并行与分布式计算基础：第十一讲

杨超
chao_yang@pku.edu.cn

2019 秋

课程基本情况

- 课程名称：并行与分布式计算基础
- 授课教师：杨超（chao_yang@pku.edu.cn，理科 1 号楼 1520）
- 课程助教：尹鹏飞（pengfeiyin@pku.edu.cn）

授课内容（暂定）

- 引言
- 硬件体系架构
- 并行计算模型
- 编程与开发环境
- MPI 编程与实践
- OpenMP 编程与实践
- GPU 编程与实践
- 前沿问题选讲

上课时间（地点：二教 211）

上课时间	星期一	星期二	星期三	星期四	星期五
第 1 节 (8:00-8:50)					
第 2 节 (9:00-9:50)					
第 3 节 (10:10-11:00)				单周	
第 4 节 (11:10-12:00)				单周	
第 5 节 (13:00-13:50)		每周			
第 6 节 (14:00-14:50)		每周			
第 7 节 (15:10-16:00)					
第 8 节 (16:10-17:00)					
第 9 节 (17:10-18:00)					
第 10 节 (18:40-19:30)					
第 11 节 (19:40-20:30)					
第 12 节 (20:40-21:30)					

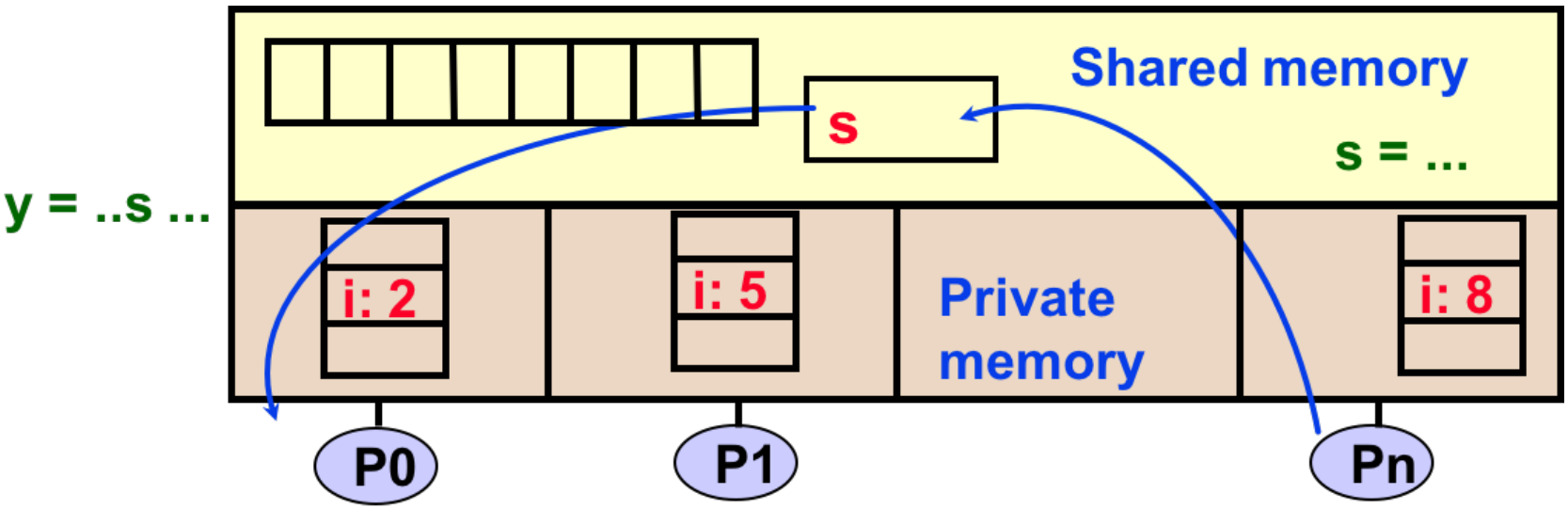
内容提纲

1. OpenMP 编程-1：入门篇 (回顾)
   - OpenMP 入门知识
   - 制导语句与并行区构造
2. OpenMP 编程-2：基础篇
   - 循环工作共享构造
   - 数据域从句
   - 线程调度从句

OpenMP 入门知识

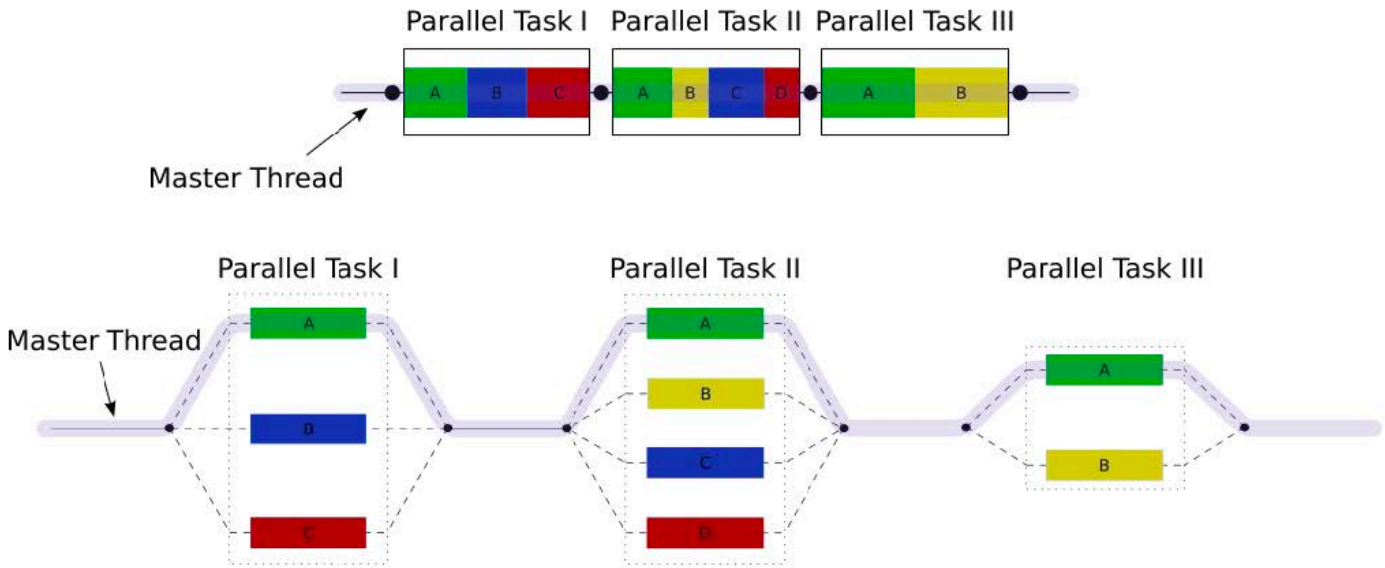
共享内存并行编程与 OpenMP

- 共享内存并行编程
  - 程序由一系列线程 (thread) 控制；
  - 每个线程有自己私有的变量，线程之间通过共享变量进行交互。
