alphacsp JAVA EDGE 09 THE FUTURE OF JAVA

The Actor Model

Towards Better Concurrency

By: Dror Bereznitsky

Warning: Code Examples

alphacsp JAVA EDGE 09 THE FUTURE OF JAVA

Agenda

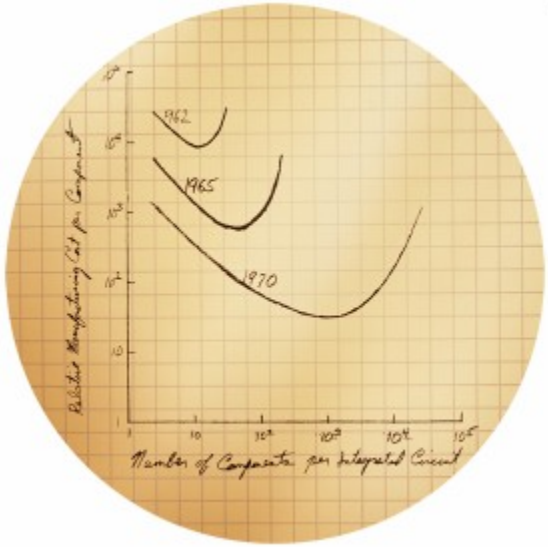
- The end of Moore law?
- Shared state concurrency
- Message passing concurrency
- Actors on the JVM
- More concurrency models
- Summary

The End of Moore Law ?

The End of Moore Law ?

Moore's law

- The number of transistors on a chip will double approximately every 18 months
- Gordon E. Moore, 1965



The End of Moore Law ?

Andy Giveth, and Bill Taketh away

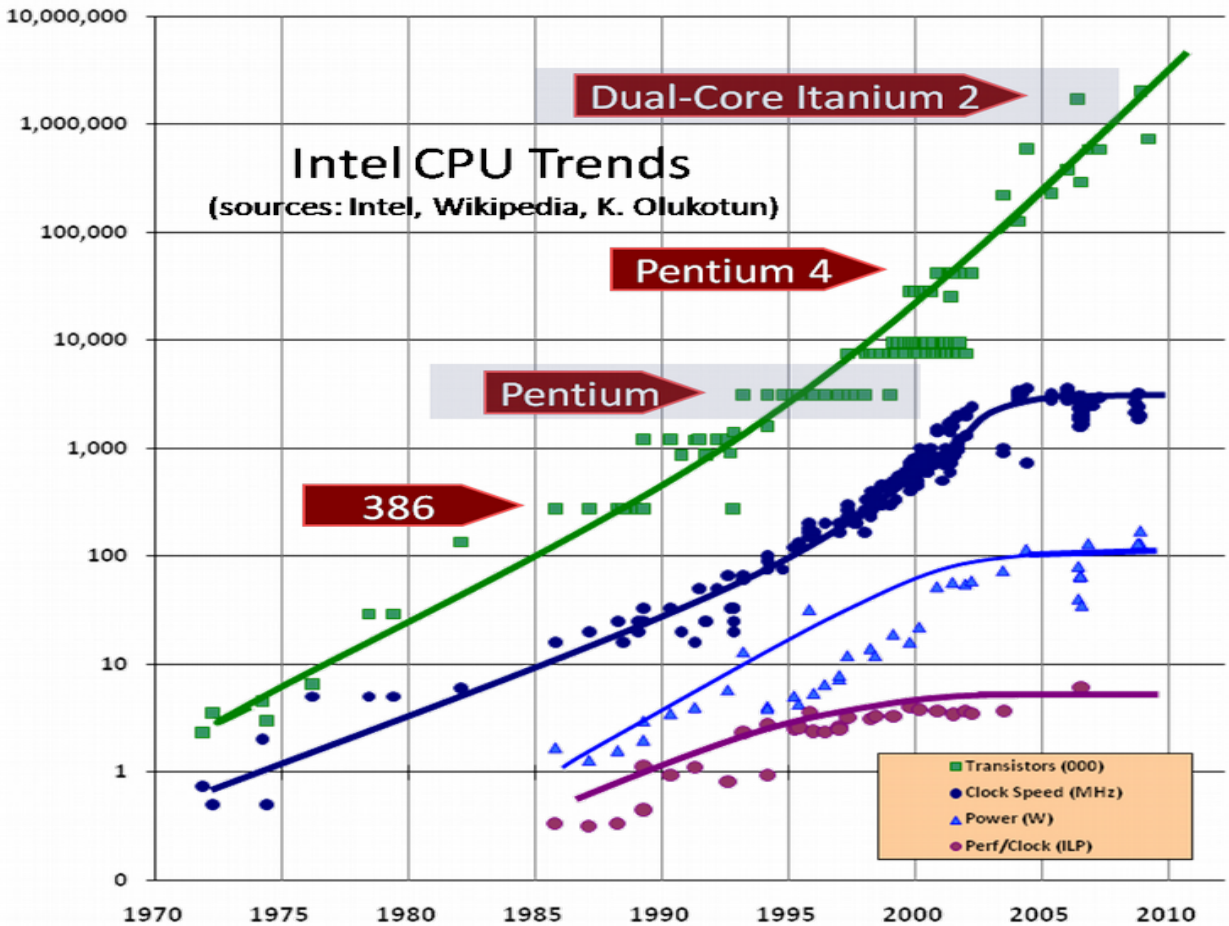
» No matter how fast processors get,
software finds new ways to eat up the extra speed

