OpenMP程序设计

目录

01 OpenMP编译指导语句

02 编译指导指令

03 OpenMP子句

04 任务的调度

05 运行库函数与环境变量

目录

01 OpenMP编译指导语句

02 编译指导指令

03 OpenMP子句

04 任务的调度

05 运行库函数与环境变量

OpenMP编译指导语句

OpenMP 通过对串行程序添加编译指导指令实现并行化

- **并行域指令：**
  生成并行域，即产生多个线程以并行执行任务，所有并行任务必须放在并行域中才可能被并行执行
- **工作共享指令：**
  负责任务划分，并分发给各个线程，工作共享指令不能产生新线程，因此必须位于并行域中
- **同步指令：**
  负责并行线程之间的同步
- **数据环境：**
  负责并行域内的变量的属性（共享或私有），以及边界上（串行域与并行域）的数据传递

OpenMP编译指导语句

C语言程序示例

```
#include <omp.h>
#include <stdio.h>
int main()
{
  int nthreads, tid;
  #pragma omp parallel private(nthreads,tid)
  {
    tid=omp_get_thread_num(); // 获取线程号
    printf("Hello world from OpenMP thread %d\n", tid);
    if (tid==0)
    {
      nthreads=omp_get_num_threads(); // 获取线程个数
      printf("Number of threads %d\n", nthreads);
    }
  }
  return 0;
}
```

头文件：omp.h

编译指导：
#pragma omp

编译命令：
gcc –fopenmp –o hello hello.c
icc –openmp hellp.c   // Intel C Compiler

说明：

- C/C++ 的 OpenMP 指令标识符为 #pragma omp
- C/C++ 程序中，OpenMP 指令区分大小写
- 每个 OpenMP 指令后是一个结构块（用大括号括起来）

■**编译指导语句的格式：**

#pragma omp <directive> [claus[[,]clause]…]

- directive部分是编译指导语句的主要指令
- clause部分是可选子句，给出相应的指令参数，影响编译指导语句的具体执行

OpenMP编译指导语句

■ **基于C/C++语言的OpenMP程序结构**

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

main()
{
      int var1, var2, var3;
      ........
       #pragma omp parallel private(var1, var2) shared(var 3)
       {
             .............
       }
      ...............
 }
```

目录

01 OpenMP编译指导语句

02 编译指导指令

03 OpenMP子句

04 任务的调度

05 运行库函数与环境变量

编译指导指令 – 并行域

- 编译指导语句以#pragma omp开始，其后具体紧跟的常用指令有：

并行域指令 Parallel Constructs

parallel	创建一个并行域

