From Modelling to Programming Languages

Paolo Baldan

Languages for Concurrency and Distribution

Down to earth

- What is a **process**?
- How do I specify its **behaviour**?
- **Where** are my processes executed?
- Is **communication** local or remote, **synchronous** or **asynchronous**?
- What if a process **fail**?
- …

Concurrency vs. Parallelism

- **Concurrency**
  Several independent cooperating/competing activities

  - Logical (structuring principle)
  - Intertwined with non-determinism

- **Parallelism**
  Different activities executed simultaneously

  - Operational

Parallelism everywhere

- **Task level**: Multimachine/Multiprocessor/Multicore with shared or distributed memory
- **Data level**: Same operation on multiple independent data (e.g., graphics on GPU)
- **Instruction level**: Pipelining, out of order execution (x=1; z=3 --> z=3; x=1)
- **Bit level**: The longer is the word, the more data we elaborate a the same time (8, 16, 32, 64 …)

More than exploiting parallelism

- Concurrency is used for properly exploiting parallelism but it is much more than this …
- Concurrency as a **structuring concept**
  - When things are concurrent, they should be modelled as such!
  - **Simplicity** and **Responsiveness** as nice side effects

More than exploiting parallelism/2

- **Distribution**
  - World is distributed
  - SW must be distributed on different computers in different physical locations …
  - Distribution adds complications but helps in managing failures
- **Resilience**
  - Concurrency and distribution enables resilient, fault tolerant SW (independence of activities and fault detection managed by separate entities)

How can we deal with it?

- **Threads and locks**
  At some level, they will be there, but we prefer to keep them under the carpet

- **Channel-based concurrency and Google Go**
  Emphasis on processes and channels as first class entities

- **Actor model and Erlang**
  ~ concurrent objects, communication via message passing (asynchronous), support for distribution, resilience, fault tolerance

- **The (non so-pure) functional way and Clojure**
  Functional approach to concurrency with pragma (STM)

- **Jolie and the orchestration of existing activities**
  Structuring and interoperability

- **Rust and ownership**
  Explicit managing of memory property