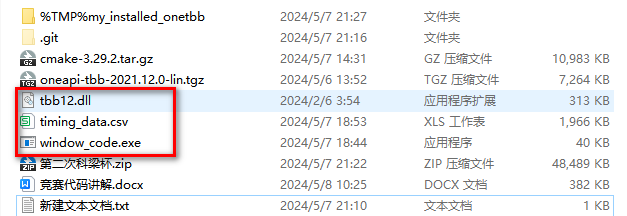
第二题讲解：

这段代码的核心是通过并行处理模拟数据传输与接收过程，并测量每个线程任何一次“发送数据+接收数据”的数据传输时间,以下是程序的安装和测试。

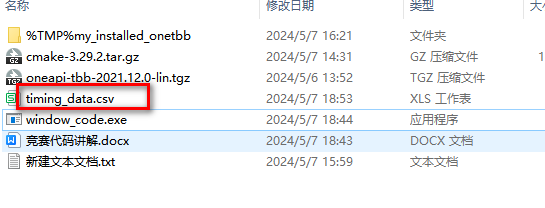
安装和运行：

Window系统：

需要tbb.12dll库

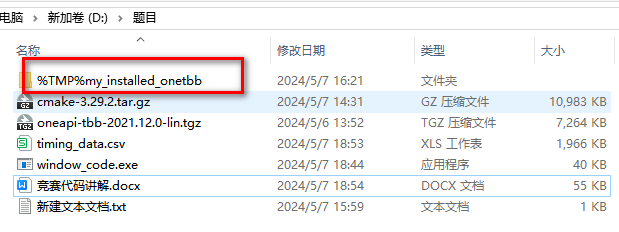


Window只需要双击之后便可以生成csv，存储发送和接收的总时间



Linux:

1:首先需要我已经编译好的TBB库，放入linux tmp文件夹中



2：更新自带系统gcc升级到gcc9

以下是在 CentOS 上安装特定版本的 GCC 和 GCC C++ 的步骤：

1)：\*\*安装 SCL：\*\*首先，确保你的系统上已经安装了 SCL。如果没有，你可以通过以下命令安装：

sudo yum install centos-release-scl

2)\*\*安装指定版本的 GCC：\*\*使用 yum 安装指定版本的 GCC 和 GCC C++。例如，安装 GCC

sudo yum install devtoolset-9-gcc devtoolset-9-gcc-c++

3)\*\*启用指定版本的 GCC：\*\*安装完成后，默认情况下并不会激活新安装的 GCC 版本。你需要通过 scl enable 命令来激活：

scl enable devtoolset-9 bash

这将激活 GCC 9，并将你的当前 shell 环境配置为使用 GCC 9。

4)\*\*验证版本：\*\*验证 GCC 是否成功安装并激活。你可以使用以下命令来检查 GCC 版本：

gcc --version

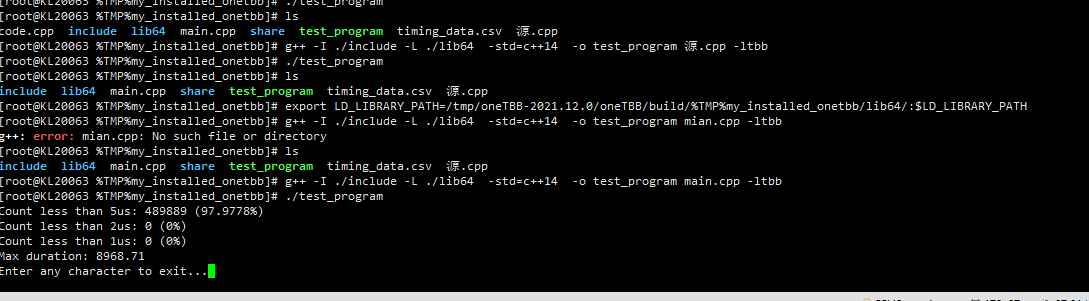
5)安装：g++

sudo yum install gcc-c++

在 %TMP%my\_installed\_onetbb目录下：

export LD\_LIBRARY\_PATH=/tmp/oneTBB-2021.12.0/oneTBB/build/%TMP%my\_installed\_onetbb/lib64/:$LD\_LIBRARY\_PATH 指定库文件之后注意目录为自己的目录%TMP%my\_installed\_onetbb

