華東師範大學 | 数学科学学院
School of Mathematical Sciences, East China Normal University

OpenMP 并行编程（二）

—— 工作共享结构
—— 同步与数据环境

目录页

Contents

目录页

Contents

1 工作共享结构（续）

- sections / section
- single
- master
- task（略）

内容提要

OpenMP 编译制导

- 工作共享结构：

  for，sections，single，master，task，workshare

- 同步指令:

  critical，barrier，atomic，flush，ordered

- 数据环境指令

  threadprivate

- 子句：

  private，firstprivate，… …

SECTIONS 结构

Fortran	**!$omp sections [clause clause ...]** **!$omp section** ***structured-block*** **!$omp section** ***structured-block*** **!$omp end sections [nowait]**
C/C++	**#pragma omp sections [clause clause ...]** **{** **#pragma omp section** ***structured-block*** **#pragma omp section** ***structured-block*** **. . . . . .** **}**

- **sections** 也可以与 **parallel** 合并，即 **#pragma omp parallel sections**

SECTIONS 结构

- 指令 sections 创建一个工作共享域
- 域中的子任务由指令 section 创建，必须是一个相对独立的完整代码块
- 每个子任务都将只被一个线程执行
- 结束处隐含障碍同步，除非显式指明 nowait
- sections 可用的字句有

private(*list*)
firstprivate(*list*)
lastprivate(*list*)
reduction(*op* : *list*)
nowait

SECTIONS 示例

```
#pragma omp parallel num_threads(3)
    #pragma omp sections
    {
        #pragma omp section
            printf("Hello world!\n");
        #pragma omp section
            printf("Hello Math!\n");
        #pragma omp section
            printf("Hello OpenMP!\n");
    }
```

OMP_sections_01.c
OMP_sections_02.c

- 注意 sections 和 section 的区别
- #pragma omp section 必须在 sections 域中
- 共享域中的每个子任务必须由 section 指令创建
- 第一个子任务前面的 section 指令可以省略
- 子任务个数小于线程个数时，多余的线程空闲等待
- 子任务个数大于线程个数时，任务分配由编译器指定，尽量负载平衡

SINGLE 结构

Fortran	**!$omp single [clause clause ...]** ***structured-block*** **!$omp end single [nowait copyprivate(*list*)]**

C/C++	**#pragma omp single [clause clause ...]**
	{
	structured-block
	}

- 用在并行域中，指定代码块只能由一个线程执行（不一定是主线程）
- 第一个遇到 single 指令的线程执行相应的代码，其它线程则在 single 结尾处等待，除非显式指明 nowait
- 可用的字句有

private(*list*)
firstprivate(*list*)
copyprivate(*list*) // 将串行计算的值广播给并行域中的同名变量
nowait

MASTER 结构

Fortran	!$omp master *structured-block* !$omp end master

C/C++	#pragma omp master
	{
	structured-block
	}

- master 块仅由线程组中的主线程执行
- 其它线程跳过并继续执行下面的代码，即结尾处没有隐式同步
- 通常用于 I/O
- 与 single nowait 的区别：

  master 指定由主线程执行，而 single 由最先到达的线程执行

华东师范大学 数学科学学院
School of Mathematical Sciences, ECNU

2 同步结构 Synchronization Constructs

- critical
- barrier
- atomic
- flush
- ordered
- taskwait/taskyield（略）

CRITICAL 结构

Fortran	!$omp critical [(name)] *structured-block* !$omp end critical [(name)]

C/C++	#pragma omp critical [(name)]
	{
	structured-block
	}

- critical 块（临界区）限定同一时间只能有一个线程执行
- 所有线程将依次执行 critical 块
- 主要用于共享变量的更新，写文件等，避免数据竞争
- 并行域中可以包含多个 critical 块
- 线程到达临界区入口时等待，直到没有其它线程执行同名 critical 块
- 可以给每个 critical 块起名字
- 同名的临界区被看作是一个整体：同一时间，同名块中只能有一个线程
- 所有没有名字的 critical 块被看作是一个整体

CRITICAL示例

```
#pragma omp parallel num_threads(4) private(i,tid,mysum) shared(a,h,mypi)
{
    tid = omp_get_thread_num();

    #pragma omp for
        for(i=1; i<n; i++)
        {
            mysum= mysum + f(a+i*h);
        }
    #pragma omp critical
        mypi = mypi + mysum;
}

mypi = mypi + (f(a) + f(b))/2;
mypi = h*mypi;
```

