并行程序设计与算法 第二次作业

刘森元, 21307289

中山大学计算机学院

1 简答题

1.1 习题 1

考虑教材中的问候程序 (程序 3.1), 如果把代码中的 `strlen(greeting) + 1` 换成 `strlen(greeting)` 来计算所发送消息的长度, 会发生什么情况?

通过实验证明, 在 Ubuntu 22.04 系统上, 两者并没有区别.

1. `strlen(greeting) + 1`

```
16      if (my_rank != 0) {
17          sprintf(greeting, "Greetings from process %d of %d!", my_rank, comm_sz);
18          MPI_Send(greeting, strlen(greeting) + 1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
19      }
20      else {
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Wed 27 Mar - 15:36 ~/GitHub/Parallel-Programming/Examples  origin ☊ main 1* 2●
chef@ChefMichelin-PC  make && mpirun MPIHello
Consolidate compiler generated dependencies of target MPIHello
[ 50%] Building C object CMakeFiles/MPIHello.dir/MPIHello.c.o
[100%] Linking C executable MPIHello
[100%] Built target MPIHello
Greetings from process 0 of 8!
Greetings from process 1 of 8!
Greetings from process 2 of 8!
Greetings from process 3 of 8!
Greetings from process 4 of 8!
Greetings from process 5 of 8!
Greetings from process 6 of 8!
Greetings from process 7 of 8!
```

2. `strlen(greeting)`

```
16      if (my_rank != 0) {
17          sprintf(greeting, "Greetings from process %d of %d!", my_rank, comm_sz);
18          MPI_Send(greeting, strlen(greeting), MPI_CHAR, 0, 0, MPI_COMM_WORLD);
19      }
20      else {
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Wed 27 Mar - 15:37 ~/GitHub/Parallel-Programming/Examples  origin ☊ main 1* 2●
chef@ChefMichelin-PC  make && mpirun MPIHello
Consolidate compiler generated dependencies of target MPIHello
[ 50%] Building C object CMakeFiles/MPIHello.dir/MPIHello.c.o
[100%] Linking C executable MPIHello
[100%] Built target MPIHello
Greetings from process 0 of 8!
Greetings from process 1 of 8!
Greetings from process 2 of 8!
Greetings from process 3 of 8!
Greetings from process 4 of 8!
Greetings from process 5 of 8!
Greetings from process 6 of 8!
Greetings from process 7 of 8!
```

1.2 习题 2

考虑以下程序

```c
#include <stdio.h>
#include <mpi.h>

int main(void) {
    int my_rank, comm_sz;

    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    printf("Proc %d of %d > Does anyone have a toothpick?\n", my_rank, comm_sz);

    MPI_Finalize();
    return 0;
}   /* main */
```

每个进程都会打印一行输出, 但是会是乱序的. 请你提出一种修改程序的思路, 使得输出能够按照进程号的顺序打印, 即进程 0 先输出, 然后是进程 1, 以此类推.

通过在输出前进行进程同步, 使用循环确定输出进程来达到目的, 修改后的代码如下

```c
#include <stdio.h>
#include <mpi.h>

int main(void) {
    int my_rank, comm_sz;

```

```
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    for (int rank = 0; rank < comm_sz; rank++) {
        MPI_Barrier(MPI_COMM_WORLD);
        if (my_rank == rank)
            printf("Proc %d of %d > Does anyone have a toothpick?\n", my_rank, comm_sz);
    }

    MPI_Finalize();
    return 0;
}   /* main */
```

