MPI 作业实验报告

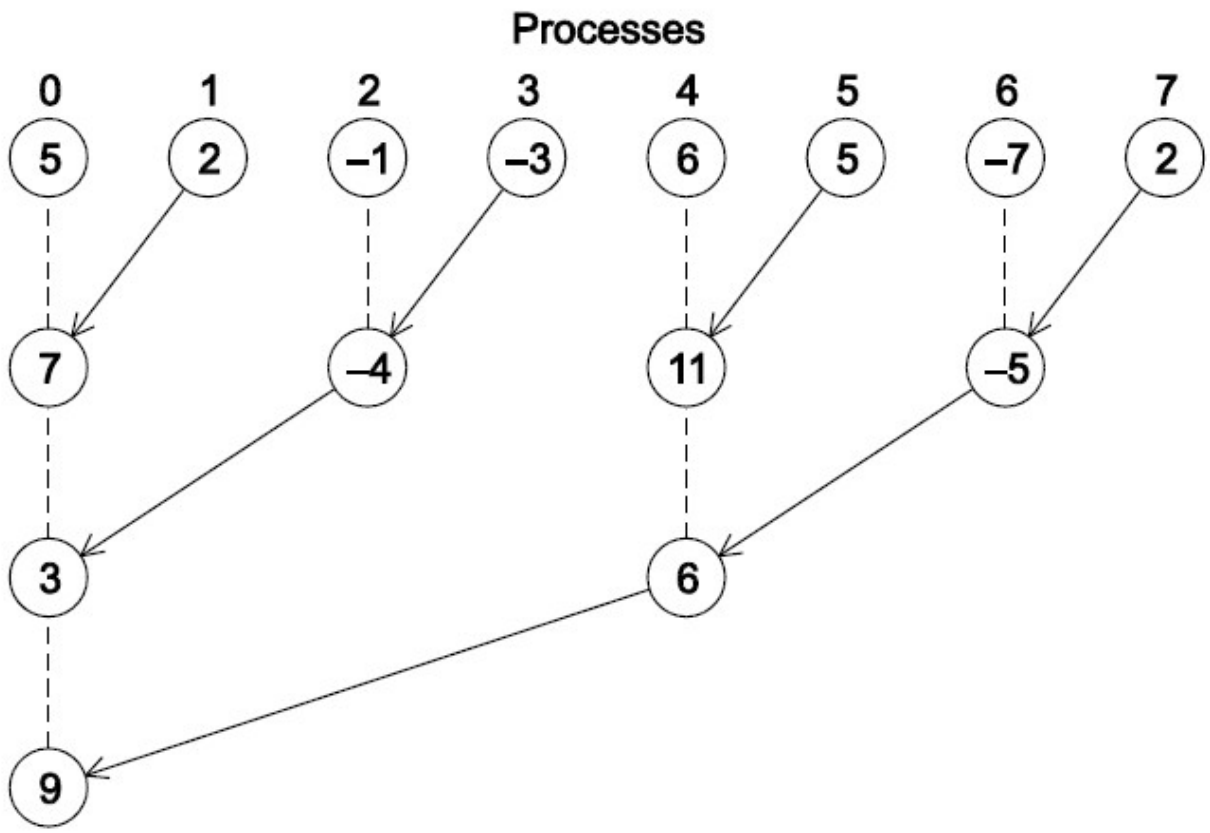
1. 题目

编写一个MPI程序，分别采用树形和蝶形通信结构计算全局总和。首先计算通信域comm_sz的进程数是2的幂的特殊情况，若能够正确运行，改变该程序使其适用于comm_sz中任意进程数目的值。

