并行与分布式作业

第五次作业

姓名：陈琮昊
班级：人工智能与大数据
学号：18340013

一、问题描述：

1. Consider a sparse matrix stored in the compressed row format (you may find a description of this format on the web or any suitable text on sparse linear algebra). Write an OpenMP program for computing the product of this matrix with a vector. Download sample matrices from the Matrix Market and test the performance of your implementation as a function of matrix size and number of threads.
2. Implement a producer-consumer framework in OpenMP using sections to create a single producer task and a single consumer task. Ensure appropriate synchronization using locks. Test your program for a varying number of producers and consumers.
3. 利用 MPI 通信程序测试本地进程以及远程进程之间的通信时延和带宽.

二、实验结果：

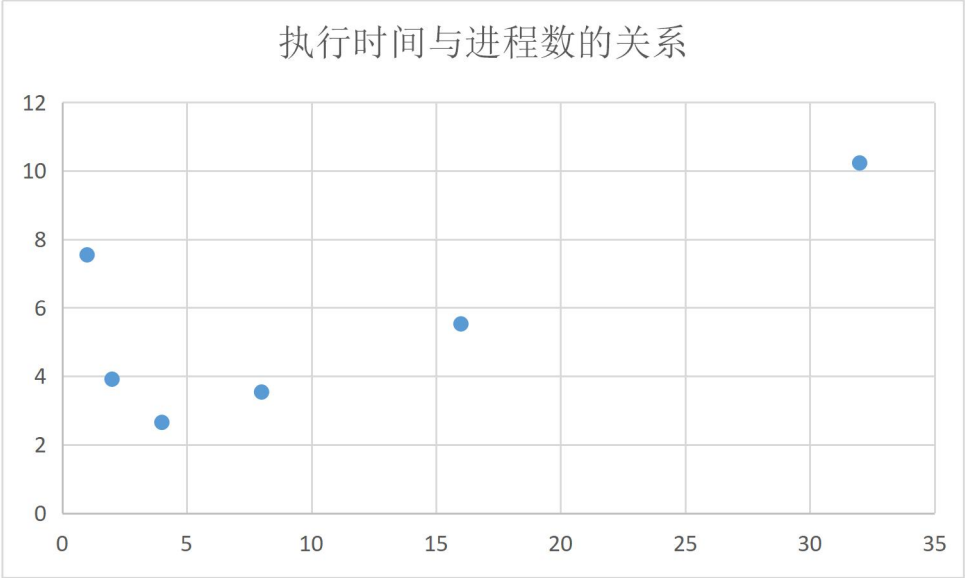
1. 代码见 1138matrix.cpp 文件，可执行文件为 1138matrix.out。
需要注意的是，稀疏矩阵的存储方式是行，列，元素，因此在读入存有稀疏矩阵的文件时要按照其格式进行读入：

```
ifstream fin("1138_bus.mtx");
while (fin.peek() == '%')
    while (fin.get() != '\n')
        ;
fin >> m >> n >> th;
Matrix mat(m);
for (int x, y, i = 0; i < th; i++)
{
    double t;
    fin >> x >> y >> t;
    mat[x - 1].emplace_back(y - 1, t);
}
Vec vect(n);
for (int i = 0; i < n; i++)
    vect[i] = rand();
auto begin = std::chrono::system_clock::now();
for (int i = 1e5; i; i--)
    vect *mat;
auto end = std::chrono::system_clock::now();
std::chrono::duration<double> elapsed_seconds = end - begin;
```

向量则随机生成。一次乘法的时间太小，所以输出的是 100000 次乘法所需时间。
（1）首先测试进程数对性能的影响，因此矩阵大小要保持不变，在这里就选取 1138X1138 这个矩阵进行测试。结果如下：

```
chench@LAPTOP-TOEITVUA: $ ./1138matrix.out
please input the number of threads: 1
elapsed time: 7.54218s
chench@LAPTOP-TOEITVUA: $ ./1138matrix.out
please input the number of threads: 2
elapsed time: 3.91123s
chench@LAPTOP-TOEITVUA: $ ./1138matrix.out
please input the number of threads: 4
elapsed time: 2.64988s
chench@LAPTOP-TOEITVUA: $ ./1138matrix.out
please input the number of threads: 8
elapsed time: 3.53598s
chench@LAPTOP-TOEITVUA: $ ./1138matrix.out
please input the number of threads: 16
elapsed time: 5.52602s
chench@LAPTOP-TOEITVUA: $ ./1138matrix.out
please input the number of threads: 32
elapsed time: 10.2262s
```

