Homework

MPI编程

Name:Li Shibo Student ID:119033910046 Email: ShiboLi@sjtu.edu.cn

MPI_allgather $MPI_allgather$是将所有的数据聚合到每个进程中。工作原理如下图所示。

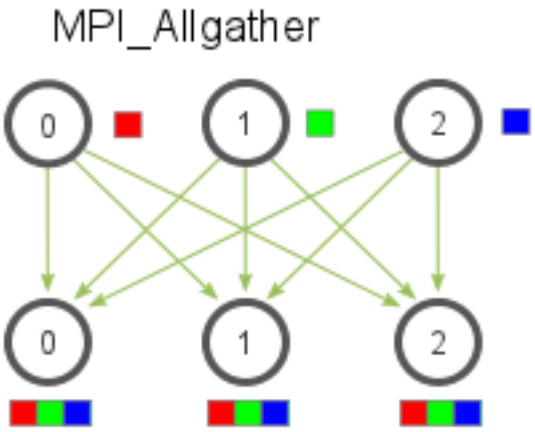

图 1: allgather工作原理

本次实验的目的是自己实现$MPI_allgather$函数，然后与官方实现的函数进行对比。在本次实验中我使用MPI_send和MPI_recv来实现$MPI_allgather$。实现方法如下。

1. 所有进程向出自己之外的进程发送消息，然后将自己进程的数据直接拷贝到接收数组的对应位置。

2. 所有进程接除自己进程以外的消息，然后写道接收数组的对应位置。

实验过程中，每个进程发送大小为32个整数的数组。实验结果如图所示。

```
**********0**********
My implement:
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  3  3  3  3  3  3  3  3
  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  4  4  4  4  4  4  4  4  4  4
4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7
7  7  7  7  7  7  7  7  7  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
 8  8  8  8  8  8
MPI:
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  3  3  3  3  3  3  3  3
  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  4  4  4  4  4  4  4  4  4  4
4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7
7  7  7  7  7  7  7  7  7  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
 8  8  8  8  8  8
**********0**********
```

图 2: allgather实验结果与官方all_gather函数结果对比。图中使用0号进程分别打印出了接收到的结果。

从图中可以看出得到的结果相同。分别使用8个和16个进程时，allgather实验与官方all_gather函数所用时间对比如下图所示。

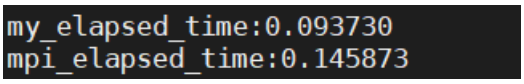

图 3: allgather实验结果与官方all_gather函数结果时间对比对比。图中使用8个进程进行100次试验的平均结果。

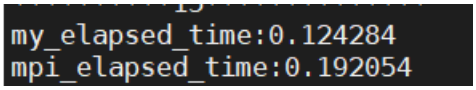

图 4: allgather实验结果与官方all_gather函数结果时间对比对比。图中使用16个进程进行100次试验的平均结果。

从图3图4可以看出，在发送32个整数时，分别使用8和16个进程进行对比时，自己实现的all_gather函数和官方库函数性能相差不大。

矩阵相乘

分块矩阵乘法 本次实验采用分块的思想将两个1024*1024大小的方阵进行相乘。本次实验采用connon算法进行实现。connon算法是一种存储有效的算法。为了使两矩阵的下标满足相乘的要求，各个分块有目的地的在各行和各列施行循环位移，从而使处理器的总存储要求可以降下来。实验中降矩阵A和B分成p块$A_{ij}, B_{ij}, (0 \leq i, j \leq \sqrt{p}-1)$，$p$是处理器的个数，每块大小为$(n/\sqrt{p}) * (n/\sqrt{p})$。并将这些块分给$p$个处理器，开始时处理器$P_{ij}$存放块$A_{ij}$和块$B_{ij}$，并负责计算块$C_{ij}$，然后算法开始执行：