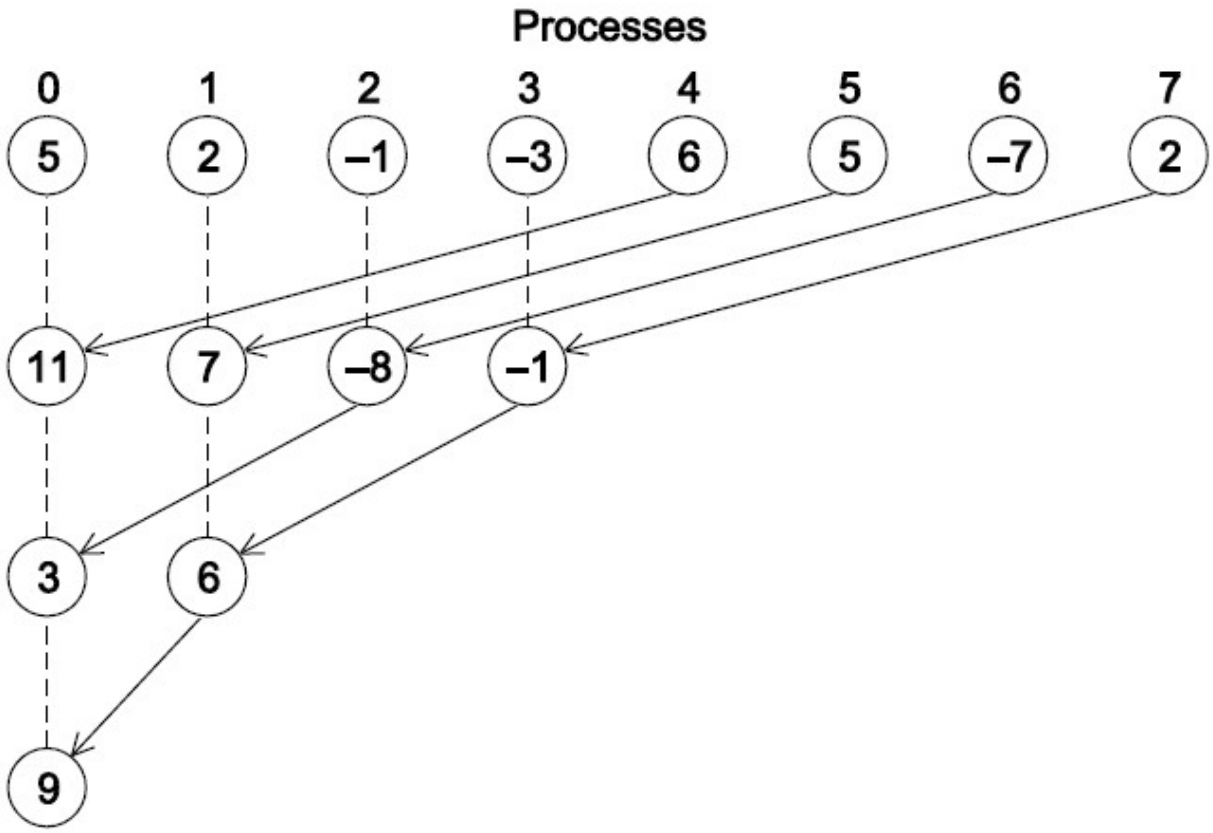
2. 树形

2.1 进程数是 2 的幂的特殊情况

2.1.1 分析



上图是明显的树形全局求和流程，八个进程只需要三次消息的发送和接收。



为了更容易编写代码，一般采用上图所示形式的树形求和流程。

当进程数为 2 的幂时，那么前半部分的每个节点可以都可以和后半部分的节点配对并接收它们的和，每次有有效通信域减半，最后会只剩下 0 号节点。

2.1.2 代码

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

void tree_sum(int my_rank, int size) {
    int remain = size, sum = my_rank, half, temp;
    while(remain != 1) {
        half = remain/2;
        if(my_rank < half) {
            MPI_Recv(&temp, 1, MPI_INT, my_rank+half, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            sum+=temp;
        }
        else {
            MPI_Send(&sum, 1, MPI_INT, my_rank-half, 0, MPI_COMM_WORLD);
            return;
        }
        remain = half;
    }
    if(my_rank == 0)
        printf("%d\n", sum);
}

```

```c
int main() {
    int comm_sz;
    int my_rank;

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

    tree_sum(my_rank, comm_sz);

    MPI_Finalize();
    return 0;
}
```

2.1.3 代码分析

remain 是有效通信域，初始值是总进程数，每次减小一半。

```c
    while(remain != 1)
```

当有效通信域只剩一个时退出循环。

```c
        if(my_rank < half) {
                MPI_Recv(&temp, 1, MPI_INT, my_rank+half, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
                sum+=temp;
            }
            else {
                MPI_Send(&sum, 1, MPI_INT, my_rank-half, 0, MPI_COMM_WORLD);
                return;
            }
```

前半部分接收值，将接收到的值加到自己的 sum 中。后半部分发送值，当一个结点发送完它的 sum 时它的任务就完成了，所以 return 退出函数。

2.1.4 结果

