

OpenMP并行程序设计——for循环并行化详解

2014年10月22日 18:12:19

阅读数：15838

转载请声明出处<http://blog.csdn.net/zhongkejingwang/article/details/40018735>

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，如果想要学习OpenMP可以到网上查查资料。

工欲善其事，必先利其器。如果还没有搭建好omp开发环境的可以看一下[OpenMP并行程序设计——Eclipse开发环境的搭建](#)

首先，如何使一段代码并行处理呢？omp中使用parallel制导指令标识代码中的并行段，形式为：

```
#pragma omp parallel

{

    每个线程都会执行大括号里的代码

}
```

比如下面这段代码：

```
1 | #include <iostream>
2 | #include "omp.h"
3 | using namespace std;
4 | int main(int argc, char **argv) {
5 |     //设置线程数，一般设置的线程数不超过CPU核心数，这里开4个线程执行并行代码段
6 |     omp_set_num_threads(4);
7 | #pragma omp parallel
8 |     {
9 |         cout << "Hello" << ", I am Thread " << omp_get_thread_num() << endl;
10 |     }
11 | }
```

omp_get_thread_num()是获取当前线程id号

以上代码执行结果为：

```
Hello, I am Thread 1
Hello, I am Thread 0
Hello, I am Thread 2
Hello, I am Thread 3
```

可以看到，四个线程都执行了大括号里的代码，先后顺序不确定，这就是一个并行块。

带有for的制导指令：

for制导语句是将for循环分配给各个线程执行，这里要求数据不存在依赖。

使用形式为：

```
for()
```

(2) #pragma omp parallel

{//注意：大括号必须要另起一行

```
#pragma omp for
```

```
for()
```

```
}
```

注意：第二种形式中并行块里面不要再出现parallel制导指令，比如写成这样就不可以：

```
#pragma omp parallel
```

```
{
```

```
#pragma omp parallel for
```

```
for()
```

```
}
```

第一种形式作用域只是紧跟着的那个for循环，而第二种形式在整个并行块中可以出现多个for制导指令。下面结合例子程序讲解for循环并行化需要注意的地方。

假如不使用for制导语句，而直接在for循环前使用parallel语句：（为了使输出不出现混乱，这里使用printf代替cout）

```
1 | #include <iostream>
2 | #include <stdio.h>
3 | #include "omp.h"
4 | using namespace std;
5 | int main(int argc, char **argv) {
6 |     //设置线程数，一般设置的线程数不超过CPU核心数，这里开4个线程执行并行代码段
7 |     omp_set_num_threads(4);
8 | #pragma omp parallel
9 |     for (int i = 0; i < 2; i++)
10 |         //cout << "i = " << i << ", I am Thread " << omp_get_thread_num() << endl;
11 |         printf("i = %d, I am Thread %d\n", i, omp_get_thread_num());
12 | }
```

输出结果为：

```
i = 0, I am Thread 0
i = 0, I am Thread 1
i = 1, I am Thread 0
i = 1, I am Thread 1
i = 0, I am Thread 2
i = 1, I am Thread 2
i = 0, I am Thread 3
i = 1, I am Thread 3
```

从输出结果可以看到，如果不使用for制导语句，则每个线程都执行整个for循环。所以，使用for制导语句将for循环拆分开来尽可能平均地分配到各个线程执行。将并行代码改成这样之后：

```
1 | #pragma omp parallel for
2 |     for (int i = 0; i < 6; i++)
3 |         printf("i = %d, I am Thread %d\n", i, omp_get_thread_num());
```

输出结果为：

```

3 | i = 0, I am Thread 0
4 | i = 1, I am Thread 0
5 | i = 3, I am Thread 1
6 | i = 5, I am Thread 3

