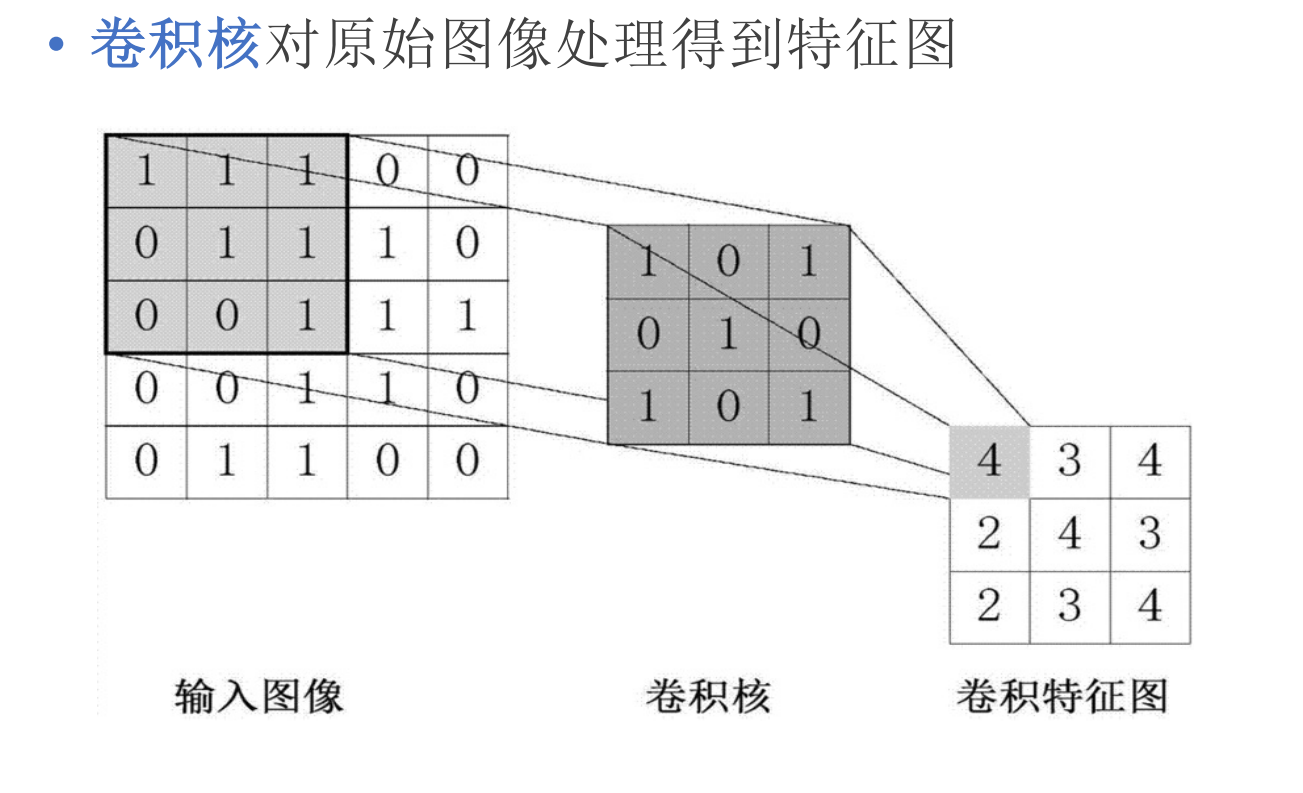
《并行计算》综合实验报告

姓名 刘恒星 学名 2022229044 完成时间 2023-5-5

1. 实验名称与内容

实验2名称：多线程计算卷积

实验内容：卷积是一种积分变换的数学方法，广泛应用于通信、物理、图像处理等领域。图像处理中，卷积操作就是卷积核（过滤器 / Filter）在原始图像中进行滑动得到特征图的过程，如图所示。



