Divide and Allocate: The Trace Register Allocation Framework

Josef Eisl

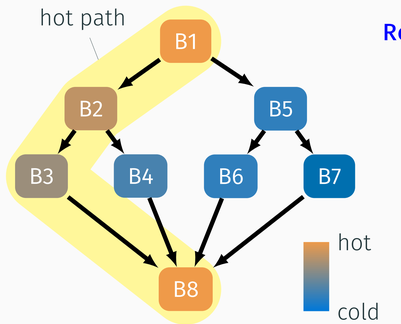
JOHANNES KEPLER UNIVERSITY LINZ

Institute for System Software

Just-in-Time Register Allocation

- Register allocation at run time
- Compilation units are potentially large
- Profiling information available!

Problem with Global Approaches



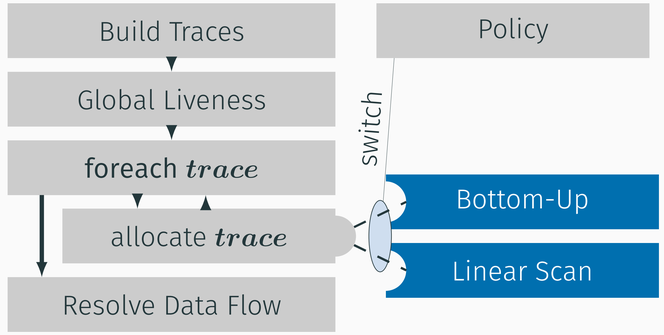
	hot	cold
global	50%	50%
goal	*80%*	*20%*

```
Result accessArray(
    Object[] o, int i) {
  Result r =
    new Result();       //B1
  if (o != null) {      //B2
    if (i >= 0) {       //B3
      r.val = o[i];
    } else {            //B4
      r.ex = idxOOB(o, i);
    }
  } else {              //B5
    if (shldDeopt()) {//B6
      r.frm = toInt(o, i);
    } else {            //B7
      r.ex = npe();
    }
  }                     //B8
  return r;
}
```

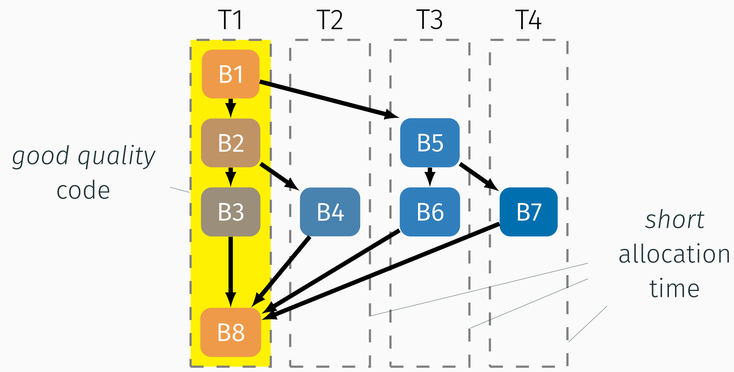
Global Scope not Adequate

- Not all parts are equally important
- Cold parts can influence hot parts
- Compile time does not scale

Trace Register Allocation Framework



Focus on the *Hot Parts*



Advantages of Traces

Easy to compute

Single pass over basic blocks, based on profiling

Simple structure

Linear sequence of code, no lifetime holes

Allocation Strategies

	code quality	compile time
Bottom-Up	👎	👍
Linear Scan	👍	👎

Research Questions

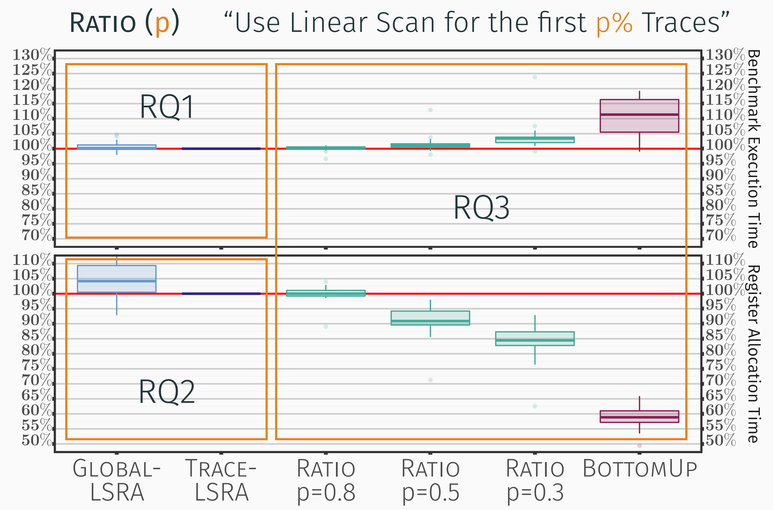
Can the trace-based approach...

RQ1: ...reach the same code quality?
(vs. *state-of-the-art* global allocator)

RQ2: ...be as fast as a global allocator?
(for the same quality of allocation)

RQ3: ...enable fine-grained trade-offs?
(compile-time vs. allocation quality)

Results – Evaluation in GraalVM



DaCapo and Scala-Dacapo.
Values relative to TRACELSRA mean.
↓ Lower is better.

Eisl, Josef, Matthias Grimmer, et al. (2016). "Trace-based Register Allocation in a JIT Compiler". *PPPJ '16*.

Eisl, Josef, Stefan Marr, et al. (2017). "Trace Register Allocation Policies: Compile-time vs. Performance Trade-offs". *ManLang 2017*.

https://github.com/oracle/graal @zapstercc @ josef.eisl@jku.at http://ssw.jku.at/General/Staff/Eisl/

Oracle Labs