```

可以看到线程0执行i=0和1，线程1执行i=2和3，线程2执行i=4，线程3执行i=5。线程0就是主线程

这样整个for循环被拆分并行执行了。上面的代码中parallel和for连在一块使用的，其只能作用到紧跟着的for循环，循环结束了并行块就退出了。

上面的代码可以改成这样：

```

1 | #pragma omp parallel
2 | {
3 |     #pragma omp for
4 |         for (int i = 0; i < 6; i++)
5 |             printf("i = %d, I am Thread %d\n", i, omp_get_thread_num());
6 | }

```

这写法和上面效果是一样的。需要注意的问题来了：如果在parallel并行块里再出现parallel会怎么样呢？回答这个问题最好的方法就是跑一遍代码看看，所以把代码改成这样：

```

1 | #pragma omp parallel
2 | {
3 |     #pragma omp parallel for
4 |         for (int i = 0; i < 6; i++)
5 |             printf("i = %d, I am Thread %d\n", i, omp_get_thread_num());
6 | }

```

输出结果：

```

1 | i = 0, I am Thread 0
2 | i = 0, I am Thread 0
3 | i = 1, I am Thread 0
4 | i = 1, I am Thread 0
5 | i = 2, I am Thread 0
6 | i = 2, I am Thread 0
7 | i = 3, I am Thread 0
8 | i = 3, I am Thread 0
9 | i = 4, I am Thread 0
10 | i = 4, I am Thread 0
11 | i = 5, I am Thread 0
12 | i = 5, I am Thread 0
13 | i = 0, I am Thread 0
14 | i = 1, I am Thread 0
15 | i = 0, I am Thread 0
16 | i = 2, I am Thread 0
17 | i = 1, I am Thread 0
18 | i = 3, I am Thread 0
19 | i = 2, I am Thread 0
20 | i = 4, I am Thread 0
21 | i = 3, I am Thread 0
22 | i = 5, I am Thread 0
23 | i = 4, I am Thread 0
24 | i = 5, I am Thread 0

```

可以看到，只有一个线程0，也就是只有主线程执行for循环，而且总共执行4次，每次都执行整个for循环！所以，这样写是不对的。

当然，上面说的for制导语句的两种写法是有区别的，比如两个for循环之间有一些代码只能有一个线程执行，那么用第一种方法口要这样就可以了。

```

1 | #pragma omp parallel for 2 |           for (int i = 0; i < 6; i++)
3 |         printf("i = %d, I am Thread %d\n", i, omp_get_thread_num());
4 |         //这里是两个for循环之间的代码, 将会由线程0即主线程执行
5 |         printf("I am Thread %d\n", omp_get_thread_num());
6 | #pragma omp parallel for
7 |     for (int i = 0; i < 6; i++)
8 |         printf("i = %d, I am Thread %d\n", i, omp_get_thread_num());

```

离开了for循环就剩主线程了, 所以两个循环间的代码是由线程0执行的, 输出结果如下:

```

1 | i = 0, I am Thread 0
2 | i = 2, I am Thread 1
3 | i = 1, I am Thread 0
4 | i = 3, I am Thread 1
5 | i = 4, I am Thread 2
6 | i = 5, I am Thread 3
7 | I am Thread 0
8 | i = 4, I am Thread 2
9 | i = 2, I am Thread 1
10 | i = 5, I am Thread 3
11 | i = 0, I am Thread 0
12 | i = 3, I am Thread 1
13 | i = 1, I am Thread 0

```

但是如果用第二种写法把for循环写进parallel并行块中就需要注意了!

