Multi-core Programming

Lecture 07. **Components of OpenMP**

Today's Topics

- What is OpenMP?
- The OpenMP Execution Model
- Components of OpenMP

- Refer to OpenMP 2.5 specification
  - http://www.openmp.org/mp-documents/spec25.pdf

(也可以参考 3.0 和 3.1 版的 Specification)

What is OpenMP?

- Defacto standard API for writing multithreaded programs in C/C++ and Fortran
- Consists of
  - Compiler directives
  - Runtime routines
  - Environment Variables

A first OpenMP example

For-loop with independent iterations

```
for (i = 0; i < n; i++)
  c[i] = a[i] + b[i];
```

相互独立的迭代

Compiler directives

data-sharing

① Parallel for (没有指明如何分割)

② shared

③ private

④ 环境变量.

For-loop parallelized using an OpenMP pragma

```
#pragma omp parallel for  \
        shared(n, a, b, c)\
        private(i)
for (i = 0; i < n; i++)
    c[i] = a[i] + b[i];
```

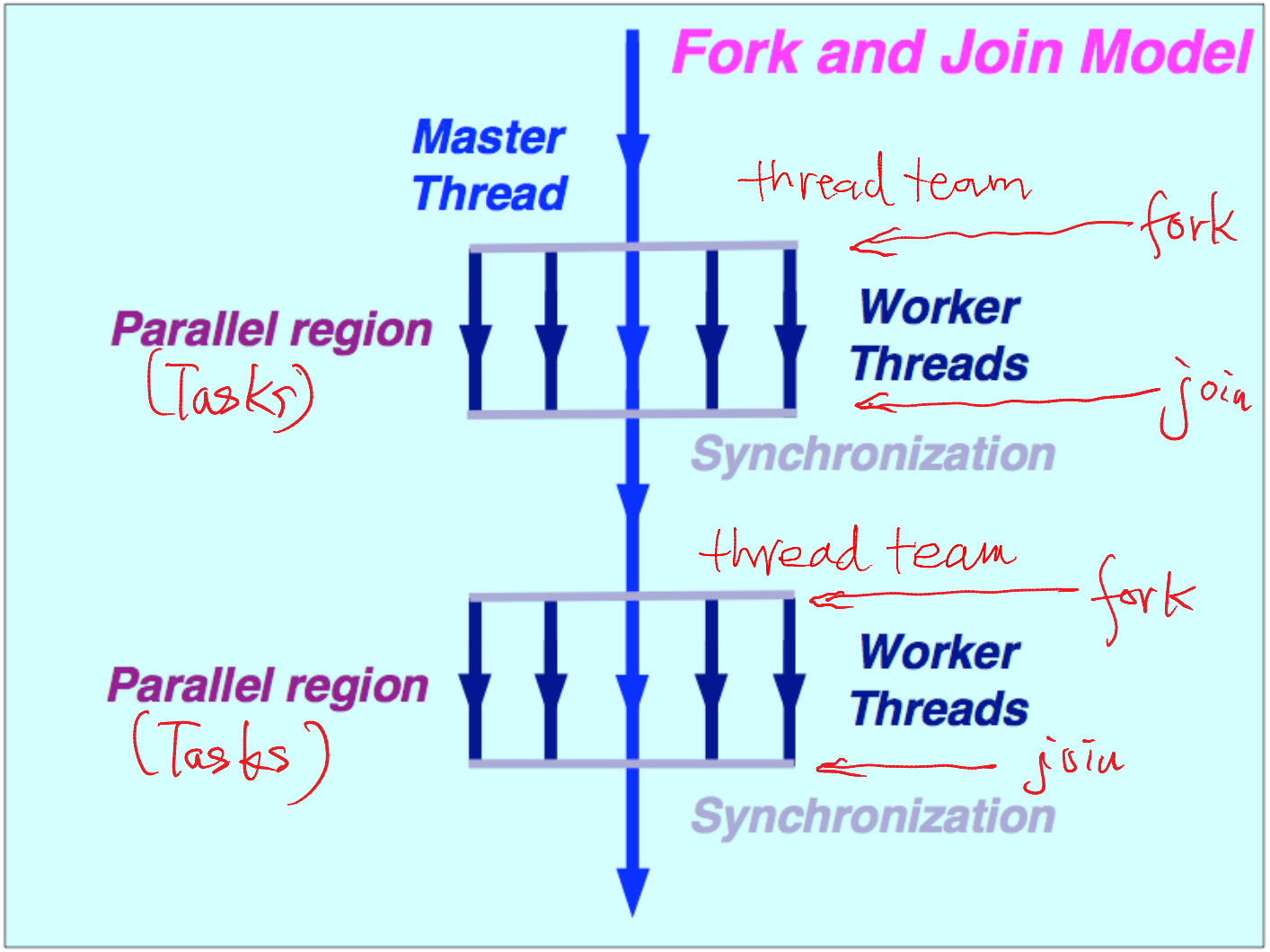
并行化 for loop.

```
% cc -xopenmp source.c
% setenv OMP_NUM_THREADS 4
% a.out
```

