共享内存编程：OpenMP

陈俊清

jqchen@math.tsinghua.edu.cn

清华大学数学科学系

April 26, 2013

Outline

Outline

OpenMP之基本概念

OpenMP是作为共享存储标准出现的，它为共享存储环境编写并行程序提供一系列应用编程接口。 目前支持OpenMP的语言主要有：Fortran，C/C++，操作系统 UNIX/LINUX, Windows。 OpenMP标准中包括一套编译指导语句和一个支持函数库。

- OpenMP是基于线程的并行编程模型:
- 使用Fork/Join 并行执行模式
- 可移植性
- 显式并行
- 易用性
- ...

一个例子omp_test0.c（关于并行的效率参看test.c）

```
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
int main(int argc, char **argv){
     int myrank, thread_count;
#pragma omp parallel private(myrank) \
          num_threads(4)
     {
       myrank=omp_get_thread_num();
       thread_count=omp_get_num_threads();
       printf("Hello from thread %d of %d\n", myrank,
           thread_count);
     }
     return 0;
}
```

OpenMP程序的编译与运行

编译:

```
gcc -g -Wall -fopenmp -o omp_test0 omp_test0.c
```

运行:

```
./omp_test0
```

运行结果:

```
Hello from thread 0 of 4
Hello from thread 3 of 4
Hello from thread 1 of 4
Hello from thread 2 of 4
```

编译指导（compile directive）

```
#pragma omp directive [clause list]
```

- # pragma omp: 指导指令前缀，所有的OpenMP语句都需要;
- directive：指导指令名称;
- clause list：指导子句。

并行指导语句:

```
#pragma omp parallel [clause list]
```

指导子句 clause

- if(expression)：条件并行
- private(list)：指定线程的私有变量
- firstprivate(list)：指定线程的私有变量，并继承主线程的初值
- lastprivate(list)：指定线程的私有变量，并行处理结束后将变量值复制回主线程的对应变量。
- shared(list)：指定一个或多个变量为多个线程的共享变量
- default(shared|none)：用来指定并行处理区域内变量的使用方式，缺省为shared
- reduction(operation:list)：指定一个或多个变量为是私有的，并在并行处理后执行指定的运算
- num_threads(int)：指定线程的个数

两个数值积分的例子

参看(pi_omp.c pi_omp1.c)两个例子程序.

- 指定进程数:

```
#pragma omp parallel num_threads(4)
```

  或

```
omp_set_num_threads(4);
```

  或设置环境变量: set OMP_NUM_THREADS=4;
- 指导指令 critical: 表示它的作用域内的代码一次只能被一个进程执行

parallel for 命令

Monte Carlo方法计算π（pi_omp2.c）：

```
#pragma pallel default(private) shared(npoints) \
            reduction(+;sum) num_threads(4)
{
     sum=0;
     #pragma omp for
     for(i=0;i<npoints;i++){
        rand_no_x=(double)(srand(seed))/RAND_MAX;
        rand_no_y=(double)(srand(seed))/RAND_MAX;
        if((rand_no_x-0.5)*(rand_no_x-0.5)
                +(rand_no_y-0.5)*(rand_no_y-0.5)<0.25)
            sum++;
      }
}
```

Outline

parallel

在OpenMP中，创建线程都是用编译指导语句来实现的。创建线程的命令主要有parallel、for、sections、section等命令。

parallel

parallel命令用来构造一个并行块，也可以与其它命令如for、sections等与它配合使用，其使用方法如下：

```
 #pragma omp parallel [for|sections][clause]
{
                ...
}
```

注：可以在clause中使用num_threads()来指定线程数量。

parallel的嵌套执行

按照OpenMP3.0版本的规范，缺省情况下不支持嵌套并行，但可以使用函数omp_set_nested()打开嵌套支持，只要使用非0参数传递给omp_set_nested()函数，就可以支持嵌套并行。

```
 int main(int argc, char **argc)
{
    omp_set_nested(TRUE);
    #pragma omp parallel num_threads(2)
    {
        ...
        #pragma omp parallel num_threads(2)
        {
          ...
        }
     }
    return 1;
}
```

for 和parallel for

for 指令用来将一个for循环分配到多个线程中执行。for 命令一般可以与parallel命令合起来形成parallel for命令使用， 也可以单独用在parallel语句的并行块中。

```
#pragma omp [parallel] for [clause]
 for 循环语句
```

