A Block-Based Bytecode Format
to Simplify and Improve Just-in-Time Compilation

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

Department of Computer Science
University of California, Irvine
Software Engineering

Unstructured Bytecodes Method Granularity

JIT Compiler Design derived from Static Compilers

Method Structure used in Compiler because “it’s there”

Compiler Optimization
Block Granularity

Software Engineering

Block-Based Bytecodes with Structure Information

Novel JIT Compilation Strategies using Profiling Data

Compilation Units based on Execution Frequency

Compiler Optimization
Compiler Research

- Profiling-Based Method Inlining
- Partial Method Inlining
- Region-Based Compilation
- On-Stack Replacement
- Deoptimization
- Trace Compilation
Main Idea and Challenges

Definition of a Block-Based Bytecode Format

Execution in Virtual Machine
- Efficient Interpretation
- Simple Compilation
- Profile Information
- Trace Compilation

Source Languages
- For Static and Dynamic Languages
- Functional Languages
- Dynamic Method Calls
- Closures, Tail Calls, …

Design Decisions
- High-Level Structure, e.g., Loops
- “Calling Convention” of Blocks
- Stack Based vs. Register Based
- Static Single Assignment Form?

Safety and Security
- Easy Verification
- Inherently Safe Code Formats
SPECjvm2008 – mpegaudio

Method Execution Frequency

Block Execution Frequency

Execution Frequency
Relative to Most Executed

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
Summary

Software Engineering
- Method Structuring
  - Readability
  - Extendibility
  - Reusability
  - ...

Compiler Optimization
- Block Structuring
  - Execution Frequency
  - Feedback-Directed Optimization
  - Profiling
  - ...

Methods hide the real focus where compiler optimizations should be applied