MPI树形通信结构和蝶形通信结构的实现

1. 内容简介

编写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。

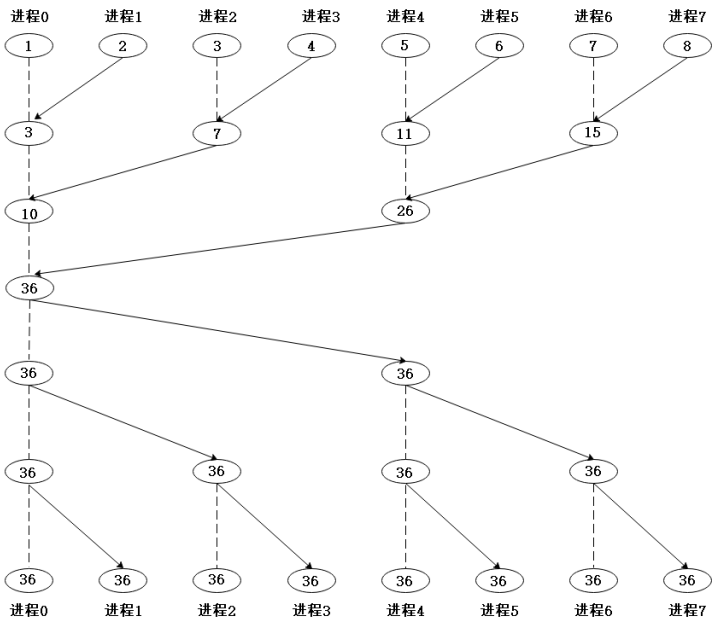
2. 树形通信结构

2.1 树形通信结构分析

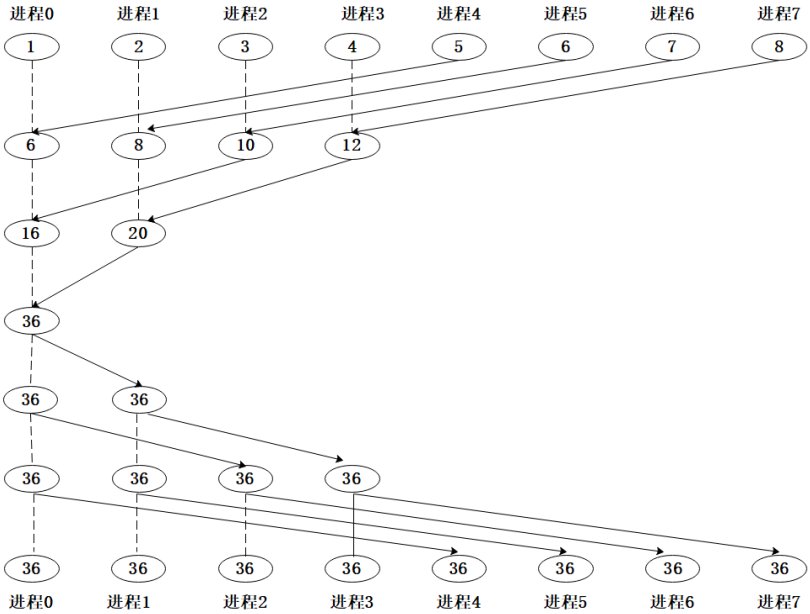
树形通信结构是用于MPI集合通信中的一种实现结构。树形方式也有多种（如图所示），其基本思想都是将一个进程和另一个进程配对，由一个进程向另一个进程发送消息，然后发送消息的进程进入不活跃状态，接收消息的进程对收到的数据消息和自己已有的消息进行操作，之后再把消息发送给配对的进程，这个过程一直进行直到只有一个活跃的进程，这个进程的数据就是要求的全局的数据，如和，最大值，最小值等。在树形通信结构中，每一轮通信都会减少一半进程数。每个进程可能接收多次消息，但是只会发生一次消息。通信的轮次为:$O(logN)$次，其中N是进程的数量。

当一个进程获得了全局的结果后，需要分发到每一个进程，过程和求全局结果相反。通信的轮次也为:$O(logN)$次，其中N是进程的数量。分发的结构和求全局结果的结构上下对称，箭头相反。

