西北工业大学

实 验 报 告

（2020~2021 学年秋季学期）

课程名称 《并行计算》

姓名 严愉程

学号 2017300138

学院 教育实验学院

班级 HC001706

专业 计算机科学与技术

目录

《并行计算》实验报告

严愉程

2020/12/30

1 实验目的

1. 熟悉 MPI 编程环境，掌握 MPI 编程基本函数及 MPI 的相关通信函数用法，掌握 MPI 的主从模式及对等模式编程；

2. 熟悉 OpenMP 编程环境，初步掌握基于 OpenMP 的多线程应用程序开发，掌握 OpenMP 相关函数以及数据作用域机制、多线程同步机制等。

2 实验环境

ubuntu18.04，inteli7-7700HK 4 核 8 线程，gcc8.2，OpenMPI 4.2.0。使用 CMake 构建。

3 实验内容

分别进行以下实验：1. 用 Monte-Carlo 随机积分算法估算 π 值的并行算法；2.Floyd 算法的并行化；3.N 皇后问题并行算法

3.1 Monte-Carlo 随机积分算法估算 π 值的并行算法

3.1.1 算法原理

算法的原理比较简单，在第一象限上的单位正方形内随机放点，落在单位圆内的点的个数 / 点的总个数 = π / 4。

多个线程并行的进行多次投掷，然后 reduce 到一个线程上。

3.1.2 实验步骤

（1）写 CMakelists，配置编译环境

（2）编写代码主要有几个模块： 1. 随机数生成模块，需要阻止每个线程生成相同的随机数序列

2. 为了保证运算精度我用了大数运算的库（受限于电脑性能，我最后没到达那个精度）

3. 主线程进行 reduce 求和

3.1.3 运行结果

运行结果如图1所示。

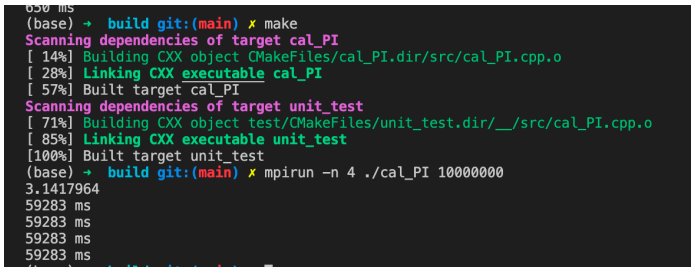

图 1: 计算 π 值算法的运行结果图

3.1.4 算法分析

表 1: 并行计算 π 运行结果

模拟次数	线程数为 1 的运行时间	线程为 2		线程为 4	
		运行时间	加速比	运行时间	加速比
1000000	12287 ms	7508 ms	1.636	5825 ms	2.109
5000000	69005 ms	40313 ms	1.711	30667 mss	2.250

并行性能分析 算法随不同 n 和 p 的值性能如表3所示。随着数据量的增加，加速比有所提高。

算法可扩展性分析 算法对处理器上限没有要求，随着处理器的个数增多，可以扩大问题的计算范围 π 的精度也会有所提高。

3.2 Floyd 并行算法

3.2.1 算法原理

```
for(int k = 0; k < mat_size; k++){
    for(int i = 0; i < mat_size; i++){
        for(int j = 0; j < mat_size; j++){
            mat[i][j] = min(mat[i][j], mat[i][k] + mat[k][j]);
        }
    }
}
```

图 2: floyd 串行代码

floyd 串行的原理 串行版代码比较简单，见图2，采用了动态规划的思想。

最外层的 k 就是考虑以节点 k 为中转点，尝试节点 i 到节点 j 通过节点 k 的中转，能否更近。

第 k 次迭代的含义是，在尝试过前面 k-1 个节点作为中转点的基础上，尝试节点 k 作为中转点，求任意两点之间的最短路。迭代最后会最后尝试遍所有中转点。

并行的设计思路 根据四步设计法：

（1）划分：算法对 $mat[i][j] = \min(mat[i][j], mat[i][k] + mat[k][j])$ 进行了 n^3 次操作。可以把矩阵 $mat[][]$ 上的 n^2 个元素作为一个原子任务。

（2）通信：每个任务在第 k 次迭代的时候，需要知道 $mat[i][k]$ 和 $mat[k][j]$。

（3）聚合：每个原子任务都有相似的计算时间，我们在聚合的时候考虑通讯量。

读取矩阵的时候是按行读取的，自然采用按行聚合的方式。而且按行聚合不需要通讯 $mat[i][k]$，只需要通讯 $mat[k][j]$，即，在 k 次迭代之前，通讯 k 行。

