Parallel Programming

Prof. **Paolo Bientinesi**

pauldj@aices.rwth-aachen.de

WS 16/17

HPAC

RWTH AACHEN UNIVERSITY

Scenario

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

Objective

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

Scenario

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

Objective

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

Scenario

1D domain, logically split among p processes.

Objective

Run a finite difference scheme, e.g.,

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

Scenario

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

Objective

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

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
);
```

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 ⇒ data transfer

Note: Meaning of `count`: send ≠ recv

`count` in send = size of message vs. `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?”

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.

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`.

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`*.

Note: Careful with multithreading!!

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

Recv Modes

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

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.

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 Immediately. 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.