The Oracle - Linz JVM Research Cooperation

Hanspeter Mössenböck

Johannes Kepler University Linz, Austria
hanspeter.moessenboeck@jku.at

How it all began

2000
Sabbatical at Sun Microsystems
(Java HotSpot group)

Worked with Robert Griesemer
(architect of the Client compiler)

Work during the sabbatical

- started to add SSA form to the HIR of the Client compiler
- implemented a graph-coloring register allocator

Project continued in Linz (initially with master students)

Current project landscape & people

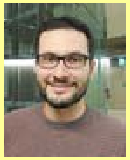
Oracle Labs @ JKU

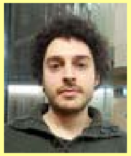
Thomas Würthinger

Christian Wirth

Lukas Stadler

Roland Schatz


Danilo Ansaloni


Daniele Bonetta

HotSpot @ JKU

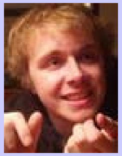
Thomas Schatzl

JKU-funded student

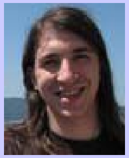
Matthias Grimmer

Oracle-funded students

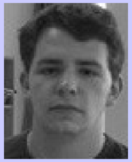
Gilles Duboscq

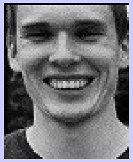

Bernhard Urban

Andreas Wöss

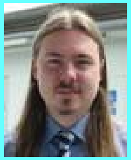

Josef Eisl

Christian Humer

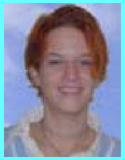

David Leopoldseder


Thomas Feichtinger

Externally funded students


Philipp Lengauer

Peter Hofer

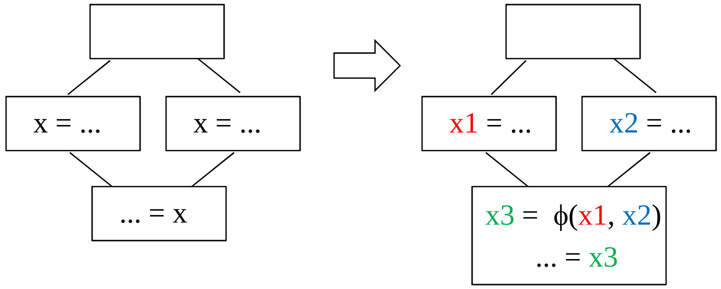

Verena Bitto

David Gnedt

Christian Häubl

Overview of some previous projects

SSA Form for the Client Compiler



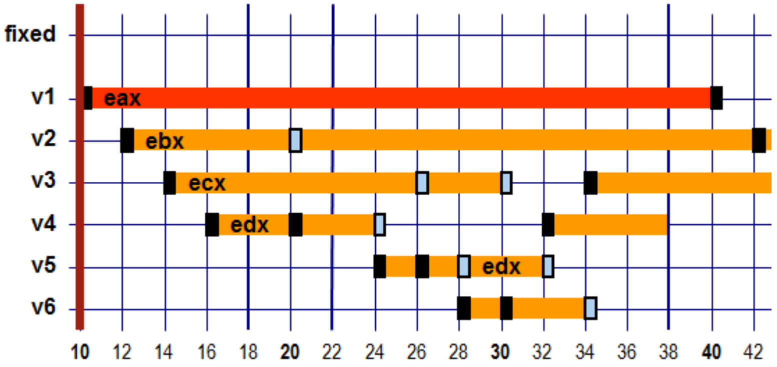
Basis for many optimizations

Single-pass generation of SSA form

Now part of the product Client compiler

Brandis & Mössenböck (TOPLAS'94)

Linear Scan Register Allocation



Faster than Graph Coloring
Better worst-case behavior
=> very suitable for JIT compilers

Now part of the product Client compiler

Wimmer et al. (CC'02, VEE'05)

Escape Analysis & Scalar Replacement

```
void foo(int x, int y) {
    Point p = new Point(x, y);
    ...
    ... p.x ...
    ... p.y ...
}
```

⇨

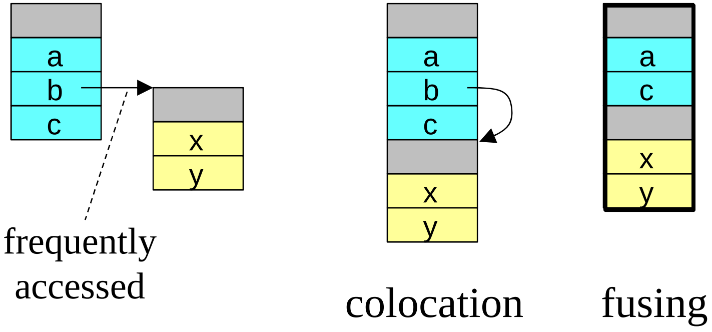
```
void foo(int x, int y) {
    ...
    ...
    ... x ...
    ... y ...
}
```

Does an allocation
- escape the allocating method?
- escape the allocating thread?