1. 将块$A_{ij}(0 \leq i, j \leq \sqrt{p}-1)$向左循环移动$i$步；$B_{ij}(0 \leq i, j \leq \sqrt{p}-1)$向上循环移动j步。
2. P_{ij}执行乘法和加法运算；然后将块$A_{ij}(0 \leq i, j \leq \sqrt{p}-1)$向左循环移动1步；$B_{ij}(0 \leq i, j \leq \sqrt{p}-1)$向上循环移动1步。
3. 重复第二步，在P_{ij}中一共执行$\sqrt{p}$次乘法和加法运算，和$\sqrt{p}$次块$A_{ij}(0 \leq i, j \leq \sqrt{p}-1)$和$B_{ij}(0 \leq i, j \leq \sqrt{p}-1)$的循环单步位移。

实验过程中，为方便矩阵的发送和接收，我使用一维数组存储矩阵。为方便每个块之间的循环位移，建立处理器的2维笛卡尔拓扑。使用MPI_scatter和MPI_gather进行数据的分发和接收。使用MPI_Sendrecv_replace函数进行块与块之间转移时进程之间的通信。在分别使用1,4,16,64个处理器的情况下，所使用的的时间如下图所示。

```
root@ecs-00:~/MPIPrograme/matrix_multiplication# mpicc matrix_multi.c -o matrix_multi -lm
root@ecs-00:~/MPIPrograme/matrix_multiplication# mpiexec -n 4  ./matrix_multi
elapsed time when paeallel with 4 processes: 13.374916
root@ecs-00:~/MPIPrograme/matrix_multiplication# mpiexec -n 16  ./matrix_multi
elapsed time when paeallel with 16 processes: 6.687063
root@ecs-00:~/MPIPrograme/matrix_multiplication# mpiexec -n 64  ./matrix_multi
elapsed time when paeallel with 64 processes: 14.505389
root@ecs-00:~/MPIPrograme/matrix_multiplication# mpiexec -n 1  ./matrix_multi
elapsed time when no paeallel: 25.546282
```

图 5: 矩阵乘法在不同数量处理器的情况下时间对比。图中分别使用1,4,16,64个进程进行试验的结果。

从图中可以看出，在使用16个进程时，所花费的时间最少；当进程数量增多的时候由于进程间相互通信的开销，使得所花费的时间反而增多。由于矩阵较大，而且是随机产生的，在已经试了较小矩阵保证乘法正确的情况下，在这里不打印矩阵相乘的结果。

矩阵池化 在本实验中，*pooling*操作是求一个1024*1024的矩阵在4*4核情况下的最大值。本次实验依然选择分块的思想，但因为分块后矩阵进行pooling操作，每一块的边界部分需要其他块的内容，由于使用4*4大小的核，因此分块的过程中，主进程在分发数据时，每一个块多发送3行核3列的数据。这样，每一块处理器都在自己的部分进行pooling操作，从而不需要进程间的信息交互，会较大程度的加快执行效率。由于每一块都多发送了三行和三列，因此在边界上对原1024*1024的矩阵后面扩充三行和三列0，这样就可以保证每个进程数据大小的一致性，方便用MPI_scatter和MPI_gatge行分发和收集。收集到结果后再进行处理，就可以得到最后的结果。实验结果如图所示。

```
root@ecs-00:~/MPIPrograme/matrix_pooling# mpicc matrix_pooling.c -o matrix_pooling -lm
root@ecs-00:~/MPIPrograme/matrix_pooling# mpiexec -n 1  ./matrix_pooling
elapsed time when no paeallel: 0.110509
root@ecs-00:~/MPIPrograme/matrix_pooling# mpiexec -n 4  ./matrix_pooling
elapsed time when paeallel with 4 processes: 0.440709
root@ecs-00:~/MPIPrograme/matrix_pooling# mpiexec -n 16  ./matrix_pooling
elapsed time when paeallel with 16 processes: 1.376360
root@ecs-00:~/MPIPrograme/matrix_pooling# mpiexec -n 64  ./matrix_pooling
elapsed time when paeallel with 64 processes: 7.511726
```

