**《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 | 刘子康 | | 院系 | | 计算学部 | | | 学号 | | 2022113416 | |
| 任课教师 | | 张伟哲 | | | | 指导教师 | 郝萌 | | | | |
| 实验地点 | | 格物213 | | | | 实验时间 | 2024.09.25 | | | | |
| 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** | | | | | | | | | | | |
| 1.熟练掌握C++语言；  2.掌握PThread 开发多线程程序；  3.掌握PThread 中互斥机制的使用方式；  4.掌握 OpenMP 的基本功能、构成方式、句法；  5.掌握 OpenMP 体系结构、特点与组成；  6.掌握采用 OpenMP 进行多核架构下多线程编程的基本使用方法。  通过查阅课程PPT和实验指导书，在华为云服务器利用PThread与OpenMP编写、编译和运行并行程序，熟练C++语言能力，深入理解和掌握PThread与OpenMP使用方法。 | | | | | | | | | | | |
| **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。  **2.1配置华为云环境**  购买1台华为云的弹性云服务器ECS，实例CPU架构选择鲲鹏计算，操作系统镜像选择openEuler，系统盘至少40GB。    通过CloudShell远程登录ECS，创建个人账户并加入wheel组，取得root权限。  **2.2并行环境下Pthread程序的编译和运行**  2.2.1 hello-world程序  创建Pthread目录，存放所有Pthread程序。创建hello-world程序源码，使用gcc pthread\_hello\_world.c -lpthread -o pthread\_hello\_world.bin和./pthread\_hello\_world.bin指令进行编译和运行，查看运行结果。  2.2.2利用中值积分定理计算Pi值  采用中值积分定理计算Pi值，编写对应串行程序源码，使用gcc Pi.c -lpthread -o pi.bin和./pi.bin指令进行编译和运行，查看运行结果。  使用PThrad进行线程化，修改程序代码，将for循环的计算过程分到几个线程中，且使用临界区，将sum放到临界区中，一次只能有一个线程访问和修改sum的值。  2.2.3互斥量的使用——CountWords  创建Serial.c串行程序源码，上传输入文件InFile1.txt，使用gcc Serial.c -lpthread -o Serial.bin和./Serial.bin指令进行编译和运行，查看运行结果。  使用PThrad进行线程化，修改程序代码，引入互斥量。将CountWords函数中的工作分配给多个线程，每个线程将处理文件的一部分，并更新全局变量（如TotalWords, TotalEvenWords, TotalOddWords）时需要使用互斥量来避免数据竞争。  **2.3并行环境下OpenMP程序的编译和运行**  2.3.1 hello-world程序  创建OpenMP目录，存放所有OpenMP程序。创建hello-world程序源码，使用gcc openmp\_hello\_world.c -fopenmp -o openmp\_hello\_world.bin和./openmp\_hello\_world.bin指令进行编译和运行，查看运行结果。  2.3.2利用中值积分定理计算Pi值  采用中值积分定理计算Pi值，编写对应串行程序源码，使用gcc pi.c -fopenmp -o pi.bin和./pi.bin指令进行编译和运行，查看运行结果。  使用OpenMP实现多线程并行计算，修改程序代码，在主程序体使用 #pragma omp parallel定义并行区域，在for循环前加上#pragma omp for指令，并使用reduction来处理sum的累加。    2.3.3 Pi值的蒙特卡洛算法  创建pimonte\_serial.c串行程序源码，使用gcc pimonte\_serial.c -fopenmp -o pimonte\_serial.bin和./pimonte\_serial.bin指令进行编译和运行，查看运行结果。  使用OpenMP实现多线程并行计算，修改程序代码。由于rand函数是非线程安全的，使用多线程时应明确指定seed值的rand\_r。在主程序体使用 #pragma omp parallel定义并行区域，在for循环前加上#pragma omp for指令，使用 #pragma omp atomic 来确保对共享变量 dUnderCurve 的更新是原子的，从而避免竞争条件。 | | | | | | | | | | | |
| **三、实验结果** | | | | | | | | | | | |