比如:

```
#pragma omp parallel
{
    #pragma omp for
    for(i=0;i<n;i++) {
          ...
    }
}
```

注: 单独用for不会创建多线程执行。

if子句（条件执行并行）

在并行构造中，可以设置条件来决定是否对并行区域中的代码块并行运行。OpenMP提供if子句来实现条件执行并行。

```
#pragma omp parallel if(n>10) num_threads(4)
 {
            ..............
 }
```

上述代码中，如果$n > 10$，则并行执行，否则串行执行。

sections和section

section 和sections是OpenMP用来创建线程的另外一种方式。先用sections定义一个区块， 然后用section将sections区块划分成几个不同的段，并行执行这些段。 与for命令一样，sections命令必须用在parallel区域内，或与parallel命令结合成parallel sections命令。

```
 #pragma omp parallel sections [clause]
{
 #pragma omp section
    {   .......      }
  #pragma omp section
   {          ......      }
    ......
}
```

注： for命令由系统分配并行任务，而sections则由编程者手动分配任务，需要注意任务分配合理性。

single命令

single命令用来指定某块程序由一个线程执行。除非使用了nowait子句，否则在此线程执行期间其他线程都在等候，直到single块结束隐含的barrier.

```
#pragma omp single [clause]
{
        ... ...
}
```

比如:

```
#pragma omp parallel num_threads(4)
{
    ... ...
    #pragma omp single
     {
     .........
     }
    ......
```

master 命令

master命令用于指定一块程序在主线程中执行。master 命令的用法如下:

```
 #pragma omp master [clause]
{
 ... ...
}
```

master 与single有些类似，其后的程序块只被主线程执行，但是，不同的是，master命令没有隐含的barrier，主线程在执行其后的程序块时，其他进程不会等候，可以往下执行。

Outline

private子句

private子句用于将一个或多个变量声明成线程的私有变量，然后指定每个线程都有它自己的变量私有副本。即使在并行区域外有同名的共享变量， 该共享变量在并行区域内不起任何作用，而且并行区域内的操作不会影响到外面的共享变量。

private子句

private(list)

```
  int k = 100;
#pragma omp parallel for private(k)
  for (k=0;k<10;k++)
  {
     printf("k=%d\n",k);
  }
printf("k=%d\n",k);
```

注：用private子句声明的私有变量的初始值在并行区域的入口处是没有定义的，它不会继承同名共享变量的值。

firstprivate子句

OpenMP提供firstprivate子句来使得私有变量继承同名共享变量的值。

```
 int k=100;
#pragma omp parallel for firstprivate(k)
for (i=0;i<4;i++)
{
    k+=i;
    printf("k=%d\n",k);
}
printf("k=%d",k);
```

lastprivate子句

private,firstprivate子句在退出并行域时没有将私有变量的值赋给同名共享变量。而lastprivate子句用来在退出并行域的时候将私有变量的值赋给共享变量。

```
int k =100;
#pragma omp parallel firstprivate(k),lastprivate(k)
for(i=0;i<4;i++)
{
  k+=i;
  printf("k=%d\n",k);
}
printf("k=%d\n",k);
```

注：并行区域内的程序由多个线程执行，OpenMP规范中指出，若是循环迭代，将最后一个循环迭代中的值赋给共享变量。若是section构造，则将最后一个section语句中对应的值赋给共享变量。

threadprivate子句

threadprivate子句用来指定全局的对象被各个线程各自获得一个私有的副本，即各个线程具有各自私有的全局对象。

```
#pragma omp threadprivate(list)

 int counter=0;
#pragma omp threadprivate(counter)
int increment_counter()
{
          counter++;
          return counter;
}
```

