Revisiting Out-of-SSA Translation for Correctness, Code Quality, and Efficiency

Benoit Boissinot (LIP), Alain Darte (LIP),
Benoît Dupont de Dinechin (STMicro),
Christophe Guillon (STMicro), Fabrice Rastello (LIP)

Compsys Team
Laboratoire de l'Informatique du Parallélisme (LIP)
École normale supérieure de Lyon

CGO'09, March 24, 2009, Seattle, WA

Outline

1. SSA foundations
   - Out-of-SSA translation
2. Correctness and code quality
   - Translation with copy insertions
   - Improving code quality
   - Qualitative experiments
3. Speed and memory footprint
   - Linear-time algorithm for coalescing congruence classes
   - Experimental results for speed and memory footprint
4. Conclusion

Outline

1. SSA foundations
   - Out-of-SSA translation
2. Correctness and code quality
   - Translation with copy insertions
   - Improving code quality
   - Qualitative experiments
3. Speed and memory footprint
   - Linear-time algorithm for coalescing congruence classes
   - Experimental results for speed and memory footprint
4. Conclusion

Static single assignment (SSA)

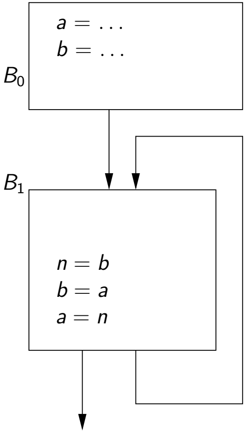
SSA with dominance property

- Unique definition for each variable;
- Each definition dominates its uses.

Static single assignment (SSA)

SSA with dominance property

- Unique definition for each variable;
- Each definition dominates its uses.



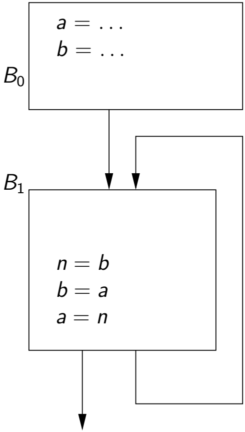
Static single assignment (SSA)

SSA with dominance property

- Unique definition for each variable;
- Each definition dominates its uses.

Conversion into SSA

- Need to introduce ϕ-functions at dominance frontier.



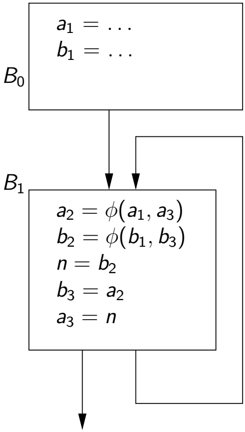
Static single assignment (SSA)

SSA with dominance property

- Unique definition for each variable;
- Each definition dominates its uses.

Conversion into SSA

- Need to introduce ϕ-functions at dominance frontier.



Static single assignment (SSA)

SSA with dominance property

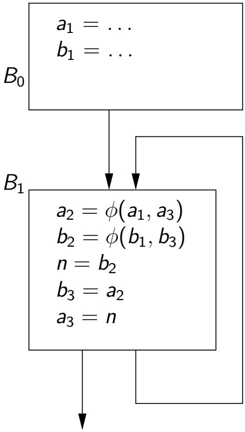
- Unique definition for each variable;
- Each definition dominates its uses.

Conversion into SSA

- Need to introduce ϕ-functions at dominance frontier.

Interests of SSA

- Code optimizations: efficient, easy-to-implement, fast;
- Two-phases register allocation;
- Program analysis/verification.



Static single assignment (SSA)

SSA with dominance property

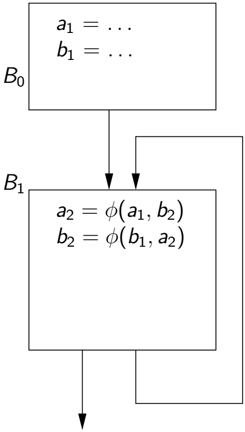
- Unique definition for each variable;
- Each definition dominates its uses.

Conversion into SSA

- Need to introduce ϕ-functions at dominance frontier.

Interests of SSA

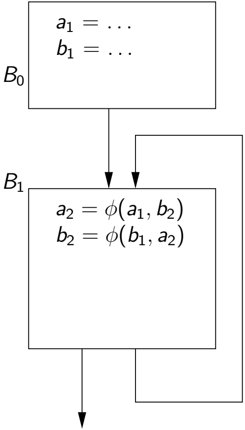
- Code optimizations: efficient, easy-to-implement, fast;
- Two-phases register allocation;
- Program analysis/verification.



Why is out-of-SSA translation difficult?

- Cytron et al. (1991): copies in predecessor basic blocks.

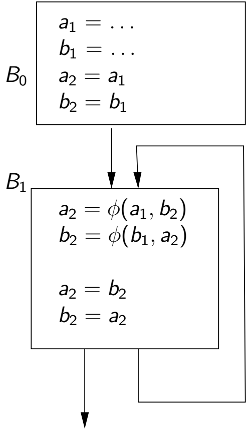
Swap problem



Why is out-of-SSA translation difficult?

- Cytron et al. (1991): copies in predecessor basic blocks.

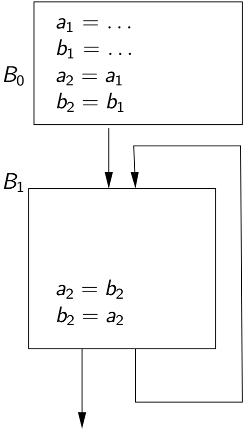
Swap problem



Why is out-of-SSA translation difficult?

- Cytron et al. (1991): copies in predecessor basic blocks. Incorrect!
  - Bad understanding of parallel copies.