（4）映射：按行，分组映射到每个处理器即可。最后一个处理器来处理余下的部分，并进行矩阵的读取与发送，作为 master 线程。

3.2.2 实验步骤

（1）写 CMakelists.txt。准备编译环境。

（2）编写并行的 floyd 代码。首先进行矩阵的读取，然后计算，然后 master 线程，通过 send，recv 的形式控制线程依次写结果。

（3）写验证用的串行 floyd 代码。验证结果正确。

第一版的算法，在通讯的手段上比较粗糙，随着最外层变量 k 的迭代过程中，每一次矩阵 mat[][] 更新都进行一次矩阵的同步，即每个线程都广播一次自己管辖的矩阵区域，如图3。在 $mat_size = 4000$ 的节点数下，多线程的运行时间和单线程的差不多，虽然能够得出正确的结果，但加速比不理想。

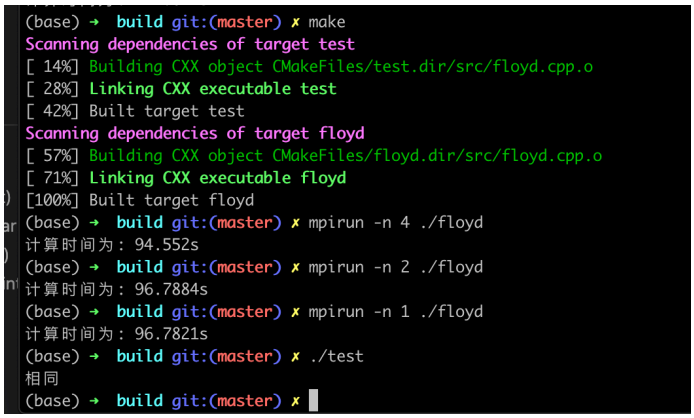

图 3: floyd 代码第一版的运行结果

考虑从通信量上的进行优化。在操作 $mat[i][j] = \min(mat[i][k] + mat[k][j])$ 中，最外层循环 k 迭代的时候，$mat[i][k]$ 的数据永远在本线程管辖的区域内，不需要同步。而 $mat[k][j]$ 会随着 k 的迭代，出现其他线程管辖的区域或者本线程管辖的区域是不确定的。

所以每次 k 迭代的时候，各个线程需要的是 k 行数据，迭代 k 次之前要广播 k 行。

需要注意的是，每次迭代的时候，实际上每个线程管辖的区域都更新了，最后还需要把这些更新进行同步，因为题目要求的是线程依次按照顺序分别写入，就不需要进行同步（在多节点的系统上注意需要进行同步）。

改进之后的运行结果如图4所示。

3.2.3 运行结果

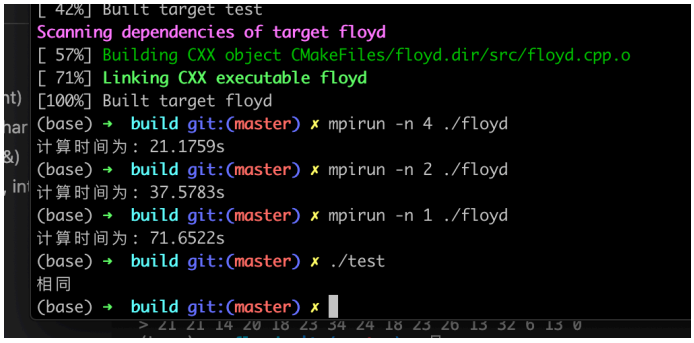

图 4: floyd 优化版命令行运行结果

并行版的 floyd 算法与串行版的 floyd 算法运行结果相同，得出：并行版的 floyd 算法运行正确。且在节点个数为 4000 的问题规模下，线程个数为 4 时加速比 3.384。

3.2.4 算法分析

并行性能分析 不同线程个数和不同问题规模下，该算法的并行性能，如表2所示。

表 2: floyd 运行结果

矩阵的大小	线程数为 1 的运行时间	线程为 2		线程为 1	
		运行时间	加速比	运行时间	加速比
4000	71.6522s	37.5783s	1.907	21.1759s	3.384
2000	9.36538s	5.47045s	1.711	3.69849s	2.536

算法可扩展性分析 该算法是按照行来进行划分的，可以使用的处理器个数的上限为行的数量。如果有充足的处理器，可以考虑棋盘式划分的方法。对于一个集群系统，在节点的内部还可以使用 openmp 进行并行。openmp 相对于 mpi 更加时候统一内存架构，且在一定的情况下编程相对方便。

