挑战性题目DSCT401：全源最短路径Floyd算法的并行实现

问题描述：*n*个顶点的有向图采用邻接矩阵进行储存，dist和path分别表示邻接矩阵和路径矩阵，初始化过程伪代码在下方给出。任意输入一个顶点规模正整数*n*，正确实现Floyd算法的基础上返回算法运行时间。例如，输入512。

初始化过程: 对i行j列的dist[]数组进行初始化的精神是，dist[i\*n+j]=rand()%100，若i==j则dist[i\*n+j]=0 。

思路分析[[1]](#footnote-1)与算法描述[[2]](#footnote-2)：

本人实现了基于MSMPI的分块划分的Floyd并行加速算法，并在不同规模的测试数据集上对比了串行与并行(进程数分别为4、8、12)实现所需的时间，并给出了加速比。

针对传统串行算法的并行设计，主要从以下两个角度来进行：一个是功能分解，即将问题按功能分解为若干个可以并行解决的子问题；一个是域分解，即将问题处理的对象分解为若干个较小规模的区域，然后对他们进行并行地求解。在对传统Floyd算法进行分析后，容易发现其模块构成为：一个针对比较与赋值语句进行的三重循环。这三重循环是基于动态规划思想设计而来的，总体而言已经较难以进行进一步的优化，除非能够对这些语句进行进一步的划分。但是我们却可以很容易的发现，将有向图的邻接方阵进行模块划分，分解为不同的子任务的方式似乎更容易实现。

一般而言，传统的并行任务划分可分解为按行划分、按列划分与按块划分。针对Floyd处理的方阵，易知按行与按列划分的事实上是等效的；从直觉出发，在可以忽略任务间通信的条件下，易知子任务并行数越多，原问题的解决效率应该越高。因此，按块划分的方式应该是当问题达到一定规模后的最佳解决方案。因此本人在同时实现了基于MSMPI的按行与按块并行算法的基础上，着重探讨了按块分解的优势。

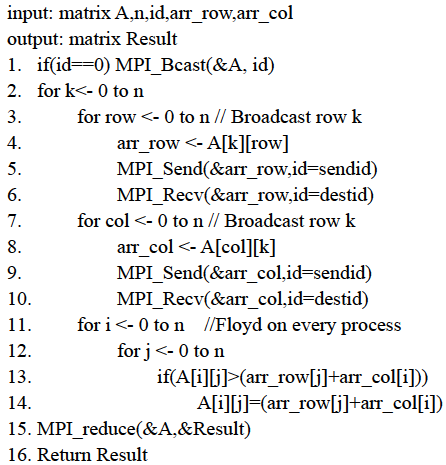
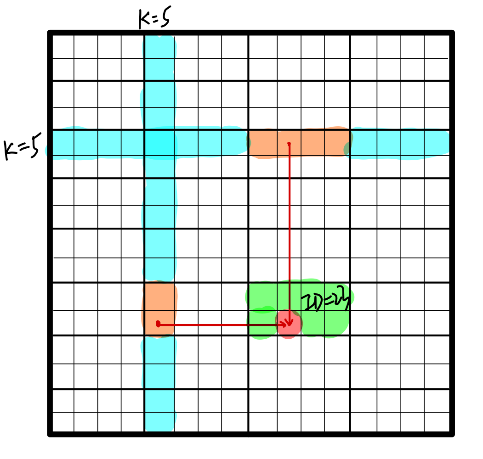
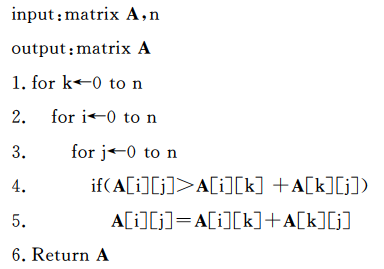


图 1 Floyd算法的串行、并行伪代码与示意图(其中k=5，处理机ID=23，划分方式为矩形棋盘划分)

