Divide and Allocate: The Trace Register Allocation Framework

CGO 2018 Student Research Competition (SRC)

Josef Eisl

February 2018

@zapstercc oracle/graal zapster.cc/work

INSTITUTE FOR SYSTEM SOFTWARE

Oracle Labs

Motivation

Register Allocation in a Just-in-Time Compiler

Register Allocation at run time

Spend compile time budget wisely

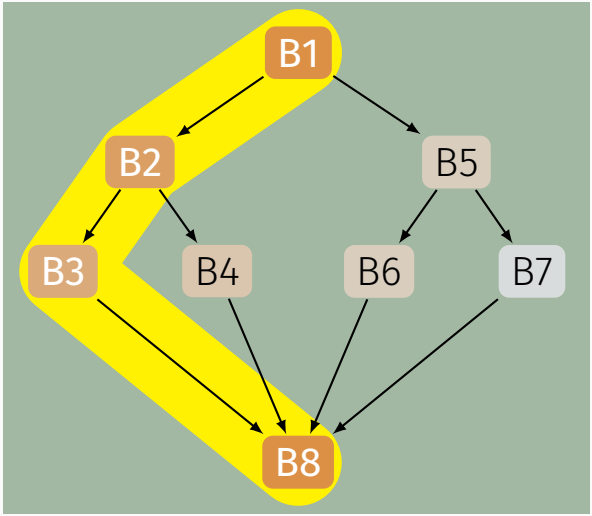
Compilation units are potentially large

Aggressive *speculative* optimizations, e.g., inlining or duplication

Profiling information available!

Optimize for the common case

Global Register Allocation



time budget spent for 50% hot and 50% cold blocks

Problem with Global Approaches

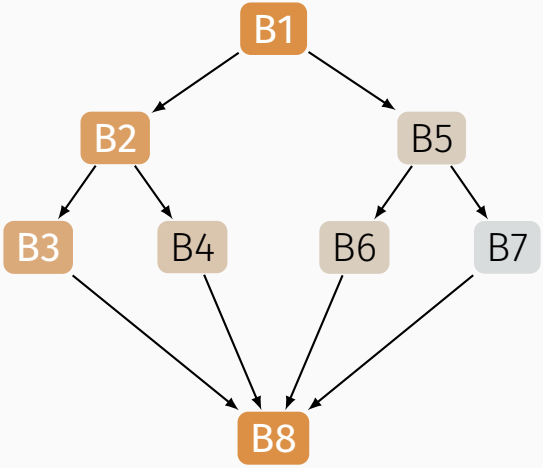
The global scope is not adequate because...

...not all parts are equally important

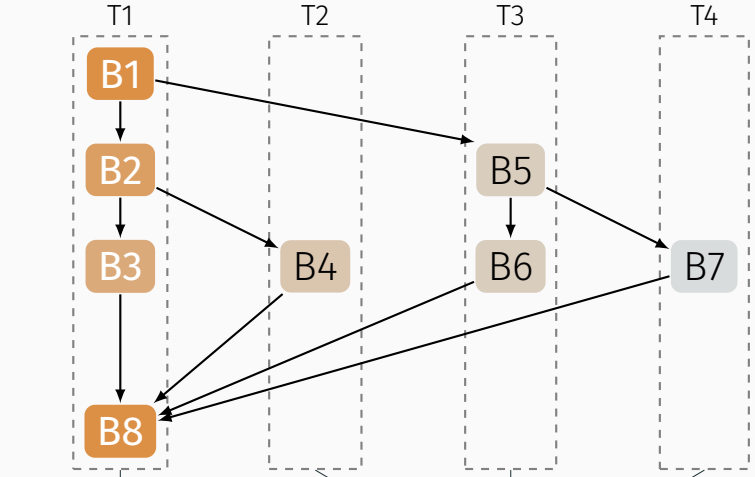
...cold parts can (negatively) influence hot parts

...compile time does not scale w.r.t. method size

Focus on the Common Case



Focus on the Common Case



Research Questions

Can the trace-based approach...