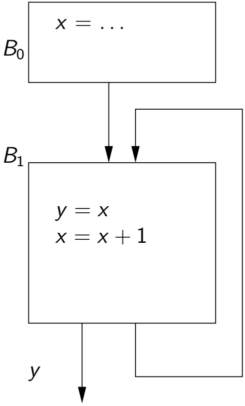
Swap problem



Why is out-of-SSA translation difficult?

- Cytron et al. (1991): copies in predecessor basic blocks. Incorrect!
    - Bad understanding of parallel copies.

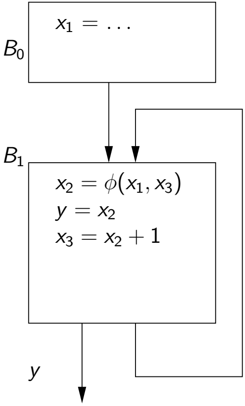
Lost copy problem



Why is out-of-SSA translation difficult?

- Cytron et al. (1991): copies in predecessor basic blocks. Incorrect!
  - Bad understanding of parallel copies.

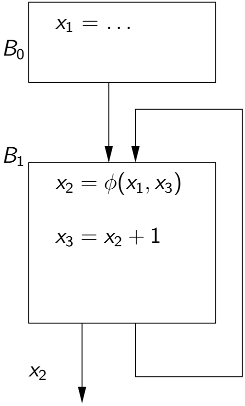
Lost copy problem



Why is out-of-SSA translation difficult?

- Cytron et al. (1991): copies in predecessor basic blocks. Incorrect!
  - Bad understanding of parallel copies.

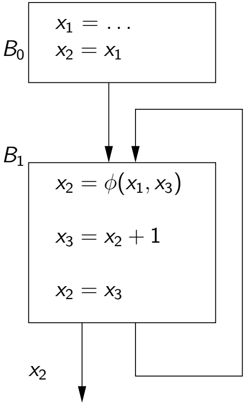
Lost copy problem



Why is out-of-SSA translation difficult?

- Cytron et al. (1991): copies in predecessor basic blocks. Incorrect!
  - Bad understanding of parallel copies.

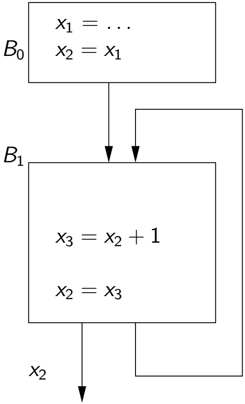
Lost copy problem



Why is out-of-SSA translation difficult?

- Cytron et al. (1991): copies in predecessor basic blocks. Incorrect!
  - Bad understanding of parallel copies;
  - Bad understanding of critical edges and interferences.

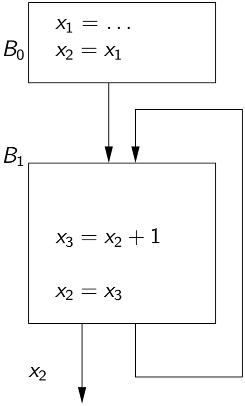
Lost copy problem



Why is out-of-SSA translation difficult?

- Cytron et al. (1991): copies in predecessor basic blocks. Incorrect!
  - Bad understanding of parallel copies;
  - Bad understanding of critical edges and interferences.
- Briggs et al. (1998): both problems identified. General correctness unclear.

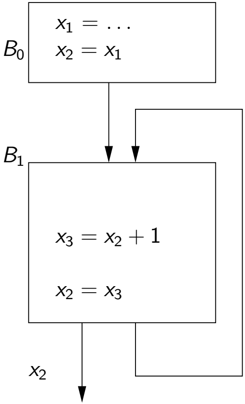
Lost copy problem



Why is out-of-SSA translation difficult?

- Cytron et al. (1991): copies in predecessor basic blocks. Incorrect!
  - Bad understanding of parallel copies;
  - Bad understanding of critical edges and interferences.
- Briggs et al. (1998): both problems identified. General correctness unclear.
- Sreedhar et al. (1999): correct but
  - handling of complex branching instructions unclear;
  - interplay with coalescing unclear;
  - "virtualization" hard to implement.

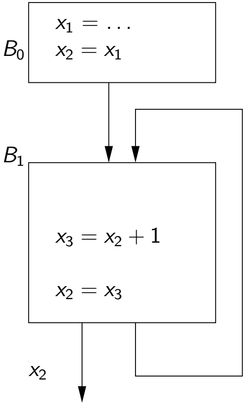
Lost copy problem



Why is out-of-SSA translation difficult?

- Cytron et al. (1991): copies in predecessor basic blocks. Incorrect!
  - Bad understanding of parallel copies;
  - Bad understanding of critical edges and interferences.
- Briggs et al. (1998): both problems identified. General correctness unclear.
- Sreedhar et al. (1999): correct but
  - handling of complex branching instructions unclear;
  - interplay with coalescing unclear;
  - "virtualization" hard to implement.

Lost copy problem



☛ Many SSA optimizations turned off in gcc and Jikes.

Outline

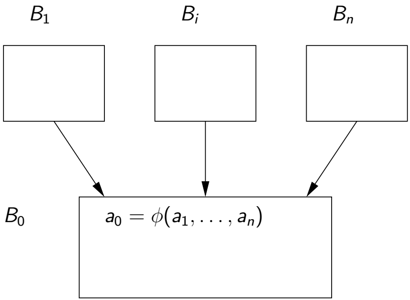
1. SSA foundations
   - Out-of-SSA translation
2. Correctness and code quality
   - Translation with copy insertions
   - Improving code quality
   - Qualitative experiments
3. Speed and memory footprint
   - Linear-time algorithm for coalescing congruence classes
   - Experimental results for speed and memory footprint
4. Conclusion

Going to CSSA (conventional SSA)

Conventional SSA (CSSA)

For any ϕ-function $a_0 = \phi(a_1, \ldots, a_n)$, the variables $a_0, \ldots, a_n$ can be safely replaced by a common ressource.

