华中科技大学

课程设计报告

题目： 并行编程原理与实践

院　　系：计算机科学与技术学院
姓　　名：
专业班级：
学　　号：
指导教师：
报告日期：

计算机科学与技术学院

目录

实验一

1. 实验目的与要求

1）实验目的

掌握在 pthread、OpenMP、MPI 环境下的并行排序算法设计。能够正确使用上述环境提供的 API 函数和接口，实现排序算法的并行化。最终了解并掌握并行优化思路和并行复杂度的计算，以及能够处理程序并行化可能存在的问题。

2）实验要求

（1）在串行环境下，自实现任意一个排序算法，能够将数组内数据从小到大排序；

（2）在 pthread 环境下，使用 pthread.h 文件内的相关函数，将（1）中的排序算法进行多线程执行，达到并行化效果；

（3）在 OpenMP 环境下，掌握 pragma 指令的使用，将（1）中的排序算法进行并行化处理；

（4）在 MPI 环境下，掌握 mpi.h 内的 6 个常用函数的使用，重点掌握进程间的信息传递过程，将（1）中的排序算法进行并行化处理。

2. 算法描述

（1）串行算法描述

第一步：初始化数组，将待排序数组存放在 a 中。

第二步：取 i 为 n-1，选取 a[i]为基准值，在 0~i-1 范围内寻找比 a[i]大的值中的最大值，将其与 a[i]进行交换。

第三步：再依次选取 i 为 n-2、n-3、...、1 为基准值，重复第二步操作。

第四步：输出排序完后的数组 a'。

该算法为常见的选择算法，时间复杂度容易计算为 $O(n^2)$。

（2）并行算法描述

第一步：初始化数组，将待排序数组存放入数组 a 中。

第二步：设使用处理机数为 p+1，其中 1 台处理机为主处理机，用于进行非并行程序处理过程，另外 p 台处理机用于并行排序。

第三步：将 a 数组分为 p 段，每段数组为[0...n/p]、[n/p+1...n*(2/p)]、...、[n*(p-1/p)+1...n-1]。对每段数组并行执行选择排序。

第四步：将每段排序完成的数组进行归并，即将相邻的奇数段与偶数段进行归并，如[0...n/p]与[n/p+1...n*(2/p)]进行归并，此时仍然可以进行并行归并处理。

第五步：重复第四步操作，直到所有段均实现归并成一个完整数组 a'，并输出该数组。

并行算法时间复杂度分析，该并行算法运行时间包括 3 部分，固有串行部分、并行处理部分、并行处理开销。其中固有串行部分仅仅为固定的输入输出，开销为 $O(n)$，并行处理部分包含两个环节：分解 p 段数组并行排序与 p 段数组归并组合，并行处理开销时间计算不易，主要包括并行资源分配开销、数据的传递时间开销等，主要与问题大小 n 与处理机数 p 相关，可使用函数 f 进行归纳，开销为 $O\big(f(n,p)\big)$。

对于并行处理部分两个环节，分解 p 段数组部分由于 p 个处理机分别处理 n/p 大小的数组，且采用选择排序，故处理时间为 $O((n/p)^2)$。而将排序完的数据进行归并，如图 1 所示分析。进行 $\log_2(p)$ 次归并，且每层归并是并行操作，那么从上至下的归并时间依次为 n*(2/p)、n*(4/p)、...、n*(p/p)，其处理时间和为(n/p)*(2p-2)，即 $O(2n(p-1)/p)$。

