Parallel Programming

Prof. **Paolo Bientinesi**

pauldj@aices.rwth-aachen.de

WS 16/17

Point-to-point communication

Send

- `MPI_Ssend`
- `MPI_Send`
- `MPI_Isend`

  :

- `MPI_Bsend`

Receive

- `MPI_Recv`
- `MPI_Irecv`

Point-to-point communication

Send

- `MPI_Ssend`
- `MPI_Send`
- `MPI_Isend`

  ⋮
- `MPI_Bsend`

Receive

- `MPI_Recv`
- `MPI_Irecv`

Send+Receive

- `MPI_Sendrecv`
- `MPI_Sendrecv_replace`

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`
`send( &a, ..., j, ...);`	`recv( &b, ..., i, ...);`

- What are we doing?

Wildcards

Process `i`	Process `j`
`send( &a, ..., j, ...);`	`recv( &b, ..., i, ...);`

- What are we doing? $b^{(j)} := a^{(i)}$

(PGAS: Partitioned Global Address Space Languages)

Wildcards

Process `i`	Process `j`
`send( &a, ..., j, ...);`	`recv( &b, ..., i, ...);`

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

(PGAS: Partitioned Global Address Space Languages)

Wildcards

Process `i`	Process `j`
`send( &a, ..., j, ...);`	`recv( &b, ..., i, ...);`

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

(PGAS: Partitioned Global Address Space Languages)

- Wildcards: `MPI_ANY_SOURCE`, `MPI_ANY_TAG`

Wildcards

Process `i`	Process `j`
`send( &a, ..., j, ...);`	`recv( &b, ..., i, ...);`

- What are we doing? $\mathtt{b}^{(\mathtt{j})} := \mathtt{a}^{(\mathtt{i})}$
  Hint: Mentally, associate a time diagram to the operation

(PGAS: Partitioned Global Address Space Languages)

- 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

Process `i`	Process `j`
`send( &a, ..., j, ...);`	`recv( &b, ..., i, ...);`

- What are we doing? $\texttt{b}^{(\texttt{j})} := \texttt{a}^{(\texttt{i})}$
  Hint: Mentally, associate a time diagram to the operation

(PGAS: Partitioned Global Address Space Languages)

- 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`)

Request, Status

```
MPI_Status status;
MPI_Request requestS, requestR;

MPI_Isend( send, size, type, dest, tag, COMM, &requestS );
...
MPI_Recv ( recv, size, type, root, tag, COMM, &status );
MPI_Irecv( recv, size, type, root, tag, COMM, &requestR );
```

Request, Status

```
MPI_Status status;
MPI_Request requestS, requestR;

MPI_Isend( send, size, type, dest, tag, COMM, &requestS );
...
MPI_Recv ( recv, size, type, root, tag, COMM, &status );
MPI_Irecv( recv, size, type, root, tag, COMM, &requestR );
```

```
int MPI_Wait(
    MPI_Request *request,
    MPI_Status *status
)
```

```
int MPI_Test(
    MPI_Request *request,
    int *flag,
    MPI_Status *status
)
```

`MPI_Waitany`, `MPI_Waitall`, `MPI_Waitsome`, `MPI_Testany`, `MPI_Testall`, `MPI_Testsome`

In all cases, every receive has a corresponding status.

Request, Status

```
MPI_Status status;
MPI_Request requestS, requestR;

MPI_Isend( send, size, type, dest, tag, COMM, &requestS );
...
MPI_Recv ( recv, size, type, root, tag, COMM, &status );
MPI_Irecv( recv, size, type, root, tag, COMM, &requestR );
```

```
int MPI_Wait(
    MPI_Request *request,
    MPI_Status *status
)
```

```
int MPI_Test(
    MPI_Request *request,
    int *flag,
    MPI_Status *status
)
```

`MPI_Waitany`, `MPI_Waitall`, `MPI_Waitsome`, `MPI_Testany`, `MPI_Testall`, `MPI_Testsome`

In all cases, every receive has a corresponding status.

