

原 OpenMP并行构造的schedule子句详解

2011年11月22日 23:22:51

阅读数：13129

schedule子句是专门为循环并行构造的时候使用的子句，只能用于循环并行构造（parallel for）中。

根据OpenMP Spec (<http://openmp.org/mp-documents/OpenMP3.1-CCard.pdf>) 可以知道：

schedule的语法为：

schedule(kind[, chunk_size])

kind:

- static: Iterations are divided into chunks of size chunk_size. Chunks are assigned to threads in the team in round-robin fashion in order of thread number.
- dynamic: Each thread executes a chunk of iterations then requests another chunk until no chunks remain to be distributed.
- guided: Each thread executes a chunk of iterations then requests another chunk until no chunks remain to be assigned. The chunk sizes start large and shrink to the indicated chunk_size as chunks are scheduled.
- auto: The decision regarding scheduling is delegated to the compiler and/or runtime system.
- runtime: The schedule and chunk size are taken from the run-sched-var ICV

(1)schedule的作用：

上面知道，schedule只能用于循环并行构造中，其作用是用于控制循环并行结构的任务调度。一个简单的理解，一个for循环假设有10此迭代，使用4个线程去执行，那么哪些线程去执行哪些迭代呢？可以通过schedule去控制迭代的调度和分配，从而适应不同的使用情况，提高性能。

(1) schedule语法：

schedule(kind [, chunk_size])，其中kind可以取值为static, dynamic, guided, auto, runtime。其中，测试发现，auto貌似是不行的（可能是Fortran才支持？不清楚）。参数chunk_size是一个size的参数，所以是整数了。这里的runtime在后面分析，其实本质上是前面三种类型之一。

所以，schedule的类型一共是三种：static、dynamic、guided。下面先逐一分析这三种，最后再了解runtime的含义。

(2) 静态调度static：

大部分的编译器实现，在没有使用schedule子句的时候，系统就是采用static方式调度的。

对于schedule(static,size)的含义，**OpenMP会给每个线程分配size次迭代计算。这个分配是静态的，“静态”体现在这个分配过程跟实际的运行是无关的，可以从逻辑上推断出哪几次迭代会在哪几个线程上运行。**具体而言，对于一个N次迭代，使用M个线程，那么，[0,size-1]的size次的迭代是在第一个线程上运行，[size, size + size -1]是在第二个线程上运行，依次类推。那么，如果M太大，size也很大，就可能出现很多个迭代在一个线程上运行，而某些线程不执行任何迭代。需要说明的是，这个分配过程就是这样确定的，不会因为运行的情况改变，比如，我们知道，进入OpenMP后，假设有M个线程，这M个线程开始执行的时间不一定是一样的，这是由OpenMP去调度的，并不会因为某一个线程先被启动，而去改变for的迭代的分配，这就是静态的含义。分析下面的例子：

```
1  #include <omp.h>
2  #define COUNT 4*3
3
4  int main(int argc, _TCHAR* argv[])
5  {
6  #pragma omp parallel for schedule(static,4)
7      for(int i = 0; i < COUNT; i++)
8      {
```

```

12 |         return 0;
13 |     }

```

下面是其中多次运行的几次结果（四核系统，所以线程数量为4）：

Thread: 0, Iteration: 0	Thread: 0, Iteration: 0	Thread: 1, Iteration: 4
Thread: 0, Iteration: 1	Thread: 0, Iteration: 1	Thread: 1, Iteration: 5
Thread: 0, Iteration: 2	Thread: 0, Iteration: 2	Thread: 0, Iteration: 0
Thread: 0, Iteration: 3	Thread: 0, Iteration: 3	Thread: 0, Iteration: 1
Thread: 1, Iteration: 4	Thread: 2, Iteration: 8	Thread: 0, Iteration: 2
Thread: 1, Iteration: 5	Thread: 2, Iteration: 9	Thread: 0, Iteration: 3
Thread: 1, Iteration: 6	Thread: 2, Iteration: 10	Thread: 1, Iteration: 6
Thread: 1, Iteration: 7	Thread: 2, Iteration: 11	Thread: 1, Iteration: 7
Thread: 2, Iteration: 8	Thread: 1, Iteration: 4	Thread: 2, Iteration: 8
Thread: 2, Iteration: 9	Thread: 1, Iteration: 5	Thread: 2, Iteration: 9
Thread: 2, Iteration: 10	Thread: 1, Iteration: 6	Thread: 2, Iteration: 10
Thread: 2, Iteration: 11	Thread: 1, Iteration: 7	Thread: 2, Iteration: 11
Press any key to continue . . .	Press any key to continue . . .	Press any key to continue . . .