- OpenMP = Open Multi-Processing
  - 是一种支持共享内存并行的应用开发接口 (API) 和规范 (specification)；
  - 支持多种编程语言、指令集架构和操作系统。



Fork-Join 模式

- OpenMP 主要采用 Fork-Join 模式进行并行执行；
- 程序开始时只有一个线程：主线程 (master thread)，编号为 0；
- 主线程仅当进入并行区 (parallel region) 才并行执行：
  - Fork：主线程创建一组并行线程，编号 0 ～ $n-1$；
  - 执行：并行线程在并行区中并行执行；
  - Join：在并行区结尾并行线程进行同步和结束，只保留主线程；
- 并行区的个数，以及每个并行区中的线程数，都可以任意设置。



OpenMP API 的三个要素

- 运行时库 (run-time library)：头文件 (omp.h)、库函数的调用和链接。

Fortran	`INTEGER FUNCTION OMP_GET_NUM_THREADS()`
C/C++	`#include <omp.h>` `int omp_get_num_threads(void)`

- 环境变量 (environment variable)：由 OpenMP API 预定义，用于控制程序的行为。

csh/tcsh	`setenv OMP_NUM_THREADS 8`
sh/bash	`export OMP_NUM_THREADS=8`

- 编译制导语句 (compiler directive)：程序中一些特殊格式的注释，如编译器支持，则可实现 OpenMP 的一些功能。

Fortran	`!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(BETA,PI)`
C/C++	`#pragma omp parallel default(shared) private(beta,pi)`