```
#pragma omp parallel private(tid)
{
		tid=omp_get_thread_num(); // Obtain thread id
		printf("Hello world from OpenMP thread %d\n", tid);
		if (tid==0) // Only master thread does this
		{
				nthreads=omp_get_num_threads();
				printf("Number of threads: %d\n", nthreads);
		}
}
```

编译指导指令 – 并行域

Fortran	!$omp parallel [clause clause …] structured-block !$omp end parallel

C/C++	#pragma omp parallel [clause clause …]
	{
	    structured-block
	}

子句用来添加一些补充信息。 若有多个，则用空格隔开

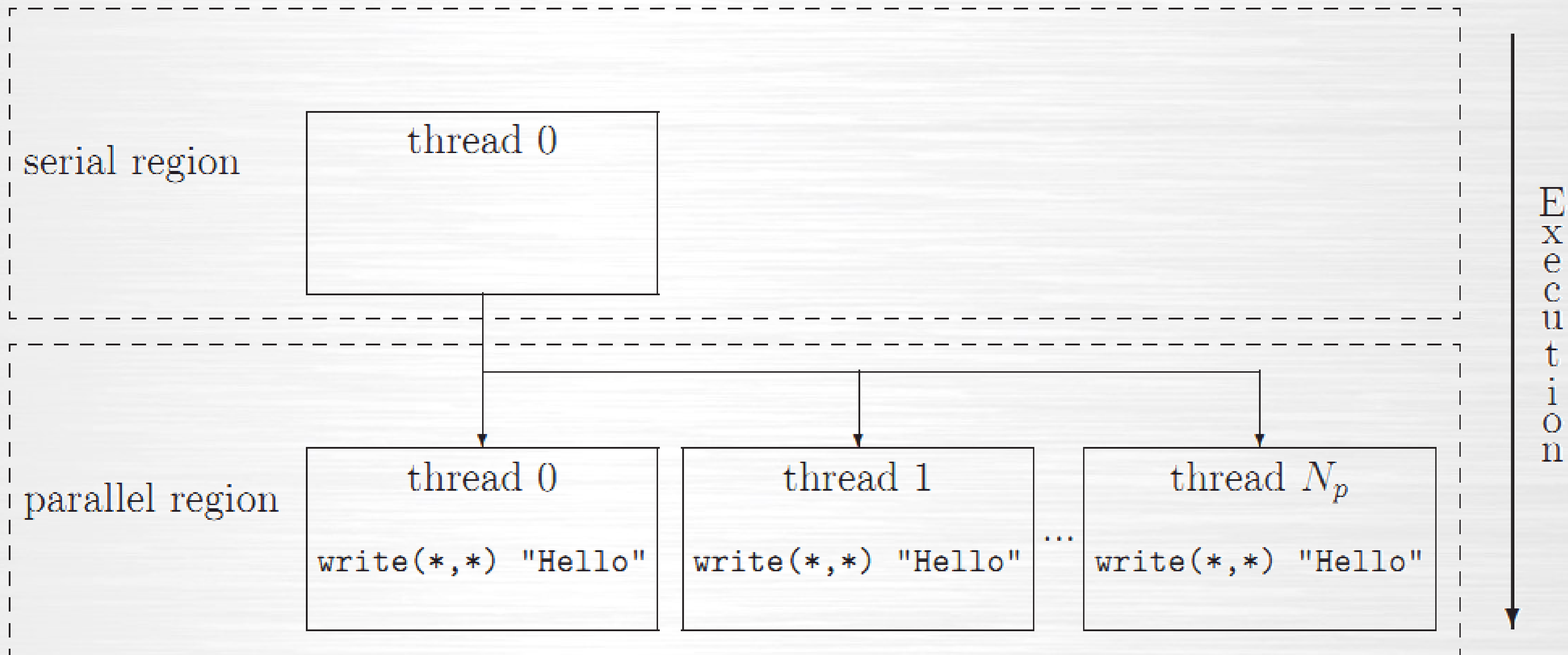
若没有指定线程个数，则产生最大可能的线程个数

例：指定线程个数，设置变量属性

```
#pragma omp parallel private(tid) num_threads(3)
{
       . . . . . . .
 }
```

编译指导指令 – 并行域

■ 并行域的执行过程



编译指导指令 – 并行域

- **omp_set_nested(1)：打开并行域嵌套功能**

```
omp_set_nested(1); // 打开并行域嵌套功能
#pragma omp parallel private(tid) num_threads(2)
{
        tid=omp_get_thread_num();
        printf("Hello world from OpenMP thread %d\n", tid);
        #pragma omp parallel private(tid) num_threads(3)
        {
                tid=omp_get_thread_num();
                printf("Hello math from OpenMP thread %d\n", tid);
        }
}
```

缺省不支持嵌套，需要利用 OpenMP 的 API 过程 omp_set_nested 开启嵌套功能（该过程的缺省值是 false）

编译指导指令 – 并行域

并行域C程序示例1

```c
// test1.c
#include <omp.h>
#include <stdio.h>
int main()
{
  printf("-----------\n");
  #pragma omp parallel
  printf("Hello\n");

  printf("=============\n");
  return 0;
}
```

```
gcc -fopenmp -o test1 test1.c
export OMP_NUM_THREADS=4
./ test1
```

```
------------
Hello
Hello
Hello
Hello
=============
```

编译时须加选项-fopenmp

icc选项为-openmp

环境变量OMP_NUM_THREADS指定使用的线程数目

紧跟指令的第一个结构块被并行，其它串行

编译指导指令 – 并行域

并行域C程序示例2

