TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für technische Informatik, Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur

Implementierung der Jikes Research Virtual Machine

Hauptseminar Technische Informatik – 9. Juli 2008

Stefan Alex
s2174321@inf.tu-dresden.de

Gliederung

1. Einleitung
2. Initialisierung der Jikes RVM
3. Design und Implementierung
4. Compiler
5. Benchmarks, Ausblick
6. Quellen

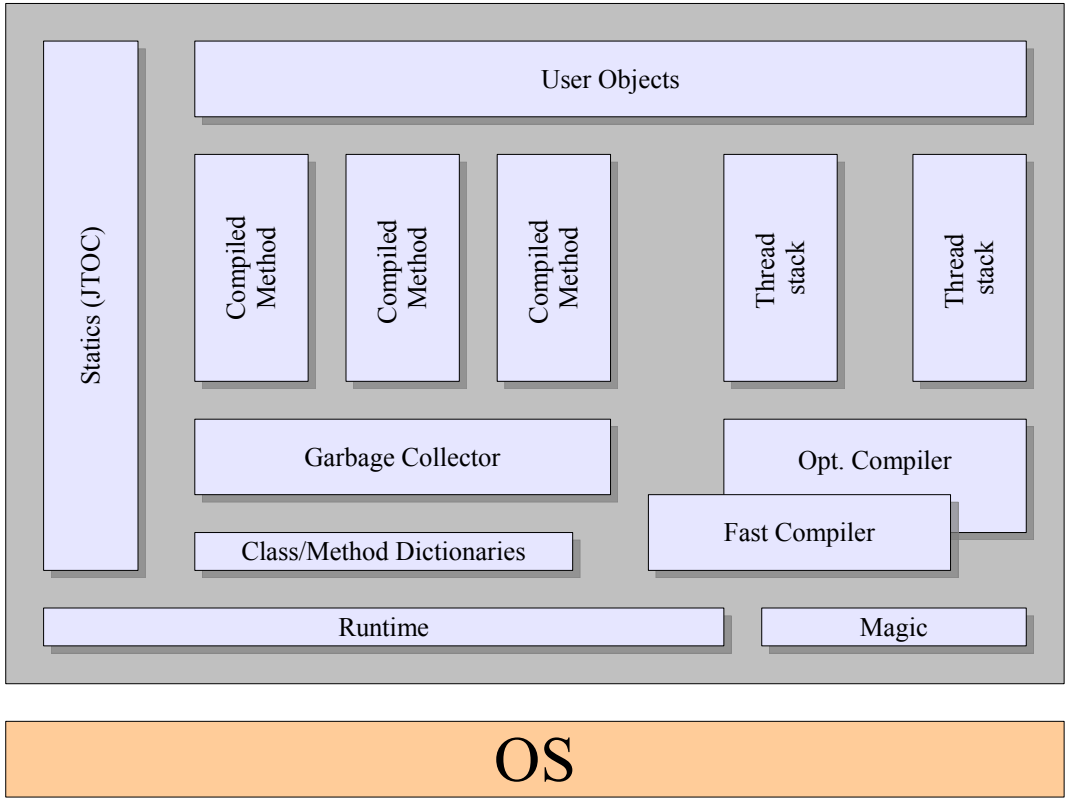
1. Einleitung

- Java Virtual Machine
- fast komplett in Java geschrieben
- Ausführung direkt auf OS ohne zweite VM
- Projektbeginn November 1997 an IBM's T.J. Watson Research Center als Jalapeño VM
- OpenSource seit Oktober 2001
- verfügbar Linux/IA-32, AIX/PowerPC, OSX/PowerPC, Linux/PowerPC