由于用parallel标识的并行块中每一行代码都会被多个线程处理, 所以如果想让两个for循环之间的代码由一个线程执行的话就需要在代码前用single或master制导语句标识, master由是主线程执行, single是选一个线程执行, 这个到底选哪个线程不确定。所以上面代码可以写成这样:

```

1 | #pragma omp parallel
2 |     {
3 |     #pragma omp for
4 |         for (int i = 0; i < 6; i++)
5 |             printf("i = %d, I am Thread %d\n", i, omp_get_thread_num());
6 |     #pragma omp master
7 |         {
8 |             //这里的代码由主线程执行
9 |             printf("I am Thread %d\n", omp_get_thread_num());
10 |        }
11 | #pragma omp for
12 |     for (int i = 0; i < 6; i++)
13 |         printf("i = %d, I am Thread %d\n", i, omp_get_thread_num());
14 |     }

```

效果和上面的是一样的, 如果不指定让主线程执行, 那么将master改成single即可。

到这里, parallel和for的用法都讲清楚了。接下来就开始讲并行处理时数据的同步问题, 这是多线程编程里都会遇到的一个问题。

为了讲解数据同步问题, 先由一个例子开始:

```

1 | #include <iostream>
2 | #include "omp.h"
3 | using namespace std;
4 | int main(int argc, char **argv) {
5 |     int n = 100000;
6 |     int sum = 0;
7 |     omp_set_num_threads(4);
8 | #pragma omp parallel
9 |     {

```

```

13 |                                     sum += 1;
14 |                                     }
15 |                                 }
16 |                             }
17 |                             cout << " sum = " << sum << endl;
18 | }

```

期望的正确结果是100000，但是这样写是错误的。看代码，由于默认情况下sum变量是每个线程共享的，所以多个线程同时对sum操作时就会因为数据同步问题导致结果不对，显然，输出结果每次都不同，这是无法预知的，如下：

```

1 | 第一次输出sum = 58544
2 | 第二次输出sum = 77015
3 | 第三次输出sum = 78423

```

那么，怎么去解决这个数据同步问题呢？解决方法如下：

方法一：对操作共享变量的代码段做同步标识

代码修改如下：

```

1 | #pragma omp parallel
2 | {
3 |     #pragma omp for
4 |         for (int i = 0; i < n; i++) {
5 |             {
6 |                 #pragma omp critical
7 |                     sum += 1;
8 |             }
9 |         }
10 |     }
11 |     cout << " sum = " << sum << endl;

```

critical制导语句标识的下一行代码，也可以是跟着一个大括号括起来的代码段做了同步处理。输出结果100000



12



收藏



评论



微信



微博



QQ

方法二：每个线程拷贝一份sum变量，退出并行块时再把各个线程的sum相加

并行代码修改如下：

```

1 | #pragma omp parallel
2 | {
3 |     #pragma omp for reduction(+:sum)
4 |         for (int i = 0; i < n; i++) {
5 |             {
6 |                 sum += 1;
7 |             }
8 |         }
9 |     }

```

reduction制导语句，操作是退出时将各自的sum相加存到外面的那个sum中，所以输出结果就是100000啦~~

方法三：这种方法貌似不那么优雅

代码修改如下：

```

1 | int n = 100000;
2 | int sum[4] = { 0 };
3 | omp_set_num_threads(4);
4 | #pragma omp parallel
5 | {
6 |     #pragma omp for

```

```

9 |                                     sum[omp_get_thread_num()] += 1;
11 |                                     }
12 |                                     }
13 |     cout << " sum = " << sum[0] + sum[1] + sum[2] + sum[3] << endl;

```

每个线程操作的都是以各自线程id标识的数组位置，所以结果当然正确。

数据同步就讲完了，上面的代码中for循环是一个一个i平均分配给各个线程，如果想把循环一块一块分配给线程要怎么做呢？这时候用到了schedule制导语句。下面的代码演示了schedule的用法：

```

1 | #include <iostream>
2 | #include "omp.h"
3 | #include <stdio.h>
4 | using namespace std;
5 | int main(int argc, char **argv) {
6 |     int n = 12;
7 |     omp_set_num_threads(4);
8 | #pragma omp parallel
9 |     {
10 | #pragma omp for schedule(static, 3)
11 |         for (int i = 0; i < n; i++) {
12 |             {
13 |                 printf("i = %d, I am Thread %d\n", i, omp_get_thread_num());
14 |             }
15 |         }
16 |     }
17 | }