From SSA to CSSA



Going to CSSA (conventional SSA)

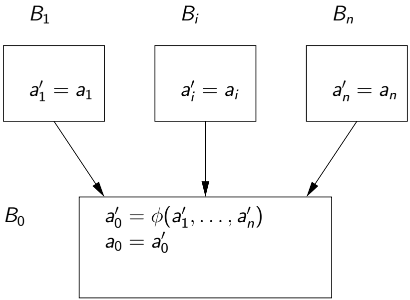
Conventional SSA (CSSA)

For any ϕ-function $a_0 = \phi(a_1, \ldots, a_n)$, the variables $a_0, \ldots, a_n$ can be safely replaced by a common ressource.

Correctness

Add copies with new local variables around every ϕ.
$\Longrightarrow$ CSSA

From SSA to CSSA



Code quality

Removing copies

Useless copies can be removed by standard aggressive coalescing.
Using an accurate notion of interference (value-based) gives excellent results.

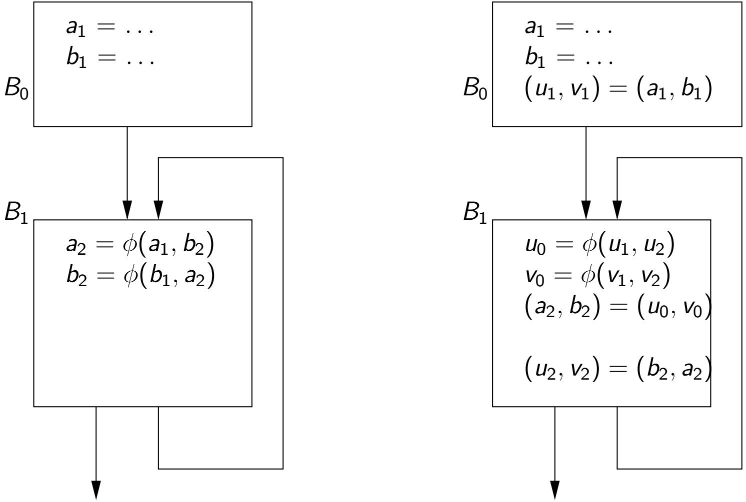
Code quality

Removing copies

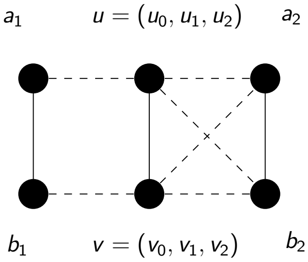
Useless copies can be removed by standard aggressive coalescing. Using an accurate notion of interference (value-based) gives excellent results.

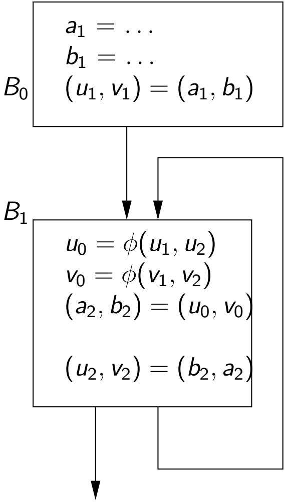
"Liveness of ϕ" defined by the a'_i. ✝ Be careful with potential bugs due to conditional branches that use or define variables.

