**上海大学 计算机学院**

**《计算机系统结构》报告**

**姓名 徐驰 学号 20124652 指导教师 沈文枫**

**实验名称: 多核环境下OpenMP并行编程**

**一、实验目的**

1.在Linux平台上编译和运行OpenMP程序。

2.在Windows平台上编译和运行OpenMP程序。

3.掌握OpenMP并行编程基础。

4.用OpenMP实现最基本的数值算法“矩阵乘法”。

5.掌握for编译制导语句。

6.对并行程序进行简单的性能调优。

**二、实验原理**

1.OpenMP简介：

·OpenMP是一种共享内存并行编程模型，可以在多核处理器上并行执行程序。

·OpenMP使用指令来标记要并行化的代码段，并使用线程来执行这些代码段。

·OpenMP支持多种类型的指令，例如并行for循环、并行区域和任务并行等。

2.Linux平台上的OpenMP编译和运行：

·在Linux平台上，可以使用GCC编译器来编译和运行OpenMP程序。

·要编译OpenMP程序，请确保在命令行中使用-fopenmp选项，在代码中包含<omp.h>头文件，并使用#pragma omp指令来标记要并行化的代码段。

·要运行OpenMP程序，请使用终端窗口或命令行运行可执行文件。

3.Windows平台上的OpenMP编译和运行：

·在Windows平台上，可以使用Microsoft Visual Studio或MinGW等编译器来编译和运行OpenMP程序。

·要编译OpenMP程序，请确保在代码中包含<omp.h>头文件，并使用#pragma omp指令来标记要并行化的代码段。

·要运行OpenMP程序，请使用命令行或IDE运行可执行文件。

4.矩阵乘法的并行实现：

·矩阵乘法是一种常见的数值算法，可以使用OpenMP来并行计算。

·在矩阵乘法中，可以使用嵌套的for循环来遍历矩阵，并使用#pragma omp指令来并行化内部循环。

·在将矩阵乘法并行化时，请注意避免竞争条件和数据依赖等并发问题。

5.OpenMP指令和编译制导语句：

·OpenMP支持多种类型的指令和编译制导语句，例如并行for循环、reduction指令和private指令等。

·在使用OpenMP指令和编译制导语句时，请注意遵循正确的语法和语义规则，避免并发问题和性能瓶颈。

6.并行程序性能调优：

·在调优并行程序时，可以使用一些技术来提高程序的性能。

·例如，可以使用循环展开来减少循环迭代次数，使用数据局部化来减少内存访问等。

·在调优并行程序时，请注意评估程序的性能，并使用性能分析工具来识别瓶颈和优化机会。

**三、实验内容**

**1．实验任务一：在Linux平台上编译和运行OpenMP程序**

1. 实验步骤

1.在linux中，使用vim hellomp.c

2.输入代码：

#include <omp.h>

#include <stdio.h>

int main()

{

int nthreads,tid;

omp\_set\_num\_threads(8);

#pragma omp parallel private(nthreads,tid)

{

tid=omp\_get\_thread\_num();

printf("Hello World from OMP thread %d\n",tid);

if(tid==0)

{

nthreads=omp\_get\_num\_threads();

printf("Number of threads is %d\n",nthreads);

}

}

}

3.保存退出。

4.安装gcc，使用“sudo yum install -y gcc”

5.“gcc -fopenmp -O2 -o hellomp.out hellomp.c”生成可执行程序。

6.“./hellomp.out”执行程序，多次执行并记录结果。

7.修改线程数为10“OMP\_NUM\_THREADS=10”

8.将线程数添加为环境变量“export OMP\_NUM\_THREADS”

