Concurrency

and the Functional Paradigm

Joey W. Coleman, Stefan Hallerstede

AARHUS UNIVERSITY

DEPARTMENT OF ENGINEERING

13 May 2014

Admin

Concurrency

- What is it?
- Shared State
- Message Passing
- Comparison

Shared State + FP

Message Passing + FP

FP in Industry

- Admin

- Concurrency
  - What is it?
  - Shared State
  - Message Passing
  - Comparison

- Shared State + FP

- Message Passing + FP

- FP in Industry

Admin items

- Today: Concurrency + FP
- Friday: Store Bededag
- Next week: Concurrency + LP, Course Feedback
- Soon: Exam questions

Admin

Concurrency

- What is it?
- Shared State
- Message Passing
- Comparison

Shared State + FP

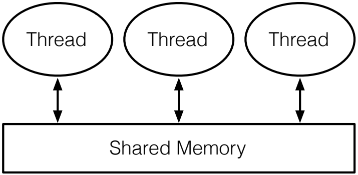
Message Passing + FP

FP in Industry

What is concurrency?

- Multiple things happening “at the same time”
- Not really a standalone paradigm
    - . . . well, not right now
    - . . . and there is the π-calculus
- Simultaneity vs. Interleaving

Shared State

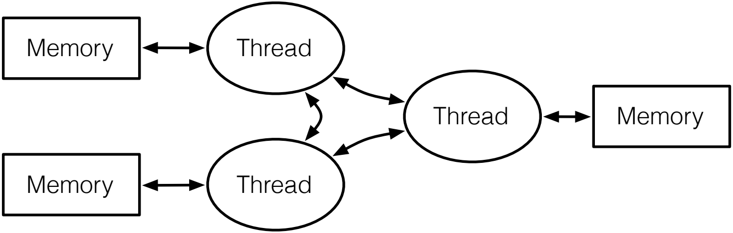


- *Common workbench*
- Inter-thread communication only via state
- Reading/writing
  - Atomicity
  - Write conflicts
  - Synchronization
  - Critical Sections
- Weak memory models
  - Modern CPU architectures

Shared State Primitives/Abstractions

- Compare-And-Swap
- Mutual exclusion (Mutexes)
  - Locks
  - Semaphores
  - Monitors
- Immutable data
- Private data
- Fork/join
- Transactions

Message Passing



- *Network communication*
- Direct inter-thread communication via messages
- Separate, isolated memory stores
- Messaging
  - Priority
  - Addressing
  - Reliability
  - Pass-by-value

Message Passing Primitives/Abstractions

- Send/receive message
- Synchronous/Asynchronous send
- Polling
- Message queues

Combined Shared State + Messaging Systems

- Equivalence of shared state and message passing
    - Lauer, Needham — On the duality of operating system structures
    - In File Sharing/Background Material
- Efficiency vs scalability vs distribution
- Architectural split

Admin

Concurrency

What is it?

Shared State

Message Passing

Comparison

Shared State + FP

Message Passing + FP

FP in Industry

Shared State and the Functional Paradigm

- Shared state is very imperative
- Communication through changes to variable values
- Requires mutability of data
    - i.e. assignment/`set!`
- Kawa exposes the basic Java thread operations to Scheme
    - Useful for Scheme-driven applications
    - Your MP assignment is essentially Java-driven
- Clojure
    - LISP tightly on the JVM
    - Many concurrent features, e.g. Software Transactional Memory

Admin

Concurrency

What is it?

Shared State

Message Passing

Comparison

Shared State + FP

Message Passing + FP

FP in Industry

Message Passing and the Functional Paradigm

- Erlang — `http://erlang.org/`
  - Eponym: Agner Krarup Erlang, Danish engineer
  - **Er**icsson **lang**uage
- Developed at Ericsson in from 1986 by Joe Armstrong
- Initially used in telephone switch development
  - Notably: Ericsson AXD301 switch; nine “9”s reliable (31.5ms/year)
- History paper by Armstrong in File Sharing/Background Material

Factorial in Erlang

```
-module(fact).
-export([fac/1]).

fac(0) -> 1;
fac(N) when N > 0, is_integer(N)
       -> N * fac(N-1).
```

- Definition by pattern matching
- Substitutional/equational
- Note the correspondence to mathematical notation

$$\mathrm{fac}(n) = \begin{cases} 1 & \text{when } n = 0 \\ n \times \mathrm{fac}(n-1) & \text{when } n > 0, n \in \mathbb{Z} \end{cases}$$

Concurrency

- Lightweight processes
- No shared state between processes
- Communication via asynchronous messages
- Values in messages are copied — no references
- Distribution is transparent
  - It's the same to send messages to 'local' and 'remote' processes

Ping/Pong across Processes — Toplevel

```
-module(tut15).
-export([start/0, ping/2, pong/0]).

ping() ...
pong()  ...

start() ->
    Pong_PID = spawn(tut15, pong, []),
    spawn(tut15, ping, [3, Pong_PID]).
```

From the Erlang documentation

Ping/Pong across Processes — pong

```
-module(tut15).
-export([start/0, ping/2, pong/0]).

ping() ...

pong() ->
    receive
        finished -> io:format("Pong finished~n", []);
        {ping, Ping_PID} ->
            io:format("Pong received ping~n", []),
            Ping_PID ! pong,
            pong()
    end.

start() ...
```

From the Erlang documentation

Ping/Pong across Processes — ping

```
-module(tut15).
-export([start/0, ping/2, pong/0]).

ping(0, Pong_PID) ->
    Pong_PID ! finished,
    io:format("ping finished~n", []);
ping(N, Pong_PID) ->
    Pong_PID ! {ping, self()},
    receive
        pong -> io:format("Ping received pong~n", [])
    end,
    ping(N - 1, Pong_PID).

pong() ...

start() ...
```

From the Erlang documentation

What's going on there?

- Process creation — `spawn`
- Message send — e.g. `Pong_PID ! finished`
- Prioritized message receipt — `receive`

- Note that functions can be sent in messages
  - . . . what does this imply?

Admin

Concurrency
- What is it?
- Shared State
- Message Passing
- Comparison

Shared State + FP

Message Passing + FP

FP in Industry

So, who's using this stuff, anyway?

- Apache CouchDB/Amazon SimpleDB
  - Erlang
  - Distribution and scalability
  - Shared-nothing approach supports this
- Jane Street
  - OCaml
  - Low-latency trading
  - Strong type system, high performance
- Galois, Inc
  - Haskell, Cryptol
  - High assurance systems
  - Strong type system, reasoning power
- Bluespec
  - Haskell + a term rewriting system
  - Tools for ASIC/FPGA design
  - Correctness