环境变量

Fork-Join 執行模型

The OpenMP Execution Model



Terminology

- *OpenMP Team := Master + Workers*
- *A Parallel Region is a block of code executed by all threads simultaneously*
  - *The master thread always has thread ID 0*
  - *Thread adjustment (if enabled) is only done before entering a parallel region*
  - *Parallel regions can be nested, but support for this is implementation dependent*
  - *An "if" clause can be used to guard the parallel region; in case the condition evaluates to "false", the code is executed serially*
- *A work-sharing construct divides the execution of the enclosed code region among the members of the team; in other words: they split the work*

A first OpenMP example

For-loop with independent iterations

```
for (i = 0; i < n; i++)
  c[i] = a[i] + b[i];
```

Parallel Region and work-sharing constructs

For-loop parallelized using an OpenMP pragma

```
#pragma omp parallel for  \
        shared(n, a, b, c)\
        private(i)
for (i = 0; i < n; i++)
    c[i] = a[i] + b[i];
```

```
% cc -xopenmp source.c
% setenv OMP_NUM_THREADS 4
% a.out
```

Components of OpenMP

- Parallel regions
  - master & worker threads
- Work-sharing constructs
  - parallel loop w/ schedule clauses
  - parallel sections
  - Single
- Synchronization
  - nowait
  - Barrier
- Data-sharing attributes
  - private, firstprivate, lastprivate, shared, reduction, threadprivate
- Runtime routines
  - omp_get_thread_num (), int omp_get_num_threads (), omp_in_parallel()
- Environment variables
  - OMP_NUM_THREADS, OMP_SET_DYNAMIC, OMP_NESTED, OMP_SCHEDULE

Components of OpenMP

Directives

- Parallel regions
- Work sharing
- Synchronization
- Data-sharing attributes
  - private
  - firstprivate
  - lastprivate
  - shared
  - reduction
- Orphaning

Environment variables

- Number of threads
- Scheduling type
- Dynamic thread adjustment
- Nested parallelism

Runtime environment

- Number of threads
- Thread ID
- Dynamic thread adjustment
- Nested parallelism
- Timers
- API for locking

编译指令及其子句

OpenMP directives and clauses

- C: directives are case sensitive
  - Syntax: #pragma omp directive [clause [clause] ...]
- Continuation: use \ in pragma
- Conditional compilation: _OPENMP macro is set

句法

if (scalar expression)

- Only execute in parallel if expression evaluates to true
- Otherwise, execute serially

```
#pragma omp parallel if (n > threshold) \
        shared(n,x,y) private(i)
  {
    #pragma omp for
     for (i=0; i<n; i++)
        x[i] += y[i];
  } /*-- End of parallel region --*/
```

if guard

编译指令及子句指定一个并行区

指明之后分析

子句

private (list)

- No storage association with original object
- All references are to the local object
- Values are undefined on entry and exit

shared (list)

- Data is accessible by all threads in the team
- All threads access the same address space

firstprivate (list)

- All variables in the list are initialized with the value the original object had before entering the parallel construct

用原始值初始化

lastprivate (list)

- The thread that executes the sequentially last iteration or section updates the value of the objects in the list

用上一次串行段的结果更新值.