注：threadprivate和private的区别在于，threadprivate声明的变量只能是全局变量或是静态变量，而private声明的变量可以是局部变量。

shared子句

shared子句用来声明一个或多个变量是共享变量，

```
shared(list)
```

注：在并行区域内使用共享变量时，如果存在写操作，必须对共享变量加以保护，否则不要轻易使用共享变量， 尽量将共享变量的访问转化为私有变量的访问。

default子句

default子句允许用户控制并行区域中变量的共享属性。

```
default(shared|private|none)
```

使用shared时，缺省情况下，传入并行区域内的同名变量作为共享变量，除非使用private等子句指定某些变量为私有。如果使用none作为参数，那么线程中用的变量必须显式指定是共享还是私有。

reduction子句

reduction子句主要用来对一个或多个参数指定一个操作符；每个线程将创建参数的一个私有拷贝，在区域的结束处， 将私有拷贝的值通过指定的运算符运算，用运算结构更新参数。
注：如果写操作涉及共享变量，可能会出现不可预测的异常结构，注意写保护。

copyin子句

copyin子句用来将主线程中threadprivate变量的值拷贝到执行并行区域的各个线程的threadprivate变量中，便于线程访问主线程中的变量值，其用法如下。

```
copyin(list)
```

copyin中的变量必须被声明成threadprivate的，参看例子程序（omp_test1.c）。

copyprivate子句

copyprivate子句提供了一种机制：用一个私有变量将一个值从一个线程广播到同一并行区域中的其他线程。copyprivate子句可以关联single构造， 在single的barrier到达之前完成广播。copyprivate可以对private和threadprivate子句中的变量操作，但如果当使用single构造的时候， copyprivate的变量不能用于private和firstprivate子句中。(omp_test2.c)

Outline

schedule子句

OpenMP中，任务调度主要用于for循环。当循环中每次迭代的计算量不等时， 如果简单地给每个线程分配相同次数的迭代，则会造成各个线程计算负载不均衡。OpenMP提供了几种对for循环并行化的任务调度方案。 schedule子句的使用格式:

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

其中，type参数表示调度方式，有如下4种:

- dynamic
- guided
- static
- runtime

schedule子句

size参数表示循环次数，必须是整数。static、dynamic、guided三种调度方式都可以使用size参数，而runtime时size参数是非法的。
runtime实际上根据环境变量OMP_SCHEDULE来选择前三种的某一种，用户可以自定义环境变量的值。如在bash下：

```
set OMP_SCHEDULE="static,2"
```

静态调度

当parallel for编译指导语句没有带schedule子句是，大部分系统会默认采取静态调度方式。假设有n次循环，t个线程，那么每个线程大概分n/t次循环。 n/t不一定是整数，实际分配时可能存在差1的情况，如果指定了size，则可能相差size。 参看例子程序(omp_test3.c)

动态调度与guided调度

动态调度（dynamic）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，不使用时是将迭代逐个分配到各线程， 指定时，每次分配给线程的迭代次数为size个。
guided调度采用指导性的启发式自调度方式进行调度。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逐渐递减。 （参看omp_test3.c）

Outline

barrier命令

OpenMP的许多命令都内置了障碍，但这些障碍都是隐含的。OpenMP提供barrier命令给程序员显式调用障碍:

```
#pragma omp barrier
```

所有遇到barrier命令的线程都需要等待其它的线程全部跟上后才能继续执行后面的程序；如果在嵌套的parallel命令中使用， 则barrier应绑定到最近的parallel命令中。并不是任意位置都可以设置障碍，其放置的位置需满足当你把障碍删除后仍然能保证程序的正确性。

atomic命令

critical命令表示其后的一块程序（临界区）一次只能有一个线程执行。而atomic命令也是对临界区的操作,表示其中的变量只能原子地更新。

```
in i, nvar=0;
 #pragma omp parallel for shared(nvar)
for (i=0;i<10000;i++)
{
#pragma omp atomic
     nvar+=1;
}
```

ordered命令

ordered命令和子句用来指定循环区域内的某结构块按循环迭代的顺序来执行，ordered命令需要和ordered子句结合起来使用,ordered子句为， 放在for或parallel for命令里使用：

```
#pragma omp parallel for ordered
  for(i=0;i<10;i++)
#pragma omp ordered
        printf("i=%d\n",i);
```

注：同一个循环内只能有一个ordered命令。

flush命令

flush命令是用来确保执行中存储器中数据的一致性，该命令可以保证线程可见的变量写回存储器， 使得一个线程写入的变量值可以被另外一个线程读取。

```
#pragma omp flush(list)
```