TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut fÜr Technische Informatik

ENTWICKLUNG UND CHARAKTERISIERUNG VON JAVA-BENCHMARKS

Hauptseminar

Bastian Buder

Dresden, 23.6.2010

Gliederung

Einführung

Java Benchmarks
- SPEC Benchmarks
- DaCapo Benchmark Suite

Einflussfaktoren / Vorgehensweisen

Metriken
- Statische Metriken
- Dynamische Metriken

01 Einführung

Leistungsbewertung via Benchmarks

- Benchmarking (”Maßstäbe setzen”): vergleichende Analyse mit Referenzwert
- Treiben Forschung und Entwicklung
- Welcher Benchmark repräsentiert am besten die realen Anforderungen?

Hardware - Benchmarks

- Vergleich von CPU- und Speicherzugriffsleistung
- MIPS / FLOPS, Dhrystones / Whetstones, SPEC CPU2006

Software - Benchmarks

- Vergleich verschiedener Programmiersprachen
- Implementierung eines Algorithmus in verschiedenen Sprachen

01 Einführung
Java Benchmarks

Vergleich verschiedener...

- Java Virtual Machine (JVM)
- Hardware (CPU, Cache, Speicher, ...)

Bewertung der Effizienz von...

- Garbage-Collection - Algorithmen
- JIT (just-in-time) - Compilern
- dynamischen Optimierungen
- u.v.m.

01 Einführung

Bewertung von Benchmarks

Bestandteile bislang nach gleichen Kriterien wie für C, C++, Fortran ausgewählt

- Entwicklung der Methoden hielt nicht Schritt mit den "Managed languages" und Multicore - Architekturen
- Höhere Anforderungen an Design von Benchmark-Suites
- Bewertung und Auswahl geeigneter Applikationen nach neuen Kriterien erforderlich
- Falsche Vorgehensweisen können gute Entwicklungen schlecht aussehen lassen und umgekehrt
- Grundlage zur Bewertung von Innovationen

01 Einführung

Schwierigkeiten bei Java Benchmarks

Komplexe Laufzeitabhängikeiten zwischen

- JVM
- Architektur
- just-in-time Compiler (JIT) / Optimizer
- Speicherverwaltung / Garbage-Collector / Heap-Größe
- Applikation

Welchen Benchmark sollte man verwenden?
Nach welchen Metriken lassen sich Benchmarks bewerten?

02 Übersicht
Typische Java Benchmarks

Java Grande	wissenschaftliche Probleme	frei
JOlden Benchmarks	adaptierte C-Benchmarks	frei
Ashes Benchmarks	versch. Compiler	frei
SPECjAppServer2004	JEE 1.3	$2000
SPECjEnterprise2010	JEE 5	$2000
SPECjms2007	Java Message Service	$1800
SPECjbb2000 / SPECjbb2005	Server JVM Benchmark	$500
SPECjvm98 / SPECjvm2008	Client JVM Benchmark	$100 / frei
Colorado Bench 2002	JVM Benchmark	frei
DaCapo Benchmark Suite	JVM Benchmark	frei

Weitere spezialisierte Benchmarks verfügbar (J2ME, Linpack, ...)

02 SPEC JVM Benchmarks
Probleme der SPECjbb2000

- Score nach fester Laufzeit anstatt nach verstrichener Zeit für feste "Aufgabe"
- Durchsatz ist EIN wichtiges Kriterium, die meisten Applikationen aber nicht durchsatzbasiert
- Schleife beeinflusst JIT-Optimierungen
- Unterschiedliche Anzahl (re-)optimierter Klassen
- Unterschiedliche Anzahl notwendiger GC-Läufe

- teilweise modifizierte Version mit festem Durchläufen verwendet (pseudoJbb)