```
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 8 ./tree_sum_square
28
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 16 ./tree_sum_square
120
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 32 ./tree_sum_square
496
```

假定我们对每个进程的进程号求和。

3 个数字分别是有 8 个进程、16 个进程和 32 个进程的结果。

当有 8 个进程时，全局和为 $\frac{(0+7)\times 8}{2} = 28$，结果正确。

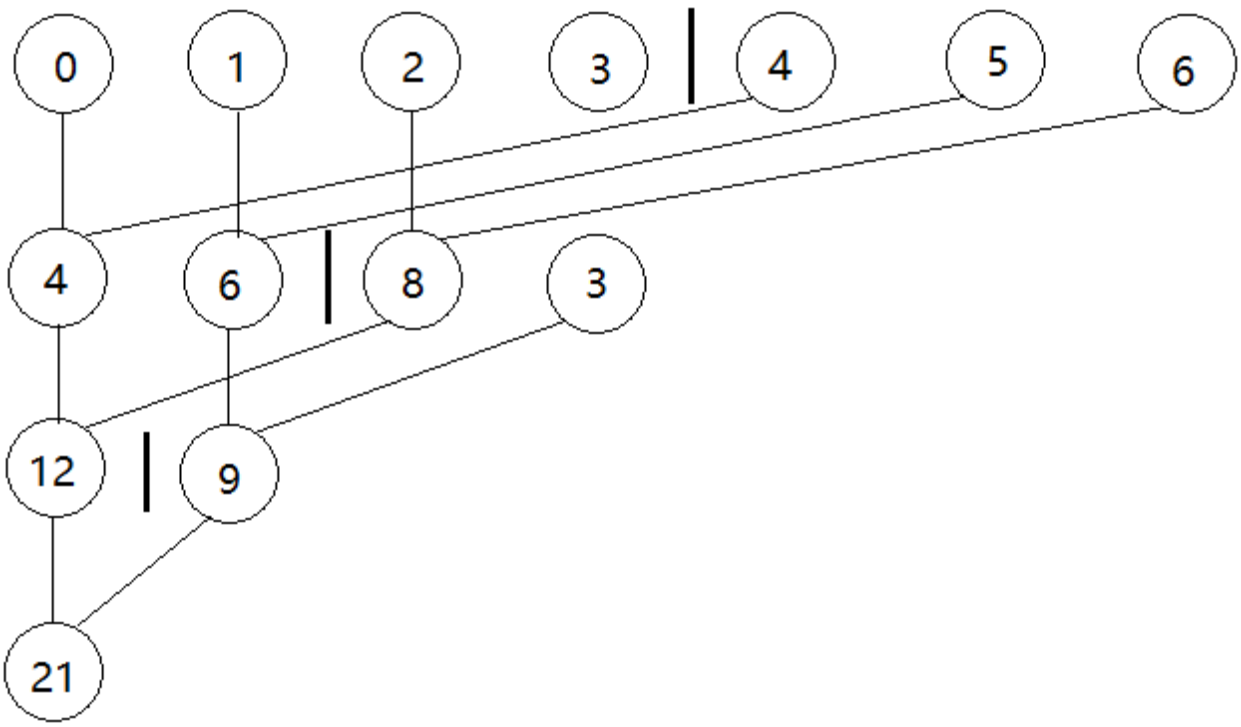
当有 16 个进程时，全局和为 $\frac{(0+15)\times 16}{2} = 120$，结果正确。

当有 32 个进程时，全局和为 $\frac{(0+31)\times 32}{2}=496$，结果正确。

2.2 任意进程数

2.2.1 分析

任意进程数和 2 的幂的进程数的树形全局求和过程类似，它们两个的区别在于任意进程数的有效通信域可能是奇数，要对这种情况做特殊的处理。我的处理方法依然是将有效通信域分为前后两部分，当有效通信域为奇数时，前半部分的节点数比后半部分多一个，但是那多出的一节点不参与配对通信。



2.2.2 代码

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

void tree_sum(int my_rank, int size) {
    int remain = size, sum = my_rank, half, rm, temp;
    while(remain != 1) {
        half = remain/2;
        rm = remain%2;
        if(my_rank < half) {
            MPI_Recv(&temp, 1, MPI_INT, my_rank+half+rm, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            sum+=temp;
        }
        else if(my_rank >= half+rm) {
            MPI_Send(&sum, 1, MPI_INT, my_rank-half-rm, 0, MPI_COMM_WORLD);
            return;
        }
        remain = half+rm;
```

```
    }
    if(my_rank == 0)
        printf("%d\n", sum);
}

int main() {
    int comm_sz;
    int my_rank;

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

    tree_sum(my_rank, comm_sz);

    MPI_Finalize();
    return 0;
}
```

