**实验3**

学号 姓名

**实验题目:**

**利用MPI实现Floyd算法**

**实验环境(操作系统,编译器,硬件配置等):**

**本次实验运行在NVIDIA DGX（本次实验只使用了CPU：Intel Xeon E5-2698 v4 @ 2.2GHz, 20 cores/40 threads）硬件平台的ubuntu 16.04 RTS系统上，采用了支持C++11标准的gcc 7.3编译器来进行编译工作和cmake 3.9来辅助编译（只是为了程序写起来比较简单）。**

**算法设计与分析(写出解题思路和实现步骤)**

**Floyd算法基于动态规划技术而设计，其有三层循环，从外至内分别是中间节点，起始节点与目标节点。算法通过这样的动态规划方式，不断松弛f[i][j]为更小的f[i][k] + f[k][j]。由于一条最短路径中必不包括环，因此任一条最短路的顶点数都不超过这个图有的顶点数。每遍历一个中间顶点，可以认为在该次遍历结束后，长度小于等于k的最短路都得到了生成，在遍历结束之后，图中的所有最短路自然就得到了生成。**

**本算法基于很简单的并行思路来设计：在每一步k中，对不同的起始顶点i进行f[i][j]的并行计算，在该步结束后进行通信统一结果。**

**核心代码(写出算法实现的关键部分,如核心的循环等)**

**具体的代码都可以在https://github.com/thoh-testarossa/pc-exp上找到**

**Floyd算法的原型仅仅是i循环变成了全顶点循环而已。事实上在mpirun -np 1时，可以简单地知道该算法就是简单的floyd算法**

//Floyd-Warshall  
**for**(**int** k = 0; k < graphSize; k++)  
{  
 //Parallel computation for sor vertex  
 **int** my\_startFrom = (myid \* graphSize) / numprocs, my\_endTo = ((myid + 1) \* graphSize) / numprocs;  
 **for**(**int** i = my\_startFrom; i < my\_endTo; i++)  
 {  
 **if**(graph[edge\_No(graphSize, i, k)] != NOT\_CONNECTED)  
 {  
 **for**(**int** j = 0; j < graphSize; j++)  
 {  
 **if**(graph[edge\_No(graphSize, k, j)] != NOT\_CONNECTED)  
 {  
 **if**(graph[edge\_No(graphSize, i, j)] == NOT\_CONNECTED || graph[edge\_No(graphSize, i, j)] > graph[edge\_No(graphSize, i, k)] + graph[edge\_No(graphSize, k, j)])  
 graph[edge\_No(graphSize, i, j)] = graph[edge\_No(graphSize, i, k)] + graph[edge\_No(graphSize, k, j)];  
 }  
 }  
 }  
 }  
  
 //Combine the result together for next iteration  
 **for**(**int** p = 0; p < numprocs; p++)  
 {  
 **int** bcast\_startFrom = ((p \* graphSize) / numprocs) \* graphSize;  
 **int** bcast\_endTo = (((p + 1) \* graphSize) / numprocs) \* graphSize;  
 MPI\_Bcast(&graph[bcast\_startFrom], bcast\_endTo - bcast\_startFrom, MPI\_INT, p, MPI\_COMM\_WORLD);  
 }  
}

**实验结果:**

**运行时间**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 顶点数\线程 | 1 | 2 | 4 | 8 | 16 | 24 | 32 |
| 2000 | **132.426** | **71.413** | **44.638** | **36.670** | **28.822** | **38.224** | **39.856** |

**加速比**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 顶点数\线程 | 1 | 2 | 4 | 8 | 16 | 24 | 32 |
| 2000 | **1** | **1.854** | **2.967** | **3.611** | **4.595** | **3.464** | **3.323** |

**以上的时间与加速比都是排序部分的时间统计。IO时间通过对比实验排除了。**

**分析与总结**

**该算法在核数增加时，通信次数也是线性的增长的。当达到一定程度的并行度时反而出现了负优化。这说明用这种简单优化方法时需要结合数据规模来考虑具体的并行度**