```

上面代码中for循环并行化时将循环很多很多块，每一块大小为3，然后再平均分配给各个线程执行。

输出结果如下：

```

1 | i = 6, I am Thread 2
2 | i = 3, I am Thread 1
3 | i = 7, I am Thread 2
4 | i = 4, I am Thread 1
5 | i = 8, I am Thread 2
6 | i = 5, I am Thread 1
7 | i = 0, I am Thread 0
8 | i = 9, I am Thread 3
9 | i = 1, I am Thread 0
10 | i = 10, I am Thread 3
11 | i = 2, I am Thread 0
12 | i = 11, I am Thread 3

```

从输出结果可以看到：线程0执行i=0 1 2，线程1执行i=3 4 5，线程2执行i=6 7 8，线程3执行i=9 10 11，如果后面还有则又从线程0开始分配。


OK，for循环并行化的知识基本讲完了，还有一个有用的制导语句barrier，用它可以在并行块中设置一个路障，必须等待所有线程到达时才能通过，**这个一般在并行处理循环前后存在依赖的任务时使用到。**

是不是很简单？


文章标签：[openmp](#) [并行处理](#) [多线程](#)

个人分类：[并行程序设计](#)

想对作者说点什么？ [我来说一句](#)

 **小田田_XOW** 2018-01-04 18:25:56 #6楼

谢谢分享

 **free3conan** 2017-07-15 10:27:00 #5楼 [查看回复\(1\)](#)

博主你好，你的讲解非常清晰，只是有一点我不是很明白，希望你能赐教。在数据同步的例子中，你用了sum求解前10000项和，其中你提到“多个线程同时对sum操作时就会因为数据同步问题导致结果不对”你能讲一下导致结果不对的机制吗，我觉得8个进程同时一个共享内存sum进行累加，这个过程为什么会产生数据冲突？


 **Coder--** 2016-12-26 11:17:02 #4楼

通俗易懂

[查看 7 条热评](#)

【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

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...

 **qq_19764963** 2016-05-17 15:47:12 阅读数：5057

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

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！ ...

 **Allyli0022** 2016-09-29 15:44:40 阅读数：6045

OpenMP中数据属性相关子句详解（1）：private/firstprivate/lastprivate/threadprivate之间的比较

private/firstprivate/lastprivate/threadprivate，首先要知道的是，它们分为两大类，一类是private/firstprivate/lastprivate子句...

 **gengshenghong** 2011-11-22 11:11:56 阅读数：11008

OpenMP中数据属性相关子句详解(3)：reduction子句

#pragma omp parallel与#pragma omp parallel for

今天写OpenMP的程序，遇到很让人恼火的问题，三个#pragma omp parallel没有问题，再有一个#pragma omp parallel，计算结果就错误了。修改调试了一个晚上...

akunpopping 2012-12-09 22:09:28 阅读数: 3091

OpenMP并行构造的schedule子句详解

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-document...>)

gengshenghong 2011-11-22 23:22:51 阅读数: 13129

OpenMP #pragma omp parallel for并行化小探究

今天用了一下openmp，本人表示非常喜欢openmp的傻瓜化模式，导入一个头文件直接parallel for#include #include using namespace std;int ma...

Scythe666 2015-05-05 10:47:52 阅读数: 3214

OpenMP对于嵌套循环应该添加多少个parallel for

一个原则是：应该尽量少的使用parallel for，因为parallel for也需要时间开销。即：（1）如果外层循环次数远远小于内层循环次数，内层循环较多时，将parallel for加...

