《并行计算》
结课报告

姓名：陆子毅

学号：3022206045

时间：2024 年 5 月 2 日

成绩：

评分：

审核：

天津大学 智能与计算学部

2024 年

一、实验内容概述

本课程实验要求在远程超算平台上实现包括 sin 函数值的近似计算，矩阵幂次的计算任务。算法要求分别使用 Pthread，MPI，MPI+OpenMP 实现。

对于第一个实验，用 Pthread 计算 sin 的近似值，采用泰勒展开对 sin 函数在 x 处进行函数近似，求出所输入的 x 的函数值。数据分析要求计算加速比，效率等并给出图像。对加速性能进行分析。（不在结课报告中对比分析）

实验二，采用 Pthread 对矩阵幂次进行计算，算法的设计需要考虑数据的划分方式，确定并行部分，合理分配线程，协调线程之间的负载，尽量提高加速比。

实验三，使用 MPI 计算矩阵幂次。多进程协调计算，需要在实验二的基础上确定进程之间的通信数据，尽量减少通信需求和通信数据量来提升 MPI 程序的效率。

实验四，使用 MPI+OpenMP 多级并行再次计算矩阵幂次，需要在之前的基础上对计算的数据进一步划分，需要思考多级并行的具体划分方式，第一级和第二级并行需要执行哪些任务。需要计算加速比和程序效率对程序进行进一步分析。

二、并行算法分析设计

2.1 PThread 多线程算法设计

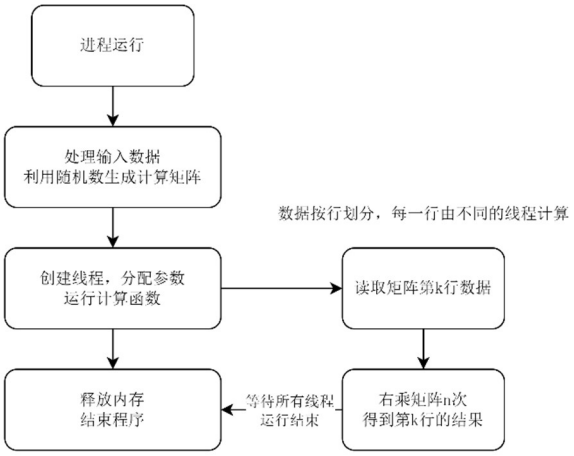


图1 PThread多线程算法设计

首先在串行计算的四重循环基础上，确定并行计算部分，为每一行的 n 次计算，因为每一行的 n 次计算都只需要用到这一行的 n-1 次幂和初始矩阵，计算过程独立，不依赖于其他线程的计算数据。

算法设计过程中还需要考虑到课上讲到的诸如伪共享，线程同步，数据划分，线程负载均衡，数据一致性等问题，设计需要考虑的细节较多。多个线程需要频繁读取同一个矩阵结构体，而缓存加载一般是以 64 字节为单位，CPU 会将包含自己所需数据的 64 字节加锁，避免其他线程访问，会导致其他需要访问内存的线程处于阻塞状态，由于每一个线程读取的是矩阵数据结构中的一行，所以避免伪共享的方案是，将矩阵一行的数据量对齐到 64 字节，来提高 CPU 读取修改速率。不同线程修改同一个内存容易造成内存数据不一致，需要添加互斥锁，来防止多个线程同时写入一个内存地址。

