OpenMP编程

- What does OpenMP stand for?
  - Short version: **Open Multi-Processing**
  - Long version: **Open** specifications for **Multi-Processing** via collaborative work between interested parties from the hardware and software industry, government and academia.

提纲

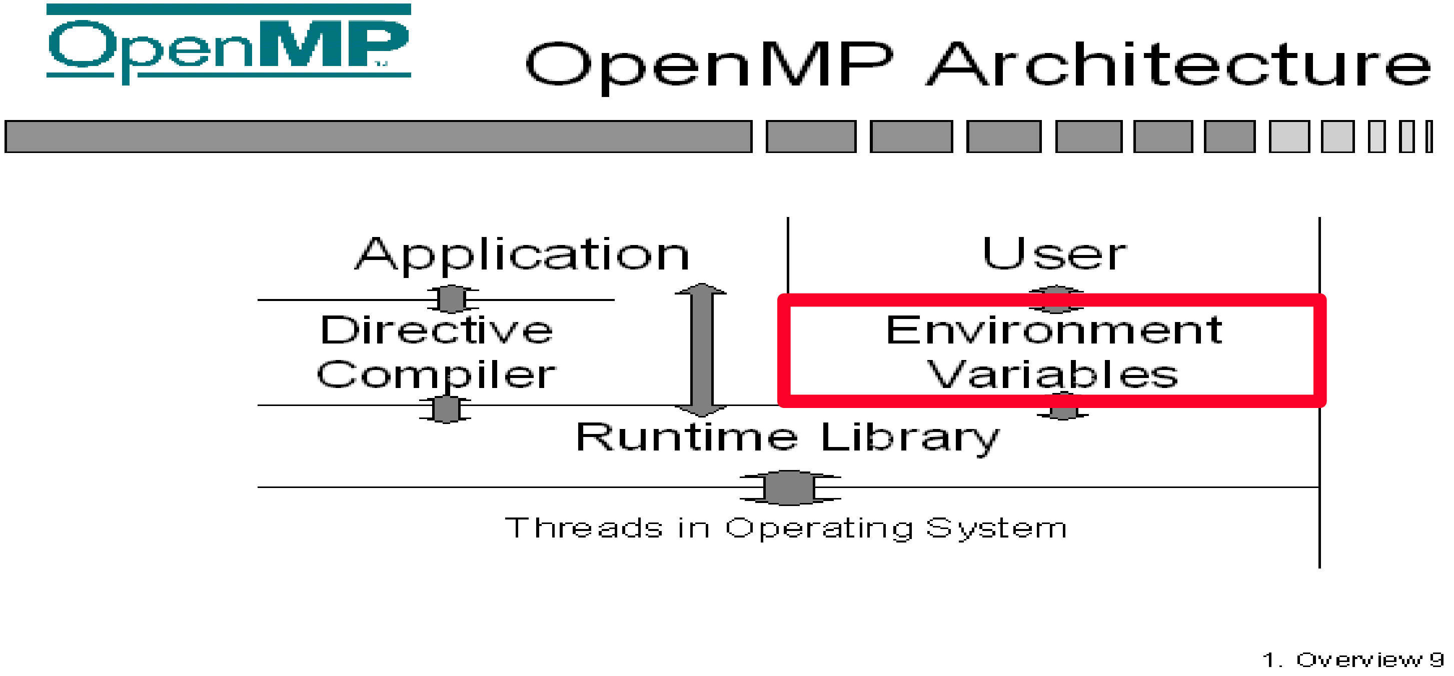
- OpenMP简介
- OpenMP编程基础
- OpenMP习题课

为什么建立OpenMP标准?

- Pthreads (IEEE Posix 1003.4a)
  - 是为低端(low end)的共享机器(如SMP)的标准
  - 对FORTRAN的支持不够
  - 适合任务并行, 而不适合数据并行
- MPI消息传递的编程标准, 对程序员要求高
- **大量已有的科学应用程序需要很好地被继承和移植**
  - **简化开发复杂度**

共享存储编程—OpenMP

- 一组**编译制导语句**和可调用的运行(run-time)**库函数**, 扩充到**基本语言**中用来表达程序中的并行性



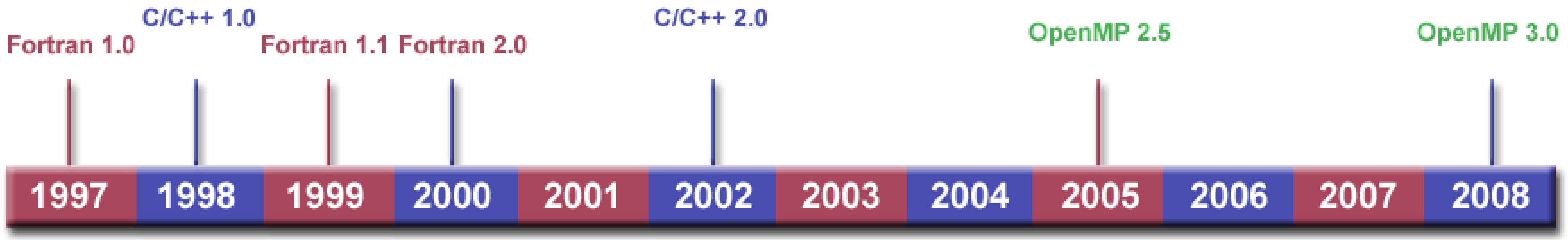
- The API is specified for C/C++ and Fortran
- Most major platforms have been implemented including Unix/Linux platforms and Windows NT

• OpenMP Is **Not**:

- Meant for distributed memory parallel systems (by itself)
- Necessarily implemented identically by all vendors
- Guaranteed to make the most efficient use of shared memory
- Required to check for data dependencies, data conflicts, race conditions, or deadlocks
- Required to check for code sequences that cause a program to be classified as non-conforming
- Meant to cover compiler-generated automatic parallelization and directives to the compiler to assist such parallelization
- Designed to guarantee that input or output to the same file is synchronous when executed in parallel. The programmer is responsible for synchronizing input and output.

