



Producer-Side Platform-Independent Optimizations and Their Effects on Mobile-Code Performance

Philipp Adler, Wolfram Amme, Jeffery von Ronne,
Michael Franz

Presentation: Andreas Gampe

Mobile Code

SSA & SafeTSA

Platform-Independent Optimizations

Results

Conclusion

References

Mobile Code

- ▶ Mobile Code \neq Mobile Agents
- ▶ Definition (ADL-TABATABAI et.al., 1996):

“The term *mobile code* describes any program that can be shipped unchanged to a heterogenous collection of processors and executed *with identical semantics* on each processor.”



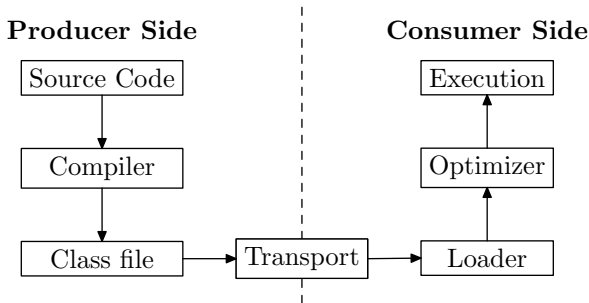
Mobile Code

- ▶ What falls under this definition?
 - ▶ Source Code (limited)
 - ▶ Fat Binaries
 - ▶ Scripts
 - ▶ Platforms with intermediate code (Java, .NET, etc.)
- ▶ Today's research focuses on last one, mainly Java



Producer & Consumer

- ▶ Environment split in two parts:
 - ▶ Code Producer
 - ▶ Code Consumer





Consumer Side

- ▶ Loads and executes Mobile code
- ▶ Loading
 - ▶ Binary data → Internal structures
 - ▶ Verification
- ▶ Executing
 - ▶ Interpreter
 - ▶ Just-in-Time (JIT) Compiler

Consumer Side Optimization

- ▶ All machine-dependent optimizations
 - ▶ Strength reduction
 - ▶ Register allocation
 - ▶ Instruction scheduling
 - ▶ ...
- ▶ Needs to be fast and light on resources
- No deep analysis
- Simple Algorithms

Producer Side

- ▶ Compiles source code to intermediate representation (IR)
 - ▶ Has “all the time in the world”
- Can do heavy analysis
- ▶ Aim: do as many optimizations on producer side as possible
 - ▶ Limits:
 - ▶ Only machine-independent optimizations
 - ▶ Safety (e.g. bounds check elimination)



SSA & SafeTSA

Mobile Code

SSA & SafeTSA

Platform-Independent Optimizations

Results

Conclusion

References



SSA

- ▶ Static Single Assignment form
- ▶ Developed during the 1980s at IBM
- ▶ Most important paper:

R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck
Efficiently computing static single assignment form and
the control dependence graph.

ACM Transactions on Programming Languages and Systems, 13(4):451-490,
Oct 1991

SSA - Definition & Example

- ▶ A program is in SSA form if every variable is defined exactly once.
- ▶ Simple Example

Not SSA

```

a = f(1)
g(a)
⋮
a = f(2)  !!!
g(a)
    
```

SSA

```

a = f(1)
g(a)
⋮
b = f(2)
g(b)
    
```

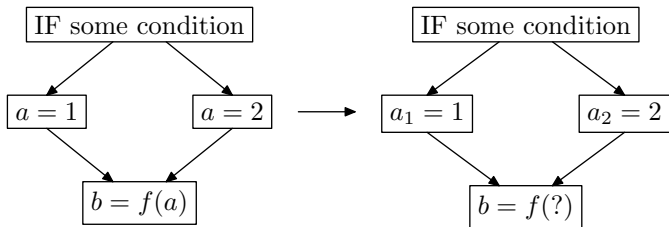
Transformation to SSA

- ▶ Transform by renaming variables
- ▶ Usual visualization: subscripts
- ▶ Example:

$$\begin{array}{ccc} a = f(1) & & a_1 = f(1) \\ g(a) & & g(a_1) \\ \vdots & \rightarrow & \vdots \\ a = f(2) & & a_2 = f(2) \\ g(a) & & g(a_2) \end{array}$$

Problem: Nontrivial Control Flow

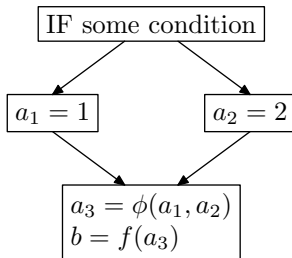
- Assume a conditional control flow, e.g. IF-THEN-ELSE, where a variable is modified in both flows



- Which renamed variable should be used?

