

Parallel Programming

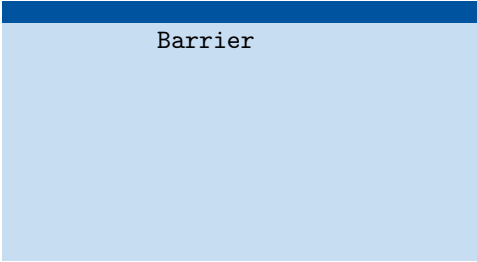
Prof. **Paolo Bientinesi**

`pauldj@aices.rwth-aachen.de`

WS 17/18



Collective Communication



Barrier

Collective Communication

Barrier

Broadcast ↔ Reduce

Collective Communication

Barrier

Broadcast ↔ Reduce

Scatter ↔ Gather

Collective Communication

Barrier

Broadcast ↔ Reduce

Scatter ↔ Gather

Allgather ↔ Reduce-scatter

Collective Communication

Barrier

Broadcast ↔ Reduce

Scatter ↔ Gather

Allgather ↔ Reduce-scatter

Allreduce

Collective Communication

```
Barrier
Broadcast ↔ Reduce
Scatter ↔ Gather
Allgather ↔ Reduce-scatter
Allreduce
Alltoall
⋮
```

Collective Communication

```
Barrier
Broadcast ↔ Reduce
Scatter ↔ Gather
Allgather ↔ Reduce-scatter
Allreduce
Alltoall
⋮
```

References

- “Collective Communication: Theory, Practice, and Experience”, Chan, Heimlich, Purkayastha, van de Geijn. (FLAME working note #22)
- Collective Communications in MPI
<http://www.mcs.anl.gov/research/projects/mpi/tutorial/gropp/node72.html>

Collective Communication

- Synchronization

Barrier

← **Almost never needed!**

Collective Communication

- Synchronization

Barrier ← **Almost never needed!**

- Data Movement

Broadcast, Scatter, Gather, Allgather, Alltoall

Collective Communication

- Synchronization

Barrier ← **Almost never needed!**

- Data Movement

Broadcast, Scatter, Gather, Allgather, Alltoall

- Reductions

Reduce, Reduce-scatter, Allreduce, Scan, ...

Collective Communication

- Synchronization

Barrier ← **Almost never needed!**

- Data Movement

Broadcast, Scatter, Gather, Allgather, Alltoall

- Reductions

Reduce, Reduce-scatter, Allreduce, Scan, ...

All processes invoke the same operation with the same arguments. No tags.

Collective Communication

- Synchronization

Barrier ← **Almost never needed!**

- Data Movement

Broadcast, Scatter, Gather, Allgather, Alltoall

- Reductions

Reduce, Reduce-scatter, Allreduce, Scan, ...

All processes invoke the same operation with the same arguments. No tags.

Collectives are **blocking**, but **not synchronous**: The routines return as soon as their participation in the communication is complete. No guarantee about the status of the receiving processes.

```
int MPI_BCast(...)
```

Before:

Node ₀	Node ₁	Node ₂	Node ₃
		α	

```
int MPI_BCast(...)
```

Before:

Node ₀	Node ₁	Node ₂	Node ₃
		α	

After:

Node ₀	Node ₁	Node ₂	Node ₃
α	α	α	α

```
int MPI_BCast(...)
```

Before:	Node ₀	Node ₁	Node ₂	Node ₃
			α	

After:	Node ₀	Node ₁	Node ₂	Node ₃
	α	α	α	α

- How would you implement the broadcast in terms of individual sends and receives?


```
int MPI_BCast(...)
```

Before:	Node ₀	Node ₁	Node ₂	Node ₃
			α	

After:	Node ₀	Node ₁	Node ₂	Node ₃
	α	α	α	α

- How would you implement the broadcast in terms of individual sends and receives?
- How many steps does it take to broadcast to np processes?

```
int MPI_BCast(...)
```

Before:	Node ₀	Node ₁	Node ₂	Node ₃
			α	

After:	Node ₀	Node ₁	Node ₂	Node ₃
	α	α	α	α

```
MPI_Bcast( Buffer, count, type, root, communicator );
```

```
MPI_Datatype: MPI_CHAR, MPI_INT, MPI_UNSIGNED,  
              MPI_FLOAT, MPI_DOUBLE, ...
```

```
int MPI_Reduce(...)
```

Before:

Node ₀	Node ₁	Node ₂	Node ₃
δ_0	δ_1	δ_2	δ_3

int MPI_Reduce(...)

Before:	Node ₀	Node ₁	Node ₂	Node ₃
	δ_0	δ_1	δ_2	δ_3

After:	Node ₀	Node ₁	Node ₂	Node ₃
	δ_0	δ_1	δ_2 $\text{Op } \delta_i$ $i=0$ ^{$p-1$}	δ_3

int MPI_Reduce(...)