OpenMP的历史

- OpenMP was born in 1996 a group formed to create an industry standard set of directives for SMP programming



- OpenMP is an evolving standard.(http://www.openmp.org)

OpenMP

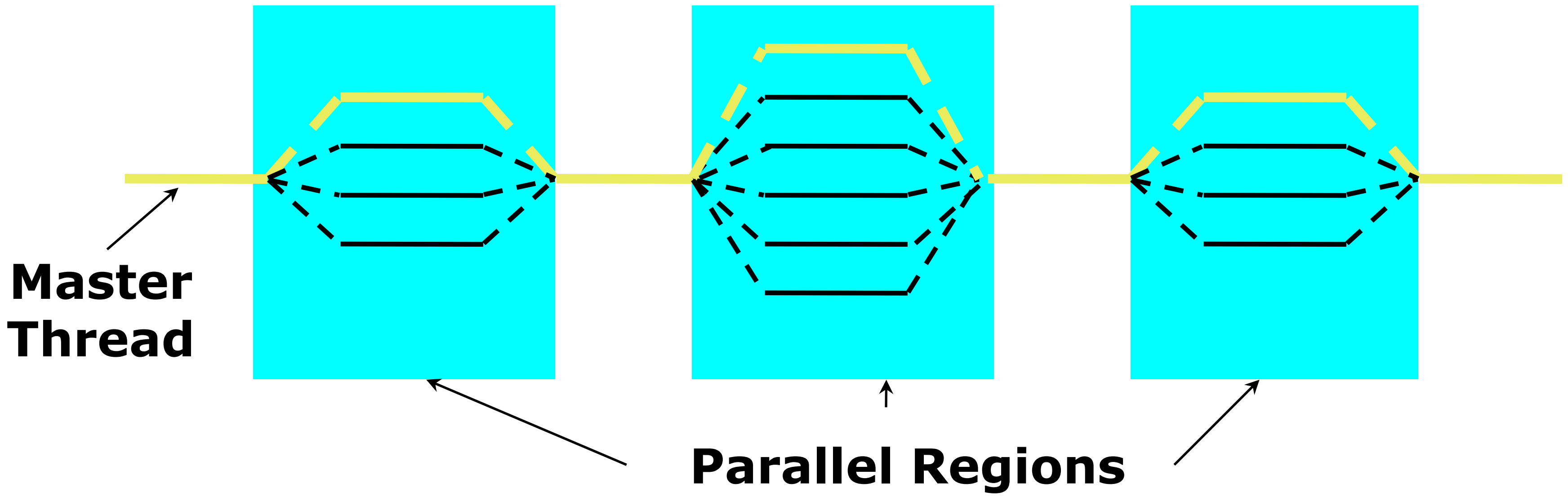
Who's Involved?

- Solution Vendors
  - *ADINA R&D, Inc.*
  - *ANSYS, Inc.*
  - *CPLEX division of ILOG*
  - *Fluent, Inc.*
  - *LSTC Corp.*
  - *MECALOG SARL*
  - *Oxford Molecular Group PLC*
- Research Organizations
  - *Purdue University*
  - *US Department of Energy ASCI Program*
- Software Vendors
  - *Absoft Corp.*
  - *Edinburgh Portable Compilers*
  - *Kuck & Associates, Inc.*
  - *Myrias Computer Technologies*
  - *Numerical Algorithms Group Ltd.*
  - *The Portland Group, Inc.*
- Hardware Vendors
  - *Compaq Computer Corp.*
  - *Hewlett-Packard Company*
  - *International Business Machines*
  - *Intel Corp.*
  - *Silicon Graphics/Cray Research*
  - *Sun Microsystems*

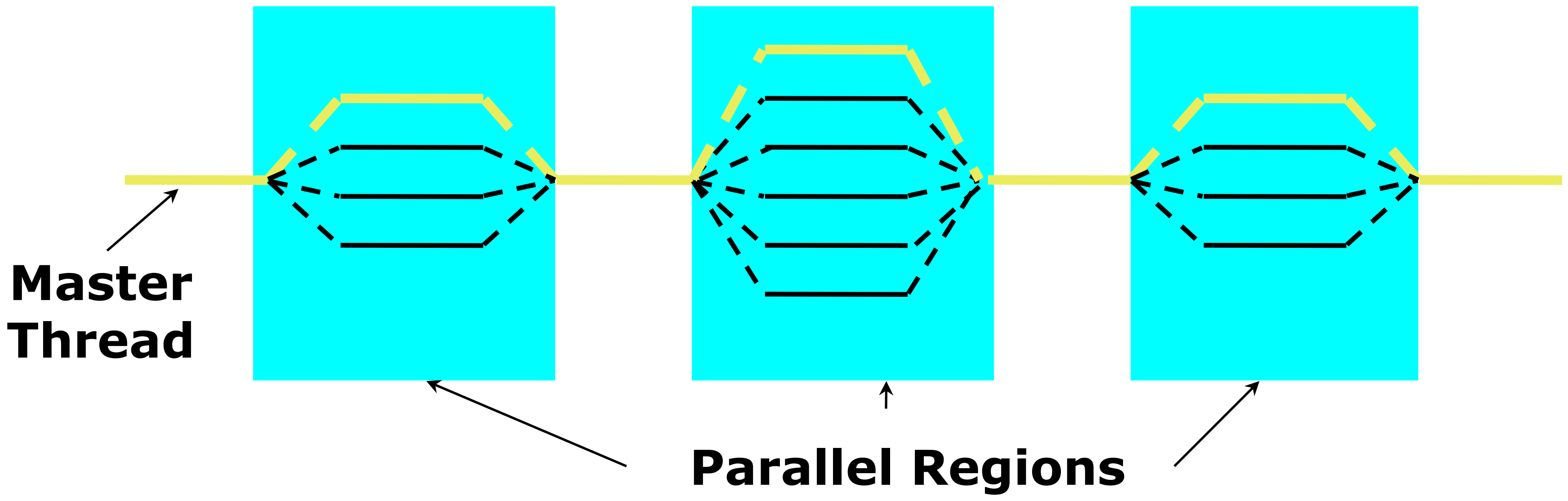
OpenMP的主要特点

- 面向共享存储体系结构，特别是SMP系统
- 显式并行方法
- 基于fork-join的多线程执行模型，但同样可以开发SPMD（Single Program Multi-Data)类型的程序
- **可以进行增量式并行开发(Incremental development)，支持条件编译(Conditional Compilation)和条件并行**
- 允许嵌套的并行性（nested Parallelism)和动态线程
    - 并不是在所有的编译器实现中支持

OpenMP的程序执行模型(Fork-Join模型)



