Parallel Programming

pauldj@aices.rwth-aachen.de

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

- What are we doing? $\mathtt{b}^{(\mathtt{j})} := \mathtt{a}^{(\mathtt{i})}$
- Wildcards: `MPI_ANY_SOURCE`, `MPI_ANY_TAG`
- . . . but then, “from whom did I receive?”,
  and most importantly, “what is the size of the message?”
  $\rightarrow$ `MPI_Status` (or `MPI_STATUS_IGNORE`)
- `status.MPI_SOURCE`

  `status.MPI_TAG`

  `MPI_Get_count(&status, MPI_INT, &size)`

- 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!

Persistent communication

Optimization

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

`MPI_Send_init, MPI_Recv_init`

binds all the arguments 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

- What if . . .

  non contiguous data and/or **non elementary datatypes**?

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

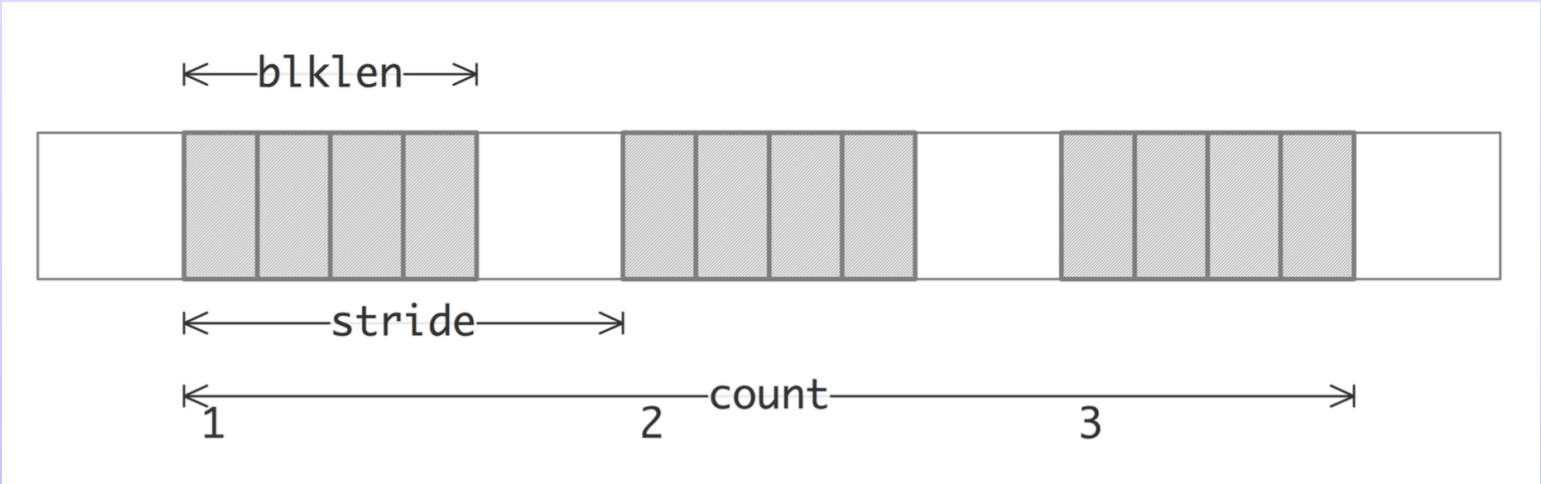
- MPI **derived datatypes**: "Create, commit, use, free"

```
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 )`

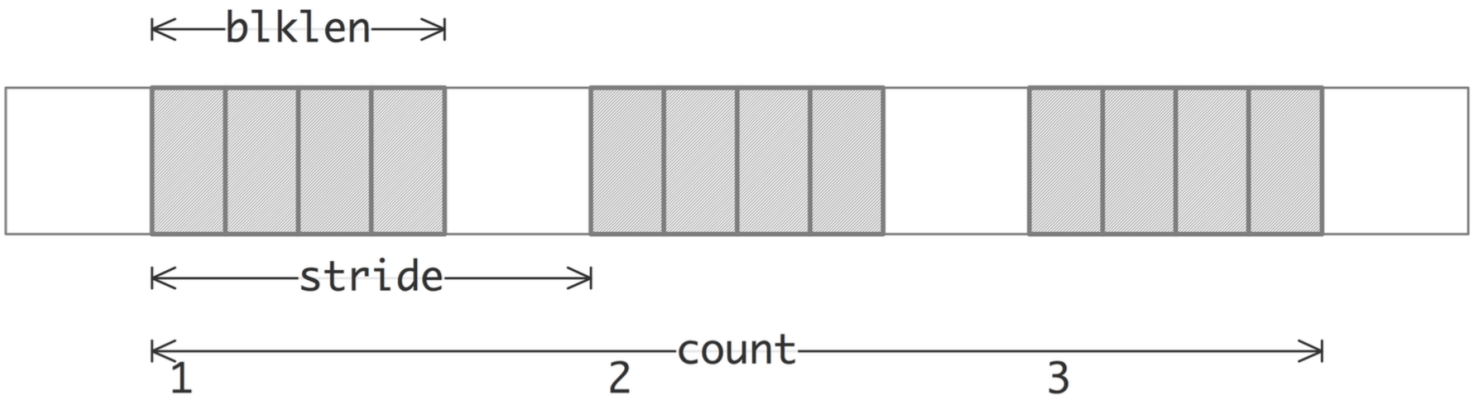
Same as sending `count` entries of `old_type`

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

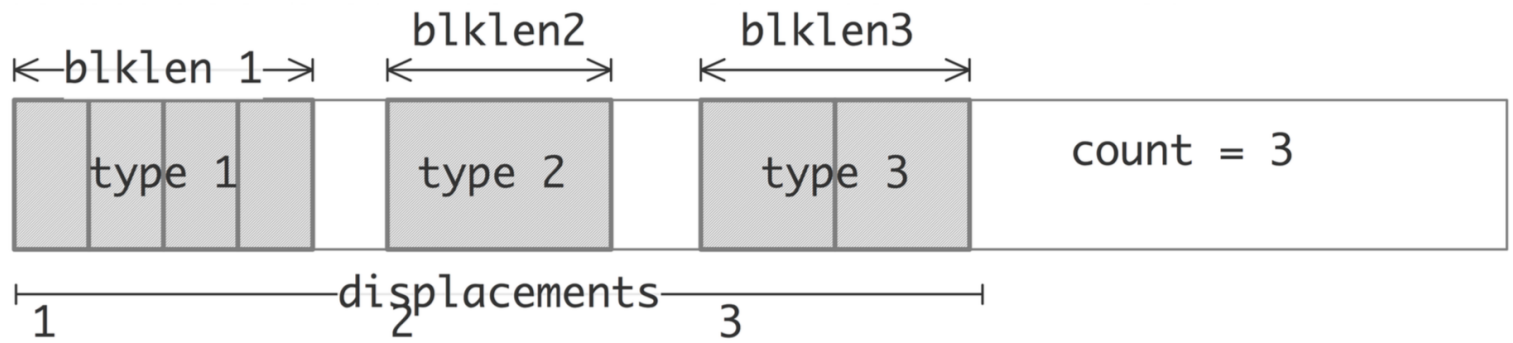


It selects a regular pattern of entries from one array.

- 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
  ⋮