2.2.3 代码分析

> 代码结构与上一节类似

```
        rm = remain%2;
```

rm 用来判断通信域是否是奇数。特别的，用 rm 标志可以使以下代码既适用于奇数也适用于偶数。

```
        if(my_rank < half) {
                MPI_Recv(&temp, 1, MPI_INT, my_rank+half+rm, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
                sum+=temp;
            }
```

如果是奇数，那么发送节点的 dst 会多出 1 以越过前半部分多出的那个节点。

```
        else if(my_rank >= half+rm) {
                MPI_Send(&sum, 1, MPI_INT, my_rank-half-rm, 0, MPI_COMM_WORLD);
                return;
            }
```

如果通信域是奇数，发送节点是从 `half+1` 开始的。

2.2.4 结果

```
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 7 ./tree_sum
21
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 8 ./tree_sum
28
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 20 ./tree_sum
190
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 35 ./tree_sum
595
```

4 个数字分别是有 7 个进程、8 个进程、20个进程和 35 个进程的结果。

当有 7 个进程时，全局和为 $\frac{(0+6)\times 7}{2}=21$，结果正确。

当有 8 个进程时，全局和为 $\frac{(0+7)\times 8}{2}=28$，结果正确。

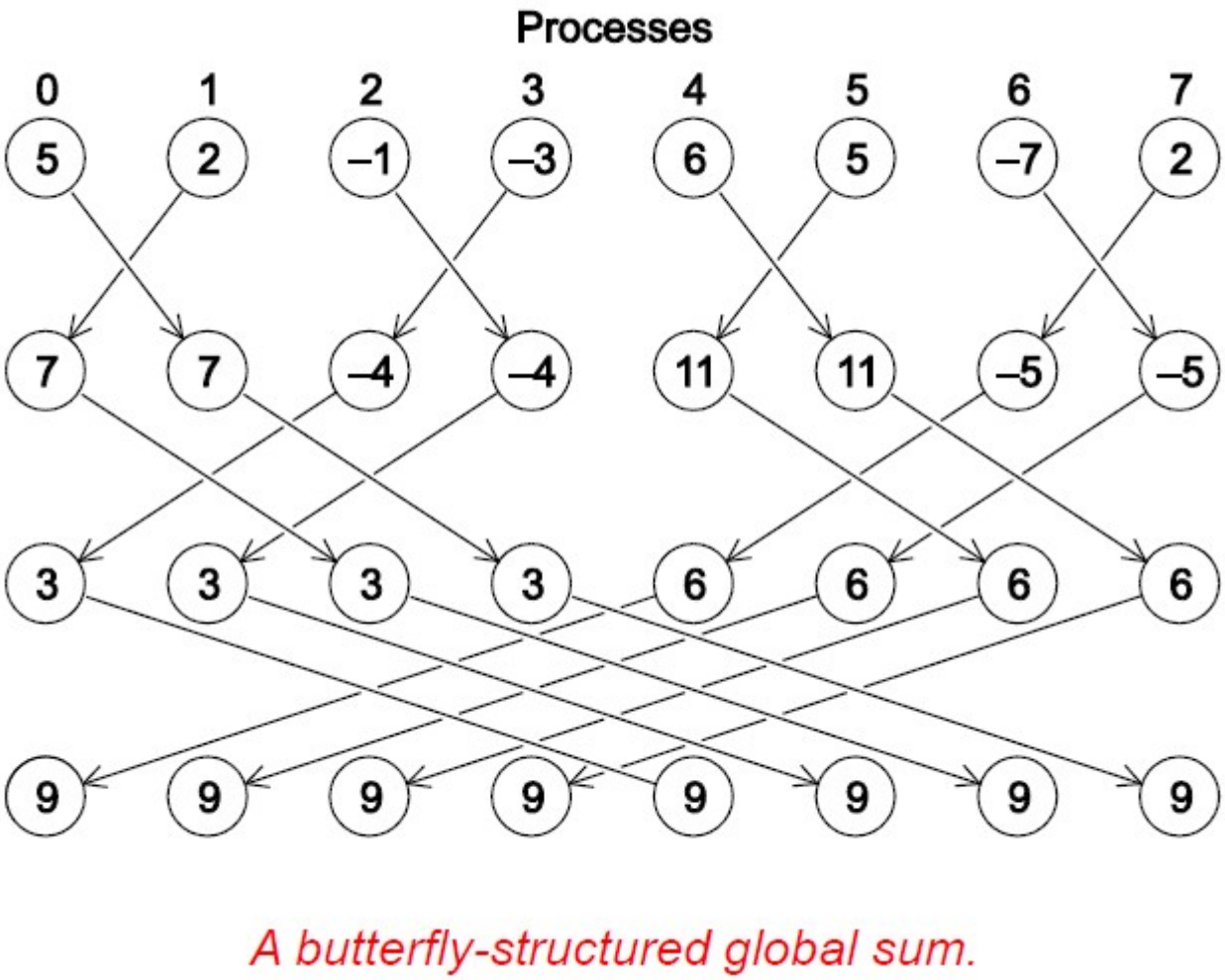
当有 20 个进程时，全局和为 $\frac{(0+19)\times 20}{2}=190$，结果正确。

