Parallel Programming

Prof. **Paolo Bientinesi**

pauldj@aices.rwth-aachen.de

WS 17/18

Scenario

Process P_i owns matrix A_i, with $i = 0, \ldots, p-1$.

Objective

$$\begin{cases} \text{Even}(i): & \text{compute } T_i := A_i + A_{(i+1) \bmod p} \\ \text{Odd}(i): & \text{compute } T_i := A_i - A_{(i-1+p) \bmod p} \end{cases}$$

Scenario

1D domain, logically split among p processes.

Objective

Run a finite difference scheme, e.g.,

$$u(x_i) := \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1})}{h^2}$$

$\Rightarrow$ **point-to-point communication**

Anatomy of `MPI_Send` and `MPI_Recv`

```
int MPI_Send(
    *buffer, count, datatype,          ← "data"
    destination, tag, communicator     ← "envelope"
);

int MPI_Recv(
    *buffer, count, datatype,          ← "data"
    source, tag, commmunicator,        ← "envelope"
    *status
);
```

message = data + envelope (+ info)

matching envelopes $\Rightarrow$ data transfer

Note: Meanining of `count` in send $\neq$ in recv

`count` in send = size of data; `count` in receive = size of buffer.

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`

Send Modes

The stress is on the buffer being sent: “When I can I safely overwrite it?”

- `MPI_Ssend`: The program execution is blocked until a matching receive is posted. The buffer is usable as soon as the call completes.
- `MPI_Send`: MPI attempts to copy the outgoing message onto a local (hidden) buffer. If possible, the execution continues and the send buffer is immediately usable, otherwise same as `Ssend`.
- `MPI_Isend`: The execution continues `I`mmediately. The send buffer should not be accessed until the `MPI_request` allows it. To be used in conjunction with `MPI_Wait` or `MPI_Test`*.

*: See also `MPI_Waitany`, `MPI_Waitall`, `MPI_Waitsome`, `MPI_Testany`, `MPI_Testall`, `MPI_Testsome`.

Note: Careful with multithreading!! Thread-safety guaranteed?

Recv Modes

The stress is on the incoming buffer: “When I can I safely access it?”

- `MPI_Recv`: The program execution is blocked until a matching send is posted. The incoming buffer is usable as soon as the call completes.

- `MPI_Irecv`: The execution continues `I`mmediately. The incoming buffer should not be modified until the `MPI_request` allows it. To be used in conjunction with `MPI_Wait` or `MPI_Test`*.

$*$: See also `MPI_Waitany`, `MPI_Waitall`, `MPI_Waitsome`, `MPI_Testany`, `MPI_Testall`, `MPI_Testsome`.

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
    )
```

MPI_Sendrecv

```
int MPI_Sendrecv(
    *sendbuf, sendcount, sendtype,
     dest, sendtag,
    *recvbuf, recvcount, recvtype,
     source, recvtag,
     communicator, status
);
```

- `MPI_Sendrecv`: Executes a blocking send and receive operation. Both send and receive use the same communicator, but possibly different tags. The send buffer and receive buffers must be disjoint, and may have different lengths and datatypes.
- `MPI_Sendrecv_replace`: Execute a blocking send and receive. The same buffer is used both for the send and for the receive, so that the message sent is replaced by the message received.

Send Modes (2)

- `MPI_Bsend`: “Buffered” send. The user must provide a buffer to copy the outgoing message (`MPI_Buffer_attach`).
- `MPI_Ibsend`: Non-blocking version of Bsend.
  The sender should not modify the send buffer.
- `MPI_Rsend`: “Ready” send. The corresponding receive must have been already posted. Otherwise, error.
- `MPI_Irsend`: Non-blocking version of Rsend.
  The sender should not modify the send buffer.
- `MPI_Issend`: Non-blocking synchronous send.
  The sender should not modify the send buffer.

Persistent communication

Optimization

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

- `MPI_Send_init( ..., request )`, `MPI_Recv_init( ..., request )`

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

- `MPI_Start( request )`

  initiates the send (receive)