Implementation
Strategies for Single Assignment Variables

Frej Drejhammar
KTH – Royal Institute of Technology, Stockholm, Sweden
and
SICS – Swedish Institute of Computer Science

joint work with
Christian Schulte (KTH)
Motivation

- Flow Java: SAVs in the language
  - Need efficient representation
- Previous analysis not applicable
  - True concurrency
  - No search
- Three schemes
  - Forwarding
  - Taylor
  - Hybrid
Overview

- Motivation
- Flow Java
  - Language
  - Implementation
- Implementation of SAVs
  - Forwarding
  - Taylor
  - Hybrid
- Evaluation
- Conclusion
Single Object \( s \);

Introduce \( s \) as a single assignment variable (SAV) of type \( \text{Object} \)

- SAV typed
- Place holder for object
- SAV initially \textit{unbound}
Flow Java: Binding

- SAV of type $t$ bound to objects of type $t$
- Binding $@=$
- `Object o = new Object();`
- `s @= o;`
  - $s$ equivalent to $o$ in any subsequent computation
  - $s$ bound
  - Already bound: exception, if different object
Synchronization

- Accessing unbound SAV suspends
  - Field access
  - Method invocation
- Automatic resumption when bound
Aliasing

- SAVs can be aliased
  - When unbound
  - $x$ and $y$ SAVs: $x @= y$ aliases $x$ and $y$
  - $x$ and $y$ equivalent
  - Binding $x$ or $y$ to $\circ$, binds both
Implementation

- Based on GCJ/libjava
- Native code
- Object representation as C++
- Conservative GC
- OS threads, pthreads
- Synchronization: wait() / notify()
- Monitors: lock() / unlock()
Overview

- Motivation
- Flow Java
  - Language
  - Implementation
- Implementation of SAVs
  - Forwarding
  - Taylor
  - Hybrid
- Evaluation
- Conclusion
Why study known SAV implementation strategies?

- OS threads vs VM (Mozart) ⇒
  - Atomicity
  - Deadlock
- No search ⇒ no trailing
- Token equality ⇒ no unification
Terminology

Synchronization Object
  Place holder
  Small, only aliasing information

Equivalence class
  - Objects aliased to each other
  - $c_0 \in c_1$, merges equiv. classes

Leader
  - An uniquely designated member
SAV implementation

Language

Atomicity / Correctness

Equivalence Class Impl.

bind(a,b)
alias(a,b)
waitdet(r)

ll_is_so(r)
ll_bind(a,b)
ll_alias(a,b)
ll_leader(r)
ll_compress(orig,new)
Invariants

- **Leaders**
  - Determined object \( o \), \( \text{leader}(o) == o \)
  - All \( \text{ll}_{\text{except}} \text{leader}() \) and \( \text{is}_\text{so}() \) operate on leaders
- **Locking**
  - Equiv. Class only modified when lock held
  - Locking, in order of increasing addresses
- **Notification**
  - Binding: all suspended on leader
  - Merge: leader at highest address
Concurrency and Binding

- `a = ll_leader(a)`
- `lock(a)`
- `unlock(a)`
- `b = waitdet(b)`
- `a = ll_leader(a)`
- `lock(a)`
- `unlock(a)`
- `ll_bind(a,b)`
- `notify(a)`
- `unlock(a)`
- `b = waitdet(b)`
- `bind(a,b)`

- `valid?`
- `no` → `unlock(a)`
- `yes` → `ll_bind(a,b)`

- `a leader?`
- `no` → `unlock(a)`
- `yes` → `done`
Concurrency and Alias

\[
\text{alias}(a, b) \\
a = \text{ll\_leader}(a) \\
b = \text{ll\_leader}(b) \\
\text{a==b} \\
\text{none SO?} \\
\text{one SO?} \\
[h, l] = \text{sort}(a, b) \\
\text{lock}(l); \text{lock}(h) \\
\text{unlock}(h) \\
\text{unlock}(l) \\
\text{ll\_alias}(l, h) \\
\text{notify}(h) \\
\text{unlock}(h) \\
\text{unlock}(l) \\
\text{leaders?} \\
\text{done} \\
\text{fail} \\
\text{bind} \\
\text{yes} \\
\text{no} \\
\text{yes} \\
\text{no} \\
\text{yes} \\
\text{no} \\
\text{yes} \\
\text{no} \\
\text{no} \\
\text{no} \\
\text{no} \\
\text{yes} \\
\text{unlock}(h) \\
\text{unlock}(l) \\
\text{valid?} \\
\text{yes} \\
\text{unlock}(h) \\
\text{unlock}(l) \\
\text{fail} \\
\text{done} \\
\text{binding} \\
\text{locking} \\
\text{synchronization}
Concurrency and Synchronization

```
waitdet(o)

\[ t = o \]

unlock(o)

is SO?

\[ o = ll\_leader(o) \]

lock(o)

11\_compress(t,o)

return o

wait(o)

leader?

\[ yes \]

\[ no \]

\[ yes \]

\[ no \]
```
SAV implementation