2.2 MPI 并行算法设计

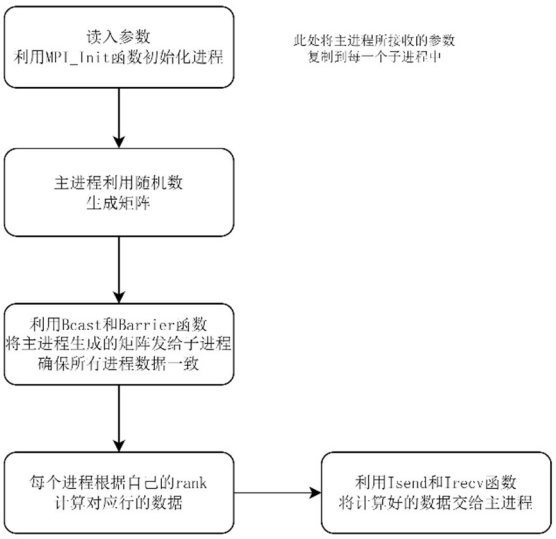


图2 MPI多进程算法设计

MPI（Message Passing Interface）是一个广泛使用的并行计算标准，用于在分布式内存系统中实现并行程序。MPI 定义了一组用于进程间通信的库函数，允许编写能够在多个处理器上运行的并行应用程序。MPI 允许在不同处理器上运行的进程之间进行通信。它提供了多种通信模式，如点对点通信、集体通信和广播等。在 MPI 中，每个进程拥有自己的私有内存空间。进程之间通过消息传递来共享数据。

利用 MPI 可以将矩阵幂次运算放到多个处理器上运行。

算法设计首先确定并行化部分，也就是每一行矩阵 n 次的计算。理由是，计算相对独立，不需要频繁的数据交换。接下来确定程序需要通信的数据，为了确保所有进程计算所使用的数据一致，需要利用组通信来将主进程利用随机数生成的矩阵交给其他进程。在计算完成之后，需要各个进程将自己计算得到的结果通过 Isend 和 Irecv 函数发送到主进程，由于多个进程同时传输数据到主进程，所以需要在每一条发送的数据上打上 tag 来区分不同行的计算结果。最后结束 MPI 环境。

2.3 MPI+OpenMP 混合并行算法设计

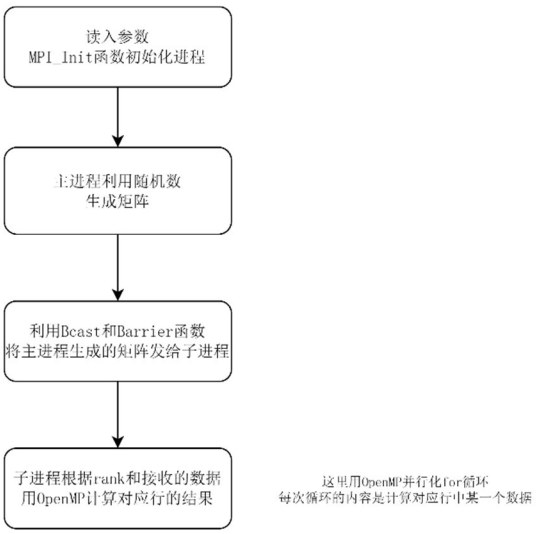


图3 OpenMP+MPI多级并行算法设计

OpenMP（Open Multi-Processing）是一个用于共享内存多处理器系统上的多线程并行编程的 API。可以利用编译制导语句和库函数简单快速地实现并行化，简单高效。

在 MPI 的基础上使用 OpenMP，考虑每个进程的计算，分析每个进程的计算，确定 OpenMP 并行化的是一行中每一个元素的计算，例如。Matrix[0][1]和 Matrix[0][2]的计算，相对独立，不依赖于彼此的计算结果，所以一行数据可以利用多线程同时计算，在计算结束后，设置同步，确保本次行乘计算完成以后再开始下一轮的计算，避免数据不一致。

