计算科学与工程中的并行编程技术

Parallel Programming Technology in Computational Science and Engineering

都志辉

清华大学计算机系

Email：duzh@tsinghua.edu.cn

Phone: 62782530

http://hpclab.cs.tsinghua.edu.cn/~duzh

OpenMP

都志辉

参考网站

- http://openmp.org/ 规范与官网
- http://forum.openmp.org/forum/
- http://www.compunity.org/ 用户群

OpenMP简介

- **OpenMP (Open Multi-Processing)**
  - 共享内存并行编程
- 支持的语言
  - C, C++, and Fortran
- 通过对已有语言的标注扩展来实现并行，类似HPF
- 支持的操作系统
  - Solaris, AIX, HP-UX, Linux, OS X, and Windows
- 包括什么东西
  - 编译制导、库函数与过程、环境变量

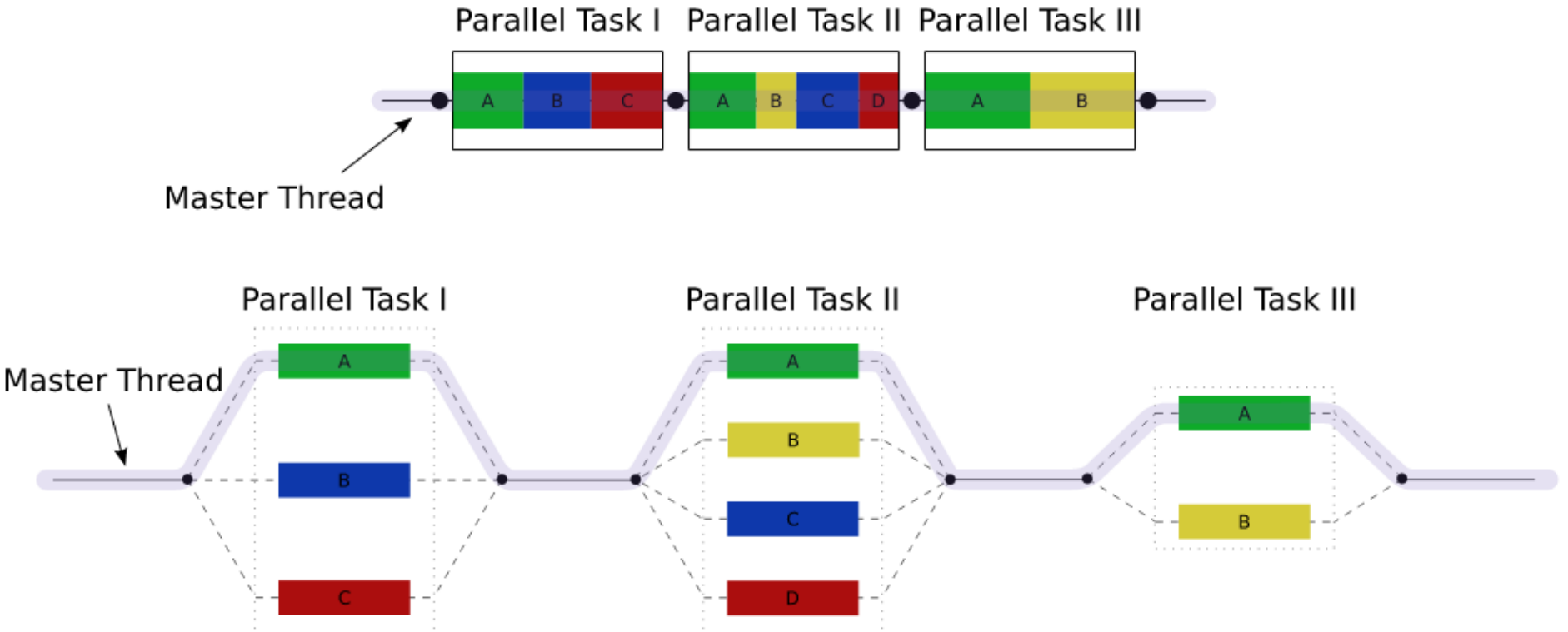
创立者

- *OpenMP Architecture Review Board* (or *OpenMP ARB*)
  - 非盈利组织
  - 主流计算机软硬件销售商
    AMD, IBM, Intel, Cray, HP, Fujitsu,Nvidia, NEC, Red Hat, Texas Instruments, Oracle Corporation, and more
  - 1997年10月发布第一个针对Fortran的版本

与MPI的对比

- 库扩展←→语言扩展
- 进程并行←→线程并行
- 分布（不同地址空间）←→共享内存（相同地址空间）
- 编写新的程序←→扩展已有程序（逐渐增加并行的比重）
- 程序编写 难←→容易
- 程序性能 高←→依赖并行的结构（规则并行性能高）
- 扩展性 高←→ 不如MPI
- 通信语句 需要←→不需要（共享变量）
- 并行IO 支持←→不支持