图 6: 矩阵池化在不同数量处理器的情况下时间对比。图中分别使用1,4,16,64个进程进行试验的结果。

从以上结果种可以看出，矩阵池化操作，在1个进程的情况下执行速度最快，而随着进程数量的增多，所消耗的时间也越来越多，原因时矩阵池化与矩阵相乘比起来计算量较少，因此在单个进程上运行时间较少，在多个进程上运行时由于进程间通信开销，反而使得运行所用的时间增多。

矩阵卷积 在本实验中，卷积操作也是是在一个1024*1024的矩阵和4*4核的情况下进行的，和池化操作类似，只是具体计算不一样。本实验也是考虑矩阵分块进行处理，处理方式与池化操作类似。处理结果如下图所示。

```
root@ecs-00:~/MPIPrograme/matrix_cov# mpicc matrix_cov.c -o matrix_cov -lm
root@ecs-00:~/MPIPrograme/matrix_cov# mpiexec -n 1  ./matrix_cov
elapsed time when no paeallel: 0.108251
root@ecs-00:~/MPIPrograme/matrix_cov# mpiexec -n 4  ./matrix_cov
elapsed time when paeallel with 4 processes: 0.301332
root@ecs-00:~/MPIPrograme/matrix_cov# mpiexec -n 16  ./matrix_cov
elapsed time when paeallel with 16 processes: 1.188015
root@ecs-00:~/MPIPrograme/matrix_cov# mpiexec -n 64  ./matrix_cov
elapsed time when paeallel with 64 processes: 6.941562
```

图 7: 矩阵卷积运算在不同数量处理器的情况下时间对比。图中分别使用1,4,16,64个进程进行试验的结果。

结果与池化类似，进程数量越多消耗的时间越多。

单词数统计 本实验采用manager/slave的模式进行单词统计工作，分别在一个文件夹下包含比较大的文件和一个文件夹下包含较多小文件的数据上进行实验。0号进程担任manager的角色，其他进程担任worker的角色。manager从文件夹中读取文件名字和文件个数，然后空闲进程会向manager发送申请。manager收到申请后选择一个文件名发送给worker，worker收到文件名之后读取文件，然后统计每个单词的个数，最后讲统计结果发送给manager。manager将最终的结果进行合并，得到所有的单词个数统计结果。由于统计词频需要以单词作为键值的hash结构，因此本程序中使用了c++ stl中的map结构进行统计单词频率操作。下图为分别在4，8，16，64个进程的情况下读取小文件的实验结果。

