Improving Trace-Based JIT Optimisation using Whole-Program Information

Maarten Vandercammen, Coen De Roover

Software Languages Lab, Vrije Universiteit Brussel, Belgium

Trace-based JIT Compilation

Program

```java
function f(a) {  
    var b = a > 0 ? 1 : 2;  
    return a + b;  
}
```

Execution Trace

```java
loop:  
    LOAD n  
    LOAD_CONST 1  
    CMP_GREATER  
    GUARD_TRUE  
    LOAD a  
    LOAD_CONST 0  
    CMP_GREATER  
    GUARD_TRUE  
    LOAD result  
    ADD_INT  
    LOAD n  
    DEC  
    JUMP loop
```

AOT Compilation

Imprecise static information

Compile-time analysis

Whole-program optimisation

My Approach

Hybrid:

Refine static information with observed information

Time

Run-time + Compile-time analysis

Scope

Local + whole-program analysis

Extended trace optimisations

Example

Initial + Run-Time analysis 1

No Analysis

- LookupVariable(n)
- PushVal()
- PushVal()
- PushVal()
- PushVal()
- PushVal()
- PushVal()
- PrimCall(4,-)
- ...

Initial + Run-Time analysis 1 & 2

- LookupVariable(n)
- PushVal()
- PushVal()
- PushVal()
- PushVal()
- PushVal()
- PushValue(1)
- PushValue(4)
- PushValue(1)
- PushValue(4)
- PushValue(1)
- PushValue(6)
- PushValue(4,-)
- ...

Initial Analysis

- LookupVariable(n)
- PushVal()
- PushValue(1)
- PushValue(4)
- PushValue(1)
- PushValue(4)
- PushValue(6)
- PushValue(4,-)
- ...

Future Work

1) Analysis Launch Point?
2) Extent of Scope?

Trace optimisations

Precise dynamic information

Run-time analysis

Local analysis

Trace-based JIT Compilation

Compile time

Run time

Initial Analysis

Compute heavyweight initial analysis

1st run-time analysis

Refine initial analysis with run-time information

2nd run-time analysis

Further refine run-time analysis with extra run-time information

Initial + Run-Time analysis

Eliminated

Extended

maarten.vandercammen@vub.ac.be

https://github.com/mvdcamme/scala-am