Fork-Join并行执行模式



制导符

- 额外增加的可以被非OpenMP编译器忽略（当作注释）的符号与内容
  - 在Fortran中以注释的形式存在
  - 在C/C++中以预处理宏指令的形式存在
  - 串行编译器与并行OpenMP编译器都可以处理，
    - 并行制导符对串行执行没有任何影响，但是不正确的制导符会导致错误的并行执行

基本的制导符

- Fortran固定格式
  - COMP, cOMP,*$OMP,!$OMP
  - C$OMP& 表示续行
- Fortran自由格式
  - !$OMP
- C/C++
  - #pragma omp
  - 续行用\

一种最基本的并行结构

- parallel Construct
  - 第一个执行到的语句是主线程
  - 并行线程有唯一的ID
  - 执行过程中线程个数不变
  - 在该块最后进行隐式同步

Hello World

```
PROGRAM Main
!$OMP PARALLEL
print *, "Hello World !"
!$OMP END PARALLEL
END
```

内部控制变量ICV OMP_NUM_THREADS决定使用多少线程来执行

Hello World

```
#include <stdio.h>

int main(void)

{   #pragma omp parallel
    printf("Hello, world.\n");
    return 0;

}
```

几种可以放在并行执行区域中的制导结构

- 用该区域已经fork的线程并行执行（如没有并行线程则串行执行）
- 入口无同步但是在退出该结构的时候需要同步（有nowait制导符除外）

（1）LOOP Worksharing Constructs

- Loop Construct
  - #pragma omp for
  - for-loops

LOOP例子

```
PROGRAM WORKSHARELOOP
REAL A(N), B(N)
!$OMP PARALLEL DO
        DO I = 1, N
                A(I) = I * 1.0
                B(I) = A(I)
        ENDDO
!$OMP END PARALLEL
END
```

LOOP例子

```
int main(int argc, char **argv)
{ int a[100000];
    #pragma omp parallel
    { #pragma omp for
      for (int i = 0; i < 100000; i++)
      { a[i] = 2 * i;
      }
    }
    return 0;
}
```

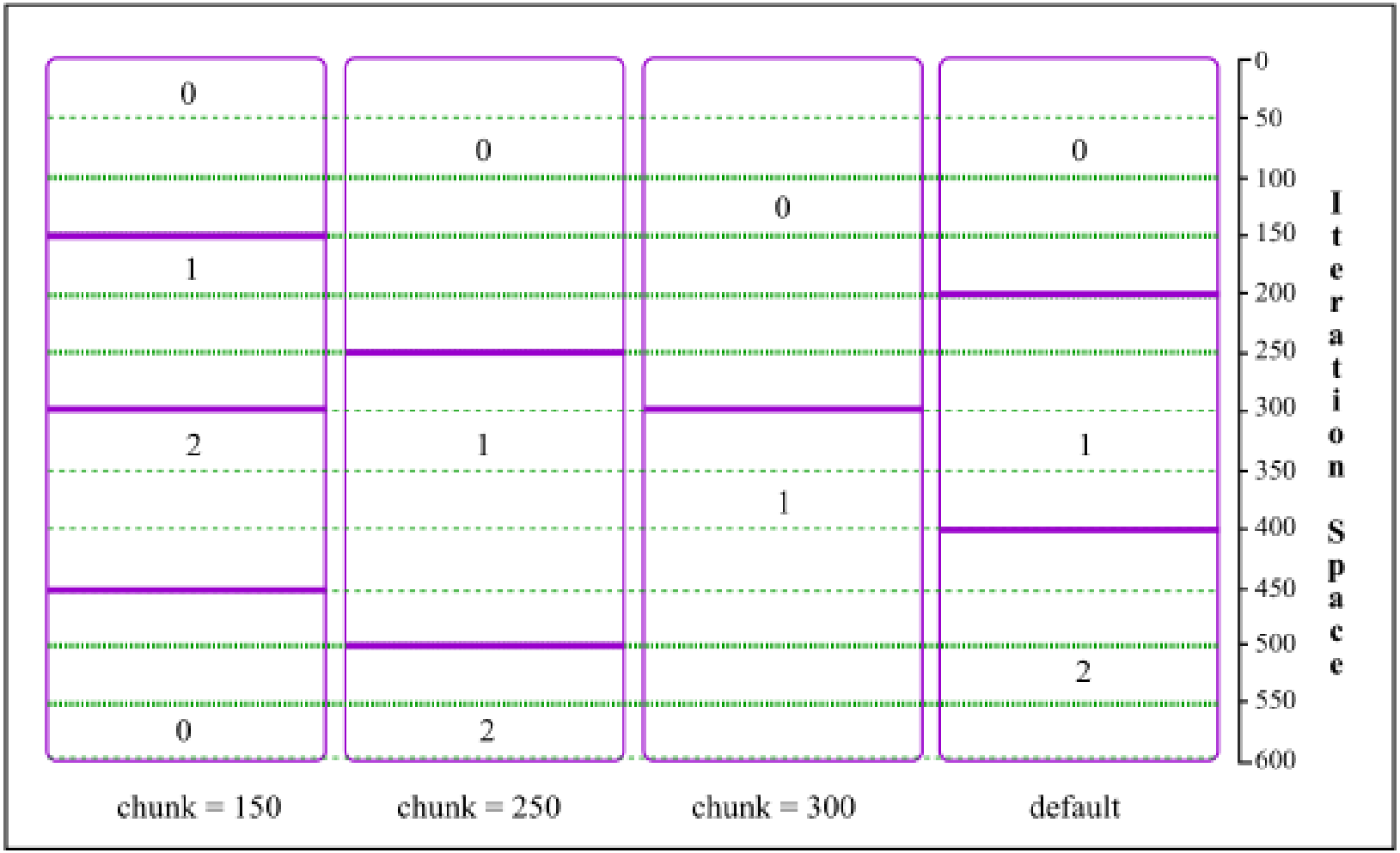
FFT例子

```
#pragma omp parallel for
    for(i = 0; i <= 1; i++) {
      FFT(D + i * n, a + i * stride, W, n, stride << 1, A + i * n);
    }
    /* Combination stage */
    B = D;
    C = D + n;
```