02 SPEC JVM Benchmarks
Probleme der SPECjvm98

- 7 Applikationen, mangelnde Komplexität
- beliebig wiederholte Ausführung des Benchmarks in einer JVM, nur Report der besten und schlechtesten Zeit
- Beziehung zwischen Ausführungszeit und Kompilierzeit nicht berücksichtigt
- keine Ergebnisse für unterschiedliche Heap-Größen: Aufwand/Effizienz der Garbage-Collection bleibt außen vor
- definiert 3 feste Heap-Größen: 48, 48-256, $>$ 256MB, alle recht groß
  - allokieren 1 - 271MB
  - max. 8MB lebendig im HEAP
  - Ausnahme: pseudojbb: 21MB lebendig

02 DaCapo - Benchmarks

Grundlagen

Beginn: Mitte 2003
Ziel: Umgebung zum Messen / Auswerten, Auswahl von Benchmarks
ca. 10.000 Personenstunden
Release: 08/2006

- Einführung neuer stat. Metriken für dyn. und statische Eigenschaften
- Menge von realitätsnahen, verbreiteten Java-Applikationen
- einfach nutzbar / messbar
- Empfehlung von Methoden zur Evaluierung Java Benchmarks, JVMs und ihrer Speicherverwaltung
- Abdeckung möglichst aller relevanten Typen von Programmen

02 DaCapo - Benchmarks

Zusammensetzung

1. Open-Source-Software
2. aus verschiedenen Anwendungsdomänen, mit unterschiedlichem Verhalten
3. Fokus auf Client-Side Benchmarks und Server-Side Apps mit minimalen Abhängigkeiten außerhalb der Host-VM
4. Ausschluss von GUI-Apps (Eclipse: Teilmenge ohne GUI)
5. ausreichend Eingabedaten für ausführliche Tests

Auswahl offen und transparent um Feedback von der Community zu bekommen!
(z.B. auch 3 Beta-Versionen veröffentlicht)

Einzelnes JAR-File, inkl. aller Bibliotheken
Prüfung auf korrekte Ausführung des Benchmarks