相对 Pthread 计算，OpenMP 需要考虑的技术细节较少。实现较为容易。

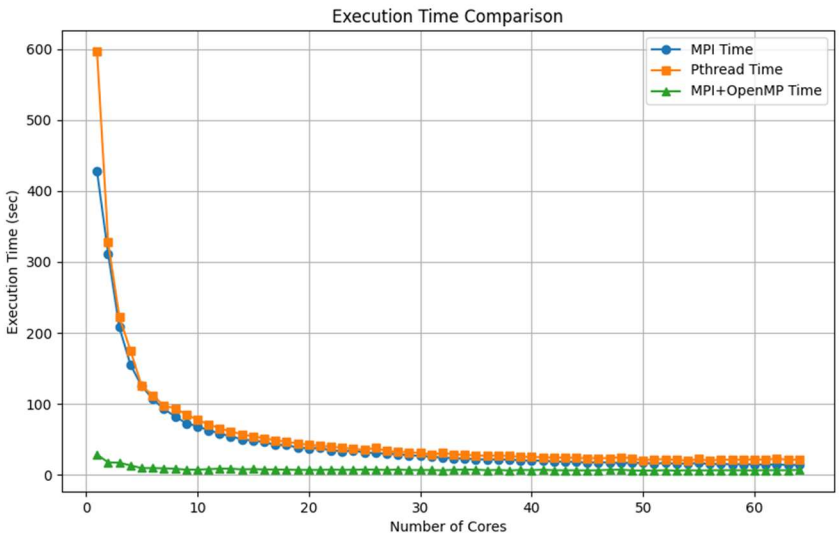
三、实验数据分析

3.1 实验环境

环境：国家计算天津中心天河超算
CPU：飞腾@2.3GHz
CPU 数量：64
运行内存容量：128GB
存储容量：>11PB
并行数据传输：1PB
内存带宽：204.8GB/s
每个核心的线程数：1
每个插槽的核心数：64
NUMA 节点数：8
制造商 ID：0x70
型号：2
步进：0x1
BogoMIPS：100.00
L1 数据缓存：2 MiB
L1 指令缓存：2 MiB

3.2 实验数据综合分析

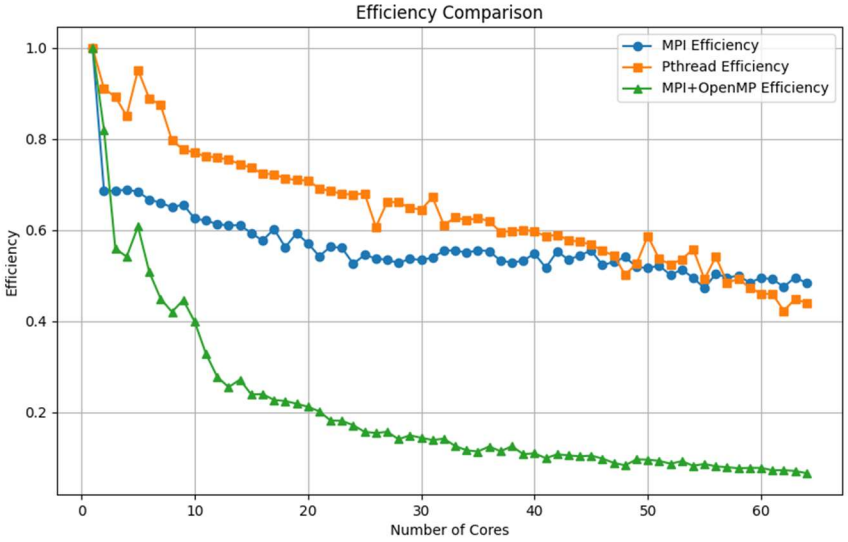
3.2.1 运行时间分析



从运行时间上来看，MPI 和 Pthread 的时间相似，原因是两者使用的都是单级并行，pthread 的时间较高的原因是，线程的创建，参数分配需要占用一定的时间，MPI 虽然需要在不同处理器之间通信，但是通信数据量较少，并且使用的节点在物理距离上相距较近，所以通信时间基本可以忽略不计。

