universidade de aveiro

Computação Paralela
Módulo MPI
2022/2023

Rui Costa

Email: americo.costa@ua.pt

Why parallel computing?

universidade de aveiro

- Limitations of single computers, like effectiveness of heat dissipation, speed of light, etc.
- Cost of advanced single processors grows faster than their power.
- Already existing (or cheaply acquired) computational resources may be employed, like in the case of SCANs (SuperComputers At Night), which consist in networked workstations used a parallel computer.

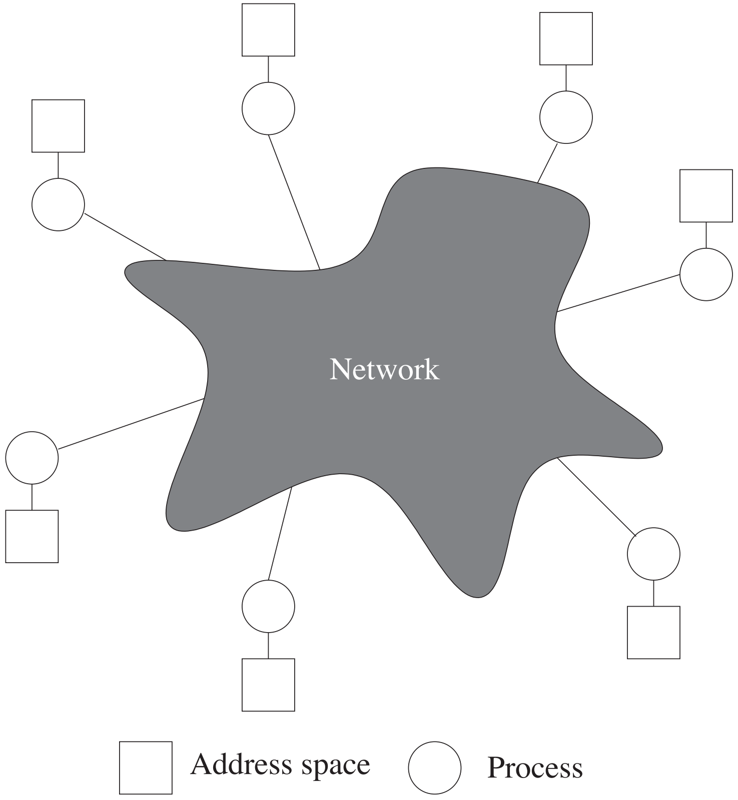
Obstacles

- Inter-communication networks (switches) are still slow compared with intra-processor speeds (although significant advances have been made recently).
- Compilers that automatically parallelize sequential algorithms remain very limited in their capabilities.
- Trade-off between expressivity, portability, and efficiency.

The Message-Passing Model

universidade de aveiro

- Processes have only local memory.
- Processes are able to communicate with each other by sending and receiving messages.
- Communication operations between two process (i.e., transfer data from local memory of one process to local memory of another) must be performed by both process.



Advantages of the Message-Passing Model (particularly MPI)

- Universality
  - Works well with fast and slow communication networks (from parallel supercomputers to workstation networks or dedicated PC clusters).
  - Whenever the hardware supplies shared-memory, the message-passing can use it to speed up data transfer among processes.
  - GPUs can be used with the MPI.

- Expressivity
  - Is a complete model to express parallel algorithms.
  - Provides high control over local data.
  - It is well suited for self-scheduling algorithms, and to deal with imbalances in process speed found in heterogeneous networks.

Advantages of the Message-Passing Model (particularly MPI)

universidade de aveiro

- Ease of debugging
  - Debugging parallel programs remains a challenge.
  - Compartmentalization of memory in MPI makes it easier to find the wrong reads and writes.

- Performance:
  - Memory (and cache) management is key to extract maximum performance from modern CPUs.
  - MPI provides a way for the programmer explicitly associate specific data to processes, which allows both compilers and cache-management hardware to function fully.

What is MPI?

universidade de aveiro

- MPI is a library, not a language. It specifies names, calling sequences and results of functions (which are called from C or Fortran).
- MPI is a standard, or specification, it is not a particular implementation. In this discipline we will use the implementation *mpich*, although a correct MPI program should run on any MPI implementation without changes.
- MPI addresses the message-passing model of parallel computing described above. That is, a collection of processes (with only local memory) communicating using messages.

MPI basic concepts: a minimal interface

universidade de aveiro

- Each communication requires the cooperation of the processes involved; while one process execute a send operation, the other must execute a receive.

- Minimal set of arguments for the `send` and `receive` functions:

  - *sender*: data to be sent (address and length of message), and identity of destination process.

  - *receiver*: address and length of space in local memory to store received data, variable to be filled with identity of *sender* process (so the receiver can know who sent the message).

MPI basic concepts: a minimal interface

- In practice more features may be useful, or even required, by many applications.
- Matching: a process is able to control the messages it receives by using a `tag` (an integer that specifies the 'type' of message). The `tag` is an argument of both the `send` and `receive` functions. In a `receive` operation, it may also be convenient to specify the identity of the *sender*, as an additional screening parameter.
- The `length` of the message received may not be known beforehand. The `receive` specifies a maximum length for the message but allows shorter messages to arrive. So, the actual length of the message is returned in `actlen`.

MPI basic concepts: a minimal interface

universidade de aveiro

```
send(address, length, destination, tag)

receive(address, length, source, tag, actlen)
```

- Problems:
  - The buffer may not be continuous.
  - Different representations of the same information (integer values, floating-point values, etc) in different machines.