Scalar Replacement
- fewer objects
- faster access

Kotzmann (VEE'05, CGO'07)

Object Colocation & Object Fusing



colocation

fusing

Adaptive dynamic profiling

Colocation improves cache behavior

Fusing replaces field accesses with address calculations

Wimmer (JMLC'06, VEE'07, CGO'08, TACO'10)

Dynamic Code Evolution

Run-time changes to classes

Staff

change supertypes

```
class Employee extends ~~Person~~ {
    int     id;
    String  department;
    ~~int     salary;~~
    ...          ⇦ int phone;
    ...
}
```

delete fields/methods

insert fields/methods

Can happen at any point in time
(even if affected methods are running)

Changes all existing objects to their new structure

Type-safe continuation after changes

Würthinger (PPPJ'10, GCPE'10, OOPSLA'11, SCP'11)

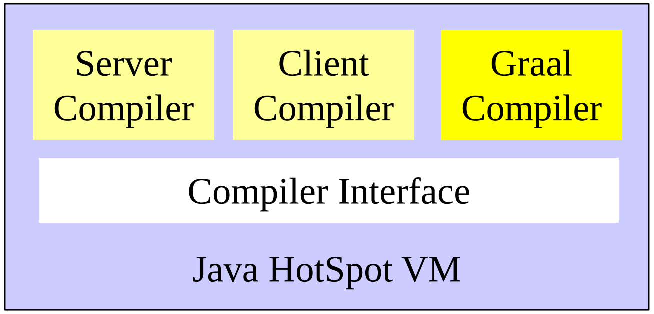
Other Former Research Topics

• Array Bounds Check Elimination	Würthinger (PPPJ'07, SCP'09)
• IR Graph Visualization	Würthinger (PPPJ'08)
• Java String Optimizations	Häubl (PPPJ'08, SCP'10)
• Java Continuations & Coroutines	Stadler (PPPJ'09, PPPJ'10)
• Optimizations of Class Metadata in GC	Schatzl (PPPJ'11)
• ...	

Overview of some recent projects

Graal

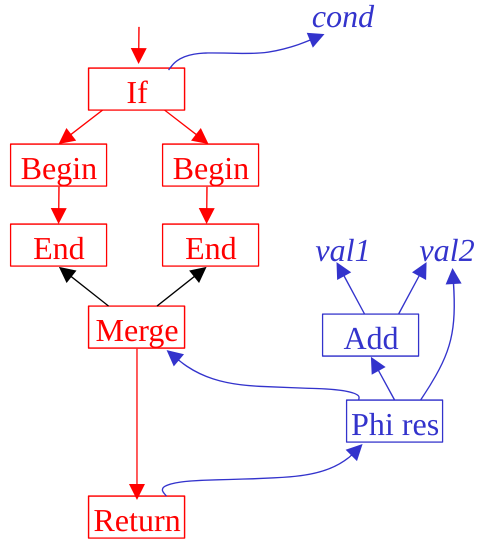
A new Java JIT compiler ...



- written in Java
- using Java snippets (for injecting Graal IR)
- part of OpenJDK project
- basis for most our ongoing projects

... based on a new IR

```
if (cond)
    res = val1 + val2;
else
    res = val2;
return res;
```



