Лабораторная работа №4	M3136	2023
OpenMP	Рязанова Екатерина Владимировна	

Цель работы: знакомство с основами многопоточного программирования.

Инструментарий и требования к работе: Microsoft Visual C++ в составе Visual Studio Community 2022

Описание конструкций OpenMP для распараллеливания команд.

Директива parallel создает параллельный регион (блок кода, который всеми потоками исполняется одновременно) для структурированного блока, следующего за ней. Также указывает, что этот блок кода выполняется параллельно. В этом параллельном регионе каждый из созданных потоков выполняет одинаковый код, но не одинаковый набор команд.

В данной лабораторной работе использовались следующие конструкции OpenMP:

- `#pragma omp for schedule(static)`

Для распараллеливания цикла используется директива разделения работы «for». Она сообщает, что при выполнении цикла for в параллельном регионе итерации цикла распределяются между потоками группы. Например, если имеется машина с 8 ядрами, то теоретически мы можем получить ускорение в 8 раз, но это не так. В реальности ускорение меньше из-за синхронизации потоков в конце параллельного региона (в конце все потоки блокируются до тех пор, пока последний не закончит работать).

Schedule – способ разбиения циклов на потоки, по умолчанию dynamic. У static все итерации распределены по потокам поровну, без учёта их сложности. У dynamic наоборот, потоки исходно ничего не делают, но просят задачи. В таком случае некоторые потоки заканчивают работу быстрее, открываются новые и тд. Chunk size – количество циклов, выделяемое на поток.

- `#pragma omp atomic`

Директива «atomic» относится к оператору, стоящему прямо за ней. Она гарантирует корректную работу с переменной в левой части. Эта директива распространяется только на простые, не перегруженные операторы. На практике эту директиву рационально использовать только при довольно редком обращении к общим переменным, например, счетчик. `omp_set_num_threads()`

Задает число потоков для выполнения следующего параллельного региона.

- `omp_get_wtime`

Возвращает время в секундах.

Описание работы написанного кода

Сначала обрабатываем возможные ошибки при открытии и чтении файлов и количества потоков:

```
int main(int argc, char* argv[]) {
    if (argc < 4) {
        printf("Too few parameters");
        exit(-1);
```

```
    }
    key1 = atoi(argv[1]);
    if (key1 < -1) {
        printf("Wrong number of threads");
        exit(-1);
    }
    if (key1 > omp_get_max_threads()) {
        printf("Too many number of threads");
        exit(-1);
    }
    if (key1 == -1) key1 = 1;         -обработка случаев при кол-ве потоков 0 или -1
    if (key1 == 0) key1 = omp_get_max_threads();

    FILE* inp = fopen(argv[2], "rt");
    if (!inp) {
        printf("Can`t open input file");
        exit(-1);
    }
    if (fscanf(inp, "%f%d", &R, &cnt) != 2) {
        printf("Error reading input file");
        exit(-1);
    }
    if ((R <= 0) || (cnt <= 0)) {
        printf("Wrong input parameters");
        exit(-1);
    }

          fclose(inp); ---закрываем входной файл.

omp_set_num_threads(key1);              -назначаем число потоков
double tstart = omp_get_wtime();        -начало времени отсчета
int hitcnt = 0;                         -счетчик кол-ва попаданий
```

Затем начинается распараллеленная часть.

```
#pragma omp parallel
    {
        srand(int(time(NULL)) ^ omp_get_thread_num());   -в каждом потоке запускаем
изолированный random seed в зависимости от совокупности времени номера потока (таким
образом у каждого свой индивидуальный seed)
#pragma omp for schedule(static)
        for (int i = 1; i < cnt; i++) {                  -цикл на нужное кол-во
итераций
            double x = ((double)rand() / 32767) * 2 - 1;  -рандомная координата по x
            double y = ((double)rand() / 32767) * 2 - 1;  -рандомная координата по y
            bool hit = (x * x + y * y) <= 1;              -проверка попадания в круг
            if (hit) {                                    -если попали в круг
#pragma omp atomic          --точечная работа с переменной
                hitcnt += 1;                              -увеличение счетчика
            }
        }}
```

После этого определяем время завершения:

```
double tend = omp_get_wtime();
```

Затем остался только вывод:

```
printf("Time(%i thread(s)): %g ms\n", key1, (tend - tstart) * 1000); -вывод времени
в консоль
    FILE* outp = fopen(argv[3], "wt"); -открываем выходной файл
    fprintf(outp, "%f", (double)hitcnt / cnt * R * R * 4);  -записываем ответ в
выходной файл

          fclose(outp); - закрываем этот файл
```

