**《并行计算》课程实验报告**

**实验2：基于共享内存的并行程序设计**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 郑佳艺 | | 院系 | | 计算机科学与技术 | | | 学号 | | 2021110666 | |
| 任课教师 | | 郝萌 | | | | 指导教师 | 张伟哲 | | | | |
| 实验地点 | | G001 | | | | 实验时间 | 2023/10/11 | | | | |
| 实验课表现 | | 出勤、表现得分 | |  | | 实验报告  得分 |  | | 实验总分 | |  |
| 操作结果得分 | |  | |
| **一、实验目的** | | | | | | | | | | | |
| 要求：需分析本次实验的基本目的，并综述你是如何实现这些目的的？  1.了解华为云环境的使用过程；  2.熟练掌握C++语言；  3.掌握PThread 开发多线程程序；  4.掌握PThread 中互斥机制的使用方式  5.掌握 OpenMP 的基本功能、构成方式、句法；  6.掌握 OpenMP 体系结构、特点与组成；  7.掌握采用 OpenMP 进行多核架构下多线程编程的基本使用方法。  如何实现：  本次实验中，我购买并配置了弹性云服务器，创建虚拟机，使用Pthread方法并行化了计算pi值的程序、CountWords程序；使用OpenMP优化了利用中值积分定理和蒙特卡洛算法计算pi值的程序。 | | | | | | | | | | | |
| **二、实验内容** | | | | | | | | | | | |
| 该部分填写在实验过程中，你都完成了哪些工作。  本实验中，我完成的工作如下：  我购买并配置了弹性云服务器，创建虚拟机；  使用Pthread方法，将原始串行的程序并行化，多线程运行，减少了运行时间；  使用OpenMP方法，优化了利用中值积分定理和蒙特卡洛算法计算pi值的程序。 | | | | | | | | | | | |
| **三、实验结果** | | | | | | | | | | | |
| Pthread： 中值积分定理计算Pi：   1. 串行：     ②并行，线程数=4：  创建进程，在for循环里使用：  具体函数实现：    具体结果：    CountWords： ①串行：  ②并行：    OpenMP：  积分中值定理计算pi：    将x设为私有变量，最后对每个线程的sum进行求和。下图为原始运行结果和并行化运行结果的对比：    可以看到原始运行时间7.855s，并行化后运行时间0.908s，提速88%。  蒙特卡洛算法计算pi：  串行代码：    修改后代码：    结果对比：    速度显著提升。 | | | | | | | | | | | |
| **四、思考题** | | | | | | | | | | | |
| 思考题1：串行程序采用PThread进行线程化基本步骤？  步骤1.添加头文件<pthread.h>，利用函数pthread\_create()创建线程；  步骤2.编写线程函数void\* Calculate(void\* arg)，函数中放入需要多线程的内容，同时注意使用互斥锁pthread\_mutex\_lock(&mutex); pthread\_mutex\_unlock(&mutex)来避免线程间的影响；  步骤3.在主程序中调用线程函数；  步骤4.等待进程结束，pthread\_join；  步骤5.释放资源pthread\_mutex\_destroy(&mutex)。  思考题2：PThread有几种共享数据互斥机制可以采用，如何应用？  机制①：互斥锁（Mutex）：互斥锁确保在任意时刻只有一个线程可以访问该资源。使用 pthread\_mutex\_init 函数进行初始化， pthread\_mutex\_lock 函数进行加锁， pthread\_mutex\_unlock 函数进行解锁，使用 pthread\_mutex\_destroy 函数销毁互斥锁。  机制②：读写锁（ReadWrite Lock）：读写锁允许多个线程同时读取共享资源，但只允许一个线程进行写操作，适用于读操作远远多于写操作的场景，可以提高并发性能。使用 pthread\_rwlock\_init 函数进行初始化，使用 pthread\_rwlock\_rdlock 函数进行读加锁，使用 pthread\_rwlock\_wrlock 函数进行写加锁，使用 pthread\_rwlock\_unlock 函数进行解锁，最后使用 pthread\_rwlock\_destroy 函数销毁读写锁。  机制③：条件变量（Condition Variable）：条件变量用于在某个条件满足时通知等待线程，从而避免线程忙等待。使用 pthread\_cond\_init 函数进行初始化，使用 pthread\_cond\_wait 函数进行等待，使用 pthread\_cond\_signal 函数或 pthread\_cond\_broadcast 函数发送信号通知等待线程，最后使用 pthread\_cond\_destroy 函数销毁条件变量。  机制④：屏障（Barrier）：同步多个线程，在所有线程都到达屏障点之前，它们将被阻塞。一旦所有线程都到达屏障点，它们将被释放并可以继续执行。可以使用 pthread\_barrier\_init 函数进行初始化，使用 pthread\_barrier\_wait 函数进行等待，最后使用 pthread\_barrier\_destroy 函数销毁屏障。  思考题3：OpenMP的主要功能，基本构成体有哪些？  标记并行区域#pragma omp parallel、并行循环#pragma omp parallel for、标记任务并行#pragma omp task、属性私有或公有#pragma omp private/shared、线程同步#pragma omp barrier等。基本构成体有：制导指令前缀+制导指令+子句。  思考题4：试分析如何使用OpenMP实现多线程并行运算，提高系统运算效能，其引入环节应如何选取？  如何使用OpenMP实现多线程并行运算：创建多线程并行域。  引入环节选取：选择for结构的循环，openmp可以对n次循环进行划分，让每个并发线程各自负责一部分循环工作；或者使用sections结构，创建工作共享域，一个线程执行一个section。  思考题5：对本次实验最开始时的pthread\_hello\_world.c修改后编译并多次运行程序，记录线程执行顺序，分析线程程序执行顺序是否不可预见及其产生原因。    线程程序执行顺序是不可预见的。线程的调度是由操作系统决定的，操作系统可能会采用不同的调度算法和策略。线程的创建和启动时间也是有一定延迟的，可能会导致线程的创建顺序和实际执行顺序之间存在差异。  思考题6：根据你的实验，OMP\_NUM\_THREADS是不是设置得越大越好？说明理由。  不是的。  1.每个线程都需要占用一定的系统资源，包括内存、CPU时间和上下文切换开销等。如果线程数设置过大，可能会超出系统的资源限制，导致系统性能下降甚至崩溃。  2.线程的创建、销毁和调度也都会引入一定的开销。如果线程数设置过大，这些开销会增加，可能会抵消掉并行化带来的性能提升。  3.如果任务的并行性较低，即任务中的计算量较少或存在较多的串行部分，设置过多的线程反而会增加线程间的竞争和同步开销，导致性能下降。 | | | | | | | | | | | |
| **五、实验心得体会** | | | | | | | | | | | |
| 本次实验我运用了PThread 开发多线程程序，掌握了PThread 中互斥机制的使用方式；学习了 OpenMP 的基本功能、构成方式、句法，并采用 OpenMP 在多核架构下多线程编程，体会到了并行计算的优势与美丽。 | | | | | | | | | | | |
| 指导教师评语：  日期： | | | | | | | | | | | |