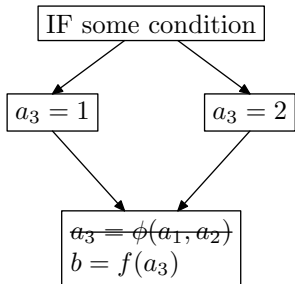
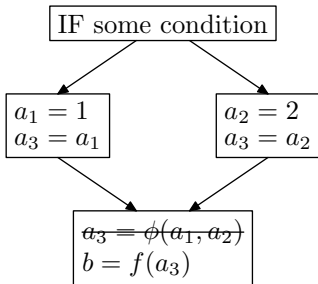
Solution: ϕ -Functions

- ▶ New “statement” type: ϕ -Function
 - ▶ Has as many parameters as control flows entering (IF=2)
 - ▶ Unites the data flow
 - ▶ Produces the output of the actual branch taken



Transformation out of SSA

- ▶ SSA usually modelled after target IR
- ▶ ϕ -Functions have to be removed
 - ▶ Create Copy instructions at end of branches
 - ▶ Place variables in same registers



Advantages of SSA

- ▶ Explicit Def-Use-Chains
- No Def-Use Analysis necessary
- Simplifies most optimizations
- ▶ Recent research: Register Allocation in SSA form



SafeTSA

- ▶ Michael Franz, Jeffery von Ronne, Wolfram Amme
- ▶ Around 2000 (first paper)
- ▶ Safe Typed SSA form
- ▶ IR that replaces Byte-code
- ▶ Based on functional SSA form with some modifications



Modifications

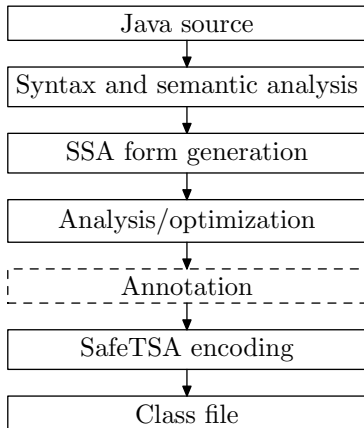
- ▶ Unified Abstract Syntax Tree
 - ▶ Retains high-level structure of source program
- ▶ Referential Integrity
 - ▶ Definition before Use
 - ▶ Ensured by Encoding
- ▶ Type Safety
 - ▶ Strongly typed
 - ▶ Not discarded for transfer

Advantages of SafeTSA

- ▶ Faster compilation
- ▶ Class files usually smaller
- ▶ Simple verification at load-time
- ▶ No SSA conversion necessary
- ▶ UAST allows optimizations to recognize high-level structures
- ▶ Many optimizations can be done on producer side
- ▶ Type concept allows safe annotations

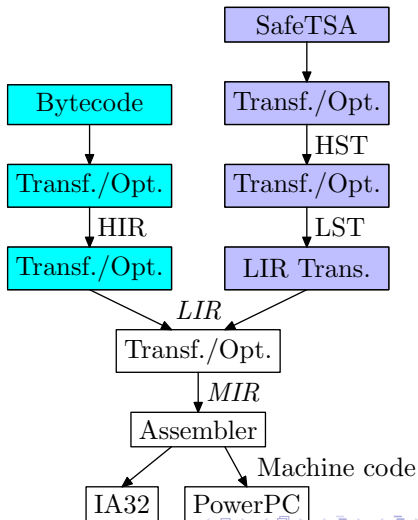
Implementation - Producer Side

- ▶ Called Otter
- ▶ Based on Pizza compiler



Implementation - Consumer Side

- ▶ Based on Jikes RVM
- ▶ Extended for SafeTSA files
- ▶ Can run mixed code





Platform-Independent Optimizations

Mobile Code

SSA & SafeTSA

Platform-Independent Optimizations

Results

Conclusion

References



Platform-Independent Optimizations

- ▶ Optimizations that can be done without knowledge of the consumer side:
 - ▶ CPU
 - ▶ RAM
 - ▶ Cache

Note: for some optimizations assumptions are made.

