

Design of Virtual Machines for Object-Oriented Languages

University of Aarhus, Fall 2008

Goals

- Having completed this course you should be able to:
 - Describe the design ideas of a selection of modern VMs
 - Implement core components of a virtual machine
 - Evaluate VM technology for efficiency and simplicity

Lecturer: Kasper Lund

- Candidate from Aarhus University, 2003
- Eight years of experience in building high-performance VMs
 - 2000-2002: CLDC HI (Sun Microsystems)
 - 2002-2006: Resilient (OOVM)
 - 2006-2008: ? (Google)
- Primary areas of interests:
 - Dynamic programming languages
 - Interpretation and dynamic code generation
 - Method dispatching

Plan for the course

- Week 35: Motivation and presentation of a Java VM
 - CLDC HI: High performance Java VM for mobile devices
- Week 36: Byte codes and interpreters
 - Resilient: An efficient, interpreted Smalltalk VM
- Week 37: Automatic memory management
- Week 38: Dynamic compilation
- Week 39: Type feedback and aggressive function inlining
- Week 40: Threading and locking
- Week 41: Programming tools and conclusions

Exercises

- Mandatory weekly exercises
 - Read articles
 - Short written report - a few pages
- Form groups of 2-3 persons
 - One written report per group per week
 - Hand in Tuesdays to Mathias (schwarz@daimi.au.dk)

What is a virtual machine?

- High-level runtime environment
 - Memory management
 - Execution engine
 - Language support
- Usually executes non-native instructions
 - Byte codes or virtual instructions
 - Instruction level debugging?

Object model

- Representation of objects in memory:
 - How are object fields represented?
 - Where are methods and functions stored?
 - String representations: UTF-8, UTF-16, ASCII
- Also contains reflective information:
 - Object types: strings, arrays, class instances, ...
 - Sizes of objects, location of pointers

Object model (2)

- Programming language may need support for:
 - Method dispatching (virtual method calls)
 - Static data - class instance variables
 - Identity hash codes
 - Synchronization
 - ...

Execution model

- Byte codes or syntax trees
- Interpretation or compilation
 - Speed / complexity trade-off
- Native threading or light-weight alternative

Project Monty

- High-performance Java VM for mobile devices
- Developed at Sun Microsystems from 2000 to 2002
 - Part of the team working from Aarhus
- Shipped as CLDC HI
 - CLDC: Connected Limited Device Configuration
 - HI: HotSpot Implementation

Exercises for next week

- Read the articles
 - Threaded Code
 - The Structure and Performance of Efficient Interpreters
 - Background reading: The Smalltalk-80 System
- Implement a threaded interpreter in GNU C
 - Measure performance impact of threading
 - Compare compiled code: switch ~ threaded
- Hand in written "report" - a few pages containing
 - Source code
 - Results of performance measurements
 - Compiled code for the interpreter variants (objdump -d)

Interpreter (slow)

```
static int Interpret(const byte* opcodes) {
    int result = 0;
    while (1) {
        switch (*opcodes++) {
            case 0: return result;
            case 1: result++; break;
            case 2: result--; break;
            case 3: result <<= 1; break;
            case 4: result = (result << 16) | (result >> 16); break;
        }
    }
}
```