OMP_critical_pi.c

BARRIER 结构

Fortran	!$omp barrier
C/C++	#pragma omp barrier

- 障碍同步：线程遇到该指令都停下来等待，直到同组所有线程都到达该点，然后再继续执行后面的代码

- barrier 指令必须放在同组所有线程都能到达或都不能到达的地方，否则可能会引起死锁！比如不能放在 single，critical，master，section 中！在循环或选择结构中使用时也需要注意，如：

```
if (myid<5) then
   #pragma omp barrier
end
```

ATOMIC 结构

Fortran	!$omp atomic [clause]
C/C++	#pragma omp atomic [clause]

- atomic 指令保证对某个存储地址做原子改写，不允许多个线程同时改写。
- 该指令仅作用于紧随其后的一条语句，所支持的运算参见 OpenMP 手册

支持的子句： read | write | update | capture

- 使用 atomic 和 critical 都可以避免数据竞争
- 使用 atomic 可能比 critical 更高效：
  使用 atomic 指令可以并行地更新数组内的不同元素，
  而使用 critical 的话，则只能串行执行

原子改写的含义是指：读取该存储地址的内容、做所需运算、然后把新值写回该存储地址这一连串操作不会被其它线程间断，它保证所有操作要么全部完成，要么保持原封不动。

FLUSH 结构

Fortran	!$omp flush [(*list*)]
C/C++	#pragma omp flush [(*list*)]

- flush 标识数据同步点，在这些点上，系统将提供一致的内存视图，在该指令出现的点上，共享变量被写回内存；
- 不带 list，则表示针对所有变量；
- 下列指令隐含不带 list 的 flush 指令：barrier，critical，parallel，ordered

注：若有 nowait 子句，则不再隐含 flush 指令。

ORDERD 结构

Fortran	!$omp ordered *structured-block* !$omp end ordered

C/C++	#pragma omp ordered
	{
	structured-block
	}

- 指定循环必须按串行时的顺序执行
- 作用与 critical 类似，但指定了循环执行顺序
- ordered 只能出现在 for 或 parallel for 指定的循环区
- 此时 for 或 parallel for 需要加子句 ordered
- 一个循环体中最多只能有一个 ordered 块

ORDERD 示例

```
void example_ordered(int a, int b)
{
    #pragma omp parallel for ordered num_threads(4)
        for(int i=a; i<=b; i++)
        {
            printf("*tid=%-2d, %-2d\n", omp_get_thread_num(), i);
            #pragma omp ordered
                printf(" tid=%-2d, %-2d\n", omp_get_thread_num(), i);
        }
}

int main()
{
    example_ordered(1,8);

    return 0;
}
```

OMP_ordered.c

华东师范大学 数学科学学院
School of Mathematical Sciences, ECNU

3 数据环境结构
Data Environment Constructs

- threadprivate
- copyin

数据环境结构

■ 数据环境指令 Data Environment Constructs

Fortran	**!$omp threadprivate(*list*)**
C/C++	**#pragma omp threadprivate(*list*)**

指定变量或公共数据块是线程私有的，且在同一个线程内是全局的（多个并行域）

- **threadprivate 需出现在变量声明之后，其它语句之前**
- **变量需具有静态生存期，如全局变量或静态局部变量**
- **数据在同一个线程内是全局的**