1.3 习题 3

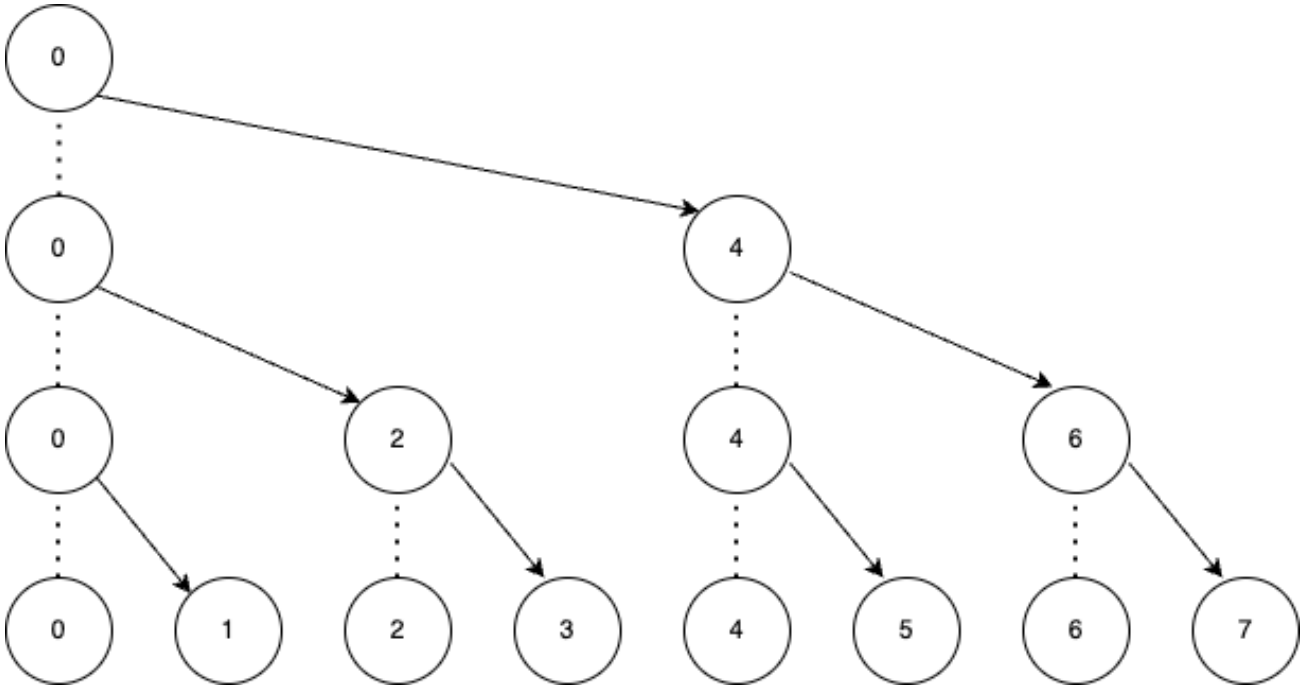
如果通信子中只包含一个进程, 不同的 MPI 集合通信函数分别会做什么?

1. **MPI_Bcast**: 这个函数用于广播数据, 即从一个进程 (根进程) 发送数据到通信子中的所有其他进程. 如果通信子中只有一个进程, 那么这个函数实际上不会做任何事情, 因为没有其他进程可以接收数据.
2. **MPI_Scatter**: 这个函数用于将数据从一个进程 (根进程) 分散到通信子中的所有其他进程. 如果通信子中只有一个进程, 那么这个函数也不会做任何事情, 因为没有其他进程可以接收数据.
3. **MPI_Gather**: 这个函数用于将数据从通信子中的所有进程收集到一个进程 (根进程). 如果通信子中只有一个进程, 那么这个函数也不会做任何事情, 因为没有其他进程可以发送数据.
4. **MPI_Allgather**: 这个函数用于将数据从每个进程收集, 并发送到所有进程. 如果通信子中只有一个进程, 那么这个函数也不会做任何事情, 因为没有其他进程可以发送或接收数据.
5. **MPI_Reduce**: 这个函数用于将数据从所有进程收集到一个进程, 并对这些数据进行某种操作 (如求和、求最大值等). 如果通信子中只有一个进程, 那么这个函数也不会做任何事情, 因为没有其他进程可以发送数据.
6. **MPI_Allreduce**: 这个函数与 `MPI_Reduce` 类似, 但是结果会被发送到所有进程. 如果通信子中只有一个进程, 那么这个函数也不会做任何事情, 因为没有其他进程可以发送数据.
7. **MPI_Barrier**: 这个函数用于同步所有进程, 即阻止任何进程继续执行, 直到所有进程都调用了这个函数. 如果通信子中只有一个进程, 那么这个函数会立即返回, 因为没有其他进程需要同步.