- ▶ Usually high-level changes

Dead Code Elimination

- ▶ Unreachable Code

```
if true :
```

```
    something
```

```
else :
```

```
    something else
```

- ▶ More general: unnecessary code

```
 $x = a \times b$ 
```

```
 $y = c \times d$ 
```

```
return x
```

Constant Propagation & Folding

- ▶ References of constant value variables

$$a = 3$$

$$b = a \times 100$$

- ▶ All constant parameters

$$b = 3 \times 100$$

- ▶ Usually done together, because propagation makes folding possible and vice versa



Static Final Field Resolution

- ▶ Reference of a final, static field
 $y = \text{AClass.ACONSTANT} \times 100$
- ▶ Removes the field access
- ▶ Required by the Java Language Specification

Common Subexpression Elimination

- ▶ Two identical expressions evaluating the same result

$$\begin{array}{ccc}
 x = a \times b & & x = a \times b \\
 f(x) & & f(x) \\
 \vdots & \rightarrow & \vdots \\
 y = a \times b & & \\
 g(y) & & g(x)
 \end{array}$$

- ▶ Can be done
 - ▶ globally (GCSE)
 - ▶ locally (LCSE)



Local Get Elimination

- ▶ Removes redundant array accesses

$$a_{i+1} = a_i + a_i$$

- ▶ Only possible on local or synchronized arrays

Global Code Motion

- ▶ Rearrange code structure by moving instructions
- ▶ Move loop invariants out of loops
- ▶ Move variable assignments to shorten life span

$y = 2 \times a$		$x = 2 \times b$
$\text{while } (i < 10)$		$\text{while } (i < 10)$
$x = 2 \times b$	\rightarrow	$i = i + x$
$i = i + x$		$y = 2 \times a$
$f(y)$		$f(y)$

Global Value Numbering

- ▶ Similar to Common Subexpression Elimination
- ▶ Computes a mapping: expression $\rightarrow N$
- ▶ Same mapped number means expressions are equal
- ▶ Amongst other things: finds common expressions in divergent execution paths

if x :		$y = a \times b$
$f(a \times b)$		if x :
else :	\rightarrow	$f(y)$
$g(a \times b)$		else :
		$g(y)$

