Введение в OpenMP (практика)

Параллельные и последовательные области

#pragma omp firstprivate(a), lastprivate(b)

#pragma omp for firstprivate(a), lastprivate(b)

#pragma omp sections firstprivate(a), lastprivate(b)

Новые опции:

firstprivate(a) – для переменной “a” порождается локальный экземпляр в каждой нити. Начальное значение этих локальных экземпляров (копий) переменных равно значению этой переменной в нити-мастере. Остается верным и для списка переменных.

lastprivate(b) – переменной “b” в нити-мастере будет присвоено значение переменной, полученной последней нитью/в последней итерации цикла/ в последней секции. Остается верным и для списка переменных.

Параллельные и последовательные области

#pragma omp single copyprivate(a) nowait

// участок кода в параллельной области, который должен быть выполнен один раз. Не определено, какой именно нитью код будет выполнен. После выполнения этого участка происходит неявная барьерная синхронизация.

Новые опции:

copyprivate(a) – после выполнения нити новое значение переменной “a” будет скопировано всем переменным “a” других нитей. Не может использоваться совместно с опцией nowait.

nowait – если в синхронизации нет необходимости, т.е. нить в конце участка single не ожидает синхронизации с другими потоками. Служит для увеличения производительности.

#pragma omp master

// участок кода в параллельной области, который будет выполнен только нитью-мастером. Остальные нити это участок кода пропускают. Неявной синхронизации нет.

Распределение работы в циклах

#pragma omp for collapse(n) nowait schedule(тип, объем)

Новые опции:

collapse(n) – n вложенных циклов (т.е. кол-во циклов) ассоциируются с директивой. Для циклов образуется общий объем итераций, который равномерно распределяется между потоками. Если опция не задана, то директива относится только ко внешнему циклу.

nowait – в конце цикла происходит неявная синхронизация нитей, т.е. нити ожидают, пока все потоки не выполнят до конца итерации и не достигнут этой точки. Если в ожидании нет необходимости данная опция позволяется нитям продолжать работу без ожидания остальных потоков.

Распределение работы в циклах

#pragma omp for collapse(n) nowait schedule(тип, объем)

Новые опции:

schedule(тип, объем) – задает шаблон, по которому итерации распределяются между нитями. “Тип” задает правило распределения итераций:

static – блочно-циклическое распределение итераций. Размер блока задается параметром “объем”. Первый блок из “объем” итераций выполняет нить 0, второй блок – нить 1, и т.д. до последней нити, затем распределение снова начинается с нити 0.

dynamic – динамическое распределение итераций с фиксированным размером блока. Каждая нить получает “объем” итераций, затем каждая освободившаяся нить получает первую свободную порцию итераций. И так далее, пока все порции не будут исчерпаны. Последняя порция может содержать меньше итераций, чем остальные.

Параллелизм на уровне команд
Директива sections

```
#pragma omp parallel private(myid)
{
myid = omp_get_thread_num();
#pragma omp sections
    {
#pragma omp section
        {
            printf("First section! Myid = %d \n", myid);
        }
#pragma omp section
        {
            printf("Second section! Myid = %d \n", myid);
        }
#pragma omp section
        {
            printf("Third section! Myid = %d \n", myid);
        }
    } //#pragma omp sections
printf("Parallel domain now! Myid = %d \n", myid);
} //#pragma omp parallel
```

Директива определяет набор независимых областей программы, каждая из которых выполняется отдельной нитью.

Директива **section** описывает участок программы внутри области **sections** для выполнения одной нитью.

Если кол-во нитей больше кол-ва секций, то часть нитей задействована не будет.

Если кол-во нитей меньше кол-ва секций, то некоторым нитям достанется больше одной секции.

Параллелизм на уровне команд
Директива task 1

Задача (task) помещается внутрь параллельной области, и задает блок операторов, который может выполняться в отдельном потоке. Задача не создает новый поток. Задача (блок операторов) помещается в пул, из которого ее может взять один из свободных потоков для выполнения.

Можно отметить, что директива omp task имеет необязательную опцию if, задающую условие. Если условие истинно – будет создана задача и добавлена в пул, как описано выше, если же условие ложно – задача создана не будет, а ассоциированный с ней блок операторов будет немедленно выполнен в текущем потоке.

Для синхронизации задач используется директива taskwait. Поток не будет выполнять код, размещенный после taskwait до тех пор, пока не будут выполнены все созданные этим потоком задачи.

Параллелизм на уровне команд
Директива task 2

```
1 #pragma omp parallel num_threads(4)
2           {
3                       #pragma omp task
4                       {
5                                   printf("Running task. My id%d \n",omp_get_thread_num());
6                       } //#pragma omp task
7                       #pragma omp taskwait
8                       printf("Bye! %d \n", omp_get_thread_num());
9 }     //#pragma omp parallel
```

В строке 2 порождается 4 нити посредством опции num_threads(4). Каждый поток помещает в пул задачу (строки 3-6), ожидает завершения своей задачи (строка 7), выводит на экран слово “Bye!” (строка 8). Задачи из пула берутся первой освободившейся нитью.

Параллелизм на уровне команд
Директива task 3 (пример)

```
#pragma omp parallel num_threads(4)
{
        myid = omp_get_thread_num();
        if (myid == 0){
            for (i = 1; i <= 4; i++){
                #pragma omp task
                {
                   printf("Running task. My id%d \n",omp_get_thread_num());
                }
            }
        }
        #pragma omp taskwait
        printf("Bye! %d \n", omp_get_thread_num());
}
```