OpenMP的程序结构



OpenMP需要支持哪些功能？

1 定义并行区

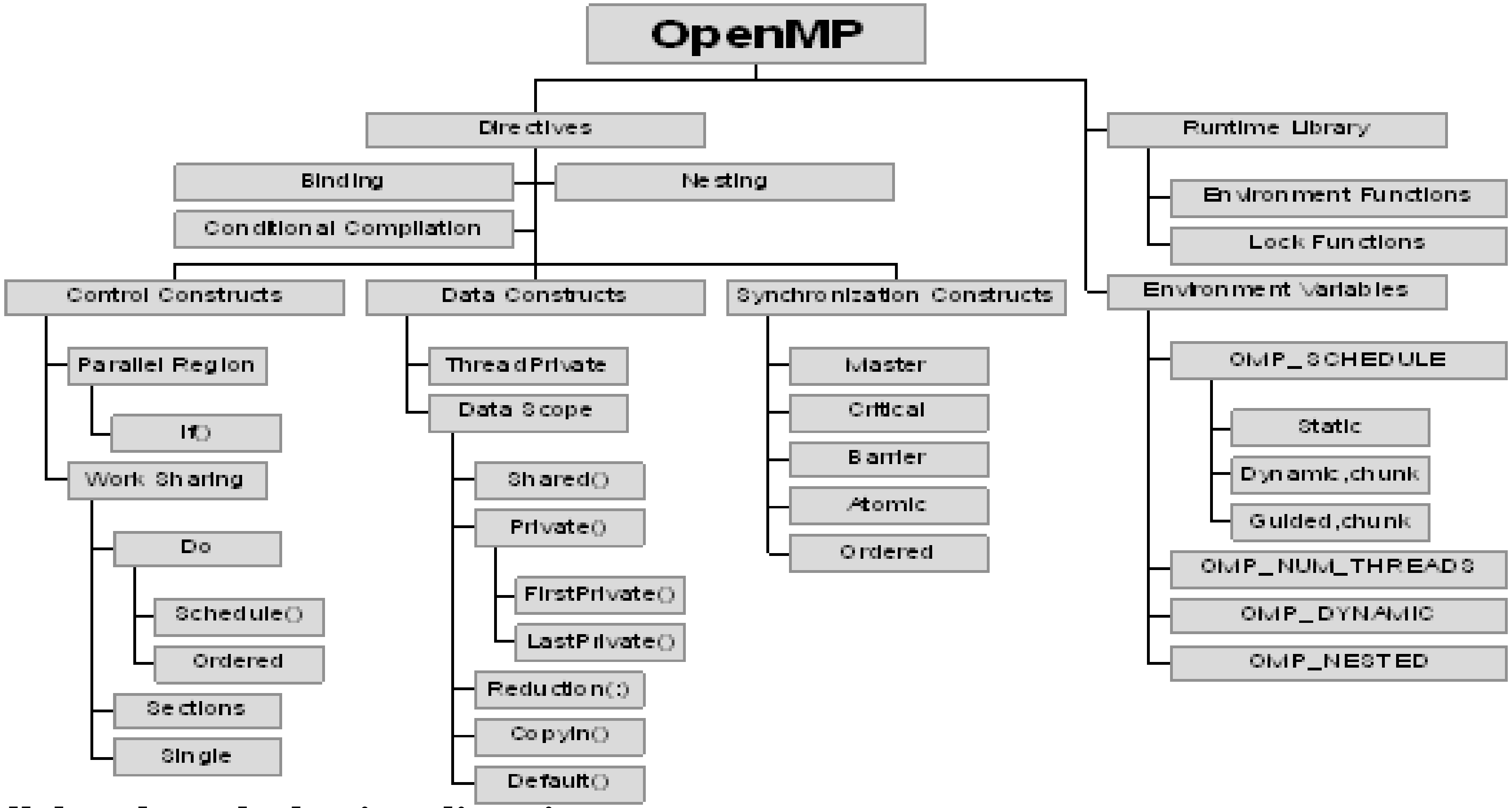
2 设置并行度

3 并行结构

4 并行区数据管理

5 同步机制

1.A.3 OpenMP Constructs



- **Parallel and work sharing directives**
- **data environment directives**
- **synchronization directives**

指导语句辨识

C/C++ (本课程以C语言作为目标语言)

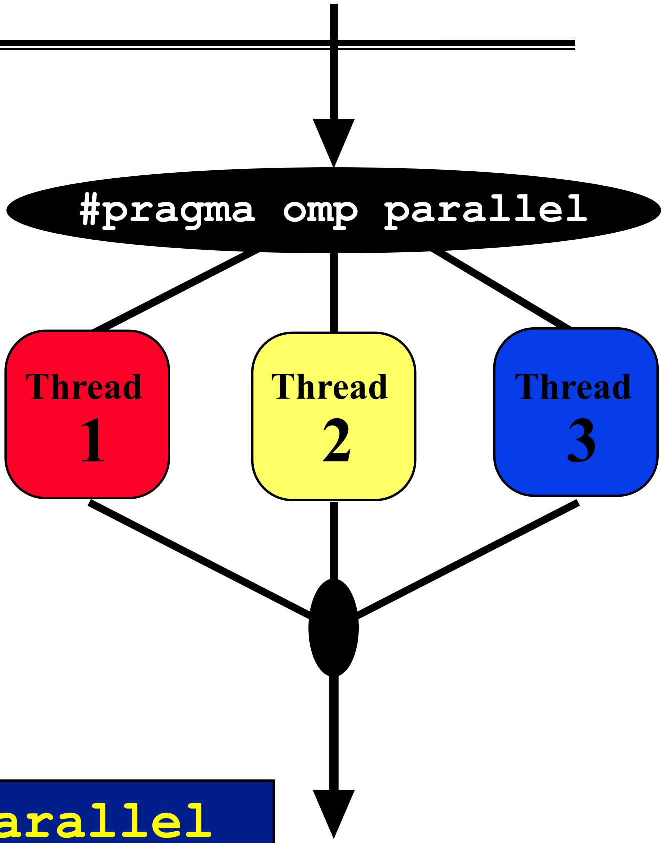
#pragma omp directive_name [clause[[,]clause]…]

Fortran

- 固定程序格式
  - **!$OMP 或**
  - **C$OMP 或**
  - ***$OMP**
  - 从程序的第一列开始，中间不能有空格
  - 第六列必须为空格，0或者是续行符（如+)
  - 指导语句的关键字之间可以不用有空格
- 自由程序格式
- **!$OMP**
  - 可以从任意列开始，中间不能有空格，续行符为&。

Parallel Regions

- Defines parallel region over structured block of code
- Threads are created as **`'parallel'`** pragma is crossed
- Threads block at end of region



C/C++ :

```
#pragma omp parallel
    {
            block
    }
```

并行块的要求

- 用OpenMP标注的块都应该是结构化的块
    - 这个块不能有多个入口（不能有跳转语句的目标在这个块中），但有可能允许多个出口。

我是谁？都有谁？(库函数)

- 线程被包含在线程组中，线程组中的线程数目可以用运行时函数调用

C/C++: int omp_get_num_threads(*void*)

Fortran: integer function omp_get_num_threads()

- 每个线程都有一个组内的线程号（从0开始），线程号为0的线程作为线程组的组长，它可以进行一些特别的操作。线程的线程号可以由运行时函数调用

C/C++: int omp_get_thread_num(*void*)

Fortran: integer function omp_get_thread_num()

How Many Processors?