然后运行之后会有test\_program文件生成,在运行该文件则会生成csv文件



csv数据查看：使用wps功能

目前由于设备硬件问题配置不高测试结果如下

目前linux 虚拟机 2核 ,10进程，5万次该环境下 非空载状态：

5微秒以下数据：98.3%

2微妙以下：暂无

1微秒以下：暂无

Window 虚拟机 4核: 8RAM Intel(R) Core(TM)2 Duo CPU T7700 @ 2.40GHz 2.90 GHz 非空载状态

5微秒以下数据：98.6%

2微妙以下：96.4

1微秒以下：88.8%

表格中数据为 微妙

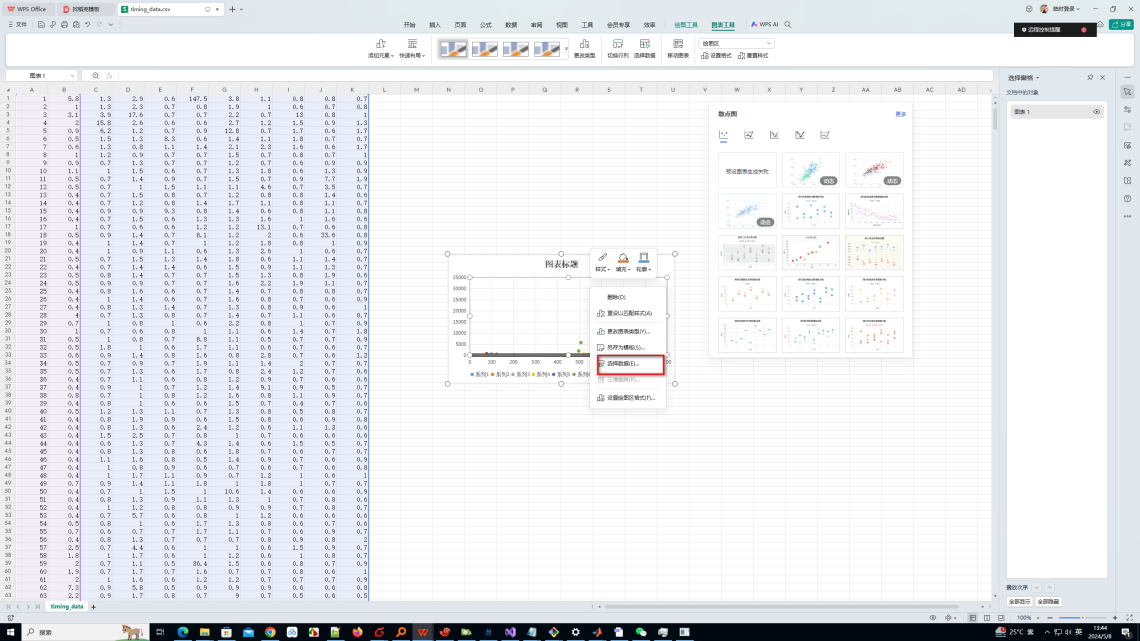
Linux 和window 如果计算机配置较高可以基本满足需求，需进行测试

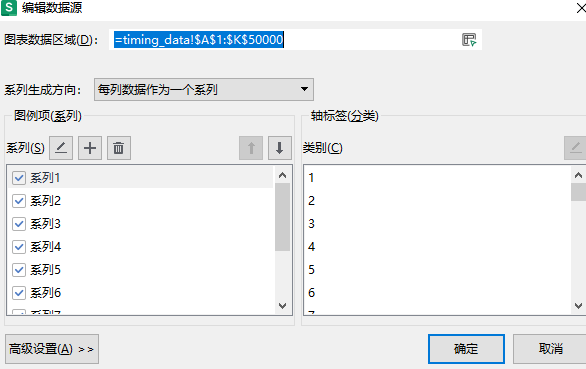
第一列是循环次数

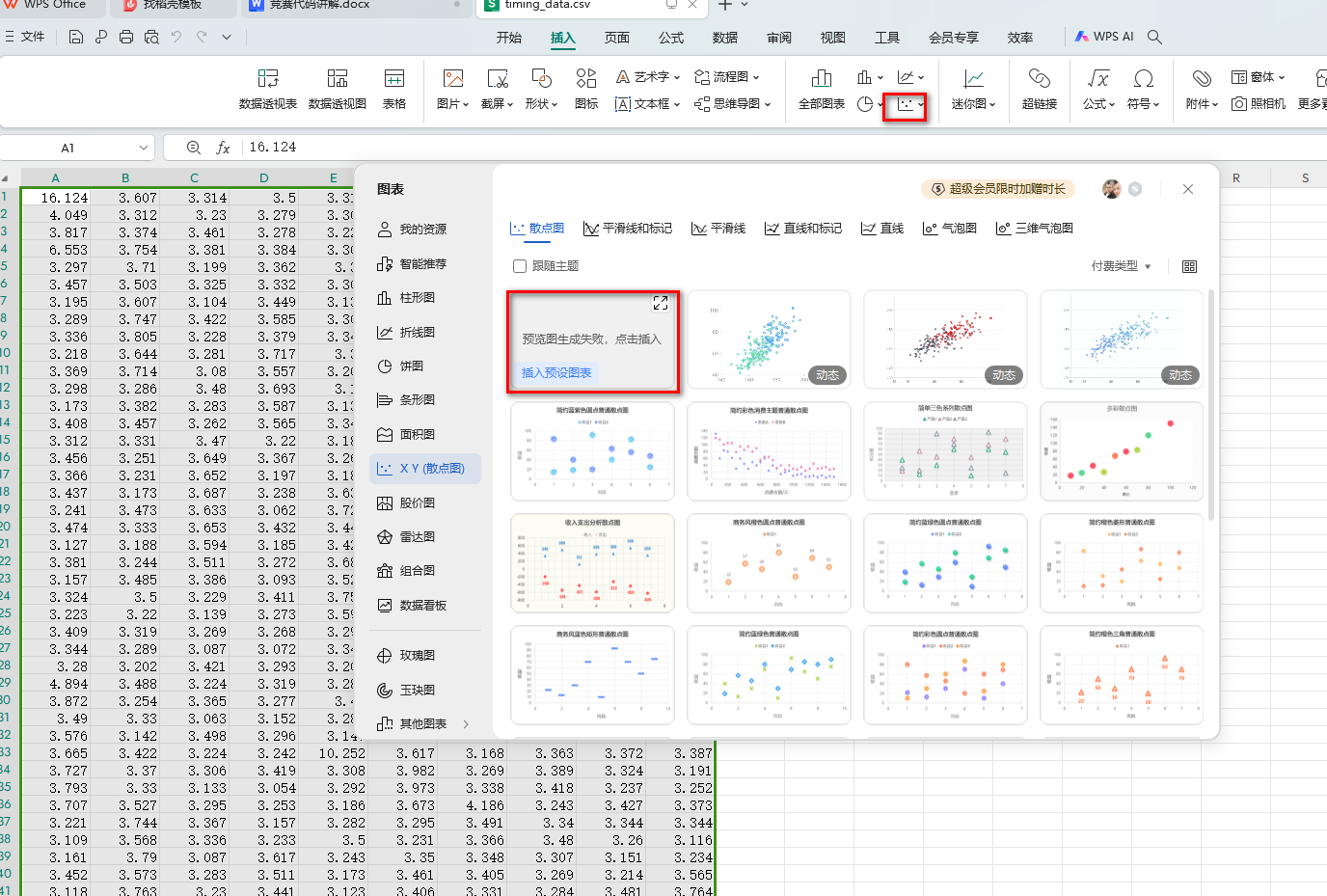
第一列进程1的接收数据

第二列是进程2的接收数据依次

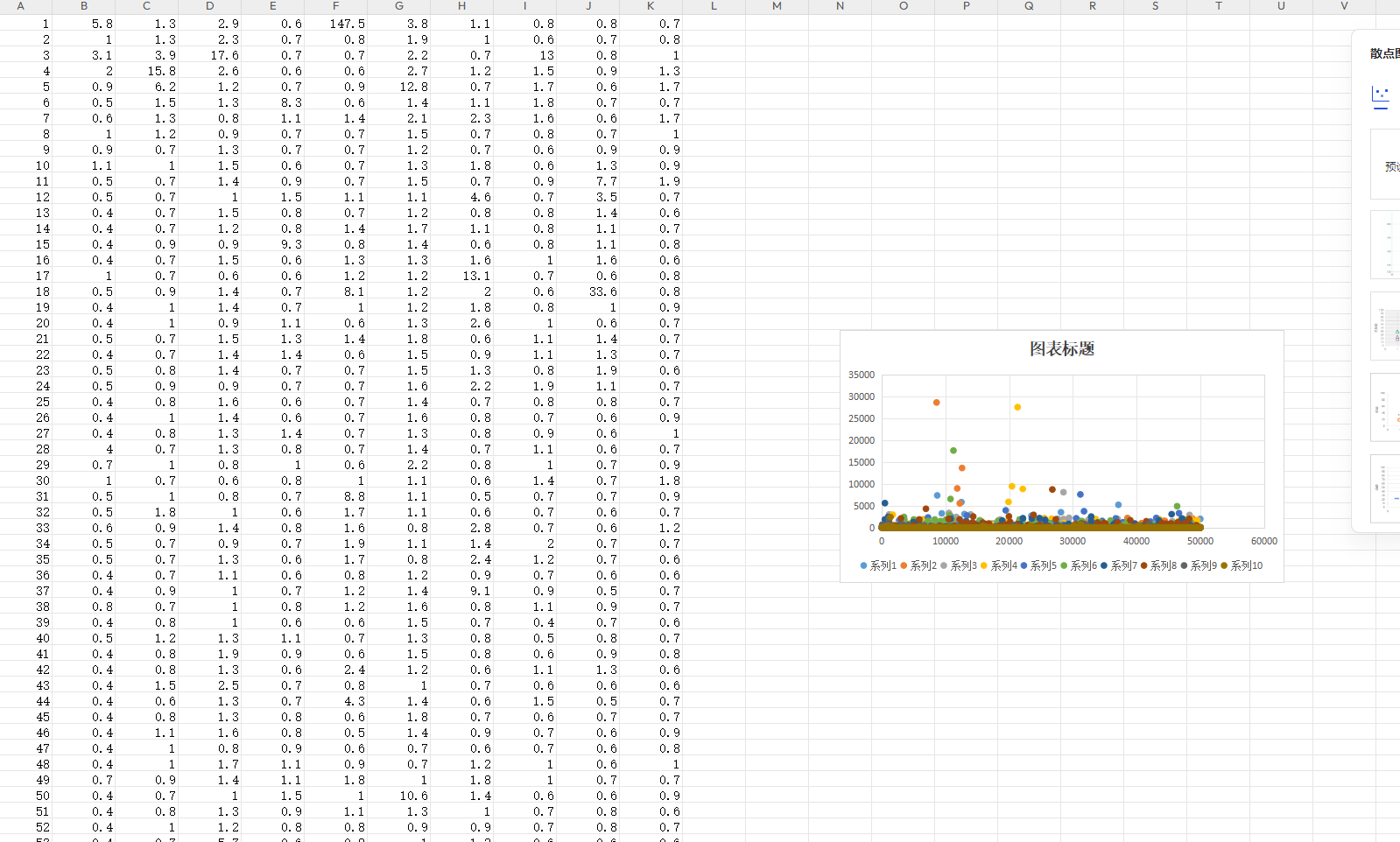
选择数据：



填入图标区域：=timing\_data!$A$1:$K50000



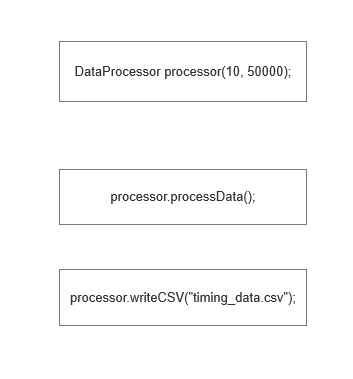
最后：



具体设计思路：

1:程序设计思路

总计三个函数：



这三个函数是在一个名为 DataProcessor 的类中的成员函数。

1:DataProcessor processor(10, 50000); 这行代码创建了一个名为 processor 的 DataProcessor 类型的对象，并将参数 10 和 50000 传递给它的构造函数。具体来说，这个构造函数接受两个参数：threads 表示要使用的线程数，iterations 表示要执行的迭代次数。因此，这行代码的作用是创建了一个具有 10 个线程和执行 50000 次迭代的 DataProcessor 对象。