而由于 OpenMP+MPI 使用的是多级并行，并行程度更大，所以运行时间相较于 pthread 和 MPI 更短。

3.2.2 效率分析



MPI Efficiency：随着核心数的增加，MPI 的效率逐渐下降，这表明增加核心数所带来的性能提升正在减少。

Pthread 与 MPI 混合模型的结果与 MPI 相似。

效率是加速比除以核心数的结果，用于衡量并行计算的性能。理想情况下，对于完美并行化的程序，效率应该接近 100%。

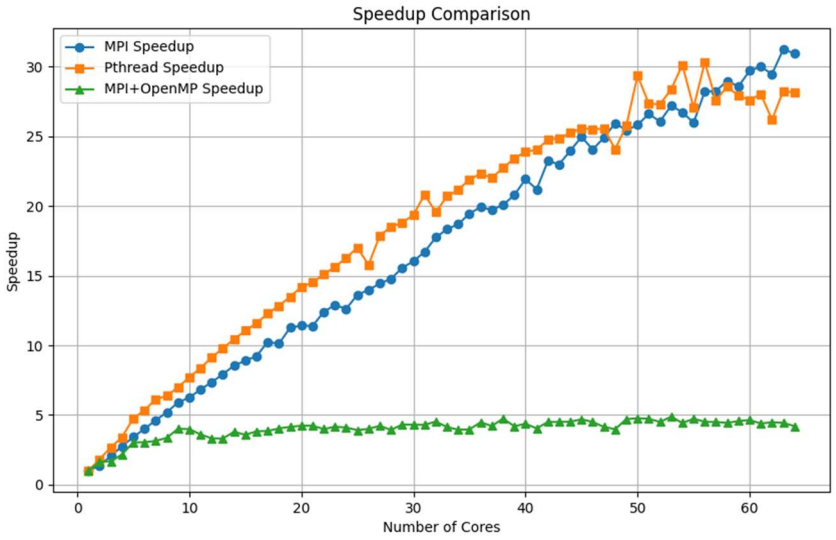
所有模型的效率随着核心数的增加而下降，这表明并行开销（如通信、同步等）开始占据主导效率下降通常意味着并行开销开始占据主导，如通信延迟、负载不均衡、资源竞争等。

比较不同模型：

MPI：随着核心数的增加，MPI 的效率下降，这可能是由于并行开销的增加，如通信延迟和同步开销。这表明在 MPI 并行化中可能存在优化空间，比如通过减少通信或改进数据分发策略。

Pthread：Pthread 的效率下降可能表明线程之间的竞争加剧，或者上下文切换的开销增加。优化线程的管理工作和减少锁的竞争可能可以提高效率。

3.2.3 加速比分析



Pthread 加速比分析：

Pthread 加速比在核心数较少时增长迅速，这可能是因为线程之间的通信开销较小，且共享内存的访问速度较快。当核心数增加到一定程度后，Pthread 的加速比增长也开始减缓，这可能同样是由于并行开销的增加。随着核心数的增加，Pthread 的效率逐渐下降，这表明线程之间的竞争和上下文切换等开销开始影响性能。结合 amdahl 定律去分析，计算分析得到串行部分占比约为 0.015，反推加速比，随着核心数量的增加不可能超过 66，分析可知，在 64 核心的时候，没有达到 Amdahl 定律的极限加速比，说明随着核心数量的提升，加速比仍然会上升，从图像上看也确实如此，为了确保三个程序的计算规模相同，使用了较大的矩阵来计算，导致 64 核心时，加速比没有稳定，另一方面也说明，在数据量非常大的时候，Pthread 能够充分利用计算资源，比较符合效率图像。

MPI 加速比分析：

在较低核心数时，MPI 加速比随着核心数的增加而迅速增长，这表明 MPI 并行模型在初期能够有效地利用额外的核心来加速计算。当核心数超过 20 个之后，加速比的增长速度开始放缓，这可能是由于并行开销（如通信延迟、同步开销）开始占据主导地位，减少了额外核心带来的性能提升。随着核心数的进一步增加，加速比趋于饱和，这表明 MPI 程序的可并行化部分已经被充分利用，进一步增加核心数难以获得更多的性能提升。根据 Amdahl 定律，计算分析得到串行部分占比约为 0.015，和 Pthread 得到的结果相似，所以结论也相似。在如此大的数据规模下，MPI 能够充分利用每一个处理器，与上面的效率分析一致。

OpenMP+MPI 分析：

由于程序默认是启用 OpenMP 多线程的，所以图中的加速比数据意义不大，侧重于在默认开启的情况下，MPI 的使用对于程序加速比的提高上，根据图像可以看出，在采用了 OpenMP 以后，增加进程数量对于加速比的提升作用比较小，说明随着进程数量的增加，程序对每个处理器的利用率变低了。