- 运行时函数—程序可得到的处理器数
  - C/C++

int omp_get_num_procs(*void*)

  - Fortran

integer function omp_get_num_procs()

线程组和线程号API的说明

- Usually not needed for OpenMP codes
    - Can lead to code not being serially consistent
    - Does have specific uses (debugging)
    - Must include a header file

```
#include <omp.h>
```

设置并行度：How Many Threads?

- There is no standard default for this variable
  - Many systems:
    - # of threads = # of processors
    - Intel® compilers use this default

- Set environment variable for number of threads

```
export OMP_NUM_THREADS=4    (sh)
setenv OMP_NUM_THREADS "4"  (csh)
```

How Many Threads?

- 运行时函数
    - C/C++

void omp_set_num_threads(*int number_threads*)

- Fortran

subroutine omp_set_num_threads(*number_threads*)

integer num_threads

How Many Threads?

- NUM_THREADS(*scalar_integer_expression*)
  - 这个子句(clause)规定对应的并行块所产生的线程组的（最大）线程数目
  - 它只对这个并行块有效
  - 系统实际产生的线程数目可能会比这个值小
  - 线程数目是一个在运行时刻可以定值的整型标量表达式

线程数目的讨论

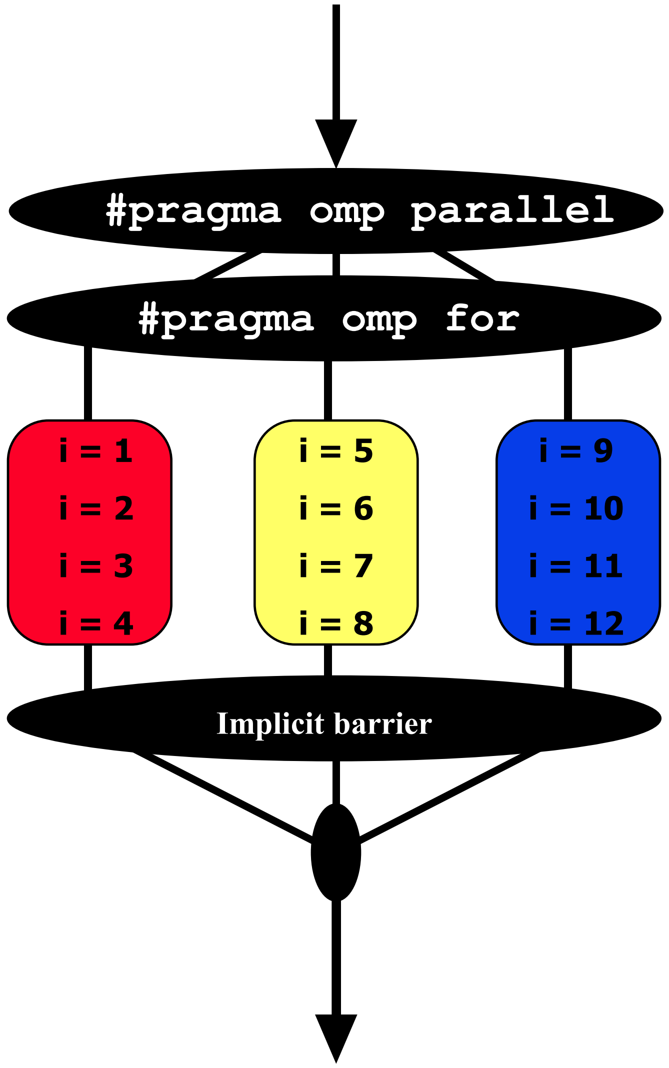
- 通常情况下线程组内线程数目由环境变量OMP_NUM_THREADS控制
- 如果parallel语句有num_threads子句，或者用户调用了omp_set_num_threads函数，线程数目由它们给出，num_threads具有高优先级
- 上述三种设置方法作用域分别为系统、并行块级以及程序级
- 这里给出的线程数目可以大于系统中处理器个数，它是一个上限值
- 系统实际产生的线程数目可能由于资源的限制而比上限值要小

- 实验1
  - ✓ 目的：启动openmp, 了解重要库函数
  - ✓ 任务
    - ✓了解并行区概念
    - ✓打印线程号和线程总数
    - ✓控制线程数量
  - ✓ 位置：c01b02:/tmp/lec04/hello_world_*.c

并行结构：Work-sharing Construct(1) - loop

- Threads are assigned an independent set of iterations
- Threads must wait at the end of work-sharing construct

```
#pragma omp parallel
#pragma omp for
   for(i = 1, i < 13, i++)
      c[i] = a[i] + b[i]
```



Combining pragmas

- These two code segments are equivalent

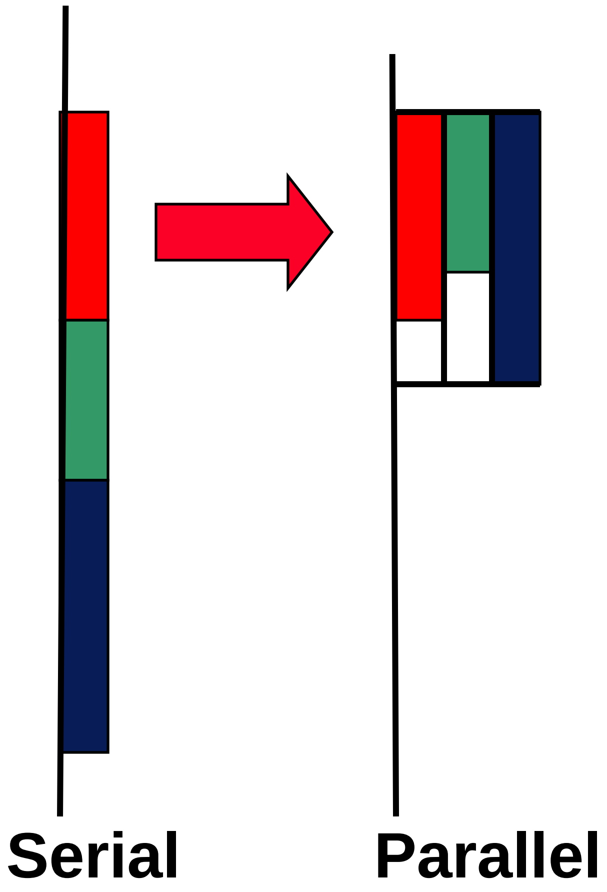
```
#pragma omp parallel
{
    #pragma omp for
    for (i=0;i< MAX; i++) {
        res[i] = huge();
    }
}
```

```
#pragma omp parallel for
    for (i=0;i< MAX; i++) {
         res[i] = huge();
    }
```