Before:	Node ₀	Node ₁	Node ₂	Node ₃
	δ_0	δ_1	δ_2	δ_3

After:	Node ₀	Node ₁	Node ₂	Node ₃
	δ_0	δ_1	δ_2 $\text{Op } \delta_i$ $i=0$ ^{$p-1$}	δ_3

MPI_Op:

MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_BAND, ...,
MPI_MINLOC, MPI_MAXLOC

int MPI_Reduce(...)

Before:	Node ₀	Node ₁	Node ₂	Node ₃
	δ_0	δ_1	δ_2	δ_3

After:	Node ₀	Node ₁	Node ₂	Node ₃
	δ_0	δ_1	δ_2 $\text{Op } \delta_i$ $i=0$ ^{$p-1$}	δ_3

MPI_Op:

MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_BAND, ...,
MPI_MINLOC, MPI_MAXLOC

Op: Associative. Why is this needed?

```
int MPI_Reduce(...)
```

Before:	Node ₀	Node ₁	Node ₂	Node ₃
	δ_0	δ_1	δ_2	δ_3

After:	Node ₀	Node ₁	Node ₂	Node ₃
	δ_0	δ_1	δ_2 $\text{Op } \delta_i$ $i=0$ ^{$p-1$}	δ_3

```
MPI_Reduce( sendBuffer, recvBuffer, count, type,  
           operation, root, communicator );
```

int MPI_Reduce(...)

Before:	Node ₀	Node ₁	Node ₂	Node ₃
	δ_0	δ_1	δ_2	δ_3

After:	Node ₀	Node ₁	Node ₂	Node ₃
	δ_0	δ_1	δ_2 $\text{Op } \delta_i$ $i=0$ ^{$p-1$}	δ_3

```
MPI_Reduce( sendBuffer, recvBuffer, count, type,  
            operation, root, communicator );
```

```
MPI_Op_create
```


int MPI_Scatter(...)

Before:

Node ₀	Node ₁	Node ₂	Node ₃
		v[0] v[1] v[2] v[3] v[4] v[5] v[6] v[7]	

int MPI_Scatter(...)

	Node ₀	Node ₁	Node ₂	Node ₃
Before:			v[0] v[1] v[2] v[3] v[4] v[5] v[6] v[7]	

	Node ₀	Node ₁	Node ₂	Node ₃
After:	v[0] v[1]	v[2] v[3]	v[0] v[1] v[2] v[3] v[4] v[5] v[6] v[7]	v[6] v[7]

int MPI_Scatter(...)

	Node ₀	Node ₁	Node ₂	Node ₃
Before:			v[0] v[1] v[2] v[3] v[4] v[5] v[6] v[7]	

	Node ₀	Node ₁	Node ₂	Node ₃
After:	v[0] v[1]	v[2] v[3]	v[0] v[1] v[2] v[3] v[4] v[5] v[6] v[7]	v[6] v[7]

```
MPI_Scatter( sendBuffer, sendCount, sendType,  
            recvBuffer, recvCount, recvType,  
            root, communicator );
```

int MPI_Gather(...)

Before:

Node ₀	Node ₁	Node ₂	Node ₃
v[0] v[1]	v[2] v[3]	v[4] v[5]	v[6] v[7]

int MPI_Gather(...)

	Node ₀	Node ₁	Node ₂	Node ₃
Before:	v[0] v[1]	v[2] v[3]	v[4] v[5]	v[6] v[7]

	Node ₀	Node ₁	Node ₂	Node ₃
After:	v[0] v[1]	v[2] v[3]	v[0] v[1] v[2] v[3] v[4] v[5] v[6] v[7]	v[6] v[7]

int MPI_Gather(...)

	Node ₀	Node ₁	Node ₂	Node ₃
Before:	v[0] v[1]	v[2] v[3]	v[4] v[5]	v[6] v[7]

	Node ₀	Node ₁	Node ₂	Node ₃
After:	v[0] v[1]	v[2] v[3]	v[0] v[1] v[2] v[3] v[4] v[5] v[6] v[7]	v[6] v[7]

```
MPI_Gather( sendBuffer, sendCount, sendType,  
           recvBuffer, recvCount, recvType,  
           root, communicator );
```

```
int MPI_Allgather(...)
```

	Node ₀	Node ₁	Node ₂	Node ₃
Before:	v[0]	v[1]	v[2]	v[3]

```
int MPI_Allgather(...)
```

	Node ₀	Node ₁	Node ₂	Node ₃
Before:	v[0]	v[1]	v[2]	v[3]

	Node ₀	Node ₁	Node ₂	Node ₃
After:	v[0]	v[0]	v[0]	v[0]
	v[1]	v[1]	v[1]	v[1]
	v[2]	v[2]	v[2]	v[2]
	v[3]	v[3]	v[3]	v[3]


```
int MPI_Allgather(...)
```