The End of Moore Law ?

The Free Lunch

- Free and regular performance gains, even without releasing new versions or doing anything special

The End of Moore Law ?



Why You Don't Have 10GHz Today

- Hardware crossed a boundary in the early 2000s:
  - chips got big enough, cycle speed got fast enough
  - a signal can no longer reach the whole chip in a clock cycle
  - problems with heat dissipation

9

The End of Moore Law ?

The Multicore Era

- Processor manufacturers have turned towards multi-core processors
- Capable of doing multiple calculations in parallel
- CPU speeds are likely to stay relatively flat in the near future

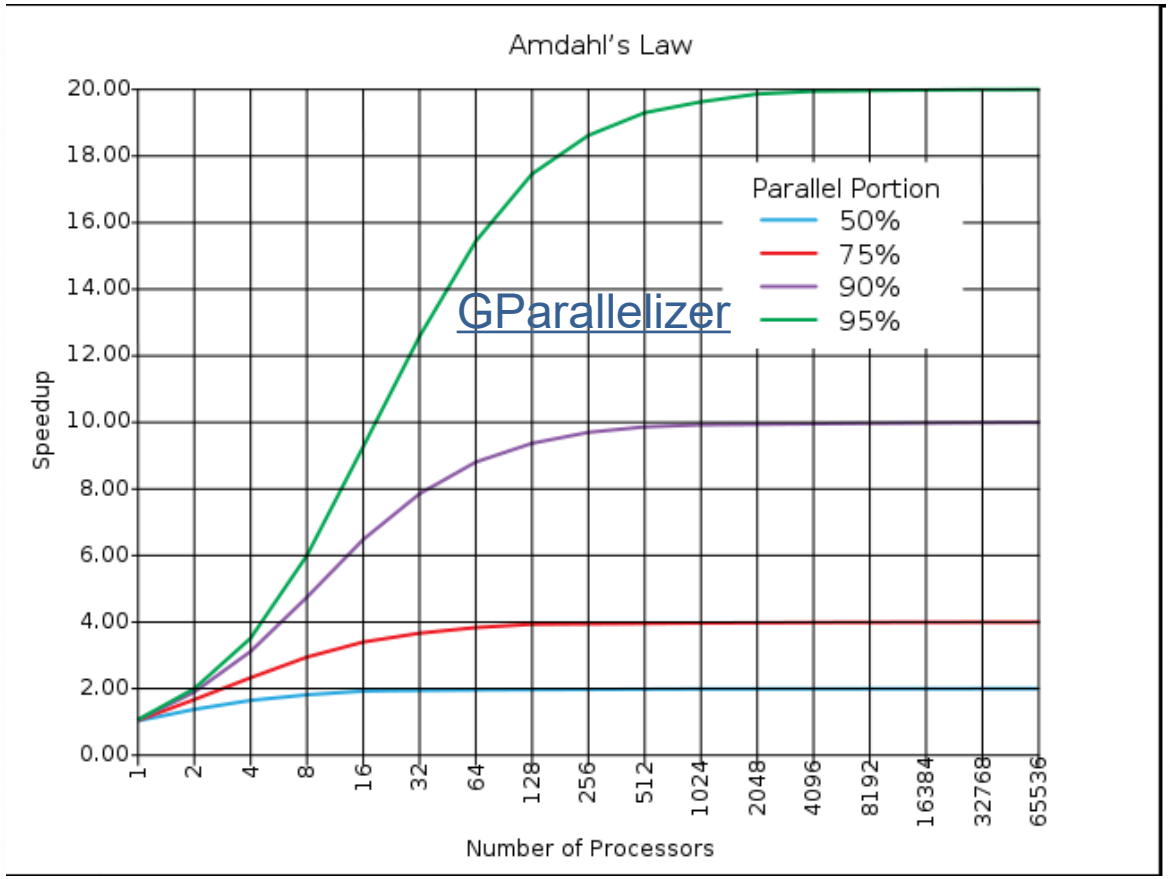
The End of Moore Law ?