将这些数据导入 Excel，可以观察到大致趋势：



可以看到，进行矩阵乘法所需的时间在使用 4 个线程的时候达到最小值，也就是性能最佳；此后随着线程数的上升，其性能不断下降。可能是由于本机的 CPU 为 4 核，当线程数超过 4 时，线程之间的通信开销就更耗费时间，因此会导致性能下降。

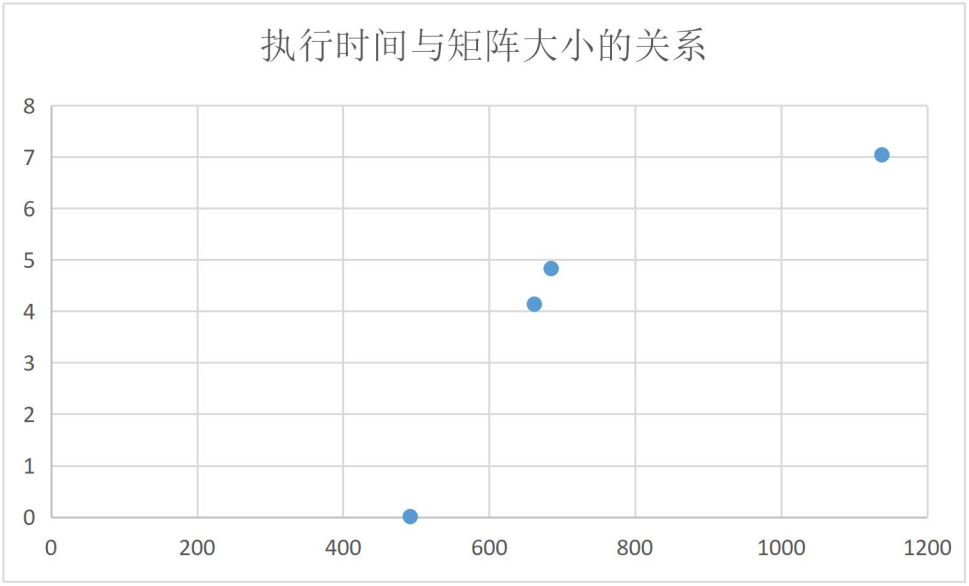
（2）接下来是矩阵大小对性能的影响，在该网站下载其他大小的稀疏矩阵，且固定线程数为 4，进行测试：

```
//ifstream fin("492_bus.mtx");
//ifstream fin("662_bus.mtx");
//ifstream fin("685_bus.mtx");
ifstream fin("1138_bus.mtx");
```

结果如下：

```
chench@LAPTOP-TOEITVUA: ~ $ g++ -o 1138matrix.out 1138matrix.cpp
chench@LAPTOP-TOEITVUA: ~ $ ./1138matrix.out
please input the number of threads: 4
elapsed time: 0.0074153s
chench@LAPTOP-TOEITVUA: ~ $ g++ -o 1138matrix.out 1138matrix.cpp
chench@LAPTOP-TOEITVUA: ~ $ ./1138matrix.out
please input the number of threads: 4
elapsed time: 4.13632s
chench@LAPTOP-TOEITVUA: ~ $ g++ -o 1138matrix.out 1138matrix.cpp
chench@LAPTOP-TOEITVUA: ~ $ ./1138matrix.out
please input the number of threads: 4
elapsed time: 4.82866s
chench@LAPTOP-TOEITVUA: ~ $ g++ -o 1138matrix.out 1138matrix.cpp
chench@LAPTOP-TOEITVUA: ~ $ ./1138matrix.out
please input the number of threads: 4
elapsed time: 7.03895s
```

