中山大学数据科学与计算机院本科生实验报告

（2020 学年秋季学期）

课程名称：**高性能计算程序设计**　　任课教师：**黄聃**　　**批改人：**

年级+班级	2018 级五班	专业（方向）	计算机科学与技术
学号	18340126	姓名	罗仁良
Email	luorliang@mail2.sysu.edu.cn	完成日期	2020 年 10 月 8 日

1　实验目的

1）使用 MPI 实现通用矩阵乘法（两种通信方式）

- 点对点通信
- 集合通信

2）比较两种矩阵乘法实现的性能，并做一定的优化

3）改造 Lab1，封装一个矩阵乘法的库函数

2　实验过程和核心代码

2.1　点对点通信矩阵乘法

使用 MPI 中的 MPI_Send( )和 MPI_Recv( )函数，实现点对点通信，实现在多个节点上并行计算矩阵乘法。

2.1.1　矩阵存储

考虑到高速缓存的硬件结构，矩阵使用连续空间上的一维数组进行存储。没使用二维数组，主要原因是二维数组在 C/C++中使用 malloc/new 函数申请的内存空间并非连续的，破坏了局部性原理，可能会增大访存的时间开销，降低高速缓存的命中率。同时 MPI 通信的发送数据是连续存储的数据块，为减少通信次数，降低通信开销，采用了一维数组的存储方式。

另外部分矩阵是以**转置的形式存储**的，具体原因见任务划分中分析

2.1.2　任务划分

设矩阵$A \in R^{m \times n}$，矩阵$B = [\ b_1\ b_2\ b_3 \dots b_k\ ] \in R^{n \times k}$，则

$$A \cdot B = [\ Ab_1\ Ab_2\ Ab_3 \dots\ Ab_k\ ] \in R^{m \times k}$$

根据上式可以得到一种任务划分的方式，每个节点的进程执行部分矩阵和向量的乘法运算，最后再汇总得到最后的计算结果。为使得各个进程负载均衡，把 k 个列向量平均分配到运行的线程中。通信工作主要是是矩阵 A 由 0 号进程发送到其他进程，

同时分发对应的列向量到对应的进程，最后 0 号进程回收其他进程的计算结果。注意到分配的任务是根据矩阵 B 的列向量分配的,所以**矩阵 B 需要以转置的形式存储，使得列向量在内存空间上是连续的。**

```
/* ***
 * 进程任务
 * 一个小任务是 矩阵 A 与向量 b 的乘法运算 A*b
 * 进程任务由多个小任务组成
 * ***/

struct Task
{
    int first;  //起始位置
    int last;   //结束位置
    int count;  //任务数量
};

/* ***
 * 任务划分
 * total 任务总量 size 进程数量 rank 进程序号
 * 尽量平均分配保证负载均衡
 * ***/

task get_task(int total, int size, int rank) {
    int q = total / size;
    int r = total % size;
    int count;
    task mytask;
    if(rank < r) {
        count = q + 1;
        mytask.first = rank*count;
    }
    else {
        count = q;
        mytask.first = rank*count + r;
    }
    mytask.last = mytask.first + count;
    mytask.count = count;
    return mytask;
}
```

2.1.3 通信和计算

矩阵数据初始化和数据分发由进程0完成。由于每个进程得到的数据不同，多个进程间进行通信可以使用进程号作为MPI_Send()和MPI_Recv()的tag参数，以保证每个进程仅接受自己需要的数据。

```
        //分发数据
    for(int i=1;i<comm_sz;i++) { // i = rank
          col_task = get_task(k,comm_sz,i);
          MPI_Send(A,m*n,MPI_INT,i,i,MPI_COMM_WORLD);
MPI_Send(B+col_task.first*n,col_task.count*n,MPI_INT,i,i,MPI_COMM_WORL)
;
        }

        // 接受数据
        // 矩阵
        MPI_Recv(A,m*n,MPI_INT,0,my_rank,MPI_COMM_WORLD,NULL);
        // 列
MPI_Recv(subCol,col_task.count*n,MPI_INT,0,my_rank,MPI_COMM_WORLD,NULL)
;
```

计算的过程和一般矩阵乘法没有区别，需要注意的是列向量构成的矩阵是以转置的形式存储的。

```
            // 计算
            for(int i=0;i<m;i++){
                for(int j=0;j<col_task.count;j++){
                    int sum = 0;
                    for(int l=0;l<n;l++){
                        sum += A[i*n+l]*subCol[j*n+l];
                    }
                    result[j*m+i] = sum;
                }
            }
```