对比总结：

初期性能：在核心数较少的情况下，Pthread 由于线程间的通信开销较小，可能会展现出更快的加速比增长。

可扩展性：随着核心数的增加，MPI 和 MPI+OpenMP 由于其设计上更适合大规模并行计

算，可能会展现出更好的可扩展性。

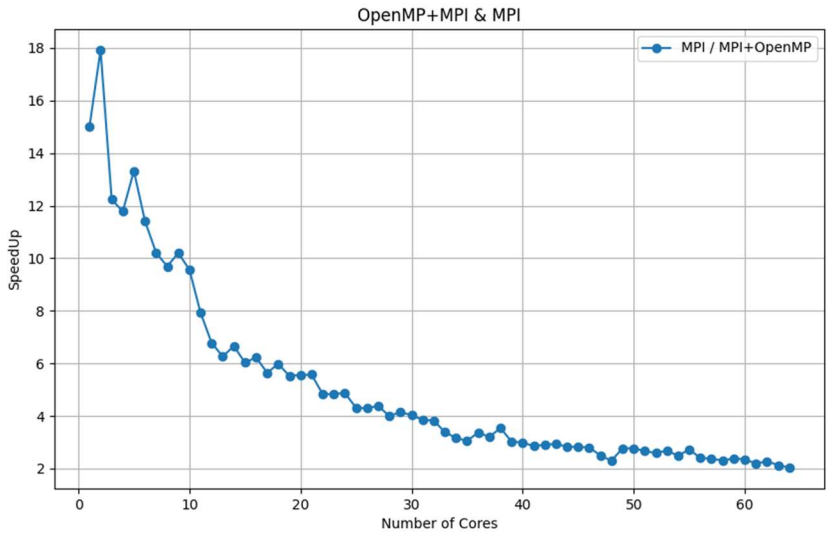
性能饱和：所有模型都会遇到性能饱和的问题。

效率考量：虽然加速比是衡量并行性能的一个重要指标，但效率也同样重要。在选择并行模型时，应综合考虑加速比和效率，以及程序的特定需求和硬件环境。

瓶颈识别：如果加速比低于预期，需要深入分析程序的并行部分和串行部分，识别并解决性能瓶颈，如优化通信协议、减少同步点、改进数据访问模式等。

模型选择：根据程序的特点和运行环境，选择最合适的并行模型。对于需要大规模分布式计算的应用，MPI 可能是一个好选择；而对于主要在单个节点上运行的多线程应用，Pthread 可能更为合适；MPI+OpenMP 混合模型则可能适用于需要同时利用多节点和多核的场景。

3.2.3 OpenMP+MPI 与 MPI 的对比分析



使用 **MPI 运行时间/OpenMP+MPI 运行时间**得到的图像，可以近似作为使用 OpenMP 后的加速比，默认线程数量是 64，可以看到，加速比呈现出下降的趋势，分析可能是由于在处理器数量增加的情况下，OpenMP 带来的线程创建开销变大，也有可能是因为，每个进程所需要计算的数据量减少，使用 OpenMP 带来的收益减少，就像是使用 Pthread 计算最后呈现出来的趋势一样，效率是不断下降的。

四、实验总结

4.1 困难总结以及收获

（1）最大的困难，就是 debug 过程中，尤其是像利用 MPI 进行编程的时候，会出现很多错误，错误的说明也不像 GDB 那样简单易懂方便修改，比如在第一次使用 MPI 环境编程的时候出现了 overflow，栈溢出，而这一行小字夹在很多错误的中间，找了很久才解决，而且，程序不允许野指针以及未释放的内存存在，只是在运行过程中出错，很难找到出错的地方，只能在程序中一行一行加入输出来排查错误。这个过程极大地锻炼了耐心和细心，考验检索资料和获取重点信息的能力。

（2）第二个困难是，程序的运行结果往往会和预期的不相符，有的时候是因为程序的优化并不完善，比如线程的创建和销毁的开销过大，可以利用线程池来减少。比如伪共享，解决数据一致性带来的死锁，互相等待等。还有的时候是因为数据规模，提交任务时申请使用的节点数量，结果的不理想往往和很多因素相关，需要综合考虑各方面的因素，对出现的现象进行解释分析，锻炼解决问题和发现问题的能力。

（3）得到数据以后，要结合 Amdahl 等定律对程序进行分析，为什么程序没有达到预期效果，为什么效率会很低，等等，需要根据分析结果继续修改程序或者数据规模去验证想法与猜测。最后能得到一个相对较为正确的结果，并在一步一步验证的过程中对于并行计算的原理以及性能有着更深一步的理解。