THREADPRIVATE示例

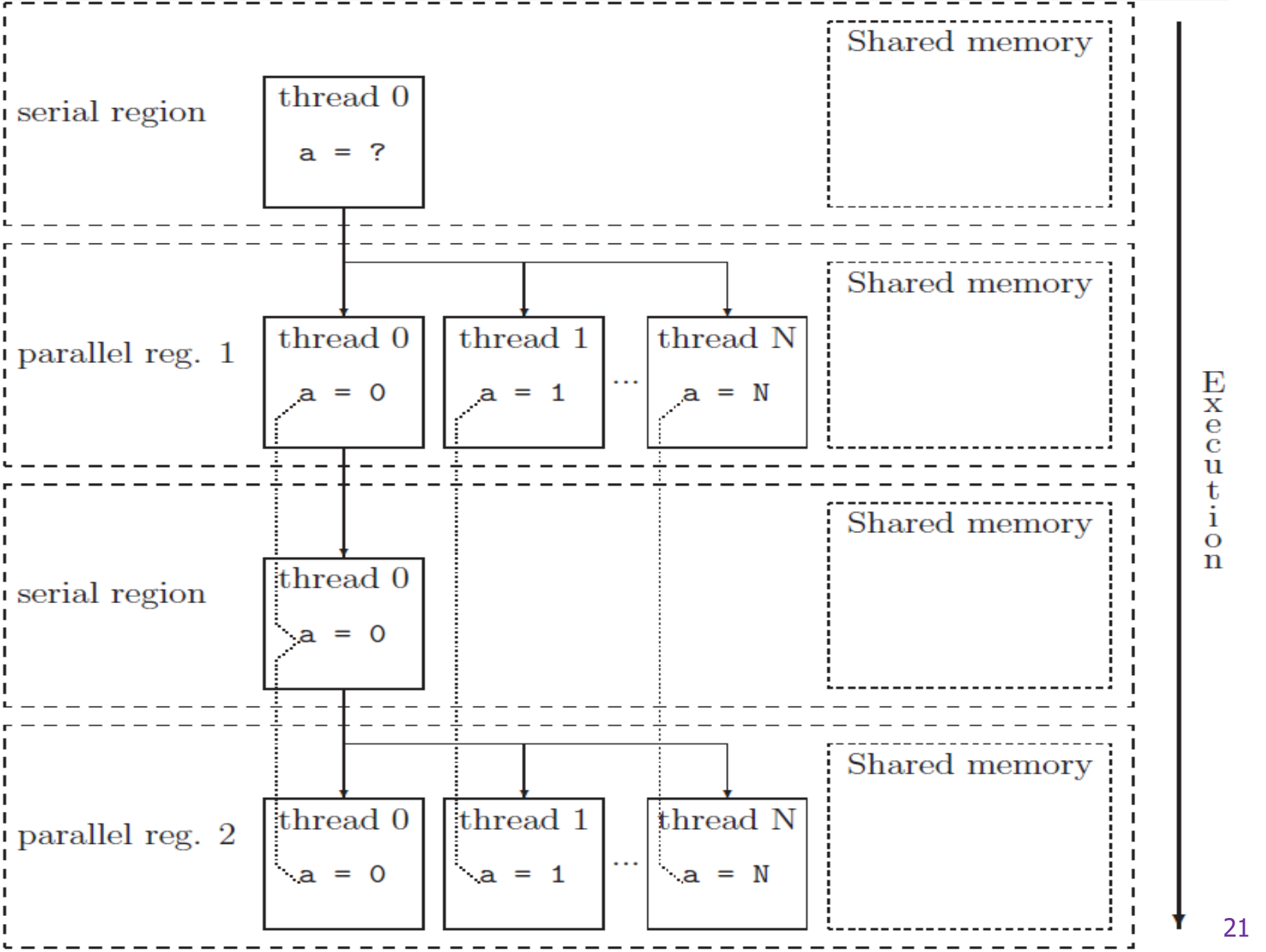
```
int a;
#pragma omp threadprivate(a)

int main()
{
    ... ...
    #pragma omp parallel // 第一个并行域
    {
        a=omp_get_thread_num();
    }

    ... ...

    #pragma omp parallel // 第二个并行域
    {
       ... ...
    }
    ... ...
}
```

- 变量 a 在第一个并行域中获取不同的值
- 在第二个并行域内，a 保持原有的值，见后面的示意图


serial region
thread 0
a = ?
Shared memory
parallel reg. 1
thread 0
a = 0
thread 1
a = 1
thread N
a = N
Shared memory
serial region
thread 0
a = 0
Shared memory
parallel reg. 2
thread 0
a = 0
thread 1
a = 1
thread N
a = N
Shared memory
Execution

COPYIN子句

copyin(list)