```
// test1-1.c
#include <omp.h>
#include <stdio.h>
int main()
{
  printf("-----------\n");
  #pragma omp parallel
{
    printf("Hello1\n");
    printf("Hello2\n");
 }
  printf("=============\n");
  return 0;
}
```

```
gcc -fopenmp -o test1-1 test1-1.c
export OMP_NUM_THREADS=4
./ test1-1
```

```
-----------
Hello1
Hello1
Hello2
Hello2
Hello1
Hello2
Hello1
Hello2
=============
```

用{ }将多条语句封装成一个结构块

编译指导指令 – 并行域

■ 并行域的关闭

- 并行区域在结构块后结束，各线程的本地变量（或称为私用变量）被销毁，主线程之外的所有线程都被杀死

- 关闭并行区域前，主线程等待其它线程到达，实际上，这个“等待”就是一次“隐式同步”

- 并行区域内代码要求
  - 一个完整的结构化代码块，不能使用GOTO语句转入或跳出并行区域
  - 没有更多语法限制。实用代码中，不但要保证语法正确，还要保证结果正确

编译指导指令 – 工作共享

- 编译指导语句以#pragma omp开始，其后具体紧跟的常用指令有:

工作共享结构 Work-Sharing Constructs

for	创建循环共享结构，代表典型的数据并行
sections /section	创建 sections 结构，将任务划分成独立的子任务(section)， 每个子任务由一个线程执行，典型的任务并行
single	创建仅由一个线程执行的任务，先到先执行，其他线程等 待其执行结束后再一起执行后面的任务
master	与 single 类似，但指定由主线程执行，而且其他线程无需等待
task taskyield	创建一个显式任务，可以立即被执行，也可以挂起并推迟 执行，便于实现一些复杂结构，如递归。

- **负责任务的划分和分配**
- **在每个工作分享结构入口处无需同步**
- **每个工作分享结构结束处会隐含障碍同步**

编译指导指令 – 工作共享

循环共享

<table>
<tr><td>Fortran</td><td>!$omp parallel [clause clause ...]
    do-loops
!$omp end parallel</td></tr>
<tr><td>C/C++</td><td>#pragma omp parallel [clause clause ...]
{
    for-loops
}</td></tr>
</table>

- 只负责工作分享，不负责并行域的产生和管理，一般需放在并行域中
- 如果不放在并行域内，则只能串行执行

编译指导指令 – 工作共享

- 如果并行域中只有循环共享结构，则可以合写在一起

Fortran	`!$omp parallel do [clause clause …]`
	`    do-loops`
	`!$omp end parallel do`

C/C++	`#pragma omp parallel for [clause clause …]`
	`{`
	`    for-loops`
	`}`

编译指导指令 – 同步结构

■ 编译指导语句以#pragma omp开始，其后具体紧跟的常用指令有：

同步结构 Synchronization Constructs

critical (临界区)	避免线程竞争，其包含的代码同一时刻只能有一个线程执行
barrier (路障)	障碍同步：用在并行域内，所有线程执行到 barrier 都要停下等待，直到所有线程都执行到 barrier，然后再继续往下执行