从左到右假设为结果1/2/3，从结果可以看到，无论是哪一个线程先启动，team内的ID为0的线程，总是会执行0，1，2，3对应的迭代，team内ID为1的线程，总是会执行4，5，6，7对应的迭代，team内ID为2的线程，总是会执行8，9，10，11的线程，而team内线程ID为4的线程，并没有去执行任何迭代。如果把上面的size的大小改为12，那么无论如何，所有的迭代都只会在线程0上执行，其实就跟串行的效果一样了。对于这里的情况，4个线程，却只使用了3个线程去计算，所以这样分配当然是不平衡的。

上面是针对给定size的情况，如果不指定size，只是指定static类型，那么OpenMP为使用迭代数/线程数作为size的值，采取同样的策略来进行分配，这样每个线程执行的迭代数目就是一样的（注意，如果迭代数/线程数不是整除的，那就不完全一样了，但是整体是比较平衡的），一般而言，这就是不加任何schedule修饰下的调度情况了。

(3) 动态调度dynamic

动态调度迭代的分配是依赖于运行状态进行动态确定的，所以哪个线程上将会运行哪些迭代是无法像静态一样事先预料的。**对于dynamic，没有size参数的情况下，每个线程按先执行完先分配的方式执行1次循环**，比如，刚开始，线程1先启动，那么会为线程1分配一次循环开始去执行（i=0的迭代），然后，可能线程2启动了，那么为线程2分配一次循环去执行（i=1的迭代），假设这时候线程0和线程3没有启动，而线程1的迭代已经执行完，可能会继续为线程1分配一次迭代，如果线程0或3先启动了，可能会为之分配一次迭代，直到把所有的迭代分配完。所以，动态分配的结果是无法事先知道的，因为我们无法知道哪一个线程会先启动，哪一个线程执行某一个迭代需要多久等等，这些都是取决于系统的资源、线程的调度等等。看下面的例子：

```

1 | #include <omp.h>
2 | #include <Windows.h>
3 | #define COUNT 4*3
4 |
5 | int main(int argc, _TCHAR* argv[])
6 | {
7 |     #pragma omp parallel for schedule(dynamic)
8 |         for(int i = 0; i < COUNT; i++)
9 |         {
10 |             printf("Thread: %d, Iteration: %d\n", omp_get_thread_num(), i);
11 |         }
12 |
13 |     return 0;
14 | }

```

下面也是多次运行的结果：

Thread: 0, Iteration: 0	Thread: 1, Iteration: 1	Thread: 0, Iteration: 0
Thread: 0, Iteration: 2	Thread: 1, Iteration: 3	Thread: 0, Iteration: 1
Thread: 0, Iteration: 3	Thread: 1, Iteration: 5	Thread: 0, Iteration: 3
Thread: 0, Iteration: 6	Thread: 2, Iteration: 2	Thread: 0, Iteration: 4
Thread: 0, Iteration: 7	Thread: 2, Iteration: 7	Thread: 0, Iteration: 5
Thread: 0, Iteration: 8	Thread: 0, Iteration: 0	Thread: 1, Iteration: 2
Thread: 0, Iteration: 9	Thread: 0, Iteration: 9	Thread: 1, Iteration: 7
Thread: 0, Iteration: 10	Thread: 0, Iteration: 10	Thread: 1, Iteration: 8
Thread: 0, Iteration: 11	Thread: 0, Iteration: 11	Thread: 1, Iteration: 9
Thread: 2, Iteration: 1	Thread: 3, Iteration: 4	Thread: 3, Iteration: 10
Thread: 3, Iteration: 4	Thread: 1, Iteration: 6	Thread: 1, Iteration: 11
Thread: 1, Iteration: 5	Thread: 2, Iteration: 8	Thread: 0, Iteration: 6
Press any key to continue . . .	Press any key to continue . . .	Press any key to continue . . .