OpenMP 的 19 个常用核心 (common cores)

OpenMP pragma, function, or clause	Concepts
#pragma omp parallel	Parallel region, teams of threads, structured block, interleaved execution across threads
int omp_get_thread_num() int omp_get_num_threads()	Create threads with a parallel region and split up the work using the number of threads and thread ID
double omp_get_wtime()	Speedup and Amdahl's law. False Sharing and other performance issues
setenv OMP_NUM_THREADS N	Internal control variables. Setting the default number of threads with an environment variable
#pragma omp barrier #pragma omp critical	Synchronization and race conditions. Revisit interleaved execution.
#pragma omp for #pragma omp parallel for	Worksharing, parallel loops, loop carried dependencies
reduction(op:list)	Reductions of values across a team of threads
schedule(dynamic [,chunk]) schedule (static [,chunk])	Loop schedules, loop overheads and load balance
private(list), firstprivate(list), shared(list)	Data environment
nowait	Disabling implied barriers on workshare constructs, the high cost of barriers. The flush concept (but not the concept)
#pragma omp single	Workshare with a single thread
#pragma omp task #pragma omp taskwait	Tasks including the data environment for tasks.

一些重要用法

- 返回当前线程的墙钟时间和刻度：

```
double omp_get_wtime(void)
double omp_get_wtick(void)
```

- 通过环境变量设置和获取线程数：

```
$ export OMP_NUM_THREADS=4
$ echo $OMP_NUM_THREADS
```

- 通过库函数设置和获取线程数：

```
void omp_set_num_threads(int)
int omp_get_num_threads(void)
int omp_get_max_threads(void)
```

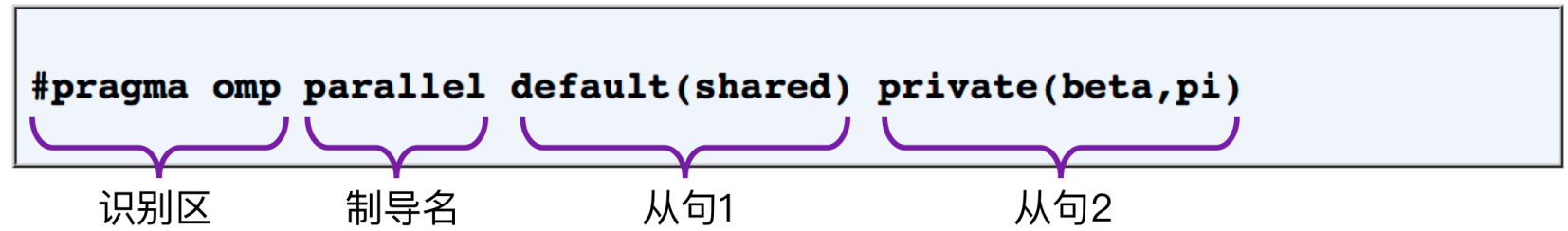
- 获取线程号：

```
int omp_get_thread_num(void)
```

制导语句与并行区构造

OpenMP 制导语句的构造和用法

- 同一类 OpenMP 制导语句称为一种构造 (construct)：
  - 比如并行区构造、工作共享构造、任务构造、同步构造等。
- OpenMP 制导语句的用法为：
  - 以 `#pragma omp` 开始；
  - 接着是某一个制导名 (directive-name)，比如 `parallel`；
  - 接着是零至多个从句 (clause)，从句出现的顺序不重要。



  - 注意：如果一行过长，换行时行末需要加 “\”。

最基本的 OpenMP 构造：并行区构造

- 用途：划定并行区的范围，并做相关设置，格式：

```
#pragma omp parallel [clause1 clause2 ...]
{
  ...
}
```

- 支持的从句：

```
if (scalar_expression)
num_threads (integer-expression)
private (list)
shared (list)
default (shared | none)
firstprivate (list)
reduction (operator: list)
copyin (list)
```

控制从句和数据域从句

- 控制从句：决定是否以并行的方式执行，指定并行区的线程数。

```
if (scalar_expression)
num_threads (integer-expression)
```