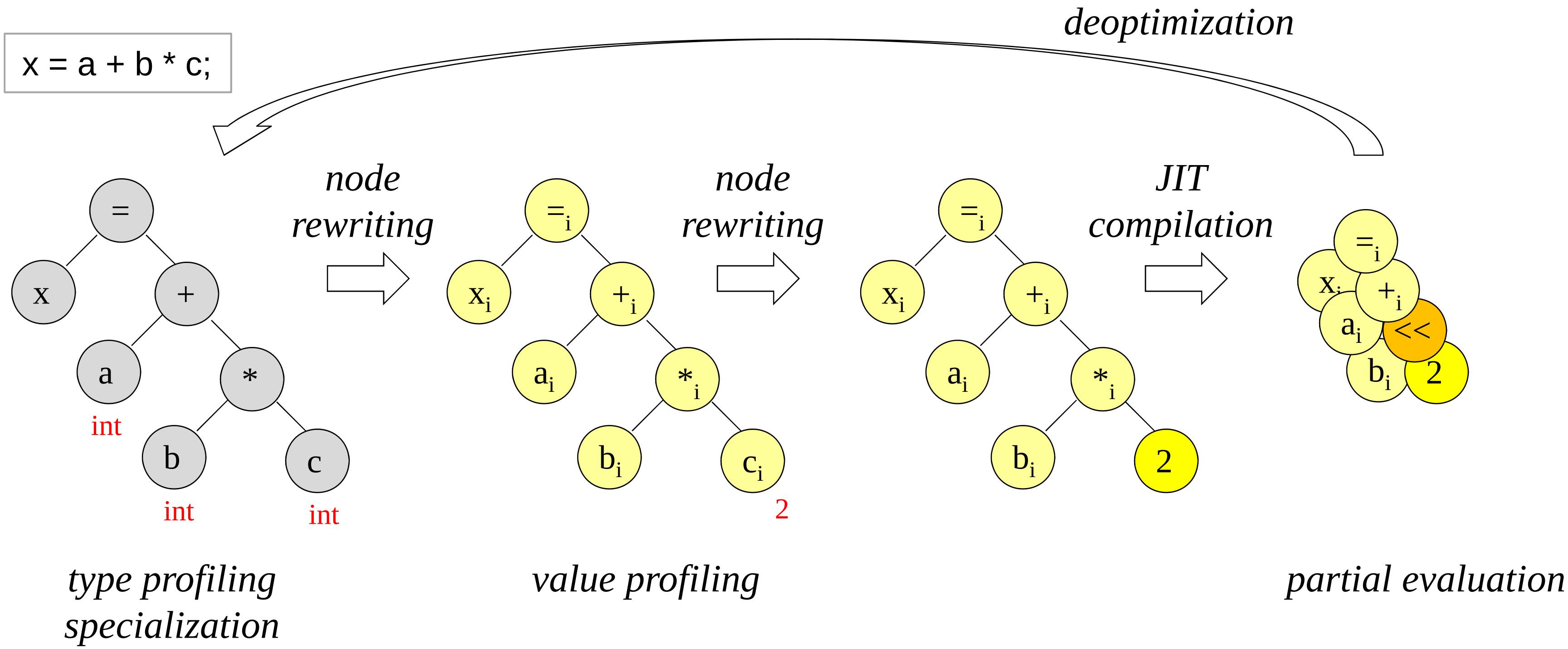
→ control flow

→ data flow

- represents both control flow and data flow
- floating nodes allow for more optimizations
- several lowering steps
  VM independent => VM dependent
  VM dependent => Architecture dependent
- special support for speculative optimizations
- SPLASH'11, VMIL'12, Onward!'13

Truffle

Self-optimizing AST interpreter with JIT compilation



x = a + b * c;

deoptimization

node rewriting

node rewriting

JIT compilation

type profiling specialization

value profiling

partial evaluation

Truffle is a language implementation framework under Graal
Languages implemented so far: JavaScript, Python, Ruby, Smalltalk, R, C, ...

Highly efficient

Publications: DLS'12, PPPJ'13, PPPJ'14, SPLASH'14

Projects under Graal and Truffle

Graal

- Partial Escape Analysis (Stadler: CGO'14)
- Speculative Guard Optimizations (Duboscq: PPPJ'14)
- Vectorization (Schatz)
- Compilation Queuing & Graph Caching (Stadler: VMIL'12)

Truffle

- Several language implementations (JavaScript, C, (R))
- Cross-language interoperability (Grimmer: PPPJ'14, SPLASH'14, PPPJ'13)
- Self-optimizing bytecode interpretation (Urban)

Other

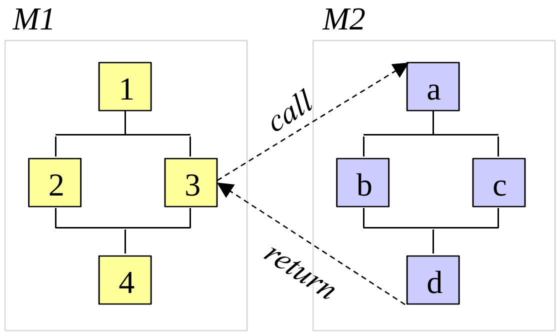
- Control-flow sensitive Linear Scan Optimizations (Eisl)

