UpScale Project (FP7)

- CWI, Imperial College London, University of Oslo and Uppsala University

- Study how an OO PL can support the development of applications that seamlessly scale to the available parallelism of manycore chips.

- Take an actor-based concurrency model as a starting point, rather than multi-threading, and use types to "open up the actors" for further parallelism, keeping the amount of concurrency implicit in the language.

- Capability types, Ownership types, Session Types, etc.
An overview of
UpScale @ Imperial College London

Nobuko Yoshida
Sophia Drossopoulou
Juliana Franco
STOPS: Session Types + Ownership + Sizes

• A language that combines binary session types, ownership types and size annotations to calculate communication costs for OO distributed programs.
• The goal is to optimise performance of distributed programs by:
  ∗ changing the topology of the participants,
  ∗ change the order of messages sent and received,
  ∗ avoid the serialisation of unnecessary data,
  ∗ etc

Paper:
J. Franco, S. Drossopoulou and N. Yoshida Calculating communication costs with Sessions Types and Sizes, ICCSW’14.
B-COOL!: Behaviour for in Location-aware OO Code

- Behavioural and Ownership Types for Non Uniform Memory Access systems
- A small OO language that combines behavioural types with ownership types.
- Ownership types represent the topology.
- Behavioural types describe reads, writes and messages sent to remote locations.

Paper:
ORCA: Ownership and Reference Count collector for Actors

- GC protocol for actor-based object oriented languages.
- Based on Ownership and Reference Counting
- It relies on the type system and on message passing
- No Stop the World Steps: the application does not need to stop in order to do object deallocation
- No Synchronization mechanisms required for Heap Mutation

Papers:
- Sylvan Clebsch, Sebastian Blessing, Juliana Franco and Sophia Drossopoulou. Ownership and Reference Counting based Garbage Collection in the Actor World. ICOOOLPS’15
- Another paper in preparation.

Collaboration with:
An Overview of

UPSCALE @ UPPSALA

Stephan Brandauer, Dave Clarke, Elias Castegren, Kiko Fernandez-Reyez, Phuc Vo, Tobias Wrigstad, Albert Yang
The Encore Language

A scalable programming language

Actors as the fundamental unit of concurrency

Asynchronous messages returning futures

Support for millions of concurrent actors that block or await future results

Cooperative scheduling at the implementation level (also at surface level using suspend/await)

Runs on-top of the PonyRT extended with futures and lightweight threading

Extends a subset of the ABS family of languages but not intended as a modelling language
<table>
<thead>
<tr>
<th><strong>Parallel Combinators</strong></th>
<th><strong>Reference Capabilities</strong></th>
<th><strong>Hot Objects</strong></th>
<th><strong>Disjointness Domains</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors are <strong>sequential</strong>; need to abstractly specify parallel computations for <strong>scheduler flexibility</strong></td>
<td>Avoiding <strong>data-races/copying/non-determinism</strong> both at application level and in the run-time</td>
<td>Popular actors risk becoming bottlenecks—lowering <strong>throughput</strong>; increasing <strong>latency</strong></td>
<td>Sharing semantics of object-oriented programs <strong>misaligned</strong> with language defaults</td>
</tr>
</tbody>
</table>

**Problem**

**Strongly typed mini-DSL for orchestrating parallel computation**

**A hierarchy of annotations** that express how programmers **share** objects **across computations**

**New class of actors whose message processing semantics is controlled from deployment spec.**

**Invert defaults** — track sharing instead of alias-freedom

**Approach**

**Reify** pipeline; **operations** on pipeline for **forking, pruning, speculating**, etc.

**Capabilities are tracked in types** which are formed from **sharing modes** and **traits**

**Several possible implementations: lock-based synchronous actors, STM, lock-free capabilities**

**Invert defaults** — track sharing instead of alias-freedom

**Details**

Integration with scheduling of actors; killing on-going computation

Balancing expressivity and syntactic overhead; capturing the relevant properties; polymorphism

Actor semantics sequential, difficult engineering to get good performance

Balancing expressivity and syntactic overhead; measuring how aliasing is used in practise

**Difficulties**

Submitted to **COORDINATION’16 + UU Master Thesis**

**IWACO’14, submitted to ECOOP’16, submission in prep for OOPSLA’16**

**In prep. Implementation on-going**

**Deliverables**

*Implemented in Encore*

*Partial impl. in Encore*

*Published OOPSLA’15, submission in prep for OOPSLA’16*
Encore Compiler

- Haskell front-end and C back-end
- **Currently:** Source-to-source translation from Encore to (readable) C
- **Future:** replace C back-end with LLVM back-end

- LLVM back-end will enable ”deep optimisations”, e.g.,
  - alias analysis
  - allocation analysis
An Overview of UPSCALE @ OSLO

Shiji Bijo, Einar Broch Johnsen, Violet Ka I Pun, S. Lizeth Tapia Tarifa
Model-based exploration of deployment concerns for multicore architectures

Guide decisions about scheduling, load-balancing and locality using abstract models of architectures
Achievements

Problem: Understand how data moves between caches and memory
- **Approach:** Formal model of multicore architectures with cores, caches and memory
- **Details:** SOS with labels for parallel communication between cores
- **Difficulties:** model the parallel communication, cache coherency
- **Status:** Accepted paper at SAC 2016, MUSEPAT track

Problem: Enable exploration in the abstract model
- **Approach:** Enable measurements and configurable parameters in the model
- **Details:** Executable model for simulations and analysis implemented in Maude
- **Difficulties:** Transform declarative aspects of the SOS to make the model executable
- **Status:** Accepted paper at WRLA2016

Problem: Establish a formal basis for the development of Encore
- **Approach:** SOS of a fragment of Encore via a calculus called \( \mu_{\text{Encore}} \)
- **Details:** concurrency model of Encore with strong notion of locality
- **Difficulties:** SOS with sheep cloning and closures
- **Status:** Collaboration with Uppsala University. Book chapter at SFM 2015
**Problem:** Extend the model with multilevel caches in an abstract way

- **Approach:** Define an abstract notion of penalty to capture the distance to the data
- **Details:** Categorize tasks in classes reflecting average/worst-case data access
- **Difficulties:** Extract the worst-case abstract description from real programs
- **Status:** Ideas still in early phase

**Problem:** Scheduling decisions depending on data layout - model

- **Approach:** Make the scheduler in the model aware of where data is located
- **Details:** Annotations determine data layout, infer logical locations for data accesses
- **Difficulties:** Use behavioural descriptions to infer better scheduling
- **Status:** Collaboration with Imperial College London. Work in progress.
An Overview of UPSCALE @ CWI

Keyvan Azadbakht, Frank de Boer
Overview

- Transferring parallelism from inter-object to intra-object level
  - Notion of Multi-threaded Actors
- Extending the notion of Cooperative Scheduling
  - Efficient await on boolean conditions by means of Promises in Haskell
• Await on boolean conditions in a wait-notify manner by means of promises
  ▪ Extending ABS2Haskell to support the await

• The Preferential Attachment case study
  ▪ Well-known problem in the context of social networks
  ▪ The solution:
    ▪ No need for explicit complex low-level synchronization mechanisms
    ▪ Implemented in ABS and compiled in ABS2Haskell
Multi-threaded Actors

- **Active objects share a queue of messages**
  - A notion complementary to COG in ABS where the objects share the control
  - The Actor contains active objects and has an identity and the shared queue
  - The temporal order of the events with the same values of synchronized arguments is guaranteed on activation.

- **Operational semantics**
- **Type system**
- **Runtime system in Java 8**
HAPPY BIRTHDAY FRANK