当有 35 个进程时，全局和为 $\frac{(0+34)\times 35}{2}=595$，结果正确。

3. 蝶形

3.1 进程数是 2 的幂的特殊情况

3.1.1 分析



A butterfly-structured global sum.

如上图，蝶形全局求和可以让每个结点都得到全局和，相当于 `MPI_Allreduce()` 函数。

蝶形全局求和的过程是，第一步将两个相邻的节点分作一组，互相通信他们的 sum，那么这个两节点小组的每个结点中的 sum 都是这个小组的局部和。第二步将四个节点分作一组，前半部分与后半部分相互通信，那么这个四节点小组的每个结点中的 sum 都是这个小组的局部和。循环进行这个步骤直到小组容量大于总进程数。

3.1.2 代码

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

void butterfly_sum(int my_rank, int size) {
    int sum = my_rank, step = 2, dst, temp;
    while(step <= size) {
        if(my_rank%step < step/2)
            dst = my_rank + step/2;
        else
            dst = my_rank - step/2;

        MPI_Sendrecv(&sum, 1, MPI_INT, dst, 0,
                     &temp, 1, MPI_INT, dst, 0,
                     MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        sum += temp;
        step *= 2;
    }
    if(my_rank == 0)
        printf("%d\n", sum);
}

int main() {
    int comm_sz;
    int my_rank;

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

    butterfly_sum(my_rank, comm_sz);

    MPI_Finalize();
    return 0;
}
```

3.1.3 代码分析

step 是当前步骤的小组容量。

```c
        if(my_rank%step < step/2)
            dst = my_rank + step/2;
        else
            dst = my_rank - step/2;
```

以上代码用于判断当前节点属于小组的前半部分还是后半部分。

```
        MPI_Sendrecv(&sum, 1, MPI_INT, dst, 0,
                     &temp, 1, MPI_INT, dst, 0,
                     MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

每个结点在每个步骤都要经历一次发送和接收的过程，如果互相通信的进程都在发送等待对方接收会造成死锁，`MPI_Sendrecv()` 函数会自动协调两个节点的发送和接收顺序，避免死锁。

3.1.4 结果

```
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 8 ./butterfly_sum_square
28
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 16 ./butterfly_sum_square
120
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 32 ./butterfly_sum_square
496
```