4.2 对不同类型并行计算方式的理解与分析。

首先是 Pthread 并行化，Pthreads 是一种基于线程的并行计算方式，它允许一个进程创建多个线程，这些线程共享进程的内存空间。Pthreads 适用于多核处理器系统，可以有效地利用 CPU 的多核特性来提高程序的执行效率。Pthreads 的并行主要是在单个机器上实现的，因此它的通信开销相对较小。使用 Pthreads 时，程序员需要自己管理线程的创建、同步和销毁。

其次是 MPI，与 Pthread 一样，MPI 也是单级并行，所以并行化的方式相似，MPI 是一种基于消息传递的并行计算模型，它允许多个独立进程之间通过发送和接收消息来通信和同步。MPI 主要用于分布式内存系统，如集群或超级计算机，它在不同的计算节点上运行独立的进程，这些进程通过消息传递来交换数据。MPI 的通信机制使得它在大规模集群上具有良好的可扩展性。相比与 Pthread，MPI 可以更好的利用分布式大规模系统的性能。

最后是 MPI+OpenMP，MPI 与 OpenMP 的结合使用是一种混合并行编程模型。在这种模型中，MPI 用于处理节点间通信，而 OpenMP 用于处理节点内的多核并行。这种混合模型结合了 MPI 的可扩展性和 OpenMP 的简便性，适用于那些既需要节点间通信又需要节点内多核并行的复杂应用。

通过对比，可以知道 Pthreads 适合于共享内存系统中的多核并行计算，MPI 适合于大规模分布式系统中的并行计算，通过实验可以发现，MPI 和 OpenMP 的运行性能相近，MPI+OpenMP 适合于既需要节点间通信又需要节点内并行的混合环境，可以在复杂的环境中充分发挥系统的性能。

MPI 基于更高的操作系统抽象，更适合粗粒度的并行划分，而 Pthread 和 OpenMP 基于线程并行化，适合细粒度的并行划分，多级结合可以让程序的并行化进一步提高。

同时，随着性能利用的不断提高，并行化的不断提高，所适用的场景也不同，单级并行化适合数据规模相对不是很大的程序，其中 OpenMP 和 Pthread 适合在单节点上运行，而 MPI 更适合在大规模集群系统和分布式系统上运行。OpenMP 和 MPI 相结合使用可以胜任数据规模更庞大的工作，实测 600x600 矩阵的 15 次幂，约 600^{45} 次计算，仍然没有让多级并行化达到性能极限。

此外，如果利用 OpenMP 和 MPI 来计算规模较小的数据，性能与单级并行所差无几，甚至可能因为各种开销而慢于单级并行。

在实验过程中，发现随着核心数量的提高，加速比的增速是放缓的，也就是说，在超过一定阈值之后，增加核心数量所带来的收益是不断减少的，所以在实际应用过程中所申请的节点或者处理器数量不用太大，控制在合理范围内可以是资源的分配和利用更加合理。

五、课程总结

5.1 本学期授课方式有助于提升学习质量的方面

课程的安排非常清晰，每堂课都有预习内容，理论课和实验课相结合，使得学习更加高效。在理论课上，老师会系统地讲解相关知识，而每节课后都有一次作业，有助于巩固理论知识，也为后续实验提供了复习的机会。相比于先学理论再进行实验的课程安排，这种交替进行的方式更合理，避免了在实验时忘记理论知识的尴尬情况。

实验课的安排也十分贴合理论课的内容，能够让我更加直观地感受到理论知识在实际操作中的应用。通过实验，我不仅加深了对理论的理解，还提高了解决问题和动手能力。这种理论和实践相结合的教学方式，让我学到的知识更加系统和全面，也让学习过程更加生动有趣。

课程的安排十分合理，理论与实践相辅相成，正反馈及时到位。这种教学方式不仅提高了学习效率，也增强了我对所学知识的理解和记忆。

5.2 不合理之处及建议

对于实验报告的内容需要进行调整，可以考虑两种方式：要么只给出大纲，要么将细节写清楚。这样可以让学生更清晰地了解实验报告的要求和结构，同时也有助于他们更好地理解实验内容和目的。如果只提供大纲，学生可以更自由地根据大纲的指引来完成报告，但需要确保大纲涵盖了实验报告所需的所有内容。另一方面，如果提供了详细的内容要求，学生则可以更加系统地编写实验报告，确保每个部分都得到了充分的阐述和解释。