9.修改hellomp.c,删除 omp\_set\_num\_threads(8)语句

10.“gcc -fopenmp -O2 -o hellomp2.out hellomp.c”再编译生成另一段程序。

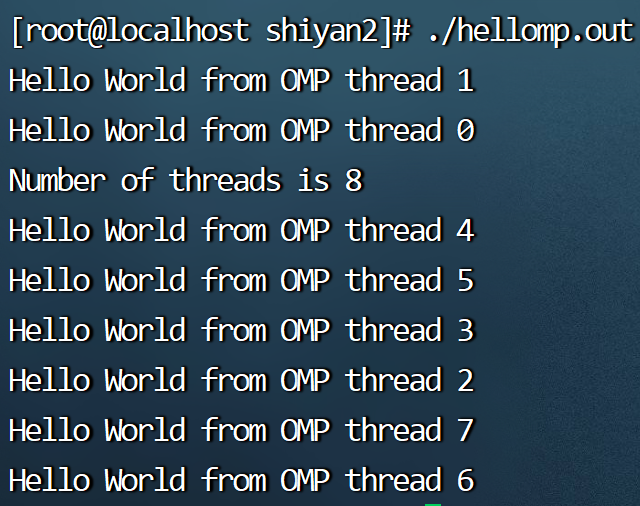
11.“./hellomp2.out”执行程序，多次执行并记录结果。

12.“unset OMP\_NUM\_THREADS”，取消环境变量设置。

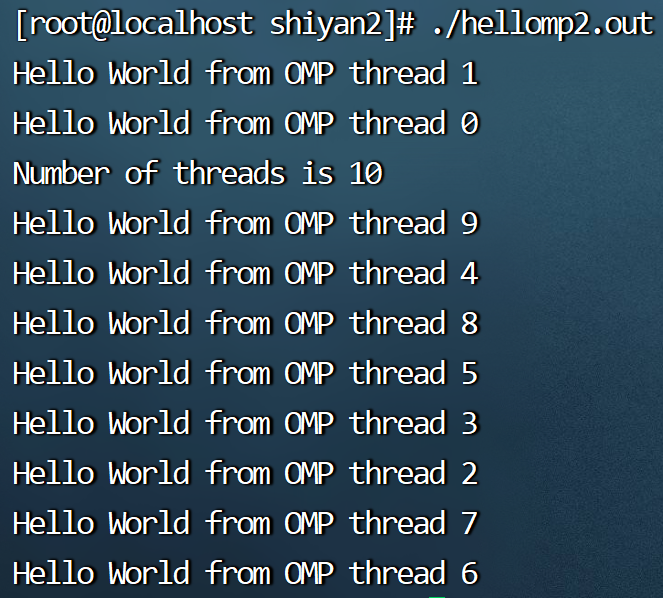
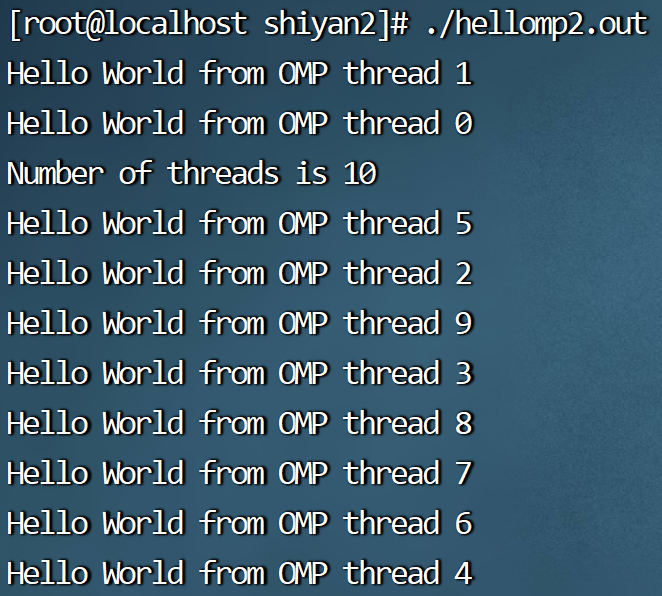
13.“./hellomp2.out”执行程序，多次执行并记录结果。

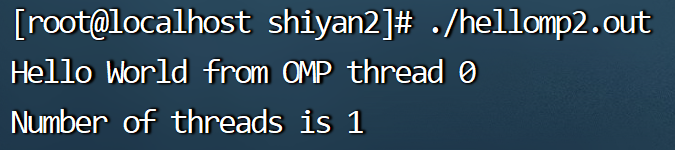
1. 实验现象

1.第6步中，实验结果：



2.第11步中，实验结果：

3.第13步中，实验结果：



1. 数据记录、分析与处理

1.代码分析

该代码使用OpenMP库实现了多线程并行计算，通过设置私有变量nthreads和tid，以及使用#pragma omp parallel指令，并结合omp\_get\_thread\_num和omp\_get\_num\_threads函数，实现了线程编号和线程总数的打印输出功能。

2.实验结果分析：

根据实验步骤，我先是使用8个线程执行程序hellomp.out，并记录了多组输出结果。由于每次执行程序时线程编号和线程总数都会随机分配，因此每次的输出结果都会有所不同，但是可以发现每次输出结果中线程总数均为8，符合设置的线程数。

接着，我将线程数修改为10，将线程数添加为环境变量，删除了设置线程数的语句，再次执行程序hellomp.out并记录多组输出结果。此时，每次线程总数均为10。

最后，取消环境变量设置，再次编译生成程序hellomp2.out，并执行多次并记录输出结果。此时，每次输出结果中线程总数不再是8或10，而是系统默认分配的数字，说明系统默认使用的线程数为最大可能的值（可能是CPU核心数），并且程序的执行速度也会受到影响。

1. 实验结论

OpenMP库可以方便地实现多线程并行计算，提高程序的执行速度。

可以通过设置环境变量或代码中的函数调用，来控制线程数的设置。

线程总数的随机分配可能会影响程序的执行速度和输出结果，因此建议在编写OpenMP程序时，尽可能使用固定的线程数来保证程序的稳定性和可重复性。

**2．实验任务二：Windows下OpenMP程序的编译和运行。**

1. 实验步骤

1.打开或者新建一个c++项目，依次选择Project -> 属性 -> 配置属性(configuration property) -> c/c++ -> 语言(Language)，打开OpenMP支持；

2.设置环境变量：我的电脑 -> 属性 -> 高级 -> 环境变量，新建一个OMP\_NUM\_THREADS变量，值设为2，即为程序执行的线程数。

1. 实验现象

与实验任务一结果一致，因为两任务不同点只在于平台不一样，代码并无区别。

**3．实验任务三：矩阵乘法的OpenMP实现及性能分析**

1. 实验步骤

┆

┆

┆

1. 实验现象

┆

┆

┆

┆

1. 数据记录、分析与处理

┆

┆

┆

1. 实验结论

┆

┆

┆

**n．实验任务n**

1. 实验步骤

┆

┆

┆

1. 实验现象

┆

┆

┆

┆

1. 数据记录、分析与处理

┆

┆

┆

1. 实验结论

┆

┆

┆

**n．实验任务n**

1. 实验步骤

┆

┆

┆

1. 实验现象

┆

┆

┆

┆

1. 数据记录、分析与处理

┆

┆

┆

1. 实验结论

┆

┆

┆

**四、建议和体会**

┆

┆

┆

**五、思考题**

┆

┆

┆

**(实验报告请上传至超星教学平台,请注意上传实验报告的截止日期,一般为一个星期。请及时上传，迟交实验报告会影响报告成绩。)**