调度属性的说明

- *schedule(type, chunk)*: 对于循环迭代，指明给线程分配迭代次数的方法，有三种：
  - *static*: 执行循环前已决定分配的情况，缺省均匀分配。如有chunk则按照指定的数值分配
  - *dynamic*: 先执行完的线程先领任务，任务量均匀
  - *guided*: 与dynamic类似，但是任务中迭代的数量先多后少，直到chunk指定的数目

STATIC


0
1
2
0
0
1
2
0
0
1
1
2
0
50
100
150
200
250
300
350
400
450
500
550
600
Iteration Space
chunk = 150
chunk = 250
chunk = 300
default

（2）Sections Worksharing Constructs

- 不同的块用不同的线程执行

```
#pragma omp sections [clause[ [,] clause] ... ] new-line
    {
    [#pragma omp section new-line]
        structured-block
    [#pragma omp section new-line
        structured-block]
    ...
    }

!$omp sections [clause[ [,] clause] ... ]
    [!$omp section]
        structured-block
    [!$omp section
        structured-block]
    ...
!$omp end sections [nowait]
```


20

例子

```
#pragma omp parallel sections
{       #pragma omp section
        {       printf ("section 1 id = %d, \n",
                omp_get_thread_num());
        }
        #pragma omp section
        {       printf ("section 2 id = %d, \n",
                omp_get_thread_num());
        }
        #pragma omp section
        {       printf ("section 3 id = %d, \n",
                omp_get_thread_num());
        }
}
```

- 如有4个线程，可能打印出什么结果？
- 如有2个线程，可能打印出什么结果？

（3）Single Worksharing Constructs

- 只有一个线程做相关的内容，其它的都不做

#pragma omp single *[clause[[,] clause] ...] new-line*
structured-block

!$omp single *[clause[[,] clause] ...]*
structured-block
!$omp end single *[end_clause[[,] end_clause] ...]*

Single Construct例子

```
#include <stdio.h>

void work1() {}
void work2() {}

void single_example()
{
  #pragma omp parallel
  {
    #pragma omp single
      printf("Beginning work1.\n");

    work1();

    #pragma omp single
      printf("Finishing work1.\n");

    #pragma omp single nowait
      printf("Finished work1 and beginning work2.\n");

    work2();
  }
}
```

（4）WorkShare Worksharing Constructs

- 块内是由独立的单元组成，每个单元执行一次

```
!$omp workshare
    structured-block
!$omp end workshare [nowait]
```

- 仅仅fortran语言有，块中的语句可以为数组赋值、标量赋值、FORALL语句与结构、WHERE语句与结构、atomic结构、critical结构、parallel结构

例子

```
      SUBROUTINE WSHARE1(AA, BB, CC, DD, EE, FF, N)
      INTEGER N
      REAL AA(N,N), BB(N,N), CC(N,N), DD(N,N), EE(N,N), FF(N,N)

!$OMP    PARALLEL
!$OMP     WORKSHARE
            AA = BB
            CC = DD
            EE = FF
!$OMP     END WORKSHARE
!$OMP   END PARALLEL

      END SUBROUTINE WSHARE1
```