树形通信结构1：



树形通信结构2：



2.2 进程数目为2的幂次的树形通信结构实现

（代码为：mpi_tree_v1.cpp）

2.2.1 实现分析

不同的树形通信结构方式有不同的配对策略，他们可能因此有不同的性能。我实现的是上图的第二种树形通信结构。

实现树形通信结构的关键在于确定进程的角色和与之配对的进程，在不同的时刻相互匹配的进程是不同的。

但是我们可以观察到，求全局和的过程中，每一轮的通信中，总是有一半的活跃进程发送消息，另一半的活跃进程接收消息，并且活跃的进程数量每一轮次减少一半。我们可以因此得出进程的配对伙伴：

- 当进程处在当前活跃进程的前半部分：
  $my_rank < current_comm_size/2$时，它是接收进程，配对进程是：$my\_rank + current\_step\_size/2$；
- 当进程处于当前活跃进程的后半部分：
  $my\_rank >= current\_comm\_size/2$，它是发送进程，配对进程是：$my\_rank - current\_step\_size/2$；

分发全局和的过程中，每一轮的通信中，同样总是有一半的活跃进程发送消息，另一半的活跃进程接收消息，并且活跃的进程数量每一轮次增加一倍。我们可以因此得出进程的配对伙伴：

- 当进程处在当前活跃进程的前半部分时，它是发送进程，并且配对进程是：$my\_rank + current\_step\_size/2$；
- 当进程处于当前活跃进程的后半部分时，它是接收进程，并且配对进程是：$my\_rank - current\_step\_size/2$；

2.2.2 代码说明

2.2.2.1 全局求和

首先，注意到只有当当前活跃的进程数量不少于1，并且当前进程在活跃进程范围内才需要进行send或者receive：

```
while(current_comm_size > 1 && my_rank < current_comm_size)
```

如果进程处于当前活跃进程的后半部分时，它是发送进程，需要发送自己的局部和：

```
// in this case, process should send and exit
      if( my_rank >= current_comm_size/2 ){

         // destination is my_rank - current_comm_size/2
         dst = my_rank - current_comm_size/2;
         MPI_Send(&local_sum, 1, MPI_INT, dst, 0, MPI_COMM_WORLD);
      }
```

如果进程处于当前活跃进程的前半部分时，它是接收进程，需要接收数据，收到数据之后把它与本地和相加：

```
// need to get messege
      else{

         // destination is my_rank + current_comm_size/2
         dst = my_rank + current_comm_size/2;
         MPI_Recv(&receive_data, 1, MPI_INT, dst, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
         local_sum += receive_data;
      }
```

每一轮通信完毕之后，当前的活跃进程数应该减小为原来的一半：

```
// updata current_comm_size
      current_comm_size /= 2;
```

2.2.2.2 数据分发

0号进程的局部和就是全局和：

```
// total sum is 0 process's total sum
   if(my_rank == 0) total_sum = local_sum;
```

从活跃进程数量为2开始进行全局和的分发：

```
// second, broacast total sum in all process:
   current_comm_size = 2;
```

只有当本进程处于分发全局和的范围内才开始进行接收全局和或者向其他进程发送全局和：

```
 // only in this time, should this process be involed in
   while(my_rank >= current_comm_size) current_comm_size *= 2;
```

如果进程处于当前活跃进程的后半部分时，它是接收进程，接收全局和：

```
// it should receive data
      if(my_rank >= current_comm_size/2){
         dst = my_rank - current_comm_size/2;
         MPI_Recv(&total_sum, 1, MPI_INT, dst, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
      }
```

如果进程处于当前活跃进程的前半部分，它是发送进程，发送全局和：

```
 // else it should send data
      else{
         dst = my_rank + current_comm_size/2;
         MPI_Send(&total_sum, 1, MPI_INT, dst, 0, MPI_COMM_WORLD);
      }
```

每一轮通信完毕之后，当前的活跃进程数应该增加一倍：

```
// updata current_comm_size
      current_comm_size *= 2;
```

2.3 任意进程数目的树形通信结构实现

（代码为：mpi_tree_v2.cpp）

2.3.1 实现分析

由于我们已经实现了在特殊情况下（进程数量为2的幂次）的树形通信结构，所以我们实现任意数目进程的时候，可以**把问题转化到进程数目为2的幂次的特殊情况。**我使用的方法是，在算法中，设立虚拟进程数，它是不小于实际进程数目的最小的是2的幂次的数。在需要通信，如发送数据或者接收数据的时候，判断是否存在配对的进程，仅在存在目标进程在实际进程数目的范围内才进行通信。