- Solution:
  - Message buffer is defined by a triple `(address, count, datatype)`, where `datatype` can be a user defined datatype (or *derived datatype*) that maps noncontiguous memory addresses.

MPI basic concepts: a minimal interface


```
send(address, length, destination, tag)

receive(address, length, source, tag, actlen)
```

- Another problem:
  - *Tags* are integers chosen arbitrary, but must be used in predefined a coherent way throughout the whole program. Complications arise particularly when using libraries written by others whose *tags* may overlap with ours: context is required for correct *tag* interpretation.
- Solution:
  - *Communicators* are objects that combine the notions of context and group of process. Processes belong to groups and, within a group, are identified by *ranks*. The same process may belong to several groups, and within each group is identified with a different *rank*. Each group has its own *communicator*, which is an argument of all communication operations. The destination or `source` arguments of a `send` or `receive` refers to the *rank* of the process in the group identified by the given *communicator*.

MPI basic send and receive operations (blocking)

universidade de aveiro

`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.

MPI basic send and receive operations (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 wildcards have been used.

Some other useful features

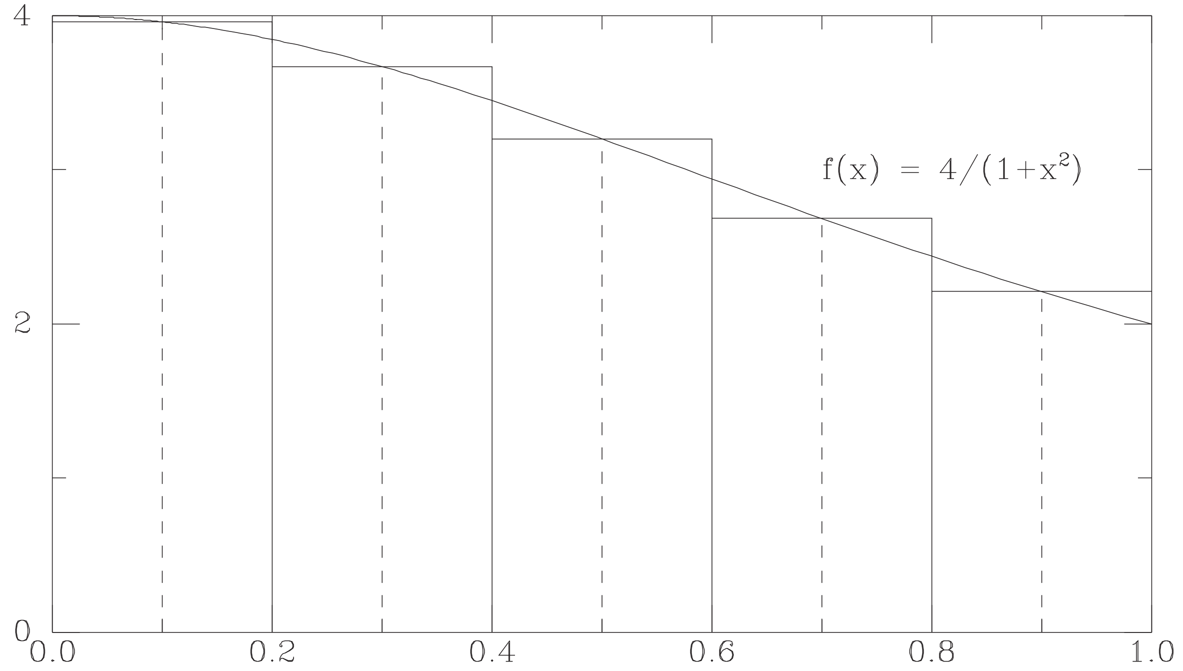
universidade de aveiro

- Collective operations: collective data movement and collective computations;
- Virtual topologies;
- Communication modes: blocking, non-blocking, synchronous, buffered, ready;
- Debugging and profiling;
- Support for libraries;
- Support for heterogeneous networks;
- Processes vs processors.

A simple parallel program - calculation of an integral

universidade de aveiro

- Goal: obtain π from $\int_0^1 {}^4/_{1+x^2}\, dx = \pi$.
- Calculate and sum the area of `n` rectangles, as in the figure.
- Each process is responsible calculating the contribution of a sub-set of rectangles.



A simple parallel program - calculation of an integral

- Keeping things simple, we will use only collective communication operations:

```
MPI_BCAST(n, 1, MPI_INTEGER, 0, MPI_COMM_WORLD)
MPI_REDUCE(mypi, pi, 1, MPI_DOUBLE_PRECISION,
MPI_SUM, 0, MPI_COMM_WORLD)
```

- Additionally, we are need to initialize the MPI 'environment':

```
MPI_Init(&argc,&argv)
```

A simple parallel program - calculation of an integral

universidade de aveiro

- Each process needs to know the total number of processes and its own identification (*rank*) within the group associated with the default communicator `MPI_COMM_WORLD`:

```
MPI_Comm_size(MPI_COMM_WORLD, &numprocs)
MPI_Comm_rank(MPI_COMM_WORLD, &myid)
```

- Finally, at the end of the program every process must terminate the MPI 'environment':

```
MPI_Finalize()
```

Compiling and running MPI programs

universidade de aveiro

- The *mpich* implementation provides an MPI C compiler: `mpicc`. The syntax is similar to `gcc`, `icc`, etc, and it allows to link any desired C libraries, and other standard C compiler options:

```
mpicc -o prog prog.c -mylib
```

- Run `prog` in 4 parallel processes with the command:

```
mpiexec -n 4 ./prog
```

Bibliography

universidade de aveiro

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