3 个数字分别是有 8 个进程、16 个进程和 32 个进程的结果。

当有 8 个进程时，全局和为 $\frac{(0+7)\times 8}{2} = 28$，结果正确。

当有 16 个进程时，全局和为 $\frac{(0+15)\times 16}{2} = 120$，结果正确。

当有 32 个进程时，全局和为 $\frac{(0+31)\times 32}{2} = 496$，结果正确。

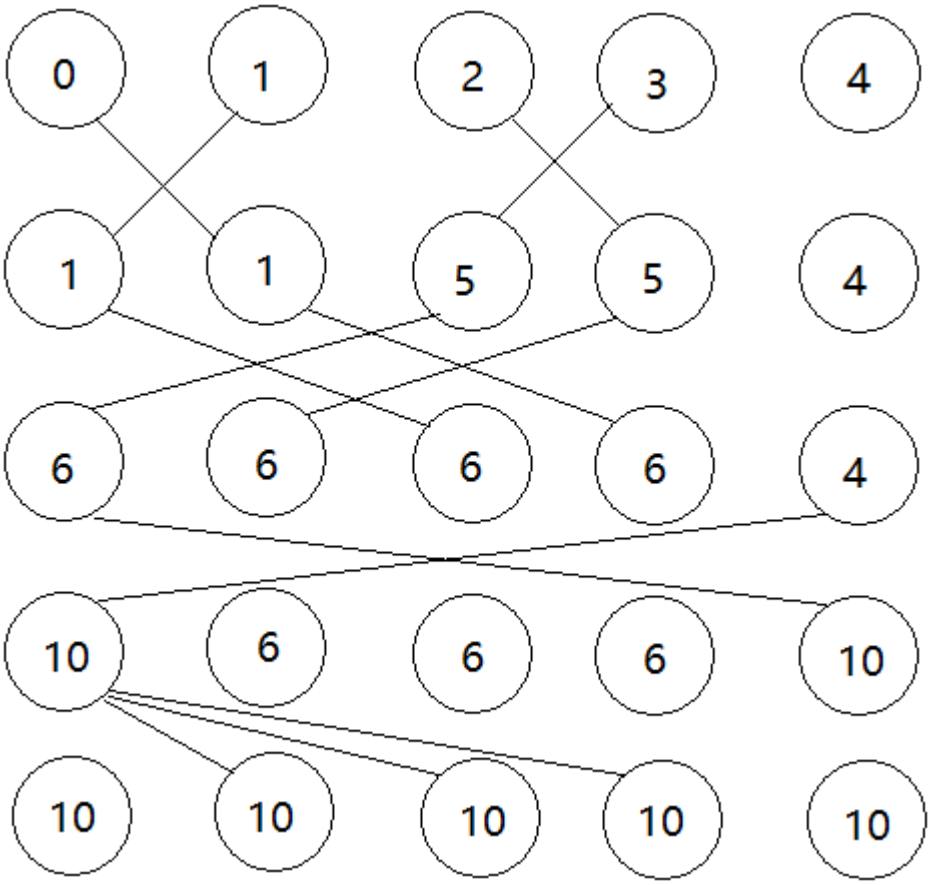
3.2 任意进程数

3.2.1 分析

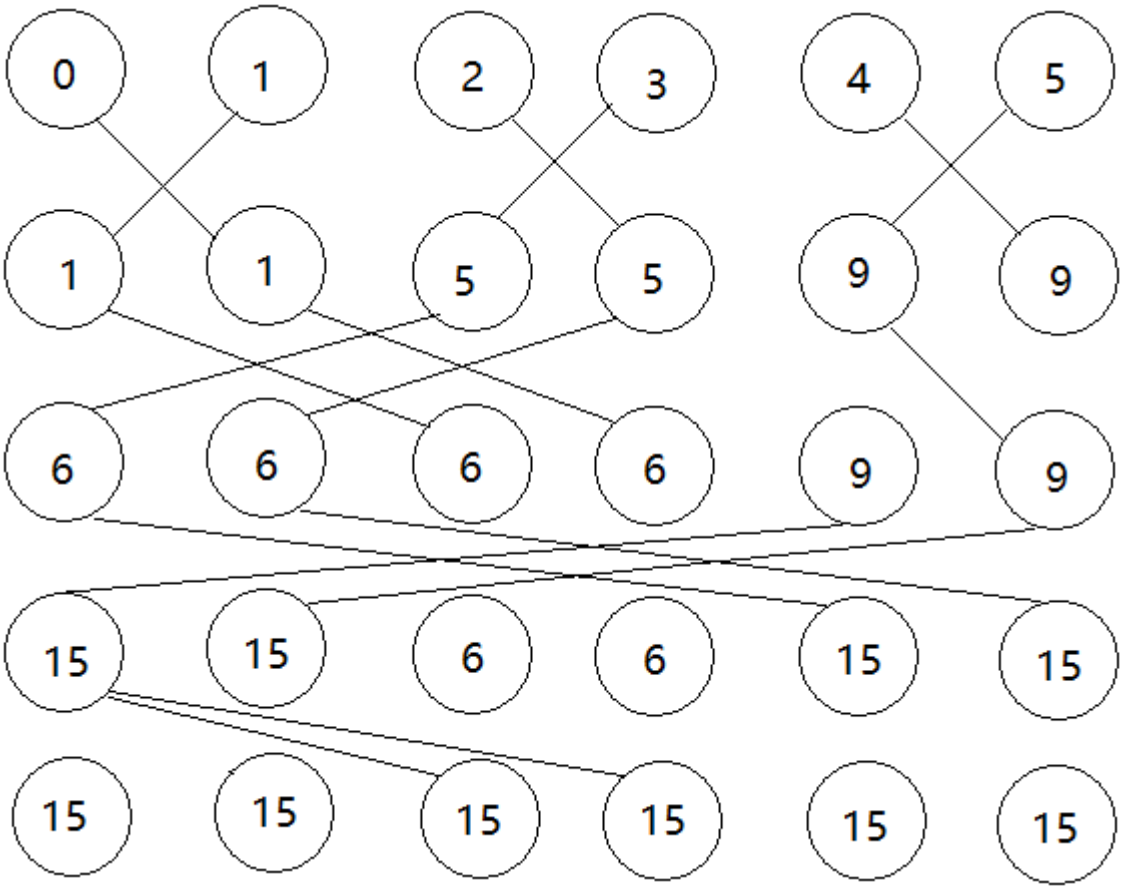
在任意进程数的蝶形全局求和中，容易遇到最后一个分组的节点数量不能填满分组容量的状况。