K346K346 2015-04-27 14:48:18 阅读数: 5909

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 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: 11962

OpenMp 程序优化，怎么让并行达到并行的效果！

参考链接：<http://blog.csdn.net/donhao/article/details/5651156> 常用的库函数 函数原型 ...

fulva 2012-10-25 20:19:44 阅读数: 5421

50万码农评论：英语对于程序员有多重要！

不背单词和语法，老司机教你一个数学公式秒懂天下英语



[并行计算] 2. OpenMP简介

OpenMP简介（这篇翻译只涉及与C/C++相关的代码和示例，忽略了与Fortran相关的代码和示例，感兴趣的读者可以参考原文）1 摘要OpenMP是由一组计算机硬件和软件供应商联合定义的应用程序接口...

magicbean2 2017-07-27 10:39:34 阅读数: 11237

一起来学OpenMP（3）——for循环并行化基本用法

一、引言在“一起来学OpenMP（1）——初体验”中给出了一个for循环并行化的例子，这里做进一步的分析，但本节仅描述for循环并行化的基本用法。二、for循环并行化的几种声明形式#include ...

donhao 2010-06-06 23:08:00 阅读数: 18261

OpenMP

Eclipse下配置：<http://www.ipd.uni-karlsruhe.de/multicore/research/download/HowToGuide-OpenMP.pdf> for循...

shengwenj 2017-05-24 16:54:19 阅读数: 971

ubuntu OpenMP parallel for

ubuntu OpenMP parallel forOpenMP并行计算for循环test.cpp#include #include #include using namespace std;i...

u011773995 2017-02-27 16:11:15 阅读数: 1146

免费云主机试用一年

云服务器免费试用



intel openmp

介绍在串行应用中找出何处能够有效实施并行。作者 Clay P. Breshears显式线程化方法（如，Windows* 线程或 POSIX* 线程）使用库调用创建、管理并同步线程。使用显式线程，需要...

yangdelong 2007-07-19 10:48:00 阅读数: 2099

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。...

wyjjk 2011-07-17 14:32:41 阅读数: 11282

OpenMP: VS2010配置使用OpenMP

一个简单的OpenMP例子首先启动VisualStudio 2010，新建一个C++的控制台应用程序，如下图所示：然后在项目解决方案资源管理器上选择项目名称，点击右键，选择“属性”，如下图所示：然后...

Augusdi 2013-04-16 12:42:15 阅读数: 24293

OpenMP Tutorial学习笔记(4)OpenMP指令之同步构造（Parallel）

OpenMP Tutorial: <https://computing.llnl.gov/tutorials/openMP/#ParallelRegion> Parallel指令。（1）Paralle...

gengshenghong 2011-11-10 19:41:19 阅读数: 3861

OpenMP: OpenMP嵌套并行

OpenMP中不建议使用并行嵌套，如果一个并行计算中的某个线程遇到了另外一个并行分支，程序运行将会变得不稳定。将一个完整的工作任务通过一组并行线程分成若干小任务，每个线程只执行指定给它的那段代码，并没...

Augusdi 2013-04-16 11:49:57 阅读数: 5840

Python代码加密，完美防止反编译

对转成的exe加密或者.py或者.pyc文件加密，防止代码被反编译



OpenMP的循环使用实例展示

转载请声明出处<http://blog.csdn.net/zhongkejingwang/article/details/40018735>在C/C++中使用OpenMP优化代码方便又简单，代码中需要并行...

qq_31839479 2016-11-08 15:51:47 阅读数: 624

openmp嵌套并行操作

本章讨论 OpenMP 嵌套并行操作的特性 1 执行模型 OpenMP 采用 fork-join（分叉-合并）并行执行模式。线程遇到并行构造时，就会创建由其自身及其他一些额外（可能为零）...

zhuxianjianqi 2012-12-12 19:34:12 阅读数: 4107

OpenMP并行程序设计（一）

OpenMP并行程序设计（一） OpenMP是一个支持共享存储并行设计的库，特别适宜多核CPU上的并行程序设计。今天在双核CPU机器上试了一下OpenMP并行程序设计，发现效率方面超出想象，因此写出...

