

石邢越 16337208

张永东教授

高性能计算程序设计实验

2018 年 9 月 24 日

### **实验 3: MPI 实现矩阵和向量的并行乘法**

#### **1. 实验目的**

- a) 尝试使用 MPI 中的集合通行函数;
- b) 对于数据量更大处理对象, 因为内存空间可能不够大, 需要学会直接读写磁盘;
- c) 尝试使用 MPI 中的计时函数 `MPI_Wtime`, 并用更多的指标来评测 MPI 并行程序的性能, 加速比、效率 etc.

#### **2. 实验要求**

- a) 用 MPI 完成矩阵向量乘法的并行算法;
- b) 按数据划分;
- c) 待乘矩阵和向量和输入、结果矩阵的输出直接读写磁盘而不经内存中;
- d) 矩阵和向量按三元组的形式储存在 `mtx` 格式文件中。

### 3. 算法设计

记向量矩阵乘法中各个变量为

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{y}$$

这次实验中，我需要解决的几个主要算法问题如下：

a) 如何对乘法过程进行分解以进行并行优化？

矩阵和向量乘法的分解方法至少有三种，按行分解矩阵，按列分解矩阵和将矩阵均分成多个更小的方阵。

我最初的选择是按行分解矩阵，这是因为这种分解方法是最直观的，和笔算矩阵和向量乘法的过程一致。但在这种分解方法中，矩阵中的每一行都需要和向量求内积，这样一来每次计算内积的时候要么读一遍向量的文件，要么将整个向量通过进程间的通行发送给正在算内积的进程。每次都读一遍文件很耗时；而第二种解决方法需要先将整个向量放在内存中，之后再发送给各个进程，需要申请较多的内存空间，当矩阵、向量很大的时候可能无法申请到足够的空间。

按列分解的算法为：对于矩阵  $\mathbf{A}$  中的第  $i$  列，求这个列向量和向量  $\mathbf{x}$  中的第  $i$  个元素数乘；之后将各个数乘后的各个列向量相加得到结果向量  $\mathbf{y}$ 。这样每个计算数乘的进程只需要用到  $\mathbf{x}$  中的一个元素，不需要反复读文件或者产生向量  $\mathbf{x}$  的副本，使用的内存空间比按行分解小。所以我最后选择按列分解。

不选择分解成小方阵的原因在于这种分解很难分成多个大小相等的小方阵（comm\_sz 可能不整除矩阵 A 的秩），之后分别处理比较麻烦。

b) 如何解决内存不足的问题？

选择按列分解的算法并实现了相应代码后，我首先测试了很小的矩阵和向量的乘法（4x4 的矩阵，4x1 的向量），得到了正确的结果。

但换用要求使用的矩阵和向量，因为**内存中没有足够的空间用于存放这些数据**，在读矩阵的时候就会报错停止运行，如图 1。

```
Proc 0 > In Read_matrix, Can't allocate temporary matrix
```

图 1. 因为算法设计不当（首先给每个进程分配了用于接收 A 中一部分的空间），消耗了过多的内存，之后在 0 号进程读矩阵时申请不到足够多的内存来存放整个 A.

我猜测一个节点可以用的内存为 36MB（没有什么根据地 LOL），矩阵 A 大小约 24MB, 向量 x 大小约 8MB。我**原来的算法**为：

①每个进程封面分配 local\_A（A 中的几列），local\_x（x 中的几个元素）,local\_y（计算出的几个向量）所需的内存空间

(24MB+8MB+8MB=40MB)

②0 号进程读入并存储 A 和 x (24MB+8MB=32MB)

③0 号进程用 scatter 函数把 A 和 x 分到各个进程中，发完中释放 0 号中为 A,x 申请的内存

以上三步用到的内存约为  $40\text{MB}+32\text{MB}=72\text{MB}$ ，内存不够了.....

```

---
147 void Allocate_arrays(
148     double** local_A_pp /* out */,
149     double** local_x_pp /* out */,
150     double** local_y_pp /* out */,
151     int      m           /* in  */,
152     int      local_m     /* in  */,
153     int      local_n     /* in  */,
154     MPI_Comm comm        /* in  */) {
155
156     int local_ok = 1;
157
158     *local_A_pp = malloc(m*local_n*sizeof(double));
159     *local_x_pp = malloc(local_n*sizeof(double));
160     *local_y_pp = malloc(local_m*sizeof(double));
161
162     if (*local_A_pp == NULL || local_x_pp == NULL ||
163         local_y_pp == NULL) local_ok = 0;
164     Check_for_error(local_ok, "Allocate_arrays",
165                     "Can't allocate local arrays", comm);
166 } /* Allocate_arrays */
---