Coalesced example: the swap problem

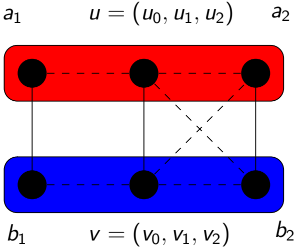


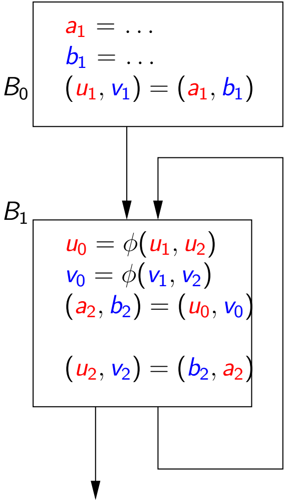
Coalesced example: the swap problem



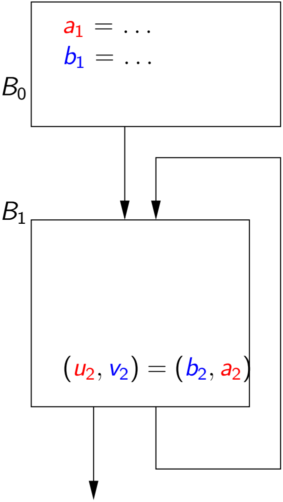


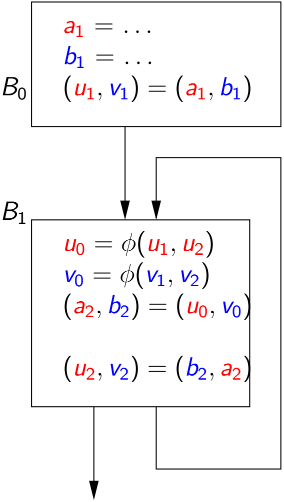
Coalesced example: the swap problem



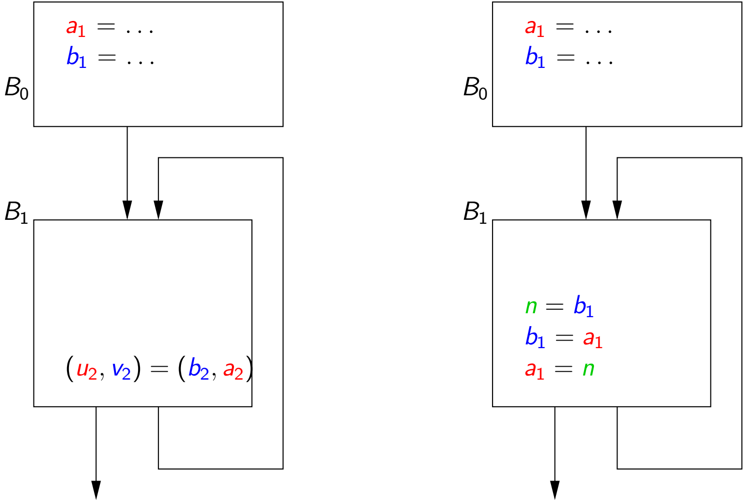


Coalesced example: the swap problem

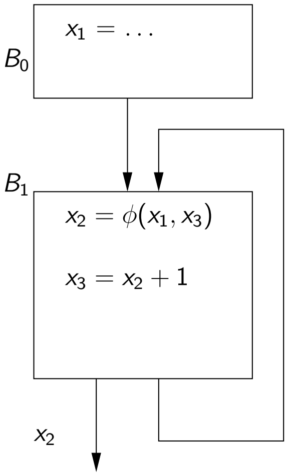


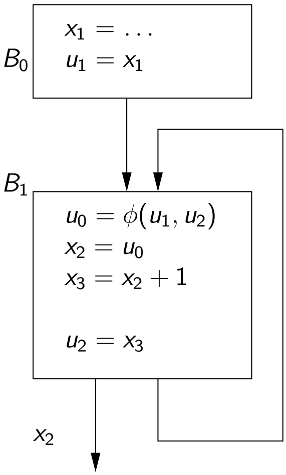


Coalesced example: the swap problem

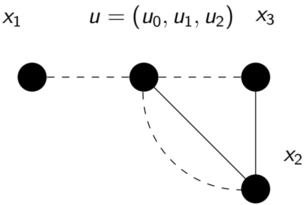


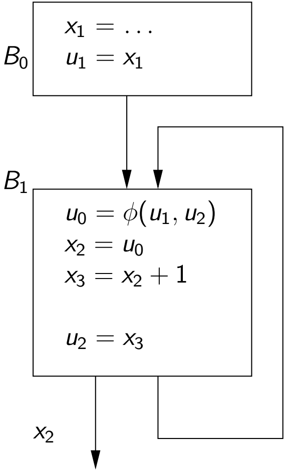
Coalesced example: the lost copy problem



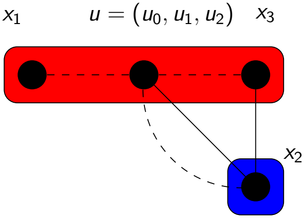


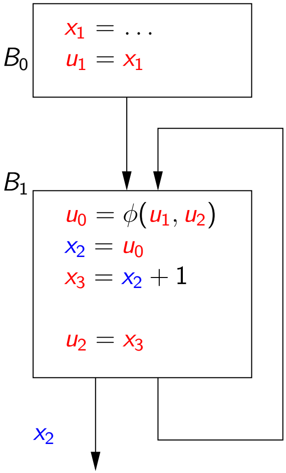
Coalesced example: the lost copy problem



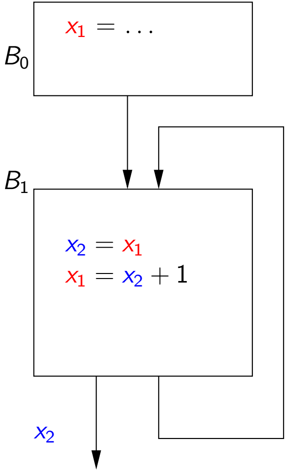


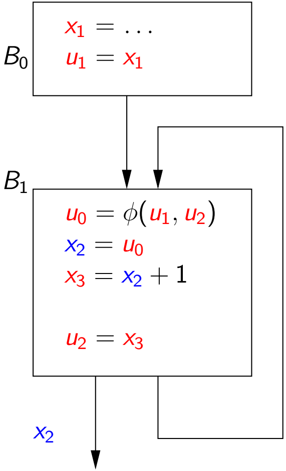
Coalesced example: the lost copy problem





Coalesced example: the lost copy problem

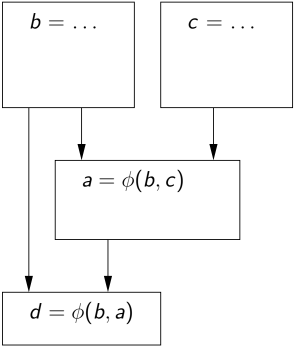




Exploiting SSA: value-based interferences

Definition (Chaitin interference)

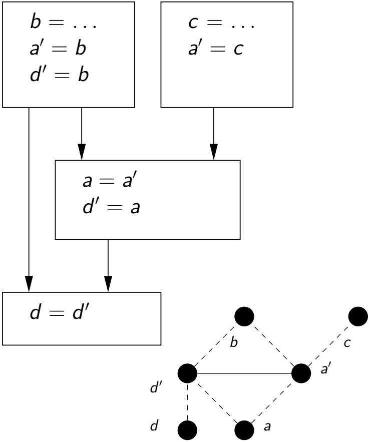
Two variables interfere if one is live at the definition of the other, and it is not a copy of the first.



