[OpenMP中的任务调度](http://blog.csdn.net/drzhouweiming/article/details/1844762)

标签： [任务调度](http://www.csdn.net/tag/%e4%bb%bb%e5%8a%a1%e8%b0%83%e5%ba%a6)[thread](http://www.csdn.net/tag/thread)[parallel](http://www.csdn.net/tag/parallel)[多线程](http://www.csdn.net/tag/%e5%a4%9a%e7%ba%bf%e7%a8%8b)[编译器](http://www.csdn.net/tag/%e7%bc%96%e8%af%91%e5%99%a8)[windows](http://www.csdn.net/tag/windows)

2007-10-26 12:31 21238人阅读 [评论](http://blog.csdn.net/drzhouweiming/article/details/1844762#comments)(3) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/drzhouweiming/article/details/1844762#report)

版权声明：本文为博主原创文章，未经博主允许不得转载。

**OpenMP中的任务调度**

OpenMP中，任务调度主要用于并行的for循环中，当循环中每次迭代的计算量不相等时，如果简单地给各个线程分配相同次数的迭代的话，会造成各个线程计算负载不均衡，这会使得有些线程先执行完，有些后执行完，造成某些CPU核空闲，影响程序性能。例如以下代码：

int i, j;

int a[100][100] = {0};

for ( i =0; i < 100; i++)

{

for( j = i; j < 100; j++ )

{

    a[i][j] = i\*j;

}

}

如果将最外层循环并行化的话，比如使用4个线程，如果给每个线程平均分配25次循环迭代计算的话，显然i＝0和i＝99的计算量相差了100倍，那么各个线程间可能出现较大的负载不平衡情况。为了解决这些问题，OpenMP中提供了几种对for循环并行化的任务调度方案。

在OpenMP中，对for循环并行化的任务调度使用schedule子句来实现，下面介绍schedule字句的用法。

**1.1.1**       **Schedule子句用法**

schedule子句的使用格式为：

**schedule(type[,size])**

schedule有两个参数：type和size，size参数是可选的。

1.       type参数

表示调度类型，有四种调度类型如下：

· **dynamic**

· **guided**

· **runtime**

· **static**

这四种调度类型实际上只有static、dynamic、guided三种调度方式，runtime实际上是根据环境变量来选择前三种中的某中类型。

*run-sched-var*

2.       size参数 (可选)

size参数表示循环迭代次数，size参数必须是整数。static、dynamic、guided三种调度方式都可以使用size参数，也可以不使用size参数。当type参数类型为runtime时，size参数是非法的（不需要使用，如果使用的话编译器会报错）。

**1.1.2**       **静态调度(static)**

当parallel for编译指导语句没有带schedule子句时，大部分系统中默认采用static调度方式，这种调度方式非常简单。假设有n次循环迭代，t个线程，那么给每个线程静态分配大约n/t次迭代计算。这里为什么说大约分配n/t次呢？因为n/t不一定是整数，因此实际分配的迭代次数可能存在差1的情况，如果指定了size参数的话，那么可能相差一个size。

静态调度时可以不使用size参数，也可以使用size参数。

3.       不使用size参数

不使用size参数时，分配给每个线程的是n/t次连续的迭代，不使用size参数的用法如下：

schedule(static)

例如以下代码:

    #pragma omp parallel for schedule(static)

    for(i = 0; i < 10; i++ )

    {

         printf("i=%d, thread\_id=%d/n", i, omp\_get\_thread\_num());

}

上面代码执行时打印的结果如下：

i=0, thread\_id=0

i=1, thread\_id=0

i=2, thread\_id=0

i=3, thread\_id=0

i=4, thread\_id=0

i=5, thread\_id=1

i=6, thread\_id=1

i=7, thread\_id=1

i=8, thread\_id=1

i=9, thread\_id=1

可以看出线程0得到了0～4次连续迭代，线程1得到5～9次连续迭代。注意由于多线程执行时序的随机性，每次执行时打印的结果顺序可能存在差别，后面的例子也一样。

4.       使用size参数

使用size参数时，分配给每个线程的size次连续的迭代计算，用法如下：

schedule(static, size)