1. Einleitung

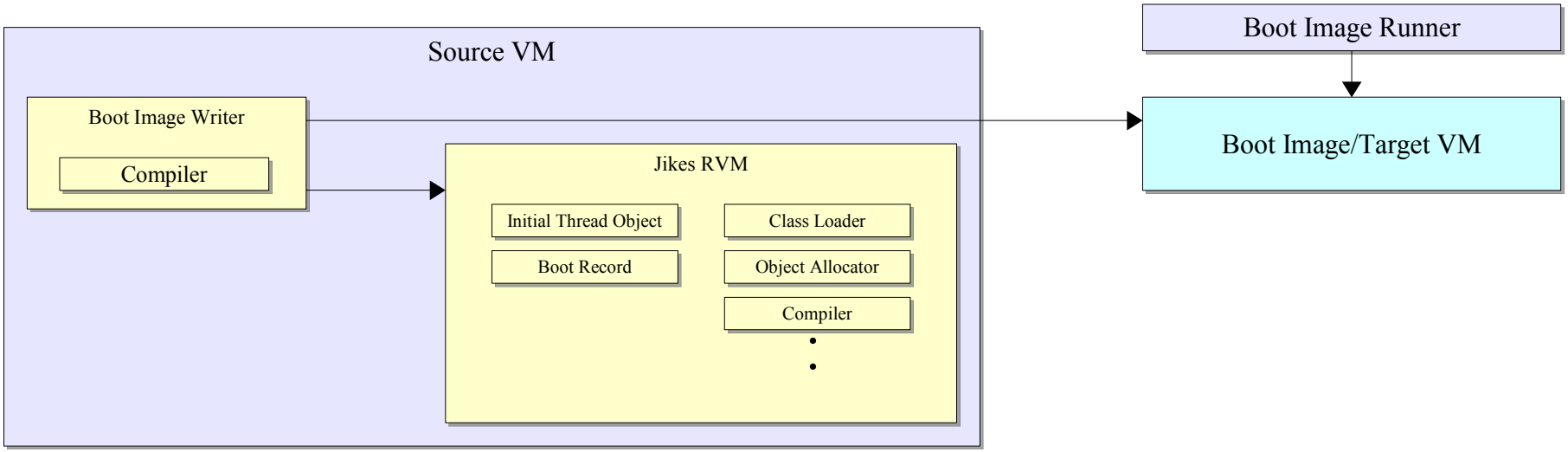
- ursprünglich für den Server-Einsatz vorgesehen
- mittlerweile Testumgebung für VM-Research
- von über 80 Universitäten genutzt
- 188 Publikationen
- 36 Dissertationen basierend auf Jikes

2. Initialisierung der Jikes RVM
Hauptkomponenten



2. Initialisierung der Jikes RVM

- Instanziieren von Jikes auf Target Virtual Machine
- Erstellen eines ausführbaren Boot Images mit Basisdiensten der VM
- Initial Thread Object, Bootrecord, Object Allocator, Class Loader, Compiler, etc.



3. Design und Implementierung
Run-time subsystem

- Bereitstellung von Basisdiensten
  - Dynamic Type Checking
  - Dynamic Class Loading
  - Exception-Handling
  - Input/Output-System
  - Reflection
  - ...