变量的作用域
共享与私有变量

- 共享变量（每个线程都可以访问）
- 私有变量（只有声明的线程可以访问）

数据共享属性说明

- Default (shared)
  - 除了循环迭代变量缺省都是共享的
- Default (none)
  - 必须对所有的变量明确声明其属性
- default(firstprivate)
  - 私有变量初值从共享变量得到
- default(private)
  - 缺省为私有变量

几种变量声明方式

- Shared（列出变量）
  - 所有列出的都为共享变量
- private（列出变量）
  - 所有列出的都为私有变量
- firstprivate（列出变量）
  - 所有列出的都为firstprivate变量，从共享变量得到初值
- lastprivate（列出变量）
  - 所有列出的都为lastprivate变量，最后一个私有变量返回结果给共享变量

Schedule例子

```
#define CHUNKSIZE 1 /*defines the chunk size as 1
contiguous iteration*/ /*forks off the threads*/
#pragma omp parallel private(j,k)
{ /*Starts the work sharing construct*/
      #pragma omp for schedule(dynamic, CHUNKSIZE)
      for(i = 2; i <= N-1; i++)
      for(j = 2; j <= i; j++)
      for(k = 1; k <= M; k++)
            b[i][j] += a[i-1][j]/k + a[i+1][j]/k;
}
```

综合例子

```
#pragma omp parallel for default(none) shared(M,L,size,k)
private(i,j) schedule(static,1)
      for (i=k+1; i<size; i++) {
            /* 4.1.1. COMPUTE L COLUMN */
            L[i][k] = M[i][k] / M[k][k];

            /* 4.1.2. COMPUTE M ROW ELEMENTS */
            for (j=k+1; j<size; j++)
                  M[i][j] = M[i][j] - L[i][k]*M[k][j];
            }

/* 4.2. END ITERATIONS LOOP */
      }
```

其它的属性说明

- **Synchronization clauses**
  - *critical*: 有且仅有一个线程可在同一时间执行该区域的程序
  - *atomic*: 内存更新语句是原子操作
  - *ordered*: 按照串行的次序执行
  - *barrier*: 所有线程在此同步
  - *nowait*: 不需要同步

Critical例子

```
int a=0, b=0;
#pragma omp parallel num_threads(4)
{
#pragma omp single a++;
#pragma omp critical b++;
}
printf("single: %d -- critical: %d\n", a, b);
```

Atomic例子

```
void atomic_example(float *x, float *y, int *index, int n)
{
  int i;

  #pragma omp parallel for shared(x, y, index, n)
    for (i=0; i<n; i++) {
      #pragma omp atomic update
      x[index[i]] += work1(i);
      y[i] += work2(i);
     }
}
```

User-level runtime routines

- 用于设置或者获取线程数目，是否在并行区域执行，当前系统有多少处理器，加锁或者开锁，计时函数等

例子（我是谁？共有多少线程？）

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
{ int th_id, nthreads;
      #pragma omp parallel private(th_id)
      {     th_id = omp_get_thread_num();
            printf("Hello World from thread %d\n", th_id);
            #pragma omp barrier
            if ( th_id == 0 )
            { nthreads = omp_get_num_threads();
                  printf("There are %d threads\n",nthreads);
            }
      } return EXIT_SUCCESS;
}
```

Fortran例子

```
program hello90
use omp_lib
integer:: id, nthreads
!$omp parallel private(id)
id = omp_get_thread_num()
write (*,*) 'Hello World from thread', id
!$omp barrier
if ( id == 0 ) then
nthreads = omp_get_num_threads()
write (*,*) 'There are', nthreads, 'threads'
end if
!$omp end parallel
end program
```

C++的写法

```
#include <iostream>
#include <omp.h>
int main()
{       int th_id, nthreads;
        #pragma omp parallel private(th_id) shared(nthreads)
        {       th_id = omp_get_thread_num();
                #pragma omp critical
                { std::cout << "Hello World from thread " << th_id
<< '\n'; }
                #pragma omp barrier
                #pragma omp master
                { nthreads = omp_get_num_threads();
                        std::cout << "There are " << nthreads << "
threads" << '\n';
                }
        } return 0;
}
```

Reduction例子

```
x = init_x
#pragma omp parallel for reduction(min:x)
for (int i=0; i<N; i++)
x = min(x,data[i]);
```

Reduction例子

```
#define N 10000 /*size of a*/
void calculate(long *); /*The function that
calculates the elements of a*/
int i; long w; long a[N]; calculate(a);
long sum = 0; /*forks off the threads and starts
the work-sharing construct*/
#pragma omp parallel for private(w)
reduction(+:sum) schedule(static,1)
for(i = 0; i < N; i++)
{ w = i*i;
  sum = sum + w*a[i];
}
printf("\n %li",sum)
```

Reduction