实验三：多进程计算卷积

采用 MPI 编程模型实现卷积计算。

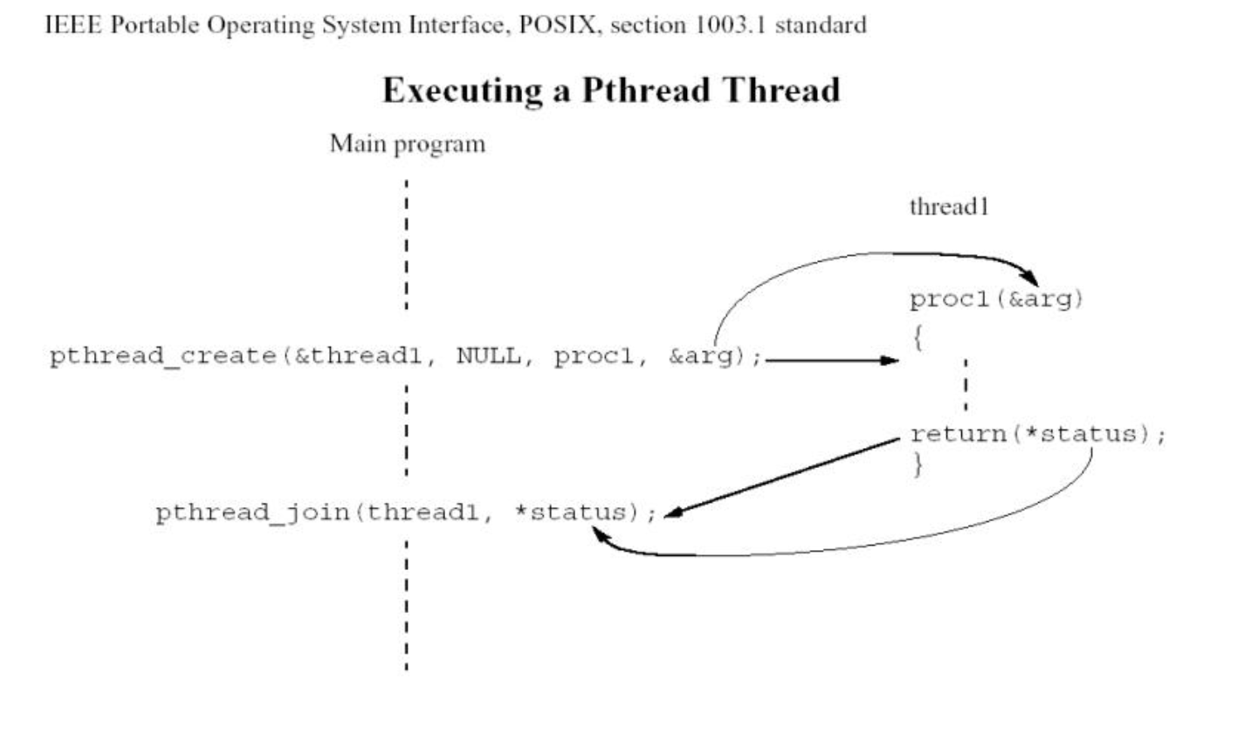
划分方法可参考课程中的 Jacobi 迭代，将原始图像划分成 p（进程数）个子块，每个进程处理一个子块，进行 N 次卷积计算，计算中每一个进程都要向相邻的进程发送数据，同时从相邻的进程接收数据

