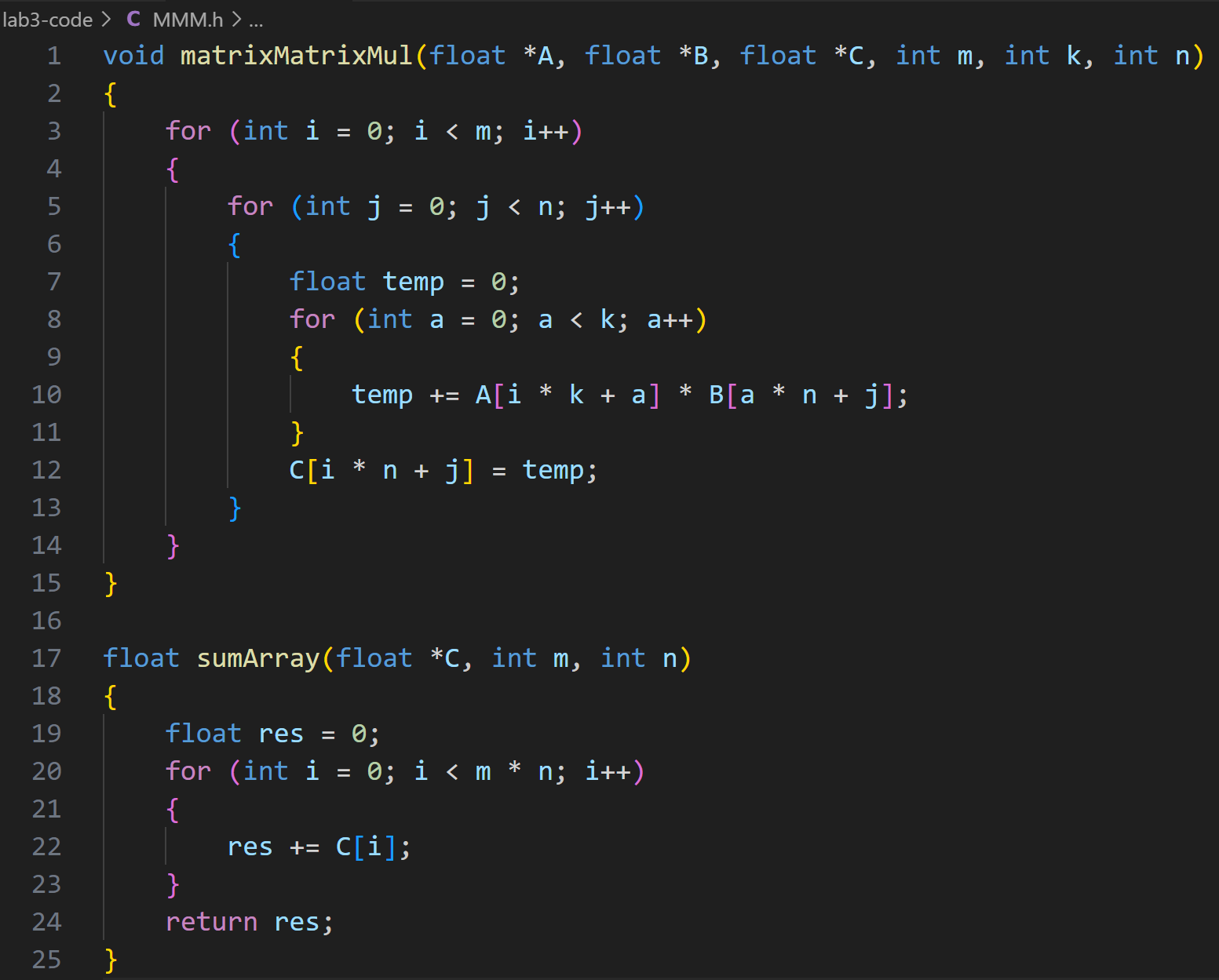
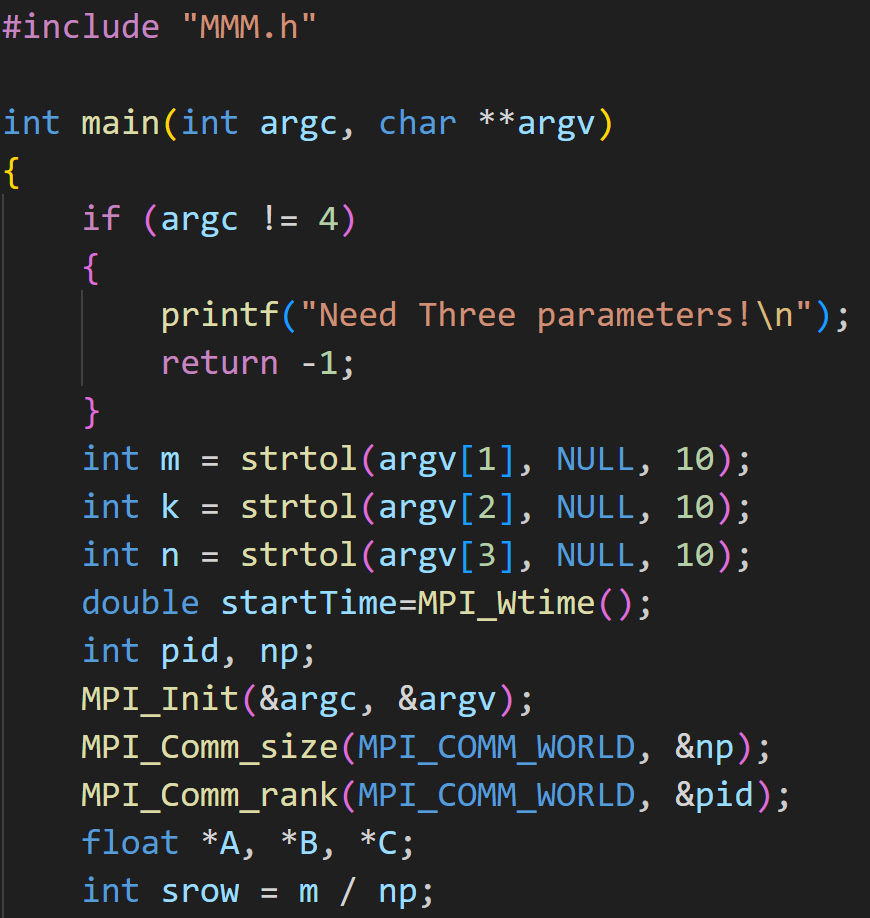
# 问题1

