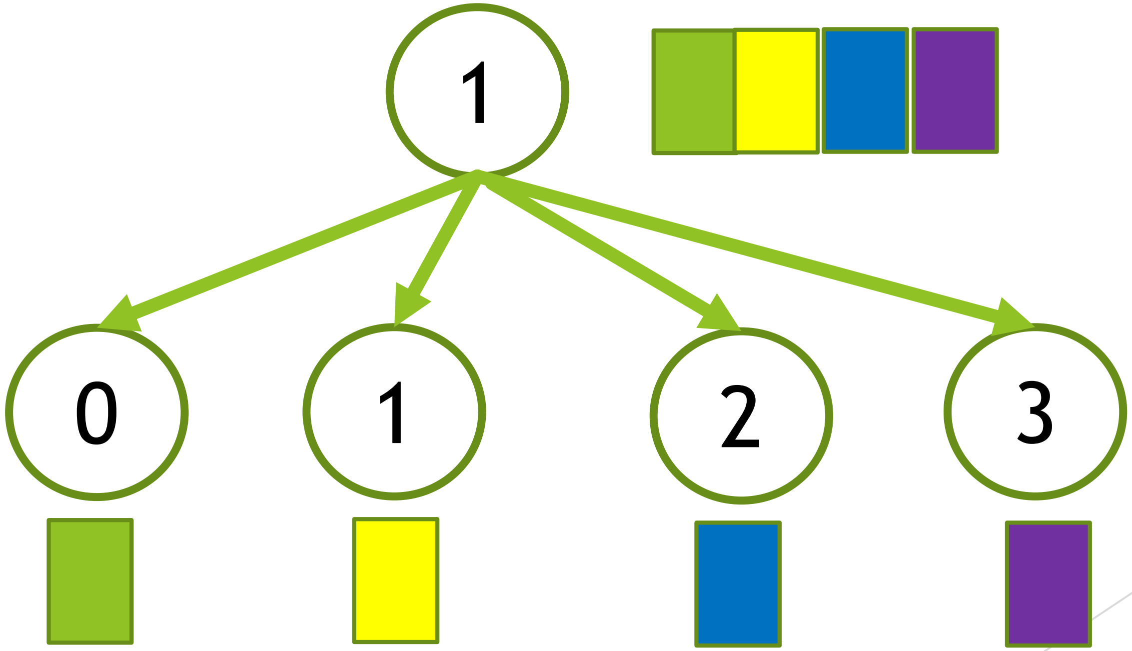
5. MPI集体通信及分组

罗翔宇
中国人民大学统计与大数据研究院

此课件内容主要基于Blaise Barney的网络资料[Message Passing Interface (MPI)]及Wes Kendall的[MPI Tutorial]

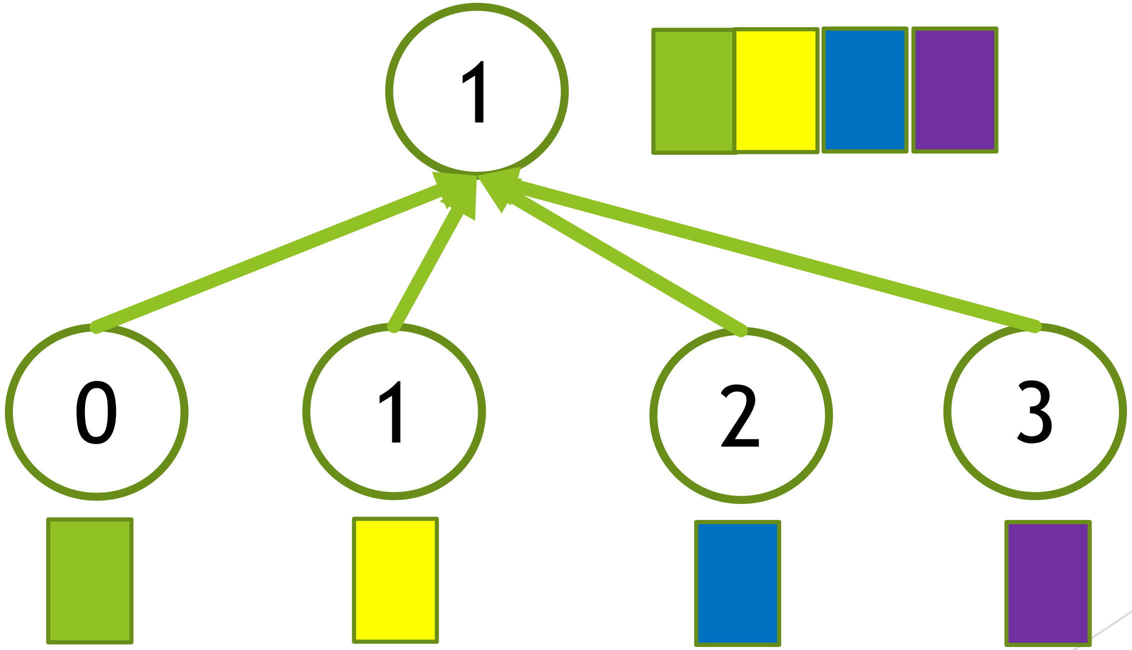
上周课堂复习

- int MPI_Scatter(void* sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuff, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)



上周课堂复习

- int MPI_Gather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuff, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)；



上周课堂复习

- int MPI_Reduce(void *sendbuf, void *recvbuf,
  int count, MPI_Datatype datatype,
  MPI_Op op, int root, MPI_Comm comm)