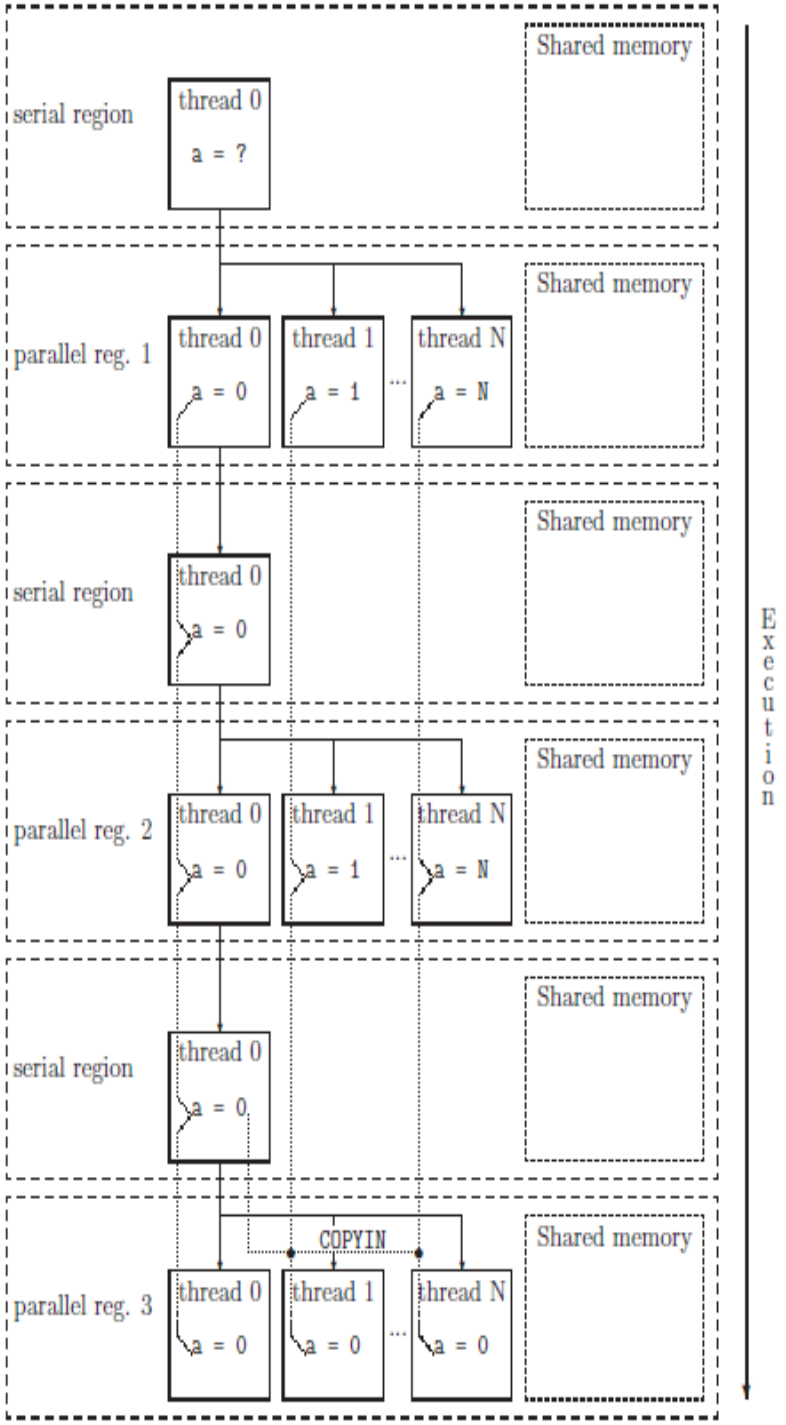
- 配合 threadprivate，用主线程同名变量的值对并行域中的 threadprivate 变量进行初始化，使得所有这些私有变量在个线程中的值相同

```
int a;
#pragma omp threadprivate(a)

#pragma omp parallel // 第一个并行域
{   a=omp_get_thread_num(); }
. . . . . .

#pragma omp parallel // 第二个并行域
{    . . . . . .  }
. . . . . .

#pragma omp parallel copyin(a)
{   . . . . . .  }
```



共享结构与子句小结

子句	共享结构					
	PARALLEL	for	SECTIONS	SINGLE	PARALLEL for	PARALLEL SECTIONS
IF	✓				✓	✓
PRIVATE	✓	✓	✓	✓	✓	✓
SHARED	✓	✓			✓	✓
DEFAULT	✓				✓	✓
FIRSTPRIVATE	✓	✓	✓	✓	✓	✓
LASTPRIVATE		✓	✓		✓	✓
REDUCTION	✓	✓	✓		✓	✓
COPYIN	✓				✓	✓
COPYPRIVATE				✓		
SCHEDULE		✓			✓	
ORDERED		✓			✓	
NOWAIT		✓	✓	✓		

指令与子句小结

- 不带子句的编译制导指令
  - MASTER
  - CRITICAL
  - BARRIER
  - FLUSH
  - ORDERED
  - THREADPRIVATE

共享结构指令绑定规则

① **for**, **sections**, **single**, **master** 和 **barrier** 指令绑定到动态的封装 **parallel** 中，如果没有并行域，这些语句是无效的。