实验可以定期查收，这有助于避免学生最后一刻的赶工。定期查收实验报告可以促使学生按时完成实验，并在过程中及时发现和纠正问题，从而避免最后一刻的赶工和不完整的报告。通过定期查收，老师可以及时给予学生反馈，指导他们在实验报告的撰写过程中改进和提高,从而提高学生的学习效果和报告质量。同时,定期查收还有助于监督学生的学习进度，确保实验课程的顺利进行。

应该允许学生有一次重新提交作业或者实验的机会，学生提交了错误的报告，或者报告里面忘记加入一些文件，或者想要丰富一下自己的报告，可以给一次重新提交的机会，但是分数上可以适当减少一分两分，重交次数不超过三次，这样是比较合理的。

附：上机实验与课程知识点分析

序号	上机实验内容	理论知识点	分析总结
1	熟悉天河环境，初步修改代码	并行程序的编译运行，Pthread 库	利用智慧树上的代码，在天河实验环境下编译运行，熟悉 terminal 指令。结合课程上对于 Pthread 库的讲解理解程序并进行初步修改
2	编写 sh 脚本，测试程序		利用 sh 进行程序运行，可以高效得到实验数据。结合并行计算性能分析对实验结果进行分析。

3		并行计算类型	时间并行的流水线方式简单易实现，但提升计 算速度有限。 空间并行的域分解和任务分解能够处理复杂问题，但需要考虑优化和任务分配等问题。
4		SIMD，双核与超线程，存储访问模式	特性要求：数据结构：连续存储、数据对齐、元素大小相同。算法：可并行性、各部分相互独立、负载均衡。 双核：真实的两个核心；超线程：单核心模拟两个核心，可能出现性能问题。 SMP：UMA 均匀存储访问。 MPP：NUMA 非均匀存储访问。 Cluster：分布式存储访问。
5		线程与进程区别，竞态条件	线程是系统资源分配的基本单位，进程是 CPU 调度的基本单位。线程不独立拥有资源，但可以访问隶属进程的资源；进程拥有系统资源。创建线程消耗资源较少，多线程更轻量化。线程共享资源和打开文件，而进程相对独立，无法互相访问。线程切换不会引起进程切换，节省系统开销。 竞态条件出现情况：多个线程同时读写共享数据。
6		Amdahl 定律与 Gustafson 定律	对于并行计算性能的大致分析与预测，但是两种算法都具有局限性。只是用来大致的预测与验证，以及解释一些实验现象。
7		OpenMP	通过“编译制导语句”将一个串行程序快速地转变为并行程序。可以看作是一个工具而不是一种独立的语言。
8		GPU 编程与 CUDA	线程，线程块，以及网格构成了 GPU 的三层结构，通过 CPU 控制，实现异构计算。可以让 GPU 中的多个线程块同时运算。
9	矩阵乘法		熟悉矩阵相乘的计算机实现方法，确定矩阵幂 次计算的划分方式，以及具体如何划分，根据任务书提示编写程序。
10	Lab2 算法性能分析与优化		不断地对算法进行优化，针对伪共享以及线程开销进行优化，测试并得到实验结果。 体会到数据规模（任务量），以及线程数量之间的关系。
11		MPI、集群、与作业管理系统	集群与 MPP： 集群：通用操作系统连接节点，独立机器组成。 MPP：定制组件，高速网络连接处理器，共享操作系统。