事实上，这个结果容易给人误解，以结果2来分析，为什么线程1先执行完，其对应的迭代却是1而不是0呢？而且接下来也是3而不是2呢？不是先执行完的先分配么？我的理解是，这里，刚开始的时候，我们不知道线程的状态，实际的一种可能是，线程0先启动了，所以会去执行迭代0里面的内容，但是，只是**开始**去执行，而这里用printf测试，是输出的结果，输出的顺序不代表开始执行此迭代的顺序，所以可能的

0, 这中间, 线程2可能会分配了迭代2**开始**执行, 线程1又获得了分配机会, 被分配了迭代3等等。。。总之, 这里的输出顺序是不代表每一个迭代开始被分配执行的时间顺序的。总之, 理解这个动态的过程, 简单理解, 就是“谁有空, 给谁分配一次迭代让它去跑!” :)

那么同样, dynamic也可以有一个size参数, size表示, 每次线程执行完 (空闲) 的时候给其一次分配的迭代的数量, 如果没有知道size (上面的分析), 那么每次就分配一个迭代。有了前面的理解, 这个size的含义是很容易理解的了。

(4) guided调度

类似于动态调度, 但每次分配的循环次数不同, 开始比较大, 以后逐渐减小。size表示每次分配的迭代次数的最小值, 由于每次分配的迭代次数会逐渐减少, 较少到size时, 将不再减少。如果不知道size的大小, 那么默认size为1, 即一直减少到1。具体是如何减少的, 以及开始比较大 (具体是多少?), 参考相关手册的信息。

(5) runtime

runtime表示根据环境变量确定上述调度策略中的某一种, 默认也是静态的 (static) 。

控制schedule环境变量的是OMP_SCHEDULE环境变量, 其值和上面的三中类型一样了, 比如:

```
setenv OMP_SCHEDULE "dynamic, 5"
```

就是schedule(dynamic,5)的含义了。

文章标签: [thread](#) [parallel](#) [fortran](#) [任务调度](#) [compiler](#) [each](#)

个人分类: [并行计算高性能计算HPC](#) — [OpenMP](#)

AI人工智能工程师

就业薪资高
助力转型高端岗位

[领取课程大纲](#)

【CSDN学院】转型拿高薪！什么样的人适合转型做

四个月学会深度学习加机器学习，快速进入人工智能领域！拼一次赢一生，CSDN学院助力转型高端岗位！

[马上了解 >>](#)

想对作者说点什么？ [我来说一句](#)

OpenMP中的任务调度----schedule()

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...

 freeboy1015 2012-03-19 16:05:25 阅读数: 3884

OpenMP

OpenMP 2008-8-10 version 1.0 1 简介 www.openmp.org GNU的gomp项目; Include ; 编译参数-fopenmp打...

 horizons_kong 2016-12-27 14:42:32 阅读数: 299


并行编程OpenMP基础及简单示例

OpenMP基本概念 OpenMP是一种用于共享内存**并行**系统的多线程程序设计方案, 支持的编程语言包括C、C++和Fortran。OpenMP提供了对**并行**算法的高层抽象描述, 特别适合在多核CPU机器...

 dcrmng 2016-12-24 22:30:16 阅读数: 11425

OpenMP并行程序设计——for循环并行化详解

在C/C++中使用OpenMP优化代码方便又简单, 代码中需要**并行**处理的往往是一些比较耗时的for循环, 所以重点介绍一下OpenMP中for循环的应用。个人感觉只要掌握了文中讲的这些就足够了, 如果想要学...

 zhongkejingwang 2014-10-22 18:12:19 阅读数: 15837

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...

 drzhouweiming 2007-10-26 12:31:00 阅读数：27198

OpenMP 参考（指令详解）

共享工作（Work-Sharing）结构 共享工作结构将它作用的代码段在抵达此代码段的线程中分开执行Work-sharing 不会产生新线程进入工作共享结构时没有关卡，但结束时会有 Work...

 saga1979 2011-03-21 17:53:00 阅读数：11961

OpenMP中omp_set_dynamic()和OMP_DYNAMIC环境变量详解

理解这部分内容之前，先要理解omp_get_num_threads()和omp_get_max_threads()的含义和区别，参考：<http://blog.csdn.net/gengshenghon...>