Результат работы написанной программы с указанием процессора, на котором производилось тестирование

Процессор QuadCore Intel Core i5-10300H, 4200 MHz (42 x 100)

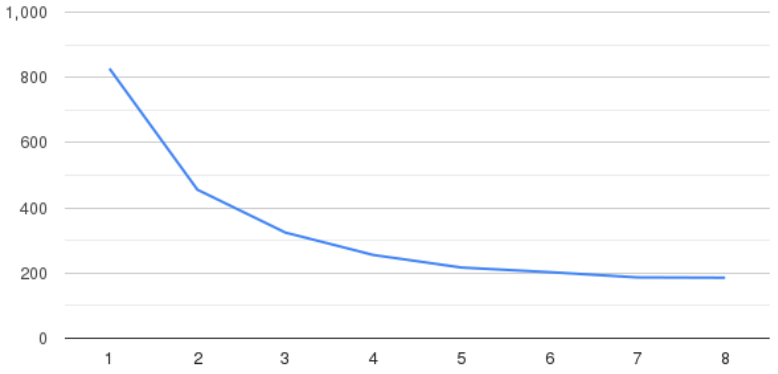
Time(8 thread(s)): 188.738 ms

In.txt 1 10000000 out.txt 3.141249

Экспериментальная часть

Для минимизации влияния степени загруженности процессора другими процессами, время усредняется по 5 запускам. В файле input 1 10000000, так как распараллеливать малые циклы невыгодно.

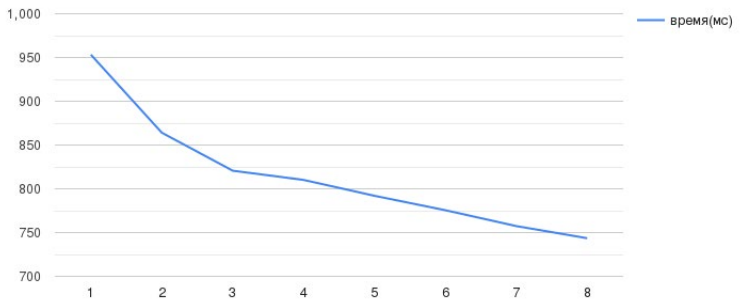
1) при различных значениях числа потоков при одинаковом параметре schedule (static);



X	Y
кол-во потоков	t(ms)
1	827,0976
2	455,668
3	324,1796
4	255,4444
5	216,9928
6	202,8762
7	186,7346
8	185,7818

кол-во потоков	время(мс)	среднее	кол-во потоков	время(мс)	среднее
1	817,51		5	217,75	
	835,677			216,155	
	827,031	827,0976		217,028	216,9928
	829,802			215,907	
	825,468			218,124	
2	455,452		6	211,424	
	454,611			210,604	
	457,071	455,668		199,776	202,8762
	463,509			191,028	
	447,697			201,549	
3	312,495		7	185,257	
	324,846			186,392	
	328,908	324,1796		190,558	186,7346
	324,413			182,839	
	330,236			188,627	
4	265,906		8	189,905	
	253,553			192,764	
	254,789	255,4444		194,488	185,7818
	251,132			163,764	
	251,842			187,988	

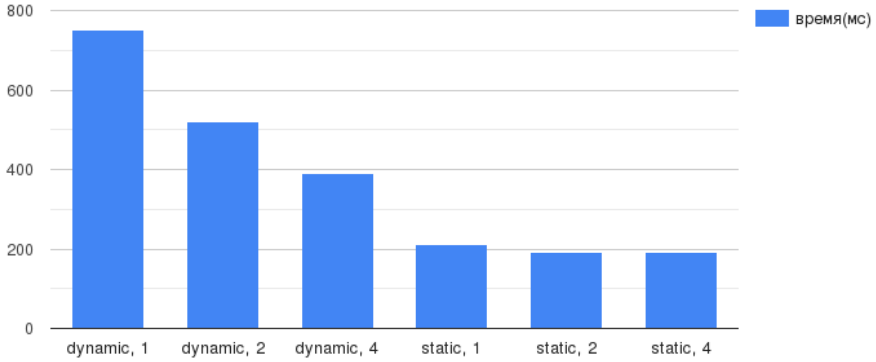
Dynamic:



X	Y
кол-во потоков	время(мс)
1	953,2482
2	864,114
3	820,6774
4	810,1186
5	791,9872
6	775,5016
7	757,3938
8	743,535