parfor —— matlab 下的并行循环

1. parfor: parallel for 循环我们知道, matlab 更适合的处理对象是矩阵, 而不是大规模的循环运算。当有时不得不使用 for 循环时, 如果提高 for 循环的执行效率呢。这就是 ...

 lanchunhui 2016-10-10 19:56:03 阅读数: 2754

OpenMP并程序序设计——for循环并行化详解（转载）

<http://blog.csdn.net/zhongkejingwang/article/details/40018735> 在C/C++中使用OpenMP优化代码方便又简单, 代码中需...

 boss212 2016-08-17 23:05:18 阅读数: 187


免费云主机试用一年

有哪些可以免费试用一年左右的云服务器



openMP编程探索2——循环并行化

openMP并不是只能对循环来并行的, 循环并行化单独拿出来说是因为它在科学计算中非常有用, 比如向量、矩阵的计算。所以我单独拿出这一部分给大家讲讲。这里主要讲解的是for循环。 ...

 bendanban 2011-04-05 17:06:00 阅读数: 4054


最简单的并行计算——OpenMP的使用

简介 OpenMP的英文全称是Open Multiprocessing, 一种应用程序界面 (API, 即Application Program Interface), 是一种单进程多线程并行的实现和方法, ...

 xiangxianghehe 2018-01-20 00:25:14 阅读数: 1267

关于openMP在vs中的并行

摘自[1]http://blog.sina.com.cn/s/blog_57562d890100xj3l.html 一、引言 在“一起来学OpenMP (1) ——初体验”中给出了一个for循...

 u010329292 2016-01-06 22:35:31 阅读数: 712

VS2010开启OpenMP，进行并行化程序设计

1.OpenMP介绍 OpenMP 是 Open MultiProcessing 的缩写, 用于共享内存并行系统的多处理器程序设计的一套指导性编译处理方案。在项目程序已经完成好的情况下不需要大幅度的修改...

 HW140701 2017-06-25 14:57:40 阅读数: 1942

openMP 嵌套循环

应该尽量少的使用paralelfor, 因为parallel for也需要时间开销 采用嵌套并行并不一定能提高效率, 只有在合适的地方设置并行才能达到事半功倍的效果...

 kl1411 2017-02-26 20:35:09 阅读数: 438

程序员不会英语怎么行？

老司机教你一个数学公式秒懂天下英语



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

openMP（并行计算）超简单快速上手

简介: OpenMp是并已被广泛接受的, 用于共享内存并行系统的多处理器程序设计的一套指导性的编译处理方案。OpenMP支持的编程语言包括C语言、C++和Fortran; OpenMp提供了对并行算法...

 hyqsong 2015-12-06 21:29:30 阅读数: 2327

奇异值分解(SVD)原理详解及推导

Linux环境下的OpenMP多线程编程

Linux环境下的OpenMP多线程编程 在正式开始前,我先说一下,我们的OpenMP编程在linux下写完以后编译工具为GCC,编译命令如下: gcc -fopenmp filena me.c -o ...

qq_20198487 2016-06-10 11:08:58 阅读数: 5931

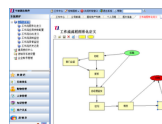
OpenMP中几个容易混淆的函数(线程数量/线程ID/线程最大数)以及并行区域线程数量的确定

说明:这部分内容比较基础,主要是分析几个容易混淆的OpenMP函数,加以理解。(1)并行区域数量的确定:在这里,先回顾一下OpenMP的parallel并行区域线程数量的确定,对于一个并行区域...

gengshenghong 2011-11-23 10:56:55 阅读数: 22119

工作流系统

一个工作流管理系统的设计和实现



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

为什么循环队列具有先天的并行性

循环队列是很多人喜欢用的一种数据结构。本着先来先服务的特性,循环队列是一种十分简单、健壮的数据结构。不像链表、二叉树,如果使用不慎,就会造成很大的麻烦,但是在循环队列上面则没有这个烦恼。...

benpaobagzb 2016-02-29 21:22:50 阅读数: 194