02 DaCapo - Benchmarks

Aktuelle Applikationen

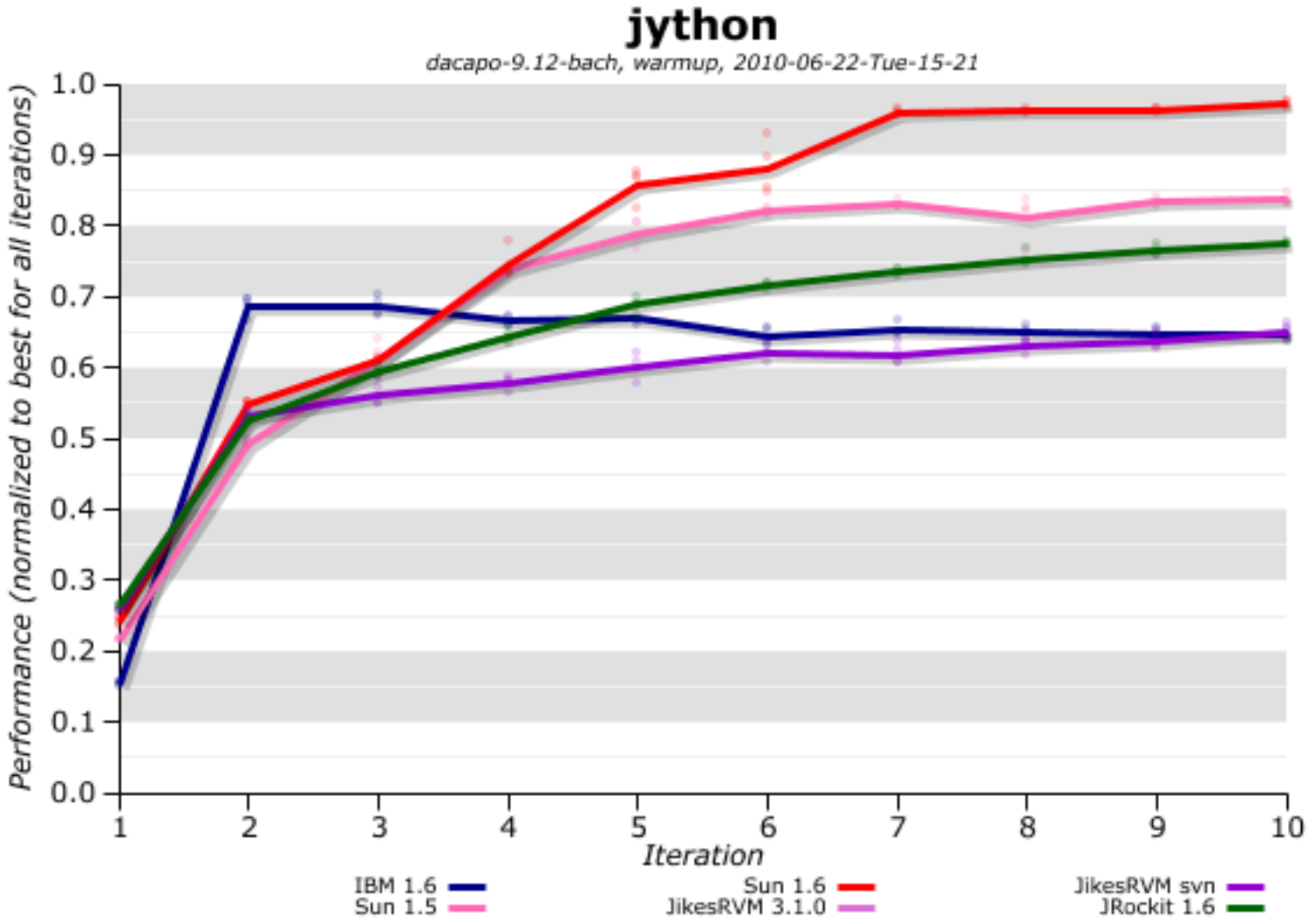
avrora	simulates a number of programs run on a grid of microcontrollers
batik	produces a number of Scalable Vector Graphics (SVG) images
eclipse	executes some of the jdt performance tests for the Eclipse IDE
fop	takes an XSL-FO file, parses it and formats it, generating a PDF
h2	executes a JDBCbench
jython	inteprets a the pybench Python benchmark
luindex	Uses lucene to indexes a set of documents
lusearch	Uses lucene to do a text search of keywords
pmd	analyzes a set of Java classes for a range of code problems
sunflow	renders a set of images using ray tracing
tomcat	runs a set of queries against a Tomcat server
tradebeans	runs the daytrader benchmark via a Jave Beans
tradesoap	runs the daytrader benchmark via a SOAP
xalan	transforms XML documents into HTML

