторой локальной переменной всем нитям, выполняющим данную параллельную область?»
Ответ: Возможно использование lastprivate для этих целей, здесь происходит работа по переносу данных с последних витках.

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>
#include <stdlib.h>
#include <iomanip>

using namespace std;

// Создание векторов для скалярного произведения
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;
}

// Хранение результатов сложения
int sequental_res, parallel_res;

// замеры
double time(bool parallel, int n, int threads)
{
	double start_time, end_time;

	// создание векторов
	vector<int> v(randVec(n));
	vector<int> v1(randVec(n));

	// объявим переменные результатов произведения
	int res = 0;
	int res_omp = 0;
	// последовательно умножим
	start_time = omp_get_wtime();

	for (int i = 0; i < n; i++)
	{
		res += v[i] * v1[i];
	}
```

```
	end_time = omp_get_wtime();

	sequental_res = 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) if(parallel)
	for (int i = 0; i < n; i++)
	{
		res_omp += v[i] * v1[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 threads)
{
	// Time table
	cout << left << fixed << setprecision(3)
	<< setw(10) << n << setw(17) << time(0, n, threads) << setw(18)
	<< time(1, n, threads) << setw(18) << sequental_res << setw(15) << parallel_res
<< endl;
}

// точка входа
int main(int argc, char* argv[])
{
	cout << "Кащенко В. А. ПрИб-181\nСкалярное произведение векторов
(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;
	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);
}
```

Результат:

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4$ g++ task1.cpp -fopenmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4$ ./a.out
Кащенко В. А. ПрИб-181
Скалярное произведение векторов (параллельная версия)

4 потока:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        9.115            753.957           -794639953        -794639953
100       1.783            2.343             -721643447        -721643447
500       3.641            2.237             1044310719        1044310719
1000      8.183            161.291           -433523783        -433523783
5000      64.461           103.791           -2049819352       -2049819352
15000     87.299           106.969           905663617         905663617
30000     125.477          113.342           935723756         935723756
50000     204.858          135.264           -1350815773       -1350815773
100000    409.261          187.077           1141580322        1141580322
1000000   4145.687         967.572           1895630133        1895630133

6 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        1.938            2.339             -2000088928       -2000088928
100       1.337            1.998             1550571463        1550571463
500       3.801            2.460             474464276         474464276
1000      6.579            2.757             2023095133        2023095133
5000      28.470           98.312            -1744903142       -1744903142
15000     82.890           105.584           -2054380985       -2054380985
30000     123.319          117.589           -830725665        -830725665
50000     203.373          137.033           -139216615        -139216615
100000    406.894          171.140           -4551371          -4551371
1000000   4150.778         958.537           399887952         399887952

12 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        1.908            2.645             -1703171841       -1703171841
100       1.615            1.971             -1036604972       -1036604972
500       3.735            2.367             -1646080060       -1646080060
1000      6.564            2.638             1750049227        1750049227
5000      28.502           95.146            754298451         754298451
15000     62.545           112.047           917253250         917253250
30000     123.690          104.620           410125614         410125614
50000     285.377          141.437           1701563726        1701563726
100000    407.232          174.182           222034417         222034417
1000000   4127.222         876.058           -2037855654       -2037855654
```

Пояснение:

- На экране терминала видно, что при малом числе итераций «распараллеливать» программу не имеет смысла.
- Выигрыш в производительности виден только начиная от 500 элементов массива.
- Самая неэффективная конфигурация оказалась в 4 потока. Самая эффективная конфигурация оказалась в 12 потоков. (виден выигрыш параллельных вычислений)

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

```
// time table
#include <stdio.h>
#include <omp.h>
#include <vector>
#include <random>
#include <algorithm>
#include <iterator>
#include <iostream>
#include <stdlib.h>
#include <iomanip>

using namespace std;

// Создание векторов для поиска
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;
}

// Хранение результатов сложения
int sequental_res, parallel_res;

// замеры
double time(bool parallel, int n, int threads)
{
	double start_time, end_time;
	// i - итератор
	int i;

	// создание векторов
	vector<int> v(randVec(n));

	//начальные значения векторов
	int count = v[0];
	int count1 = v[0];

	// последовательный поиск
	start_time = omp_get_wtime();
	for (i = 1; i < n; i++)
	{
```

```
                if (v[i] > count)
                count = v[i];
        }
        end_time = omp_get_wtime();
        sequental_res = count;

        // параллельный поиск
        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 < n; i++)
                {
                        if (v[i] > count1)
                        count1 = v[i];
                }
        }
        end_time = omp_get_wtime();
        parallel_res = count1;

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

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

// точка входа
int main(int argc, char* argv[])
{
        cout << "Кащенко В. А. ПрИб-181\nСкалярное произведение векторов
        (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;
        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);
}
```

Результат:

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4$ g++ task2.cpp -fopenmp -O0
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4$ ./a.out
Кащенко В. А. ПрИб-181
Поиск вектора (параллельная версия)

4 потока:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        765.967          11.682            2075736891        2075736891
100       2.290            2.227             2126862236        2126862236
500       2.319            2.407             2145108142        2145108142
1000      2.589            2861.127          2143750727        2143750727
5000      3.762            4191.211          2146815893        2146815893
15000     97.175           93.104            2147235806        2147235806
30000     92.766           80.022            2147325782        2147325782
50000     97.689           90.639            2147480953        2147480953
100000    95.211           96.168            2147465574        2147465574
1000000   96.773           94.815            2147481020        2147481020