```

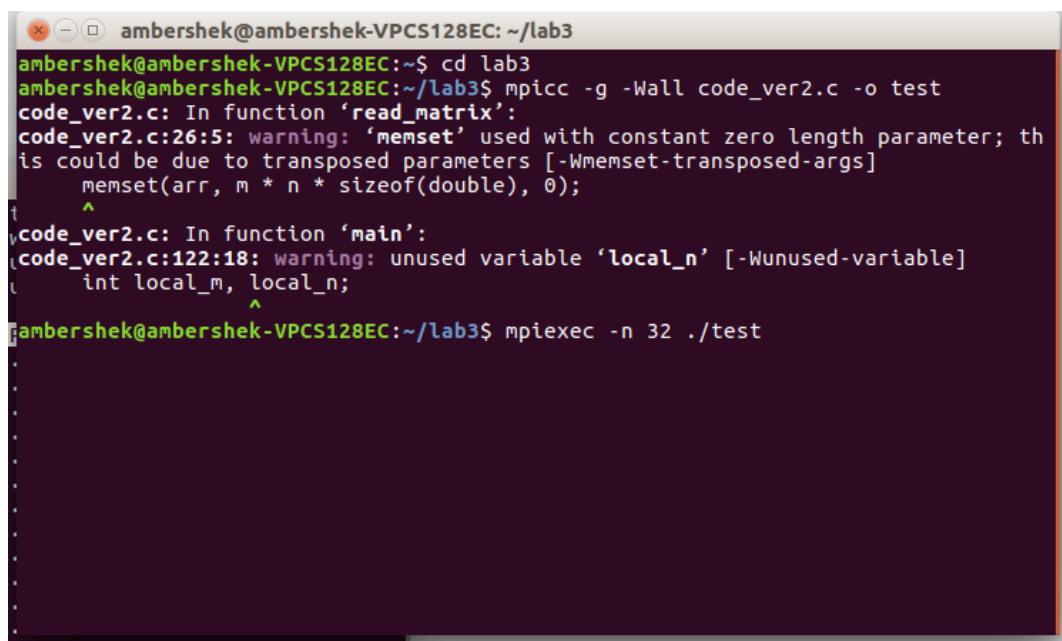
图 2.原来的算法中的第一步。每个进程都预先分配好了第三步时要用来接收 A,x 一部分的空间，相当于 A 和 x 所占的空间都被申请了两次！

分析发现可以用“**时间换空间**”的方法来减少内存使用，即用循环让各个进程在 0 号进程读完文件后**顺次申请**接收 local\_A, local\_x 所需要的内存（**而不是之前那样在 0 号进程读文件之前就都申请了**），申请到了之后 0 号进程将这个进程需要的 local\_A, local\_x 发给此进程（**用 send 函数而非 scatter 函数**，因

为其他函数可能还没申请内存无法 scatter)，发完即释放这部分内容占的内存。

#### 4. 实验过程

编译和运行的命令如图 3.



```
ambershek@ambershek-VPCS128EC: ~/lab3
ambershek@ambershek-VPCS128EC:~$ cd lab3
ambershek@ambershek-VPCS128EC:~/lab3$ mpicc -g -Wall code_ver2.c -o test
code_ver2.c: In function 'read_matrix':
code_ver2.c:26:5: warning: 'memset' used with constant zero length parameter; this could be due to transposed parameters [-Wmemset-transposed-args]
    memset(arr, m * n * sizeof(double), 0);
    ^
code_ver2.c: In function 'main':
code_ver2.c:122:18: warning: unused variable 'local_n' [-Wunused-variable]
    int local_m, local_n;
                   ^
ambershek@ambershek-VPCS128EC:~/lab3$ mpiexec -n 32 ./test
```

图 3.编译运行

#### 5. 实验结果

计算得出的向量 y 输出到文件 y.mtx”中，见提交的文件夹。

运行时用 top 查看 CPU 和内存的工作情况如图 4.

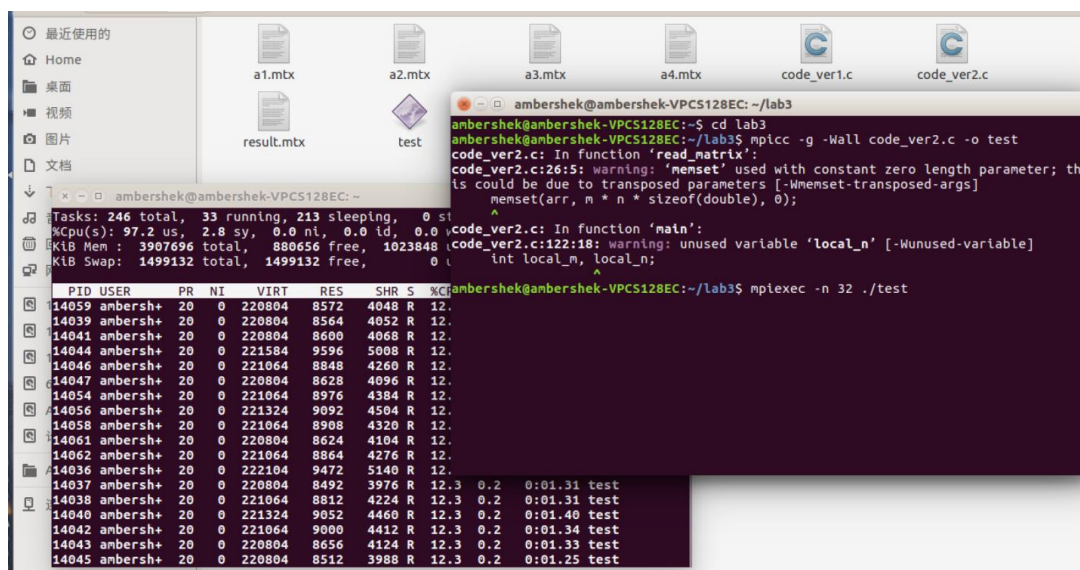


图 4.用 top 查看运行情况

我比较了 32 个线程，64 个线程和 128 个线程运行时的每个进程 CPU 利用率，

分别约为 12.5%，6.6%，3.7%，看起来实际上这个程序所有线程总共占用的 CPU

其实是差不多的（即总的计算量差不多，通信所占的比重不算特别大）。

## 引用作品

Pacheco, P. *并行程序设计导论*. 机械工业出版社, 2012. 打印。