## 实验思路

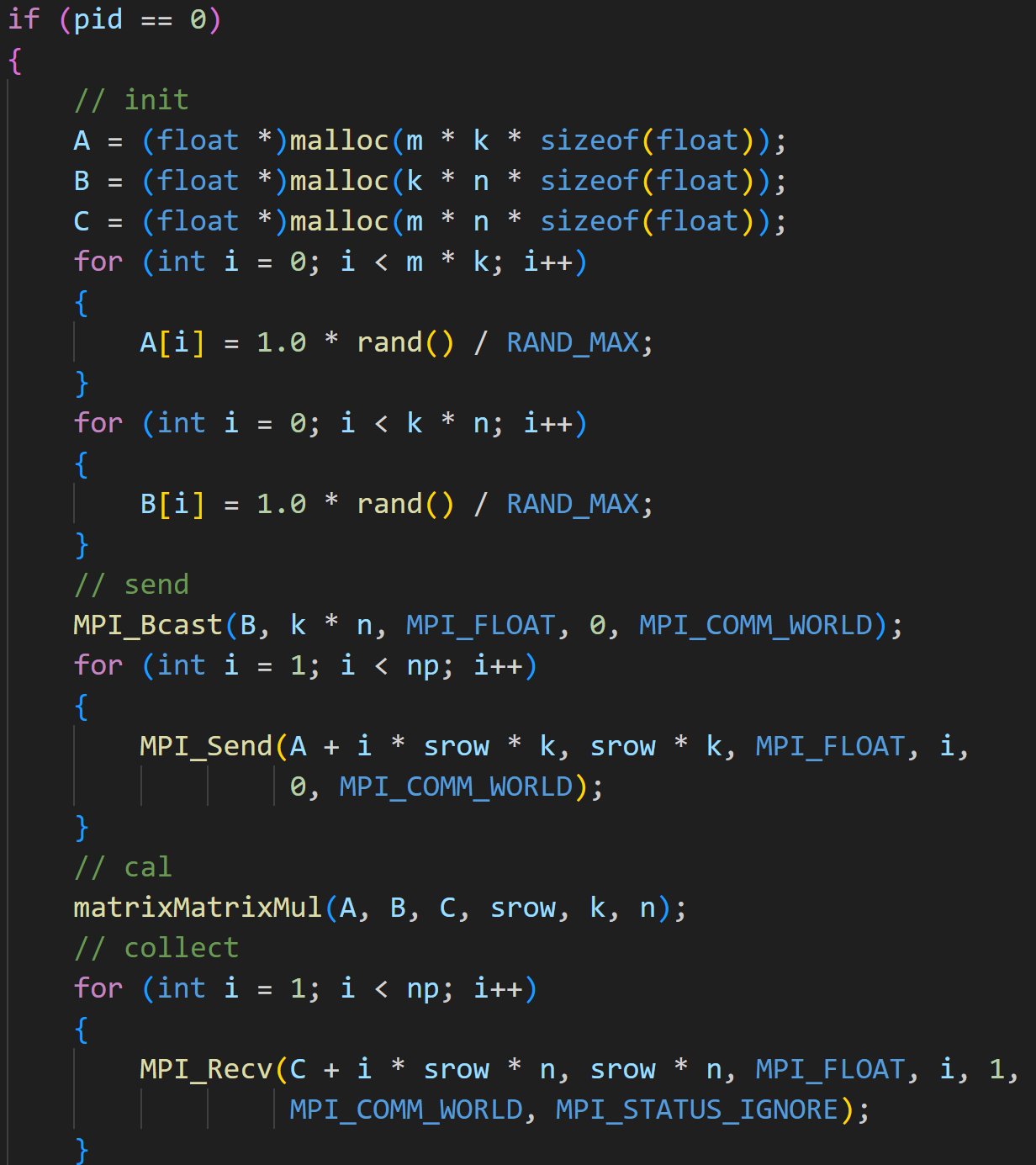
单CPU方法和MPI方法写在一个文件中，会导致程序代码过于复杂，因此分开编写两个程序。两个程序共用的函数放在一个头文件中以便调用，其中matrixMatrixMul为计算两个矩阵的乘积，sumArray为对矩阵的值求和，目的是验证计算结果，代码如下图所示。