MPI_Status

```
status.MPI_SOURCE
status.MPI_TAG
status.MPI_ERROR
```

```
MPI_GET_COUNT(
    status, datatype, count
    )
```

- 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`
`send(...,1, ..., j, ...);`	
`send(...,1, ..., j, ...);`	
	`recv(..., 2, ..., i, ...);`

- Matching number of sends and receives? yes

Process `i`	Process `j`
`send(...,1, ..., j, ...);`	
`send(...,1, ..., j, ...);`	
	`recv(..., 2, ..., i, ...);`

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.
  $\Rightarrow$ BUG: deadlock

Recap: Deadlock

- 2+ processes want to exchange data
- A closed chain of processes (cycle), each one waiting for another, is formed.
  $\Rightarrow$ 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.
  $\Rightarrow$ 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.
  $\Rightarrow$ 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`)

Persistent communication

Optimization

```
while(1){
    ...
    x = ...;
    MPI_Send( &x, n, type, dest, tag, comm );
    ...
}
```

Persistent communication

Optimization

```
while(1){
    ...
    x = ...;
    MPI_Send( &x, n, type, dest, tag, comm );
    ...
}
```

- `MPI_Send_init, MPI_Recv_init`

  binds all the arguments of a send (receive), for later reuse

- `MPI_Start`

  initiates the send (receive)

1-sided communication

Communication happens without the agreement of both sides!

- `MPI_Put`

  write into target's memory

- `MPI_Get`

  read from target's memory

- `MPI_Win_create, MPI_Win_start, MPI_Win_complete, ...`

  define & manage memory space accessible from other processes

Datatypes

- So far: { `memory address, count, datatype` }

  $\Rightarrow$ only contiguous entries

  only entries of the same MPI type

Datatypes

- So far: { `memory address, count, datatype` }

  $\Rightarrow$ only contiguous entries

  only entries of the same MPI type

- What if . . .
  non contiguous data and/or **non elementary datatypes**?

  Examples: vector from matrix, submatrix, descriptor+data, ...

Datatypes

- So far: { `memory address, count, datatype` }

  $\Rightarrow$ only contiguous entries

  only entries of the same MPI type

- What if . . .
  non contiguous data and/or **non elementary datatypes**?

  Examples: vector from matrix, submatrix, descriptor+data, ...

- Entirely wrong idea: ~~many small messages~~

Datatypes

- So far: { `memory address, count, datatype` }

  ⇒ only contiguous entries

  only entries of the same MPI type

- What if . . .
  non contiguous data and/or **non elementary datatypes**?

  Examples: vector from matrix, submatrix, descriptor+data, ...

- Entirely wrong idea: ~~many small messages~~

- MPI **derived datatypes**: "Create, commit, use, free"

```
MPI_Datatype newtype;
MPI_Type_*( ..., &newtype);
MPI_Type_commit( &newtype );

// code

MPI_Type_free( &newtype );
```

- `int MPI_Type_contiguous(`
  `int count, MPI_Datatype old_type, MPI_Datatype *new_type )`