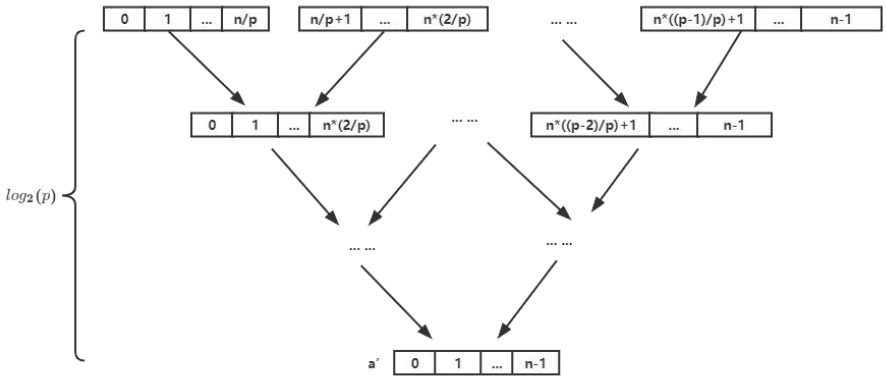

图 1 并行分段排序数组的归并分析

故总的并行排序时间复杂度为：

$$O(n)+O((n/p)^2)+O(2n(p-1)/p)+O\big(f(n,p)\big)$$

相比与串行而言，关键路径的复杂度并无数量级式的减少，但其处理时间减少了 p^2 的时间。

3. 实验方案（含开发与运行环境描述）

（1）在 pthread 环境下的实验方案

开发平台使用Linux 系统,与线程相关的函数都定义在 pthread.h 头文件中。运行平台为 educoder 平台，处理机数目为 4 台。

将串行算法实现为一个排序函数,传递值为排序数组的最低下标值和最高下标值。值得注意的是，使用 pthread_create 创建排序线程时，传递参数为单值，但实际使用需要传递 low 和 high 两个参数。故将参数设计为一个结构体 parameter，里面存放函数所需的两个参数。在创建线程前，创建结构体，赋入两个参数，调用 pthread_create 函数，具体实现为 pthread_create(&p1, NULL, sort, (void *)par1);。参数 par1 在 sort 函数体内进行解析获取。

开设两个线程进行并行排序，其中一个线程处理排序[0...n/2]，另一个线程处理排序[n/2+1...n-1]。使用 pthread_join 等待两个线程排序结束，最后进行两个数组的 merge 归并。

（2）在 OpenMP 环境下的实验方案

OpenMP 是一种用于共享内存并行系统的多线程程序设计方案，支持的编程语言包括 C、C++和 Fortran。开发平台使用 Linux 系统，使用 OpenMP 提供的 API 进行开发,使用 OpenMP 与 gcc 支持的框架编译运行。运行平台为 educoder 平台，处理机数目为 4 台。

使用 OpenMP 提供的 pragma 指令进行并行排序处理。具体使用#pragma omp parallel sections 指令，指令块内设置多个#pragma omp section 子语句块，多个子语句块进行并行执行。将数组平均分成 4 份,对这 4 份数组进行并行排序，排序完成后，进行两份两份的并行 merge 归并，最后再执行依次归并即可。

（3）在 MPI 环境下的实验方案

MPI 是一个跨语言的通讯协议，支持点对点和广播。其特色就在于数据的通信。开发平台使用 Linux 系统，通过包含 mpi.h 来获得 MPI 相关的函数声明与数据结构的定义，使用 mpicc 进行编译，mpirun 进行执行。运行平台为 educoder

平台，处理机数目为 4 台。

首先进行 MPI 并行环境初始化 MPI_Init(&argc, &argv)。接着使用 MPI_Comm_rank 获取当前进程的 id 值，保存在 myid 中。使用 MPI_Comm_size 获取当前处理机数。

主进程（进程号为 0）进行数据载入、并行数据分配和数据输出。首先将参数传递给其他进程，tag 值设置为传递目标进程号+10。需要注意的是，各进程间的变量值是相互独立的，只能通过数据传递与接收进行进程间的通信。此时除了传递 low 和 high 数值外，还需要传递数值的大小 n，以便其他进程知道后续将要获取的数组大小。接着将待排序数组传递给其他进程，tag 值设置为传递目标进程号+20。接着接收其他进程排序完成的数组。进行归并即可。

其他进程（设置为 2 个进程，进程号为 1 或 2）则进行参数接收和待排序数组接收，进行 sort 排序，将排序完成的数组段发送给主进程。需要说明的是，必须先接收参数再接收数组，因为 MPI 环境下的进程间数据相互独立，该进程并不知道 n 的大小为何值，故变量之间只能通过通信获取。由此，需要先获取数组的大小参数 n，才能顺利接收后续数组。

最后使用 MPI_Finalize()退出 MPI 系统。

4. 实验结果与分析

在上述串行环境与并行环境下进行测试，可以通过 educoder 上所有测试集。以下进行串并行的执行效率分析，以 OpenMP 环境下的测试为例。使用 clock 函数进行计时。

设置测试集为{5,4,3,2,1}。分别进行串行执行、2 分段并行执行和 4 分段并行执行。耗时结果采用多次运行取平均值的方式进行，如表 1 所示。

表 1 样例测试结果

测试方式	测试结果(clock)
串行	2
2 分段并行	1654
4 分段并行	2878

可以发现串行执行和并行执行之间存在较大差别，且并行算法所用时间周期明显大于串行算法。但不难看出，在上述算法分析中已知串行算法时间复杂度为$O(n^2)$，并行算法时间复杂度为$O(n)$+$O((n/p)^2)$+$O(2n(p-1)/p)$+$O\big(f(n,p)\big)$，其关键路径的时间复杂度都为$O(n^2)$，且并行算法由于不可避免的串行以及并行分配、信息通信的开销导致额外开销很大。当 n 较小时，并行并不会带来效率的提升。

实验二

1. 实验目的与要求

1）实验目的

实现杨辉三角的串行算法设计。能够正确使用 OpenMP、MPI 环境提供的 API 函数和接口，实现杨辉三角计算的并行化。最终了解并掌握并行优化思路和并行复杂度的计算，以及能够处理程序并行化可能存在的问题。

