太原理工大学
TAIYUAN UNIVERSITY OF TECHNOLOGY

第5章
用OpenMP进行共享内存编程

岳俊宏
E-mail:yuejunhong@tyut.edu.cn; Tel:18234095983

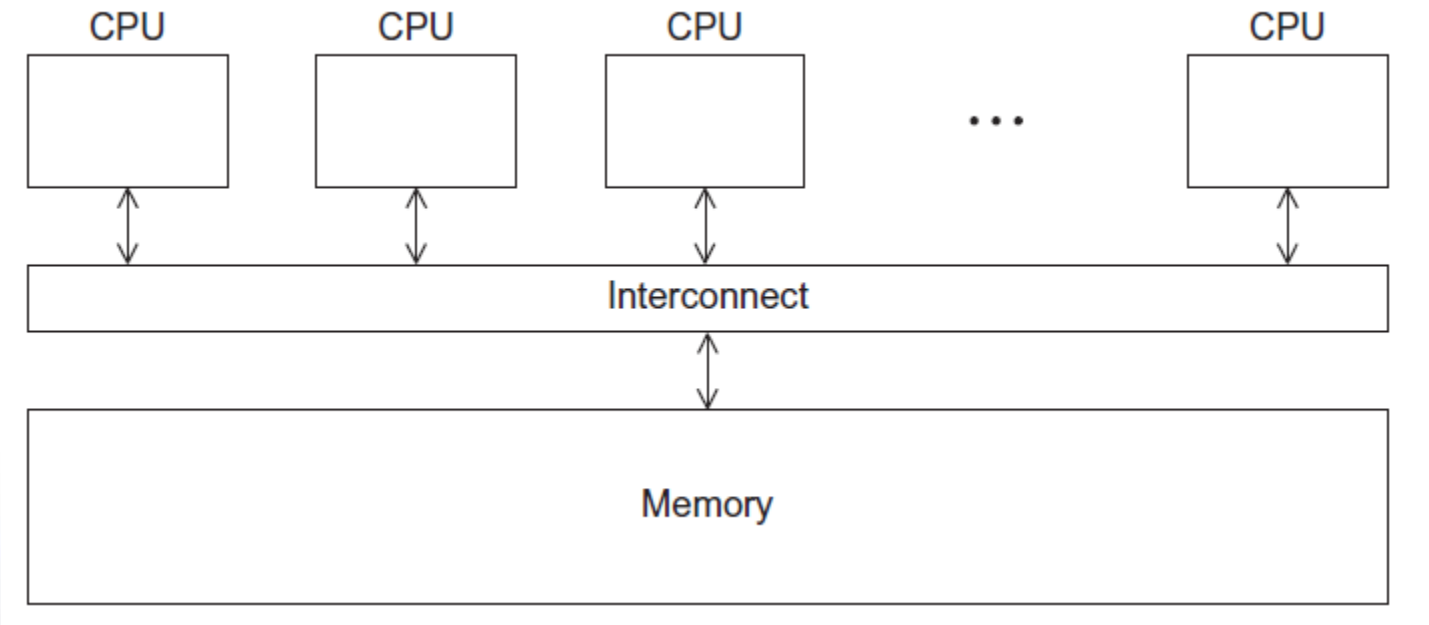
目录

OpenMP 简介

OpenMP简介

- OpenMP是一个针对共享内存编程的API。它包括一套编译指导语句和一个支持函数库。
- MP = multiprocessing（多处理）
- 在系统中，OpenMP的每个线程都有可能访问所有可以访问的内存区域
- 共享内存系统：将系统看做一组核或CPU的集合，它们都能访问主存.