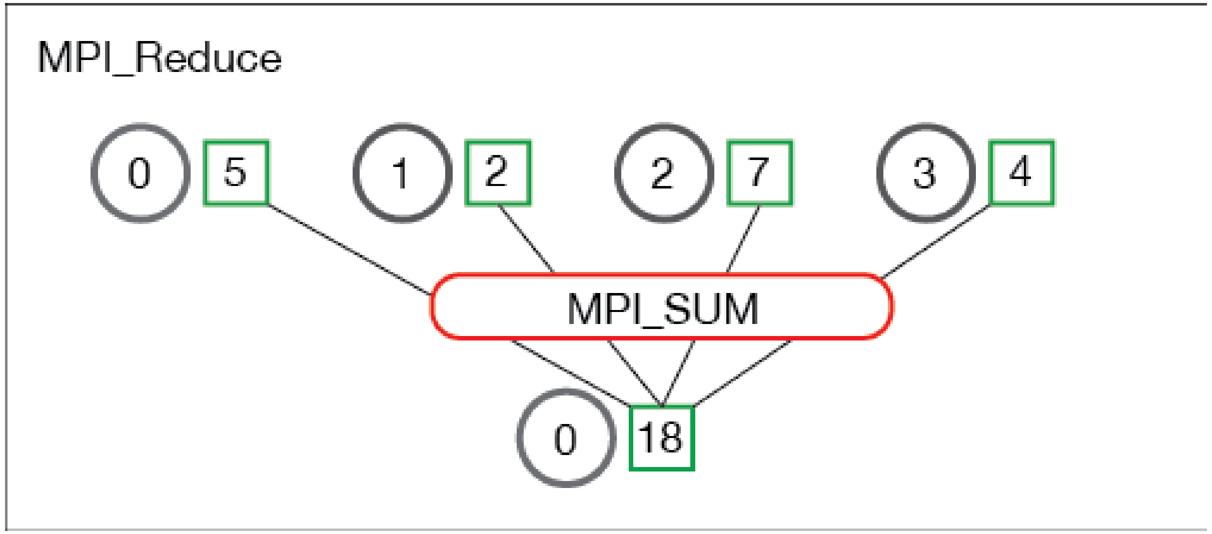


Figure credit: https://mpitutorial.com

上周课堂复习

- `MPI_MAX` - Returns the maximum element.
- `MPI_MIN` - Returns the minimum element.
- `MPI_SUM` - Sums the elements.
- `MPI_PROD` - Multiplies all elements.
- `MPI_LAND` - Performs a logical *and* across the elements.
- `MPI_LOR` - Performs a logical *or* across the elements.
- `MPI_BAND` - Performs a bitwise *and* across the bits of the elements.
- `MPI_BOR` - Performs a bitwise *or* across the bits of the elements.
- `MPI_MAXLOC` - Returns the maximum value and the rank of the process that owns it.
- `MPI_MINLOC` - Returns the minimum value and the rank of the process that owns it.

Figure credit: https://mpitutorial.com

MPI缩减操作例子：计算平均数

- 进程0有数组{1,2,...,100}
- 进程1有数组{101,102,...,200}
- 进程2有数组{201,202,...,300}
- 求每个位置上的平均数
- 利用算子：MPI_SUM

MPI缩减操作例子：计算平均数

```
MPI_Init(NULL, NULL);
int world_rank, world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

float num[100], *num_avg;

for(int i=0; i<100; i++){
        num[i] = world_rank*100+i+1;
}

if(world_rank == 0){
        num_avg = (float *)malloc(100*sizeof(float));
}

MPI_Reduce(num, num_avg, 100, MPI_FLOAT, MPI_SUM,
                        0, MPI_COMM_WORLD);

if(world_rank == 0){
        for(int i=0; i<100; i++){
                num_avg[i] /= world_size;
                printf(" %f ", num_avg[i]);
        }
        free(num_avg);
}

MPI_Finalize();
```