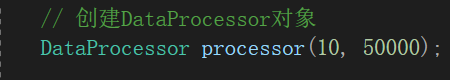
2:processor.processData() 是 DataProcessor 类中的一个成员函数，用于处理数据。这个函数通过多线程并行处理数据，每个线程执行 num\_iterations 次迭代。在每次迭代中，它生成随机数并将其加入到线程安全的并发队列中，然后等待其他线程也完成数据加入，接收其他线程加入的数据，并记录发送和接收数据的持续时间。

3:processor.writeCSV("timing\_data.csv") 是 DataProcessor 类中的另一个成员函数，用于将处理数据后的结果写入到一个 CSV 文件中。它将记录的发送和接收数据的持续时间写入到名为 "timing\_data.csv" 的 CSV 文件中，每一行代表一个迭代，每一列代表一个线程的发送或接收数据的持续时间。

设计逻辑：

1. 使用多线程模拟数据传输：程序创建10个线程，每个线程生成随机数据，每个线程5万次循环，并将其发送给下一个线程，使用并行处理库 TBB：通过 tbb::parallel\_for 管理并行线程，确保所有线程同时处理数据，使用线程安全队列传输数据

10个进程5万次配置：



2)采用 TBB 的 tbb::concurrent\_queue 实现线程间数据传输，使用互斥锁和条件变量进行线程同步：确保数据在多线程环境中安全传输，并确保线程等待条件满足时被唤醒，记录传输时间：通过高精度计时器记录线程间数据发送与接收的时间。

2:实现细节

1)初始化随机数生成器：每个线程生成1024个随机数用于模拟数据。

2)采用循环线程索引：每个线程将数据发送给下一个线程，最后一个线程发送给第一个线程，形成一个环形结构。

3)使用互斥锁确保数据安全：在发送和接收数据时，使用 std::mutex 确保线程间数据操作的安全性。

4)使用条件变量控制线程等待和唤醒：发送数据后，使用 std::condition\_variable 唤醒等待的线程。

5)记录数据传输时间：使用高精度计时器记录每个线程的发送和接收时间，存储在二维数组中,最后计算从发送到接收的时间将纳秒转化成微妙

6)写入CSV文件：将记录的时间数据写入CSV文件，方便后续分析和处理。

测试用例

1：基本功能测试：检查程序是否能够创建10个线程并执行指定次数的迭代，最终生成CSV文件,要求generateData、sendData、recvData能调用C/C++（C++14标准）函数。运行时采用gcc c++1运行符合要求，测试通过。  
g++ -I ./include -L ./lib64 -std=c++14 -o test\_program mian.cpp -ltbb

2：支持跨平台：代码可以跨平台运行，包括Windows和Linux,可参考 安装和运行。

2：时间数据记录测试：确保每次发送和接收数据的时间能够准确记录在二维数组中。

通过互斥锁和条件变量保持线程安全，输出最大值和5微妙，2微妙，1微秒以下的数据，测试通过。

3：CSV文件输出测试：确保生成的CSV文件包含正确的数据，并符合预期的格式，测试通过。

4：并行性能测试：未出现死锁，测量整体程序的运行时间，分析在多线程环境下数据传输的性能。通过以上测试，程序应该能够正确模拟多线程数据传输，并提供准确的时间记录，测试通过。

5：目前由于设备硬件问题配置不高测试结果如下

目前linux 虚拟机 2核 ,10进程，5万次该环境下 非空载状态：

5微秒以下数据：99.3%

2微妙以下：暂无

1微秒以下：暂无

Window 虚拟机 4核: 8RAM Intel(R) Core(TM)2 Duo CPU T7700 @ 2.40GHz 2.90 GHz 非空载状态

5微秒以下数据：98.6%

2微妙以下：96.4

1微秒以下：88.8%

表格中数据为 微妙

预计Linux 和window 如果计算机配置较高可以基本满足需求。