- 数据域从句：指定变量的数据域（共享、私有等）。

```
private (list)
shared (list)
default (shared | none)
firstprivate (list)
reduction (operator: list)
copyin (list)
```

线程数的确定

按照优先级从低到高，并行区中的线程数按照下面的顺序确定：

- 系统默认 (一般是可用的处理器核数)；
- `OMP_NUM_THREADS` 环境变量设定；
- `omp_set_num_threads` 库函数设定；
- `num_threads` 从句设定；
- `if` 从句。

循环工作共享构造

回忆计算 π 的示例程序

omp_cpi.c

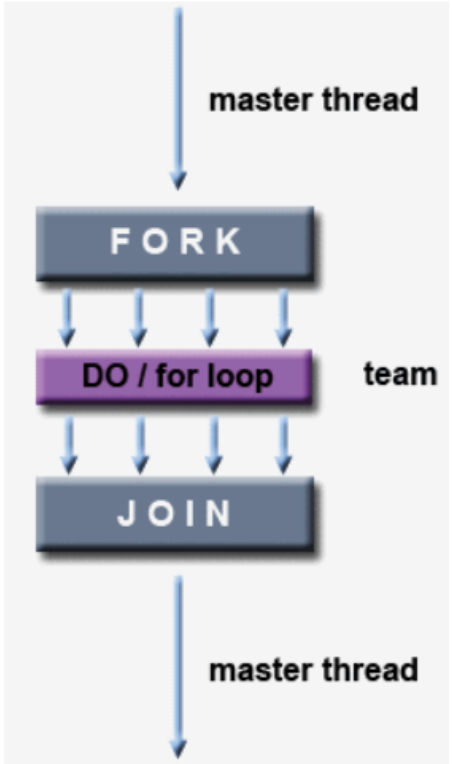
```
1   ...
2   pi = 0.0;
3 #pragma omp parallel default(shared) \
4   private(tid, i, x) reduction(+:pi)
5   {
6     nthreads = omp_get_num_threads();
7     tid = omp_get_thread_num();
8     for (i = tid + 1; i <= n; i += nthreads) {
9       x = h * ((double)i - 0.5);
10      pi += 4.0 * h * sqrt(1.-x*x);
11    }
12  }
13  ...
```

- 思考:
  - 这种基于循环的“数据”并行十分常见!
  - 是否可以采用更简单的方式实现?

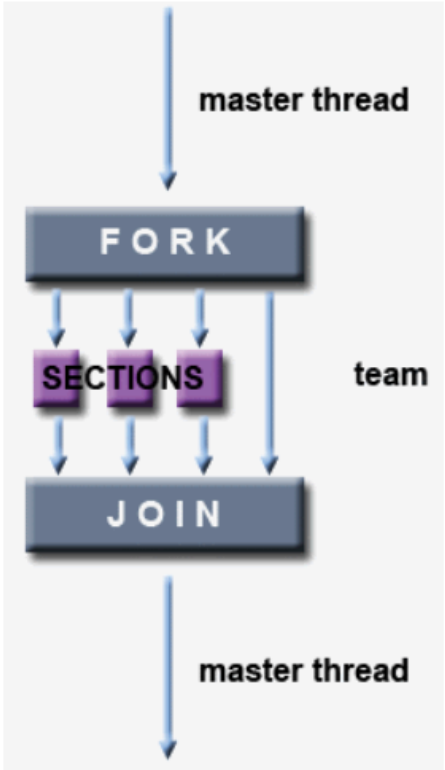
工作共享构造 (work-sharing construct)