Parallel Region 并行区

```
#pragma omp parallel [clause[[,] clause] ...]
{
    "this is executed in parallel" 代码

} (implied barrier) ← 隐含的全局同步
```

句法

A parallel region is a block of code executed by multiple threads simultaneously, and supports the following clauses:

if	(*scalar expression*)	并行化条件
private	(*list*)	声明私有变量
shared	(*list*)	〃 共享 〃
default	(*none|shared*)	设定数据共享属性省缺值 (C/C++)
default	(*none|shared|private*)	(Fortran)
reduction	(*operator: list*)	缩约操作
copyin	(*list*)	master私有按对组内共线程私有
firstprivate	(*list*)	按原始值初始化
num_threads	(*scalar_int_expr*)	线程数

子句

OpenMP Clause List

- **The clause list is used to specify conditional parallelization, number of threads, and data handling.**
  - Conditional Parallelization: **The clause if (scalar expression) determines whether the parallel construct results in creation of threads.** 条件并行
  - Degree of Concurrency: **The clause num_threads(integer expression) specifies the number of threads that are created.** 并行度
  - Data Handling: **The clause private (variable list) indicates variables local to each thread. The clause firstprivate (variable list) is similar to the private, except values of variables are initialized to corresponding values before the parallel directive. The clause shared (variable list) indicates that variables are shared across all the threads.**

数据处理(共享或私有)

Work-sharing constructs in a Parallel Region

工作共享(分派)机制

```
#pragma omp for
{
    ....
}
```

FOR

```
#pragma omp sections
{
    ....
}
```

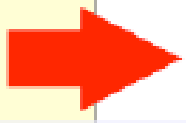
Section

```
#pragma omp single
{
    ...
}
```

多有 master 同 (workshare)

- The work is distributed over the threads
- Must be enclosed in a parallel region 包含在并行块之内
- Must be encountered by all threads in the team, or none at all
- No implied barrier on entry; implied barrier on exit (unless nowait is specified) (master not)
- A work-sharing construct does not launch any new threads (不创建)
- Shorthand syntax supported for parallel region with single work-sharing construct e.g.,

```
#pragma omp parallel
#pragma omp for
    for (...)
```



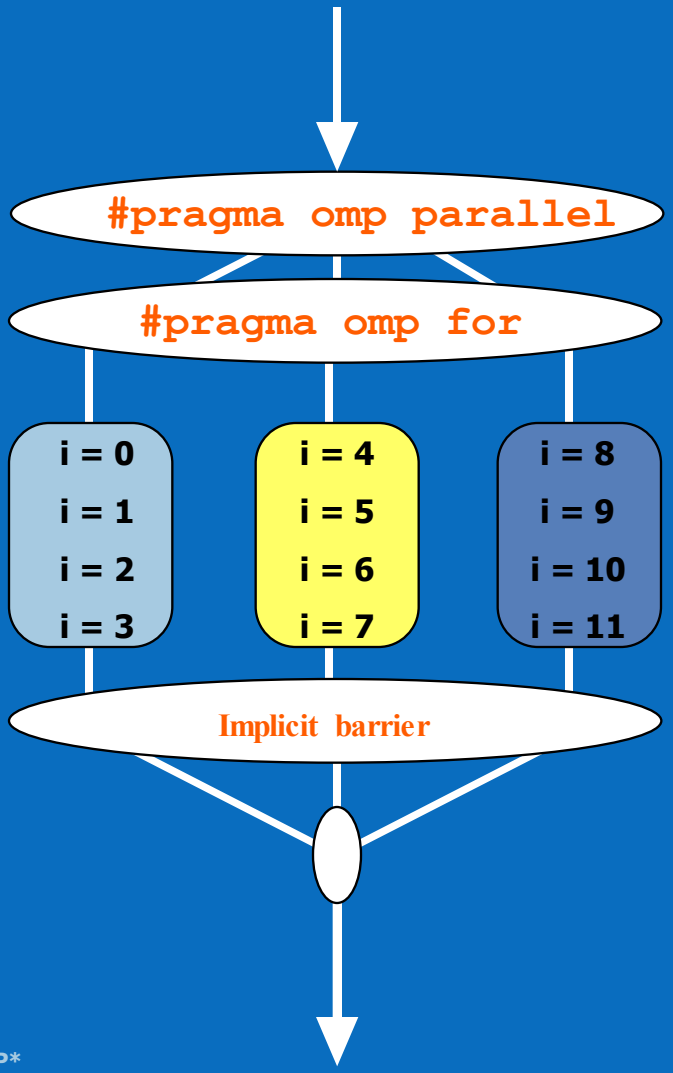
```
#pragma omp parallel for
for (....)
```

