Floyd-Warshall 算法的并行实现
上机实验报告

匡亚明学院 211240021 田铭扬

一 背景

Floyd-Warshall 算法（以下简称为“Floyd 算法”）是一个经典的多源最短路径算法，其中运用了动态规划的思想。问题如下：

> 设有编号为 0,1,…,n-1 的 n 个地点。已知矩阵 A=(a_{ij})(0≤i,j<n)，其中 a_{ij} 表示从 i 地直接走到 j 地所需的时间。试求出矩阵 B=(b_{ij})，b_{ij} 表示从 i 地（可以经由其它地点）走到 j 地所需的最少时间。

设 D[k][i][j] 表示只允许经过 0, 1, …, k-1 号地点时，从 i 地到 j 地的最少时间。则可列出如下的状态转移方程：

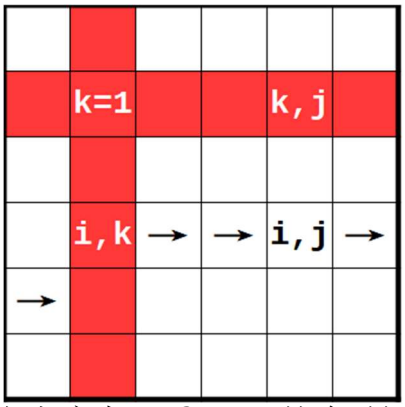
$$D[k+1][i][j]=\begin{cases} D[k][i][j] & \text{若 } i=k \text{ 或 } j=k \\ \min(D[k][i][j], D[k][i][k]+D[k][k][j]) & \text{否则} \end{cases}$$

初值为 $D[0][i][j]=a_{ij}$; 计算结果为$b_{ij}=D[n][i][j]$.

即到两点间经过 0, 1, …, k 号点的最少时间，等于经过 0, 1, …, k-1 号点的最少时间、或中转 k 点（途中可能经过 0, 1, …, k-1 号点）的最少时间中较小的。

注意到在计算第 k+1 层时，只会用矩阵 D[k]中第 k 行和第 k 列的数据，而这些数据在第 k+1 层并不会被改动。因此可以压缩存储，核心代码如下：

```
for(int k=0;k<n;k++)
  for(int i=0;i<n;i++){
    if(i==k) continue;
    for(int j=0;j<n;j++){
      if(j==k) continue;
      if(D[i][j]>D[i][k]+D[k][j])
        D[i][j]=D[i][k]+D[k][j];
    }
  }
```

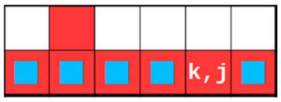


Floyd 算法的空间复杂度只有 $O(n^2)$，但是时间复杂度为 $O(n^3)$ ——这个时间复杂度并不令人满意，因此对它进行并行化是一件值得尝试的事情。

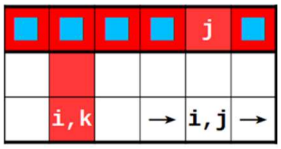
二 并行设计思路

并行算法的核心思路，是对上一部分的代码中的矩阵 D 进行分块，将各块交由不同的处理器进行计算。具体而言，则有按行分块（若所使用的编程语言是列优先存储的，应改为按列分块，此种方法以下简称为“分行”）和按行列分块（以下简称为“分块”）2 种实现方法。

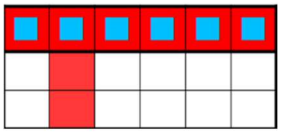
如上文所述，在最外层进行第 k 次循环时，需要用到原矩阵第 k 行和第 k 列的数据：



1) 对于“分行”的方法，如右图所示，每个处理器自己都拥有第 k 列的数据，而第 k 行的数据则需要由负责这一行处理器传递给所有其它处理器；

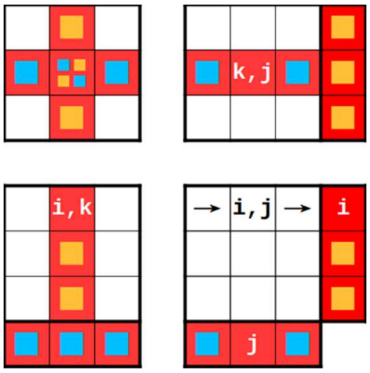


2) 对于“分块”的方法，如右下图所示，第 k 行/列的数据都需由相应处理器传给同列/行的其它处理器。



以分块方法为例，代码的整体结构没有发生变化：

```
for(int k=0;k<n;k++){
  /*负责区域含总第 k 列的各处理器发送数据,存储到数组 B；负责区域含总第 k 行的各进程发送数据，所有进程存储到数组 C*/
  for(int i=0;i<I;i++)
    if(i+iStart==k) continue;
    for(int j=0;j<J;j++){
      if(j+jStart==k) continue;
      if(D[i][j]>B[i]+C[j])
        D[i][j]=B[i]+C[j];
}}}
```



