1. 题目描述

对一个二维数组进行 FFT 变换，数组的规模为 2^n * 2^m，数组的元素类型为单精度浮点数。n 和 m 均不超过 15。

命令行输入：两个正整数 n 和 m。

结果输出：FFT 变换后的数组，数组的每个元素由两个单精度浮点数构成，第一个浮点数是该元素的实部、第二个元素是该元素的虚部。

数据输入/输出时，二维数组按照行优先顺序线性化为一维数组。

2. 评测方法

1) 将 hw3 路径下的各文件复制到你的个人目录下
2) 将所开发的程序命名为 program.cpp，存储在你的个人目录下
3) 运行 make 编译你的程序
4) 运行 Evaluate 进行程序评测：./ Evaluate flag np n m
   a) flag：0 或者 1。0 表示仅运行你开发的程序，不进行性能评测；1 表示运行你开发的程序，并进行性能评测。
   b) np：mpi 程序的进程数，本系统上最多可以运行 128 个进程。
   c) n：一个正整数，表示输入数组的行数为 2^n。
   d) m：一个正整数，表示输入数组的列数为 2^m。

3. 其它说明

1) serial.cpp 是串行实现的参考代码。
2) 请在程序代码中插入下列头文件声明语句：

   #include "fio.h"

   该文件中定义了数据结构 FILE_SEG 和下列函数原型：

   int64_t input_data(void *buf, int64_t count, FILE_SEG fseg)

   int64_t output_data(void *buf, int64_t count, FILE_SEG fseg)

   评测系统将每个数组 A[]的二进制表示看作一个线性的字节序列(a sequence of bytes) bs_A[]，要求 MPI_COMM_WORLD 通信子中的每个进程分别输入/输出 bs_A[]的一个视图(view)。每个进程用一个 struct FILE_SEG 类型的三元组< offset width stride >描述它所输入/输出的视图。在同一个视图中，bs_A[]的每个元素最多出现一次，即 width 必须为正整数且不超过 stride。

```
struct FILE_SEG {
        int64_t offset;
        int64_t width;
        int64_t stride;
}
```

3) **int64_t input_data(void *buf, int64_t count, FILE_SEG fseg)**：输入数组 A 的一个视图到当前进程的内存空间中。每个进程在其生命周期中，最多允许调用一次 input_data(void *buf, int64_t count, FILE_SEG fseg)，并且必须在完成 MPI_init()之后、调用 MPI_finalize()

之前才能够调用 input_data()。不同进程所输入的视图可以存在重叠。评测系统接收到一个进程通过 input_data() 提交的数据输入请求后，将检查当前进程的 MPI_COMM_WORLD 通信子。只有 MPI_COMM_WORLD 的每个进程都调用了 input_data() 后，才能完成当前进程的数据输入。

-buf: 输入数据在内存空间的存储地址首址

-count: buf 所指向内存空间的字节数，应不小于被读视图包含的字节数

-fseg: 输入数据在数组 A 上的视图。

返回值：成功输入，则返回输入的字节数；否则，返回-1。成功返回后，bs_A[]中下标为[flb(k) fub(k))的片段被读入并存储在地址为[mlb(k) mub(k))的内存区域，令 fsize 表示 bs_A []的长度：

-k: 满足 fseg.offset+k*fseg.stride<fsize 的全部非负整数

-flb(k): fseg.offset + k*fseg.stride

-fub(k): flb(k) + min(fseg.width fsize - flb(k))

-mlb(k): buf + k * fseg.width

-mub(k): mlb(k) + fub(k) - flb(k)

4) **int64_t output_data(void *buf, int64_t count, FILE_SEG fseg)**：从当前进程的内存空间中输出数组 B 的一个视图。

-buf: 被输出数据在内存空间的存储地址首址

-count: 被输出数据的字节总数。若 count 为 0，则 buf 可以是任意值。

-fseg: 被输出数据在数组 B 上的视图。

返回值：成功输入，则返回输出的字节数；否则，返回-1。成功返回后，内存区域[mlb(k) mub(k))中的数据被复制到 bs_A[]中下标为[flb(k) fub(k))的片段上：

-k: 满足 k*fseg. width<count 的全部非负整数

-mlb(k): buf + k * fseg.width

-mub(k): mlb(k) + min(fseg.width count - mlb(k))

-flb(k): fseg.offset + k * fseg.stride

-fub(k): flb(k)+ mub(k)- mlb(k)

每个进程在其生命周期中，最多允许调用一次 output_data(void *buf, int64_t count, FILE_SEG fseg)，并且必须在完成 MPI_init()之后、调用 MPI_finalize()之前才能够调用 output_data()。评测系统接收到一个进程通过 output_data()提交的数据输出请求后，将检查当前进程的 MPI_COMM_WORLD 通信子。只有 MPI_COMM_WORLD 的每个进程都调用了 output_data()后，并且确认这些进程的输出数据视图合法后，才能完成当前进程的数据输出。对于通信子 MPI_COMM_WORLD，合法的数据视图需满足三个条件：

Cond.1：fseg.width<= fseg.stride

Cond.2：区域[0 fseg.offset)内的每个整数被唯一一个其他进程的输出视图覆盖

Cond.3：若(k+1)* fseg. width<count, [fseg.offset + k * fseg.stride + fseg.width fseg.offset + (k+1) * fseg.stride)内的每个整数被唯一一个其他进程的输出视图覆盖

5) program.cpp 是 MPI 并行程序进行数据输入、输出的参考代码，可以在该源码的基础上开发自己的并行程序。该源码的流程如下：

a) 输入数据：对输入数组进行条块分割，每个进程读入若干行。

b) 采用群通信，把输入数组以复制方式存储在各个进程上；由各个进程分别对整个输入数组执行串行 2 维 FFT 变换。

c) 输出数据：b)阶段的数据复制使得输出数组也是以复制方式存储在各个进程上。对

输出数组进行条块划分，第 k 块由第 k#进程输出。