2.3.2 代码说明

首先，我们需要先找出不小于实际进程数的最小的是2的幂次的数，在算法中把它作为虚拟进程数：

```
// find a less int that is large than init_value and is power of 2
int adaptor(int init_value){
   int result = 2;
   if(init_value <= 2) return init_value;

   while(result < init_value) result*=2;

   // printf("adaptor is :%d\n",result);
   return result;
}
```

在需要发送消息或者接收消息的时候，判断目标进程是否存在：

计算全局和的时候，只需要对希望接收消息的目标进程进行判断，而不需要对希望发送的进程进行判断（因为接收的目标进程一定存在，但是不一定有发送的目标进程）：

```
// need to get messege
      else{

         // destination is my_rank + current_comm_size/2
         dst = my_rank + current_comm_size/2;

         // but when destination is out of actual size, it should not expect to
receive messege.
         if(dst <= comm_sz - 1){
            MPI_Recv(&receive_data, 1, MPI_INT, my_rank +
current_comm_size/2, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            local_sum += receive_data;
         }
      }
```

同理，**分发全局和的时候，只需要判断接收数据的目标进程是否存在，不需要判断发送消息的进程是否存在**（因为发送消息的进程一定存在，但是接收消息的进程不一定存在）：

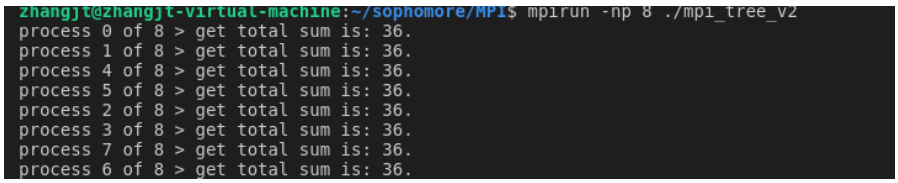
```
// else it should send data
      else{
         dst = my_rank + current_comm_size/2;

         // but when destination is out of actual size, it should not expect to
receive messege.
         if(dst <= comm_sz - 1){
            MPI_Send(&total_sum, 1, MPI_INT, dst, 0, MPI_COMM_WORLD);
         }
      }
```

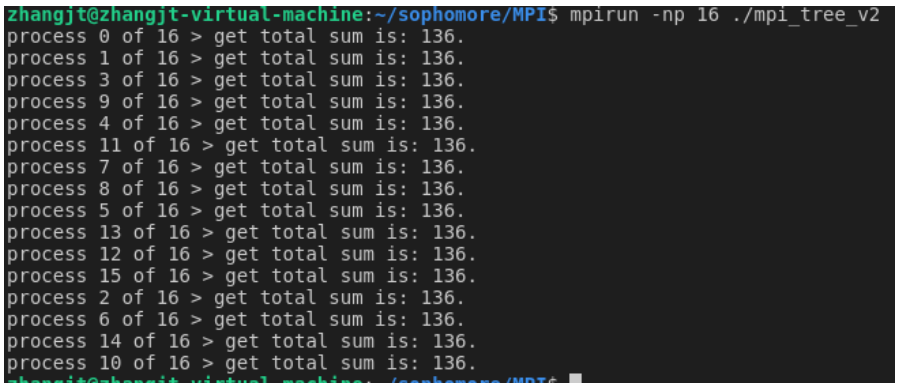
2.4 实验结果

2.4.1进程数目为2的幂次的情况：

$\sum_{i=1}^{8} i = 36$:



$\sum_{i=1}^{16} i = 136$:



2.4.2 进程数目不是2的幂次的情况：

$\sum_{i=1}^{6} i = 21$:

```
zhangjt@zhangjt-virtual-machine:~/sophomore/MPI$ mpirun -np 6 ./mpi_tree_v2
process 0 of 6 > get total sum is: 21.
process 1 of 6 > get total sum is: 21.
process 5 of 6 > get total sum is: 21.
process 4 of 6 > get total sum is: 21.
process 2 of 6 > get total sum is: 21.
process 3 of 6 > get total sum is: 21.
```