Work-sharing Construct

```
#pragma omp parallel
#pragma omp for
   for(i = 0; i < 12; i++)
      c[i] = a[i] + b[i]
```

Threads are assigned an independent set of iterations

Threads must wait at the end of work-sharing construct

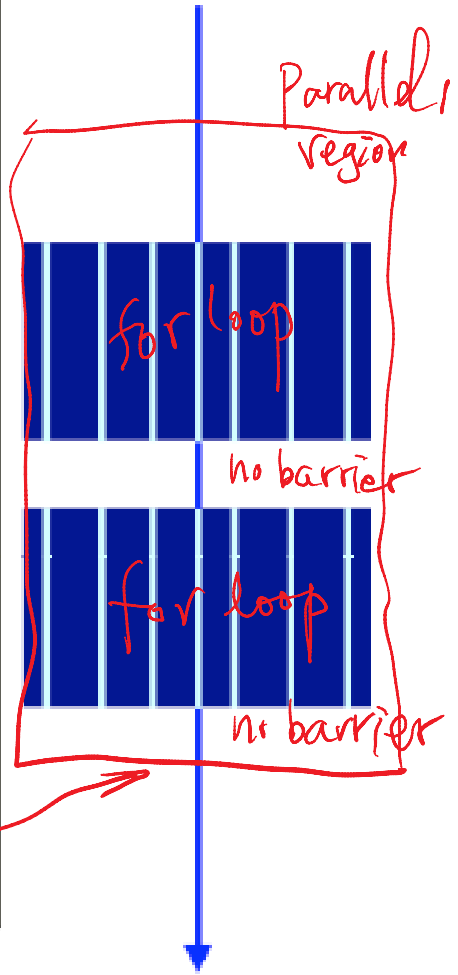


Example of work-sharing "omp for" loop

```
#pragma omp parallel default(none)\
        shared(n,a,b,c,d) private(i)
  {
    #pragma omp for nowait
      for (i=0; i<n-1; i++)
        b[i] = (a[i] + a[i+1])/2;
    #pragma omp for nowait
      for (i=0; i<n; i++)
        d[i] = 1.0/c[i];

  } /*-- End of parallel region --*/
```

(implied barrier)



Reduction Clause in OpenMP

- **The reduction clause specifies how multiple local copies of a variable at different threads are combined into a single copy at the master when threads exit.**
- **The usage of the reduction clause is reduction (operator: variable list).**
- **The variables in the list are implicitly specified as being private to threads.**
- **The operator can be one of +, *, -, &, |, ^, &&, and ||.**

```
#pragma omp parallel reduction(+ : sum) num_threads(8) {
/* compute local sums here */
}
/*sum here contains sum of all local instances of sums */
```

OpenMP Programming: Example

计算π的并行程序

```
/* ******************************************************
An OpenMP version of a threaded program to compute PI.
****************************************************** */
#pragma omp parallel default(private) shared (npoints) \
reduction(+: sum) num_threads(8)
{
      num_threads = omp_get_num_threads();
      sample_points_per_thread = npoints / num_threads;
      sum = 0;
      for (i = 0; i < sample_points_per_thread; i++) {
            rand_no_x =(double)(rand_r(&seed))/(double)((2<<14)-1);
            rand_no_y =(double)(rand_r(&seed))/(double)((2<<14)-1);
            if (((rand_no_x - 0.5) * (rand_no_x - 0.5) +
               (rand_no_y - 0.5) * (rand_no_y - 0.5)) < 0.25)
               sum ++;
      }
}
```

含clause的含义?