在应用分块方法编程时，还有两处细节可以留意：

1) 将同一列、同一行的所有处理器分别编进一个通讯器，以便使用 Bcast 方法；

2) 初始数据分发时，应由 0 号处理器先分发给每行的首个处理器，再由它们将数据分发给同一行的处理器，以加快数据分发速度；最后收集时同理。

下面分析 2 种方法的时间复杂度。设共 N 个地点，M^2 个处理器，并设函数 f(X)表示传递 X 个 double 变量所需的时间复杂度，有：

$$\text{分行：}O(N*N*\frac{N}{M^2}+N*\log(N\text{^}2)*f(N))$$

$$\text{分块：}O(N*\left(\frac{N}{M}\right)^2+N*\left(2*\log(N)\right)*f\left(\frac{N}{M}\right))$$

对比发现，分块方法相当于把消息传递的过程也进行了并行化，相比分行方法节省了一定的时间；但是考虑到分块方法的初始数据分发和结果收集比分行方法复杂许多，推测只有当矩阵规模比较大、处理器数量比较多时，分块方法才能体现出时间的优势；而计算规模较小时，分行方法或许能够略胜一筹。

由于不清楚 f 的具体表达式，要在 2 种并行方法以及串行程序之间进行比较，需要实验数据的支撑。

三 实验结果

实验结果在下页的表格中展示。数据矩阵的规模为 2500 阶，使用学校机房的设备（不清楚处理器型号等具体信息），使用 gcc 11.4.编译器，开启-O2 优化选项。每个时间数据均为 3 次计算的平均值。

分行	(串行)	2	4	6	8	9	10	12
	54.71s	29.16s	14.62s	10.51s	9.854s	15.09s	13.84s	12.74s
分块	/	/	**4**=2*2	**6**=2*3	**8**=2*4	**9**=3*3	**10**=2*5	**12**=3*4
	/	/	13.74s	10.31s	9.686s	15.23s	13.89s	12.85s

从表中可以看到，与串行相比，2 种并行方法的速度都有极大的改观。

一个值得注意的现象是，在处理器数量由 8 变成 9 时，2 种方法的速度都显著变慢，却又随处理器数量增加而逐渐变快（但仍比 8 处理器的速度更慢）。这种现象的原因很显然，却并不容易在事前就想到：算法中大量地使用了 Bcast 函数，而假设通讯器中有 k 个进程，Bcast 的时间复杂度有系数 $\lceil \log_2(k) \rceil$。所以当 k 从 8 变为 9（分行方法）或从 2 变为 3（分块方法）时，计算时间的“断崖式”增加就并不奇怪了。遗憾的是，由于设备条件限制，笔者无法使用 16 及 17 个处理器进行计算，因而无法检验上述分析的正确性。

总之，在机房的设备上，使用 8 个处理器是用时最短的。且在这个计算规模下，基本无法区分分行方法与分列方法的速度区别；鉴于分行方法的代码更简单（因为编写更快且更不易出错），可以选用分行方法。

下表是使用公式 $效率 = \frac{串行计算时间}{并行计算时间*处理器数量}$ 求得的并行计算效率，供参考：

分行	(串行)	2	4	6	8	9	10	12
	1	0.938	0.936	0.868	0.694	0.403	0.395	0.358
分块	/	/	**4**=2*2	**6**=2*3	**8**=2*4	**9**=3*3	**10**=2*5	**12**=3*4
	/	/	0.996	0.884	0.706	0.399	0.393	0.355

四 总结与展望

在本次实验中，笔者用矩阵分块的方式实现了 Floyd-Warshall 算法的并行化。而经过数值实验，并行化后的算法相比串行表现出了明显的速度优势。

受条件限制，本次实验还有许多不足，以下 3 个问题是值得后续探究的：

1. 当处理器数量更多时，分块方法能否相比分行方法体现出明显优势？

2. 一个具体问题：使用 16 个处理器时（16=4*4，16 与 4 均为 2 的幂），能否达到比 8 处理器更短的计算时间？

3. 当使用 $N=a_1*b_1=a_2*b_2$ 个处理器时，采用不同的区域分解方式（a_1 行×b_1 列或 a_2 行×* b_2 列）是否会对分块方法的计算效率产生影响？

以上。