2）实验要求

（1）在串行环境下，实现杨辉三角的计算与输出；

（2）在 OpenMP 环境下，掌握 pragma 指令的使用，将（1）中的算法进行并行化处理；

（3）在 MPI 环境下，掌握 mpi.h 内的 6 个常用函数的使用，重点掌握进程间的信息传递过程，将（1）中的算法进行并行化处理。

2. 算法描述

（1）串行算法描述

第一步：设置一维数组 PT 用于顺序存放杨辉三角数值，采用从上至下、从左至右的方式存放。

第二步：输入带输出行数 n。设置 index 为当前待填 PT 的下标值。

第三步：对于 n 为 1 或 2 的情况，直接设置 PT[0]、PT[1]、PT[2]为 1。

第四步：对于第 i 层的杨辉三角数计算，置 PT[index]为 1，index 自增。

第五步:对于第 i 层的内部节点进行计算,使用公式 PT[index] = PT[index-i] + PT[index-i+1]。

第六步：重复第五步，直到计算完所有内部节点。置该层最后的节点 PT[index]为 1，index 自增。

第七步：重复第四步到第六步，直到生成完所有 n 层的杨辉三角数。

第八步：按格式输出杨辉三角值。

分析串行算法的时间复杂度，如图 2 所示，杨辉三角输出的时间主要包括两

个部分：遍历所有节点和计算内部节点。其中遍历一个节点的复杂度为 $O(1)$，

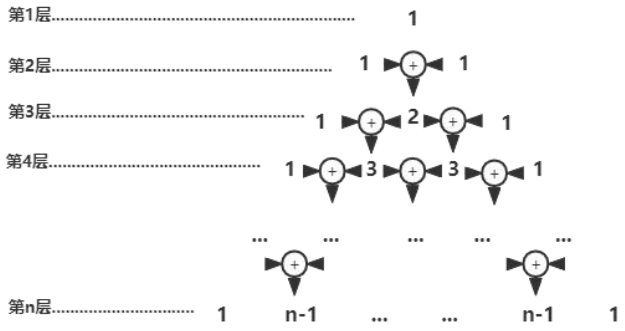

图 2 杨辉三角的串行时间复杂度分析

计算一个节点的复杂度为 $O(1)$。由此，计算完一个杨辉三角用时为 (1+n-2)*(n-2)/2，故时间复杂度为 $O(n^2)$。遍历所有边缘节点的用时为 2(n-1)+1，故时间复杂度为 $O(n)$。总的时间复杂度为 $O(n^2)$。

（2）并行算法描述

第一步：设置一维数组 PT 用于顺序存放杨辉三角数值，采用从上至下、从左至右的方式存放。

第二步：输入带输出行数 n。设置 index 为当前待填 PT 的下标值。

第三步：对于 n 为 1 或 2 的情况，直接设置 PT[0]、PT[1]、PT[2]为 1。

第四步：对于第 i 层的杨辉三角数计算，置 PT[index]为 1，index 自增。

第五步：对于第 i 层的内部节点进行计算，可以进行并行化运算，由于公式 PT[index] = PT[index-i] + PT[index-i+1]中的数值相互独立，在该层可以并行计算。

第六步：重复第五步，直到计算完所有内部节点。置该层最后的节点 PT[index]为 1，index 自增。

第七步：重复第四步到第六步，直到生成完所有 n 层的杨辉三角数。

第八步：按格式输出杨辉三角值。

由此，对杨辉三角内部节点的计算可以采用并行，如图 3 所示。图中计算使用虚线进行的运算为并行运算，计算时间复杂度时该层并行用时 $O(1)$。为此，通过并行可以优化计算的时间复杂度，此时计算处理时间为 n-2，时间复杂度为 $O(n)$。遍历边缘节点的复杂度不变为 $O(n)$。并行开销与问题大小 n 和处理机数 p

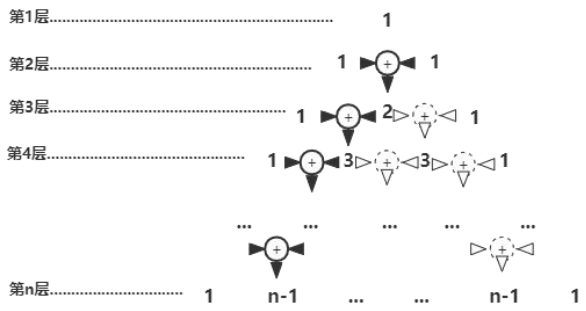

图 3 杨辉三角的并行时间复杂度分析

相关，可设为 f(n,p)。故总的时间复杂度为：

$$O(n)+O\big(f(n,p)\big)$$

可以明显看出，进行计算的并行化对整体的时间开销有很大优化。可知，杨辉三角算法时间复杂度的关键路径在于计算开销。

3. 实验方案（含开发与运行环境描述）

（1）在 OpenMP 环境下的实验方案

开发平台使用 Linux 系统，使用 OpenMP 提供的 API 进行开发，使用 OpenMP 与 gcc 支持的框架编译运行。运行平台为 educoder 平台，处理机数目为 4 台。