对于节点不足的分组，先将能配对的节点先进行配对通信，然后由这个小组的头节点把小组的局部和发给无法配对的节点。

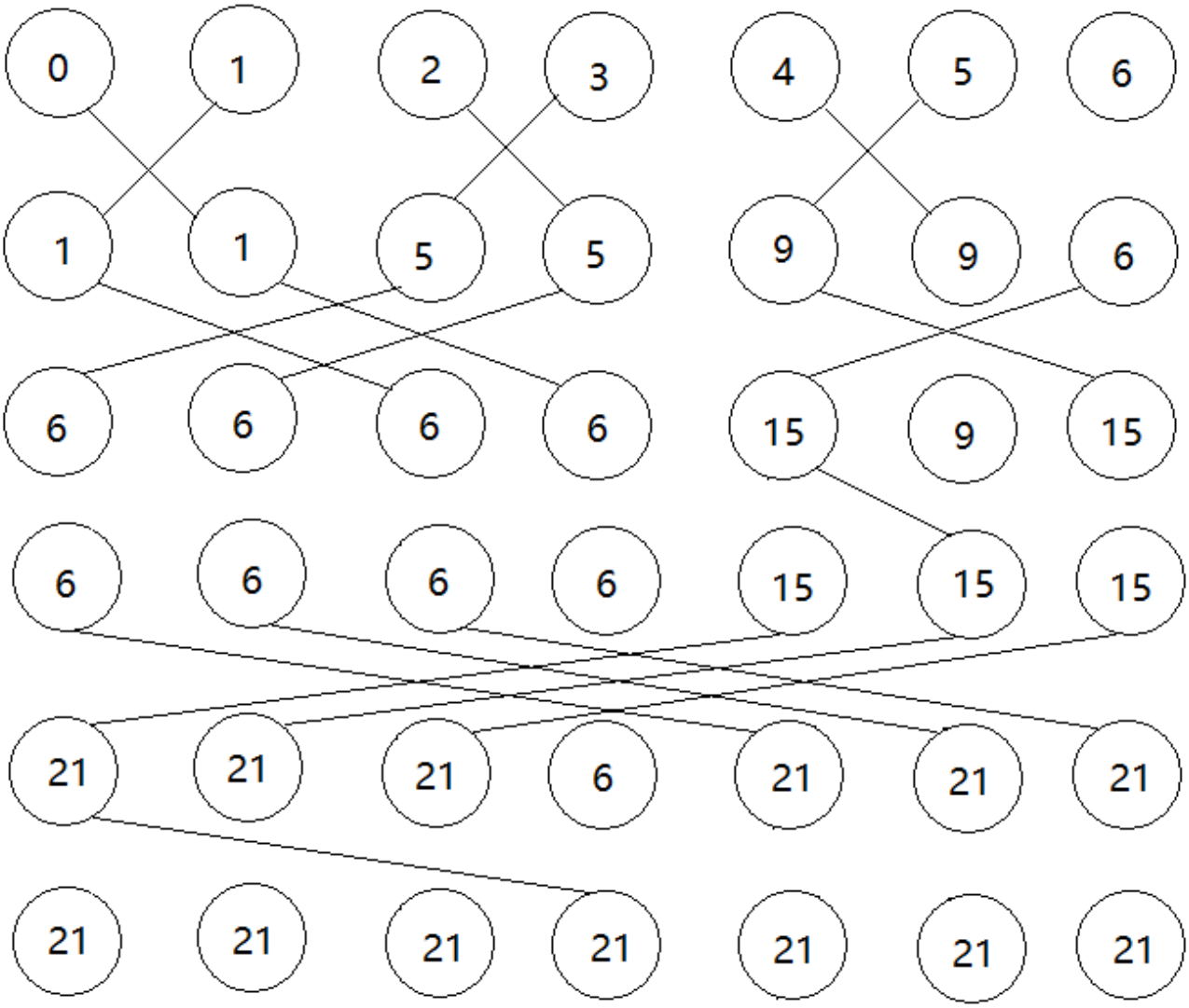
5 个进程的情况：



6 个进程的情况：



7 个进程的情况：



3.2.2 代码

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

int cal(int size) {
    if(size <= 2)
        return size;
    int res = 2;
    while(res < size)
        res *= 2;
    return res;
}

void butterfly_sum(int my_rank, int size) {
    int sum = my_rank, step = 2, dst, temp, new_size = cal(size);
    while(step <= new_size) {
        if(my_rank%step < step/2)
            dst = my_rank + step/2;
        else
            dst = my_rank - step/2;

```

```
        if(dst < size) {
            MPI_Sendrecv(&sum, 1, MPI_INT, dst, 0,
                         &temp, 1, MPI_INT, dst, 0,
                         MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            sum += temp;
        }
        else {
            dst = my_rank - my_rank%step;   //找到step分组的头位节点
            if(dst != my_rank)
                MPI_Recv(&sum, 1, MPI_INT, dst, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
        if(my_rank%step == 0 && my_rank+step > size) {  //有无法配对的节点
            int no_partner_count = my_rank + step - size;   //需要接收的节点个数（除去头节点）
            int middle = my_rank + step/2;  //未配对节点一定在middle前
            for(int node = middle-no_partner_count; node < middle; node++)
                if(node > my_rank && node < size)
                    MPI_Send(&sum, 1, MPI_INT, node, 0, MPI_COMM_WORLD);
        }
        step *= 2;
    }
    if(my_rank == 0)
        printf("%d\n", sum);
}

int main() {
    int comm_sz;
    int my_rank;

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

    butterfly_sum(my_rank, comm_sz);

    MPI_Finalize();
    return 0;
}
