中山大学计算机院本科生实验报告

(2023 学年春季学期)

课程名称：并行程序设计　　批改人：

实验	基于 MPI 的并行矩阵乘法（进阶）	专业（方向）	计算机科学与技术
学号	21307082	姓名	赖耿桂
Email	laigg@mail2.sysu.edu.cn	完成日期	2024.4.5

1. 实验目的（200 字以内）

本次实验是在上一次实验也即使用 MPI 点对点通信实现并行矩阵乘法的基础上，进一步改进代码，使用 MPI 集合通信实现并行矩阵乘法，并尝试使用 mpi_type_create_struct 聚合进程内变量后通信。

2. 实验过程和核心代码（600 字以内，图文并茂）

使用 MPI 集合通信实现并行矩阵乘法：

与点对点通信使用 MPI_Send 和 MPI_Recv 进行通信不同，在 0 号进程中使用 MPI_Bcast 将矩阵 B 广播给各个进程(包括 0 号进程)：

```
// 将矩阵B广播到其他进程
MPI_Bcast(B.data(), N * K, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

然后再将矩阵 A 的分块使用 MPI_Scatter 发送给各个进程(包括 0 号进程，因此 0 号进程也需要对矩阵 A 的第一个分块进行矩阵乘法运算)：

```
// 使用MPI_Scatter将矩阵A分块发送至其他进程
MPI_Scatter(A.data(), rows_per_process * N, MPI_DOUBLE, temp_A.data(), rows_per_process * N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

// 由于MPI_Scatter也会发送矩阵A分块给0号进程，因此需要在0号进程也进行一次矩阵乘法
for(int i = 0;i < rows_per_process; ++i){
    for(int k = 0; k < K; ++k){
        double tmp = 0;
        for(int j = 0; j < N; ++j){
            tmp += temp_A[i * N + j] * B[j * K + k];
        }
        temp_C[i * K + k] = tmp;
```

```
    }
}
```

在其他进程中，需要使用 MPI_Bcast 和 MPI_Scatter 接收 0 号进程发送过来的矩阵 B 和矩阵 A 的分块，接收代码与发送代码相同，不再赘述。而在该进程内完成矩阵乘法后，使用 MPI_Gather 将运算结果汇总到 0 号进程：

```
MPI_Gather(temp_C.data(), rows_per_process * K, MPI_DOUBLE, C.data(), rows_per_process * K, MPI_DOUBLE, 0, MPI_COMM_WORLD);
// 使用MPI_Gather 将运算结果汇总到0 号进程
```

而在 0 号进程中也使用相同的代码来接收其他进程汇总的结果。

使用 mpi_type_create_struct 聚合进程内变量后通信：

首先我定义了一个结构体，其成员有矩阵的行数和列数以及完整的矩阵信息：

```
// 定义结构体Matrix
typedef struct{
    int row, col;   // 矩阵的行数、列数
    vector<double> Mat;     // 具体的矩阵
}Matrix;
```

然后我定义了一个函数 Build_MPI_Type，用于创建自定义 MPI 数据类型：

```
// 建立自定义MPI 数据类型
void Build_MPI_Type(Matrix* mat, MPI_Datatype* mytype){
    int block_lengths[3] = {1, 1, mat->row* mat->col};      // 每个块的长度
    MPI_Datatype types[3] = {MPI_INT, MPI_INT, MPI_DOUBLE}; // 每个块的数据类型
    MPI_Aint start_addr, row_addr, col_addr, mat_addr;      // 每个块的地址，用于计算偏移量（要使用结构体的基地址）
    MPI_Aint displacements[3];      // 每个块的偏移量
    MPI_Get_address(mat, &start_addr);      // 获取基地址

    MPI_Get_address(&mat->row, &row_addr);      // 获取row 的地址
    displacements[0] = row_addr - start_addr;   // 计算row 的偏移量

    MPI_Get_address(&mat->col, &col_addr);      // 计算row 的地址
    displacements[1] = col_addr - start_addr;   // 计算col 的偏移量

    MPI_Get_address(mat->Mat.data(), &mat_addr);    // 计算mat 的地址
    displacements[2] = mat_addr - start_addr;   // 计算mat 的偏移量

```

```
    MPI_Type_create_struct(3, block_lengths, displacements, types, mytype);    // 创建MPI 数据类型
    MPI_Type_commit(mytype);    // 使其生效
}
```

接下来我打算将矩阵 B 及其行列数信息使用刚刚定义的 MPI 数据类型广播到各个进程进行并行矩阵乘法运算，现对 main 函数进行修改。

首先不在 0 号进程进行矩阵的初始化，改为在执行所有进程之前就完成 A、B 矩阵的初始化和自定义 MPI 数据类型 MAT_TYPE 的创建：

```
Matrix *B=new Matrix();
MPI_Datatype MAT_TYPE;

// 调用随机矩阵生成函数random_matrix_generator，初始化矩阵A 和B
A = random_matrix_generator(M , N);
B->row = N;
B->col = K;
B->Mat.resize(B->row * B->col);
B->Mat = random_matrix_generator(N,K);
Build_MPI_Type(B, &MAT_TYPE);
```

在 0 号进程，打印矩阵 A、B 并将 B 广播出去，广播代码如下：