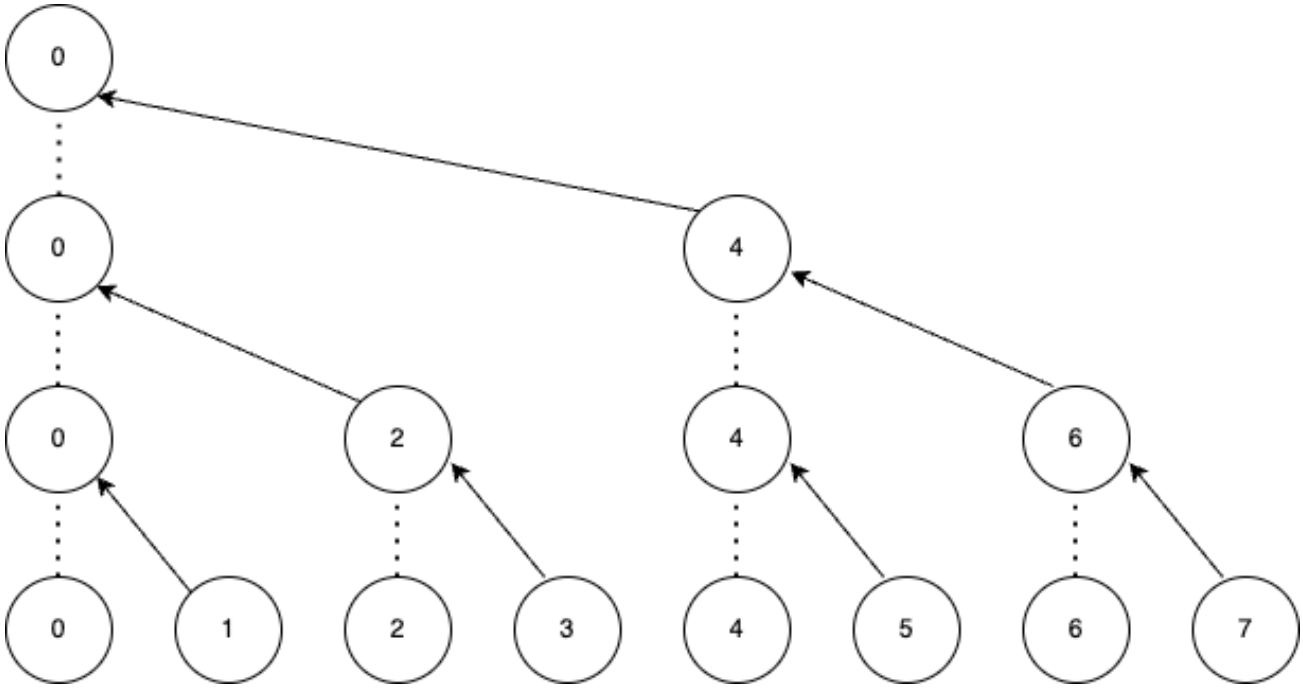
1.4 习题 4

假设 `comm_sz = 8, n = 16`

(1) 画一张图来说明当进程 0 要分发 n 个元素的数组时, 怎样使用拥有 `comm_sz` 个进程的树形结构的通信来实现 `MPI_Scatter`.