用于将代码分配采用某种机制给不同的线程执行：循环、分块、单独
(注：在入口没有同步，但是在出口包含了一个隐含的栅栏同步)。

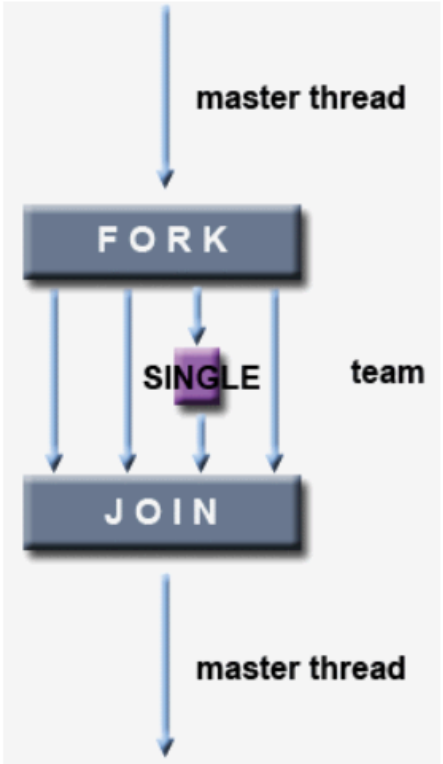
DO / for - shares iterations of a loop across the team. Represents a type of "data parallelism".



SECTIONS - breaks work into separate, discrete sections. Each section is executed by a thread. Can be used to implement a type of "functional parallelism".



SINGLE - serializes a section of code



for 循环构造

- 用于对循环进行多线程并行执行 (前提：已经在并行区内)：

```
#pragma omp for [clause1 clause2 ...]
  for (...) {
    ...
  }
```

- 支持的从句：

```
schedule (type [,chunk])
ordered
private (list)
firstprivate (list)
lastprivate (list)
shared (list)
reduction (operator: list)
collapse (n)
nowait
```

for 循环的格式

- OpenMP 的 for 循环构造对 for 循环的格式有严格要求：
  - 开始语句：必须是“变量 = 初值”形式；
  - 终止语句：必须明确变量与边界值的大小关系；
  - 计数语句：必须采用规范的等步长累加或者累减；
  - 不能使用 break、goto、return 等；
  - 循环变量必须是整数，初值、边界和增量在循环中固定。
- 思考：下面哪些用法有问题?

```
1:  for ( ; i<=10; ++i)
```

```
2:  for (k=1000; k>8; k--)
```

```
3:  for (i=1; i<=j; j=j-10)
```

```
4:  for (z=1; 100>z; z=2*z)
```

for 构造在并行区中的用法

- 如下并行区内的 code1()-code6() 分别怎么执行?

```
#pragma omp parallel
{
  code1();
#pragma omp for
  for (i=1; i<=N; i++) {
    code2();
  }
  for (j=1; j<=M; j++) {
    code3();
  }
  code4();
#pragma omp for
  for (m=L; m>=1; --m) {
    code5();
  }
  code6();
}
```

parallel for 构造

- 如果并行区中只有一个 for 构造，则可以使用：

```
#pragma omp parallel for
  for (i=1; i<=N; i++) {
    code2();
  }
```

- 练习：
  - 把 omp_cpi.c 中的 n 改为 11111111；
  - 使用 for 构造修改 omp_cpi.c；
  - 使用 parallel for 构造修改 omp_cpi.c；
  - 将新程序保存为 omp_cpi2.c.

支持的从句概览

	parallel	for	parallel for
if	●		●
num_threads	●		●
default	●		●
copyin	●		●
shared	●	●	●
private	●	●	●
reduction	●	●	●
firstprivate	●	●	●
lastprivate		●	●
schedule		●	●
ordered		●	●
collapse		●	●
nowait		●	

数据域从句

数据域从句：默认变量、共享变量和规约变量

```
default (shared | none)
shared (list)
reduction (operator: list)
```

- default 从句：指定默认变量类型；
  - shared：默认为共享变量；
  - none：无默认变量类型，每个变量都需要另外指定。
- shared 从句：指定共享变量列表；
  - 共享变量在内存中只有一份，所有线程都可以访问；
  - 编程中要确保多个线程访问同一个公有变量时不会有冲突。
- reduction 从句：指定规约变量列表；
  - 与 private 从句定义的私有变量类似，不同点是
  - 各个线程对该变量额外进行 operator 定义的规约操作。

规约操作的类型和初始值

Valid Operators and Initialization Values			
Operation	**Fortran**	**C/C++**	**Initialization**
Addition	`+`	`+`	`0`
Multiplication	`*`	`*`	`1`
Subtraction	`-`	`-`	`0`
Logical AND	`.and.`	`&&`	`.true. / 1`
Logical OR	`.or.`	`||`	`.false. / 0`
AND bitwise	`iand`	`&`	`all bits on / 1`
OR bitwise	`ior`	`|`	`0`
Exclusive OR bitwise	`ieor`	`^`	`0`
Equivalent	`.eqv.`		`.true.`
Not Equivalent	`.neqv.`		`.false.`
Maximum	`max`	`max`	`Most negative #`
Minimum	`min`	`min`	`Largest positive #`