实验四：多进程计算卷积

本实验针对实验二问题，采用 MPI+OpenMP 编程模型实现卷积计算。节点间采用 MPI，节点内采用 OpenMP。需要制定多层划分策略。

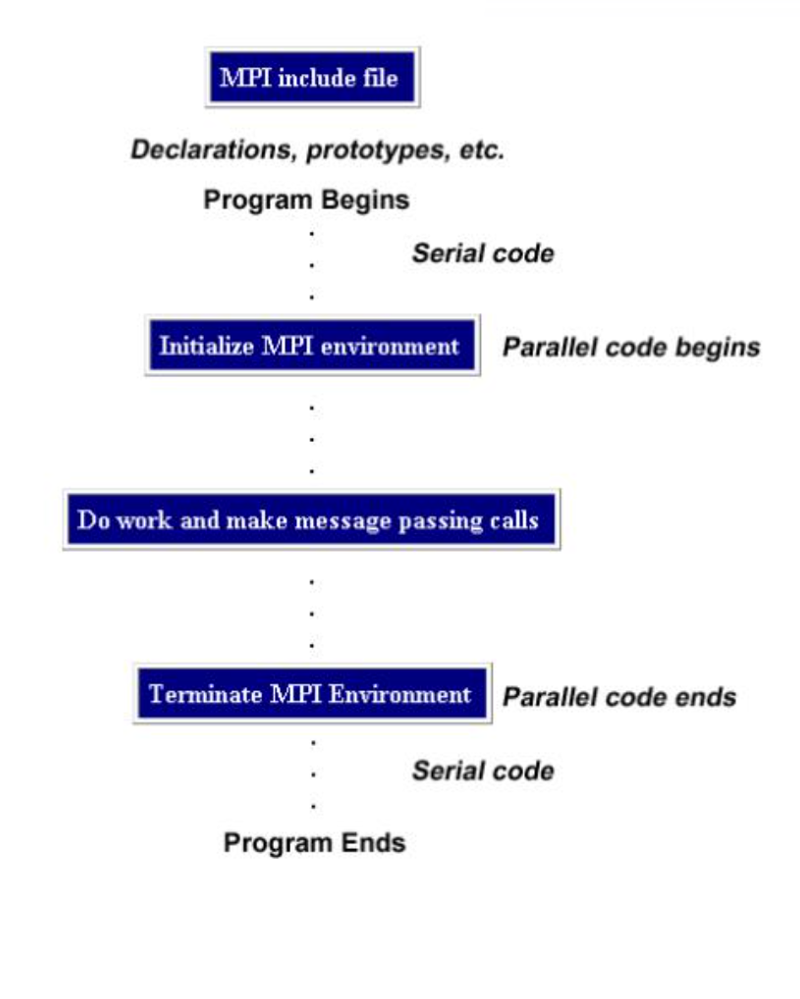
二、实验内容和对应的知识总结

实验二中需要我们使用pthread进行多线程的实验。pthread库是一种用于多线程编程的C语言库，它为程序员提供了创建、同步和管理线程的函数和数据类型。pthread库包含一组标准的线程操作函数，如创建线程、销毁线程、等待线程结束等，同时也提供了线程间同步和互斥机制的函数。通常使用pthread\_create创建线程，并指定线程运行函数，然后使用pthread\_join函数结束线程。



实验三需要使用MPI进行多进程并行。MPI是一种用于并行计算的标准化接口，它定义了一组函数和数据类型，用于实现在分布式内存系统中进行进程间通信和同步。MPI库可以在多个计算节点之间传递消息和数据，实现任务的划分、分配和协作。MPI库主要包括两部分：MPI标准和MPI实现。MPI标准定义了一组API函数和数据类型，例如MPI\_Send, MPI\_Recv等，用于实现进程间通信和同步。MPI实现是MPI库的具体实现，通常由MPI标准提供者或第三方厂商开发。MPI的核心操作包括进程初始化、进程终止、进程通信和同步等。在使用MPI库时，程序首先需要调用MPI\_Init函数进行进程初始化，并在程序结束时调用MPI\_Finalize函数进行进程终止。

在这次实验中，需要用到MPI\_Init来初始化并行环境，使用MPI\_Comm\_Size和MPI\_Comm\_Rank获取线程数和线程序号。因为每个线程有独立的数据域，所以我们在计算前和计算结束的时候需要进行进程通信，在计算的时候用MPI\_Bcast来广播矩阵信息，用MPI\_Sendrecv来发送和接受卷积运算需要的信息，用MPI\_Gather来汇总所有进程运算的结果。其中有MPI\_Barrier进行进程同步。



实验四中需要我们使用OpenMP+MPI进行多层划分的并行设计。OpenMP采用基于指令的并行模型，通过将并行代码嵌入到顺序代码中来实现并行化，从而提高程序的性能和效率。OpenMP主要包括三个部分：编译器指令、库函数和环境变量。编译器指令是OpenMP最重要的部分，它通过对源代码的特殊注释实现并行化。常见的OpenMP指令包括#pragma omp parallel、#pragma omp for、#pragma omp sections等，用于定义并行区域、循环并行以及任务并行等。库函数是OpenMP提供的一组库函数，用于实现线程同步、互斥等操作。环境变量则是OpenMP库提供的一些运行时参数，可以调整并行执行的策略。

在实验四中，使用MPI多进程的分割子矩阵计算卷积，使用OpenMP制导语句多线程并行加速卷积操作。

三、横向对比

实验均采用2048\*2048大小的矩阵，3\*3的卷积核，迭代计算10次，结果如下

单进程多线程结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 进程数 | 线程数 | 运行时间 | 加速比 | 效率 |
| 1 | 2 | 3.25352 | 1.475681723 | 0.737840862 |
| 1 | 4 | 2.33343 | 2.057554758 | 0.51438869 |
| 1 | 8 | 1.81024 | 2.65222291 | 0.331527864 |
| 1 | 16 | 1.66132 | 2.889967014 | 0.180622938 |
| 1 | 32 | 1.55514 | 3.087284746 | 0.096477648 |

多进程单线程实验结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 进程数 | 线程数 | 运行时间 | 加速比 | 效率 |
| 2 | 1 | 2.79078 | 1.720364916 | 0.860182458 |
| 4 | 1 | 1.88323 | 2.549428376 | 0.637357094 |
| 8 | 1 | 1.3298 | 3.61043766 | 0.451304707 |
| 16 | 1 | 1.32345 | 3.627760777 | 0.226735049 |
| 32 | 1 | 1.31011 | 3.664699911 | 0.114521872 |

多进程多线程实验结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 进程数 | 线程数 | 运行时间 | 加速比 | 效率 |
| 2 | 2 | 1.94328 | 2.470647565 | 0.617661891 |
| 4 | 4 | 1.24331 | 3.861595258 | 0.241349704 |
| 8 | 8 | 1.01654 | 4.723040903 | 0.073797514 |

可以看出，当处理器（线程数\*进程数）数量相同时，多进程单线程结果优于多线程单进程结果，而多进程多线程结果优于多进程单线程结果，运行时间，加速比，效率都比前两种要好。三种方案的相同点就是，随着处理器数量增多，运行时间先减小后增加，加速比先增加后减小，效率变低，是因为随着处理器增多，会引入额外的操作，比如进程之间的通信，线程进程的创建和销毁等等，这些操作会带来额外的时间代价。