Exploiting SSA: value-based interferences

Definition (Chaitin interference)

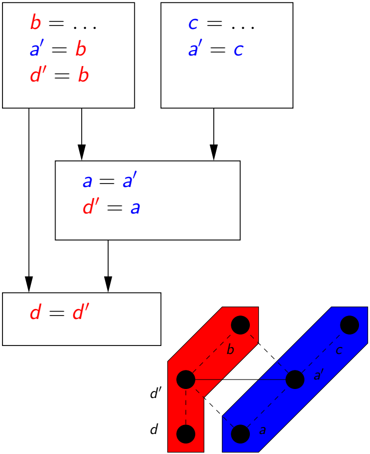
Two variables interfere if one is live at the definition of the other, and it is not a copy of the first.



Exploiting SSA: value-based interferences

Definition (Chaitin interference)

Two variables interfere if one is live at the definition of the other, and it is not a copy of the first.

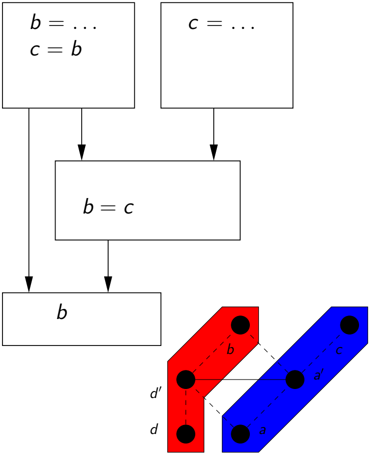


Exploiting SSA: value-based interferences

Definition (Chaitin interference)

Two variables interfere if one is live at the definition of the other, and it is not a copy of the first.

☛ Need to update interference graph after coalescing.



Exploiting SSA: value-based interferences

Definition (Chaitin interference)

Two variables interfere if one is live at the definition of the other, and it is not a copy of the first.

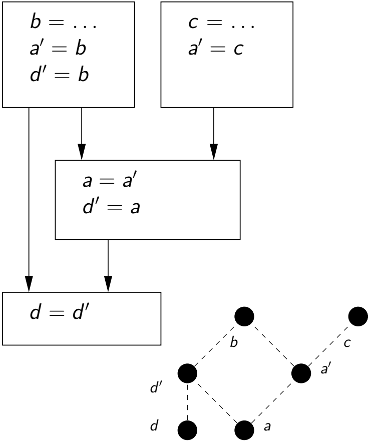
☛ Need to update interference graph after coalescing.

Unique value V of a SSA variable

For a copy $\mathtt{b} = \mathtt{a}$, $V(\mathtt{b}) = V(\mathtt{a})$ (traversal of dominance tree).

Value-based interference

$\mathtt{a}$ and $\mathtt{b}$ interfere if $V(\mathtt{a}) \neq V(\mathtt{b})$ and Live-range($\mathtt{a}$) $\cap$ Live-range($\mathtt{b}$) $\neq \emptyset$.



Exploiting SSA: value-based interferences

Definition (Chaitin interference)

Two variables interfere if one is live at the definition of the other, and it is not a copy of the first.

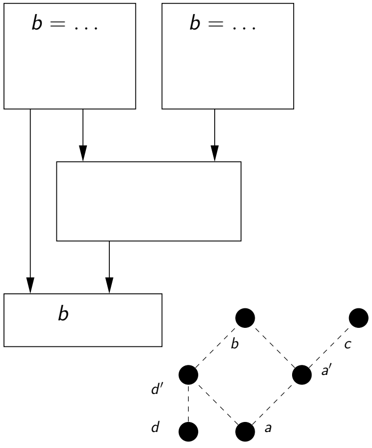
☛ Need to update interference graph after coalescing.

Unique value V of a SSA variable

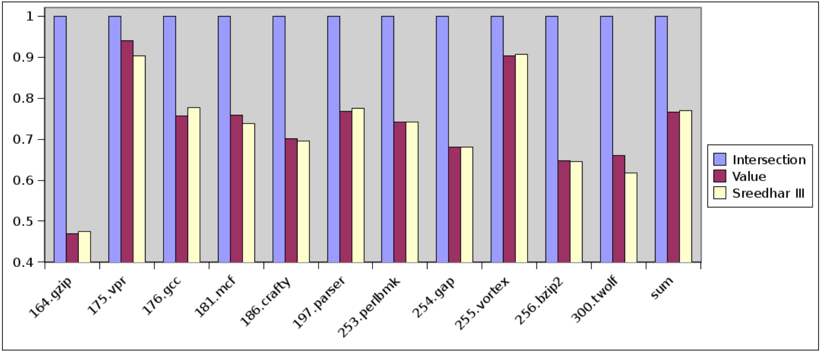
For a copy $\mathtt{b} = \mathtt{a}$, $V(\mathtt{b}) = V(\mathtt{a})$ (traversal of dominance tree).

Value-based interference

a and b interfere if $V(\mathtt{a}) \neq V(\mathtt{b})$ and Live-range(a) $\cap$ Live-range(b) $\neq \emptyset$.



Qualitative experiments with SPEC CINT2000



Number of remaining moves

Qualitative experiments with SPEC CINT2000

Key points of the out-of-SSA translation

- Copy insertion (to go to CSSA and to handle register renaming constraints) followed by coalescing.
- Value-based interferences ☛ coalescing is improved and independent of virtualization (i.e., as in Sreedhar III).
- Parallel copies followed by sequentialization.

Outline

How to coalesce variables?

Two alternatives

- Use a working interference graph where, in case of coalescing, corresponding vertices are merged. $O(1)$ interference query.
- Manipulate congruence classes, i.e., sets of coalesced variables. Interferences must be tested between sets.

Chaitin, Sreedhar, Budimlić use congruence classes. Also useful to avoid interference graph. Naive algorithm: quadratic complexity.

