Concurrency

Unleash your processor(s)

Václav Pech
About me

Passionate programmer
Concurrency enthusiast

GPars @ Codehaus lead
Groovy contributor
Technology evangelist @ JetBrains

http://www.jroller.com/vaclav
http://twitter.com/vaclav_pech
We’re all in the parallel computing business!
# of cores
# of cores
Dealing with threads sucks!

```java
public class Counter {
    private static long count = 0;

    public Counter() {
        count++;
    }
}
```
Dealing with threads sucks!

```java
public class Counter {
    private static long count = 0;

    public Counter() {
        synchronized (this) {
            count++;
        }
    }
}
```
Dealing with threads sucks!

```java
public class Counter {
    private static long count = 0;

    public Counter() {
        synchronized (this.getClass()) {
            count++;
        }
    }
}
```
Dealing with threads sucks!

```java
public class ClickCounter implements ActionListener {
    public ClickCounter(JButton button) {
        button.addActionListener(this);
    }

    public void actionPerformed(final ActionEvent e) {
        ...
    }
}
```
Stone age of parallel SW

Dead-locks
Live-locks
Race conditions
Starvation

Shared Mutable State
Multithreaded programs today work mostly by accident!
Can we do better?

- Map/Reduce
- Fork/Join
- Agent
- Dataflow
- STM
- Actors
Asynchronous invocation

```java
Future f = threadPool.submit(calculation);
...
System.out.println("Result: "+ f.get());
```
Thread Pool

Contention!
Fork/Join Thread Pool
Fork/Join Thread Pool

Work stealing
Fork/Join

- Solve hierarchical problems
  - Divide and conquer
    - Merge sort, Quick sort
    - Tree traversal
    - File scan / search
    - …
Fork/Join (GPars)

```java
runForkJoin(new File("./src")) {currentDir ->
    long count = 0;
    currentDir.eachFile {
        if (it.isDirectory()) {
            forkOffChild it
        } else {
            count++
        }
    }
    return count + childrenResults.sum(0)
}
```

Waits for children without blocking the thread!
Collections (Groovy/GPars)

images.eachParallel {it.process()}

documents.sumParallel()  

candidates.maxParallel {it.salary}.marry()
Parallel Arrays (jsr-166y)

ParallelArray namesOfWomen = people.withFilter(aWoman).withMapping(retrieveName).all();

Ops.Predicate aWoman = new Ops.Predicate() {
    public boolean op(Person friend) {return !friend.isMale();}
};

Ops.Op retrieveName = new Ops.Op() {
    public Object op(Person friend) {return friend.getName();}
};
Dataflow Concurrency

- No race-conditions
- No live-locks
- Deterministic deadlocks

Completely deterministic programs

BEAUTIFUL code

(Jonas Bonér)
Milestone

Asynchronous calculations
Fork/Join
Parallel collection processing
Dataflow variables/streams
Message Passing

Autors

Processes with mailboxes

Communicating Sequential Processes (CSP)

Sequential processes synchronously talking through channels

Dataflow Operators

Event-triggered computations connected by channels into a graph
Actors

- Isolated
- Communicating
  - Immutable messages
- Active
  - Pooled shared threads
- Activities
  - Create a new actor
  - Send a message
  - Receive a message
Actors use

HTTP

Form

Process

Fraud Detect

Gate Keeper

Finger Prints

Address Check

Email Check

SOAP

SMTP

Response
Actors patterns

- Enricher
- Router
- Translator
- Endpoint
- Splitter
- Aggregator
- Filter
- Resequencer
- Checker
Sending messages

buddy.send 10.eur

buddy << new Book(title:’Groovy Recipes’,
    author:’Scott Davis’)

def canChat = buddy.sendAndWait ‘Got time?’

buddy.sendAndContinue ‘Need money!’, {cash->
    pocket.add cash
}
Stateless Actors (pure Java)

class MyActor extends DynamicDispatchActor {
    private Account account = ...
    public void onMessage(String msg) {
        String encrypted = encrypt(msg);
        reply(encrypted);
    }
    public void onMessage(Integer number) {
        reply(2 * number);
    }

    public void onMessage(Money cash) {
        System.out.println("Received a donation " + cash);
        account.deposit(cash);
    }
}

Stateful Actors

class MyActor extends DefaultActor {
    void act() {
        def buddy = new YourActor()
        buddy << 'Hi man, how\'re things?'
        def response = receive()
    }
}
Implicit State in Actors

val me = actor {

  react { msg1 ->
    switch (msg1) {
      case Work: reply "I don't work so early" ; stop();
      case Breakfast: 
        msg1.eat()
        react { msg2 ->
          switch (msg2) {
            case Work: reply "OK, time to work"; msg2.do()
            case Lunch: ...
          }
        }
    }
  }
}
Continuation Style

loop {
    ...
    react {
        ...
        react {/* schedule the block and exit */
            ...
        }
        //Never reached
    }
    //Never reached
}
Java actor frameworks

Jetlang
Kilim
ActorFoundry
Actorom
Akka
GPars (Yes!)

…
Shared Mutable State

Misused most of the time

When really needed, use

- Agents
- Software Transactional Memory
- Locks
No more threads and locks

```kotlin
images.eachParallel {
    //concurrency agnostic code here
}

def myActor = actor {
    //concurrency agnostic code here
}

atomic { /*concurrency agnostic code here*/ }
...
Parallelism is not hard, multi-threading is
Questions?

Find more at:

http://gpars.codehaus.org
http://www.jroller.com/vaclav
http://twitter.com/vaclav_pech