数据域从句：三种私有变量

```
private (list)
firstprivate (list)
lastprivate (list)
```

- private 从句：
  - 每个线程生成一份与该私有变量同类型的数据对象；
  - 声明为私有变量的数据在并行区中都需要重新进行初始化。
- firstprivate 从句：
  - 与 private 从句定义的私有变量类似，不同点是
  - 在并行区执行伊始，对该变量根据主线程中的数据进行初始化。
- lastprivate 从句：
  - 与 private 从句定义的私有变量类似，不同点是
  - 在并行区执行结束，将执行最后一个循环的线程的私有数据取出。

示例：私有变量

omp_private.c

```
1  #include <omp.h>
2  #include <stdio.h>
3
4  int main(int argc, char *argv[]){
5    int  i, k;
6
7    printf("Test:\n");
8    k = 100;
9    printf(" first k = %d\n", k);
10 #pragma omp parallel for private(k)
11   for (i = 0; i < 10; i++) {
12     k = i;
13     printf("  private k = %d\n", k);
14   }
15   printf(" last k = %d\n", k);
16   return 0;
17 }
```

思考和练习

- 循环体中修改 `k` 的值，会影响主线程的结果吗?
- `k = i` 可以改为 `k += i` 吗?
- 将 `private(k)` 改为别的私有变量类型，会怎样?

线程调度从句

线程调度：schedule 从句

- schedule 从句：主要用于控制调度方式。

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

  - type：调度类型，包括：
    - ★ static：静态调度，chunk 大小固定 (默认：n/t)；
    - ★ dynamic：动态调度，chunk 大小固定 (默认：1)；
    - ★ guided：动态调度，chunk 大小动态缩减；
    - ★ runtime：由环境变量 OMP_SCHEDULE 确定 (上述三种之一)；
    - ★ auto：系统自选。
  - chunk：分块大小，必须是正整数。

static 静态调度

- 默认 chunk=n/t，按循环起止均匀分配；
- 调整 chunk 可以改变静态线程分配的策略；
- chunk=1，相当于 round-robin。

```
for (i=0; i<=11; i++)
```

schedule(static,4)

Thread 0 :	0,1,2,3
Thread 1 :	4,5,6,7
Thread 2 :	8,9,10,11

schedule(static,2)

Thread 0 :	0,1,6,7
Thread 1 :	2,3,8,9
Thread 2 :	4,5,10,11

schedule(static,1)