```

3.2.3 代码分析

尽管进程数目不再是 2 的幂，我们仍然假定最大分组容量是 2 的幂，且是大于进程数目的最小 2 次幂。

`cal()` 函数就是用于计算最大分组。

```
        if(dst < size) {
            MPI_Sendrecv(&sum, 1, MPI_INT, dst, 0,
                         &temp, 1, MPI_INT, dst, 0,
                         MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            sum += temp;
        }
        else {
            dst = my_rank - my_rank%step;    //找到step分组的头位节点
            if(dst != my_rank)
                MPI_Recv(&sum, 1, MPI_INT, dst, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
```

如果一个节点的配对节点存在，则互相通信，否则它将从分组头节点获取分组局部和。注意，这个节点不是头节点自己（头节点不需要从自己本身获取信息）。

```
        if(my_rank%step == 0 && my_rank+step > size) {  //有无法配对的节点
            int no_partner_count = my_rank + step - size;    //需要接收的节点个数（除去头节点）
            int middle = my_rank + step/2;  //未配对节点一定在middle前
            for(int node = middle-no_partner_count; node < middle; node++)
                if(node > my_rank && node < size)
                    MPI_Send(&sum, 1, MPI_INT, node, 0, MPI_COMM_WORLD);
        }
```

如果这个节点是分组的头节点并且这个分组中有无法配对的节点，它将计算有可能几个节点无法配对，并且它将从分组中间前面的节点（没有配对的节点一定是 middle 前 no_partner_count 个）开始发送分组局部和。

3.2.4 结果

```
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 7 ./butterfly_sum
21
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 8 ./butterfly_sum
28
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 20 ./butterfly_sum
190
root@iZwz9d8uuil3r2get1phekZ:~/mpi# mpiexec -n 35 ./butterfly_sum
595
```

4 个数字分别是有 7 个进程、8 个进程、20个进程和 35 个进程的结果。

当有 7 个进程时，全局和为 $\frac{(0+6)\times 7}{2}=21$，结果正确。

当有 8 个进程时，全局和为 $\frac{(0+7)\times 8}{2}=28$，结果正确。

当有 20 个进程时，全局和为 $\frac{(0+19)\times 20}{2}=190$，结果正确。

当有 35 个进程时，全局和为 $\frac{(0+34)\times 35}{2}=595$，结果正确。