```
#pragma omp parallel for reduction(+:error)
private(i,resid)
      for (j=1; j<m-1; j++)
        for (i=1; i<n-1; i++){
            resid =(ax * (uold[i-1 + m*j] + uold[i+1 +
                       m*j])
                   + ay * (uold[i + m*(j-1)] + uold[i +
                       m*(j+1)])
                   + b * uold[i + m*j] - f[i + m*j]
                    ) / b;
            /* update solution */
            u[i + m*j] = uold[i + m*j] - omega * resid;
            /* accumulate residual error */
            error =error + resid*resid;

        }
```

PI例子

Example: A simple Parallel pi program

```
#include <omp.h>
static long num_steps = 100000;        double step;
#define NUM_THREADS 2
void main ()
{          int i, nthreads;  double pi, sum[NUM_THREADS];
           step = 1.0/(double) num_steps;
           omp_set_num_threads(NUM_THREADS);
   #pragma omp parallel
   {
          int i, id,nthrds;
          double x;
          id = omp_get_thread_num();
          nthrds = omp_get_num_threads();
          if (id == 0)   nthreads = nthrds;
          for (i=id, sum[id]=0.0;i< num_steps; i=i+nthrds) {
                    x = (i+0.5)*step;
                    sum[id] += 4.0/(1.0+x*x);
          }
    }
          for(i=0, pi=0.0;i<nthreads;i++)pi += sum[i] * step;
}
```

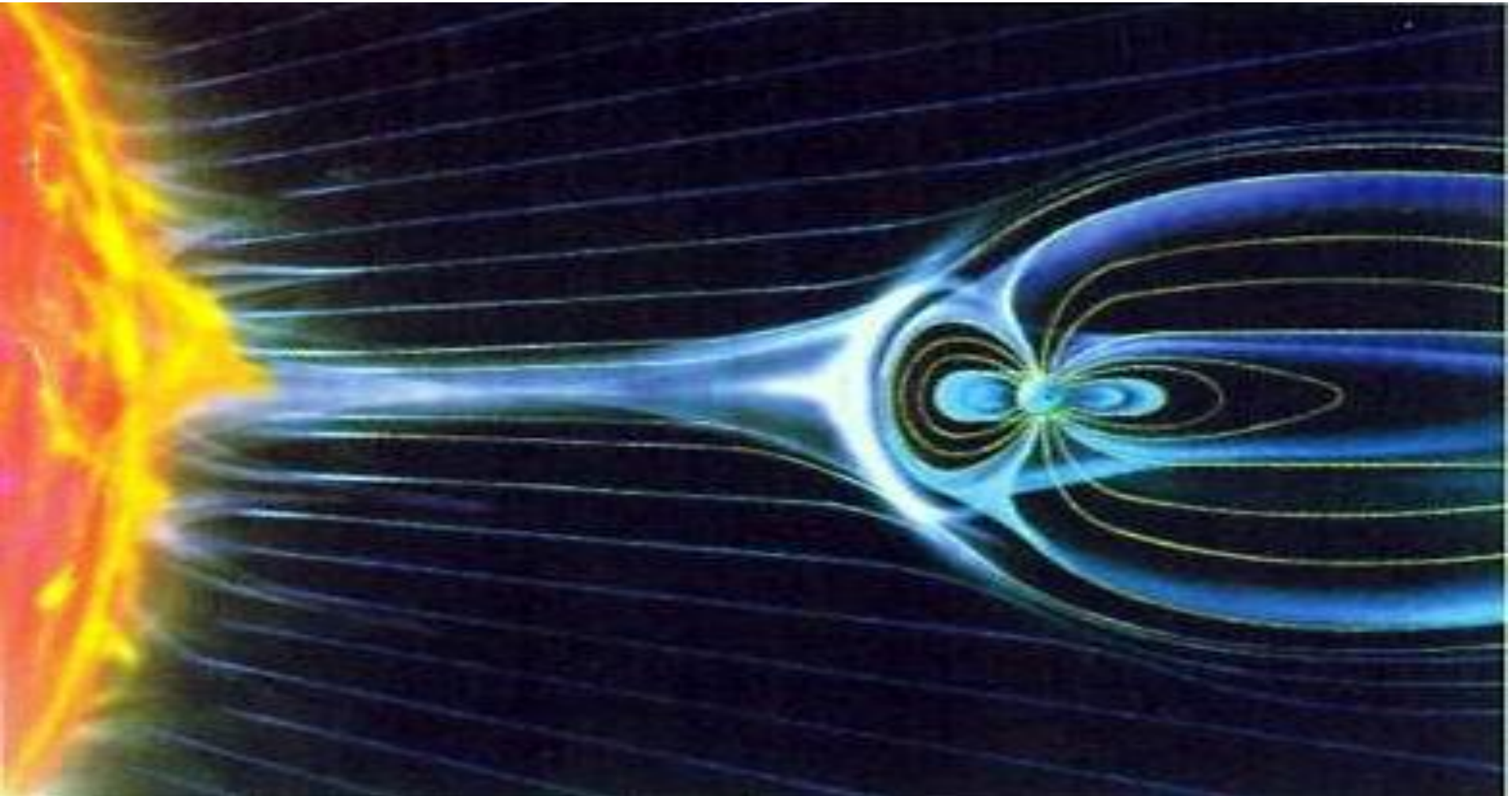
Promote scalar to an array dimensioned by number of threads to avoid race condition.

Only one thread should copy the number of threads to the global value to make sure multiple threads writing to the same address don't conflict.

This is a common trick in SPMD programs to create a cyclic distribution of loop iterations

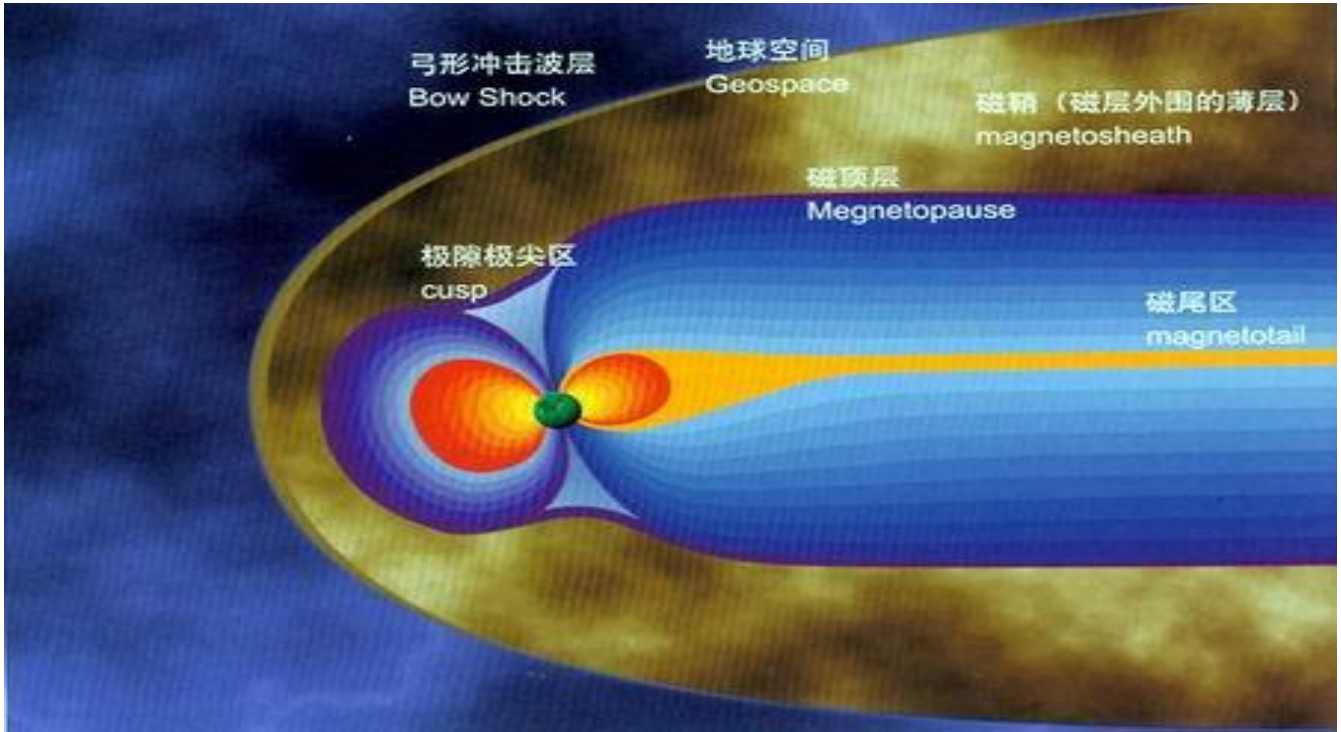
空间天气环境

- 太阳与地球交互的结果
- 太阳风（带电粒子） （图片来自空间科学中心）



对地球的影响

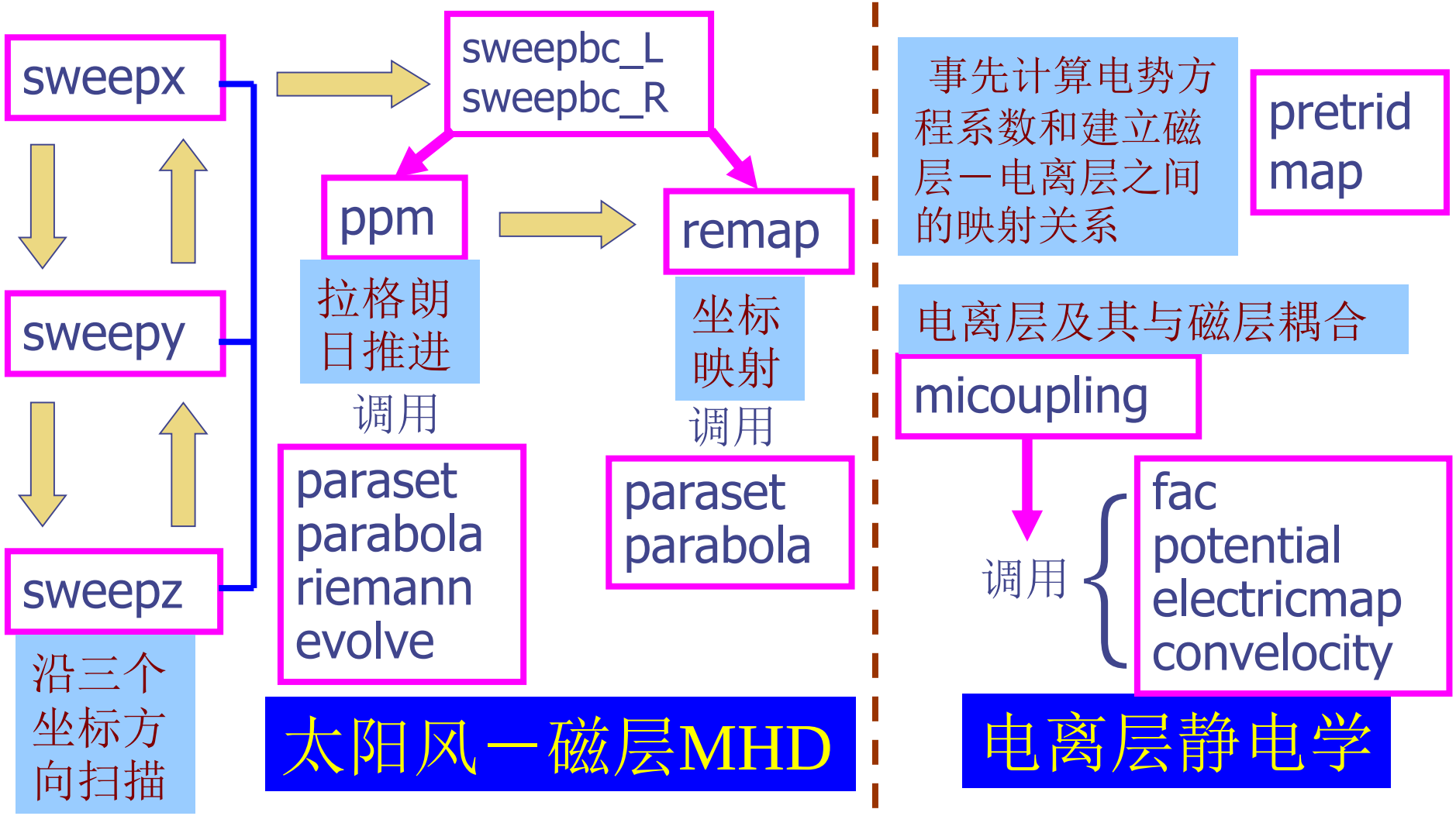
- 通信（无线电，电视传播，雷达，卫星）
- 电力网
- 身体健康（大剂量粒子射线的冲击）
- 臭氧层



问题抽象（等离子体-电磁场作用）

- MHD（ magnetohydrodynamics ）方程
- PPMLR（piecewise parabolic method with a Lagrangian remap）-MHD方法