6 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        2.569            2.496             2137688780        2137688780
100       2.450            2.160             2050244324        2050244324
500       37.368           2.546             2137733992        2137733992
1000      2.241            2.279             2142041188        2142041188
5000      2.564            2.342             2146547708        2146547708
15000     107.657          91.469            2146988746        2146988746
30000     98.690           76.953            2147340697        2147340697
50000     95.569           98.476            2147287374        2147287374
100000    93.789           91.263            2147413336        2147413336
1000000   95.617           100.123           2147482687        2147482687

12 потоков:
Кол-во    Послед-но, мкс   Паралл-но, мкс    Р-ат послед-но    Р-ат паралл-но
50        2.663            2.424             2107774626        2107774626
100       2.225            2.103             2047138427        2047138427
500       2.462            2.467             2127029147        2127029147
1000      2.278            2.184             2147045493        2147045493
5000      2.360            2.538             2147352984        2147352984
15000     100.534          87.905            2147051431        2147051431
30000     101.721          94.360            2147285937        2147285937
50000     94.774           90.707            2147350077        2147350077
100000    103.919          99.221            2147478883        2147478883
1000000   98.413           97.424            2147480130        2147480130
```

Пояснение:

- На экране терминала виден результат поиска векторов и время, которое показывает эффективность работы именно 12 потоков.
- Особого выигрыша в производительности не видно. Подобного рода задачи можно не распараллеливать вообще.
- Сильная разница видна только при 4-ех потоках. Подобная оценка появляется и при повторных замерах, что говорит о том, что это самая неэффективная конфигурация.
- Критически сильной разницы нет, можно использовать последовательные вычисления при решении подобного рода задач.

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:

```
(base) uke_zebrano@pop-os:~/Рабочий стол/labs_parallel/4$ ./a.out
4 потока:
Количество элементов  Последовательно, мкс  Параллельно, мкс
50                    4.008                 129.733
100                   1.563                 1.908
500                   2.272                 1.692
1000                  3.600                 2.562
5000                  14.911                6.009
15000                 42.627                11.947
30000                 83.873                22.803
50000                 138.752               40.185
100000                304.877               78.139
1000000               2415.027              659.688

6 потоков:
Количество элементов  Последовательно, мкс  Параллельно, мкс
50                    1.466                 1.380
100                   0.804                 1.250
500                   1.909                 1.365
1000                  3.290                 1.973
5000                  12.077                4.192
15000                 33.789                9.466
30000                 66.808                34.319
50000                 110.872               28.828
100000                220.709               56.508
1000000               2443.383              1354.677

12 потоков:
Количество элементов  Последовательно, мкс  Параллельно, мкс
50                    25.961                1.884
100                   0.780                 1.238
500                   2.084                 1.332
1000                  3.057                 1.875
5000                  12.346                4.495
15000                 34.130                11.226
30000                 67.130                21.564
50000                 111.249               28.814
100000                276.356               56.667
1000000               2770.770              717.698
```

Код программы (ex2_18.cpp):

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

using namespace std;

// замеры
double time(bool parallel, int n, int threads)
{
	if (threads <= omp_get_max_threads())
	{
		omp_set_num_threads(threads);
	}
	double start_time, end_time;
	int *A = static_cast<int *>(malloc(n * sizeof(int)));
	int *B = static_cast<int *>(malloc(n * sizeof(int)));
	int *C = static_cast<int *>(malloc(n * sizeof(int)));
	for (int i = 0; i < n; ++i)
	{
		A[i] = i;
		B[i] = 2 * i;
		C[i] = 0;
	}

	start_time = omp_get_wtime();
	#pragma omp parallel for shared(A, B, C) if(parallel)
		for (int i = 0; i < n; ++i)
			C[i] = A[i] * B[i];
	end_time = omp_get_wtime();

	free(A);
	free(B);
	free(C);
	return (end_time-start_time) * 1'000'000 // мкс;
}

// вывод на экран времени
void print_time(int n, int threads)
{
	cout << left << fixed << setprecision(3) << setw(22) << n << setw(22) <<
time(0, n, threads) << setw(16) << time(1, n, threads) << setw(16) << endl;
}

// точка входа
int main()
{
	#ifdef _OPENMP
```

```
		cout << "4 потока:\n";
		cout << setw(22) << "Количество элементов " << setw(13) << "Последовательно, мкс " << setw(16) << "Параллельно, мкс" << end l;
		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";
		cout << setw(22) << "Количество элементов " << setw(13) << "Последовательно, мкс " << setw(16) << "Параллельно, мкс" << 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";
		cout << setw(22) << "Количество элементов " << setw(13) << "Последовательно, мкс " << setw(16) << "Параллельно, мкс" << 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);
	#else
		printf ("Последовательная версия, демонстрация параллелизма невозможна\n");
	#endif
}
```

Предполагаемое поведение программы:

Инициализация трех массивов A, B, C (общие для параллельной области + логическая переменная, определяющая параллельность/последовательность). 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 по работе данной программы:

- Малое число элементов распараллелить дольше (50 против 100, к примеру), и выигрыша от параллелизма мы не получим. (потери производительности для организации параллелизма). На четырёх потоках это заметно очень сильно. 6 и 12 потоков тоже теряют в скорости, но не так сильно;

- на большом размере массива скорость параллелизма падает (это ожидаемо, различие на порядок достаточно велико);

- среднее время параллелизма (мкс) для четырех потоков: 95,42 мс
для шести потоков: 149,34 мс
для двенадцати потоков: 84,63 мс

Это говорит нам о том, что не всегда увеличение количества потоков даст выигрыш в скорости. 6 потоков — самая неэффективная реализация параллелизма.

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.