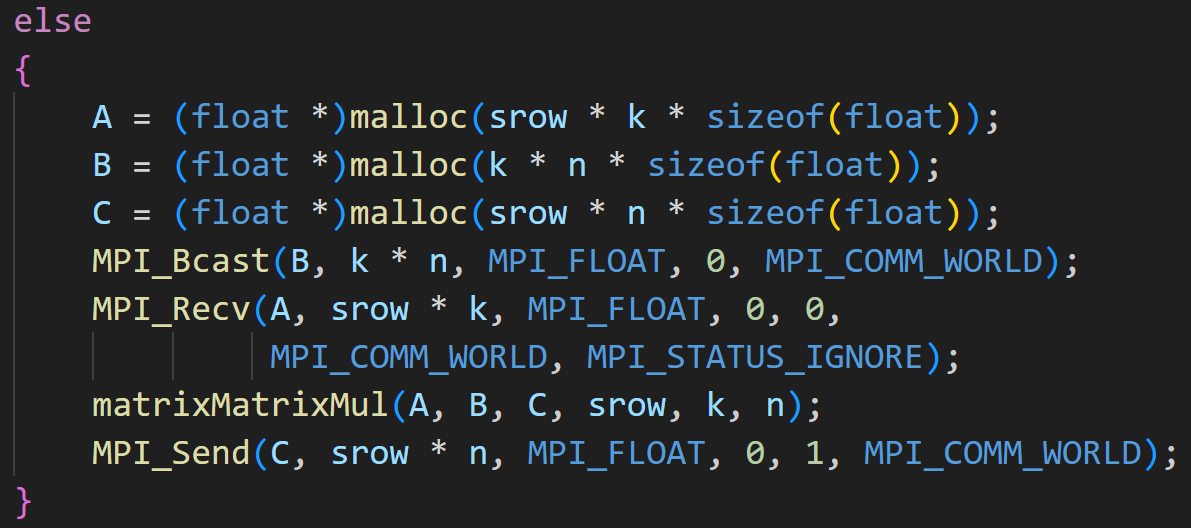
引入头文件，读取参数后，进行MPI初始化，对矩阵A按行划分，srow为每个进程的计算的行数。