Example of work-sharing “sections”

```
#pragma omp parallel default(none)\
        shared(n,a,b,c,d) private(i)
  {
    #pragma omp sections nowait
    {
      #pragma omp section
        for (i=0; i<n-1; i++)
            b[i] = (a[i] + a[i+1])/2;

      #pragma omp section
        for (i=0; i<n; i++)
            d[i] = 1.0/c[i];

    } /*-- End of sections --*/

  } /*-- End of parallel region --*/
```

用 section 进行任务分配

“single” and “master” constructs in a parallel region

Only one thread in the team executes the code enclosed

```
#pragma omp single [clause[[,] clause] ...]
{
        <code-block>
}
```

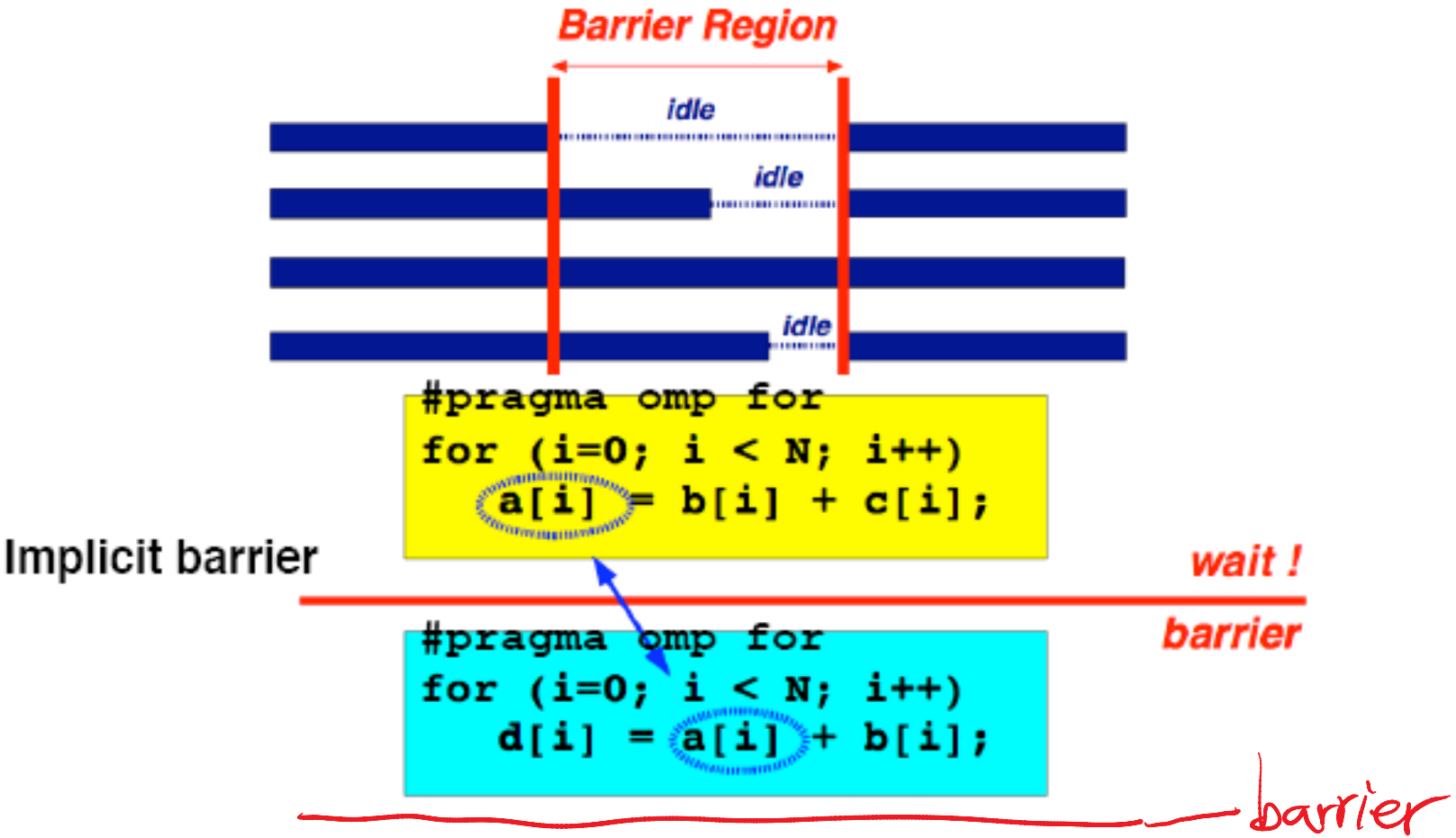
Only the master thread executes the code block.

```
#pragma omp master
{<code-block>}
```