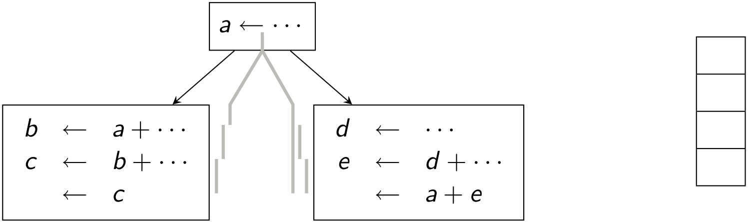
Fast interference test for a set of variables

Key properties for linear-complexity live range intersection

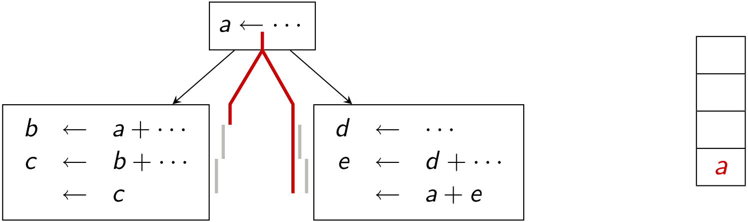
- 2 SSA variables intersect if one is live at the definition of the other.
- In this case, the first definition dominates the second one.
- Budimlić: If a and b intersect (a dom b), then $\forall c$ with a dom c and c dom b: b and c interferes.

$\Longrightarrow$ For each variable, the only test needed is with the "closest" dominating variable

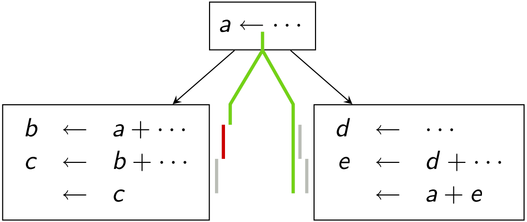
Fast interference test for a set of variables

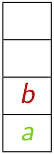


Fast interference test for a set of variables

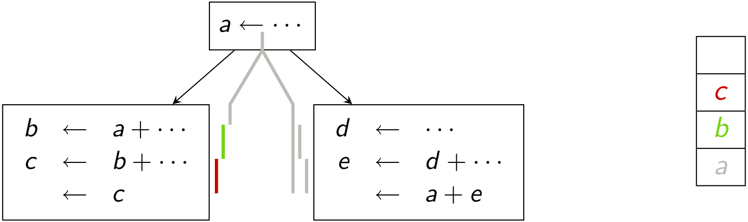


Fast interference test for a set of variables

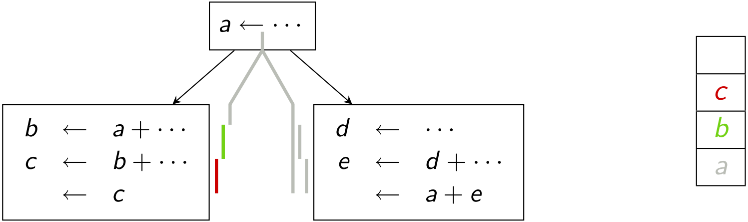




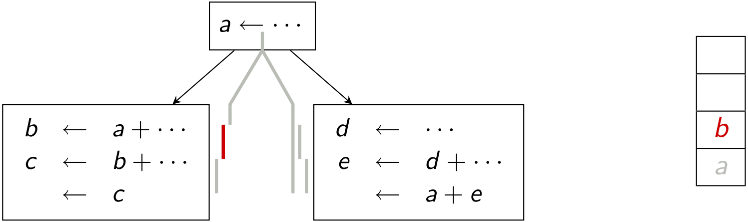
Fast interference test for a set of variables



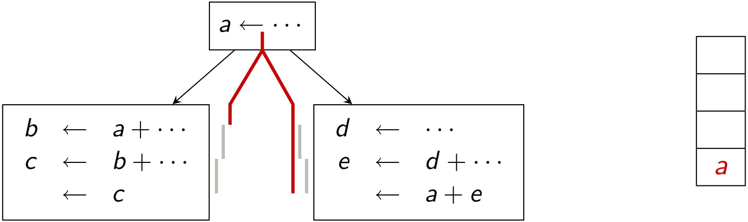
Fast interference test for a set of variables



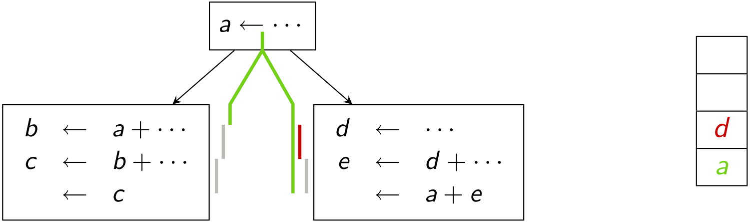
Fast interference test for a set of variables



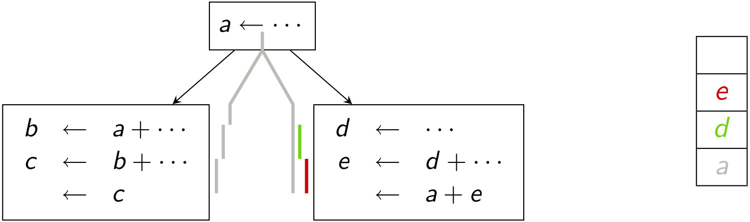
Fast interference test for a set of variables