Mobile Code

SSA & SafeTSA

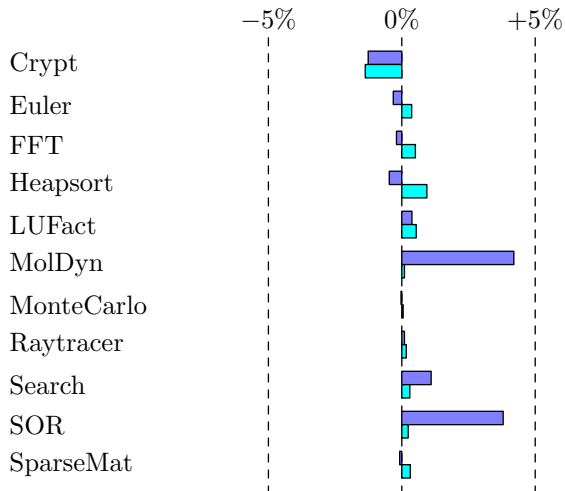
Platform-Independent Optimizations

Results

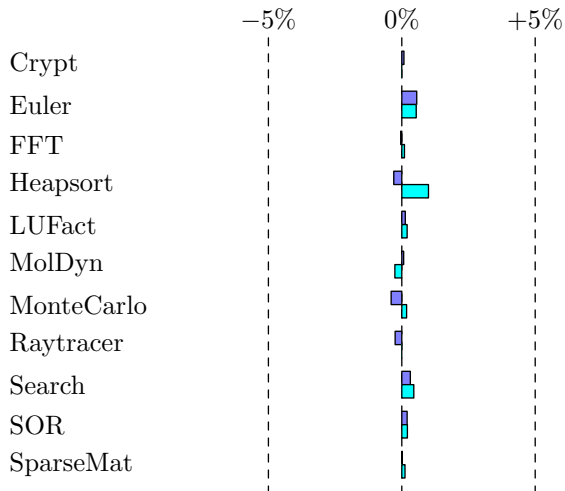
Conclusion

References

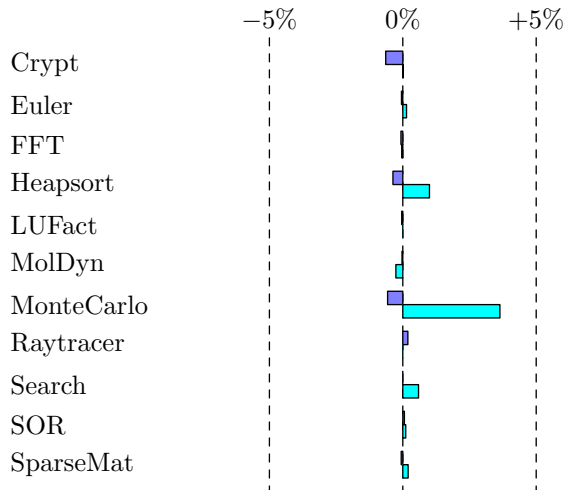
Dead Code Elimination



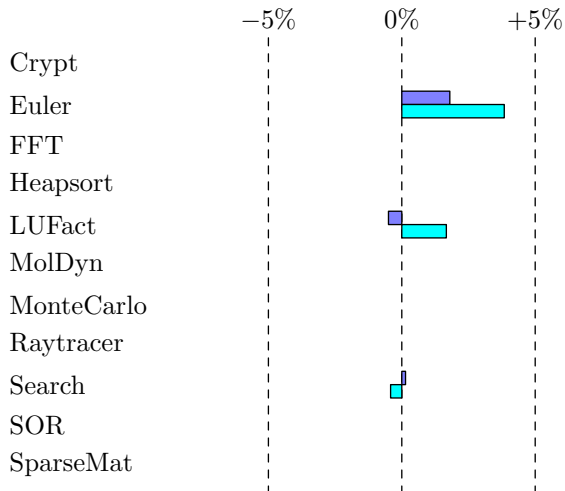
Constant Propagation & Folding



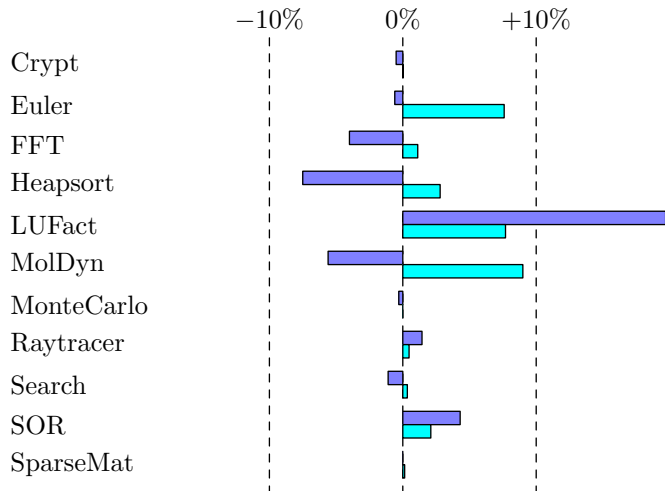
Static Final Field Resolution



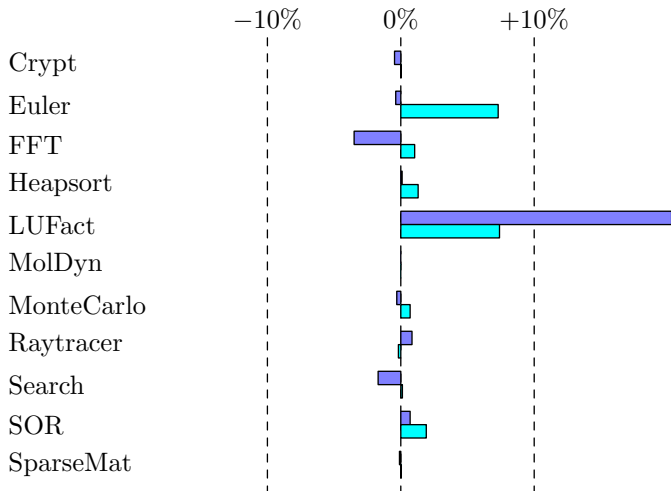
Local Get Elimination



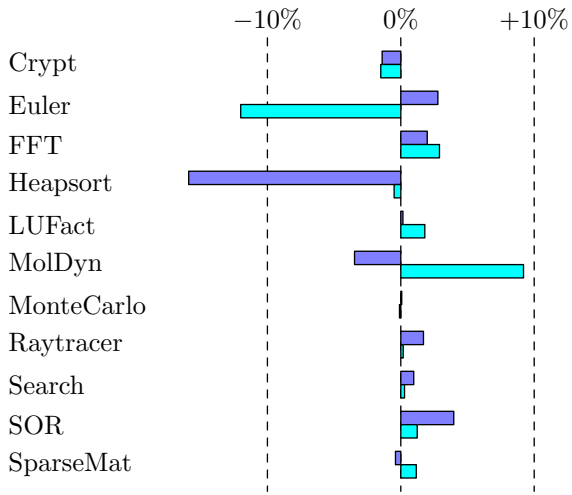
Global Common Subexpression Elimination



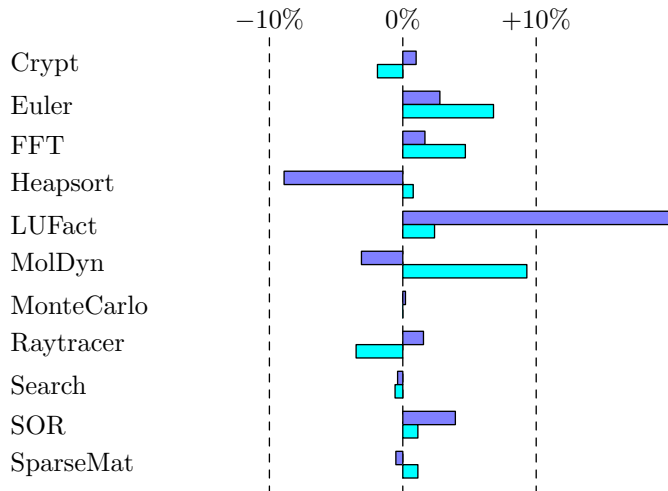
Local Common Subexpression Elimination



Global Code Motion



Global Value Numbering





Conclusion

Mobile Code

SSA & SafeTSA

Platform-Independent Optimizations

Results

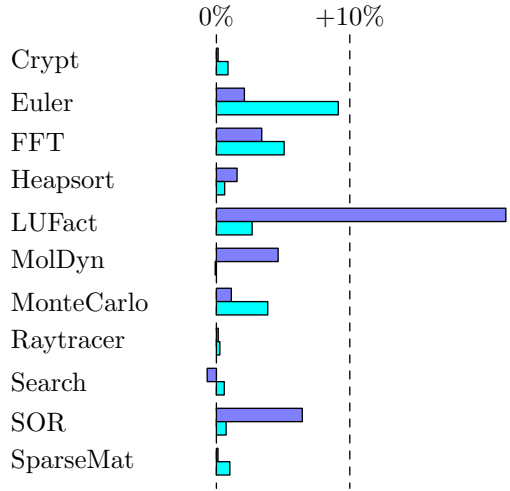
Conclusion

References



Conclusion

- ▶ There is not such a thing as a platform-independent optimization
- ▶ But it's usually a good idea to do some of them, anyways





References

Mobile Code


SSA & SafeTSA


Platform-Independent Optimizations


Results

Conclusion

References

- 
 P. Adler, W. Amme, J. von Ronne, M. Franz
Producer-Side Platform-Independent Optimizations and Their Effects on Mobile-Code performance.
 10th IEEE Workshop on Interaction between Compilers and Computer Architectures (INTERACT-10,2006)

- 
 A.-R. Adl-Tabatabai, G. Langdale, S. Lucco, R. Wahbe
Efficient and language-independent mobile programs.
 Proceedings of the Conference on Programming Language Design and Implementation (PLDI'1996)

- 
 W. Amme, N. Dalton, M. Franz, J. von Ronne
SafeTSA: A type safe and referentially secure mobile-code representation based on static single assignment form.
 Proceedings of the Conference on Programming Language Design and Implementation (PLDI'2001)