所有进程都需要声明*num_avg

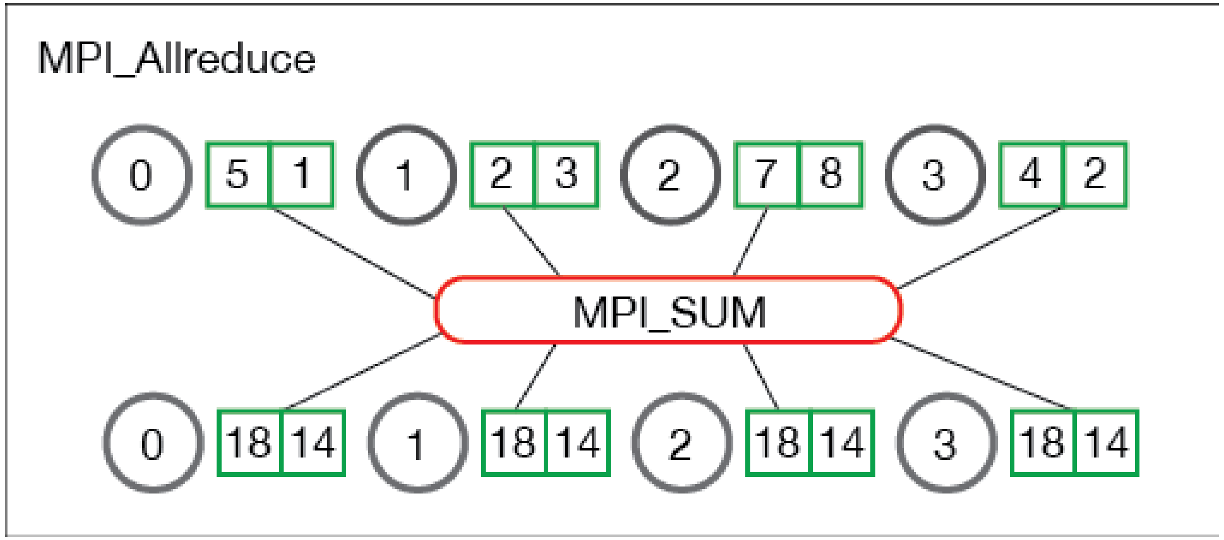
只在进程0中分配内存，以节约内存

MPI缩减操作例子：计算平均数

```
xiangyu@xiangyu-VirtualBox:~/parallel_computing_files$ mpirun -np 3 ./avg_reduc
e
 101.000000  102.000000  103.000000  104.000000  105.000000  106.000000  107.00
0000  108.000000  109.000000  110.000000  111.000000  112.000000  113.000000  1
14.000000  115.000000  116.000000  117.000000  118.000000  119.000000  120.0000
00  121.000000  122.000000  123.000000  124.000000  125.000000  126.000000  127
.000000  128.000000  129.000000  130.000000  131.000000  132.000000  133.000000
  134.000000  135.000000  136.000000  137.000000  138.000000  139.000000  140.0
00000  141.000000  142.000000  143.000000  144.000000  145.000000  146.000000
147.000000  148.000000  149.000000  150.000000  151.000000  152.000000  153.000
000  154.000000  155.000000  156.000000  157.000000  158.000000  159.000000  16
0.000000  161.000000  162.000000  163.000000  164.000000  165.000000  166.00000
0  167.000000  168.000000  169.000000  170.000000  171.000000  172.000000  173.
000000  174.000000  175.000000  176.000000  177.000000  178.000000  179.000000
 180.000000  181.000000  182.000000  183.000000  184.000000  185.000000  186.00
0000  187.000000  188.000000  189.000000  190.000000  191.000000  192.000000  1
93.000000  194.000000  195.000000  196.000000  197.000000  198.000000  199.0000
```

MPI_Allreduce

- 将MPI_Reduce的结果发送到其他所有进程可以直接通过MPI_Allreduce实现
- 即除了根进程外，所有其他进程有时也需要得到缩减后的结果



- MPI_Allreduce = MPI_Reduce + MPI_Bcast

Figure credit: https://mpitutorial.com

MPI_Allreduce

- int MPI_Allreduce(const void *sendbuf,
  void* recvbuf, int count,
  MPI_Datatype datatype,
  MPI_Op op, MPI_Comm comm);

- 和MPI_Reduce相比，不用输入root进程

MPI_Allreduce例子：计算标准差

- 进程0有数组{1,2,...,100}
- 进程1有数组{101,102,...,200}
- 进程2有数组{201,202,...,300}
- 计算合并后数组的标准差
- 利用MPI_Allreduce传递整体均值。

MPI_Allreduce例子：计算标准差

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

float mean(float *num, int length){
	float s = 0;
	for(int i=0; i<length; i++){
		s += num[i];
	}
	s /= length;
	return s;
}

int main(){
	MPI_Init(NULL, NULL);
	int world_rank, world_size;
	MPI_Comm_size(MPI_COMM_WORLD, &world_size);
	MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

	float num[100], num_avg_per_proc, num_avg;

	for(int i=0; i<100; i++){
		num[i] = world_rank*100+i+1;
	}
	num_avg_per_proc = mean(num, 100);
```

MPI_Allreduce例子：计算标准差

```c
	MPI_Allreduce(&num_avg_per_proc, &num_avg, 1, MPI_FLOAT, MPI_SUM,
			MPI_COMM_WORLD);

	num_avg /= 3;

	float sum_sq_per_proc = 0, sum_sq;
	for(int i=0; i<100; i++){
		sum_sq_per_proc += (num[i]-num_avg)*(num[i]-num_avg);
	}

	MPI_Reduce(&sum_sq_per_proc, &sum_sq, 1, MPI_FLOAT, MPI_SUM,
			0, MPI_COMM_WORLD);

	if(world_rank == 0){
		float sd;
		sd = sqrt(sum_sq / (300-1));
		printf("The standard deviation is %f\n", sd);
	}

	MPI_Finalize();
}
```

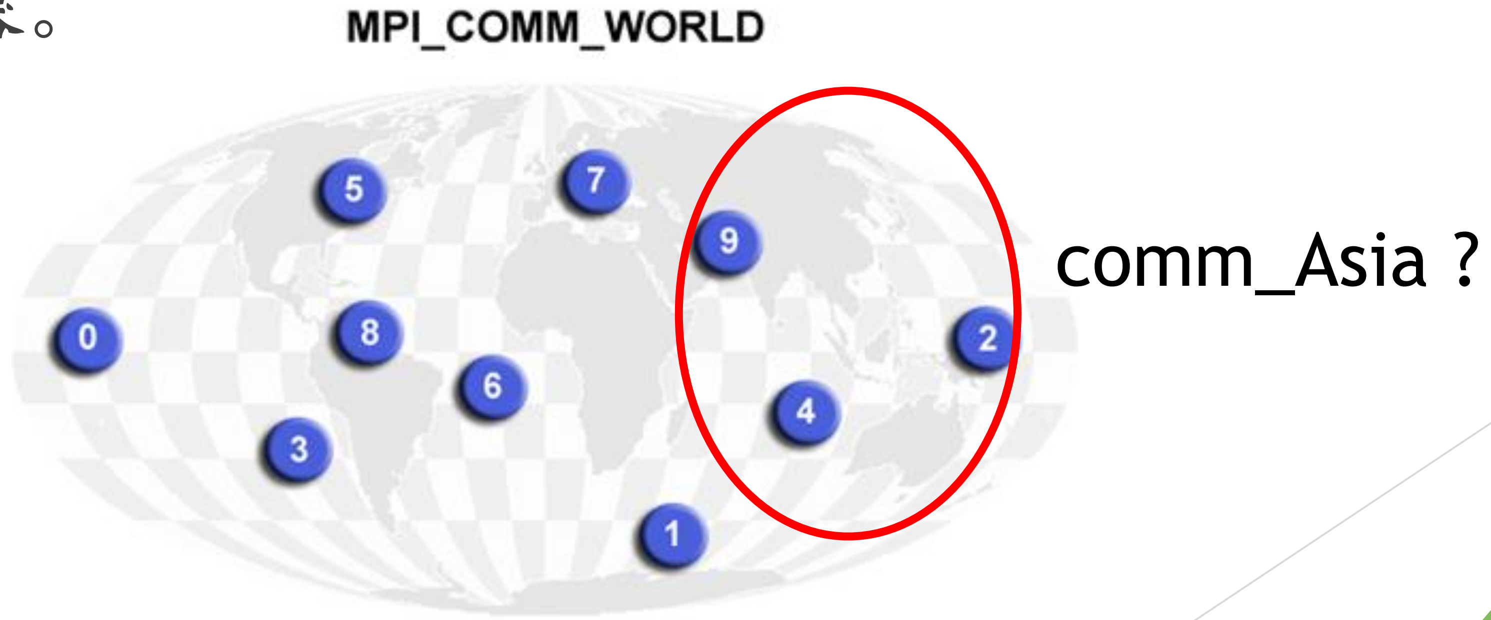
MPI_Allreduce例子：计算标准差