接下来我们从传统Floyd算法的实现中寻找可以并行的原理。图1左图为传统Floyd算法的伪代码，右图为其执行的过程示意图。容易发现如下两个重要结论：

1. 在动态规划至第k层时，的更新(行4、行5)的只与的第行、第列有关；
2. 在动态规划至第k层时，的第行、第列的元素均不会产生更新。

结论1是显然的，从图1中行5即可看出；结论2也是易见的，因为：

因此，我们可以着眼于的第行、第列的元素：在动态规划至第k层时，将第行、第列的元素(如图中蓝色区域)广播至全局所有的处理机。而事实上从代码行5可以看出，每个处理机(如图中绿色区域所示的ID=23号处理机)只需要k行k列元素中的一部分即可完成更新，例如图中红色处的数据只需要箭头所指向的两个数据即可。因此，每个处理机事实上只需要图中的橙色部分，这在论证了按块分解的合理性的基础上，进一步增加了子任务的分解规模，可使得算法在处理大规模矩阵时的性能得到更大的提升。

具体实验及结果如下(表中数据单位为秒，N代表有向图中的节点规模，P代表进程数目)。实验时我们选用了进程数分别为4、8、12的并行算法与串行算法进行对比，并将处理问题的数据规模设定在了。经多次试验取平均值，实验结果稳定，具体如下表1所示。

表1 不同进程数的并行与串行Floyd算法性能对比

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | N=24 | N=144 | N=576 | N=1152 | N=2304 | N=4608 | N=9216 | N=10368 |
| 串行p=1 | 0.001s | 0.007s | 0.602s | 4.359s | 32.643s | 263.287s | 2034.032s | 3177.711s |
| 并行p=4 | 2.038s | 1.137s | 1.217s | 2.364s | 11.889s | 82.781s | 625.462s | 881.350s |
| 并行p=8 | 6.295s | 2.829s | 3.175s | 5.179s | 11.340s | 67.153s | 479.914s | 677.888s |
| 并行p=12 | 5.986s | 6.710s | 8.118s | 7.498s | 14.338s | 69.341s | 451.456s | 564.722s |
| 最大加速比 | 0.0005 | 0.006 | 0.495 | 1.844 | 2.879 | 3.921 | 4.505 | 5.627 |

从上表可以得出如下结论：

1. 并行设计可以有效提升算法的运行速度，且当节点规模越大时，加速比越高；
2. 进程数并不是越多，加速效果越好。进程数的增多意味着进程间通信开销的增加，当节点规模大时，CPU密集型的进程的CPU利用率才能提高，此时进程数越多加速比才越高；否则进程间通信将占据绝大部分时间，且进程数越多效率越低，如表中N<规模时所示；
3. 并行算法的运行时间并不一定是随着节点规模递增的。当节点规模小时，运算时间可能随着规模不规律的摆动。这可能也是由于MPI接口进行进程间通信时内部的机制导致的，如进程的阻塞与挂起等；
4. 可以大致估计运行时间。给定并行模式与进程数，若节点规模为时的时间为，则节点规模为 的运算时间可大致估计为：。

值得注意的是，由于本实验的运行是在本机(Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz，内存8.00GB)中运行，当进程数较大时，可能由于在运行进程时同时运行了其他应用程序，从而导致运行时间变长。故运行时若不使用外设服务器，最佳的做法是保持运行环境的清洁。

下面分析并行算法的时间复杂度。前两个用于构造消息列表并进行通信的内层循环的时间复杂度为，后一个用于执行每一个处理机的Floyd算法的内层循环的复杂度为；加之考虑最外层循环，故并行算法的总时间复杂度为：。因此，相比于串行算法，加速比为。故当进程数取时，可以使加速比最大。以N=16为例，则当p=16时取得最大速率；但事实上本机一般取不到这么大的进程数，故无法取得最大速率。

1. 算法思路不但要给出解题的算法内涵，还应该分析该算法的时间复杂度。此页背面也可以答题，但不应续页。 [↑](#footnote-ref-1)
2. 代码的写作和测试建议使用GCC、G++等通用C/C++编译器进行编译以利于跨平台的性能测试。作业纸质版本和测试的可执行文件请提交给主管助教。测试样例命令格式为：DSCT401\_2018270103012.exe 512。 [↑](#footnote-ref-2)