03 Vorgehensweisen

Einfluss der JVM

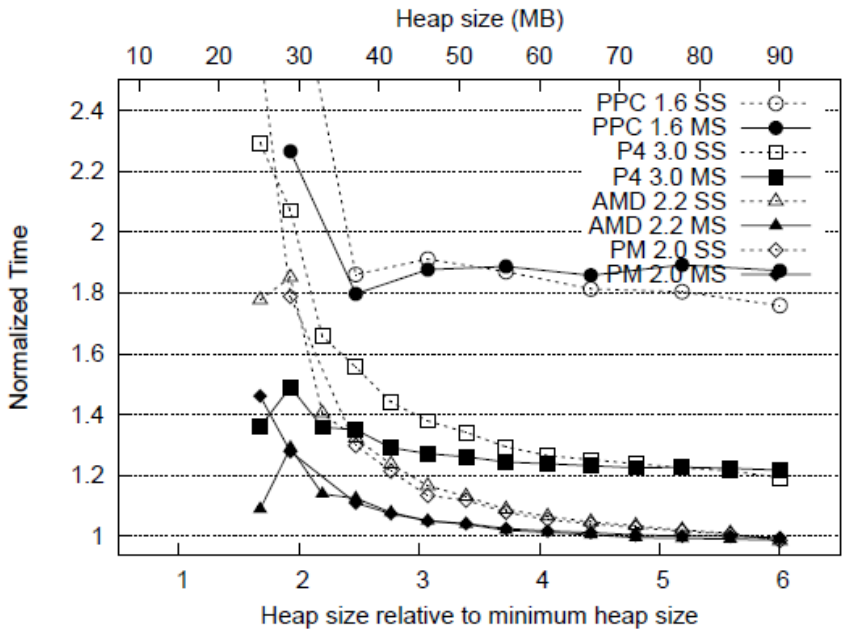
- Mix: Applikation, Compiler, Memory-Management
  - entspricht i.d.R. SPEC ”worst”
- Stable: Applikation, Memory-Management
  - nach Warm-Up
  - entspricht i.d.R. SPEC ”best”
  - finale Code-Qualität nicht garantiert
- Deterministic Stable & Mix:
  - eliminiert ”willkürliche” Kompilierung von Methoden
  - fester Kompilations-Plan (replay compilation)
  - hochoptimierte ”Hot-Methods” und normal übersetzter Code

	First iteration				Second iteration					Third iteration				
Benchmark	A	B/A	C/A	*Best*	A	B/A	C/A	*Best*	$2^{nd}/1^{st}$	A	B/A	C/A	*Best*	$3^{rd}/1^{st}$
SPEC														
_201_compress	7.2	0.76	**0.75**	*0.75*	**3.3**	1.51	1.65	*1.00*	0.75	**3.2**	1.02	1.69	*1.00*	0.65
_202_jess	3.2	0.80	**0.58**	*0.58*	2.0	1.27	**0.81**	*0.81*	0.82	2.0	**0.63**	0.79	*0.63*	0.65
_205_raytrace	2.6	0.86	**0.54**	*0.54*	1.5	0.99	**0.82**	*0.82*	0.66	1.5	**0.64**	0.82	*0.64*	0.58
_209_db	**8.0**	1.07	1.00	*1.00*	**6.9**	1.14	1.14	*1.00*	0.91	**6.8**	1.05	1.14	*1.00*	0.88
_213_javac	6.0	**0.48**	0.69	*0.48*	2.5	**0.93**	1.34	*0.93*	0.62	**2.2**	1.06	1.50	*1.00*	0.60
_222_mpegaudio	**3.8**	1.73	1.22	*1.00*	**2.8**	1.11	1.65	*1.00*	0.69	**2.7**	1.10	1.64	*1.00*	0.67
_227_mtrt	2.9	0.72	**0.51**	*0.51*	1.4	1.53	**0.94**	*0.94*	0.74	1.5	**0.61**	0.81	*0.61*	0.56
_228_jack	5.7	1.08	**0.61**	*0.61*	**3.1**	1.27	1.05	*1.00*	0.66	**3.0**	1.21	1.08	*1.00*	0.64
geomean		**0.94**	**0.74**	***0.68***		**1.22**	**1.18**	***0.94***	**0.73**		**0.91**	**1.18**	***0.86***	**0.65**
DaCapo														
antlr	6.0	**0.53**	0.68	*0.53*	3.4	**0.69**	0.94	*0.69*	0.66	3.2	**0.69**	0.97	*0.69*	0.64
bloat	12.0	**0.98**	1.03	*0.98*	**9.7**	1.28	1.20	*1.00*	0.93	**9.1**	1.24	1.28	*1.00*	0.88
chart	12.2	**0.97**	1.47	*0.97*	**9.5**	1.30	1.68	*1.00*	0.90	9.2	**0.73**	1.71	*0.73*	0.75
eclipse	61.7	1.28	**0.96**	*0.96*	**39.4**	1.60	1.17	*1.00*	0.74	**23.8**	1.60	1.94	*1.00*	0.54
fop	7.1	**0.40**	0.40	*0.40*	4.8	**0.33**	0.36	*0.33*	0.63	5.1	**0.31**	0.35	*0.31*	0.66
hsqldb	12.0	0.82	**0.47**	*0.47*	7.7	0.67	**0.66**	*0.66*	0.66	7.3	0.86	**0.68**	*0.68*	0.67
luindex	15.5	1.04	**0.94**	*0.94*	**9.8**	1.41	1.41	*1.00*	0.80	**9.1**	1.55	1.49	*1.00*	0.79
lusearch	13.1	**0.74**	0.90	*0.74*	10.6	**0.92**	1.06	*0.92*	0.91	**10.5**	1.56	1.07	*1.00*	1.10
jython	16.5	**0.52**	0.68	*0.52*	8.3	**0.92**	2.87	*0.92*	1.09	7.9	0.92	**0.83**	*0.83*	0.59
pmd	10.4	1.04	**0.95**	*0.95*	**7.5**	1.50	1.15	*1.00*	0.89	**6.9**	1.18	1.27	*1.00*	0.76
xalan	8.3	**0.87**	0.90	*0.87*	**5.3**	1.53	1.20	*1.00*	0.86	**5.0**	1.24	1.26	*1.00*	0.76
geomean		**0.84**	**0.85**	***0.76***		**1.10**	**1.25**	***0.86***	**0.83**		**1.08**	**1.17**	***0.84***	**0.74**