图 8: 使用4个进程统计多个小文件的单词个数的实验结果。

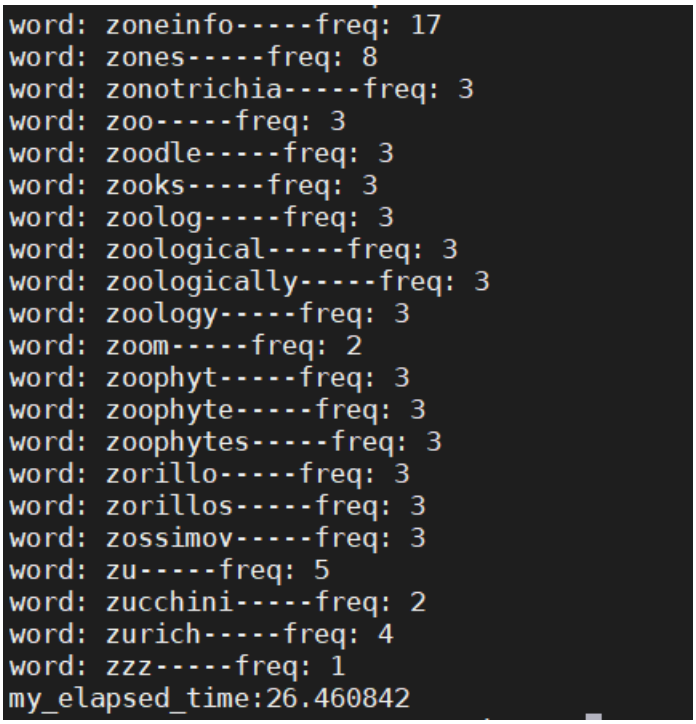

图 9: 使用8个进程统计多个小文件的单词个数的实验结果。

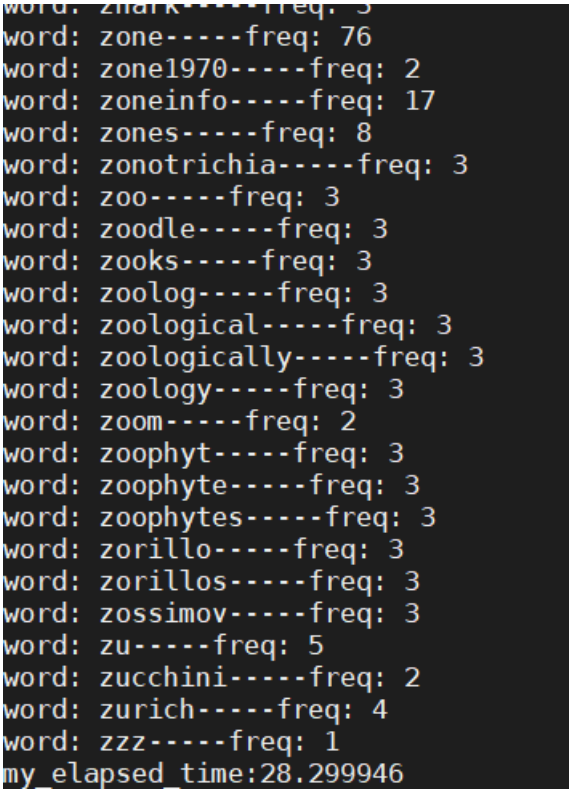

图 10: 使用16个进程统计多个小文件的单词个数的实验结果。

图 11: 使用64个进程统计多个小文件的单词个数的实验结果。

从上图种可以明显看出来，在使用4，8，16个进程时消耗时间相差不大，但是64个进程时，

消耗时间明显曾大。由于服务器只有两个核，因此，当进程数量增大时，只是增加了进程间通信的开销，并没有增加处理文件的效率。读取大文件时，由于只有一个大文件因此只有一个worker在读取大文件，而其他worker空闲，因此进程数量对时间开销影响不大。读取大文件实验结果如下图所示。

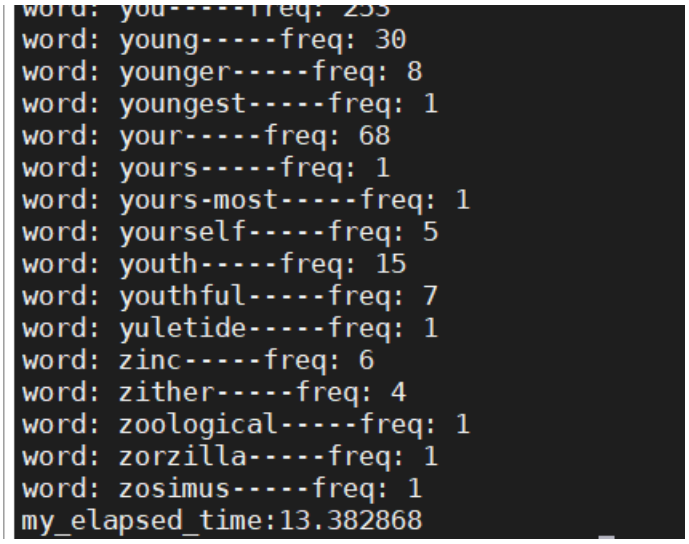

图 12: 使用64个进程统计多个小文件的单词个数的实验结果。

实验总结 MPI是主要基于通信的并行方式，进程之间互相通信，以达到合作的关系。在一些通信较为复杂得情况下，比如使用cannon算法实现矩阵相乘，效率比较高。但是在一些需要通信不多而均衡负载显得更重要得情况下效率就不是很高，比如许多文件得单词统计中，由于文件长短不同，这时候均衡负载就显得比较重要了。