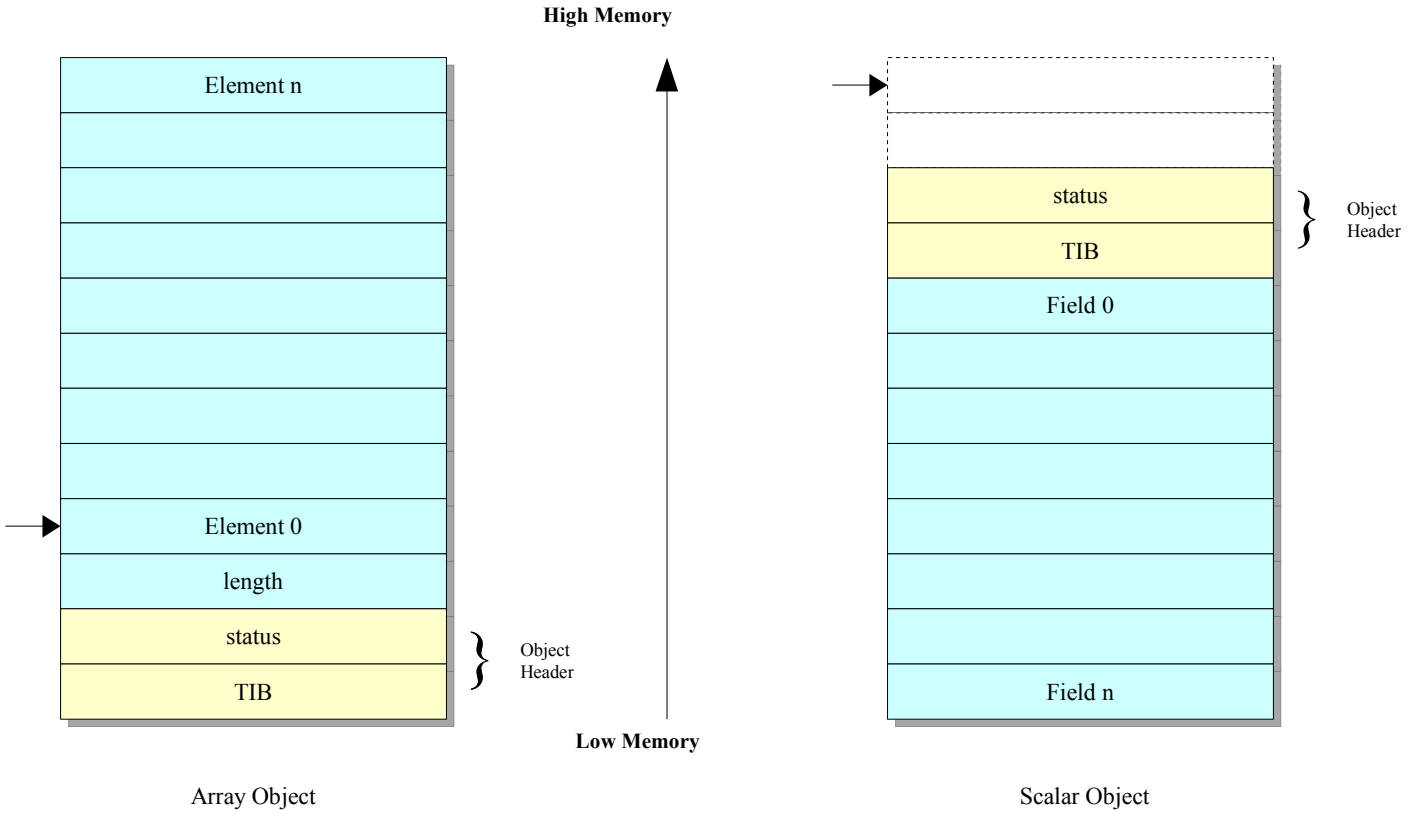
3. Design und Implementierung
Low-Level-Funktionalität

- „VM_Magic"-Klasse
  - Umgehen java-spezifischer Einschränkungen
  - Verwendung von OS-Services
  - Einfluss auf Kontrollfluß
  - Aufruf spezieller Maschineninstruktionen
- JNI 1.4
  - Java Native Interface
  - Zugriff auf „native" Funktionen aus Java-Programm
  - Zugriff auf Java-Funktionen aus „nativen" Programm