atomic	确保一个特殊存储单元只能原子更新，即不允许多线程同时去写，只能用于单一赋值语句等特殊情况
flush	确保线程存储的临时视图与共享存储中的数据一致，并且保证一个变量在共享存储中的读/写顺序
ordered	指定并行域的循环按迭代代顺序执行
taskwait	可配合 task 结构使用，创建任务调度点

编译指导指令 – 同步结构

- OpenMP线程是并发执行的，不同线程的指令不能按照固定的顺序来执行

➢ **critical 临界区**

Fortran	!$omp critical code !$omp end critical

C/C++	#pragma omp critical
	{
	code
	}

规定如果一个线程正在执行一个代码块，而第二个线程试图执行同样的代码块，那么第二个线程将暂停并等待，直到第一个线程执行完成

➢ **barrier 路障**

Fortran	!$omp barrier code !$omp end barrier

C/C++	#pragma omp barrier
	{
	code
	}

定义了程序中的某个点，在这个点上，所有线程必须在任意线程通过栅栏之前到达

编译指导指令 – 数据环境

- 编译指导语句以#pragma omp开始，其后具体紧跟的常用指令有：

数据环境指令 Data Environment Constructs

threadprivate(list)	将一个或多个私有变量声明为全局的，即在多个并行域中使用时，保留私有变量在上次并行域中的值；

目录

01 OpenMP编译指导语句
02 编译指导指令
03 OpenMP子句
04 任务的调度
05 运行库函数与环境变量

OpenMP子句

■ 子句（Clause）： 出现在编译制导指令之后，负责添加一些补充设置

数据作用域属性子句 Data Sharing Attribute Clauses

private(list)	创建一个或多个变量的私有拷贝，即在每个线程中都创建一个同名局部变量，但没有初始值； 列表中的变量必须已定义，且不能是常量和引用； 列表中的多个变量用逗号隔开。
firstprivate(list)	private 的扩展，创建私有拷贝的同时，将主线程中的同名变量的值作为初值。
lastprivate(list)	退出并行域时，将指定的私有拷贝的“最后”值复制到主线程中的同名变量中； “最后”：循环最后一次迭代（按串行方式），或 sections 最后一个 section（代码中）； 可能会增加额外开销，一般不建议使用，可以用共享变量等方式实现。
shared(list)	指定一个或多个变量为共享变量，即所有线程都可以访问这些变量
default(...)	指定并行域内的变量的缺省属性，C 语言支持 shared 和 none
copyin(list)	配合 threadprivate，用主线程同名变量的值对 threadprivate 的私有拷贝进行初始化
copyprivate(list)	配合 single，将 single 块中串行计算得到的变量值广播到并行域中其它线程的同名变量中
reduction(op:list)	创建一个或多个变量的私有拷贝，在并行结束后对这些变量执行指定的归约操作（如求和），并将结果返回给主线程中的同名变量

OpenMP子句

如何决定哪些变量是共享哪些是私有?

- 通常循环变量、临时变量、写变量一般应设置成私有的;
- 数组变量、仅用于读的变量通常是共享的;
- 能设置成共享的变量建议设置成共享的。
- default(none)：所有变量必须显式指定是私有或共享

紧跟 for 结构后面的循环变量默认是私有的，其他循环的循环变量 需显式声明成私有的。

目录

01 OpenMP编译指导语句

02 编译指导指令

03 OpenMP子句