	Node ₀	Node ₁	Node ₂	Node ₃
Before:	v[0]	v[1]	v[2]	v[3]

	Node ₀	Node ₁	Node ₂	Node ₃
After:	v[0]	v[0]	v[0]	v[0]
	v[1]	v[1]	v[1]	v[1]
	v[2]	v[2]	v[2]	v[2]
	v[3]	v[3]	v[3]	v[3]

```
MPI_Allgather( sendBuffer, sendCount, sendType,  
              recvBuffer, recvCount, recvType,  
              communicator );
```

```
int MPI_Reduce_scatter(...)
```

	Node ₀	Node ₁	Node ₂	Node ₃
Before:	v ₀ [0]	v ₁ [0]	v ₂ [0]	v ₃ [0]
	v ₀ [1]	v ₁ [1]	v ₂ [1]	v ₃ [1]
	v ₀ [2]	v ₁ [2]	v ₂ [2]	v ₃ [2]
	v ₀ [3]	v ₁ [3]	v ₂ [3]	v ₃ [3]

int MPI_Reduce_scatter(...)

	Node ₀	Node ₁	Node ₂	Node ₃
Before:	v ₀ [0]	v ₁ [0]	v ₂ [0]	v ₃ [0]
	v ₀ [1]	v ₁ [1]	v ₂ [1]	v ₃ [1]
	v ₀ [2]	v ₁ [2]	v ₂ [2]	v ₃ [2]
	v ₀ [3]	v ₁ [3]	v ₂ [3]	v ₃ [3]

	Node ₀	Node ₁	Node ₂	Node ₃
After:	Op v _i [0] _i			
		Op v _i [1] _i		
			Op v _i [2] _i	
				Op v _i [3] _i

int MPI_Reduce_scatter(...)

	Node ₀	Node ₁	Node ₂	Node ₃
Before:	v ₀ [0]	v ₁ [0]	v ₂ [0]	v ₃ [0]
	v ₀ [1]	v ₁ [1]	v ₂ [1]	v ₃ [1]
	v ₀ [2]	v ₁ [2]	v ₂ [2]	v ₃ [2]
	v ₀ [3]	v ₁ [3]	v ₂ [3]	v ₃ [3]

	Node ₀	Node ₁	Node ₂	Node ₃
After:	Op v _i [0] _i			
		Op v _i [1] _i		
			Op v _i [2] _i	
				Op v _i [3] _i

```
MPI_Reduce_scatter( sendBuffer,  recvBuffer,  
                    recvCounts[], type, operation,  
                    communicator );
```

```
int MPI_Allreduce(...)
```

Before:

Node ₀	Node ₁	Node ₂	Node ₃
δ_0	δ_1	δ_2	δ_3

```
int MPI_Allreduce(...)
```

Before:	Node ₀	Node ₁	Node ₂	Node ₃
	δ_0	δ_1	δ_2	δ_3

After:	Node ₀	Node ₁	Node ₂	Node ₃
	$\text{Op}_{i=0}^{p-1} \delta_i$	$\text{Op}_{i=0}^{p-1} \delta_i$	$\text{Op}_{i=0}^{p-1} \delta_i$	$\text{Op}_{i=0}^{p-1} \delta_i$

```
int MPI_Allreduce(...)
```

Before:	Node ₀	Node ₁	Node ₂	Node ₃
	δ_0	δ_1	δ_2	δ_3

After:	Node ₀	Node ₁	Node ₂	Node ₃
	$\prod_{i=0}^{p-1} \delta_i$	$\prod_{i=0}^{p-1} \delta_i$	$\prod_{i=0}^{p-1} \delta_i$	$\prod_{i=0}^{p-1} \delta_i$

```
MPI_Allreduce( sendBuffer,  recvBuffer,  
              count, type, operation,  
              communicator );
```

```
int MPI_Alltoall(...)
```

	Node ₀	Node ₁	Node ₂	Node ₃
Before:	v ₀ [0]	v ₁ [0]	v ₂ [0]	v ₃ [0]
	v ₀ [1]	v ₁ [1]	v ₂ [1]	v ₃ [1]
	v ₀ [2]	v ₁ [2]	v ₂ [2]	v ₃ [2]
	v ₀ [3]	v ₁ [3]	v ₂ [3]	v ₃ [3]


```
int MPI_Alltoall(...)
```

	Node ₀	Node ₁	Node ₂	Node ₃
Before:	v ₀ [0]	v ₁ [0]	v ₂ [0]	v ₃ [0]
	v ₀ [1]	v ₁ [1]	v ₂ [1]	v ₃ [1]
	v ₀ [2]	v ₁ [2]	v ₂ [2]	v ₃ [2]
	v ₀ [3]	v ₁ [3]	v ₂ [3]	v ₃ [3]