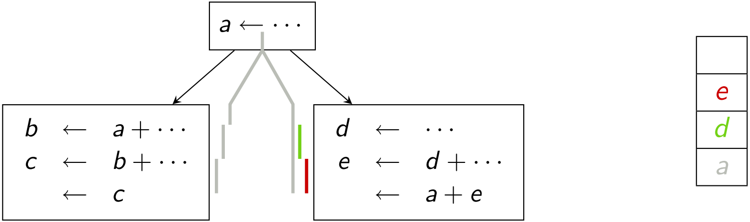
Fast interference test for a set of variables



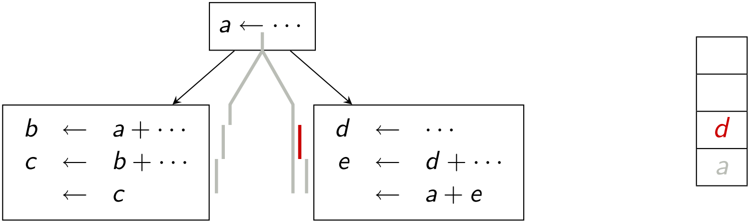
Fast interference test for a set of variables



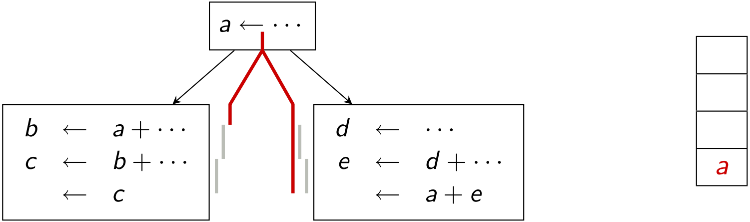
Fast interference test for a set of variables



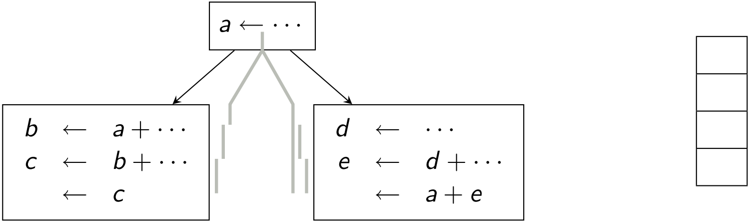
Fast interference test for a set of variables



Fast interference test for a set of variables



Fast interference test for a set of variables



Linear interference test of two congruence classes

Generalization to interference test of two sets

- Emulate a stack-based DFS traversal of dominance tree, for two sorted sets instead of one ☛ linear number of tests. Also no need to test intersection of variables in the same set.
- Take values into account for value-based interference: need links of "equal ancestors", which may increase complexity.
- Sort in linear time the resulting set, in case of coalescing.

Linear interference test of two congruence classes

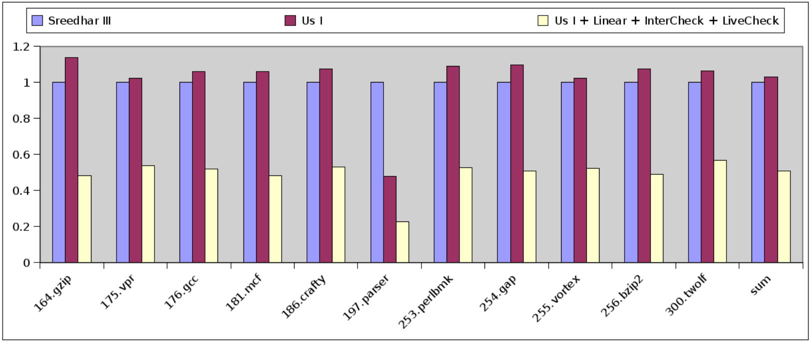
Generalization to interference test of two sets

- Emulate a stack-based DFS traversal of dominance tree, for two sorted sets instead of one ☛ linear number of tests. Also no need to test intersection of variables in the same set.
- Take values into account for value-based interference: need links of "equal ancestors", which may increase complexity.
- Sort in linear time the resulting set, in case of coalescing.

Fewer intersection tests ☛ possible now to use more expensive queries for intersection and liveness and avoid interference graph:

- Budimlić intersection test, still using liveness sets.
- Fast liveness checking of Boissinot et al. (CGO'08).

Speed-up for SPEC CINT2000: x2

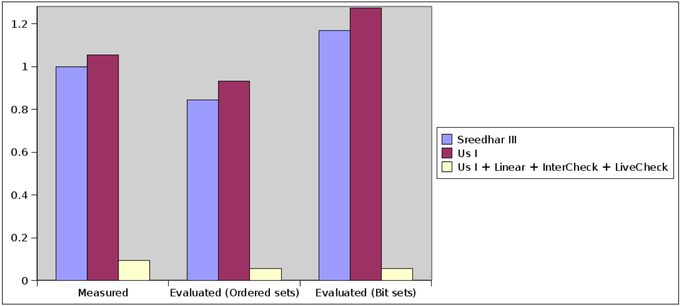


Time to go out of SSA (valgrind cycles)
Default: Liveness sets + interference graph

Memory footprint reduction for SPEC CINT2000: x10

- Interference graph: half-size bit matrix.
- Liveness sets: enumerated sets. Does not count construction.
- Livenesss check: bit sets. Construction taken into account.

Data structures grow during virtualization. "Perfect memory" evaluated, with both enumerated/bit sets for liveness sets.



Max of memory footprint

Outline

General framework

- Correctness clarified even for complex cases
- Two phases solution, based on coalescing

Results

- Value-based interference as good as Sreedhar III
- Fast algorithm: Speed-up x2, memory reduction x10.

Implementation

- No need to virtualize (at least for us)
- Simple implementation

General framework

- Correctness clarified even for complex cases
- Two phases solution, based on coalescing

Results

- Value-based interference as good as Sreedhar III
- Fast algorithm: Speed-up x2, memory reduction x10.