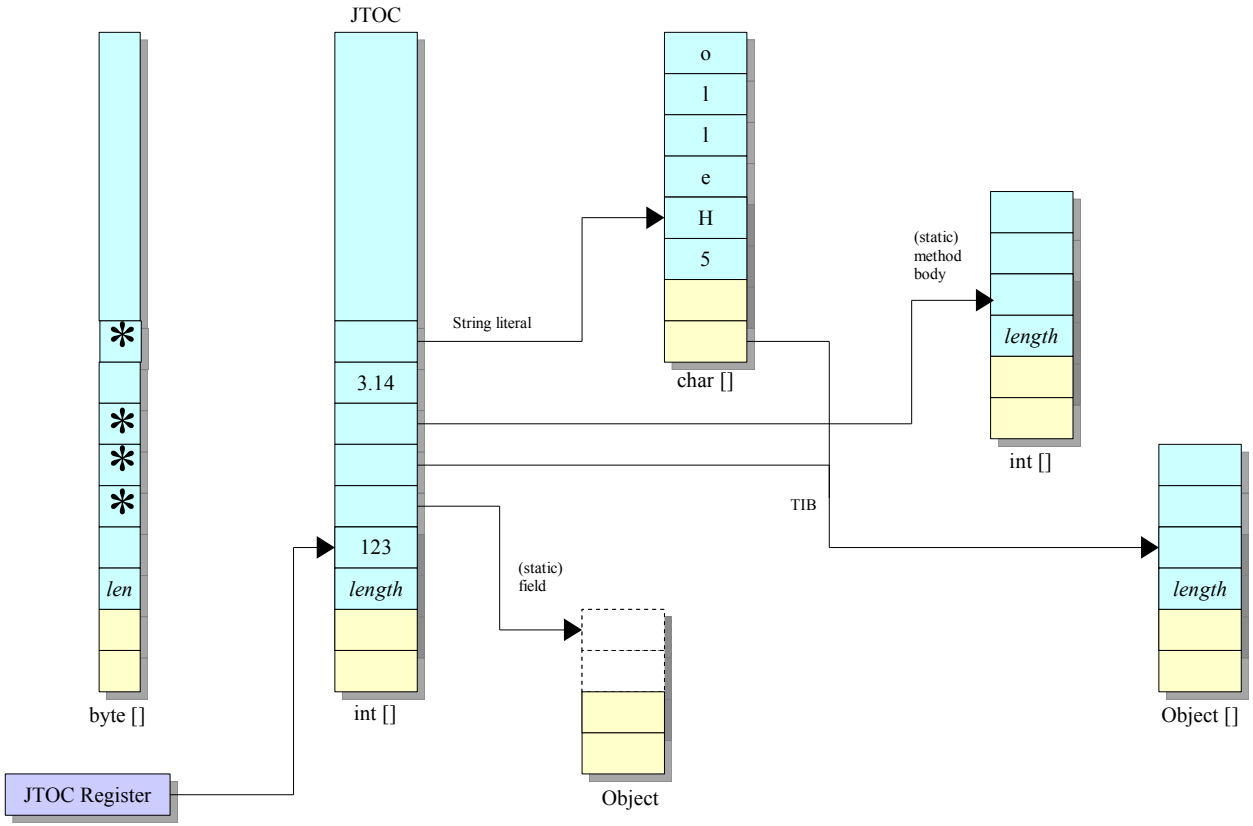
3. Design und Implementierung
Object-Model

- Zielsetzung:
  - schneller Zugriff Arrays/Felder
  - schneller Methodenaufruf
  - Hardware-NullPointer-Check

3. Design und Implementierung
Object-Model



3. Design und Implementierung
Jikes Table Of Content (JTOC)



3. Design und Implementierung
Threads und Synchronisation

- Threads auf „Virtuelle Prozessoren“ verteilt
  - N:M-Thread-Model
  - Unabhängigkeit vom Betriebssystem, Verwendung eigener Implementierungen
  - i.d.R. ein Virtueller Prozessor pro CPU
  - Load Balancing zwischen Prozessoren
- quasi-preemptives Scheduling