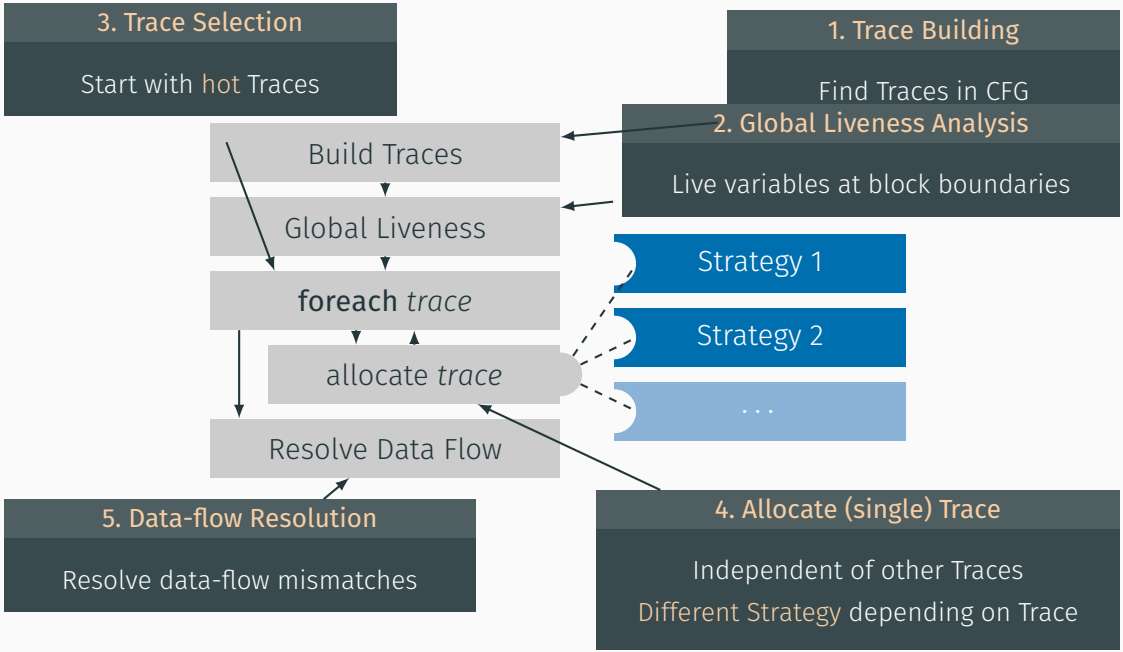
RQ1: ...reach the same quality of allocation as a (state-of-the-art) global allocator?

RQ2: ...be as fast as a global allocator?

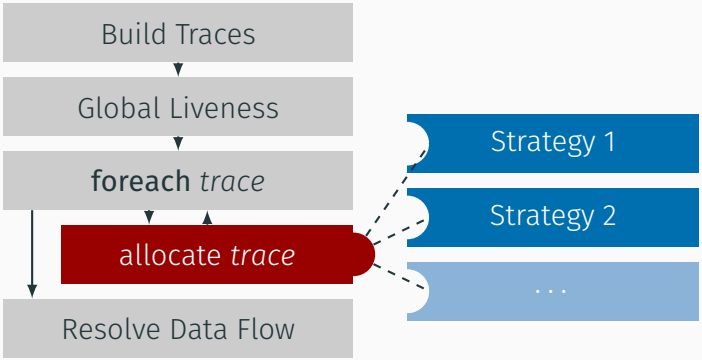
RQ3: ...enable fine-grained control w.r.t. compile-time vs. allocation quality trade-offs?

Trace Register Allocation Framework

Overview


3. Trace Selection
Start with hot Traces
1. Trace Building
Find Traces in CFG
2. Global Liveness Analysis
Live variables at block boundaries
Build Traces
Global Liveness
foreach trace
allocate trace
Resolve Data Flow
Strategy 1
Strategy 2
. . .
5. Data-flow Resolution
Resolve data-flow mismatches
4. Allocate (single) Trace
Independent of other Traces
Different Strategy depending on Trace

Allocation Strategies



What is a Register Allocation *Strategy*?