- Single and master are useful for computations that are intended for single-processor execution e.g., I/O and initializations
- There is no implied barrier on entry ~~or exit~~ of a single or master construct

(有barrier)

Implicit barrier



NOTE: barrier is redundant if there is a guarantee that the mapping of iterations onto threads is identical in both loops

nowait clause & explicit barrier

```
#pragma omp for nowait
{
        :
}
```

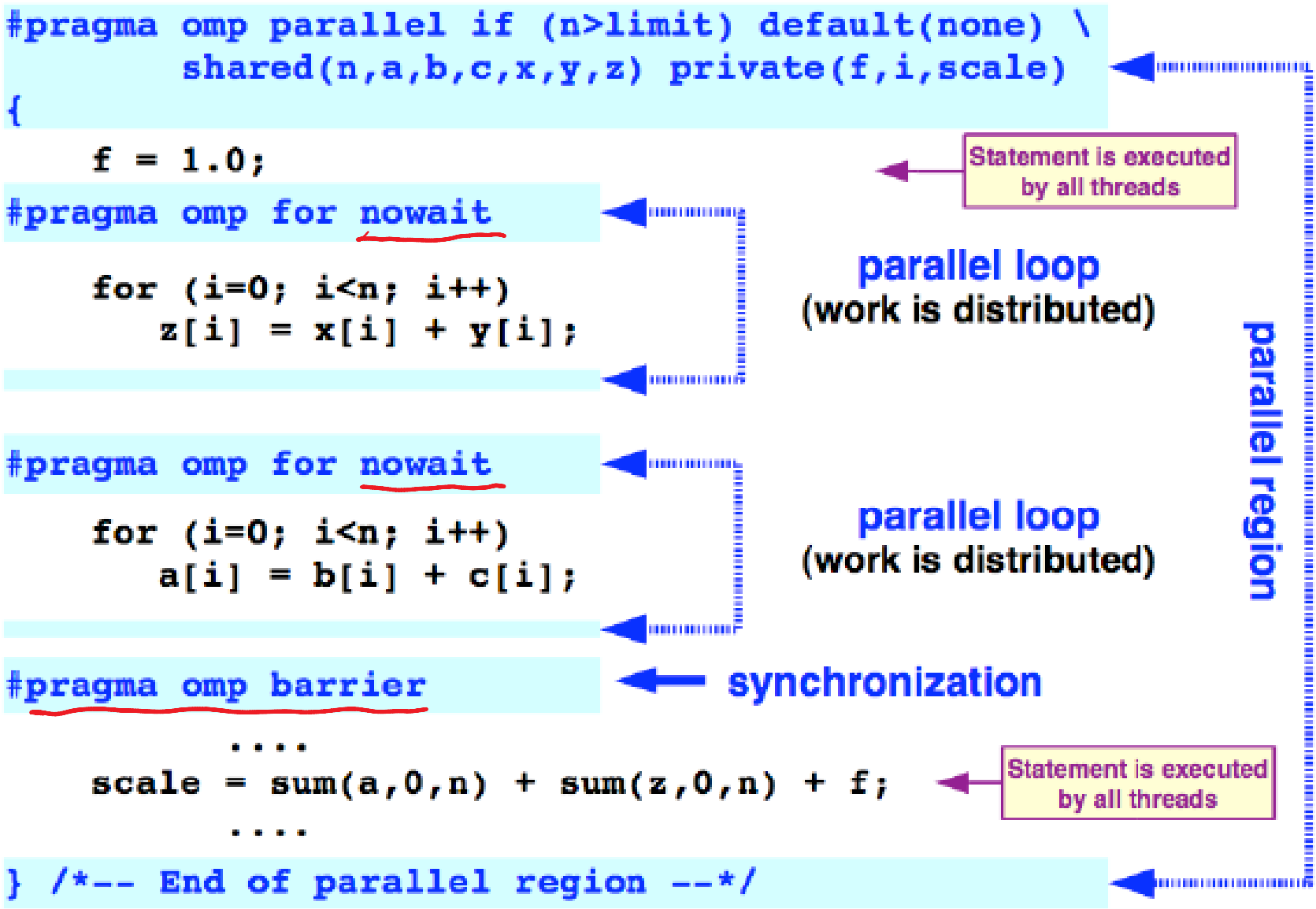
不等待 barrier

```
#pragma omp barrier
```