```
xiangyu@xiangyu-VirtualBox:~/parallel_computing_files$ mpicc -o sd_allreduce sd
_allreduce.c -lm
xiangyu@xiangyu-VirtualBox:~/parallel_computing_files$ mpirun -np 3 ./sd_allred
uce
The standard deviation is 86.746758
```

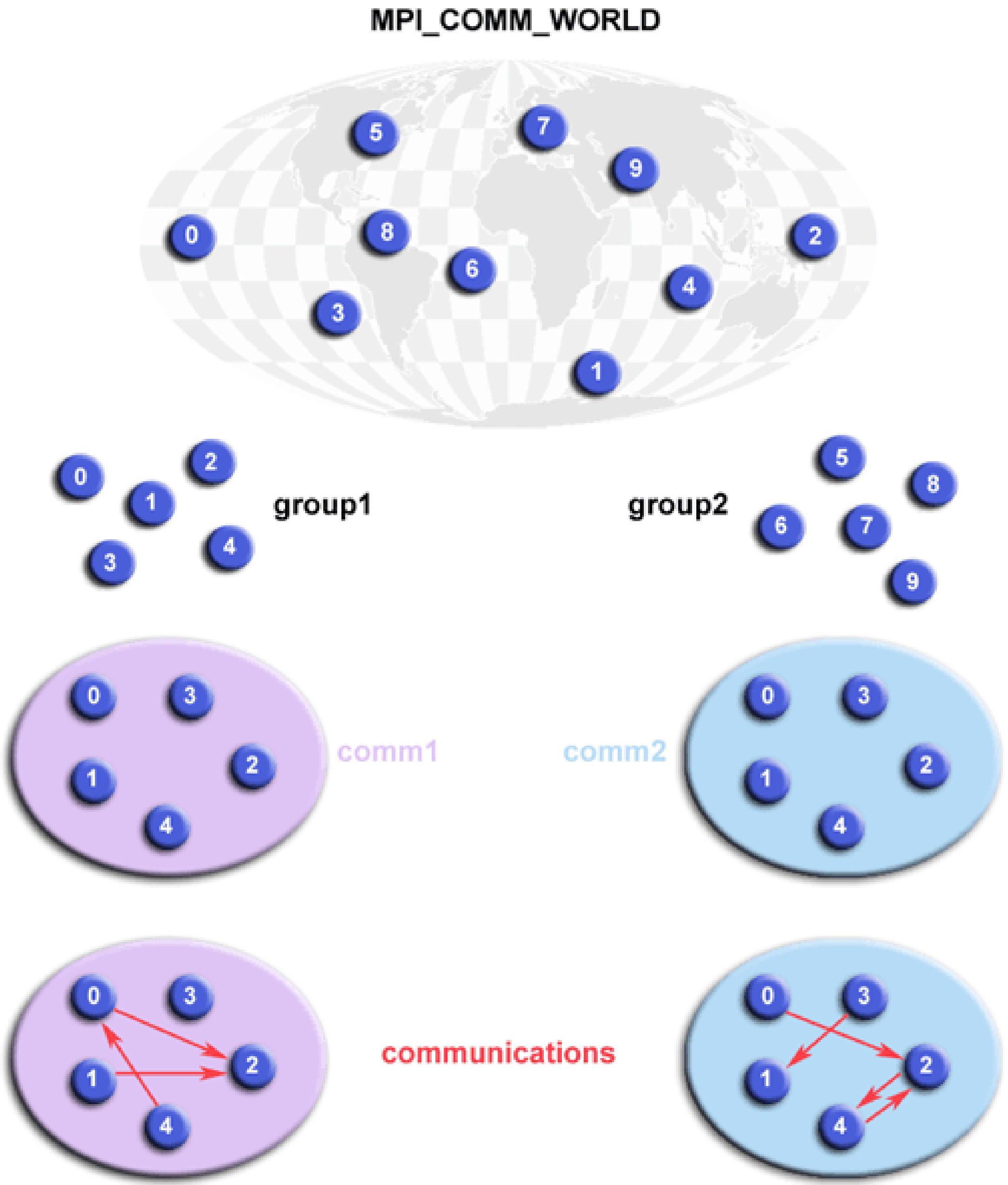
- 代码中我们利用了sqrt函数来计算二次根，此函数来自math头文件，这需要在编译命令中，通过-lm来进行链接(即link math)

组和通信器管理

- 为什么要引入组和通信器?
- 在之前的内容中，我们都是用的世界通信器MPI_COMM_WORLD。对于一些简单的应用，这已经足够了。但是当应用变得更为复杂以后，我们可能只需要一部分进程间通信。所以，我们有必要知道如何定义新的通讯器。



构建新通信器示意图


MPI_COMM_WORLD
5
7
9
0
8
2
6
4
3
1
0
2
1
group1
3
4
group2
5
8
6
7
9
0
3
comm1
1
2
4
0
3
comm2
1
2
4
0
3
1
2
4
communications
0
3
1
2
4

组和通信器的定义

- 一个组(group)指一个有顺序的进程的集合
- 组里的每一个进程都具有一个唯一的整数rank
  - rank值从0开始到S-1结束，其中S指组中进程个数
  - 一个组总是和一个通信器对象相联系
- 一个通信器(communicator)包含一组可能会互相通信的进程
  - 所有MPI消息传递都必须明确一个通信器
  - 比如世界通信器MPI_COMM_WORLD
- 从程序员的角度，组和通信器是一样的。关于组的操作主要明确哪个进程用于构造通信器

组和通信器对象的主要用途

- 根据目的，将进程编组
- 能在一个进程子集上进行集体通信操作
- 为执行用户定义的虚拟拓扑提供基础
- 提供安全的通信

编程时需要注意的地方

- 组或通信器是动态的，它们能在程序执行期间创造或破坏
- 进程可以在多于一个组或通信器中；但在每一个组或通信器中，其具有唯一的rank
- MPI提供了和组、通信器等相关的超过40个常规操作

构建新通信器的方法

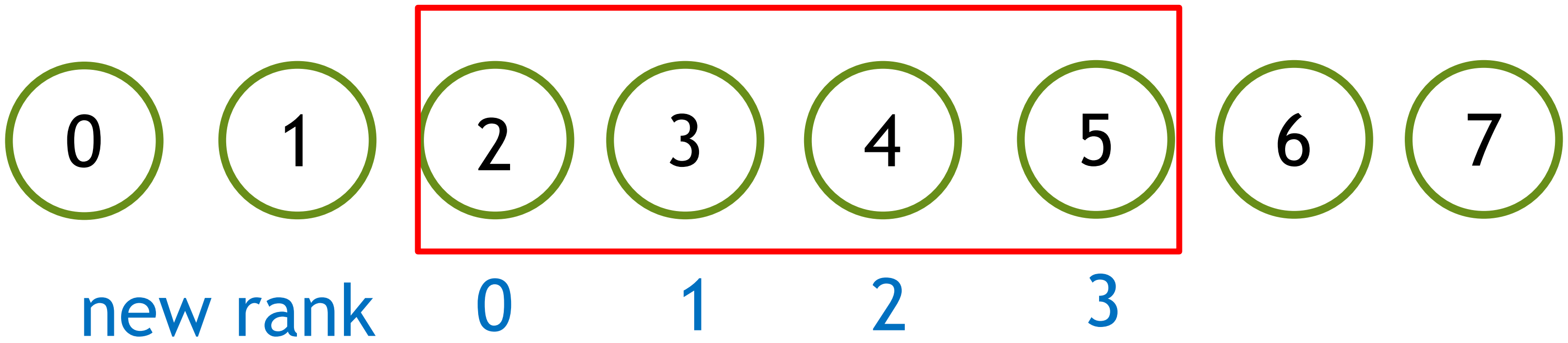
- 方法一
  - 通过MPI_Group_incl构建新的组，再用MPI_Comm_create构建基于此新组的通信器
- 方法二
  - 利用MPI_Comm_split直接划分原通信器来构建新通信器

方法一

- int MPI_Comm_group(MPI_Comm comm, MPI_Group *group);
  - 检索和通信器comm相关联的组
- int MPI_Group_incl(MPI_Group group, int n, int *ranks,
  MPI_Group *newgroup);
  - 在当前group中构建包含特定ranks的n个进程的新组
- int MPI_Comm_create(MPI_Comm comm,
  MPI_Group newgroup, MPI_Comm *newcomm);
  - 给原始通信器comm中的新组newgroup建立对应的通信器

方法一的例子

- 定义进程0、1、…、7
- 定义新的通信器{2,3,4,5}
- 在新通信器中的各个进程，把在世界通信器中的rank归约求和到进程2上



方法一的例子

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

int main(){
        MPI_Init(NULL, NULL);

        int world_size, world_rank, sendbuf, recvbuf;
        int new_rank;
        int ranks[4] = {2,3,4,5};

        MPI_Group world_group, new_group;
        MPI_Comm new_comm;

        MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
        MPI_Comm_size(MPI_COMM_WORLD, &world_size);

        sendbuf = world_rank;

        //extract the world group handle
        MPI_Comm_group(MPI_COMM_WORLD, &world_group);

        MPI_Group_incl(world_group, 4, ranks, &new_group);

        MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);
```