对于master节点，首先分配矩阵的空间，然后初始化矩阵A和B的值。接着，把B矩阵以广播的形式发送给其他进程，并发送线程需要计算的矩阵A的行数据。master节点只计算自己需要计算的部分，最后，收集其他线程计算的结果到C矩阵中。



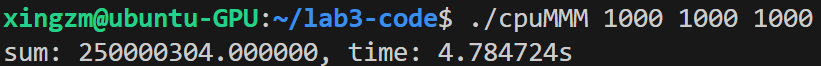
对于slave节点，B矩阵需要被分配所有空间，而A、C矩阵只被分配需要计算的行。计算后将结果向量发送给master，这里设置tag为1便于匹配。



最后，计算C矩阵的和来判断两种方法执行的结果是否一致。对于获取时间的函数，单CPU中使用了clock()，MPI中使用了MPI\_Wtime()。

## 实验结果

测试1000\*1000的两个矩阵相乘所需时间。在单CPU上计算，用时4.78s，而使用了2进程的MPI方法后，用时2.51s，提升将近2倍。根据sum的值也可以看到计算的结果是一致的。





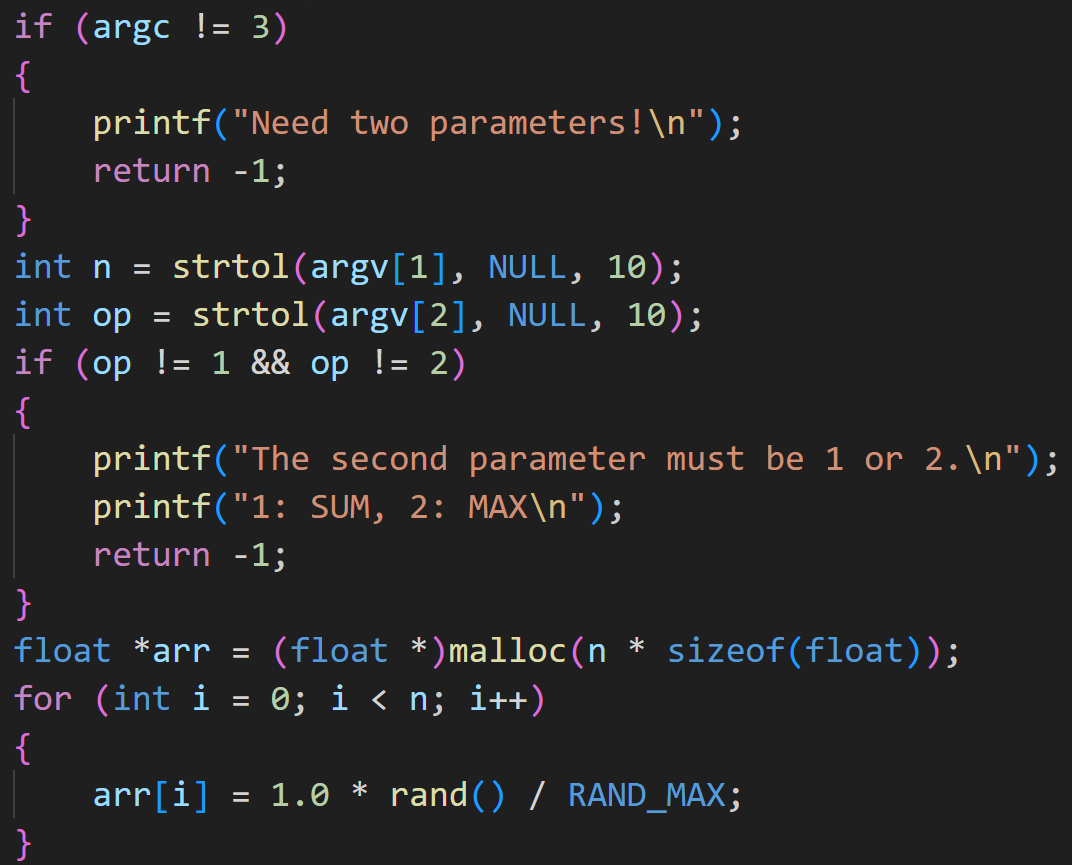
当把线程数设置为20后，计算仅用0.47s，比单CPU方法提升了10倍。



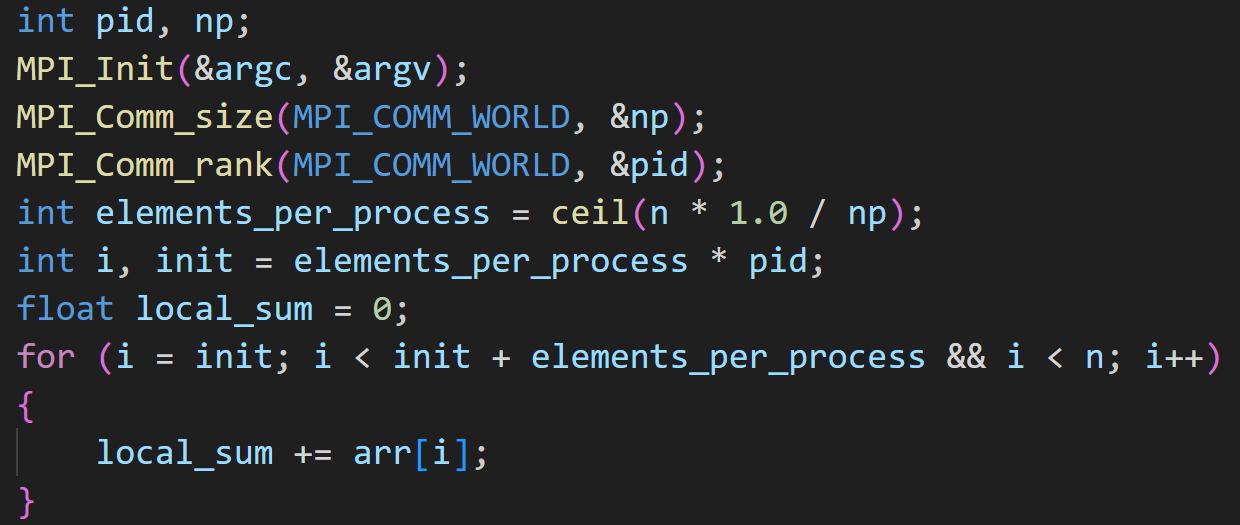
# 问题2

## 实验思路

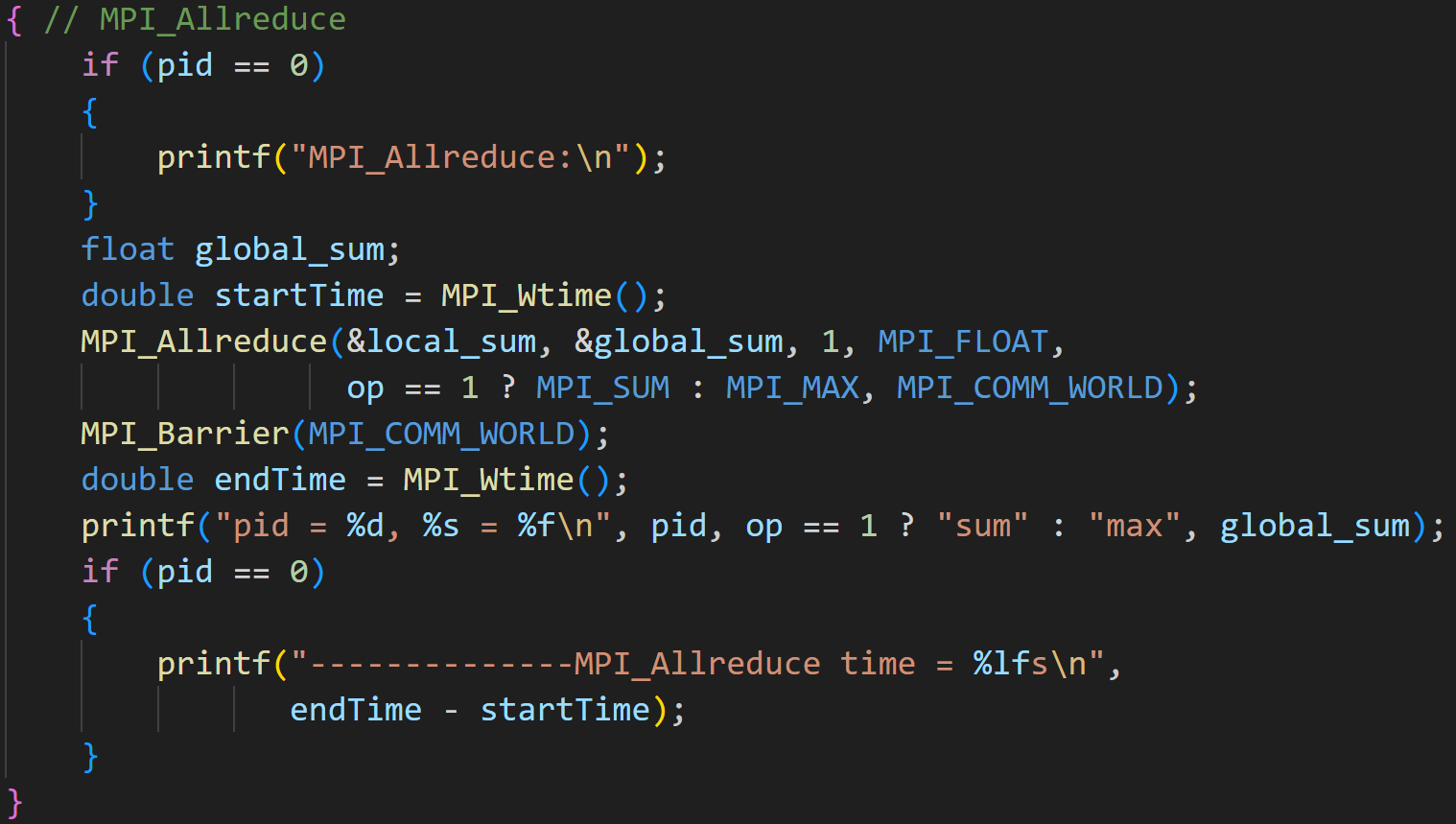
首先，程序读取命令行参数，输入数组大小和归约函数。这里数字1代表求和，数字2代表求最大值。读取参数后，为数组随机生成浮点数。