显式设定 barrier

- To minimize synchronization, some OpenMP directives/pragmas support the optional *nowait* clause
- If present, threads do not synchronize/wait at the end of that particular construct
- An explicit barrier can then be inserted at only the desired program points 在适当的地方可以插入显式的 barrier.

A more elaborate example



Assigning Iterations to Threads

向线程分派工作（调度线程）

- The schedule clause of the **for** directive deals with the *assignment* of iterations to threads.
- The general form of the schedule directive is **schedule(scheduling_class[, parameter])**.
- OpenMP supports four scheduling classes: **static, dynamic, guided**, and **runtime**.

schedule clause for parallel loops

schedule (static | dynamic | guided [, chunk])
schedule (runtime)

static [, chunk]

- ✔ *Distribute iterations in blocks of size "chunk" over the threads in a round-robin fashion*
- ✔ *In absence of "chunk", each thread executes approx. N/P chunks for a loop of length N and P threads*

Example: *Loop of length 16, 4 threads:*

TID	0	1	2	3
no chunk	1-4	5-8	9-12	13-16
chunk = 2	1-2 9-10	3-4 11-12	5-6 13-14	7-8 15-16

schedule clause for parallel loops (contd)

dynamic [, chunk]

- ✔ *Fixed portions of work; size is controlled by the value of chunk* 工作块大小数目固定，线程动态调度，每当一线程结束，便开始下一块工作
- ✔ *When a thread finishes, it starts on the next portion of work*

guided [, chunk] 同dynamic，但块大小按指数级递减

- ✔ *Same dynamic behavior as "dynamic", but size of the portion of work decreases exponentially* 指数级递减分配

runtime

- ✔ *Iteration scheduling scheme is set at runtime through environment variable OMP_SCHEDULE* 由环境变量决定

Assigning Iterations to Threads: Example

```
/* static scheduling of matrix multiplication loops */
#pragma omp parallel default(private) shared (a, b, c, dim) \
num_threads(4)
#pragma omp for schedule(static)
for (i = 0; i < dim; i++) {
for (j = 0; j < dim; j++) {
    c(i,j) = 0;
    for (k = 0; k < dim; k++) {
    c(i,j) += a(i, k) * b(k, j);
    }
}
}
```

怎样分配?

Nesting parallel Directives

- Nested parallelism can be enabled using the **OMP_NESTED** environment variable.
- If the **OMP_NESTED** environment variable is set to **TRUE**, nested parallelism is enabled.
- In this case, each parallel directive creates a new team of threads. 每一个都会特别建新线程.

不在并行块之内的分配指令将被忽略.

Out-of-line ("orphaned") directives

- *The OpenMP standard does not restrict worksharing and synchronization directives (omp for, omp single, critical, barrier, etc.) to be within the lexical extent of a parallel region. These directives can be orphaned*
- *That is, they can appear outside the lexical extent of a parallel region*

```
   (void) dowork(); !- Sequential FOR

#pragma omp parallel
{
   (void) dowork(); !- Parallel FOR
}
```

```
void dowork()
{
#pragma omp for
   for (i=0;....)
   {
      :
   }
}
```

- *When an orphaned worksharing or synchronization directive is encountered in the sequential part of the program (outside the dynamic extent of any parallel region), it is executed by the master thread only. In effect, the directive will be ignored*

Homework 6

- Read the section of *OpenMP C and C++ Application program Interface* in MSDN, and try to run some of the examples in this section.

或者在任何地方获得一个OpenMP程序，请在Linux或Windows下测试运行。

（运行环境请自己按装配置）