Thread 0 :	0,3,6,9
Thread 1 :	1,4,7,10
Thread 2 :	2,5,8,11

示例：线程调度

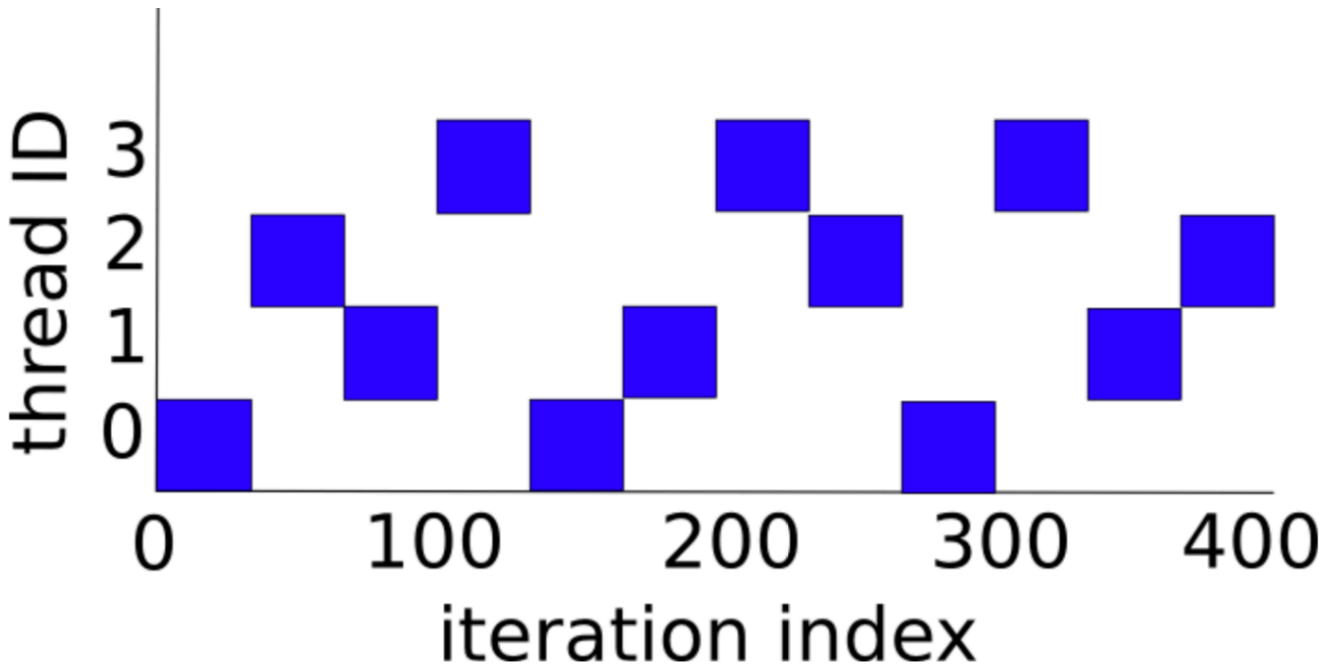
omp_schedule.c

```
1   ...
2  #define n 12
3   ...
4  #pragma omp parallel num_threads(4)
5    {
6  #pragma omp for schedule(static)
7      for (i = 0; i < n; i++) {
8        ...
9      }
10   }
11  ...
```