Language

Atomicity / Correctness

Equivalence Class Impl.

bind(a,b)
alias(a,b)
waitdet(r)

ll_is_so(r)
ll_bind(a,b)
ll_alias(a,b)
ll_leader(r)
ll_compress(orig,new)
Equiv. Class Representation: Forwarding

Object representation
- Similar to WAM
- Extra redirection field

```
ll_is_so(o): o != NULL && rptr != o
```
Equiv. Class Representation: Forwarding

\text{bind}(a, b)
Equiv. Class Representation: Forwarding

\[ \text{alias}(a, b) \]

\[
\begin{align*}
& a \at 0x2000 \quad x \at 0x1000 \quad b \at 0x3000 \\
& \begin{array}{c}
\text{vpotr} \\
0x1000
\end{array} \quad \begin{array}{c}
\text{vpotr} \\
\text{UNB}
\end{array} \quad \begin{array}{c}
\text{vpotr} \\
0x3000
\end{array}
\end{align*}
\]
Equiv. Class Representation: Forwarding

\texttt{ll\_compress(o, n)}

\begin{itemize}
\item \texttt{o @ 0x3000}
\item \texttt{vptr}
\item \texttt{0x1000}
\item \texttt{vptr}
\item \texttt{vptr}
\item \texttt{vptr}
\item field 0
\item \texttt{0x1000}
\item \texttt{vptr}
\item \texttt{vptr}
\item \texttt{vptr}
\item \texttt{0x1000}
\item o @ 0x3000
\item n @ 0x1000
\item n @ 0x1000
\item \texttt{vptr}
\item \texttt{0x1000}
\item \texttt{vptr}
\item field 0
\item \texttt{0x1000}
\item \texttt{vptr}
\item \texttt{vptr}
\item \texttt{vptr}
\item \texttt{0x1000}
\item \texttt{vptr}
\item \texttt{vptr}
\item field 0
\item \texttt{0x1000}
\end{itemize}
Equiv. Class Representation: Taylor

- Originally for Prolog
  - Eliminates reference chains
  - Trailing troublesome

Free \[\square\] \rightarrow\quad \text{Aliased} \quad \square \quad \square

\text{ll_bind}(a, V)

\begin{array}{c}
  V \\
  V \\
\end{array}
Equiv. Class Representation: Taylor

- Leader, lowest address or determined
- No overwrite, one hop left
Equiv. Class Representation: Hybrid, explicit leader

- **Taylor:** `ll_leader` is $O(n)$

- `ll_leader` $O(1)$, `ll_alias` $O(n)$
Overview

- Motivation
- Flow Java
  - Language
  - Implementation
- Implementation of SAVs
  - Forwarding
  - Taylor
  - Hybrid
- Evaluation
- Conclusion
Evaluation: Benchmarks

Alias \((n + n)\), bind \((2n)\)

- Two cases
  - Random locations
  - Maximize length
- lock + verify + update
  - Forward: \(n/2 + 1 + 1\)
  - Hybrid: \(1 + 1 + n/2\)
  - Taylor: \(n + n + 1\)
Evaluation: Results

Accessing each SAV

- Caching dominates
- Sizes
  - Forward: 2
  - Hybrid: 3
  - Taylor: 2
- Taylor < Hybrid
- Path compression
  - Minimal impact
  - Equiv. to hot cache
- Minimize memory footprint
Conclusion

- Three implementation strategies
- Differences to previous work
  - Locking
  - Token equality
  - Updates
- Two level architecture
  - Atomicity / Correctness
  - Equivalence class representation
- Minimize memory footprint for performance