кол-во потоков	время(мс)	среднее	кол-во потоков	время(мс)	среднее
1	945,561		5	781,679	
	947,375			786,863	
	961,682	953,2482		786,148	791,9872
	959,173			797,723	
	952,45			807,523	
2	838,749		6	776,237	
	864,254			784,86	
	864,841	864,114		766,355	775,5016
	877,839			776,401	
	874,887			773,655	
3	769,752		7	780,871	
	813,355			728,994	
	852,937	820,6774		778,94	757,3938
	851,659			743,159	
	815,684			755,005	
4	785,234		8	730,915	
	805,999			751,645	
	833,608	810,1186		754,062	743,535
	830,265			746,162	
	795,487			734,891	

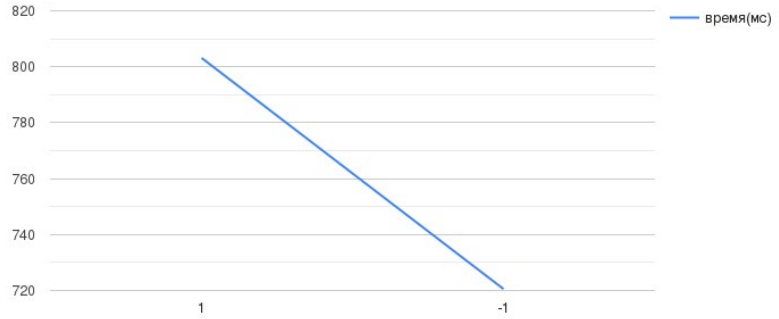
2) при одинаковом значении числа потоков(8) при различных параметрах schedule;



X	Y
кол-во потоков	время(мс)
dynamic, 1	749,6064
dynamic, 2	518,8168
dynamic, 4	390,4236
static, 1	210,872
static, 2	190,6998
static, 4	192,9228

schedule	t(ms)	среднее	schedule	t(ms)	среднее
dynamic, 1	786,055		static, 1	208,545	
	711,155			195,162	
	771,889	749,6064		270,388	210,872
	739,081			196,455	
	739,852			183,81	
dynamic, 2	585,544		static, 2	221,856	
	473,986			183,882	
	534,051	518,8168		188,794	190,6998
	506,557			185,741	
	493,946			173,226	
dynamic, 4	425,497		static, 4	217,32	
	362,982			180,173	
	419,465	390,4236		181,312	192,9228
	380,248			211,107	
	363,926			174,702	

3) с выключенным openmp и с включенным с 1 потоком.



1 поток или без omp	время(мс)	среднее
1	819,988	
	815,783	
	797,118	803,1384
	780,711	
	802,092	
без omp	732,949	
	713,86	
	722,548	720,528
	720,102	
	713,181	

Список источников

https://learn.microsoft.com/ru-ru/cpp/parallel/openmp/reference/openmp-directives?view=msvc-170

https://habr.com/ru/company/intel/blog/85273/

Листинг кода

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <omp.h>
float R;
long long cnt;
int key1;
int main(int argc, char* argv[]) {
    if (argc < 4) {
        printf("Too few parameters");
        exit(-1);
    }
    key1 = atoi(argv[1]);
    if (key1 < -1) {
        printf("Wrong number of threads");
        exit(-1);
    }
    if (key1 > omp_get_max_threads()) {
        printf("Too many number of threads");
        exit(-1);
    }
    if (key1 == -1) key1 = 1;
    if (key1 == 0) key1 = omp_get_max_threads();

    FILE* inp = fopen(argv[2], "rt");
    if (!inp) {
        printf("Can`t open input file");
        exit(-1);
    }
    if (fscanf(inp, "%f%d", &R, &cnt) != 2) {
        printf("Error reading input file");
        exit(-1);
    }
    if ((R <= 0) || (cnt <= 0)) {
        printf("Wrong input parameters");
        exit(-1);
    }
    fclose(inp);
    omp_set_num_threads(key1);
    double tstart = omp_get_wtime();
    int hitcnt = 0;
#pragma omp parallel
    {
        srand(int(time(NULL)) ^ omp_get_thread_num());
#pragma omp for schedule(static)
        for (int i = 1; i < cnt; i++) {
            double x = ((double)rand() / 32767) * 2 - 1;
            double y = ((double)rand() / 32767) * 2 - 1;
            bool hit = (x * x + y * y) <= 1;
            if (hit) {
#pragma omp atomic
                hitcnt += 1;
            }
        }
    }
    double tend = omp_get_wtime();
    printf("Time(%i thread(s)): %g ms\n", key1, (tend - tstart) * 1000);
    FILE* outp = fopen(argv[3], "wt");
    fprintf(outp, "%f", (double)hitcnt / cnt * R * R * 4);
    fclose(outp);
}
```