Same as sending `count` entries of `old_type`

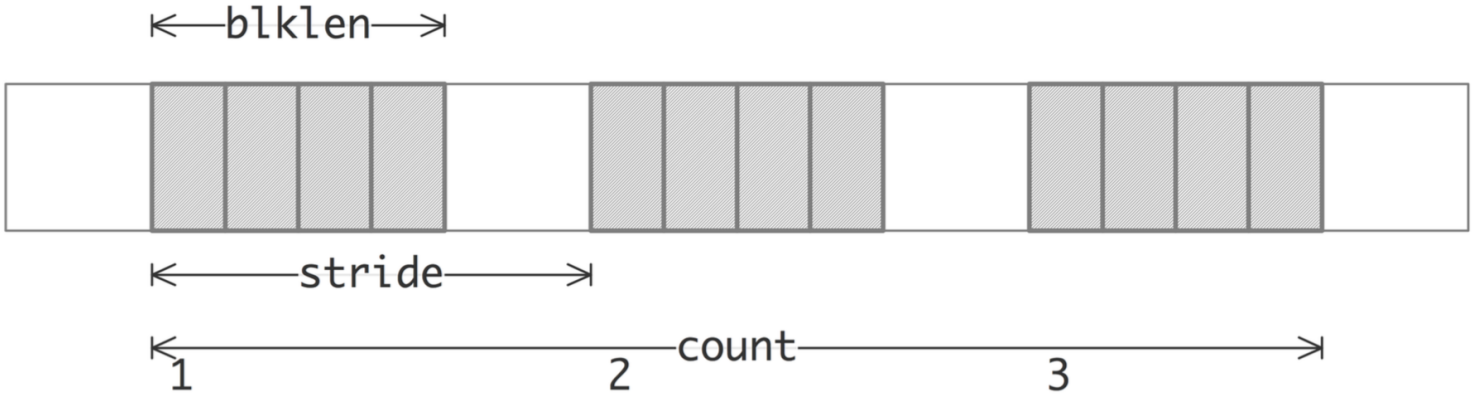
- `int MPI_Type_contiguous(`
  `int count, MPI_Datatype old_type, MPI_Datatype *new_type )`