方法一的例子

```
    if(new_comm != MPI_COMM_NULL){
        MPI_Comm_rank(new_comm, &new_rank);
        MPI_Reduce(&sendbuf, &recvbuf, 1, MPI_INT,
                        MPI_SUM, 0, new_comm);
        printf("world rank = %d, new rank = %d\n",
                        world_rank, new_rank);
    }

    if(new_rank == 0){
        printf("*** world rank = %d, new rank = %d, sum = %d\n",
                world_rank, new_rank, recvbuf);
    }


    MPI_Finalize();
}
```

方法一的例子

```
    if(new_comm != MPI_COMM_NULL){
        MPI_Comm_rank(new_comm, &new_rank);
        MPI_Reduce(&sendbuf, &recvbuf, 1, MPI_INT,
                        MPI_SUM, 0, new_comm);
        printf("world rank = %d, new rank = %d\n",
                        world_rank, new_rank);
    }

    if(new_rank == 0){
        printf("*** world rank = %d, new rank = %d, sum = %d\n",
            world_rank, new_rank, recvbuf);
    }


    MPI_Finalize();
}
```

- 请注意！MPI_Reduce中root进程为新通信器中的rank!

方法一的例子

```
xiangyu@xiangyu-VirtualBox:~/parallel_computing_files$ mpicc -o group_comm_ex g
roup_comm_ex.c
xiangyu@xiangyu-VirtualBox:~/parallel_computing_files$ mpirun -np 8 ./group_com
m_ex
world rank = 5, new rank = 3
world rank = 4, new rank = 2
world rank = 3, new rank = 1
world rank = 2, new rank = 0
*** world rank = 2, new rank = 0, sum = 14
```