Implementation

- No need to virtualize (at least for us)
- Simple implementation

Journal version should contain:

- Virtualization of register renaming constraints.
- All pseudo-codes, e.g., virtualization in general case.

The End

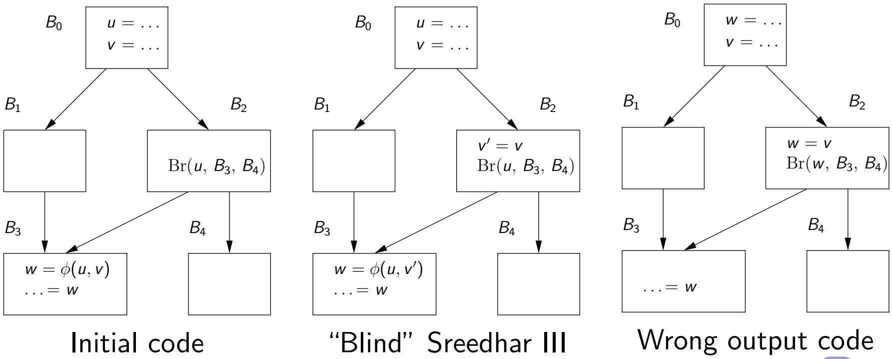
Thank you!

Bug tracking RVM-254 of Jikes RVM

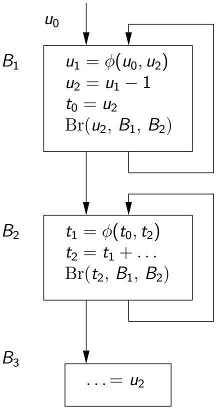
Problems with SSA form: lack of loop unrolling breaks VM

This problem is probably one of the most serious in the RVM currently. When loop unrolling is disabled and SSA enabled the created IR is corrupt. The error has in the past look like we were suffering from the "lost copy" problem, but implementing a naive solution to this didn't solve the problem. Their is sound logic behind the code so we need to identify a small test case where things are broken and then reason about what's wrong in leave SSA. This has been attempted once (with the code that removes an element from the live set) but the problem no longer appears to surface here. Currently these optimizations are disabled but by RVM 3.0 they should be re-enable and this bug cured.

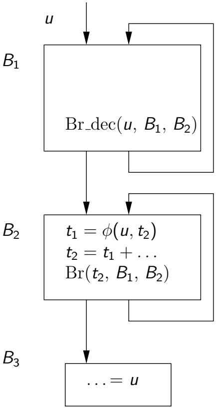
Potential bugs with conditional branches



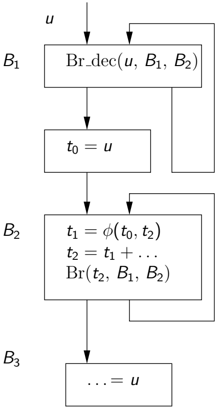
Unfeasible out-of-SSA translation example



Initial code

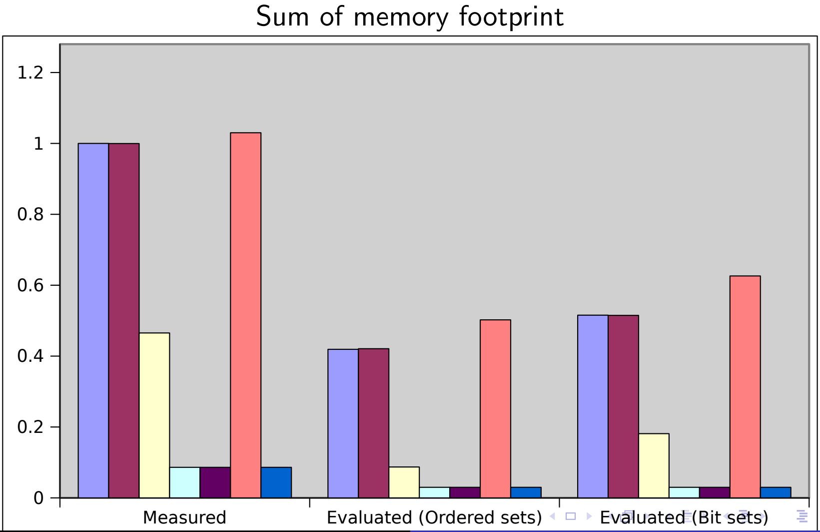


After optimization

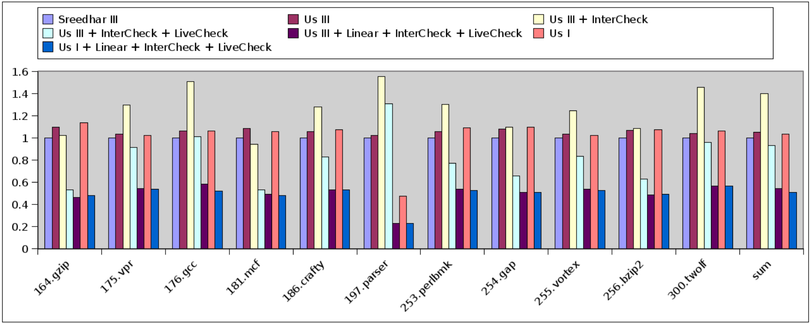


Needs edge splitting

Sum of memory over benchmarks



Speedup


Sreedhar III
Us III
Us III + InterCheck
Us III + InterCheck + LiveCheck
Us III + Linear + InterCheck + LiveCheck
Us I
Us I + Linear + InterCheck + LiveCheck
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
164.gzip
175.vpr
176.gcc
181.mcf
186.crafty
197.parser
253.perlbmk
254.gap
255.vortex
256.bzip2
300.twolf
sum

Qualitatives