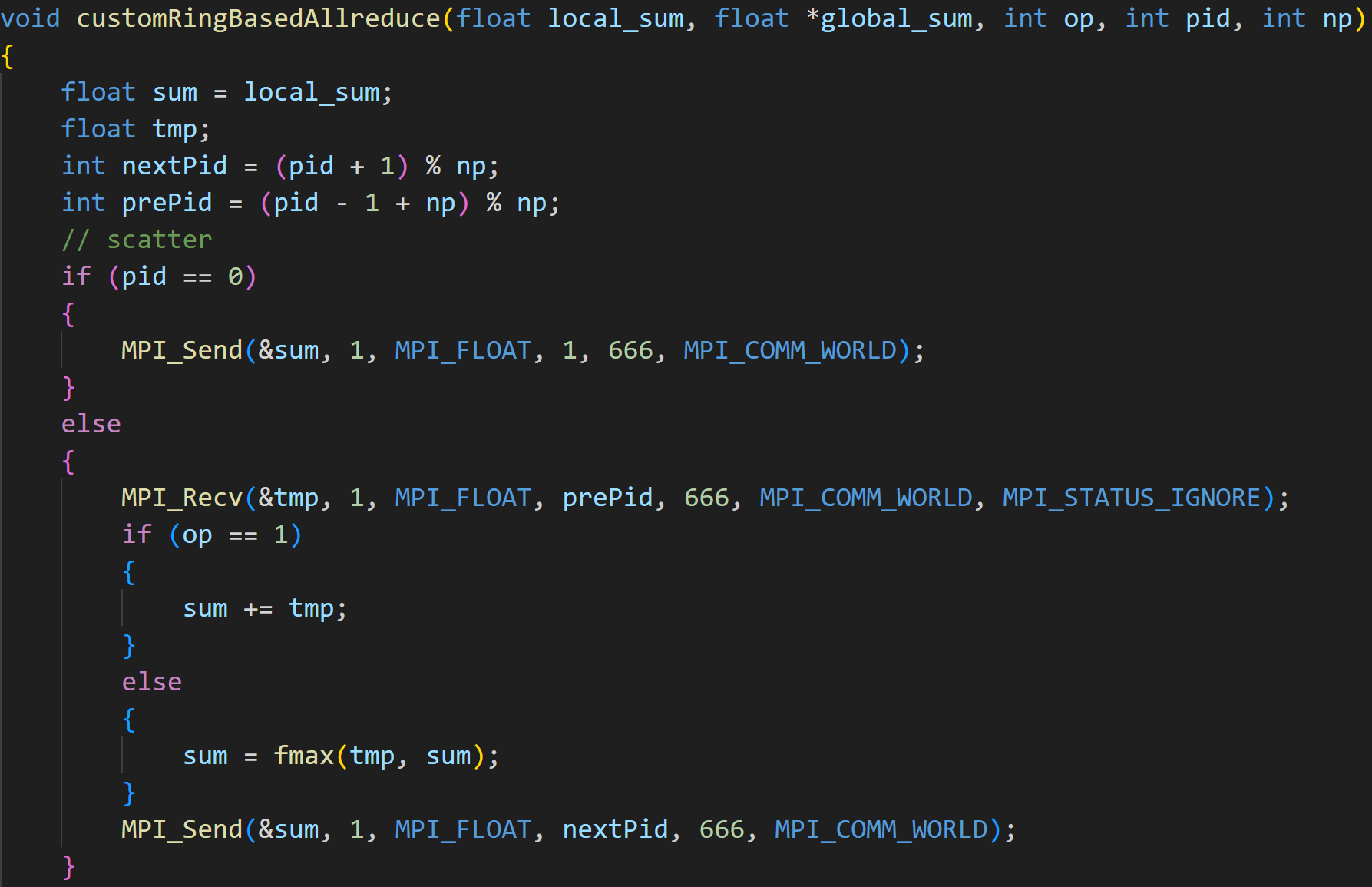
划分进程，每个进程中并求部分和。



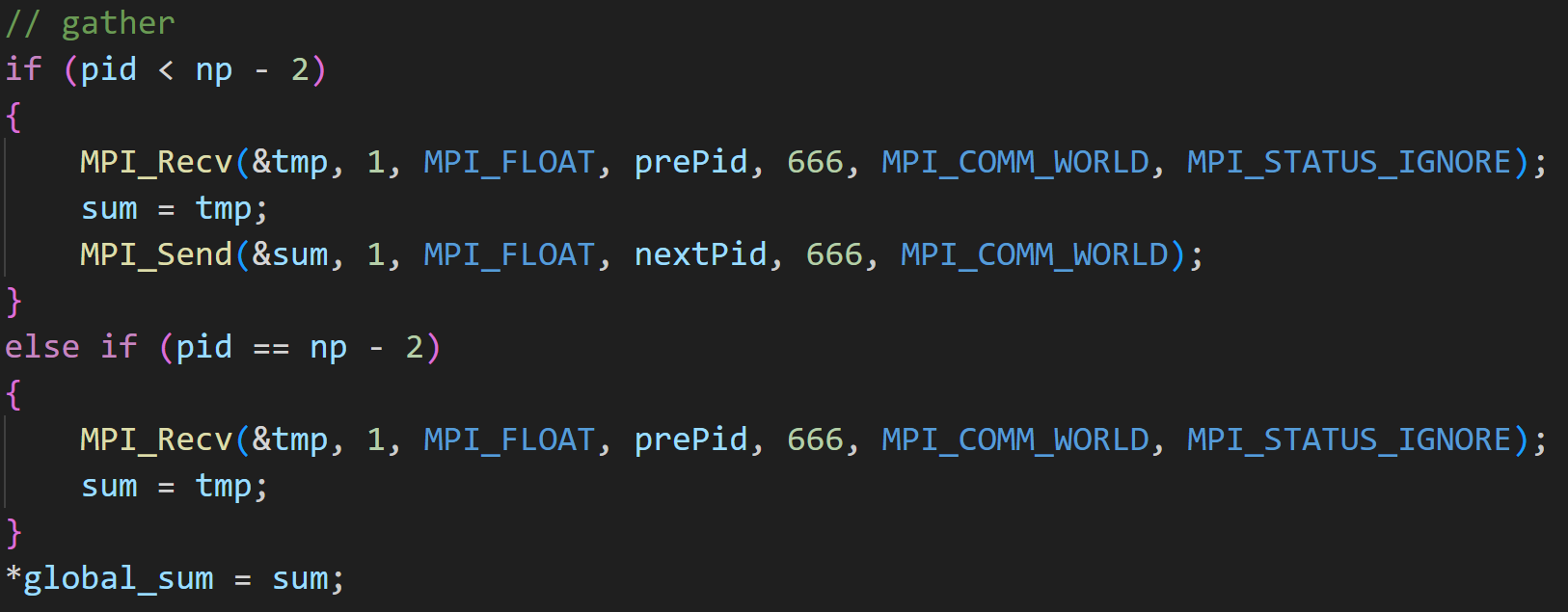
首先使用OpenMPI提供的MPI\_Allreduce方法，计算数组的和或者最大值。各个线程均打印结果，线程0额外打印运行时间，运行时间计算的是从执行归约操作开始，到所有线程归约结束的时间。



