Trace Compilation

Christian Wimmer
cwimmer@uci.edu
www.christianwimmer.at
September 2009

Department of Computer Science
University of California, Irvine
Background

Institute for System Software
Johannes Kepler University Linz, Austria

Java HotSpot™ VM
- Linear Scan Register Allocation
- Automatic Object Inlining
- Array Bounds Check Elimination
- Optimization of Strings
- Tail Calls

Department of Computer Science
University of California, Irvine

Maxine VM
- Trace Based Compilation
- Phase Change Detection
- Hierarchical Layering of VMs

Feedback-Directed Optimistic Optimizations for Java
Credit: Arnold Schwaighofer, JKU Linz

Hard tail calls
- Prefix for invoke bytecodes (reuse “wide” prefix)
- Semantics checked by bytecode verifier

Prototype implementation for Java HotSpot VM
- Interpreter, Client Compiler, Server Compiler
- Problem: Fixed size of compiled frames, parameters part of caller
  - Not enough space for parameter of tail calls
  - Non-sibling tail calls require adapter frames

Protection domains
- Security information is attached to stack frames
- Problem: Tail calls across protection domains are security hole
  - Solution 1: Don’t allow, but throw exception
  - Solution 2: Perform normal call and compress stack later if needed
Commercial 2 – Client Compiler Visualizer

https://c1visualizer.dev.java.net/
Why Trace Compilation?

Method Execution Frequency

Block Execution Frequency

Execution Frequency Relative to Most Executed
- 100% (Red)
- 0.1% (Green)
- 5% (Blue)
- 0% (Black)

1 Iteration / 1 Operation of mpegaudio benchmark from SPECjvm2008
Single-block methods omitted
Most executed method: javazoom.jl.decoder.huffcodetab.huffman_decoder
20,406,507 loop iterations / 5,078,337 calls
Elements of a Trace Tree

- Anchor
- Trunk Trace
- Guard
- Branch Trace
- Guard
- Side Exit
Trace Compilation

- Interpret
- Interpret and Record Trace
- Compile Trace
- Execute Trace

- trace entry reached
- backward branch counter overflow
- abort tracing
- tracing finished (backward branch)
- side exit
Flavors of Trace Compilers

- Dynamic Languages
  - TraceMonkey: Mozilla’s JavaScript engine for Firefox
  - Elimination of dynamic type checks
  - Low compilation threshold
  - Fast compilation, no expensive optimizations (yet)

- Java – Maxine VM
  - Evolved from a research prototype (Hotpath VM)
  - Maxine has no interpreter – where to integrate trace recording?
  - Traces currently cover only loops, no merge points

- Java – HotSpot VM
  - Research funding requested, but still pending
  - Stay tuned…
Trace Recording in Interpreter

Normal Interpreter

Recording Interpreter

Start Recording

Stop Recording

Profiling Instrumentation

Recording Instrumentation
Trace Recording in Template JIT

Normal Method

Recording Method

Profiling Instrumentation

Recording Instrumentation
Evaluation Trace Compilation

Speedup relative to Maxine baseline compilation

Preliminary Numbers!

- Series B
- LUFact C
- HeapSort C
- Crypt C
- FFT C
- SOR C
- Matmult C

Maxine JIT
Maxine JIT + OPT
Maxine JIT + Tracing
HotSpot Server

2 x 2.8GHz Quad-Core Intel Xeon Mac Pro
Mac OS X 10.5.6, 64 bit JVMs
Problems and Future Work

- Loop-centric approach
  - Trace and compile only loops
  - Good for numerical applications
  - Loops are not the first-order element of object-oriented applications
  - Need to optimize also on per-method basis if necessary

- Tail duplication
  - Trace trees lead to duplication of join blocks
  - Block count can explode for complex control flow
  - Need to support join blocks in traces

- Trace calling convention
  - Switch to and from traces with low overhead
Phase Detection using Trace Compilation

Stable Phase ➞ Unstable Phase ➞ Stable Phase

Record Trace
Few Failed Guards

Increase of Failed Guards
Discard Trace

Record New Trace
Few Failed Guards
Phases Based on Control Flow

Example from Java Grande Series benchmark (simplified and adapted)
Phase Detection Structure

Unstable Phase

Interpret

- trace entry reached
- backward branch
- counter overflow
- abort tracing

Interpret and Record Trace

- tracing finished (backward branch)

Compile without Optimizations

- no stable phase
- side exit

Execute Trace

- side exit

Check for Stable Phase

Stable Phase

Interpret

- phase change (restart optimization)

Check for Phase Change

- no phase change
- side exit

Execute Trace

- trace entry reached

Compile with Optimizations

- stable phase
- (add new traces)
Phases Based on Data Flow

Example from Java Grande LUFact benchmark (simplified)
Possibilities of Trace Compilation

- Hierarchical layering of virtual machines

---

**Ruby Application**

- **Ruby Source Code**
  - Analyzes

**JRuby VM**

- **Ruby Parser**
  - Generates
- **Ruby Interpreter**
  - Interprets

**Abstract Syntax Tree**

- Generates
- Manipulates

**Ruby Objects**

---

**Java VM**

- **Java JIT Compiler**
  - Compiles via tracing

**Machine Code**

- Generates

**Java Objects**

- Reference

**Java Class Meta Data**

---

18
Possibilities of Trace Compilation

- Hierarchical layering of virtual machines – Current structure