A language-oriented approach to teaching concurrency

Tom Van Cutsem
Stefan Marr
Wolfgang De Meuter
Context

- Organizing new graduate-level course on Concurrency at University of Brussels, Belgium

- Existing course focuses on high-performance computing

- Background:
State of the art

• Pthreads, OpenMP & MPI
  • All are based on a C/C++ substrate

• Low-level models: explicit shared-state or low-level send/receive

• Low-level environments: fighting the compiler

• Primary goal = HPC
Concurrency vs HPC

• It’s not because we have multicore machines that every programmer suddenly needs to be skilled in high-performance computing

• From an article in ACM Queue (Oct. 2008): “Real-World Concurrency” by Bryan Cantrill and Jeff Bonwick:

  • “[T]he proliferation of concurrent hardware has awakened an anxiety that all software must use all available physical resources. Just as no programmer felt a moral obligation to eliminate pipeline stalls on a superscalar microprocessor, no software engineer should feel responsible for using concurrency simply because the hardware supports it.”
Why use a concurrent language?

- Concurrent programming is difficult. Until mastered, don’t make it harder than strictly necessary

  - Use a language designed for the task

  - Changes the “path of least resistance”

    - Language should encourage use of good patterns, inhibit use of bad practices

    - Easier to unlearn/avoid existing habits that conflict with concurrency goals

- Synergy between functional and concurrent programming
Functional programming

• Overloaded term:
  • referential transparency
  • deterministic functions
  • higher-order abstractions
  • parametric type systems
  • lazy computation
  • immutable data
Immutability

• Because side-effects are at the root of most problems in concurrent programming

  • “Side effects prevent concurrency”
    [Joe Armstrong, Programming Erlang]

  • “By emphasizing pure functions that take and return immutable values, [functional programming] makes side effects the exception rather than the norm. This is only going to become more important as we face increasing concurrency in multicore architectures”
    [Rich Hickey, from foreword of Programming Clojure]

• Key message to students: “embrace immutability”
What functional language to choose?

• Loose criteria:
  
  • Promulgate functional programming style (w/ focus on immutable data)
  
  • Sufficiently practical (reliable implementation, sufficient documentation)
  
  • “A language that doesn't affect the way you think about programming, is not worth knowing.” [Alan Perlis, Epigrams in Programming]
Many choices
Many choices
We chose Erlang + Clojure. Why?

• Cultural bias: SICP, dynamic typing, experience with actor-based languages => YMMV

• Choices encompass complementary hardware architectures, concurrency models:
  • Erlang: distributed memory architectures, message passing model
  • Clojure: shared memory architectures, shared state model
Conclusion

- Concurrency control is essential to any software engineer, not just HPC programmers. It’s not { just | all } about performance.

- C + library is not the most efficient medium to teach concurrency

- Get at the core of the problem: side effects => embrace immutability

  - Choose a language that makes this the natural thing to do

- We settled on Erlang + Clojure.

- Feedback / Thoughts?