方法二：利用MPI_Comm_split

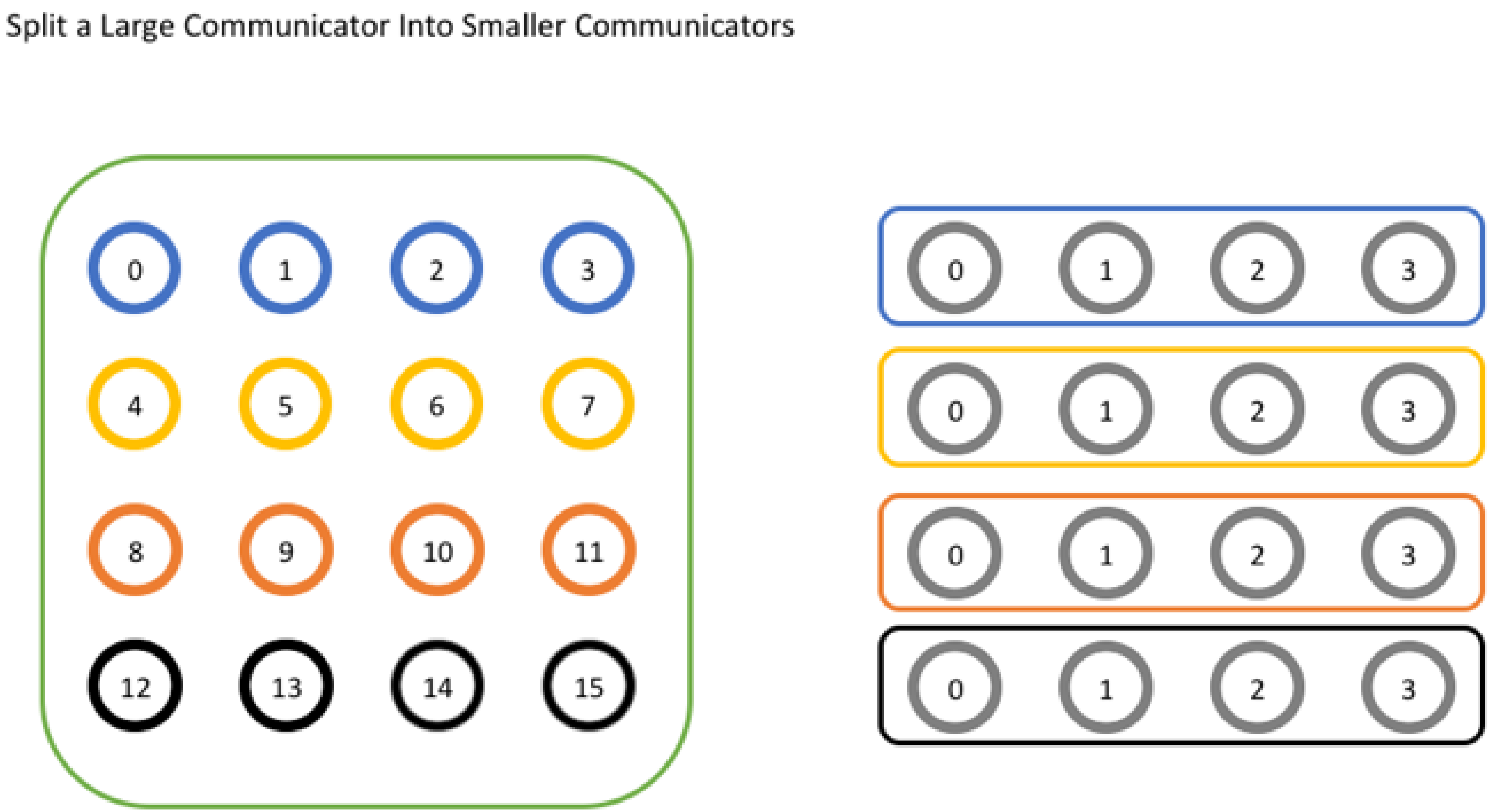


Figure credit: https://mpitutorial.com

方法二：利用MPI_Comm_split

- int MPI_Comm_split(MPI_Comm comm, int color,
  int key, MPI_Comm *newcomm);
- comm指需要进行分裂的通信器
- color指相同color的进程在相同的通信器，color的值需要非负
- key指在新通信器中各个进程的相对rank
  - 对于相同color的两个进程，当key值相同时，排列根据原始通信器中的rank
- 返回值newcomm表示指向一个新通信器的句柄
- MPI_Comm_split构建的通信器是不能相互重叠的
- 如果color是MPI_UNDEFINED，对应的那个进程不被任何一个新的通讯器包含。

方法二的例子

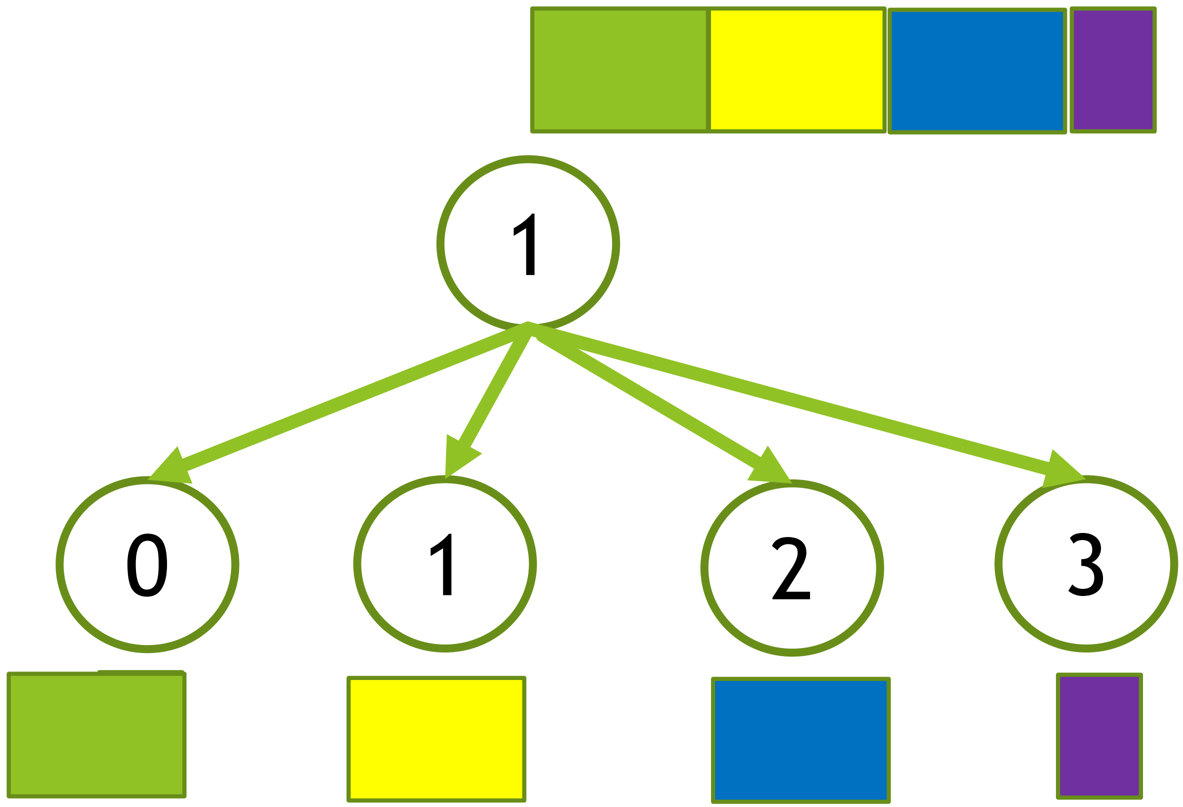
```c
#include <mpi.h>
#include <stdio.h>

int main(){
        MPI_Init(NULL,NULL);

        int world_rank, world_size;
        MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
        MPI_Comm_size(MPI_COMM_WORLD, &world_size);

        int color = world_rank / 4;

        MPI_Comm new_comm;
        MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &new_comm);

        int new_rank, new_size;
        MPI_Comm_rank(new_comm, &new_rank);
        MPI_Comm_size(new_comm, &new_size);

        printf("world rank/size: %d/%d --- new rank/size: %d/%d\n",
                world_rank, world_size, new_rank, new_size);

        MPI_Finalize();
}
```