04 任务的调度

05 运行库函数与环境变量

任务的调度

- 在循环共享结构中，将任务划分后分发给各个线程称为**调度(schedule)**
- 任务调度的方式直接影响程序的效率：

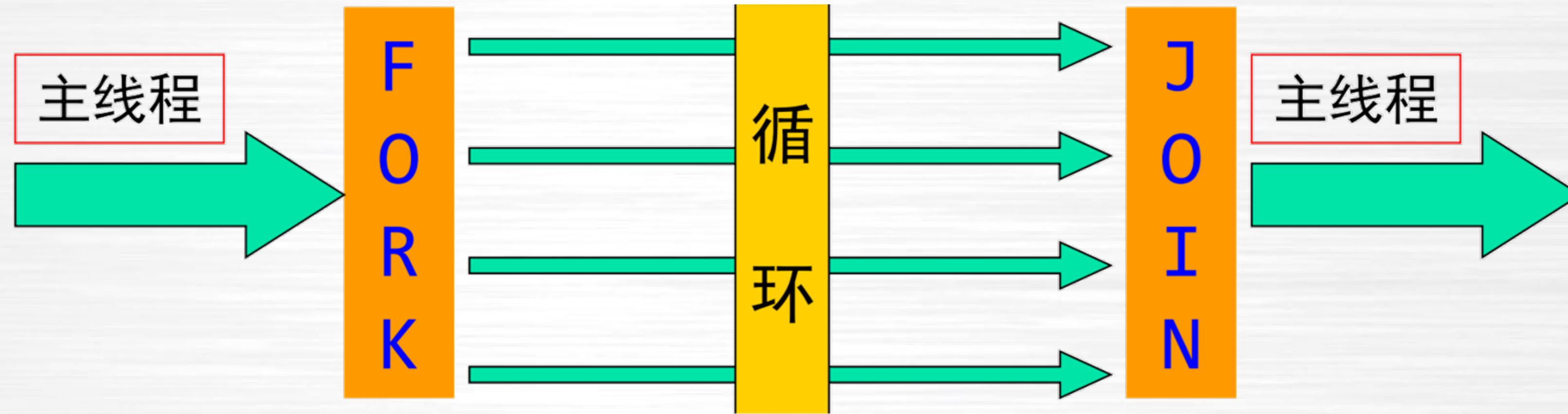
  （1）任务的均衡程度 （2）循环体内数据访问顺序与相应的 cache 冲突情况

循环体任务的调度基本原则

分解代价低：分解方法要快速，尽量减少分解任务而产生的额外开销

任务计算量要均衡，尽量避免高速缓存（cache）冲突，提高缓存命中率。

任务的调度



任务调度 SCHEDULE

- **SCHEDULE(static, size)**
  静态分配，**size**为任务块的大小，每个任务块被轮转分配给各线程

- **SCHEDULE(dynamic, size)**
  动态分配，**size** 为任务块的大小，按先来先服务原则分配

- **SCHEDULE(guided, size)**
  动态分配，任务块大小可变，先大后小，**size** 指定最小任务的大小

- **SCHEDULE(runtime)**
  具体调度方式到运行时才进行，由环境变量 OMP_SCHEDULE 确定

目录

01 OpenMP编译指导语句
02 编译指导指令
03 OpenMP子句
04 任务的调度
05 运行库函数与环境变量

运行库函数与环境变量

OpenMP标准定义了一个应用程序编程接口来调用库中的多个函数。

■ 得到线程队列中的线程数

- Fortran:
```
interger function OMP_GET_NUM_THREADS ()
```
- C/C++:
```
#include<omp.h>
int omp_get_num_threads()
```

■ 得到执行线程的线程号：

- Fortran:
```
Interger function OMP_GET_THREAD_NUM ()
```
- C/C++:
```
#include<omp.h>
int omp_get_thread_num()
```

■ 设定执行线程的数量：

- Fortran:
```
routine OMP_SET_NUM_THREADS ( )
```
- C/C++:
```
#include<omp.h>
omp_set_num_threads()
```
- 在制导语句中通过 NUM_THREADS设定。
- 通过环境变量OMP_NUM_THREADS 设定。

运行库函数与环境变量

OpenMP提供环境变量用来控制并行代码的执行

设定线程数环境变量：

例如：

1. OMP_NUM_THREADS：设定最大线程数。

   export OMP_NUM_THREADS=4

2. OMP_SCHEDULE：设定 do / for 循环调度方式环境 变量。

   export OMP_SCHEDULE= “DYNAMIC，4”

3. OMP_DYNAMIC：确定是否动态设定并行域执行的 线程数，其值为 FALSE 或 TRUE 。

   export OMP_DYNAMIC=TRUE

运行库函数与环境变量

环境变量C程序示例

```
#include<omp.h>
#include<stdio.h>
main()
{
	omp_set_num_threads(4);
	#pragma omp parallel num_threads(2)
	printf( “my thead number is %d\n",omp_get_thread_num());
}
```

- num_threads子句用来指定并行域内使用线程的个数，随后的其它并行域不受此影响
- num_threads子句的优先权高于库例程omp_set_num_threads和环境变量NMP_NUM_THREADS

谢 谢