  - 优点：精度高
  - 缺点：计算与通信量大
- 挑战
  - 如何完成实时计算？

计算流程



空间科学中心提供的材料

大型应用程序与Benchmark的不同

- 代码量
  - 十万行代码←→几百行
- 控制逻辑和数据结构
  - 复杂←→简单
- 调试
  - 1个月的移植，5个月的调试

Scale Challenge Award

16th IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER, CLOUD AND GRID COMPUTING

CARTAGENA DE INDIAS - COLOMBIA

May 16 - 19, 2016

SCALE CHALLENGE AWARD

Large Scale GPU Accelerated PPMLR-MHD Simulations for Space Weather Forecast

By Xiangyu Guo, BinBin Tang, Jian Tao, Zhaohui Huang and Zhihui Du

IEEE acm

Universidad de los Andes

Universidad Industrial de Santander

Rensselaer

Manish Parashar
Co-Chair
Scale 2016

José Tiberio Hernández
Co-Chair
Scale 2016

PPMLR-MHD代码

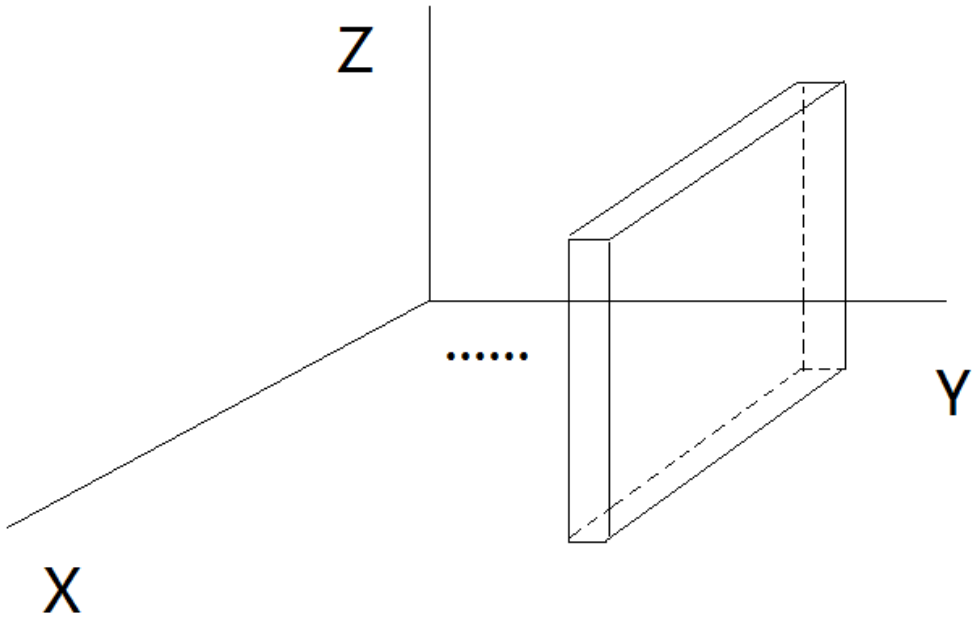
- 已经用MPI并行实现
- 特殊的非均匀网格划分（大型应用总有个性）
  - 156 *150*150
  - Y,Z必须为奇数，一个特殊进程
  - 可能的并行进程数
    - 4,28,37,55,101,151

OpenMP of the ppm-LR MHD code

- Parallel of sweepx, sweepy, and sweepz

sweepx: parallel in the y direction

(sweepy/z: x-axis)



