Point-to-point communications

Timing parallel programs

universidade de aveiro

Computação Paralela
Módulo MPI

2021/2022

Rui Costa

Email: americo.costa@ua.pt

Point-to-point communications

Standard send (blocking)

```
MPI_Send(address, count, datatype, destination,
tag, comm)
```

- `(address, count, datatype)` describes `count` occurrences of items of the form `datatype` starting at `address`,
- `destination` is the *rank* of the destination in the group associated with the *communicator* `comm`,
- `tag` is an integer used for message matching, and
- `comm` is the *communicator,* identifies a group of processes and a communication context.

Point-to-point communications
Standard receive (blocking)

```
MPI_Recv(address, maxcount, datatype, source,
tag, comm, status)
```

- `(address, maxcount, datatype)` are the same as in MPI_Send, although it is allowed for less than `maxcount` occurrences to be received,
- `tag` and `comm` are as in MPI_Send, with the addition that a wildcard, matching any `tag`, is allowed.
- The `source` is the *rank* of the source of the message in the group associated with the *communicator* `comm`, or a wildcard matching any source.
- Finally, `status` holds information about the actual message size, `source`, and `tag`, useful when wild cards have been used.

Point-to-point communications
Synchronous send

```
MPI_Ssend(address, count, datatype, destination,
tag, comm)
```

`MPI_Ssend` has the same arguments as `MPI_Send`, but only returns when *receiver* process finishes receiving the message.

The point of the synchronous send operations is avoiding the *sender* process to change the values to be sent before the sending actually occurs.

Point-to-point communications
Buffered send

```
MPI_Bsend(address, count, datatype, destination,
tag, comm)
```

`MPI_Bsend` has similar arguments as `MPI_Send` and `MPI_Ssend`, but uses a buffer to store the message while the *receiver* is not ready.

This way, the *sender* process can proceed without the risk of overwriting the message to be sent.

In buffered sends, it is necessary need to allocate enough memory for the buffer, and attach/detach it with:

```
MPI_Buffer_attach(buffer,count);
MPI_Buffer_detach(buffer,count).
```

Timing MPI programs

Timing of parallel programs is especially relevant, since the goal of parallelization is to reduce execution time.

MPI provides a function for timing programs, and sections of programs:

```
MPI_Wtime()
```

Calling `MPI_Wtime()` returns the number of seconds that have passed since some arbitrary point of time in the past, which does not change during the execution of the process.

Elapsed time can be measured with the difference two calls of `MPI_Wtime()`.

Timing MPI programs

The resolution of the output of `MPI_Wtime` is hardware dependent, and can be found by calling

```
MPI_Wtick()
```

Another function that becomes useful for timing programs is

```
MPI_Barrier(comm)
```

This function is a collective operation that does nor let the calling process to continue until all processes in the communicator `comm` have called `MPI_Barrier`.

Bibliography

Using MPI: portable parallel programming with the message-passing interface, 3rd edition, William Gropp, Ewing Lusk, and Anthony Skjellum, MIT press (2014).