一 MPI 基本点对点通信机制的分析题。

假设在编码“大”和“小” 实例的 BSP 并行算法时，只允许使用 MPI_Send/MPI_Recv 进行数据交换，如何确保问题规模和 MPI runtime 的性能不影响 MPI_Send 与 MPI_Recv 匹配成功?

二 编程练习题

**1. 题目描述**

在一个三维空间中，共有 2^N 个粒子。它们的运动规律定义如下:

(1) 在每一时刻，任意两个粒子之间存在一对作用力 F = G*M1*M2*d/r^3，其中 d 为这两个粒子之间的矢量距离，r 是这两个粒子之间的标量距离，G 是一个常量，M1 和 M2 分别是这两个粒子的质量。

(2) 假设一个粒子的质量为 m，它在 t 时刻的位置为( x(t) y(t) z(t) )，受到其他粒子的作用力为 F，则它在 t+1 时刻的位置( x(t+1) y(t+1) z(t+1) ) = ( x(t) y(t) z(t) ) + F/m。

已知各粒子的质量、以及它们在 0 时刻的位置，请计算各粒子在 2^T 时刻的位置。N 不超过 20，T 不超过 15。

数据输入输出时，每个粒子的状态用一个双精度浮点数的四元组<m x y z>表示，其中 m 表示粒子的质量，<x y z>表示粒子的空间位置坐标。全部粒子的状态一个长度为 2^(N+2)的双精度浮点数组表示。

命令行输入：两个整数 N 和 T。

结果输出：2^T 时刻各个粒子的位置。

**2. 评测方法**

1) 将 hw2 路径下的各文件复制到你的个人目录下
2) 其中的 program.cpp 是该问题的一个完整 MPI 程序。请阅读该程序，并修改其中的 void optimized_nbody(Real *pbd, int time_steps, int64_t *dist)函数，要求在该函数中只能调用 MPI_Sendrecv()函数进行数据交换。
3) 运行 make 编译你的程序
4) 运行 Evaluate 进行程序评测：./ Evaluate flag np N T
   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) size：一个正整数，表示粒子系统的粒子数 2^N
   d) steps：一个正整数，表示共模拟计算 2^T 个时间步。

**3. 其它说明**

1) serial.cpp 是串行实现的参考代码。
2) 为进行统一的正确性和性能评测，程序代码中应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 并行实现的参考代码。