| **3.2并行环境下Pthread程序的编译和运行**  3.2.1 hello-world程序    3.2.2利用中值积分定理计算Pi值  串行程序：    并行程序：    3.2.3互斥量的使用——CountWords  串行程序：    并行程序：    **3.2并行环境下OpenMP程序的编译和运行**  3.2.1 hello-world程序        3.2.2利用中值积分定理计算Pi值  串行程序：    并行程序：    3.2.3 Pi值的蒙特卡洛算法  串行程序：    并行程序： | | | | | | | | | | | |
| **四、思考题** | | | | | | | | | | | |
| 思考题1：串行程序采用PThread进行线程化基本步骤？  定义线程函数，即线程执行的主体；使用pthread\_create函数创建线程，需传入四个参数——线程标识符的指针、线程属性（通常使用NULL表示默认属性）、线程函数的指针以及传递给线程函数的参数；在线程之间进行同步和通信，利用互斥机制使程序正确运行；等待线程结束，清理和释放资源。  思考题2：PThread有几种共享数据互斥机制可以采用，如何应用？  （1）互斥锁（Mutex）：在任何时候，只有一个线程可以锁定互斥锁，从而防止其他线程对共享数据的访问。  （2）条件变量：用于阻塞一个或多个线程，直到某个特定条件为真，通常与互斥锁一起使用，以避免竞态条件。  （3）读写锁：允许多个线程同时读取共享数据，但写入时则独占访问权，提高了读操作的并发性，同时保证了写操作的安全性。  思考题3：OpenMP的主要功能，基本构成体有哪些？  OpenMP主要用于简化并行代码的编写，支持在共享内存系统上进行多线程编程，为共享内存并行程序员提供了一种简单灵活的开发并行应用的接口模型。OpenMP能够自动管理许多并行计算的细节，如线程的创建、同步和销毁，但同时也要求开发者处理依赖关系、读写冲突和死锁等问题。  OpenMP主要由编译制导、运行时库函数、环境变量三部分构成。  （1）编译制导：以#pragma omp开头的指令，用于指导编译器对代码进行并行化处理，可以指定哪些代码块需要并行执行，以及并行执行的线程数、数据共享方式等。  （2）运行时库函数：用于在程序运行时控制并行执行的行为，包括设置线程数、获取线程号、初始化锁和销毁锁等。  （3）环境变量：通过修改环境变量的值，可以对OpenMP的行为进行定制，如设置默认的线程数等。  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？  引入OpenMP头文件<omp.h>，使用#pragma omp parallel指令将代码块标记为并行区域；在并行区域内，使用#pragma omp for指令将循环标记为并行循环，这样循环的迭代就会被分配给不同的线程执行；通过omp\_set\_num\_threads()函数或设置环境变量。OMP\_NUM\_THREADS来控制并行区域中的线程数。  对程序进行需求分析，确定哪些部分适合并行化，使用性能分析工具（如gprof、Valgrind的Callgrind等）识别程序中的热点，即那些执行时间最长或占用资源最多的部分。根据热点部分的特性选择合适的并行化策略，对于循环密集型任务，可以考虑使用#pragma omp parallel for指令进行循环并行；对于任务分解型任务，则可以使用#pragma omp parallel sections或#pragma omp task等指令进行任务并行。  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。    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，其设置需要根据具体的应用场景、硬件环境以及任务特性来决定。  OMP\_NUM\_THREADS超过CPU的逻辑核心数时，并不会带来额外的并行处理能力，操作系统会通过时间片轮转等机制来管理这些线程，让它们共享CPU资源。这意味着每个线程的实际执行时间会减少，可能导致整体效率下降。  任务的并行度决定了能够同时有效运行的线程数。如果任务本身存在大量的串行部分或数据依赖，增加线程数并不能提高并行效率。  每个线程都会消耗一定的系统资源，过多的线程会加剧系统资源的消耗，可能导致其他进程或线程的资源不足。 | | | | | | | | | | | |
| **五、实验心得体会** | | | | | | | | | | | |
| 1.学会了利用华为云服务器搭建并行计算的集群环境，了解了基于华为云的PThread和OpenMP并行程序设计流程。  2.掌握了采用PThread进行线程化与使用OpenMP实现多线程并行运算的方法，提高了C++语言代码能力。  3.编译运行简单的hello-world、CountWords和计算pi值程序，通过与串行代码比较，对并行计算的高效性有了更深的理解。 | | | | | | | | | | | |
| 指导教师评语：  日期： | | | | | | | | | | | |