Work-sharing Construct(2): Parallel Sections

- Independent sections of code can execute concurrently

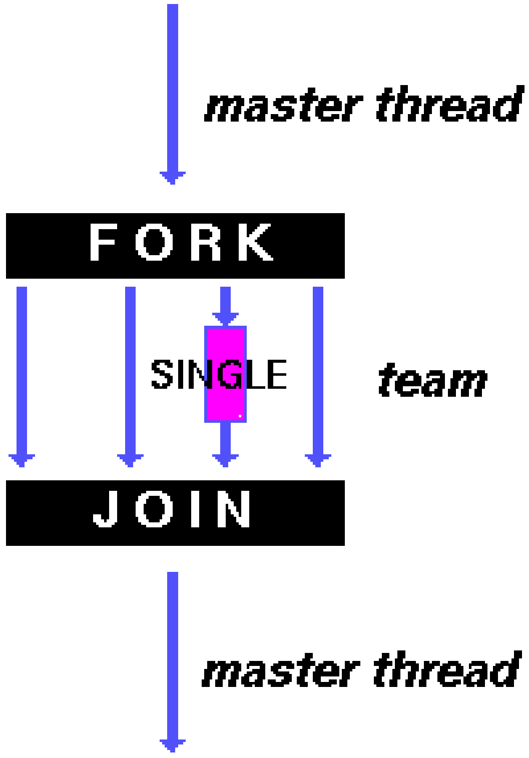
```
#pragma omp parallel sections
{
    #pragma omp section
    phase1();
    #pragma omp section
    phase2();
    #pragma omp section
    phase3();
}
```



Work-sharing Construct(3): Single Construct

- Denotes block of code to be executed by only one thread
    - First thread to arrive is chosen
- Implicit barrier at end

```
#pragma omp parallel
{
   DoManyThings();
#pragma omp single
    {
      ExchangeBoundaries();
    }  // threads wait here for single
   DoManyMoreThings();
}
```



Assigning Iterations

- The schedule clause affects how loop iterations are mapped onto threads

`schedule(static [,chunk])`
- Blocks of iterations of size "chunk" to threads
- Round robin distribution
- Default=N/t

`schedule(dynamic[,chunk])`
- Threads grab "chunk" iterations
- When done with iterations, thread requests next set
- Default=1

`schedule(guided[,chunk])`
- Dynamic schedule starting with large block
- Size of the blocks shrink; no smaller than "chunk"
- Default=1

`schedule(runtime)`
- OMP_SCHEDULE

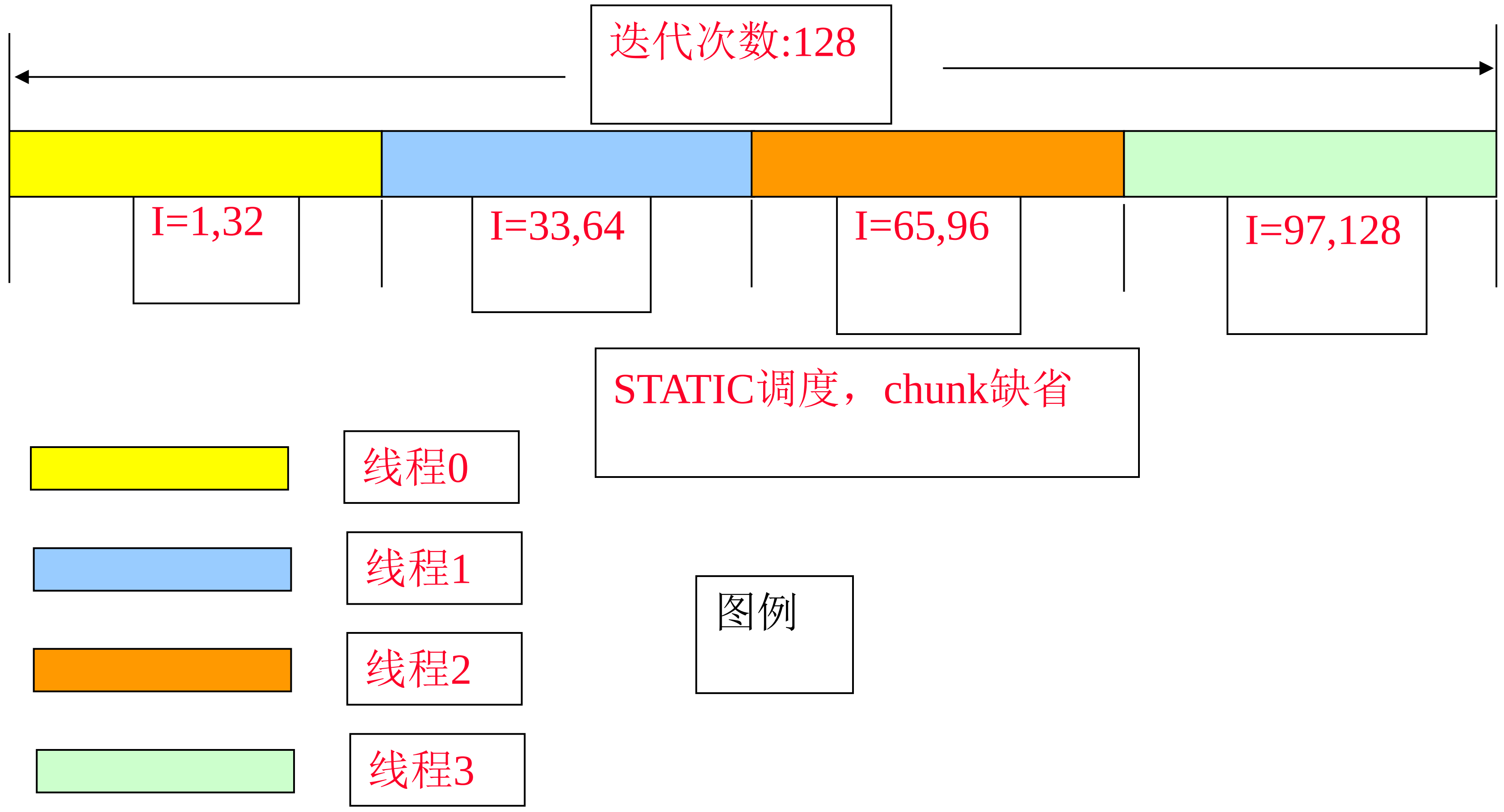
Schedule Clause Example

```
#pragma omp parallel for schedule (static)
    for( int i = 1; i <= 128; i ++ )
    {
       if ( TestForPrime(i) )  gPrimesFound++;
    }
```