The Concurrency Revolution

- The performance lunch isn't free any more
- Want to benefit from the continued throughput advances in new processors?
- You will need to develop well-written ***concurrent*** applications

The End of Moore Law ?

Amdahl's Law



12

Shared State Concurrency

Shared State Concurrency

- Shared mutable state
- Locking mechanism

Threads

- Threads concurrently execute code sections
  - Contains resources that must be shared
  - Synchronized in order to guarantee
    - Correct ordering
    - Visibility
    - Data consistency

The Popular Choice

++C/C

Why Threads are Evil

/http://www.flickr.com/photos/amagill/235453953

Not Convinced Yet ?

"Non-trivial multi-threaded programs are *incomprehensible to human …*"

Edward A. Lee, The Problem with Threads

Alternative Concurrency Models

- **Message Passing Concurrency (Actors)**
- Software Transactional Memory
- Dataflow Concurrency

Message Passing Concurrency

Message Passing concurrency

- 1973, paper by Carl Hewitt
- Avoid the problems caused by threading and locking
- Implemented in **Erlang**, Oz, Occam

Message Passing Concurrency

Key Principals

- Actors instead of objects
- No shared state between actors
- Asynchronous message-passing
- Mailboxes to buffer incoming messages

Message Passing Concurrency

Actors

- React to received messages by executing a behavior function
  - can only change the state of the actor itself
  - can send messages to other actors
- Actors never share state and thus never need to compete for locks for access to shared data

Message Passing Concurrency

Messages

- Actors exchange data by sending **immutable** messages
- Messages are sent asynchronously
- Actors do not block waiting for responses to their messages

Message Passing Concurrency

Mailbox

- Messages buffered in an actor's mailbox
- A mailbox is a queue with multiple producers and a single consumer
- Also known a channel

Message Passing Concurrency

Erlang

- A pure functional, dynamically typed language invented in 1986 at Ericsson
- Designed for concurrency, distribution and scalability
- Actors are part of the language
- No Mutable state

Hello World in Erlang

```
loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop();
      goodbye ->
         ok
   end.
```

Actors on the JVM

alphacsp JAVA EDGE 09 THE FUTURE OF JAVA

Actors on the JVM

- JVM languages actors implementations available for
  - Java
  - Scala
  - Groovy
  - Fantom

Java Actor Libraries

- **Actor's Guild**
- Akka
- ActorFoundry
- Actorom
- Functional Java
- Kilim
- Jetlang

Actors Guild

- Experimental Java framework
  - make concurrent programming easier
- Annotation based approach
- Messages handlers are implemented as Java methods
- Uses a request/response pattern for messages
  - Not a pure Actors implementation
- No special pre-processing

Actors Guild Hello World: Actor Implementation

```
public abstract class HelloActor extends Actor {
    @Prop
    public abstract String getName();

    @Message
    public AsyncResult<String> sayHello(String name) {
        return result(
         String.format("Hello %s, my name is %s",
            name, getName()));));
    }
}
```

Actors Guild Hello World: Actor Usage

```
Agent agent = new DefaultAgent();


HelloActor hello = agent.create(HelloActor.class,
   new Props("name", "Hamlet"));
AsyncResult<String> asyncResult = hello.sayHello("Claudius");
System.out.println(asyncResult.get());


agent.shutdown();
```

Thread Usage Pattern

```
@Message
@Usage(ThreadUsage.IO)
public AsyncResult<Void> writeFile(...) throws Exception {
   FileOutputStream fos = new FileOutputStream(name);
   fos.write(content.getBytes());
   fos.close();
   return noResult();
}

@Message
@Usage(ThreadUsage.Waiting)
public AsyncResult<Void> waitForKeyPrompt() throws Exception {
   System.in.read();
   return noResult();
}
```

Concurrency Model Annotations

```
@Model(ConcurrencyModel.Stateless)
class MultiplicatorActor extends Actor {
    @Message
    public AsyncResult<Integer> mul(int a, int b) {
        return result(a * b);
    }
}
```

Scala Actors

Scala actors combine the powers of functional programming along with the flexible type-system of Scala

Share Nothing, Asynch Message Passing

- Encourages shared-nothing process abstractions
  - mutable state - private
  - shared state - immutable
- Asynchronous message passing
- Pattern matching
- Fork/Join as the underlying implementation

Thread-based vs. Event-based Actors

- *Thread-based actors* - each run in its own JVM thread
  - Overhead in case of large amount of actors
  - full stack frame suspension (*receive*)
- *Event-based actors* run on the same thread
  - *Use a react* block instead of a *receive* block