	Node ₀	Node ₁	Node ₂	Node ₃
After:	v ₀ [0]	v ₀ [1]	v ₀ [2]	v ₀ [3]
	v ₁ [0]	v ₁ [1]	v ₁ [2]	v ₁ [3]
	v ₂ [0]	v ₂ [1]	v ₂ [2]	v ₂ [3]
	v ₃ [0]	v ₃ [1]	v ₃ [2]	v ₃ [3]

```
int MPI_Alltoall(...)
```

	Node ₀	Node ₁	Node ₂	Node ₃
Before:	v ₀ [0]	v ₁ [0]	v ₂ [0]	v ₃ [0]
	v ₀ [1]	v ₁ [1]	v ₂ [1]	v ₃ [1]
	v ₀ [2]	v ₁ [2]	v ₂ [2]	v ₃ [2]
	v ₀ [3]	v ₁ [3]	v ₂ [3]	v ₃ [3]

	Node ₀	Node ₁	Node ₂	Node ₃
After:	v ₀ [0]	v ₀ [1]	v ₀ [2]	v ₀ [3]
	v ₁ [0]	v ₁ [1]	v ₁ [2]	v ₁ [3]
	v ₂ [0]	v ₂ [1]	v ₂ [2]	v ₂ [3]
	v ₃ [0]	v ₃ [1]	v ₃ [2]	v ₃ [3]

```
MPI_Alltoall( sendBuffer, sendCount, sendType,  
              recvBuffer, recvCount, recvType,  
              communicator );
```

More Collectives

Variable length

- `MPI_Scatterv`
- `MPI_Gatherv`
- `MPI_Allgatherv`
- `MPI_Alltoallv`

More Collectives

Variable length

- `MPI_Scatterv`
- `MPI_Gatherv`
- `MPI_Allgatherv`
- `MPI_Alltoallv`

```
MPI_Scatter(  
    sendBuffer, sendCount,  
            sendType,  
    recvBuffer, recvCount,  
            recvType,  
    root, communicator );
```

```
MPI_Scatterv(  
    sendBuffer, sendCounts[],  
    displ[], sendType,  
    recvBuffer, recvCount,  
            recvType,  
    root, communicator )
```

Even more Collectives

Partial reduction

- `MPI_Scan`

Non-blocking collectives

- `MPI_I*`

```
int MPI_Scan(...)
```

Before:

Node ₀	Node ₁	Node ₂	...	Node _{p-1}
δ_0	δ_1	δ_2	...	δ_{p-1}

int MPI_Scan(...)

Before:	Node ₀	Node ₁	Node ₂	...	Node _{p-1}
	δ_0	δ_1	δ_2	...	δ_{p-1}

After:	Node ₀	Node ₁	Node ₂	...	Node _{p-1}
	δ_0	$\overset{1}{\text{Op}} \delta_i$ <small>$i=0$</small>	$\overset{2}{\text{Op}} \delta_i$ <small>$i=0$</small>	...	$\overset{p-1}{\text{Op}} \delta_i$ <small>$i=0$</small>

int MPI_Scan(...)

Before:	Node ₀	Node ₁	Node ₂	...	Node _{p-1}
	δ_0	δ_1	δ_2	...	δ_{p-1}

After:	Node ₀	Node ₁	Node ₂	...	Node _{p-1}
	δ_0	$\overset{1}{\text{Op}}_{i=0} \delta_i$	$\overset{2}{\text{Op}}_{i=0} \delta_i$...	$\overset{p-1}{\text{Op}}_{i=0} \delta_i$

```
MPI_Scan( sendBuffer, recvBuffer,  
          count, type, operation, communicator );
```



```
int MPI_Exscan(...)
```

Before:

Node ₀	Node ₁	Node ₂		Node _{p-1}
δ_0	δ_1	δ_2	...	δ_{p-1}

int MPI_Exscan(...)

Before:	Node ₀	Node ₁	Node ₂	...	Node _{p-1}
	δ_0	δ_1	δ_2	...	δ_{p-1}

After:	Node ₀	Node ₁	Node ₂	...	Node _{p-1}
	-	δ_0	$\overset{1}{\text{Op}} \delta_i$ $i=0$...	$\overset{p-2}{\text{Op}} \delta_i$ $i=0$

int MPI_Exscan(...)

Before:	Node ₀	Node ₁	Node ₂	...	Node _{p-1}
	δ_0	δ_1	δ_2	...	δ_{p-1}

After:	Node ₀	Node ₁	Node ₂	...	Node _{p-1}
	-	δ_0	$\overset{1}{\text{Op}} \delta_i$ $i=0$...	$\overset{p-2}{\text{Op}} \delta_i$ $i=0$

```
MPI_Exscan( sendBuffer, recvBuffer,  
            count, type, operation, communicator );
```