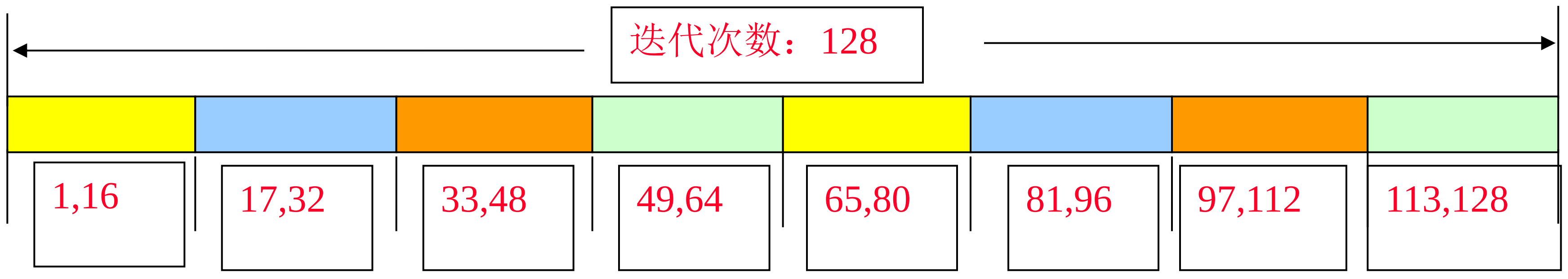
```
#pragma omp parallel for schedule (static, 8)
    for( int i = 1; i <= 128; i ++ )
    {
       if ( TestForPrime(i) )  gPrimesFound++;
    }
```

```
#pragma omp parallel for schedule (dynamic)
    for( int i = 1; i <= 128; i ++ )
    {
       if ( TestForPrime(i) )  gPrimesFound++;
    }
```

Static调度示意图(1)



Static调度示意图(2)

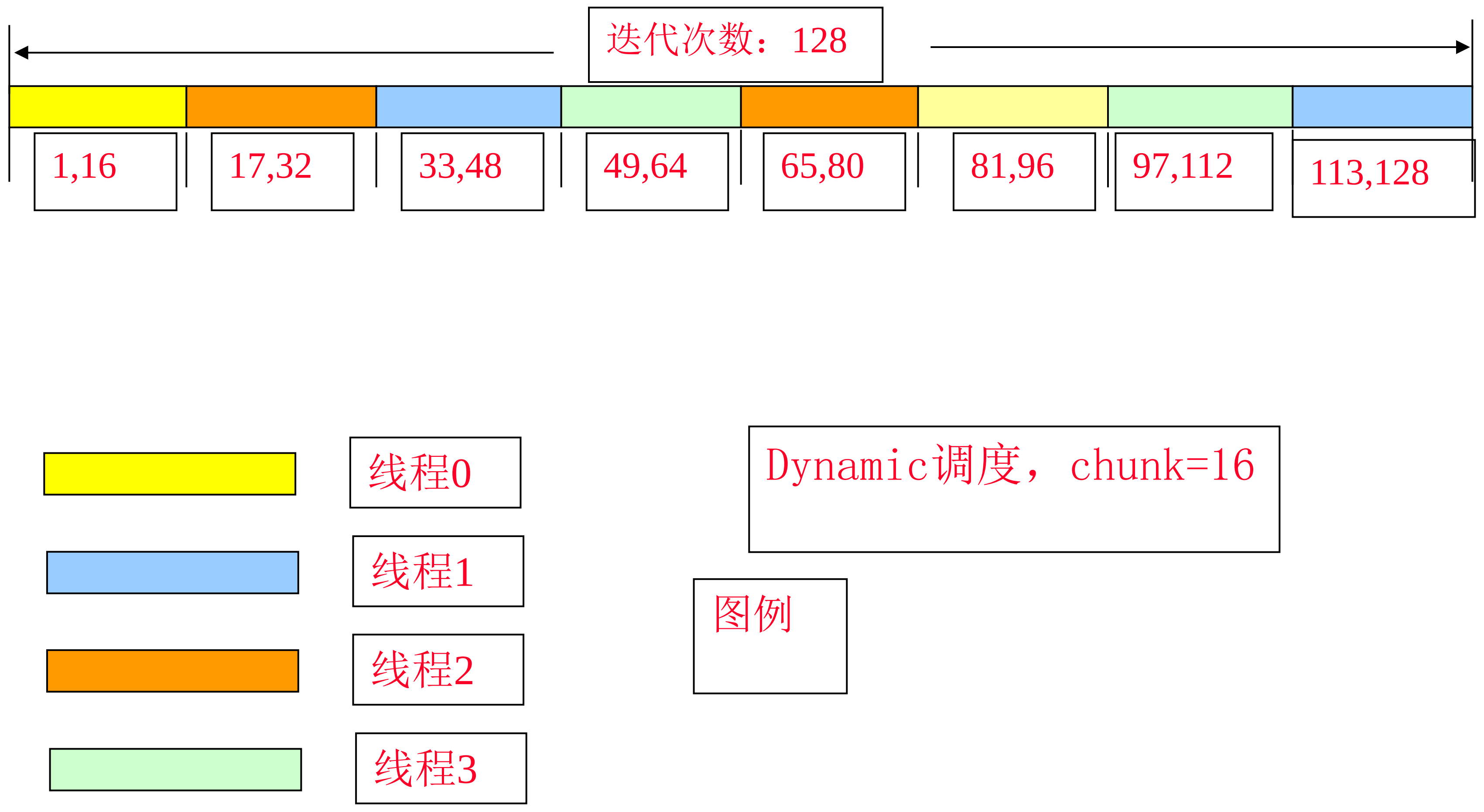


🟨	线程0
🟦	线程1
🟧	线程2
🟩	线程3

STATIC调度，chunk=16

图例

Dynamic调度示意图



Which Schedule to Use

Schedule Clause	When To Use
STATIC	Predictable and similar work per iteration
DYNAMIC	Unpredictable, highly variable work per iteration
GUIDED	Special case of dynamic to reduce scheduling overhead