OpenMP简介



共享内存系统

OpenMP与Pthreads区别

OpenMP和Pthread联系和区别

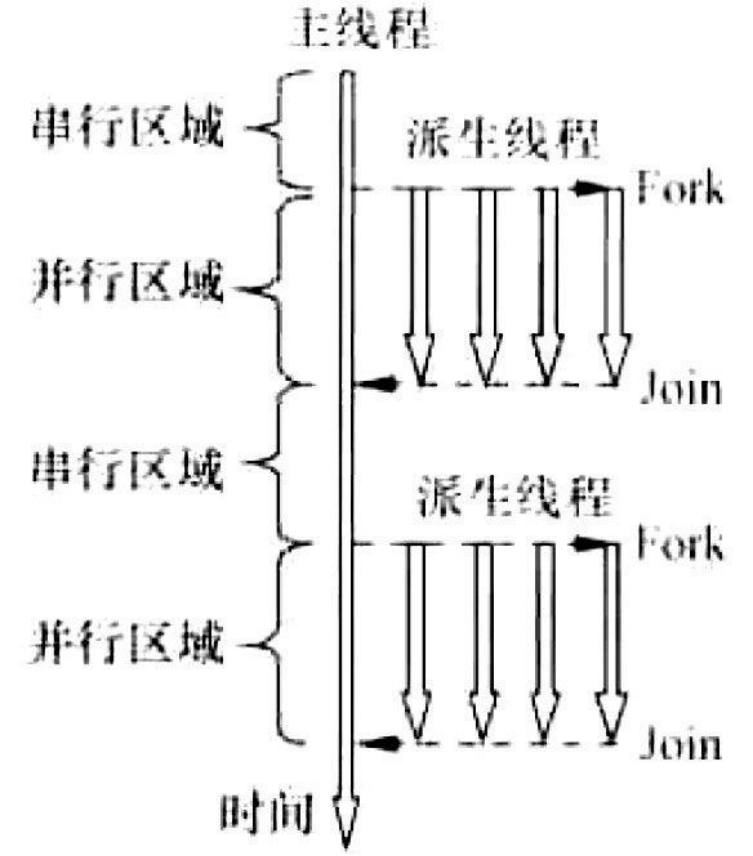
✓都是针对共享内存编程的API

- ✓ Pthreads要求程序员显式地明确每个线程的行为；
- ✓ Pthread更底层，可提供虚拟地编写任何可知线程行为的能力。
- ✓ 编写困难
- ✓ Pthreads是一个能够被链接到C程序的函数库；（任意C编译器）

- ✓ OpenMP只需简单声明一块代码应该并行执行；
- ✓ OpenMP是通过编译器和运行时系统来决定线程行为的一些细节。
- ✓ 编写容易（可对串行程序增量式并行化）
- ✓ OpenMP要求编译器支持某些操作；

OpenMP并行执行模式

采用“Fork/Join”方式



OpenMP程序编写

- OpenMP是“基于指令”的共享内存API。
- OpenMP的头文件为omp.h
- omp.h是由一组函数和宏库组成

- stdlib.h头文件即standard library标准库头文件。stdlib.h里面定义了五种类型、一些宏和通用工具函数。

OpenMP程序编写

- **预处理指令**是在C和C++中用来允许不是基本C语言规范部分的行为； eg:特殊的预处理器指令pragma。
- 不支持pragma的编译器会忽略pragma指令提示的那些语句.
- 允许使用pragma程序在不支持它们的平台上运行。

OpenMP程序编写

- C和C++中，有一些特殊的预处理器指令以#pragma开头
  - ✓ #: 放在第一列
  - ✓ pragma: 与其它代码对齐
  - ✓ pragma的默认长度是一行
  - ✓ 如果有一个pragma在一行中放不下，那么新行需要被转义——’\’
- OpenMP中，预处理器指令以#pragma omp开头

```
#   pragma omp parallel clause1 \
clause2 … clauseN new-line
```

OpenMP程序编写

- OpenMP编译指导语句格式
- 编译指导语句由directive(指令)和clause list(子句列表)组成。在C/C++中，OpenMP编译指导语句的使用格式为：

```
#pragma omp parallel [clause…] new-line
Structured-block
```

OpenMP程序编写

例如:

```
#pragma omp parallel private(i,j)
```

其中parallel就是指令，private是子句，且指令后的子句是可选的。

OpenMP程序编写

- 编译指导语句后面需要跟一个new-line（换行符）。然后跟着的是一个structured-block.
  - ✓ structured-block：for循环或一个花括号对（及内部的全部代码）
- 当整个编译指导语句较长时，也可以分多行书写，用C/C++的续行符“\”连接起来即可。如

```
#progma omp parallel claused1 \
        Claused2 … clausedN new-line
Structured-block
```