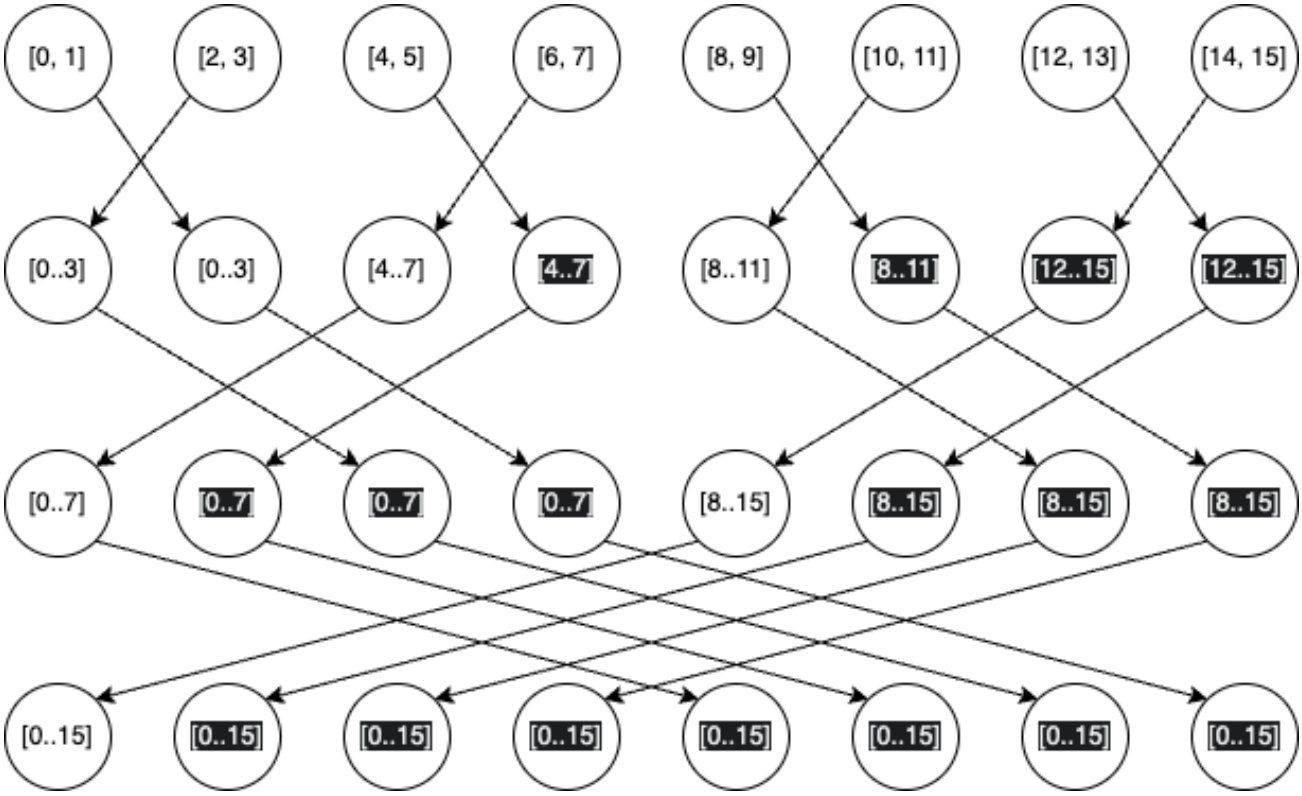


(2) 画一张图来说明已经被分发到 `comm_sz` 个进程的 n 个数组元素要保存到进程 0 时, 怎样使用树形结构的通信来实现 `MPI_Gather`.



1.5 习题 5

假定 `comm_sz = 8`, 向量 $x = (0, 1, 2, \cdots, 15)$, 通过块划分方式分配 x 给各个进程, 画图表示用蝶形通信结构实现聚焦 x 的步骤.


[0, 1]
[2, 3]
[4, 5]
[6, 7]
[8, 9]
[10, 11]
[12, 13]
[14, 15]
[0..3]
[0..3]
[4..7]
4..7
[8..11]
8..11
12..15
12..15
[0..7]
0..7
0..7
0..7
[8..15]
8..15
8..15
8..15
[0..15]
[0..15]
[0..15]
[0..15]
[0..15]
[0..15]
[0..15]
[0..15]