 Augustdi 2013-04-16 10:58:24 阅读数：3192

openmp 任务调度 for schedule static dynamic guided runtime

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**的用法。**schedule**的使用格式为：**schedule**(type[,size]) schedu...

 billbliss 2015-03-08 11:48:47 阅读数：1344

OpenMP学习笔记

原文：http://blog.sina.com.cn/s/blog_66474b160100z15b.html 1. OpenMP是一种API，用于编写可移植的多线程应用程序，无需程序员进...

 zsc09_leaf 2012-07-17 10:07:38 阅读数：32830

定时任务schedule的二种用法

时间单位是毫秒 **schedule**(TimerTask task, long delay); **schedule**(TimerTask task, long delay, long period);...

 weixin_40369725 2017-10-14 23:57:55 阅读数：115

Schedule用法实例

一、包 import org.quartz.CronTrigger; import org.quartz.DateBuilder; import org.quartz.JobBuilder; imp...

 unimme 2015-11-09 13:57:12 阅读数：3625

使用OpenMP给程序加速

最近面试总是谈到效率问题，这个问题以前一直没考虑过，就是稀里糊涂的写。之前有看到过OpenMP，也不曾深究，看到这篇博客关于OpenMP写的非常详细，就转来慢慢学习吧。OpenMP语法简介...

 CSDNMicrosoftCSDN 2015-01-29 20:53:52 阅读数：3997

#pragma omp parallel for schedule(dynamic) private(i)

```
#include #include #include int main() { int i; #pragma omp parallel for sch...
```

 adream307 2011-06-21 12:35:00 阅读数：2921

软件加密，保护软件不被破解

支持unity3D加密,防止代码反编译,防止资源被拷贝!



openmp 快速入门 常用技巧 parallel for sections reduction critical

#pragma omp parallel 自动将下面语句执行N次，(N为电脑CPU核数)，然后把每份指派给一个核去执行，而且多核之间为**并行**执行。#pragma omp parallel for **并行**...

 billbliss 2015-03-08 11:05:28 阅读数：4030

vs上C/C++并行计算#pragma omp

在一个vs内的工程进行**并行**计算，首先先修改属性内的C/C++ — 语言—OpenMP:是。头文件不一定需要#include #pragma omp parallel sections//告诉编译器有几...

【并行计算】基于OpenMP的并行编程（#pragma omp parallel for）

百度云盘：MPI 并行计算教材 <https://pan.baidu.com/s/1htgGZh6> 密码：bt1p 我们目前的计算机都是基于冯诺伊曼结构的，在MIMD作为主要研究对象的系统中，分为...

 eric_e 2018-01-24 22:46:06 阅读数：637

openmp在多重循环内的简单使用及其详解

由于项目需求，在三重循环内加入了并行计算，但由于只能在内层循环加入，而内层循环只有32维度，因此速度提高的也就那么几毫秒。在此 不再将代码贴出！以下是转载的别人博客中的详细讲解，很不错！ ...

 Allyii0022 2016-09-29 15:44:40 阅读数：6045

OpenMP中的数据处理子句

OpenMP中的数据处理子句相关文档连接：多核编程中的任务随机竞争模式的概率分析多核编程中的任务分组竞争模式 多核编程中的负载均衡难题 多核编程中的锁竞争难题 多核编程的几个难题及其应对策略（难题...

 drzhouweiming 2008-01-10 11:02:00 阅读数：32062

Docker快速入门基础教程

Docker




OpenMP并行程序设计（二）

OpenMP并行程序设计（二） ... 11、fork/join并行执行模式的概念... 12、OpenMP指令和库函数介绍... 13、parallel 指令的用法... 34、for指令的使用方法...

 drzhouweiming 2006-09-04 15:31:00 阅读数：69307

OpenMP编程入门之一

当前多核多线程CPU大行其道，如果不能充分利用岂不是太可惜了！特别在图像处理领域，简直是为并行计算而生的！在网上看了不少文章，还是自己总结一下吧。...

 wyjkk 2011-07-17 14:32:41 阅读数：11282