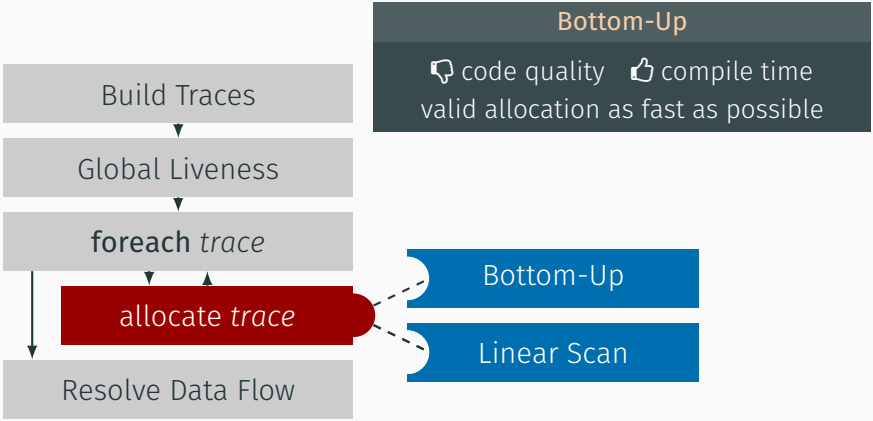
A register allocation algorithm for a single Trace

Simple structure: Linear sequence of code
nice liveness properties (no holes, SSA-form)

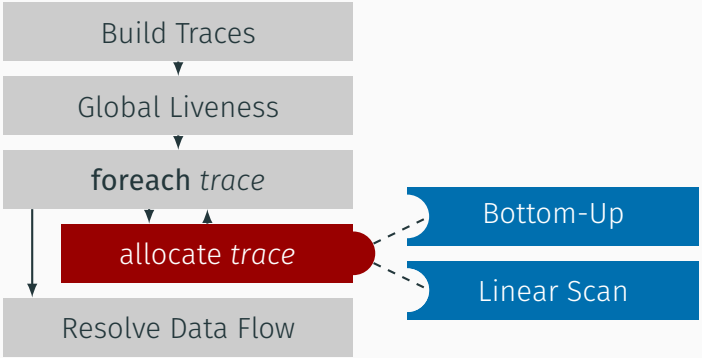
(basically a large basic block)

Evaluation using the Graal Compiler

Allocation Strategies

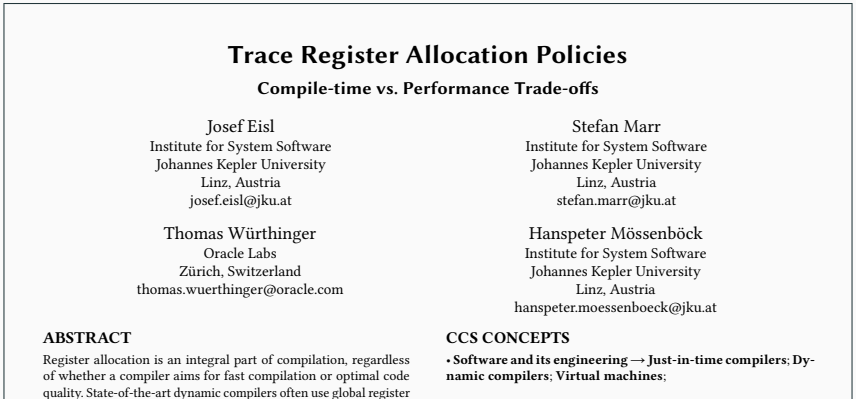


Trace Register Allocation Policy



Which Traces are important (Linear Scan) and which are not (Bottom-Up)?

- ★ Evaluated 8 *parameterized* policies
- ★ Compared 14 configurations
- ★ Analyzed the compile time vs. peak performance impact



Trace Register Allocation Policies

Compile-time vs. Performance Trade-offs

Josef Eisl
Institute for System Software
Johannes Kepler University
Linz, Austria
josef.eisl@jku.at

Stefan Marr
Institute for System Software
Johannes Kepler University
Linz, Austria
stefan.marr@jku.at

Thomas Würthinger
Oracle Labs
Zürich, Switzerland
thomas.wuerthinger@oracle.com

Hanspeter Mössenböck
Institute for System Software
Johannes Kepler University
Linz, Austria
hanspeter.moessenboeck@jku.at

ABSTRACT

Register allocation is an integral part of compilation, regardless of whether a compiler aims for fast compilation or optimal code quality. State-of-the-art dynamic compilers often use global register