Same as sending `count` entries of `old_type`

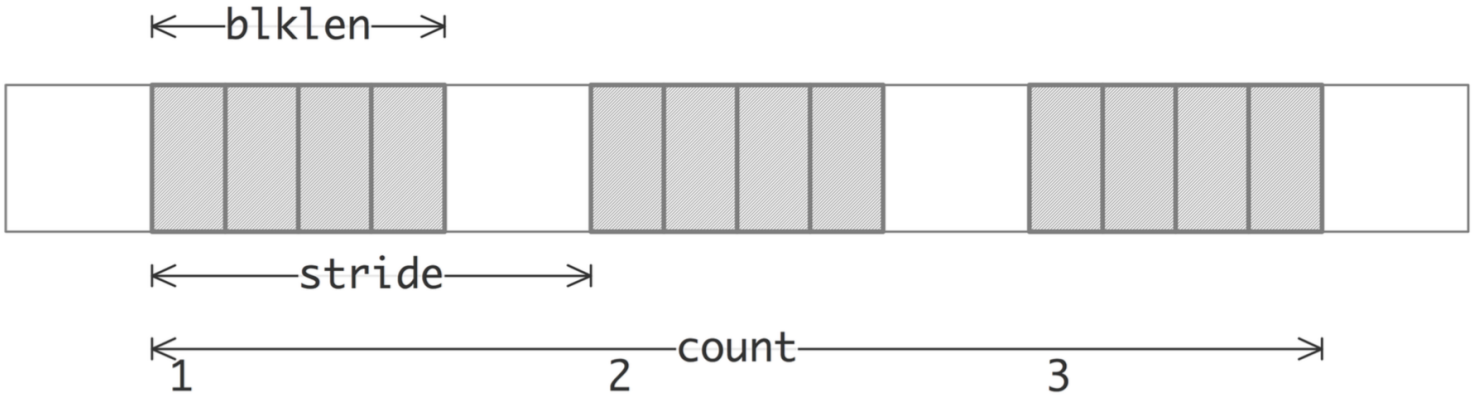
Reference

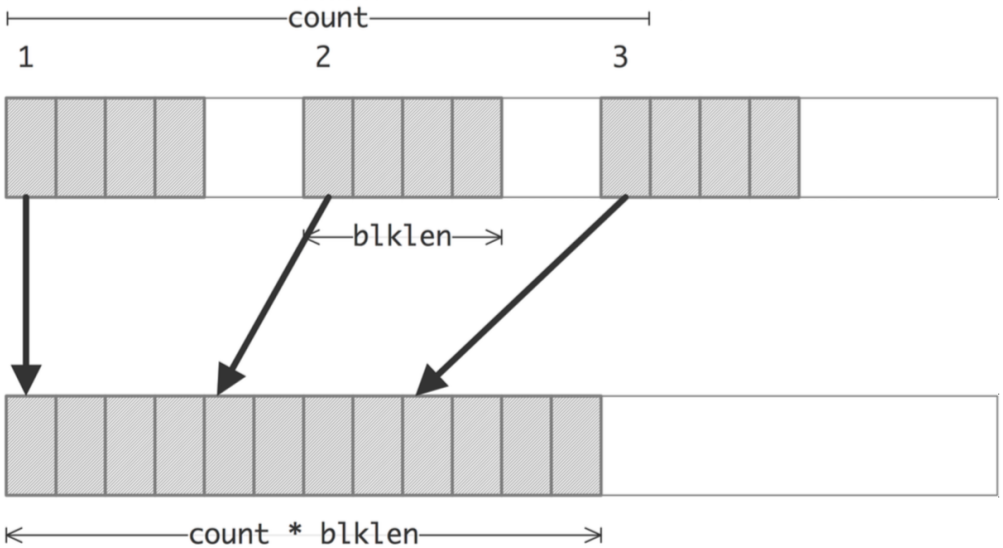
- "Parallel Programming in MPI and OpenMP"
  Victor Eijkhout, Texas dvanced Computing Center
  available online:
  `http://pages.tacc.utexas.edu/~eijkhout/pcse/html/index.html`

- `int MPI_Type_vector(`
  `int count, int blklen, int stride,`
  `MPI_Datatype old_type, MPI_Datatype *new_type )`



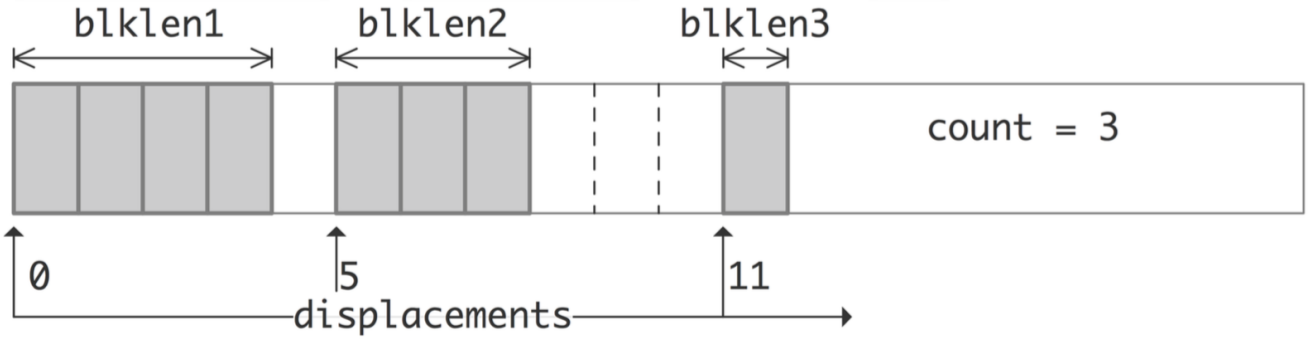
- `int MPI_Type_vector(`
  `int count, int blklen, int stride,`
  `MPI_Datatype old_type, MPI_Datatype *new_type )`