例如以下代码:

    #pragma omp parallel for schedule(static, 2)

    for(i = 0; i < 10; i++ )

    {

         printf("i=%d, thread\_id=%d/n", i, omp\_get\_thread\_num());

}

执行时会打印以下结果：

i=0, thread\_id=0

i=1, thread\_id=0

i=4, thread\_id=0

i=5, thread\_id=0

i=8, thread\_id=0

i=9, thread\_id=0

i=2, thread\_id=1

i=3, thread\_id=1

i=6, thread\_id=1

i=7, thread\_id=1

从打印结果可以看出，0、1次迭代分配给线程0，2、3次迭代分配给线程1，4、5次迭代分配给线程0，6、7次迭代分配给线程1，…。每个线程依次分配到2次连续的迭代计算。

**1.1.3**       **动态调度(dynamic)**

动态调度是动态地将迭代分配到各个线程，动态调度可以使用size参数也可以不使用size参数，不使用size参数时是将迭代逐个地分配到各个线程，使用size参数时，每次分配给线程的迭代次数为指定的size次。

下面为使用动态调度不带size参数的例子：

         #pragma omp parallel for schedule(dynamic)

         for(i = 0; i < 10; i++ )

         {

                  printf("i=%d, thread\_id=%d/n", i, omp\_get\_thread\_num());

         }

打印结果如下：

i=0, thread\_id=0

i=1, thread\_id=1

i=2, thread\_id=0

i=3, thread\_id=1

i=5, thread\_id=1

i=6, thread\_id=1

i=7, thread\_id=1

i=8, thread\_id=1

i=4, thread\_id=0

i=9, thread\_id=1

下面为动态调度使用size参数的例子：

         #pragma omp parallel for schedule(dynamic, 2)

         for(i = 0; i < 10; i++ )

         {

                  printf("i=%d, thread\_id=%d/n", i, omp\_get\_thread\_num());

         }

打印结果如下：

i=0, thread\_id=0

i=1, thread\_id=0

i=4, thread\_id=0

i=2, thread\_id=1

i=5, thread\_id=0

i=3, thread\_id=1

i=6, thread\_id=0

i=8, thread\_id=1

i=7, thread\_id=0

i=9, thread\_id=1

从打印结果可以看出第0、1，4、5，6、7次迭代被分配给了线程0，第2、3，8、9次迭代则分配给了线程1，每次分配的迭代次数为2。

**1.1.4**       **guided调度（guided）**

guided调度是一种采用指导性的启发式自调度方法。开始时每个线程会分配到较大的迭代块，之后分配到的迭代块会逐渐递减。迭代块的大小会按指数级下降到指定的size大小，如果没有指定size参数，那么迭代块大小最小会降到1。

例如以下代码：

    #pragma omp parallel for schedule(guided,2)

    for(i = 0; i < 10; i++ )

    {

         printf("i=%d, thread\_id=%d/n", i, omp\_get\_thread\_num());

}

打印结果如下：

i=0, thread\_id=0

i=1, thread\_id=0

i=2, thread\_id=0

i=3, thread\_id=0

i=4, thread\_id=0

i=8, thread\_id=0

i=9, thread\_id=0

i=5, thread\_id=1

i=6, thread\_id=1

i=7, thread\_id=1

第0、1、2、3、4次迭代被分配给线程0，第5、6、7次迭代被分配给线程1，第8、9次迭代被分配给线程0，分配的迭代次数呈递减趋势，最后一次递减到2次。

**1.1.5**       **runtime调度（rumtime）**

runtime调度并不是和前面三种调度方式似的真实调度方式，它是在运行时根据环境变量OMP\_SCHEDULE来确定调度类型，最终使用的调度类型仍然是上述三种调度方式中的某种。

例如在unix系统中，可以使用setenv命令来设置OMP\_SCHEDULE环境变量：

setenv OMP\_SCHEDULE “dynamic, 2”

上述命令设置调度类型为动态调度，动态调度的迭代次数为2。

在windows环境中，可以在”系统属性|高级|环境变量”对话框中进行设置环境变量。