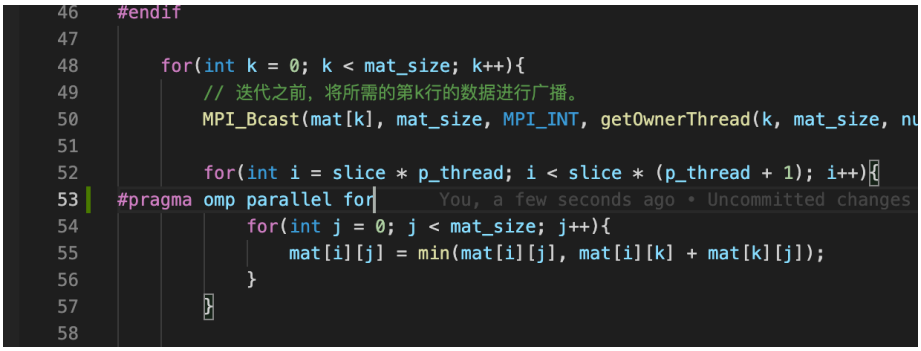

图 5: 使用 openmp+MPI 混合编程

3.3 超快速并行算法

3.3.1 算法原理

我这里举一个 8 线程情况时候的例子，就比较容易理解算法原理。

（1）首先将数组平均分给 8 个线程。每个线程分别进行快速排序。这样就得到了 8 个有序子数组，下面分别用 0, 1, 2, ..., 7 来表示

（2）此时（0，1，2，3，4，5，6，7）在同一线程组中，0 号线程将自己的中位数 pivot 广播。

（3）每个线程根据 0 号线程广播的 pivot，将本线程中的子数组进行划分，分为大的一部分 upper 和小的一部分 lower。

（4）0 和 4，1 和 5，2 和 6，3 和 7 交换两个部分。比如，0 号线程拿 0 和 4 中的 lower，进行归并。4 号线程拿 0 和 4 中的 upper，进行归并

（5）结果就有，（0，1，2，3）< (4，5，6，7) 中的元素，拆分成了两个线程组。对每个线程组重复步骤（2）到步骤（5）的操作

（6）结果依次有（0，1）< (2，3) < （4，5）< （6，7）

（7）最后有（0）< （1）< （2）< （3）< （4）< （5）< （6）< （7）

3.3.2 实验步骤

按照上面的算法原理编写代码实现即可。

3.3.3 运行结果

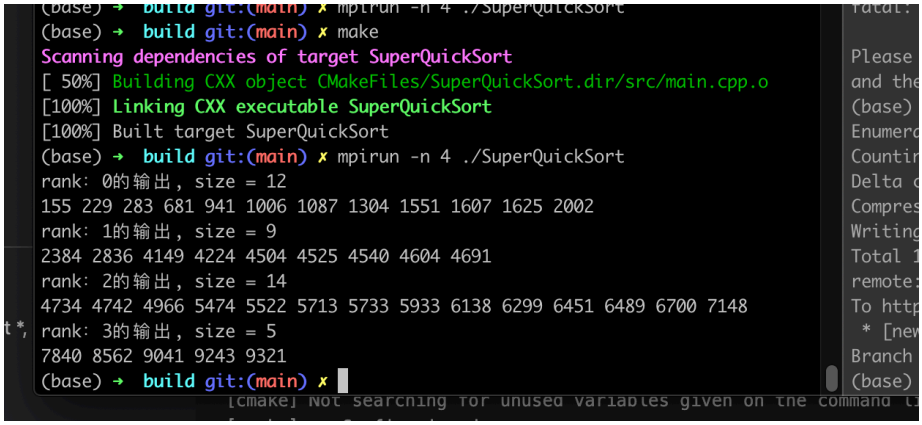

图 6: 超快速并行排序命令行运行结果图

3.3.4 算法分析

并行性能分析 很遗憾没能写出并行性能很好的算法，初步怀疑是 IO 数据量大的问题，或者我的代码没有优化好。

算法的可扩展性分析 线程数的扩展能力较强。

表 3: 并行计算 π 运行结果

模拟次数	线程数为 1 的运行时间	线程为 2		线程为 4	
		运行时间	加速比	运行时间	加速比
1e6	0.28186s	0.267679s	1.053	0.259043s	1.088
1e7	2.76637s	3.25229s	0.852	3.14133s	1.035

3.4 N 皇后问题并行算法

3.4.1 算法原理

算法比较简单，一个典型的 DFS 问题，可以在搜索的每个分之上进行并行。这里我使用 openmp 进行编程。

3.4.2 运行结果

```
(base) ➜  build git:(master) ✗ ./N-queen
棋盘的大小为:12
线程个数为1; 时间为:1128 ms
线程个数为2; 时间为:540 ms
线程个数为3; 时间为:351 ms
线程个数为4; 时间为:278 ms
线程个数为5; 时间为:314 ms
线程个数为6; 时间为:258 ms
线程个数为7; 时间为:264 ms
线程个数为8; 时间为:264 ms
```

图 7: N 皇后程序运行截图

3.4.3 算法分析

分析不同 n 值、P 值以及不同有序度时算法的运行时间，进行算法并行性能和可扩展性分析。(*)

表 4: 棋盘大小为 12 时的性能

线程个数	计算所用时间 (ms)	加速比
1	1128ms	1
2	540ms	2.08
3	351ms	3.21
4	278ms	4.05
5	314ms	3.59
6	258ms	4.37
7	264ms	4.27
8	264ms	4.27

并行性能分析

算法的可扩展性分析 目前算法扩展的线程上限数量为棋盘的大小。

4 实验总结

通过思考并行算法，以及编程实现，我在复习课堂上的基础知识的同时，也提升了我的动手能力。让我进一步掌握了，OpenMP、MPI 这两个并行计算的工具，以及混合编程的优势。