使用 OpenMP 提供的 pragma 指令进行计算并行处理。具体使用#pragma omp parallel for 指令。由于杨辉三角的每一层内部节点的计算中，实现公式为 PT[index] = PT[index-i] + PT[index-i+1]，只依赖于上一层数值，对于内部计算的 for 循环可以进行并行化处理。

（2）在 MPI 环境下的实验方案

开发平台使用 Linux 系统，通过包含 mpi.h 来获得 MPI 相关的函数声明与数据结构的定义，使用 mpicc 进行编译，mpirun 进行执行。运行平台为 educoder 平台，处理机数目为 4 台。

首先进行 MPI 并行环境初始化 MPI_Init(&argc, &argv)。接着使用 MPI_Comm_rank 获取当前进程的 id 值，保存在 myid 中。使用 MPI_Comm_size 获取当前处理机数。

主进程（进程号为 0）进行每层数据的赋值和加数的传递。当需要进行内部节点计算时，将两个加数保存至数组 addend 中，将其发送至对应的节点，对应

节点为与循环遍历 j 的线性相关函数，设置 tag 值为 j，方便后续接收时的赋值。值得注意的是，由于进程数量有限，故发送的目标进程号应该在 1~numprocs-1 之间，使用取模可以限制在其之间，具体实现为 j%(numprocs-1)+1。在接收计算完成的 sum 值时，其他进程设置好与 tag 值（与主进程发送时相等即可），将接收到的 sum 值赋给对应的 PT[index]即可。当所有计算结束后，一定要向所有的其他节点发送结束计算信息，设置 tag 值为 0。

其他进程的进程号为 1~numprocs-1。循环不断接收 0 号进程的任意 tag 的信息，具体 tag 值存放在结构 status 中。对 tag 值进行判断，当 tag 不为 0 时，表示计算还在继续，对传递来的加数进行加法计算，将计算结果 sum 传递给 0 号进程，tag 值为接收到信息的 tag 值；若 tag 为 0，表示 0 号进程发送了停止计算信息，于是进行 break，退出循环。

最后使用 MPI_Finalize()退出 MPI 系统。

4. 实验结果与分析

在上述串行环境与并行环境下进行测试，可以通过 educoder 上所有测试集。以下进行串并行的执行效率分析，使用 clock 函数进行计时。分别取 n 为 10 和 100 进行测试，耗时结果采用多次运行取平均值的方式进行，如表 2 所示。

表 2 样例测试结果

测试方式	测试结果 (clock)	测试方式	测试结果 (clock)
n = 10		n = 100	
串行	2	串行	26
OpenMP 并行	2023	OpenMP 并行	6752
MPI 并行	93	MPI 并行	3006

从上方实验结果可以看出，并行执行效率依然无法突破串行，原因可能在于 n 取值较小的缘故，测试受到 $O(f(n,p))$的影响较大。当 n 较大，且处理机数量较少时，分配到每个处理机的计算量依然需要串行等待，与并行无太大差别，并不能达到理想状态的∞处理机的并行分析结果。除此之外，由 MPI 并行结果可以看出，当 n 较大时，进程间通信频繁，导致进程间的通信开销增大。

实验小结

通过排序算法并行化实验和杨辉三角并行化实验，让我掌握了 pthread、OpenMP 和 MPI 环境下的相关函数使用。pthread 环境下，进行线程的并行化，使用 pthread_create 函数创建线程，使用 pthread_join 等待所有线程的结束。OpenMP 环境下，需要掌握 pragma 指令，其可在编译阶段自动进行优化，如循环优化。MPI 环境下，进行进程间的并行化，采取通信机制进行进程间数据的交流。

除了对以上具体环境和函数的熟悉和使用外，还掌握了程序的可并行化分析。对于排序算法的并行化，由于内部排序数之间关联性很强，很难实现处理级别的并行，故采取宏观并行。先将数组分段，各段间的排序相互独立故可并行操作，最后进行归并即可。对于杨辉三角的并行化，可以实现处理级别的并行，由于每层内部节点的运算是相互独立的，故可进行运算的并行化处理。

这两个实验，让我掌握了宏观并行与局部并行的方法，收获良多，对往后的学习有极大的帮助。此外，在进行实验分析时，发现实际结果与理论有所偏差，进行一步步分析发现，并行处理的额外开销问题不容忽视。通过串并行测试比较，了解了并行化处理的多个额外开销点，让我对并行编程原理有了更加深刻的记忆。