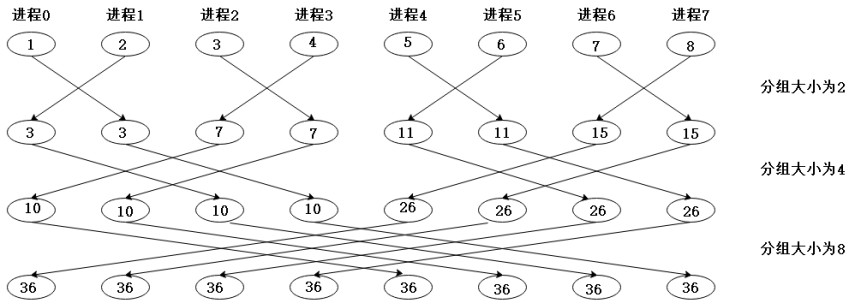
$\sum_{i=1}^{17} i = 153$:

```
zhangjt@zhangjt-virtual-machine:~/sophomore/MPI$ mpirun -np 17 ./mpi_tree_v2
process 0 of 17 > get total sum is: 153.
process 1 of 17 > get total sum is: 153.
process 16 of 17 > get total sum is: 153.
process 8 of 17 > get total sum is: 153.
process 5 of 17 > get total sum is: 153.
process 2 of 17 > get total sum is: 153.
process 6 of 17 > get total sum is: 153.
process 9 of 17 > get total sum is: 153.
process 4 of 17 > get total sum is: 153.
process 10 of 17 > get total sum is: 153.
process 13 of 17 > get total sum is: 153.
process 3 of 17 > get total sum is: 153.
process 11 of 17 > get total sum is: 153.
process 14 of 17 > get total sum is: 153.
process 12 of 17 > get total sum is: 153.
process 7 of 17 > get total sum is: 153.
process 15 of 17 > get total sum is: 153.
```

3. 蝶形通信结构

3.1 蝶形通信结构分析：

蝶形通信结构也是用于MPI集合通信的一种实现结构。蝶形通信的结构的基本思想是把所有的进程作划分，进行分组配对，分组的大小从2开始，分组的大小每次乘以2，直到所有的进程都在同一个组中。每一个分组的前半部分和后半部分进行通信，互相交换信息，确保同一个分组中的所有进程都拥有相同的信息。当分组的大小大于等于进程的数量时，蝶形通信结束，此时所有的进程都得到了全局的数据，如和，最大值，最小值等。蝶形通信的结构如下：



3.2 进程数目为2的幂次的蝶形通信结构实现

（代码为：mpi_plate_v1.cpp）

3.2.1 实现分析

实现蝶形通信结构的关键也是确定进程的配对进程和它的行为。**关键点在于确定进程的所在的分组和其在分组中位置，确定分组和位置，其配对的进程就可以确定。**

从上图我们可以观察到，每一轮的通信中，每一个进程都需要与配对的进程交换信息（即每个进程需要发送一次消息，接收一次消息）。每一个分组中有左半部分和右半部分：

- 进程判断自己是特定分组左半部分的依据：$(my\_rank\%current\_step\_size < current\_step\_size/2$；它的配对进程是：$my\_rank + current\_step\_size/2$;
- 进程判断自己是特定分组右半部分的依据：$(my\_rank\%current\_step\_size >= current\_step\_size/2$；它的配对进程是：$my\_rank - current\_step\_size/2$;

3.2.2 代码说明

从分组大小为2开始，每个进程都要参与到所有的通信轮次，直到当前的分组包含了所有的进程：

```
int current_step_size = 2;
```

```
while(current_step_size <= comm_sz)
```

每一轮的通信中，每一个进程都需要与配对的进程交换信息，所以需要发送消息一次，接收消息一次。这是一种容易导致死锁的情况，因为两个进程需要向彼此发送和接收消息，由于两个进程的序号组合有很多种可能，所以不能使用简单的奇偶分类设计两个进程发送和接收消息的顺序。所幸的是，MPI提供了Sendrecv函数，会自动帮我们协调两个进程的通信，避免死锁：

```
// it is left part of small set, partner is in right
    if(my_rank% current_step_size < current_step_size/2){
       dst = my_rank + current_step_size/2;
    }
    // else it is right part of set,partner is in left
    else{
       dst = my_rank - current_step_size/2;
    }

    // exchange data with partner
    MPI_Sendrecv(&my_sum, 1, MPI_INT, dst, 0,
            &data_receive, 1, MPI_INT, dst, 0,
            MPI_COMM_WORLD, MPI_STATUS_IGNORE);

    my_sum += data_receive;
```