2.2 集合通信矩阵乘法

使用 MPI 提供的集合通信函数 MPI_Bcast()、MPI_Scater()和 MPI_Gather()实现矩阵乘法。

数据存储和任务划分的方式和点对点通信的矩阵乘法相同，主要的差别在于通信方式上。

2.2.1 集合通信

1）矩阵 A 采用广播的方式传输到不同节点的进程中

2）矩阵 B 采用 scater 的方式将数据分发到各个进程中

由前文任务划分的算法，设每个进程分得 count 个列向量，依照 scater 分配数据的方式，每个进程得到矩阵 B 的$count \times n$个元素。

```
    int count = n*k/comm_sz;
    int col_num = k/comm_sz;
    int *subCol = new int[count];
    int *result = new int[m*col_num];
    // 分发数据
    MPI_Scatter(B,count,MPI_INT,subCol,count,MPI_INT,0,MPI_COMM_WORLD);
```

3）进程计算的结果聚集到进程 0 中

```
 // 聚集
MPI_Gather(result,m*col_num,MPI_INT,C,m*col_num,MPI_INT,0,MPI_COMM_WORLD);
```

2.2.2 计算

```
for(int i=0;i<m;i++){
        for(int j=0;j<col_num;j++) {
            int sum = 0;
            for(int l=0;l<n;l++)
                sum += A[i*n+l]*subCol[j*n+l];
            result[j*m+i] = sum;
        }
}
```

注意到 result 中的计算结果也是按照转置的形式存储的，所以在聚集之后的最终结果也是以转置的形式存储的。

2.3 矩阵乘法库函数封装

1）编译生成库函数

使用编译指令，将源文件编译成一个动态连接库

```
g++ -fPIC  - shared Matrix.cpp  - o libMatrix.so
```

- -fPIC 生成位置无关代码
- -shared 生成 so 共享库

注：在 linux 和 unix 中的 so 文件，其扩展名必须是 so，文件前缀也必须是 lib

2）使用库函数

- 用户源文件需要包含对应库的头文件
- 使用编译器链接时添加 - Lpath （库函数目录）和 - llibname（libname 不含前缀 lib 和后缀.so）

```
g++ matlibtest.cpp -L. -lMatrix -o test
```

3 实验结果

3.1 点对点通信

时间（s） 矩阵大小 / 线程数量	512	1024	2048
1	0.667560	3.798412	24.565655
2	0.526553	2.679416	24.742933
4	0.540325	2.133913	10.884814
8	0.473717	2.082084	10.161226

3.2 集合通信

时间（s） 矩阵大小 / 线程数量	512	1024	2048
1	0.669054	3.847278	24.562997
2	0.530629	2.739076	24.414293
4	0.461782	2.128111	10.836597
8	0.467499	2.025438	9.738748

```
 HpcLab2  → mpirun --mca btl self,tcp -np 1 group 1024 1024 1024
m n k :1024 1024 1024
time:3.847278 s
 HpcLab2  → mpirun --mca btl self,tcp -np 2 group 1024 1024 1024
m n k :1024 1024 1024
time:2.739076 s
 HpcLab2  → mpirun --mca btl self,tcp -np 4 group 1024 1024 1024
m n k :1024 1024 1024
time:2.128111 s
 HpcLab2  → mpirun --mca btl self,tcp -np 8 group 1024 1024 1024
m n k :1024 1024 1024
time:2.025438 s
```

3.3 结果分析

1) 无论时点对点还是集合通信,进程数量越多计算速度越快,且在4进程运行时的加速效果最为明显.导致这个现象的原因应该是实验过程中是在单节点下多进程进行的,实际的算力资源还是只有一个CPU资源,在这种情况下CPU的核心数就对计算的速度影响很大.

2) 集合通信相对与点对点通信有速度上的提升,但提升的效果并不是特别明显,可能和参与通信的进程数量不多,通信的数据量不大,导致两者之间的差异不太明显.

4 实验感想

本次实验使用了MPI实现矩阵乘法,掌握了编写基本MPI并行程序的基本框架,加深了对消息传递机制的理解.实验过程中对矩阵的存储方式尝试了两种不同的方案,即二维数组和一维数组.经过尝试发现一维数组的存储方式结合矩阵转置存储,提高了运算的效率.同时一维数组的连续空间和MPI中数据传输的地址空间更加契合.