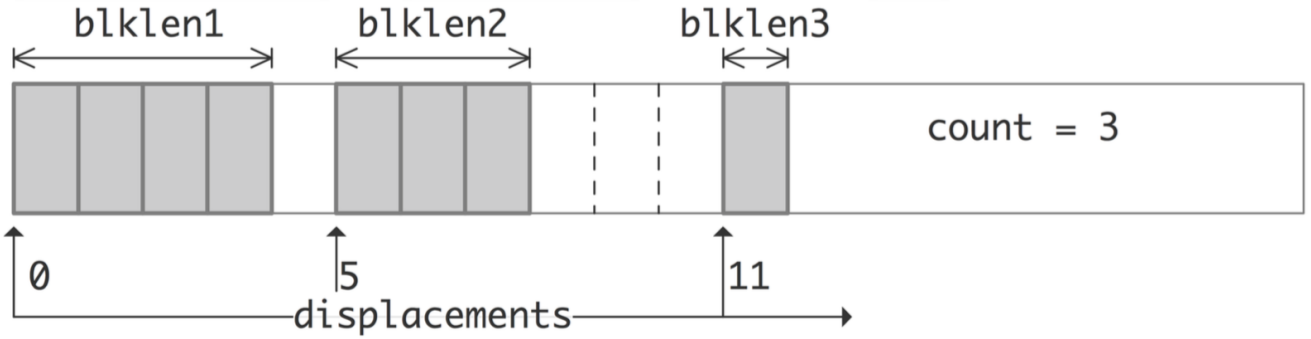


Receive type can be different from Send type

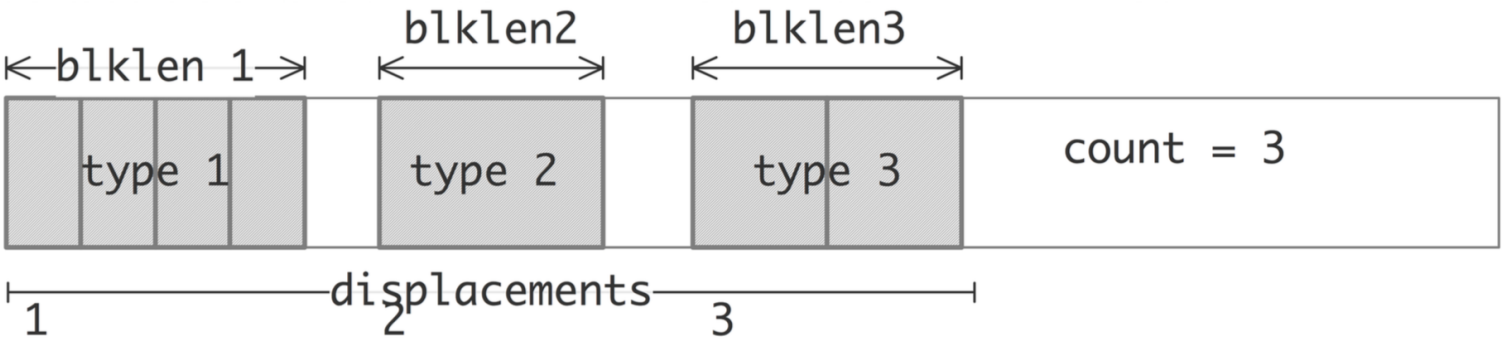
- `int MPI_Type_indexed(`
  `int count, int blklens[], int indices[],`
  `MPI_Datatype old_type, MPI_Datatype *new_type )`



- `int MPI_Type_indexed(`
  `int count, int blklens[], int indices[],`
  `MPI_Datatype old_type, MPI_Datatype *new_type )`



- `int MPI_Type_create_struct(`
  `int count, int blklen[], MPI_Aint displacements[],`
  `MPI_Datatype types[], MPI_Datatype *new_type )`



Exercise

The `root` process owns an array `v` of length 10*`p`,
where `p` is the number of processes participating in the computation.

The entries at index 0, `p`, `2p`, ..., `9p`, need to be sent to process 0;

the entries at index 1, `p+1`, `2p+1`, ..., `9p+1`, need to be sent to process 1;

⋮

Write a program that performs this distribution using a vector datatype for the send,
and a contiguous buffer for the receive.

More

- `MPI_Type_create_subarray`
  Subarray of a regular, multidimensional array

- `MPI_Type_create_darray`
  Distributed array

...and more

- `MPI_Type_extent`
  Memory span by a datatype (extension of `sizeof`)

- `MPI_Pack, MPI_Unpack`
  Pack/unpack memory into contiguous memory

- `MPI_Type_create_resized`
  Adjust strides
  :
  :