将上述数据导入 Excel，可以观察到大致趋势：



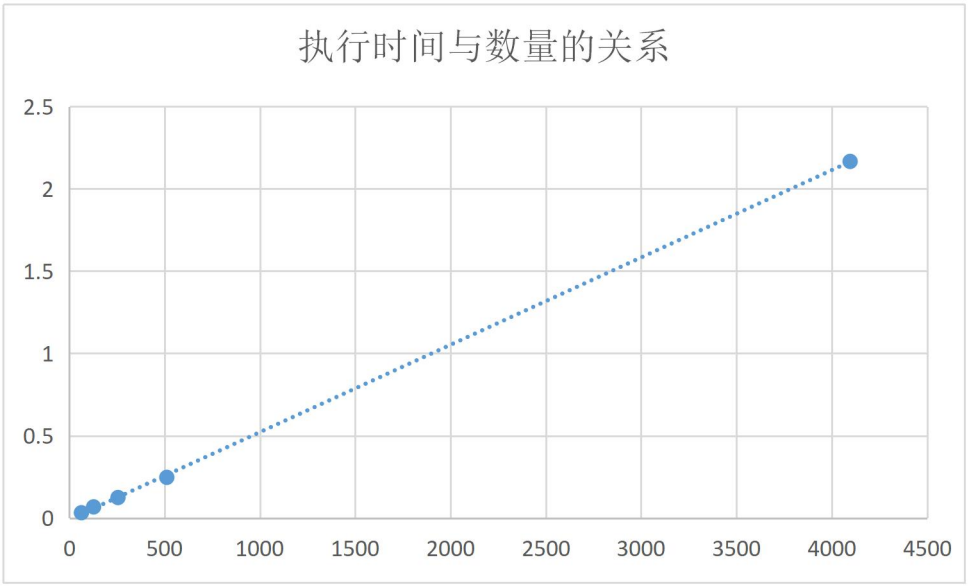
可以看到随着矩阵大小的增加，消耗时间增加，这符合我们的预期。

2. 代码见 2 文件夹，里面包含 3 个文件，一个头文件 Queue.hpp 和测试文件 queue.cpp，可执行文件为 queue.out：

编译时使用如下指令即可：`g++ queue.cpp -o queue.out -fopenmp`

结果如下：

```
chench@LAPTOP-TOEITVUA: ~ $ ./queue.out
please input the number of producer-consumers: 64
elapsed time: 0.0320719s
chench@LAPTOP-TOEITVUA: ~ $ ./queue.out
please input the number of producer-consumers: 128
elapsed time: 0.0675306s
chench@LAPTOP-TOEITVUA: ~ $ ./queue.out
please input the number of producer-consumers: 256
elapsed time: 0.124329s
chench@LAPTOP-TOEITVUA: ~ $ ./queue.out
please input the number of producer-consumers: 512
elapsed time: 0.246867s
chench@LAPTOP-TOEITVUA: ~ $ ./queue.out
please input the number of producer-consumers: 4096
elapsed time: 2.16576s
```



可以看到趋势成一条直线。

3. 本项任务只完成了本地进程的测试，未完成远地进程之间的测试。代码见文件夹 3 内的 `mpi.c` 文件。

配置的环境为 `MPICH`，在 `Linux` 下键入如下指令安装即可：

```
sudo apt install -y build-essential mpich
```

安装好后，编译时采用如下指令：

```
mpicc -o mpi.out mpi.c
mpirun -np 4 ./mpi.out
```

然后可以得到结果：

```
chench@LAPTOP-T0EITVUA: $ mpicc mpi.c -o mpi.out
chench@LAPTOP-T0EITVUA: $ mpirun -np 4 ./mpi.out
--------------------------------------------------------------------------
WARNING: Linux kernel CMA support was requested via the
btl_vader_single_copy_mechanism MCA variable, but CMA support is
not available due to restrictive ptrace settings.

The vader shared memory BTL will fall back on another single-copy
mechanism if one is available. This may result in lower performance.

  Local host: LAPTOP-T0EITVUA
--------------------------------------------------------------------------
MPI world size: 4
Test message size: 100000
MPI_Wtick resolution: 1.000000e-07
Task 0 is on LAPTOP-T0EITVUA with partner 2
Task 1 is on LAPTOP-T0EITVUA with partner 3
Task 2 is on LAPTOP-T0EITVUA with partner 0
Task 3 is on LAPTOP-T0EITVUA with partner 1
Task pair: 0 - 2: best: 11111.111104, avg: 8627.502194, worst: 477.099237, time: 0.000037
Task pair: 1 - 3: best: 11235.955077, avg: 7710.244760, worst: 675.447484, time: 0.000036
Total avg: best: 11173.533091, avg: 8168.873477, worst: 576.273360, time: 0.000037
```

可以看到，在单节点通信传输 `100000` 字节的数据，传输时延平均为 `0.000037 s`，计算可得平均带宽为 `6400 MBps`。

三、实验心得：

本次实验的前两个任务相对来说较简单，帮助我更好的理解了 `openMP`，也更能熟练的使用 `openMP`。本次实验的最后一个任务比较难，第一次接触 `MPI` 这种消息传递式的并行编程模型，在听过老师讲解的基础上，自己又在网上查找了一些资料；配置环境比较容易，一行指令就能解决；编写代码就没那么容易了，需要十分了解这个模型才能够比较顺利的完成。同时也看到了与其他学科的结合，网络课上的知识对我实现这个内容有一定帮助，尽管不是很大也不能忽视。有些不

理解的地方通过与同学交流也慢慢搞明白了是怎么回事。通过使用 MPI_Send，MPI_Recv 等一些函数进行本地的简单消息通信，最终完成了测试时延和带宽。