

Parallel Programming

Prof. **Paolo Bientinesi**

`pauldj@aices.rwth-aachen.de`

WS 17/18



Exercise

`MPI_Irecv`

`MPI_Wait`

`== ??`

`MPI_Recv`

Exercise

```
MPI_Irecv
```

```
MPI_Wait
```

```
== ??
```

```
MPI_Recv
```

```
== ??
```

```
MPI_Irecv
```

```
while( flag==0 ) MPI_Test
```

Wildcards

Process i	Process j
<code>send(&a, ..., j, ...);</code>	<code>recv(&b, ..., i, ...);</code>

- What are we doing?

Wildcards

Process i	Process j
<code>send(&a, ..., j, ...);</code>	<code>recv(&b, ..., i, ...);</code>

- What are we doing?

$$b^{(j)} := a^{(i)}$$

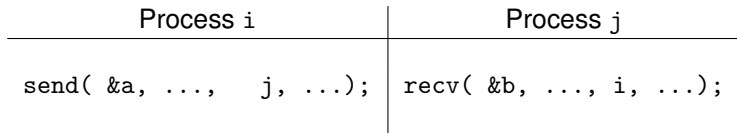
(PGAS: Partitioned Global Address Space Languages)

Wildcards

Process i	Process j
<code>send(&a, ..., j, ...);</code>	<code>recv(&b, ..., i, ...);</code>

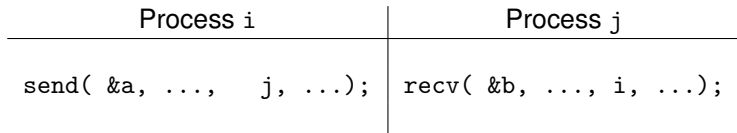
- What are we doing? $b^{(j)} := a^{(i)}$ (PGAS: Partitioned Global Address Space Languages)
Hint: Mentally, associate a time diagram to the operation

Wildcards



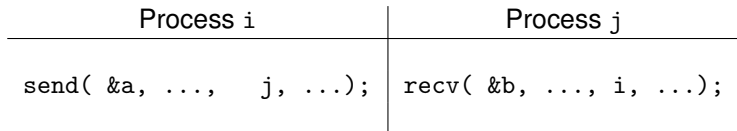
- What are we doing? $b^{(j)} := a^{(i)}$ (PGAS: Partitioned Global Address Space Languages)
Hint: Mentally, associate a time diagram to the operation
- Wildcards: `MPI_ANY_SOURCE`, `MPI_ANY_TAG`

Wildcards



- What are we doing? $b^{(j)} := a^{(i)}$ (PGAS: Partitioned Global Address Space Languages)
Hint: Mentally, associate a time diagram to the operation
- Wildcards: `MPI_ANY_SOURCE`, `MPI_ANY_TAG`
- ... but then, “from whom did I receive?”,
and most importantly, “what is the size of the message?”

Wildcards



- What are we doing? $b^{(j)} := a^{(i)}$ (PGAS: Partitioned Global Address Space Languages)
Hint: Mentally, associate a time diagram to the operation
- Wildcards: `MPI_ANY_SOURCE`, `MPI_ANY_TAG`
- ... but then, “from whom did I receive?”,
and most importantly, “what is the size of the message?”
- `MPI_Status` (or `MPI_STATUS_IGNORE`)

- Matching datatypes?

- Matching datatypes? Not really

But then ...

```
Proc i:  MPI_Send( &n, 1, MPI_INT,    z, 111, comm );  
Proc j:  MPI_Send( &x, 1, MPI_DOUBLE, z, 111, comm );  
Proc z:  MPI_Recv( ..., MPI_ANY_SOURCE, 111, comm, &status );
```

What does Proc z receive?

- Matching datatypes? Not really

But then ...

```
Proc i:  MPI_Send( &n, 1, MPI_INT,    z, 111, comm );
Proc j:  MPI_Send( &x, 1, MPI_DOUBLE, z, 111, comm );
Proc z:  MPI_Recv( ..., MPI_ANY_SOURCE, 111, comm, &status );
```

What does Proc z receive?

Solution: MPI_Probe, MPI_Iprobe

```
MPI_Probe( MPI_ANY_SOURCE, 111, comm, &status );
if( status.MPI_SOURCE == i )
    MPI_Recv( ..., MPI_INT, i, 111, comm, &status );
if( status.MPI_SOURCE == j )
    MPI_Recv( ..., MPI_DOUBLE, j, 111, comm, &status );
```

- Matching number of sends and receives?

Process i	Process j
<code>send(...,1, ..., j, ...);</code> <code>send(...,1, ..., j, ...);</code>	<code>recv(..., 2, ..., i, ...);</code>

- Matching number of sends and receives? yes

Process i	Process j
<pre>send(...,1, ..., j, ...); send(...,1, ..., j, ...);</pre>	<pre>recv(..., 2, ..., i, ...);</pre>

NOT valid!

Recap: Deadlock

Recap: Deadlock

- 2+ processes want to exchange data
- A closed chain of processes (cycle), each one waiting for another, is formed.
⇒ BUG: **deadlock**

Recap: Deadlock

- 2+ processes want to exchange data
- A closed chain of processes (cycle), each one waiting for another, is formed.
⇒ BUG: **deadlock**
- Example: All processes start with a blocking send or a blocking receive
Ssend, Send (in the worst case), Recv

Recap: Deadlock

- 2+ processes want to exchange data
- A closed chain of processes (cycle), each one waiting for another, is formed.
⇒ BUG: **deadlock**
- Example: All processes start with a blocking send or a blocking receive `Ssend`, `Send` (in the worst case), `Recv`
- Solution: BREAK SYMMETRY!
At the same time, careful not to serialize the code!
Approach: code, test and debug with `Ssend`; then replace with `Send`
- Other solutions?

Recap: Deadlock

- 2+ processes want to exchange data
- A closed chain of processes (cycle), each one waiting for another, is formed.
⇒ BUG: **deadlock**
- Example: All processes start with a blocking send or a blocking receive `Ssend`, `Send` (in the worst case), `Recv`
- Solution: BREAK SYMMETRY!
At the same time, careful not to serialize the code!
Approach: code, test and debug with `Ssend`; then replace with `Send`
- Other solutions?
 - Non-blocking send (`Isend`)
 - Non-blocking receive (`Irecv`)
 - Simultaneous send-receive (`Sendrecv`)