在自己设置的基于环的归约算法中，首先从左到右进行叠加求解，一个进程使用MPI\_Recv接收前一个线程的数据，进行求和或者最大值的处理，然后使用MPI\_Send发送给下一个线程。使用取模运算达到环形数组的效果。由于进程0不从其他进程接收数据，为了防止死锁，需要对进程0进行特殊处理。

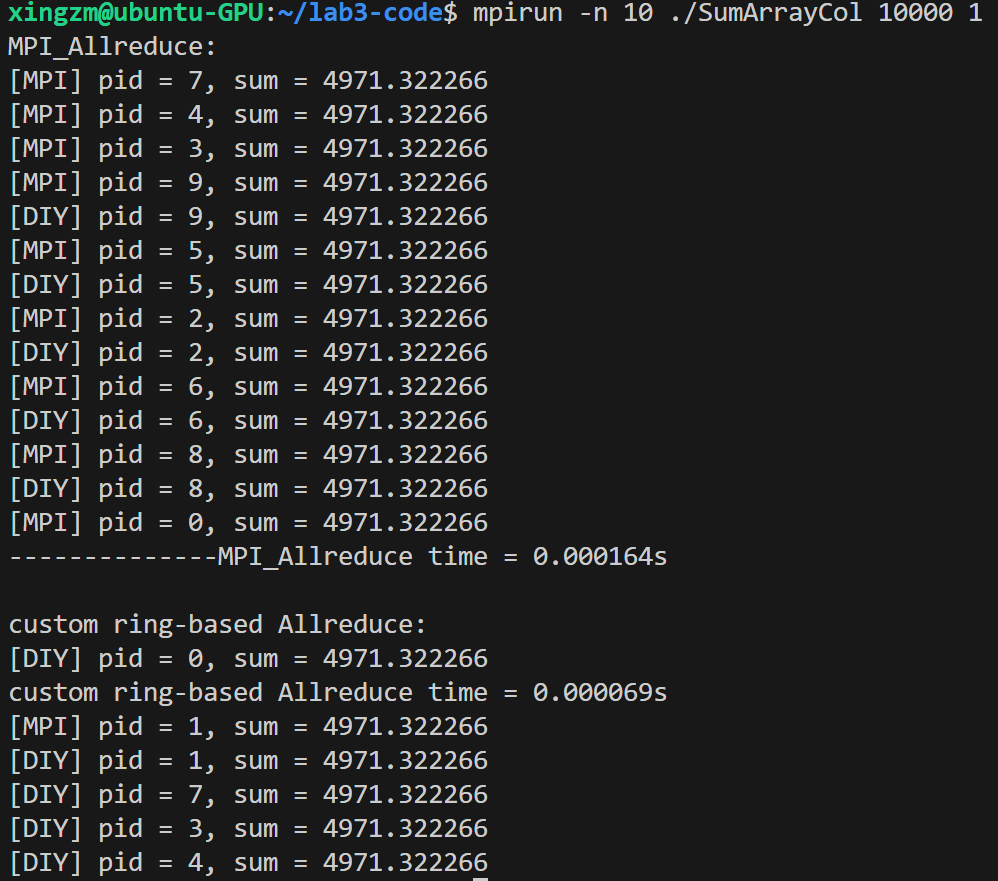


此时，最后一个进程的值为前面归约得到的最终结果。接下来，再次从线程0开始，获取前一个进程的最终结果，设置为自己的最终结果，达到了每个线程都归约到相同的值的目的。这里只需要计算到倒数第2个线程，而且倒数第2个线程中无发送指令，需要特殊处理。



## 实验结果

使用10个进程来计算10000个浮点数的和，可以看到每个线程最后都输出了相同的结果。而且我们编写的基于环的归约操作与MPI\_Allreduce执行的结果一致，且我们编写的函数执行速度约是官方函数的2.4倍，猜测是因为我们编写的方法较为轻量，没有考虑方法的完整性，没有对复杂的业务逻辑进行处理。



对于数组取最大值功能，可以看到程序的执行结果相似。