3. Design und Implementierung
Memory Management

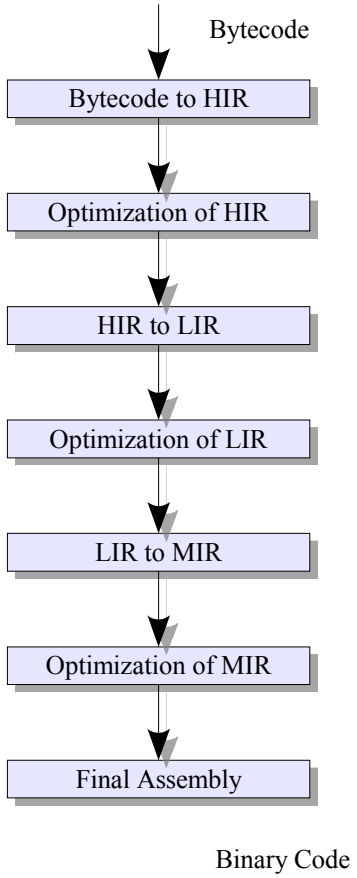
- MMTk: Memory Management Toolkit
  - Toolkit zum Erstellen von Memory Managers
- nebenläufiger Object-Allocator
  - pro virt. Prozessor: lokaler freier Speicher
  - Erreichbar von allen Prozessoren
  - nebenläufige Allocation ohne Locking/Synchronisation
- Stop-the-World-Garbage Collection
- GC-Thread pro Virtuellen Prozessor
  - Anhalten aller laufenden Threads
  - Synchronisation der einzelnen GC-Threads
- keine Unterstützung von nebenläufiger GC

4. Compiler

- Baseline Compiler
  - direkte Übersetzung von Bytecode in Maschinencode
  - Simulation des Java-Stacks
  - keine Optimierungen
- Optimizing Compiler
- (JNI Compiler)

4. Compiler
Optimizing Compiler

- mehrstufige Optimierung
- verschiedene Zwischenrepräsentationen
- High-Level-Intermedia-Representation (HIR)
  - registerbasierte Sprache
  - High-Level-Analyse
- Low-Level-Intermedia-Representation (LIR)
  - Anpassung an Jikes RVM (Object Layout)
- Machine-Specific-Intermedia-Representation (MIR)
- Machine Code
  - einschließlich GC-Maps, Exception Tables, Machine Code Informations, Debugging-Information, etc.



4. Compiler
Optimizing Compiler

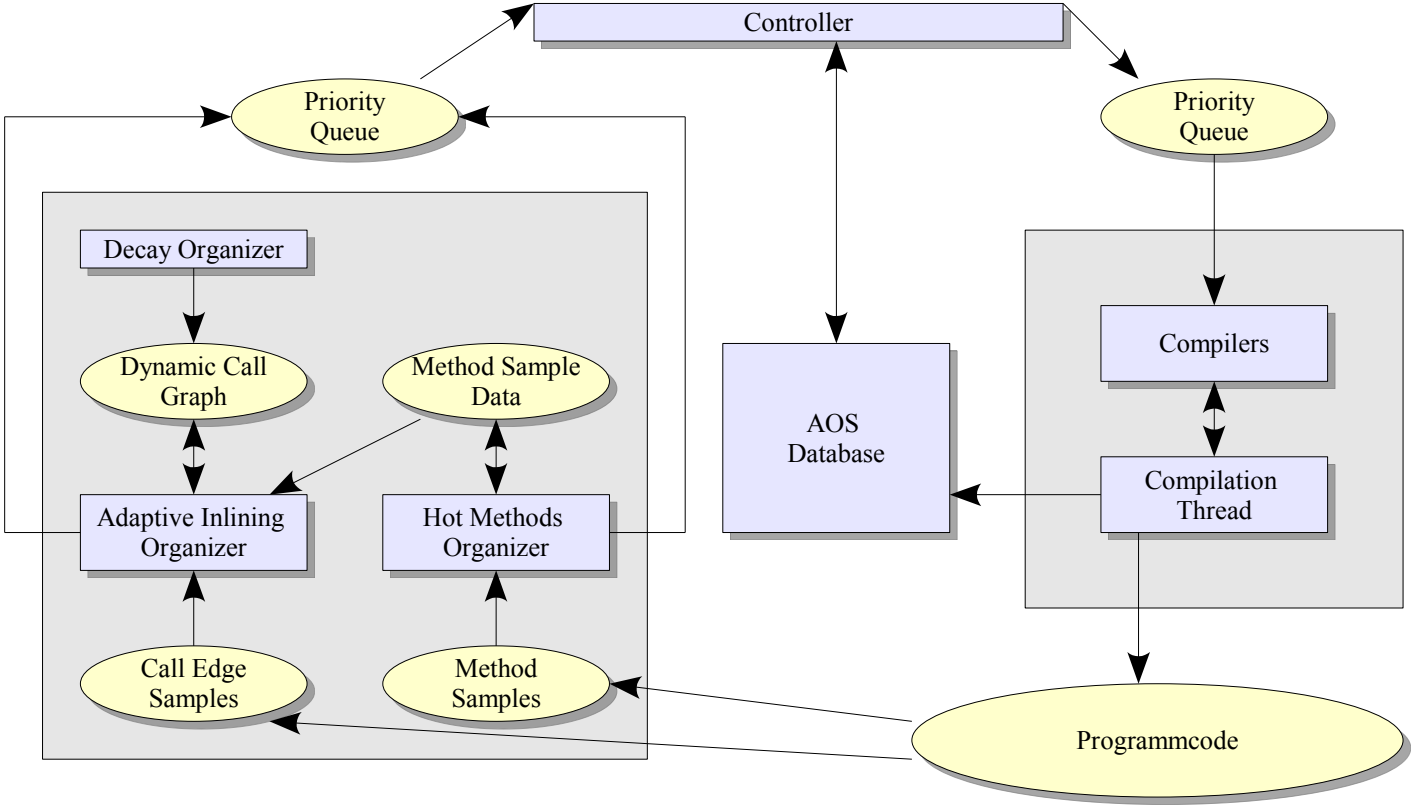
- Mehrere Optimierungslevel
- Level 0 on-the-fly-optimization
  - constant-folding, copy-propagation, unrechable code-elimination, inlining trivialer Methoden, Kontrollflußoptimierungen etc.
- Level 1 Multi-Pass-Optimierungen
  - Verwendung von Profiling-Daten
- Level 2 globale Optimierungen
  - global value numbering, global common subexpression elimination, etc.