Trace Compilation

funded by the
Austrian Science Foundation
(FWF)

Trace Compilation for Java

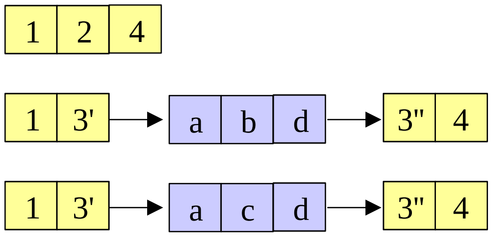
Funded by the Austrian Science Foundation (FWF)



Compiles hot traces (paths) instead of hot methods

- smaller compilation units => faster compilation
- more precise (excludes cold parts)
- context-sensitive trace inlining
  more optimization potential across methods
  (parameters, receiver type)

possible traces



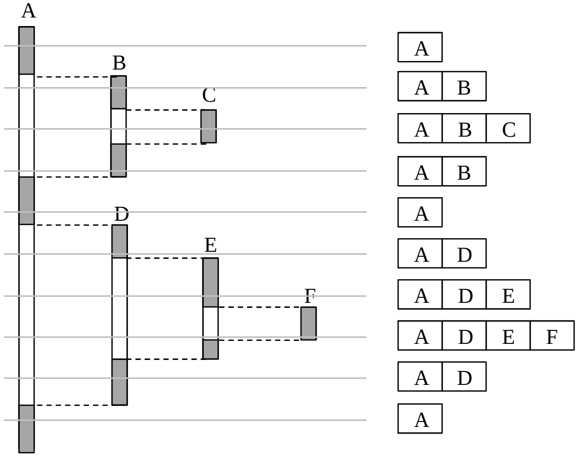
Inlining ideas influenced IR inlining in Truffle

Häubl (PPPJ'11, SAC'11, PPPJ13, Comp.Lang.'13)

Application Performance Monitoring

funded by the
Christian Doppler Society
and Compuware Austria

Efficient Stack Sampling



Efficient even for high sampling rates
(2% overhead at 1ms sampling intervals)

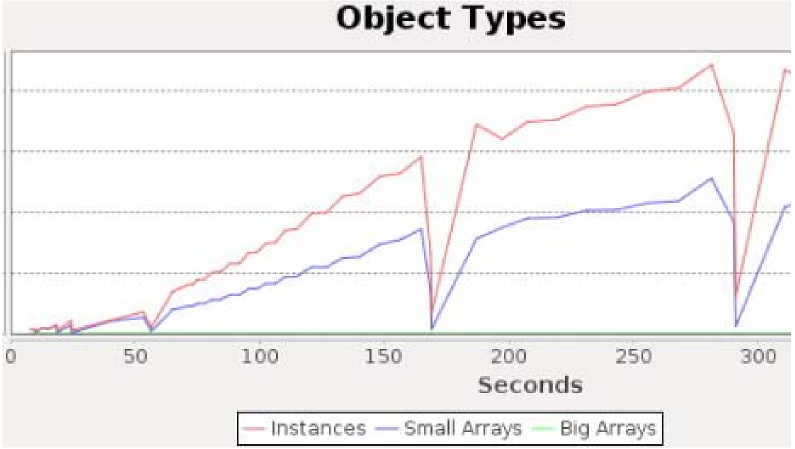
Accurate: can take samples anywhere
(not just at safepoints)

Asynchronous processing of the samples

Variant: lazy stack sampling

Hofer et al. (ICPE'14, PPPJ'14)

Memory Monitoring



We trace allocations, reclamations, GC moves

Offline analysis of
- event traces
- heap layout at any point in time

Tracing overhead < 10%

Lengauer et al. (SPLASH'13, ICPE'14, VaMoS'14)

Summary

What are the benefits of such a cooperation?

It's a Win Win situation

Benefits for Oracle

- Access to bright students
- Full IP rights at relatively low costs
- Extra visibility in the research community (conferences, journals, ...)

Benefits for JKU

- Interesting projects with practical impact
- Helps us to attract the best of our students
- Builds up a critical mass
- Long-term funding gives planning reliability