

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школы электроники и компьютерных наук
Кафедра системного программирования

ОТЧЕТ
о лабораторной работе №3.2
по дисциплине «Технологии параллельного программирования»

Выполнил:
студент группы КЭ-220
_____/Голенищев А. Б.
_____ 2024 г.

Отчет принял:
_____/Жулев А. Э.
_____ 2024 г.

Челябинск
2024

Задание 9. Параллельные секции в OpenMP: программа «I'm here»

Приведен код программы с параллельными секциями, листинг 1. Продемонстрирована ее работа, рисунок 1.

```
#include <stdio.h>
#include <omp.h>
// Golenishchev Artem, KE-220 Task 9
int main() {
    int k;
    printf("Enter the number of threads: ");
    scanf("%d", &k);
    omp_set_num_threads(k);

    #pragma omp parallel
    {
        // Вывод сообщения о параллельной области
        #pragma omp critical
            printf("[%d]: parallel region\n", omp_get_thread_num());

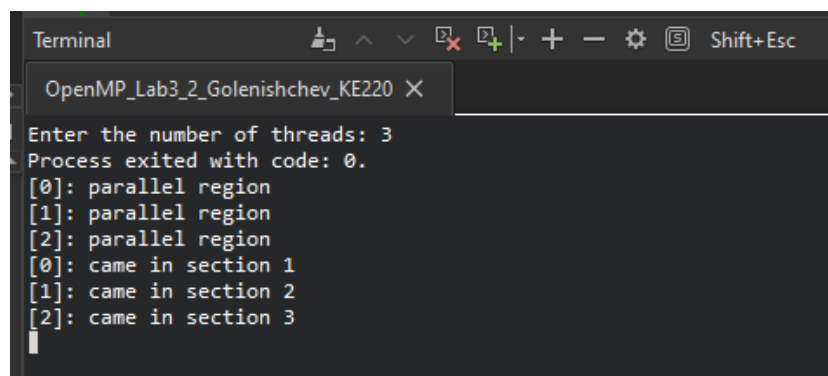
        #pragma omp barrier // Синхронизация перед секциями

        #pragma omp sections
        {
            #pragma omp section
                printf("[%d]: came in section 1\n", omp_get_thread_num());

            #pragma omp section
                printf("[%d]: came in section 2\n", omp_get_thread_num());

            #pragma omp section
                printf("[%d]: came in section 3\n", omp_get_thread_num());
        }
    }
    return 0;
}
```

Листинг 1. Код программы для подсчета суммы чисел



```
Terminal
OpenMP_Lab3_2_Golenishchev_KE220 X
Enter the number of threads: 3
Process exited with code: 0.
[0]: parallel region
[1]: parallel region
[2]: parallel region
[0]: came in section 1
[1]: came in section 2
[2]: came in section 3
```

Рисунок 1. Результат выполнения программы

Программа демонстрирует использование параллельных секций в OpenMP для выполнения независимых блоков кода с использованием нескольких потоков. Сначала пользователь вводит количество потоков, после чего создается параллельная область с заданным числом потоков. В начале параллельной области каждый поток выводит сообщение [`<номер нити>`]: `parallel region`, указывающее, что он участвует в выполнении этой области. Затем выполняется директива `#pragma omp sections`, содержащая три независимые секции, каждая из которых выполняется одним потоком. Внутри каждой секции выводится сообщение [`<номер нити>`]: `came in section <номер секции>`, указывающее номер потока и выполняемой секции. OpenMP автоматически распределяет потоки по секциям, причем порядок выполнения секций и потоков не фиксирован и может различаться между запусками. Если потоков больше, чем секций, избыточные потоки остаются неактивными, а если потоков меньше, одному потоку может быть назначено выполнение нескольких секций. Таким образом, программа демонстрирует гибкость и простоту распараллеливания независимых задач с помощью OpenMP.

Ответы на вопросы к лабораторной работе:

1. Для чего предназначена директива sections?

Директива sections в OpenMP используется для определения набора независимых блоков кода (секций), которые могут выполняться одновременно разными потоками. Она предназначена для распараллеливания задач, которые не зависят друг от друга и могут быть выполнены в произвольном порядке. Каждая секция определяется с помощью директивы section внутри блока sections.

2. В какой момент секции кода распределяются по нитям?

Секции кода распределяются по нитям в момент выполнения программы, когда потоки достигают директивы sections. OpenMP динамически решает, какой поток будет выполнять ту или иную секцию, основываясь на текущем количестве доступных потоков и необходимости завершения всех секций.

3. Почему в OpenMP не реализовано статическое распределение секций? Как это можно сделать?

В OpenMP статическое распределение секций не реализовано, поскольку стандарт не предполагает жесткой привязки секций к потокам. Это сделано для повышения гибкости и эффективности выполнения программы на системах с разным числом потоков. Однако статическое распределение можно реализовать вручную, добавив условия внутри секций, чтобы определенные потоки выполняли строго назначенные им задачи, например, используя идентификатор потока `omp_get_thread_num()`.

4. В каком порядке нити получают секции кода?

Порядок, в котором нити получают секции, не определен стандартом OpenMP и зависит от реализации компилятора и состояния выполнения программы. Это означает, что при каждом запуске программы потоки могут получать секции в разном порядке, что обеспечивает динамическую адаптацию к нагрузке.

5. Получат ли нити одинаковое количество секций?

Нет, нити не гарантированно получают одинаковое количество секций. Если число секций меньше числа потоков, некоторые потоки могут остаться без задач. Если число секций больше числа потоков, одному потоку может быть назначено выполнение нескольких секций, что зависит от внутреннего механизма распределения задач в OpenMP.

Выводы:

Изучили применение директивы `sections` в OpenMP для распараллеливания независимых участков кода. Мы разобрались, как задавать секции с помощью директивы `section` и обеспечивать их выполнение потоками в параллельной области. В ходе экспериментов мы выяснили, что распределение секций между потоками происходит динамически во время выполнения программы, а порядок и количество секций, назначенных потокам, не гарантированы стандартом и зависят от реализации OpenMP. Также мы рассмотрели способы ручного управления распределением задач, что позволяет адаптировать выполнение под специфические требования. Работа позволила понять основные принципы распараллеливания независимых задач для оптимизации программ.