每经过一个轮次，分组的大小就翻倍：

```
// updata current_step_size
    current_step_size *= 2;
```

3.3 任意进程数目的蝶形通信结构实现

（代码为：mpi_plate_v2.cpp）

3.3.1 实现分析

和任意进程数目树形通信结构的实现一样，**我们寻求把任意进程数目的蝶形通信结构的实现转化到进程数目为2的幂次的特殊情况。**但是，此时的情况更复杂。因为此时不像树形通信结构，如果通信的目标进程不存在就可以不通信。**在蝶形通信中，每一个分组通过左右两个部分的通信使得整个分组的局部和一致。**如果有进程匹配的进程不存在，就无法维持整个分组的局部和一致性，也会影响到下一次的分组信息交换，最终导致结果错误。

但是，对于配对的进程不存在的进程，我们可以采用一些办法，使得它的局部和与分组中的其他进程一致。可以验证，在一个分组中，至少存在分组左半部分的坐左边的进程（否则这个分组不会存在）。同样地，我们可以验证，发生进程无法匹配的情况总是发生在最右边的分组。

我采用的方法是，**如果一个进程配对的进程不存在（这一定发生在最右边的分组中），则它从本分组中的第一个进程（这个进程一定存在，并且经过信息交换之后，它的局部和就是本分组的局部和）出获得本分组的局部和。**

- 分组判断自己有配对进程的依据：$dst <= comm\_sz - 1$;
- 如果没有配对的进程，该进程将向本分组最左边的进程获取局部和，这个进程是：$my\_rank - my\_rank\%current\_comm\_size$;

同时，最后一个分组的第一个进程承担更正器的角色，如果它发现本分组中的有些进程没有配对的进程，则它会计算出哪些进程没有配对的进程，并且向它们发送本分组的局部和。

- 判断自己是否是更正器并且是否有进程没有配对的进程:
  $my\_rank\%current\_comm\_size == 0 \&\&$
  $my\_rank + current\_comm\_size > comm\_sz$
- 计算没有配对进程的进程数目:
  $my\_rank + current\_comm\_size - comm\_sz$;

3.3.2 代码说明

首先，我们需要先找出不小于实际进程数的最小的是2的幂次的数，在算法中把它作为虚拟进程数：

```
// find a less int that is large than init_value and is power of 2
int adaptor(int init_value){
   int result = 2;
   if(init_value <= 2) return init_value;

   while(result < init_value) result*=2;

   // printf("adaptor is :%d\n",result);
   return result;
}
```

每个进程与配对的进程交换信息，如果配对的进程不存在，就向本分组的最左边的分组获取局部和：

```
// exchange data with partner, but the parter should be invalid
      if(dst <= comm_sz - 1){
         MPI_Sendrecv(&my_sum, 1, MPI_INT, dst, 0,
               &receive_data, 1, MPI_INT, dst, 0,
               MPI_COMM_WORLD, MPI_STATUS_IGNORE);

         my_sum += receive_data;
      }

      // else it should be corrected, with current_comm_size is larger than 2
      else if(current_comm_size > 2){
         // from leftest partner to get sum
         dst = my_rank - my_rank%current_comm_size;
         if(dst != my_rank)
         MPI_Recv(&my_sum, 1, MPI_INT, dst, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
      }
```

每一个进程都要检查自己是否要作为更正器，是否需要向没有匹配进程的进程发送局部和（但是实际上只有一个进程符合这个条件）：

```
//the leftest process of left part of rightest set should act as corrector, and it only correct when necessary
      if(my_rank%current_comm_size == 0 && my_rank + current_comm_size > comm_sz){

         //there are (my_rank + current_comm_size - comm_sz) processes to correct
         int middle = my_rank + current_comm_size/2;
         for(int no_partner_count = my_rank + current_comm_size - comm_sz; no_partner_count > 0 ; no_partner_count--){
            dst = middle - no_partner_count;
            if(dst > my_rank && dst < comm_sz)
            MPI_Send(&my_sum, 1, MPI_INT, dst, 0, MPI_COMM_WORLD);
         }
      }
```