OpenMP程序编写——实例

```
#include <stdio.h>

int main(int argc, char *argv[]){
   int i;
   for (i=0; i<10; i++){
            printf("i=%d\n",i);
      }
      printf("Finished.\n");
      return 0;
}
```

OpenMP程序编写——实例

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

int main(int argc, char *argv[]){
   int i;
   int thread_count = strtol(argv[1],NULL,10);
# pragma omp parallel for num_threads(thread_count)
        for (i=0; i<10; i++){
                printf("i=%d\n",i);
        }
        printf("Finished.\n");
        return 0;
}
```

OpenMP程序编译运行

```
gcc −g −Wall −fopenmp −o omp_example omp_example .c
```

compiling

```
. /omp_example 4
```

running with 4 threads

实例-运行结果

```
[hdusr@Node1 ch5]$ ./example_omp 4
i=3
i=4
i=5
i=0
i=1
i=2
i=8
i=9
i=6
i=7
Finished.
```

```
[hdusr@Node1 ch5]$ ./example_omp 4
i=8
i=9
i=0
i=1
i=2
i=3
i=4
i=5
i=6
i=7
Finished.
```

```
[hdusr@Node1 ch5]$ ./example_omp 4
i=8
i=9
i=0
i=1
i=2
i=6
i=7
i=3
i=4
i=5
Finished.
```


Hello World
程序

Hello World程序

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

void Hello(void);  /* Thread function */

int main(int argc, char* argv[]) {
   /* Get number of threads from command line */
   int thread_count = strtol(argv[1], NULL, 10);

#  pragma omp parallel num_threads(thread_count)
   Hello();

   return 0;
}  /* main */

void Hello(void) {
   int my_rank = omp_get_thread_num();
   int thread_count = omp_get_num_threads();

   printf("Hello from thread %d of %d\n", my_rank, thread_count);

}  /* Hello */
```

OpenMP程序编译运行

```
gcc −g −Wall −fopenmp −o omp_example omp_example .c
```

compiling

```
. /omp_example 4
```

running with 4 threads

```
Hello from thread 0 of 4
Hello from thread 1 of 4
Hello from thread 2 of 4
Hello from thread 3 of 4
```

possible outcomes

```
Hello from thread 1 of 4
Hello from thread 2 of 4
Hello from thread 0 of 4
Hello from thread 3 of 4
```

```
Hello from thread 3 of 4
Hello from thread 1 of 4
Hello from thread 2 of 4
Hello from thread 0 of 4
```

OpenMP运行

- 当从命令行启动程序时，操作系统启动一个单线程的进程进行执行main函数中的代码。
- 在程序的第11行的OpenMP指令： # pragma omp parallel，该指令用来提示程序应该启用一些线程。
- 每个被启用的线程都执行Hello函数，并且当线程从Hello调用返回时，该线程就被终止。
- 在执行return语句时进程被终止。

OpenMP指令

pragma omp parallel

- ✓ 最基本的并行指令
- ✓ parallel指令是用来表示之后的结构化代码块应该被多个线程并行执行。
- ✓ 结构化代码块是一条C语言语句或者只有一个入口和出口的一组复合C语句