Compiler	Bytecode Bytes/Millisecond	Speed
Baseline	377.76	1.0
Opt. Level 0	9.29	4.26
Opt. Level 1	5.69	6.07
Opt. Level 2	1.81	6.61

4. Compiler
Adaptive Optimization System

- Recompilierung verschiedener Methoden
- Sammeln von Profiling Daten
- Identifizerung von Hot Methods, Hot Edges
  - oft aufgerufene Methoden
  - gleicher Aufrufer für Methoden

4. Compiler
Adaptive Optimization System



5. Benchmarks/Ausblick

- Vergleich zw. Jikes Baseline-Compiler/Optimizing-Compiler und JDK mit/ohne Just-in-Time-Compiler

Test	JDK w/o JIT	Jikes Baseline	JDK w/JIT	Jikes Optimizer
Bsort	77.19	34.26	3.20	3.94
Qsort	15.27	6.10	1.11	0.78
Sieve	11.47	4.74	0.34	0.42
Hanoi	17.84	7.90	1.00	1.54
Fibbo.	20.23	11.58	1.75	0.98
Drystone	7.12	2.33	0.65	0.68
Array	4.95	10.15	1.01	0.84

Execution times (seconds)

5. Benchmarks/Ausblick

- Weiterentwicklung durch IBM Research
- weitere Kooperationen mit Universitäten
- zukünftige Forschungsfelder:
  - Dynamic code optimization
  - Virtual machine technologies for web services and grid computing
  - Extensions for real-time processing
  - Integration in das Eclipse framework
  - …

6. Quellen

- Jikes RVM http://jikesrvm.org/
- Implementing Jalapeño in Java - http://cs.anu.edu.au/~Steve.Blackburn/teaching/comp4700/resources/papers/jalapeno-oopsla-1999.pdf
- The Jalapeño Virtual Machine - http://www.research.ibm.com/journal/sj/391/alpern.pdf
- MMTk: The Memory Manager Toolkit - http://cs.anu.edu.au/~Robin.Garner/mmtk-guide.pdf
- Adaptive Optimization System – Nov. 2004 – Technical Report - http://domino.research.ibm.com/comm/research_people.nsf/pages/dgrove.RC23429.html

Fragen ???