Lightweight Event Based Actors

- Lightweight event objects
- Executed on an underlying worker thread pool
  - automatically resized when all threads block on long running operations
- Suspension mechanism based on a continuation closure (react)

The Actor and Messages

```
case class Name(name: String)
case class Greet();

object HelloWorld extends Actor {
  def act = {
    var myName = "Incognito"
    loop {
      react {
        case Name(name) => myName = name
        case Greet =>
            reply("Hello, my name is " + myName); exit
      }
    }
  }
}
```

Interacting with the Actor

```
object Greetings extends Application {
  HelloWorld.start

  HelloWorld ! Name("foo")

  HelloWorld !? Greet match {
    case result: String => println(result);
  }
}
```

Groovy Actors

GPars

- GPars = Groovy Parallel Systems
  - Formerly known as GParallelizer
- Handle tasks concurrently, asynchronously, and distributed
  - Concurrent collection processing
  - **Actor programming model**
  - Dataflow concurrency constructs
  - Safe - an mt-safe reference to mutable state

Gpars Actors

- Inspired by the Actors library in Scala
- Event driven actors
  - concurrent actors that share a single pooled thread
  - Using fork/join under the hood
- Supports distributed actors

GPars Actors Hello World: Actor Implementation

```
class HelloWorld extends AbstractPooledActor {
  String name

  void act() {
    loop {
      react {
        switch (it) {
          case 'hello': println "Hello, my name is $name";
                 break
        }
      }
    }
  }
}
```

GPars Actors Hello World: Actor Usage

```
def actor = new HelloWorld(name : "Charlie").start();
actor.send('hello')
```

Actors in the Real World

Erlang Actors in the Real World

- Ericson AXD 301 switch
  - millions of calls per ,99.9999999 percent uptime
- Facebook chat application
  - 70 million concurrent users
- RabbitMQ
  - high-performance AMQP, 400,000 messages per second.
- Apache CouchDB
  - distributed, fault-tolerant document-oriented database
- Ejabberd XMPP server – jabber.org

Actor Benefits

- Easier to reason about
- Higher abstraction level
- Easier to avoid
  - Race conditions
  - Deadlocks
  - Starvation
  - Live locks
- Distributed computing

Problems with Actors

- Actors don't work well when
  - Shared state is needed F.e. bank account
  - Need to achieve global consensus
  - Synchronous behavior is required
- It is not always trivial to break the problem into smaller problems

More Concurrency Models

Software Transactional Memory

- 1995 Nir Shavit and Dan Touitou
- Memory as a transactional data set
- Alternative to lock based synchronization
- Similar to database transactions
- Optimistic Approach
  - a thread completes modifications to shared memory without regard for what other threads might be doing

More Concurrency Models

Deuce

- Java STM framework by Guy Korland (TAU, GigaSpaces)
  - @Atomic methods
  - Field based access
    - More scalable than Object bases.
    - More efficient than word based.
  - No reserved words
    - No need for new compilers (Existing IDEs can be used)
  - Using bytecode instrumentation (ASM)

Duece Example

```
public class Bank{
   private double commission = 0;

   @Atomic(retries = 64)
   public void transaction( Account ac1, Account ac2, double
       amount){
       ac1.balance -= (amount + commission);
       ac2.balance += amount;
   }

   @Atomic
   public void update( double value){
      commission += value;
   }
}
```

Deuce Example Under the Hood

```
public void update( double value){
    Context context = ContextDelegetor.getContext();
    for( int i = retries ; i > 0 ; --i){
        context.init();
        try{
            update( value, context);
            if(context.commit()) return;
        }catch ( TransactionException e ){
            context.rollback();
            continue;
        }catch ( Throwable t ){
            if( context.commit()) throw t;
        }
    }
    throw new TransactionException();
```

STM Pros and Cons

- Pros
  - Performance gains on multi processor machines
  - Simple approach to concurrency
  - No deadlocks or livelocks
- Cons
  - Transactions cannot perform any operation that cannot be undone

Summary

Summary

- The free lunch is over
- Applications must be built for concurrency
- Shared state concurrency is hard
- Alternative concurrency models
  - Message passing concurrency
  - Shared Transactional Memory

Summary

Resources

- The free lunch is over
- State: You're Doing It Wrong
- Edward A. Lee, The Problem with Threads (PDF)
- Alex Miller – Actor Concurrency
- Deuce STM

alphacsp JAVA EDGE 09 THE FUTURE OF JAVA

Thank You :-)

QUALITEST software testing experts | MalamTeam | MalamTeam TRAINING | ATLASSIAN | Sun microsystems | JFrog