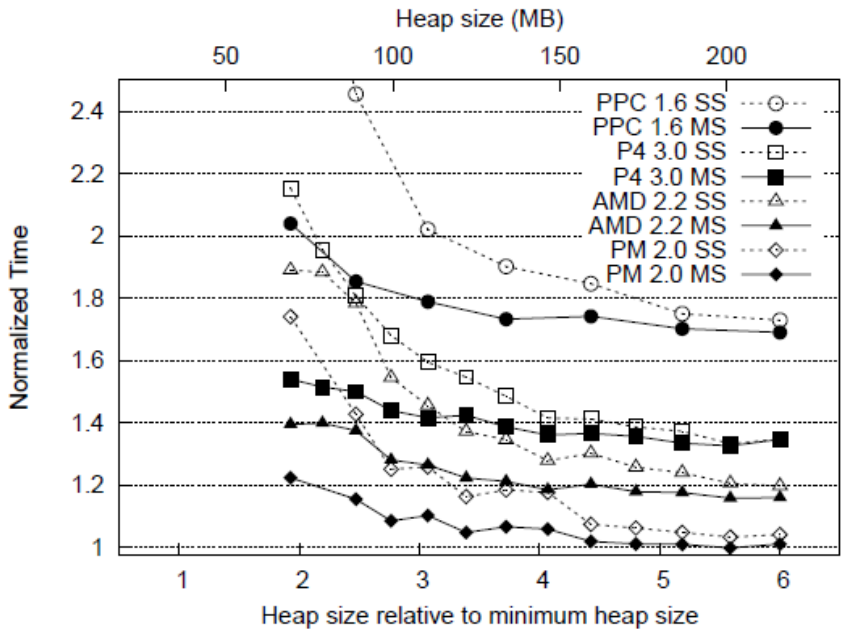

jython
dacapo-9.12-bach, warmup, 2010-06-22-Tue-15-21
Performance (normalized to best for all iterations)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
1
2
3
4
5
6
7
8
9
10
Iteration
IBM 1.6
Sun 1.5
Sun 1.6
JikesRVM 3.1.0
JikesRVM svn
JRockit 1.6