- 实验2
  - ✓ 目标：测试不同调度方案
  - ✓ 任务：寻找素数
  - ✓ 位置：c01b02:/tmp/lec04/testprime_*.c

Data Environment

- OpenMP uses a shared-memory programming model

  - Most variables are shared by default.

  - Global variables are shared among threads
    - C/C++: File scope variables, static

Data Environment

- But, not everything is shared...
  - Stack variables in functions called from parallel regions are PRIVATE
  - Automatic variables within a statement block are PRIVATE
  - Loop index variables are private (with exceptions)
    - C/C+: The first loop index variable in nested loops following a **`#pragma omp for`**

Data Scope Attributes

- The default status can be modified with

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

- Scoping attribute clauses

```
shared(varname,…)

private(varname,…)
```

The Private Clause

- Reproduces the variable for each thread
    - Variables are un-initialized; C++ object is default constructed
    - Any value external to the parallel region is undefined

```
void* work(float* c, int N) {
   float x, y; int i;
  #pragma omp parallel for private(x,y)
        for(i=0; i<N; i++) {
         x = a[i]; y = b[i];
         c[i] = x + y;
        }
 }
```

Example: Dot Product

```
float dot_prod(float* a, float* b, int N)
{
  float sum = 0.0;
#pragma omp parallel for shared(sum)
   for(int i=0; i<N; i++) {
     sum += a[i] * b[i];
   }
  return sum;
}
```

What is Wrong?

Protect Shared Data

- Must protect access to shared, modifiable data

```
float dot_prod(float* a, float* b, int N)
{
  float sum = 0.0;
#pragma omp parallel for shared(sum)
   for(int i=0; i<N; i++) {
#pragma omp critical
     sum += a[i] * b[i];
   }
  return sum;
}
```

OpenMP Critical Construct

- `#pragma omp critical [(`*`lock_name`*`)]`
- Defines a critical region on a structured block

Threads wait their turn –at a time, only one calls `consum()` thereby protecting RES from race conditions

```
float RES;
#pragma omp parallel
{ float B;
#pragma omp for privte(B)
  for(int i=0; i<niters; i++){
    B = big_job(i);
#pragma omp critical
    {
    consum (B, RES);
    }
  }
}
```

OpenMP Atomic Construct

#pragma omp atomic *new-line*

expression-stmt

Expression-stmt

- *x binop*= *expr*
- *x*++
- ++*x*
- *x*--
- --*x*

- **Binop 包括 +, *, -, /, &, ^, |, <<,** or **>>.**

OpenMP Reduction Clause

- `reduction (`*`op`*` : `*`list`*`)`

- The variables in "*list*" must be shared in the enclosing parallel region

- Inside parallel or work-sharing construct:
  - A PRIVATE copy of each list variable is created and initialized depending on the "op"
  - These copies are updated locally by threads
  - At end of construct, local copies are combined through "op" into a single value and combined with the value in the original SHARED variable

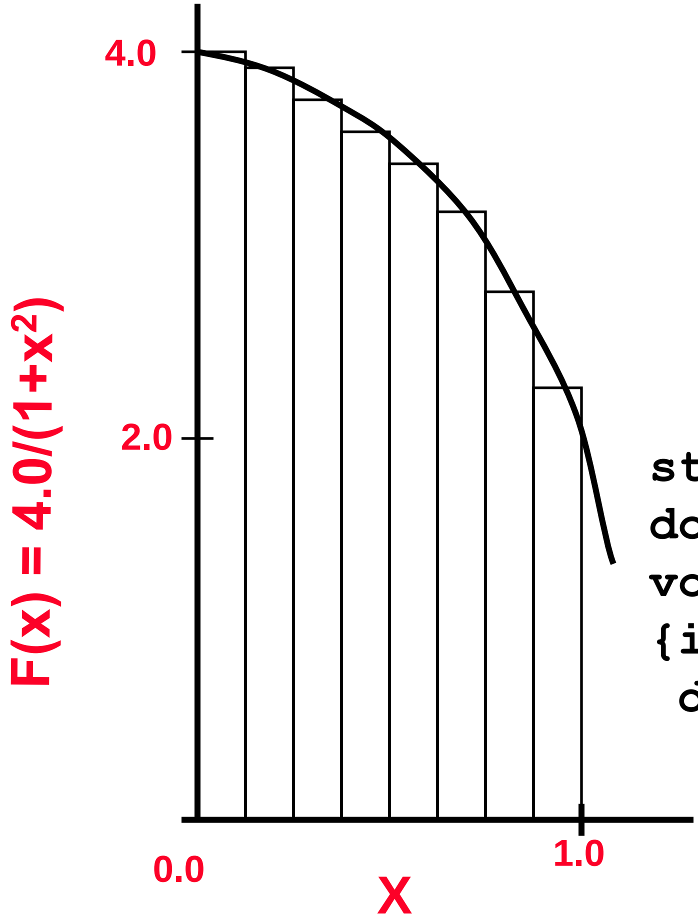
C/C++ Reduction Operations

- A range of associative operands can be used with reduction
- Initial values are the ones that make sense mathematically

Operand	Initial Value
+	0
*	1
-	0
^	0

Operand	Initial Value
&	~0
|	0
&&	1
||	0

实验三



$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

```
static long num_steps = 100000;
double step;
void main ()
{int I;
 double x, pi, sum = 0.0;
     step = 1.0/(double) num_steps;
     for (i=0; i< num_steps; i++){
        x = (i+0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
     }
     pi = step * sum;
}
```

