Type Systems For Distributed Data Structures

Ben Liblit & Alexander Aiken
University of California, Berkeley
Underlying Memory Model

- Multiple machines, each with local memory
- Global memory is union of local memories
- Distinguish two types of pointers:
  - **Local** points to local memory only
  - **Global** points anywhere: \langle machine, address \rangle
  - Different representations & operations
Language Design Options

• Make everything global?
  ✓ Conservatively sound
  ✓ Easy to use
  ✗ Hides program structure
  ✗ Needlessly slow
Language Design Options

• Expose local/global to programmer?
  ✓ Explicit cost model
  ✓ Faster execution
  ✗ Naïve designs are unsound (as we will show)
  ✗ Code becomes difficult to write and maintain
  ✗ Conversion of sequential code is problematic
A (Possibly) Gratuitous Global
A (Potentially) Unsound Local
Understand It First, Then Fix It

• Study local/global in a simpler context
  – Design a *sound* type system for a tiny language

• Move from type checking to type inference
  – Programmers see as much detail as they want

• Apply findings to design of real languages
  – Type system detects & forbids “bad things”
  – Local qualification inference as optimization
Type Grammar

$$\omega ::= \text{local} \mid \text{global}$$

$$\tau ::= \text{int} \mid \text{boxed } \omega \tau \mid \tau \times \tau$$

- Boxed and unboxed values
- Integers, pointers, and pairs
  - Pairs are not assumed boxed
- References to boxes are either local or global
Global Dereferencing: Standard Approach Unsound

\[
x : \text{boxed global } \tau \\
\downarrow x : \tau
\]

, where \( \tau = \text{boxed local int} \)

\[
x = \quad \downarrow x = \quad 5
\]
Global Dereferencing: Sound With Type Expansion

\[
x : \text{boxed global } \tau \\
\downarrow x : \text{expand}(\tau)
\]
Global Dereferencing: Tuple Expansion

- Type expansion for tuple components?
- No: would change representation of tuple

\[ x = \]

\[ \downarrow x = 8 \]

\[ \downarrow x = 5 \]
Global Dereferencing: Tuple Expansion

- Solution: Invalidate local pointers in tuples
- Other components remain valid, usable

\[
\begin{align*}
  x &= \begin{array}{c}
               \downarrow x = \\
              \end{array} \\
  \end{align*}
\]
Global Tuple Selection

• Starting at $x$, can we reach 5?
• Yes, with a proper selection operator

\[ x = \]
Global Tuple Selection

- Selection offsets pointer \textit{within} tuple

\[ x = \]
\[ \@2 \ x = \]

\[ 8 \]
\[ 5 \]
Global Tuple Selection

- Selection offsets pointer within tuple
- Global-to-local pointer works just as before

\[
x = \downarrow @2 x = \downarrow @2 x = 8 \quad \downarrow @2 x = 5
\]
Extended Type Grammar

\[ \omega ::= \text{local} \mid \text{global} \]

\[ \rho ::= \text{valid} \mid \text{invalid} \]

\[ \tau ::= \text{int} \mid \text{boxed} \omega \rho \tau \mid \tau \times \tau \]

- Allow subtyping on validity qualifiers

boxed \omega \text{ valid } \tau \leq \text{boxed } \omega \text{ invalid } \tau

\[ \tau_1 \times \tau_2 \leq \tau_3 \times \tau_4 \iff \tau_1 \leq \tau_3 \land \tau_2 \leq \tau_4 \]
Global Assignment

\[ x : \text{boxed valid global } \tau \]
\[ y : \tau \]
\[ \frac{}{x := y : \tau} \]

\[ x = \]
\[ y = \]

3

5
Global Assignment

\[ x : \text{boxed valid global } \tau \]

\[ y : \tau \quad \text{expand}(\tau) = \tau \]

\[ x := y : \tau \]

\[ x = \]

\[ y = \]

\[ 3 \]

\[ 5 \]
Type Qualifier Inference

• Efficiently infer qualifiers in two passes:
  1. Maximize number of “invalid” qualifiers
  2. Maximize number of “local” qualifiers
• Allows for a range of language designs
  – Complete inference
  – Allow explicit declarations as needed
• On large codes, does better than humans!
Titanium Implementation

• Titanium = Java + SPMD parallelism
  – Focus is on scientific codes

• Global is assumed; local is explicit
  – E.g., “Object local” or “double [] local [] local”

• Local qualification inference in compiler
  – Conservative for valid/invalid qualifiers
  – Monomorphic
Titanium Benchmarks: Speed

Reduction in Execution Time

- lu-fact manual: 1%
- lu-fact auto: 42%
- cannon manual: 6%
- cannon auto: 12%
- sample: 27%
- gnrbs: 2%
- pps: 56%
Titanium Benchmarks: Code Size

Reduction in Code Size

<table>
<thead>
<tr>
<th></th>
<th>lu-fact manual</th>
<th>lu-fact auto</th>
<th>cannon manual</th>
<th>cannon auto</th>
<th>sample</th>
<th>gsrpb</th>
<th>pps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduction</td>
<td>43%</td>
<td>49%</td>
<td>46%</td>
<td>45%</td>
<td>50%</td>
<td>35%</td>
<td>43%</td>
</tr>
</tbody>
</table>
Summary and Conclusions

• For top performance, local/global must be dealt with
• Soundness issues are subtle, but tractable
  – Analysis core is surprisingly simple
• Type qualifier inference is a double win:
  – Programming is easier
  – Optimized code is faster, smaller