PL efforts in UPMARC

an excerpt

Tobias Wrigstad

assistant professor

Uppsala Programming for Multicore Architectures Research Center

onsdag den 11 maj 2011
Short Bio

’06  PhD @ Royal Inst. of Technology, Sweden
’07  Postdoc @ Purdue University, IN US
’09  Associate prof @ Stockholm University
’10  Assistant prof @ Uppsala University

Aliasing in OOPLs
(Pluggable) type systems
Concurrent and parallel programming
Dynamic programming languages
Spaghetti Code
(~1968)
Spaghetti Data
(Present day)

Thread A

Thread B

Races etc.

Visualisation of a heap from an OO program

Object

Reference

onsdag den 11 maj 2011
Capsule Summary

Spaghetti data + ubiquitous parallelism is a no-go
— However, sharing state is key to performance!

Our goal

• Bring order to object-oriented data

Our approach

• Enable programmers to express sharing, locality, etc.

• Use this information for checking and parallelisation

  (correctness) (efficency)

• Avoid problems (races, compositionality, etc.) by design
Thread-Local Heaps in Loci [ECOOP 2009]

- Shared accesses identified
- Fine grained
- Pluggable
- Low syntactic overhead
- Shared heap still a mess
- Locks are not compositional

http://loci.java.net/
Isolated Active Objects in Joelle [TOOLS 2008]

Active Object 1

Active Object 2

Active Object n

+ Compositional
+ No races
+ Low syntactic overhead
– Coarse grained
– No internal parallelism

Immutable (no races)

Transferrable (no races)

(no access)

onsdag den 11 maj 2011
The Way Forward: Our Core Design

Loci + Joelle = Joelle 2.0
Ultimate Goals

• Replace Java as the safe mainstream programming language
  Simple & gradual system, legacy, do not ignore the programming craft...

• Deal with parallelism and concurrency better than Erlang
  Efficiency, locality, migration, high-level errors only...

• Without compromising with object-orientation
  Support shared mutable state as effortlessly as possible, encapsulation...
Current Goals

- Avoid races, deadlocks, subtle memory model considerations
- Better utilise shared resources (caches, shared buffers, etc.)
- Implicit parallelism where possible
- Facilitate manual and automated reasoning
- Language a superset of Java (or C++)

Interested industry: Ericsson, ABB, IBM, Oracle (no real commitment from anyone yet)

(Community: sc IWACO ’07–11; is UPMARC summer school ’10; pc FTfJP ‘11, IWACO ’11, OOPSLA ’11, ECOOP ’12 je LNCS state-of-the-art aliasing in OOP Journal)
The Way Forward: Our Core Design

Ownership types  Effect systems  Alias Analysis  Inference
Fine-grained parallelism inside active objects

Use run-time adaptation techniques for performance

Active object model is safe by design
Ownership types

- Effect systems
- Alias Analysis
- Inference

Static
Dynamic

- Bring structure to active object subheap
- Preserve isolation
- Safe sharing
- Safe transfer
- Minimal clone operations

onsdag den 11 maj 2011
Ownership types  Effect systems  Alias Analysis  Inference
Ownership types  Effect systems  Alias Analysis  Inference

Static
Dynamic

Improve effect analysis result
Compile fast-path and avoid synch to main memory
Simplify coding with shared and unshared values

onsdag den 11 maj 2011
Ownership types | Effect systems | Alias Analysis | Inference

Lighten annotation burden
Facilitate reuse
Facilitate refactoring

onsdag den 11 maj 2011
Immediate UPMARC Synergy

Programs in our system exhibit strong properties that facilitate program analysis
• e.g., alias freedom, locality information, effects of expressions
• Previous work on verification @ UU could capitalise on this

Migrating legacy code to active objects
• Can it be done automatically?
• Inference in isolated enclosures — a smaller problem?
• Maybe annotations can be partially inferred?

Ownership and effect information used for scheduling, resource management

onsdag den 11 maj 2011
Obstacles & Some Open Questions

Need more “warm bodies” (PhD students & PostDocs)

Need representative legacy code (tentative from ABB)

What are the effects on common idioms and programming practises?

Inference vs. programmer annotations—what is a good balance?

• Inference is flexible but brittle
• Annotations are stable but stale

Will a single active object concept fit all circumstances?

How to feedback that we cannot run something in parallel to the programmer?

...
Related Work (Excerpt)

Proactive, Scoop, Akka, etc. — active object systems for Java, Eiffel, Scala; no isolation guarantees, not for parallel programs

DPJ — share some ideas but for threads and task-based parallelism only; extreme programming overhead

X10 — captures where a computation takes place in the “place type”

CoBoxes, JCoBoxes — similar ideas for encapsulation but completely dynamic

FlexoTask and StreamFlex — shares ideas for encapsulation but for stream programming

A wealth of systems for ownership types, linearity, effect systems (Clarke, Boyland, Noble, Vitek, Aldrich, Rinard, Liskov, ...)

Ownership types inference work by Milanova et al., Ma & Foster

Jade — implicit parallelism; Futures; Erlang; ...

Our Unique Footprint:

Flexible yet powerful aliasing constraints; ownership-based effects; 2D rep splitting

Active-object based

Combination of annotations and inference; full static checking

Both coarse-grain (AO) and fine-grain (task, etc.) parallelism

Run-time reliance on ownership and effects for scheduling and implicit parallelism

Consider programming surroundings & legacy
Thank you! Questions?