- 实验3
  - ✓ 目标：测试不同同步方案，那种最快？
  - ✓ 任务：数值积分求解pi
  - ✓ 位置：/tmp/lec04/ pi_*.c

Barrier Construct

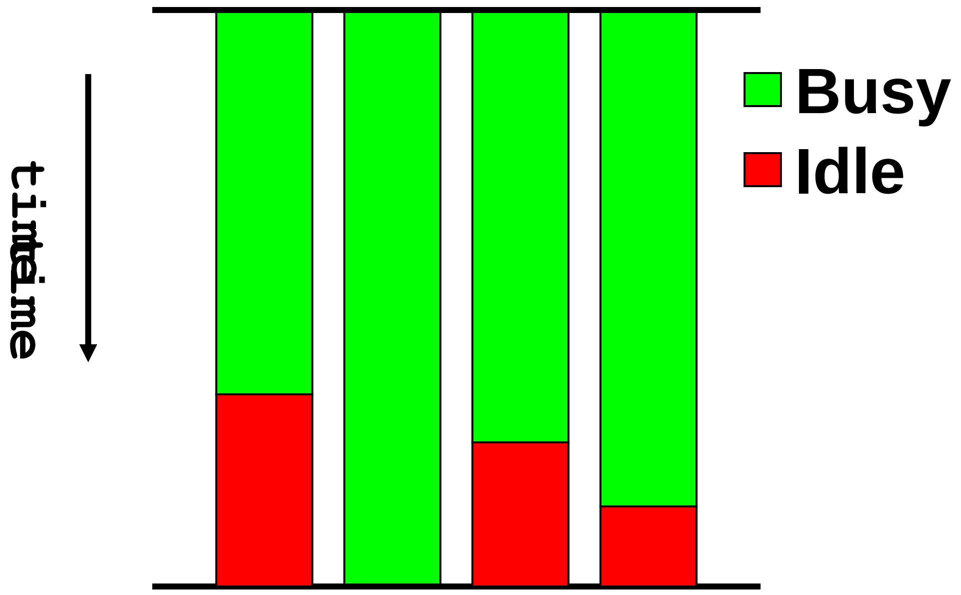
- Explicit barrier synchronization
- Each thread waits until all threads arrive

```
#pragma omp parallel shared (A, B, C)
{
       DoSomeWork(A,B);
       printf("Processed A into B\n");
#pragma omp barrier
       DoSomeWork(B,C);
       printf("Processed B into C\n");
}
```

What is the problem with Barrier ? Pros and Cons

- For correctness

- For performance
    - Unnecessary barriers hurt performance
        - Waiting threads accomplish no work!



Implicit Barriers

- Several OpenMP* constructs have implicit barriers
  - **`parallel`**
  - **`for`**
  - **`single`**

- Suppress implicit barriers, when safe, with the **`nowait`** clause

解决Barrier的性能问题 -- Nowait Clause

- Use when threads would wait between independent computations

```
#pragma omp for nowait
    for(...)
      {...};
```

```
#pragma single nowait
{ [...] }
```

```
#pragma omp for schedule(dynamic,1) nowait
 for(int i=0; i<n; i++)
   a[i] = bigFunc1(i);

#pragma omp for schedule(dynamic,1)
 for(int j=0; j<m; j++)
   b[j] = bigFunc2(j);
```

共享变量真的共享了吗？

Memory Model

- OpenMP provides a "relaxed-consistency" and "temporary" view of thread memory (in their words). In other words, threads can "cache" their data and are not required to maintain exact consistency with real memory all of the time.

- When it is critical that all threads view a shared variable identically, the programmer is responsible for insuring that the variable is FLUSHed by all threads as needed.

flush

- The FLUSH directive identifies a synchronization point at which the implementation must provide a consistent view of memory. Thread-visible variables are written back to memory at this point.

Fortran	!$OMP FLUSH *(list)*
C/C++	#pragma omp flush *(list) newline*

默认flush情况

Fortran	C/C++

BARRIER
END PARALLEL
CRITICAL and END CRITICAL
END DO
END SECTIONS
END SINGLE
ORDERED and END ORDERED
	barrier
	parallel - upon entry and exit
	critical - upon entry and exit
	ordered - upon entry and exit
	for - upon exit
	sections - upon exit
	single - upon exit

小结

- OpenMP* is:
  - A simple approach to parallel programming for shared memory machines
  - Fork-join model
- We explored basic OpenMP coding on how to:
  - 定义并行区 (**omp parallel**)
  - 设置并行度
  - 并行结构 (**omp for; omp (parallel) sections; single**)
  - 任务分配(schedule)
  - 数据管理/变量分类 (**omp private/shared, flush**)
  - 同步控制 (**omp critical**…)

文献阅读

- https://computing.llnl.gov/tutorials/openMP/

- 最新的编程接口见The Latest Official OpenMP specification - Version 3.1 , http://openmp.org/wp/openmp-specifications/

第一次作业

- 使用OpenMP对串行程序pi_serial.c进行并行及优化，并对并行之后的程序进行性能分析
  - 完成多种OpenMP实现
  - 分析性能结果
    - 不同实现的性能差别原因何在？
    - 能否用更多的核获得线性加速效果？为什么？

Backup for use

Master Construct

- Denotes block of code to be executed only by the master thread
- No implicit barrier at end

```
#pragma omp parallel
{
   DoManyThings();
#pragma omp master
   {                  // if not master skip to next stmt
     ExchangeBoundaries();
   }
   DoManyMoreThings();
}
```

More OpenMP*

- Data environment constructs
    - **`FIRSTPRIVATE`**
    - **`LASTPRIVATE`**
    - **`THREADPRIVATE`**

Firstprivate Clause

- Variables initialized from shared variable
- C++ objects are copy-constructed

```
incr=0;
#pragma omp parallel for firstprivate(incr)
for (I=0;I<=MAX;I++) {
      if ((I%2)==0) incr++;
      A(I)=incr;
}
```

Lastprivate Clause

- Variables update shared variable using value from last iteration
- C++ objects are updated as if by assignment

```
void sq2(int n,
      double *lastterm)
{
  double x; int i;
  #pragma omp parallel
  #pragma omp for lastprivate(x)
  for (i = 0; i < n; i++){
     x = a[i]*a[i] + b[i]*b[i];
     b[i] = sqrt(x);
  }
  lastterm = x;
}
```

Threadprivate Clause

- Preserves global scope for per-thread storage
- Legal for name-space-scope and file-scope
- Use copyin to initialize from master thread

```
struct Astruct A;
#pragma omp threadprivate(A)
…

#pragma omp parallel copyin(A)
  do_something_to(&A);
…

#pragma omp parallel
  do_something_else_to(&A);
```

Private copies of "A" persist between regions