3.4 实验结果

3.4.1 进程数目为2的幂次的情况：

$\sum_{i=1}^{8} i = 36$：

```
zhangjt@zhangjt-virtual-machine:~/sophomore/MPI$ mpirun -np 8 ./mpi_plate_v2
process 2 of 8 > get total sum is: 36.
process 0 of 8 > get total sum is: 36.
process 7 of 8 > get total sum is: 36.
process 1 of 8 > get total sum is: 36.
process 6 of 8 > get total sum is: 36.
process 3 of 8 > get total sum is: 36.
process 5 of 8 > get total sum is: 36.
process 4 of 8 > get total sum is: 36.
```

$\sum_{i=1}^{16} i = 136$

```
zhangjt@zhangjt-virtual-machine:~/sophomore/MPI$ mpirun -np 16 ./mpi_plate_v2
process 1 of 16 > get total sum is: 136.
process 4 of 16 > get total sum is: 136.
process 12 of 16 > get total sum is: 136.
process 6 of 16 > get total sum is: 136.
process 5 of 16 > get total sum is: 136.
process 2 of 16 > get total sum is: 136.
process 14 of 16 > get total sum is: 136.
process 3 of 16 > get total sum is: 136.
process 0 of 16 > get total sum is: 136.
process 9 of 16 > get total sum is: 136.
process 13 of 16 > get total sum is: 136.
process 8 of 16 > get total sum is: 136.
process 7 of 16 > get total sum is: 136.
process 15 of 16 > get total sum is: 136.
process 10 of 16 > get total sum is: 136.
process 11 of 16 > get total sum is: 136.
```

3.4.2 进程数目不是2的幂次情况：

$\sum_{i=1}^{5} i = 15$：

```
zhangjt@zhangjt-virtual-machine:~/sophomore/MPI$ mpirun -np 5 ./mpi_plate_v2
process 0 of 5 > get total sum is: 15.
process 3 of 5 > get total sum is: 15.
process 4 of 5 > get total sum is: 15.
process 1 of 5 > get total sum is: 15.
process 2 of 5 > get total sum is: 15.
```

$\sum_{i=1}^{15} i = 120$:

```
zhangjt@zhangjt-virtual-machine:~/sophomore/MPI$ mpirun -np 15 ./mpi_plate_v2
process 2 of 15 > get total sum is: 120.
process 12 of 15 > get total sum is: 120.
process 10 of 15 > get total sum is: 120.
process 6 of 15 > get total sum is: 120.
process 8 of 15 > get total sum is: 120.
process 4 of 15 > get total sum is: 120.
process 13 of 15 > get total sum is: 120.
process 11 of 15 > get total sum is: 120.
process 14 of 15 > get total sum is: 120.
process 3 of 15 > get total sum is: 120.
process 0 of 15 > get total sum is: 120.
process 5 of 15 > get total sum is: 120.
process 7 of 15 > get total sum is: 120.
process 9 of 15 > get total sum is: 120.
process 1 of 15 > get total sum is: 120.
```

4. 两种通信结构的比较

可以通过计算得知，当进程的数目为N时：

- 树形通信结构的轮次是$O(2logN)$,蝶形通信的轮次是$O(logN)$；他们的区别在于：树形通信结构中，除了第一轮和最后一轮通信，每一轮都有大量的进程空闲，而蝶形通信结构中所有的进程都参与每一轮的通信，所以蝶形通信的轮次少；
- 树形通信结构的通信次数（计一次发送和一次对应的接收为一次通信）为$(2N-2)$，而蝶形通信结构的通信次数为$(NlogN)$；

由上述分析可知，树形通信结构的通信轮次比蝶形多，但是当进程的数目很大时蝶形通信结构的通信次数比树形结构多得多，而且蝶形通信结构处理进程数目为非2的幂次的方法比较复杂且相对低效。所以实际上哪一个结构的性能更好需要进行大量的实验进行比较分析。