方法二的例子

```
xiangyu@xiangyu-VirtualBox:~/parallel_computing_files$ mpicc -o split split.c
xiangyu@xiangyu-VirtualBox:~/parallel_computing_files$ mpirun -np 16 ./split
world rank/size: 0/16 --- new rank/size: 0/4
world rank/size: 1/16 --- new rank/size: 1/4
world rank/size: 8/16 --- new rank/size: 0/4
world rank/size: 12/16 --- new rank/size: 0/4
world rank/size: 10/16 --- new rank/size: 2/4
world rank/size: 2/16 --- new rank/size: 2/4
world rank/size: 9/16 --- new rank/size: 1/4
world rank/size: 3/16 --- new rank/size: 3/4
world rank/size: 11/16 --- new rank/size: 3/4
world rank/size: 4/16 --- new rank/size: 0/4
world rank/size: 13/16 --- new rank/size: 1/4
world rank/size: 14/16 --- new rank/size: 2/4
world rank/size: 5/16 --- new rank/size: 1/4
world rank/size: 15/16 --- new rank/size: 3/4
world rank/size: 6/16 --- new rank/size: 2/4
world rank/size: 7/16 --- new rank/size: 3/4
```

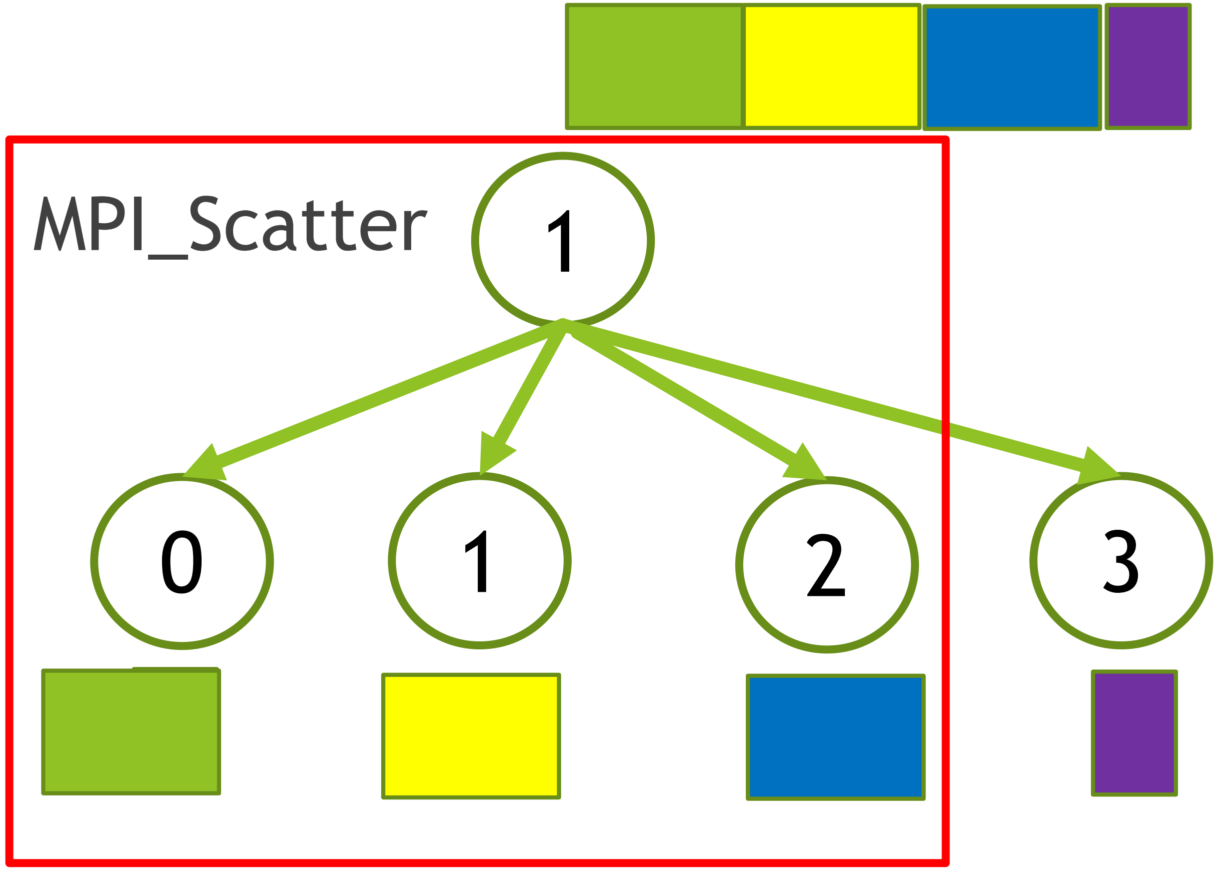
不能均分数据的例子

- MPI_Scatter中数据不可均分的例子
- 进程1的数组{1,2,3,4,5,6,7,8,9,10}分给四个进程



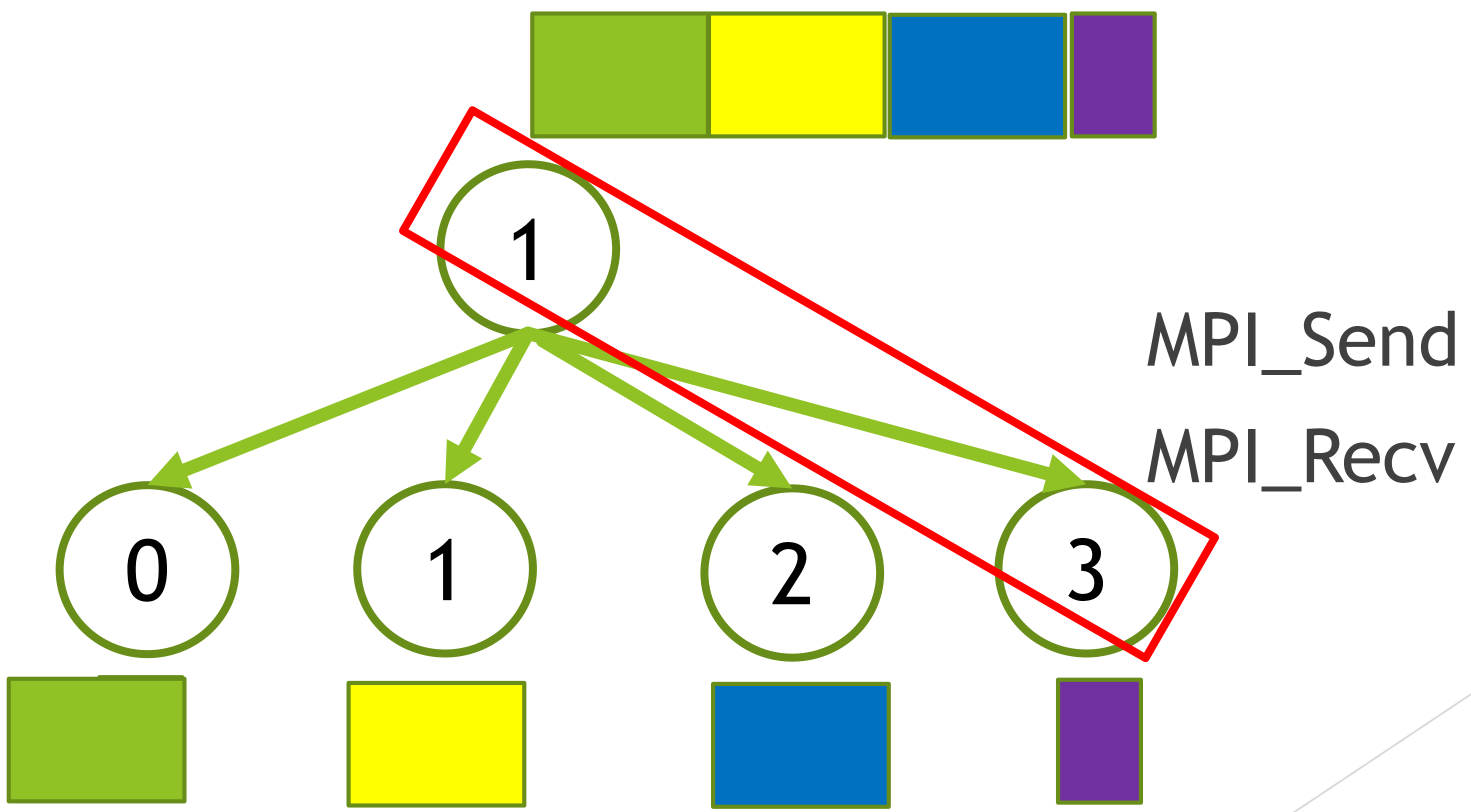
不能均分数据的例子

- 算法思路：先定义一个通信器（0,1,2），在新通信器上用MPI_Scatter，将剩下的利用点对点通信传给进程3。



不能均分数据的例子

- 算法思路：先定义一个通信器（0,1,2），在新通信器上用MPI_Scatter，将剩下的利用点对点通信传给进程3。



不能均分数据的例子

```
int main(){
        MPI_Init(NULL, NULL);

        int world_size, world_rank, *sendbuf, *recvbuf;
        int recvbuf_proc3;
        int new_rank;
        int ranks[3] = {0,1,2};

        MPI_Group world_group, new_group;
        MPI_Comm new_comm;

        MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
        MPI_Comm_size(MPI_COMM_WORLD, &world_size);

        if(world_rank == 1){
                sendbuf = (int *)malloc(10*sizeof(int));
                for(int i=0; i < 10; i++){
                        sendbuf[i] = i+1;
                }
        }

        if(world_rank < 3){
                recvbuf = (int *)malloc(3*sizeof(int));
        }
```

不能均分数据的例子

```
//scatter numbers to processes 0-2
MPI_Comm_group(MPI_COMM_WORLD, &world_group);

MPI_Group_incl(world_group, 3, ranks, &new_group);

MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);

if(new_comm != MPI_COMM_NULL){
        MPI_Scatter(sendbuf, 3, MPI_INT, recvbuf, 3, MPI_INT,
                    1, new_comm);
        printf("Process %d received numbers %d %d %d\n",
                world_rank, recvbuf[0], recvbuf[1], recvbuf[2]);
}

//send the remaining number to process 3
if(world_rank == 1){
        MPI_Send(&(sendbuf[9]), 1, MPI_INT, 3, 888, MPI_COMM_WORLD);
}

if(world_rank == 3){
        MPI_Recv(&recvbuf_proc3, 1, MPI_INT, 1, 888,
                        MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process %d received number %d\n",
                world_rank, recvbuf_proc3);
}
```

不能均分数据的例子

```
xiangyu@xiangyu-VirtualBox:~/parallel_computing_files$ mpirun -np 4 ./scatter_u
nbalanced
Process 1 received numbers 4 5 6
Process 3 received number 10
Process 2 received numbers 7 8 9
Process 0 received numbers 1 2 3
```