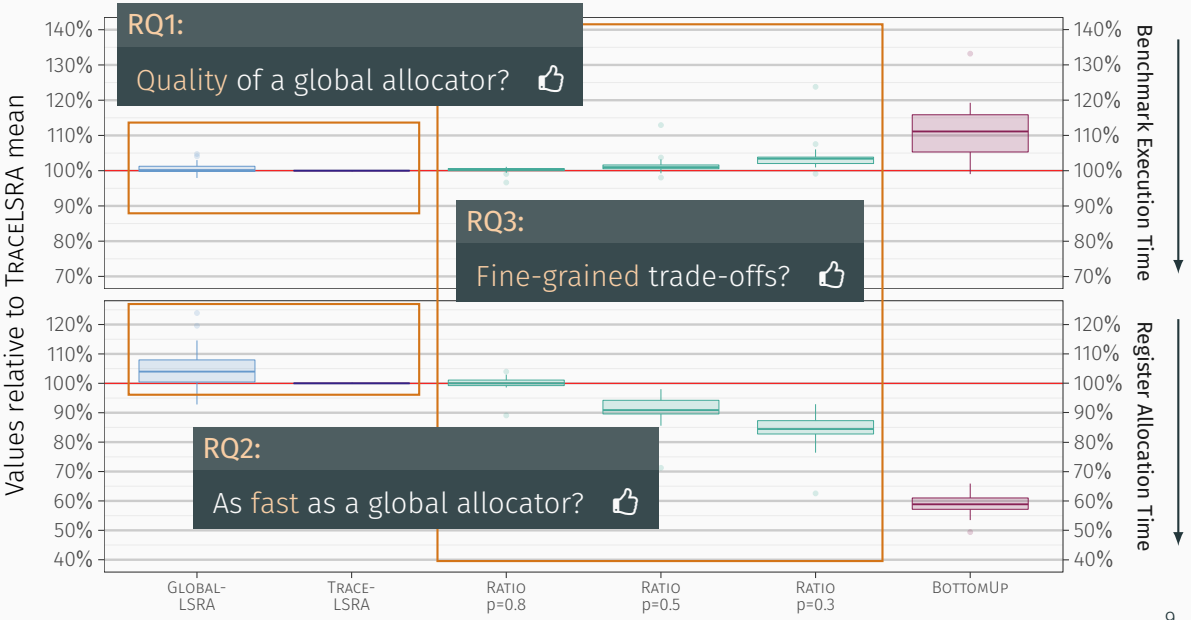
CCS CONCEPTS

• **Software and its engineering → Just-in-time compilers**; **Dynamic compilers**; **Virtual machines**;

Allocation Policies (Selection)

RATIO (p)	"Use Linear Scan for the first p% Traces"
GLOBALLSRA	"*Global* Linear Scan" (for reference)
TRACELSRA	"Always use *Trace-based* Linear Scan"
BOTTOMUP	"Always use Bottom-Up"

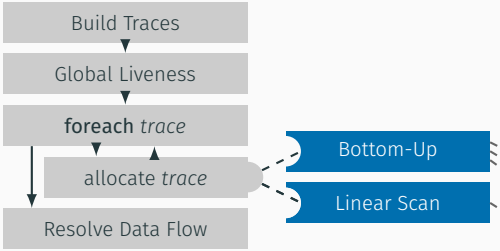
Results – Peak Performance vs. Compile Time (Ratio)



Results for DaCapo/Scala DaCapo (lower is better)

Summary

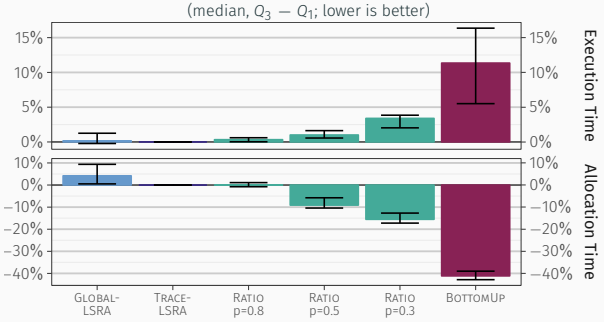
Highlights of the *Trace Register Allocation Framework*

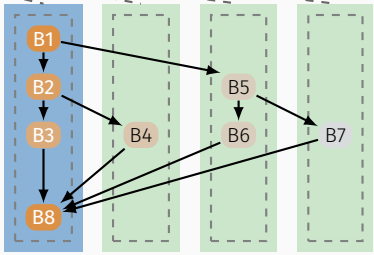


Novel non-global register allocator

More flexible than existing approaches

Designed for JIT compilation





References

References i