			线程跨节点限制： 进程在节点内运行，线程共享进程资源，不能跨节点。 作业管理系统： 管理资源、调度任务、监控运行、提高利用率。 MPI 六调用： Init/Finalize：初始化/释放 MPI 环境。 Comm_size/Rank：获取进程数/线程 ID。 Send/Recv：发送/接收消息。
12		MPI 通信进阶	非阻塞通信的优缺点： 优点：避免性能资源浪费，不需要等待通信完成。 缺点：后续依赖通信消息的计算需要确保通信 成功，并需要额外判断通信完成的语句。 组通信操作及场景： 一对多（广播）：MPI_Bcast。场景：节点 0 有大量数据需要发送给其他所有节点。 多对一（归约）：MPI_Reduce。场景：每个节点有局部计算结果，需要汇总成一个全局结果。 同步：MPI_Barrier。场景：需要等待所有进程 计算完成后才进行下一步操作，如矩阵乘法。 MPI 消息中使用标签的原因： 区分不同进程发送的消息，避免消息混乱，确保正确处理每个消息。 MPI 消息传递中可能出现死锁的情况及避免方法： 当数据发送和接收都采用阻塞方式时，可能因 为互相等待而产生死锁。 避免死锁的方法是将发送或接收操作中的其中一个改为非阻塞方式，或者确保发送和接收 顺序的一致性。
13		MPI 分析比较	注意事项：捆绑发送接收操作需考虑消息大小、内存使用、通信重叠和错误处理等。 自定义数据结构：MPI 支持自定义数据结构，提高灵活性和通信效率。 MPI 与多线程：MPI 消息传递，多线程共享内存；MPI 通信复杂，多线程简单；MPI 可跨节点，多线程局限于单节点。 多层次并行架构：提高性能、缓解功耗问题、

			适应多样性任务需求。 MPI+多线程通信： MPI_THREAD_FUNNELED、 MPI_THREAD_SERIALIZED、 MPI_THREAD_MUTIPLE。
14	Lab3 实验代码编写调试		Source ~/.bashrc //更新环境变量 Module 加载 openmpi 尝试利用 lab2 的代码修改，无果。 尝试 MPI 函数。 周末完成代码编写
15	Lab3 实验报告撰写		收集实验数据，分析并给出报告
16		MapReduce	MapReduce 适于处理大规模数据的原因： 分布式处理：将任务分解为多个小任务，利用多台计算机的计算能力进行并行处理。 容错机制：自动处理节点故障，重新调度任务，保证高可靠性。 数据局部性：计算任务在数据存储节点附近执行，减少数据传输，提高效率。 可扩展性：通过增加计算节点线性扩展处理能力。 MapReduce 采用 key-value 数据结构的原因： 简化并行计算：通过键对数据进行分组，便于并行处理和聚合。 数据分片：将数据按键分割，分配到不同节点独立处理。 适用性：key-value 结构也适用于串行编程，如哈希表或关联数组。 Hadoop 封装的 MapReduce 并行计算功能： 分布式存储：使用 HDFS 存储大规模数据集，支持分布式存储和处理。 资源管理：YARN 管理集群资源分配和调度，提高资源利用率。 任务调度：JobTracker 和 TaskTracker 管理作业调度和任务执行。 容错性：处理节点故障，重新分配任务，确保作业连续性。 数据复制和备份：HDFS 数据块复制，提高数据可靠性。
17		并行计算的关键概念和设计方法	计算密集型与数据密集型算法的并行计算环境要求： 计算密集型：需要强大的计算资源，如多核处理器或 GPU，以及有效的任务调度和负载均衡

			机制。 数据密集型：需要高速的数据传输能力和大容量的存储空间，以及优化的数据分布和访问策略。 并行求前缀和算法与数组求和算法的关系： 数组求和：将数组分割，多处理器独立计算子数组和，再合并结果。 前缀和算法：计算每个元素的累积和，涉及处理器间的通信和数据交换。 并行程序设计方法： 直接并行化：将串行算法改造为并行算法，如数组求和算法的并行化。 从头设计：根据问题属性设计并行算法，如求前缀和算法。 借用算法：利用已知并行算法解决新问题，如3PCF 问题通过矩阵乘法解决。
18		PCAM 与并行计算任务调度	并行化（Parallelization）、通信（Communication）、调度（Scheduling）和映射（Mapping）。 任务的划分与分配、数据的分布与管理、通信开销、负载均衡、调度策略、资源管理、容错和异常处理、性能优化、任务调度算法的设计、动态环境下的调度、任务的优先级和公平性、能耗考虑
19	Lab4 实验代码编写		熟悉 OpenMP 的基础函数，在实验三代码的基础上分析并加入编译制导语句。
20	Lab4 数据分析以及是要报告编写		运行代码，选择合适的数据规模，绘制图像，分析实验结果。