```
!$omp parallel
 call sweepx
!$omp end parallel

!$omp parallel
 call sweepy
!$omp end parallel

!$omp parallel
 call sweepz
!$omp end parallel
```

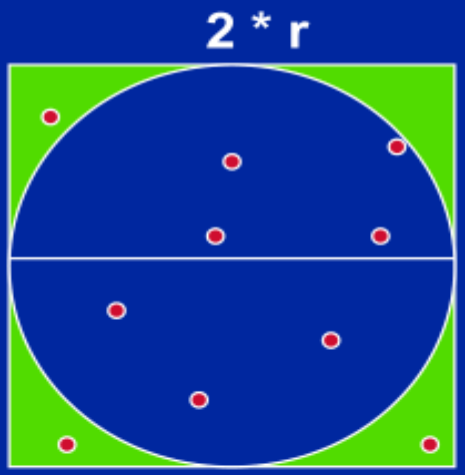
```
subroutine sweepx
...
nthreads = omp_get_num_threads()
iam = omp_get_thread_num()
ichunk = (jmax+nthreads-1)/nthreads
istart = iam*ichunk+1
iend = min((iam+1)*ichunk,jmax)
do 1 j = istart, iend
...
```

练习：另外一种求PI的方法

Exercise 9: Monte Carlo Calculations

Using Random numbers to solve tough problems

- Sample a problem domain to estimate areas, compute probabilities, find optimal values, etc.
- Example: Computing π with a digital dart board:



N= 10	π = 2.8
N=100	π = 3.16
N= 1000	π = 3.148

- Throw darts at the circle/square.
- Chance of falling in circle is proportional to ratio of areas:

  $A_c = r^2 * \pi$

  $A_s = (2*r) * (2*r) = 4 * r^2$

  $P = A_c/A_s = \pi/4$
- Compute π by randomly choosing points, count the fraction that falls in the circle, compute pi.

```
#include <stdio.h>
#include <omp.h>
#include "random.h"
static long num_trials = 10000;
int main ()
{
   long i;  long Ncirc = 0;
   double pi, x, y, test;
   double r = 1.0;   // radius of circle. Side of squrare is 2*r
   seed(-r, r);  // The circle and square are centered at the origin
   for(i=0;i<num_trials; i++)
   {
      x = drandom();
      y = drandom();
      test = x*x + y*y;
      if (test <= r*r) Ncirc++;
    }
    pi = 4.0 * ((double)Ncirc/(double)num_trials);
    printf("\n %ld trials, pi is %lf \n",num_trials, pi);
    return 0;
}
```