OpenMP子句

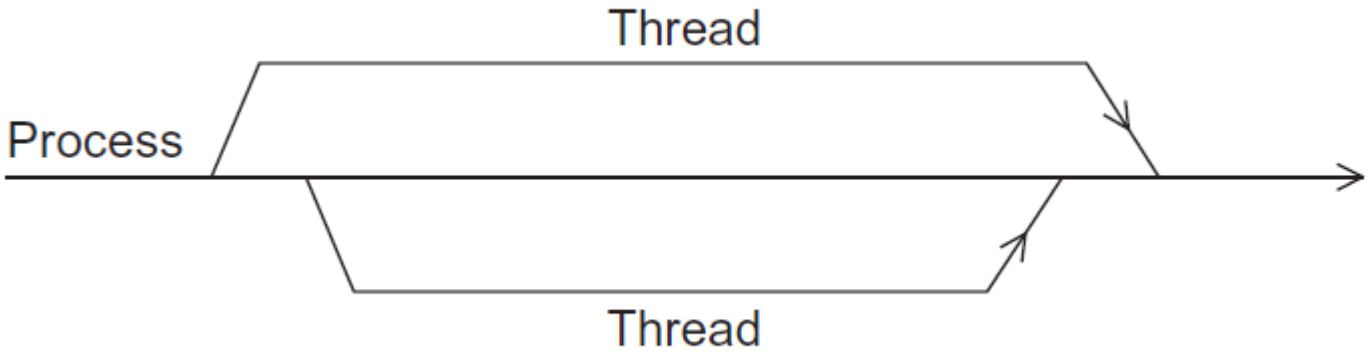
- 修改指令的文本
- num_threads子句加在 parallel指令之后
- 用来指定执行之后的代码块的线程数目.

```
# pragma omp parallel num_threads ( thread_count )
```

OpenMP线程

- 在Pthreads中，派生和合并多个线程时，需要为每个线程的特殊结构分配存储空间，需要使用一个for循环来启动每个线程，并使用另一个for循环来终止这些线程。
- 在OpenMP中，不需要显式地启动和终止多个线程。它比Pthreads层次更高。

OpenMP线程



- 启动一个进程，然后由进程启动这些线程。
- 线程共享启动它们的进程的大部分资源,如对标准输入和标准输出的访问，但每个线程有它自己的栈和程序计数器。

OpenMP线程

- 在parallel之前，程序只使用一个线程，而当程序开始执行时，进程开始启动。当程序达到parallel指令时，原来的线程继续执行，另外thread_count-1个线程被启动。
- 在OpenMP语法中，执行并行代码块的线程集合（由原始线程和新产生的线程组成）被称作**线程组**;
- 原始的线程被称**作主线程**,新产生的线程被称作**从线程**.

OpenMP线程

- 当线程从Hello调用中返回时，有一个隐式路障。即完成代码块的线程将等待线程组中的所有其他线程返回。
- 当所有线程都完成了代码块，从线程将终止，主线程将继续执行之后的代码。
- 每个线程有它自己的栈，所以一个执行Hello函数的线程将在函数中创建它自己的私有局部变量。

OpenMP函数

- 获得每个线程的编号（0-thread_count-1）：

  int omp_get_thread_num(void);

- 获得线程组中的线程数：

  int omp_get_num_threads(void);

- 注意因为标准输出被所有线程共享，所以每个线程都能够执行printf语句，打印它的线程编号和线程数。由于对标准输出的访问没有调度，因此线程打印它们结果的实际顺序不确定。

OpenMP错误检查

- 如果编译器不支持OpenMP，那么它将只忽略parallel指令。
- 头文件omp.h以及调用omp_get_thread_num和omp_get_num_threads将引起错误。
- 为了处理这些问题，可以检查预处理器宏_OPENMP是否定义。如果定义了，则能够包含omp.h并调用OpenMP函数。

OpenMP错误检查

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

```
#ifdef _OPENMP
# include <omp.h>
#endif
```

```
# ifdef _OPENMP
   int my_rank = omp_get_thread_num ( );
   int thread_count = omp_get_num_threads ( );
# e l s e
   int my_rank = 0;
   int thread_count = 1;
# endif
```

小结

- 指令：
  - ✓ parallel：用在一个代码段之前，表示这段代码将被多个线程并行执行。
  - ✓ parallel for: 是用在一个for循环之前，表示for循环的代码将被多个线程并行执行。
- 子句
  - ✓ private: 指定每个线程都有它自己的变量私有副本。
  - ✓ num_threads：用来指定执行之后的代码块的线程数目
- 函数
  - ✓ omp_get_num_threads：返回当前并行区域中的活动线程个数。
  - ✓ omp_get_thread_num: 返回线程号。

上机实验

编写运行书上的OpenMP “Hello, World”程序

结束!