《高性能程序设计基础》

实 验 报 告

实验序号： 实验 3

实验名称： MPI 矩阵向量乘法的并行算法

姓名： 劳马东

学号： 16337113

实验名称：用 MPI 完成矩阵向量乘法的并行算法

一、 实验目的：

1、熟悉 MPI 全局聚集函数的使用及任务划分方法。

二、 实验要求：

1. 任务划分按照数据划分方法
   - 输出数据划分
   - 按输入数据划分
2. 矩阵和向量从磁盘读入
3. 结果输出到磁盘

三、 实验过程

1. 矩阵读入

   矩阵的读入需要注意两个问题。第一是输入文件的格式为“列号 行号 元素”，而且从 1 开始；第二是由于只有矩阵的一部分是要划分给某个进程的，因此需要有一个读停止的地方——行号超过划分行号或到文件结尾。代码如下：

```
int pos_x = 0, pos_y = 0;
double entry = 0;
void read_matrix(FILE* file, double *arr, int m, int n, int end, int* cnt) {
    memset(arr, m * n * sizeof(double), 0);
    if (pos_x) {
        int i = (pos_y - 1) % m, j = (pos_x - 1) % n;
        arr[i * n + j] = entry;
    }

    while ((*cnt)--) {
        if(fscanf(file, "%d %d %lf", &pos_x, &pos_y, &entry) == EOF)
            break;
        if (pos_y > end)
            break;
        int i = (pos_y - 1) % m, j = (pos_x - 1) % n;
        arr[i * n + j] = entry;
    }
}
```

arr 是一个 m*n 二维矩阵，end 是划分的终止行号， cnt 代表文件中剩下的非零元个数。函数首先将输出的矩阵清零，去掉上次读入残留的结果，其次是很关键的一步——在上一次读超的一行，需要在这一次中写入 arr 矩阵，否则就漏掉了一个元素。接下来的循环一行行读取元素直到超过终止行或文件结尾。

2. 矩阵划分

对于一个 m*n 的矩阵，需要将其第一维尽可能平均地划分到每个进程。方法是先求 m 整除进程个数 comm_size 的结果，这是每个进程至少获得的矩阵行数。对于余数 remain，将这 remain 行非给前 remain 个进程，即 rank 为 0 到 remain-1。下图是该过程的代码。

```
void divide(FILE* file, int m, int n, int cnt, int comm_size, double **local_A, int *local_m) {
    int tmp_m;
    int remain = m % comm_size;
    int i;
    MPI_Request request;

    double *tmp_A = NULL;

    int end = 1;

    for (i = 0; i < comm_size; i++) {
        tmp_m = m / comm_size;
        tmp_m += i < remain;
        end += tmp_m;
        if (i == 0) {
            *local_m = tmp_m;
            *local_A = malloc(tmp_m * n * sizeof(double));
        } else {
            MPI_Isend(&tmp_m, 1, MPI_INT, i, 0, MPI_COMM_WORLD, &request);
        }
        if (tmp_A == NULL) {
            tmp_A = malloc(tmp_m * n * sizeof(double));
        }
        read_matrix(file, tmp_A, tmp_m, n, end, &cnt);
        if (i == 0) {
            memcpy(*local_A, tmp_A, tmp_m * n * sizeof(double));
        } else {
            MPI_Isend(tmp_A, tmp_m * n, MPI_DOUBLE, i, 0, MPI_COMM_WORLD, &request);
        }
    }
}
```

将局部的矩阵发送给对应进程的过程采用了异步发送，因为进

程 0 接下来不需要用到这个局部矩阵，这样消息发送和矩阵读取可并发执行，从而提高性能。

3. 矩阵输出

矩阵输出的过程就是主进程从其他进程接受局部的 y 并输出的过程。需要注意的问题有两个，第一是每个进程发送给主进程的时机是不确定的，因此主进程的接收函数需要用 MPI_ANY_SOURCE 并在 status 中获取实际的源进程；第二是将局部的行号、列号转化为全局的行号、列号。下图是该过程的代码。

```
if (rank == 0) {
    FILE *out = fopen("result.mtx", "w");
    fprintf(out, "%d\t1\t60222\n", m);
    int i;

    output(out, local_y, local_m, 0);

    MPI_Status status;
    int cnt;
    for (i = 1; i < comm_size; ++i) {
        MPI_Recv(local_y, local_m, MPI_DOUBLE, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status);
        MPI_Get_count(&status, MPI_DOUBLE, &cnt);
        output(out, local_y, cnt, status.MPI_SOURCE);
    }
} else {
    MPI_Send(local_y, local_m, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
}
```

```
int num_nonzero = 0;
void output(FILE* out, double *arr, int n, int rank) {
    int i;
    int base = n * rank + 1;
    for (i = 0; i < n; ++i) {
        if (arr[i]) {
            fprintf(out, "%d\t1\t%lf\n", base + i, arr[i]);
            num_nonzero++;
        }
    }
}
```

四、 实验数据分析与问题讨论

测试环境为集群，输入数据为 matrix.mat 文件和 vector.mat 文

件，测试进程数为 16 个，输出文件为 result.mat，下图是部分截图。可以看到矩阵大小为 60222*1，非零元个数为 60222。