- 练习：
  - 尝试给 `static` 加上不同的 `chunk` 值。

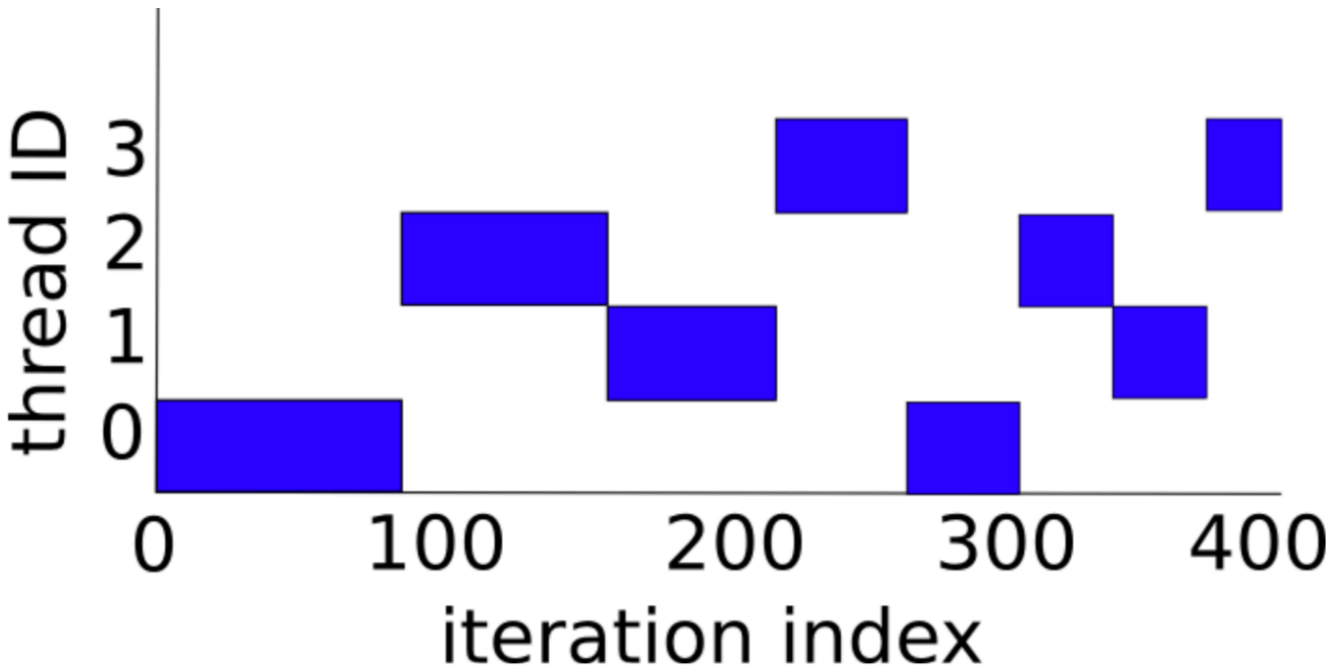
dynamic 动态调度

- 根据线程空闲情况，对工作进行动态分配：
  - 默认 chunk=1，动态分配的任务粒度为 1；
  - 增大 chunk 可以增大任务的粒度；
  - 调度开销不容忽视。



guided 动态调度

- 为了减少调度开销，动态分配任务的粒度逐步减小：
  - 调整策略：粒度 = 剩余迭代次数/线程数；
  - 最小粒度为 chunk，默认为 1。



练习

- 修改 omp_schedule.c 中的 schedule 从句：
  - 测试 dynamic 和不同的 chunk 值；
  - 测试 guided 和不同的 chunk 值；
  - 测试 runtime 并由环境变量 OMP_SCHEDULE 确定；
  - 测试 auto，由系统自行确定。

计算 π：采用并行区构造

omp_cpi.c

```
1   ...
2   t0 = omp_get_wtime();
3   n = 11111111;
4   h = 1.0 / (double) n;
5   pi = 0.0;
6  #pragma omp parallel default(shared) \
7   private(tid, i, x) reduction(+:pi)
8   {
9     nthreads = omp_get_num_threads();
10    tid = omp_get_thread_num();
11    for (i = tid + 1; i <= n; i += nthreads) {
12      x = h * ((double)i - 0.5);
13      pi += 4.0 * h * sqrt(1.-x*x);
14    }
15  }
16  t1 = omp_get_wtime();
17  ...
```

计算 π：采用循环构造

omp_cpi2.c

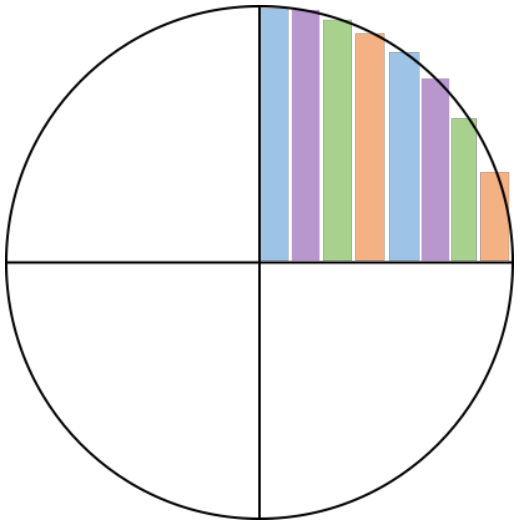
```
1   ...
2   t0 = omp_get_wtime();
3   n = 11111111;
4   h = 1.0 / (double) n;
5   pi = 0.0;
6 #pragma omp parallel for default(shared) \
7   private(x) reduction(+:pi)
8   for (i = 1; i <= n; i ++) {
9     x = h * ((double)i - 0.5);
10    pi += 4.0 * h * sqrt(1.-x*x);
11  }
12  t1 = omp_get_wtime();
13  ...
```

练习

- 尝试采用不同的调度方式，比较性能：
  - `static` 静态调度；
  - `dynamic` 动态调度；
  - `guided` 动态调度；
  - `runtime` 运行时设定 (`OMP_SCHEDULE`)。

作业 2：算 π

- 用 OpenMP 编写程序计算 π，必须利用如下数值积分公式：



$$\pi = 4 \int_0^1 \sqrt{1 - x^2} dx.$$

- 要求：数值积分所用的离散点个数 $n \leq 111111$ (六个 1)；
- 可自选并行策略，可自选 OpenMP 中任意并行方式；
- 评分依据：2 至 16 线程正确，8 线程精度高 (时间不超过 10 秒)；
- 截止时间：2019 年 11 月 12 日 24 点前；
- 提交方式：发送至助教邮箱 (助教会在群里通知其他要求)。