```
// 将矩阵B 广播到其他进程
MPI_Bcast(B, 1, MAT_TYPE, 0, MPI_COMM_WORLD);
```

然后将 A 使用 MPI_Scatter 散射出去，进行 0 号进程的矩阵乘法，使用 MPI_Gather 收集其他进程的运算结果,打印输出矩阵C(基本与集合通信相同)。

接下来是其他进程，使用 MPI_Bcast 接收来自 0 号进程广播的结果：

```
// 接收0 号进程发过来的矩阵B
MPI_Bcast(B, 1, MAT_TYPE, 0, MPI_COMM_WORLD);
```

同样是接收矩阵 A 的分块，进行矩阵乘法运算，并将运算结果使用 MPI_Gather 汇总后发回 0 号进程(基本与集合通信相同)。

最后在 MPI 结束之前释放 B 和 MAT_TYPE：

```
delete B;
MPI_Type_free(&MAT_TYPE);
```

3. 实验结果（500 字以内，图文并茂）

使用 MPI 集合通信实现并行矩阵乘法：

■ 在终端进行编译：

```
1.  mpicxx -g -Wall -o lab2_main_v1 lab2_main_v1.cpp
```

■ 然后运行(这里以两个 256*256 的方阵、4 个进程为例)：

```
1.  mpirun -np 4 ./lab2_main_v1 256 256 256 4
```

■ 下为运行示例，仅展示程序的正确性：

```
laigg@laigg-VirtualBox:~/codes/parallel/lab2$ mpicxx -g -Wall -o lab2_main_v1 lab2_main_v1.cpp
laigg@laigg-VirtualBox:~/codes/parallel/lab2$ mpirun -np 1 ./lab2_main_v1 4 4 4 1
Matrix A:
-4.095545 3.224974 -2.933412 -0.558416
3.886115 1.233563 4.903036 0.566953
4.422140 2.162536 0.495633 3.890045
-2.418343 -0.017684 2.106906 -4.850563

Matrix B:
-1.474379 -3.719827 3.642154 2.249519
2.361628 0.325377 -3.717367 -0.876426
-3.091128 -0.476390 3.476054 -2.167783
-3.745071 -4.560294 -1.630146 -2.779832

Matrix C:
24.813433 20.228041 -36.191420 -4.128158
-20.095578 -18.975534 25.687225 -4.543990
-17.513359 -33.721817 3.448682 -3.835708
15.176777 30.106349 6.488604 3.491824

在1个进程并行的情况下计算大小为4*4的矩阵A与大小为4*4的矩阵B的乘积所用时间为：0.000013 s.
```

进程数	矩阵规模				
	128	256	512	1024	2048
1	0.016942s	0.138081s	1.239442s	22.502296s	321.067672s
2	0.010285s	0.080755s	0.690520s	11.648789s	169.456280s
4	0.007949s	0.061059s	0.427376s	7.086423s	100.076745s
8	0.082842s	0.198613s	0.584331s	6.861979s	82.465020s
16	0.278671s	0.461193s	1.641237s	7.304531s	72.293211s

可以看到从总体上来说，集合通信的效率比点对点通信的效率要高。这是因为集合通信使用多播通信(一对多)，可以在一次通信向多个进程发送相同的信息(MPI_Bcast)或一次发送一组进程(MPI_Scatter)，从而可以减少通信次数，进而减少通信开销。

使用 mpi_type_create_struct 聚合进程内变量后通信：

■ 在终端进行编译：

```
1.  mpicxx -g -Wall -o lab2_main_v2 lab2_main_v2.cpp
```

■ 然后运行(这里以两个 256*256 的方阵、4 个进程为例)：

```
1.  mpirun -np 4 ./lab2_main_v1 256 256 256 4
```

■ 下为运行示例，仅展示程序的正确性：

进程数	矩阵规模				
	128	256	512	1024	2048
1	0.019560s	0.184564s	1.882533s	21.769806s	329.628576s
2	0.013050s	0.106155s	1.119324s	13.813801s	178.920197s
4	0.044283s	0.109235s	0.869553s	7.892684s	111.328950s

8	0.349310s	0.528636s	1.085440s	7.114249s	86.093597s
16	0.459165s	1.142136s	1.466814s	6.450570s	74.993253s

由于在使用 MPI_Type_create_struct 创建新的 MPI 数据类型来聚合进程内变量通信时，也使用集合通信，效率上和集合通信差不多。但是效率会稍微低一些，这是因为使用 MPI_Type_create_struct 创建新的 MPI 数据类型时，由于原结构体还包含了其他变量(矩阵的行列数)，因此在一定程度上增加了通信开销，导致效率有所下降。

4. 实验感想（200 字以内）

通过本次实验，我对 MPI 集合通信有了更深刻的理解，更加清楚相较于点对点通信它是如何提高通信效率的。除此之外，我也学习掌握了如何使用 MPI_Type_create_struct 来自定义 MPI 数据类型。

在此次实验中，由于我一开始对实验所涉及的函数不是很了解，导致在实验时出现了段错误等问题，但后面经过回顾理论课所学知识以及查阅相关资料后，顺利解决了问题并加深了对各函数以及集合通信方式的理解。