② **ordered** 指令绑定到包围它的动态 **for** 中。

③ **atomic** 指令迫使所有线程做互斥访问，而不仅是当前组里的线程。

④ **critical** 指令迫使所有的线程做互斥访问，而不仅是当前组里的线程。

⑤ 指令总是绑定到包围它的最内层 **parallel** 中。

指令嵌套

① 动态的位于另一个 **parallel** 中的 **parallel** 指令逻辑上建立一个新的组，如果不允许嵌套并行，则这个新组仅由当前线程执行。

② 受同一 **parallel** 控制的 **for**，**sections**，**single** 不允许彼此嵌套。

③ **for**，**sections** 和 **single** 不允许出现在 **critical** 和 **master** 的动态区域中。

④ **barrier** 不允许出现在 **for**，**sections**，**single**，**master** 和 **critical** 的动态区域中。

⑤ **master** 不允许出现在 **critical** 区的动态区域中。

⑥ **ordered** 不允许出现在 **critical** 区的动态区域中。

⑦ 可以在并行域的动态区域中出现的指令，也可在并行域的动态区域外出现，但它仅由主线程执行。

举例

如何修改下面的代码，使之能正确、高效率地并行执行？

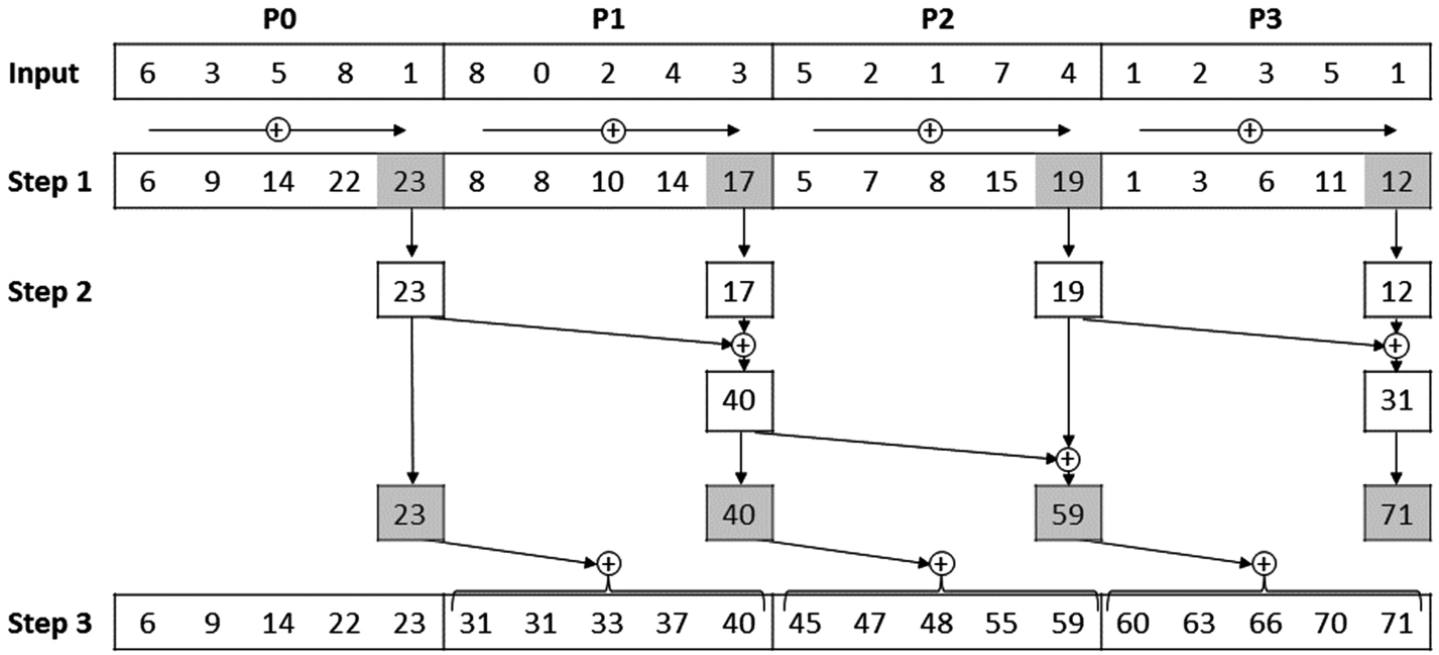
```
long a=0;
#pragma omp parallel for private(i)
for(i=1;i<=100000000; i++)
{
    a=a+1;
}
```

OMP_atomic_01.c
OMP_atomic_02.c

提示：critical / atomic

课堂作业

给定一个数组 $a_1, a_2, a_3, \ldots$, 计算 $a_i = a_i + a_{i-1}, \quad i = 2,3,\ldots$



（程序取名 **quiz02_xxx.c**，程序第一行：**//学号-姓名**）

（发送至 **hwmath@126.com**，邮件主题：**quiz02-学号-姓名**）

THANK YOU