03 Vorgehensweisen

Einfluss von Benchmark und Architektur



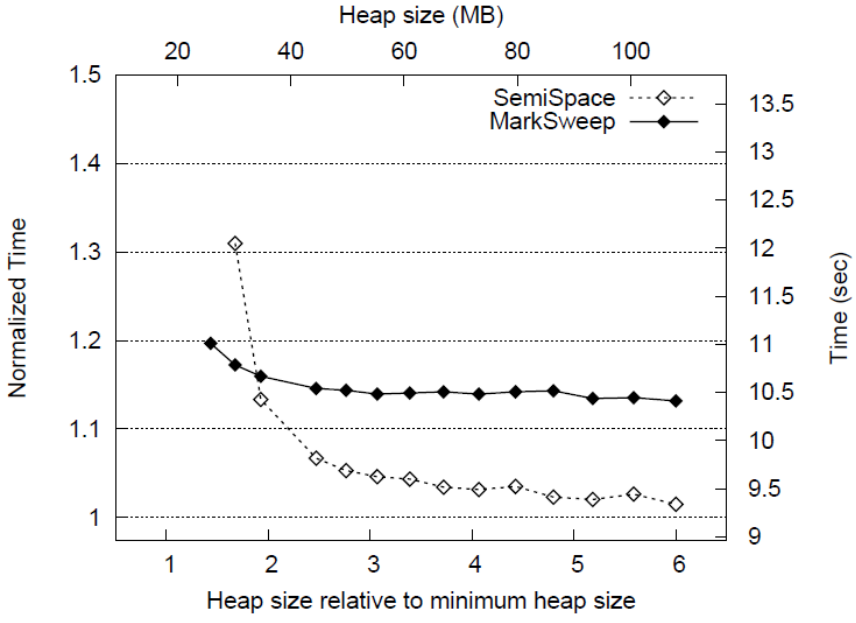
(a) SPEC



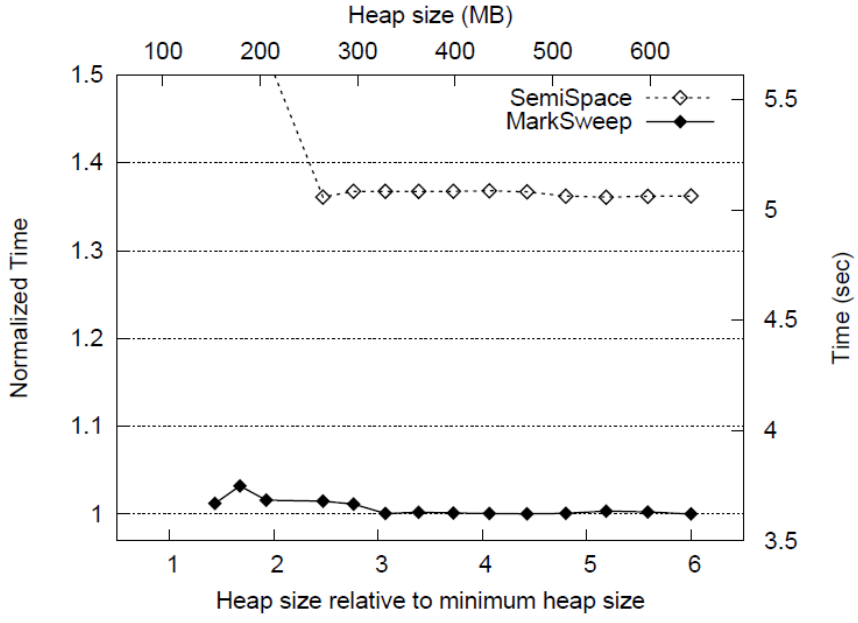
(b) DaCapo

03 Vorgehensweisen

Unterschiedliches Verhalten von Applikationen



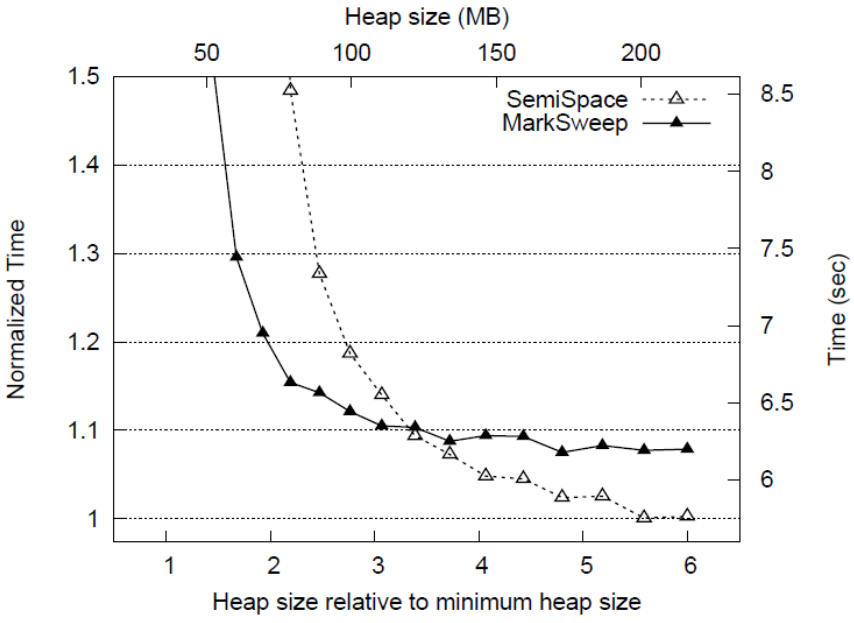
(a) *_209_db*, Pentium-M



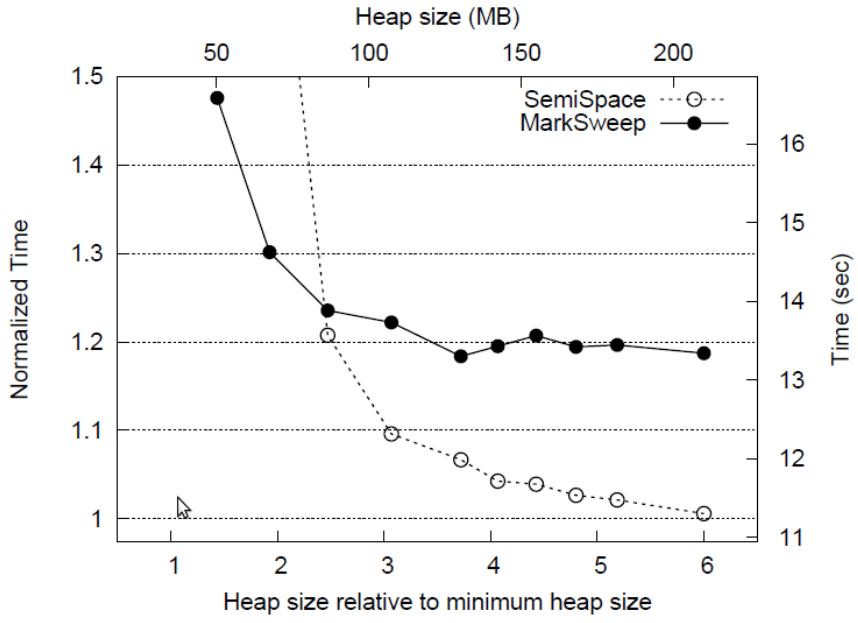
(b) *hsqldb*, Pentium-M

03 Vorgehensweisen

Einfluss von Architekturen / HEAP - Größen



(c) *pseudojbb*, AMD



(d) *pseudojbb*, PPC

04 Charakterisierung

Unterscheidbarkeit durch Metriken

Große Bandbreite an Metriken herausgearbeitet
Unterscheidung zwischen statischen und dynamischen Metriken

"Klassische" Metriken

- Code-Komplexität (OOP)
- HEAP-Graphen (Lebenszeit - Eigenschaften)
- Instruktionen - Mix

"Neue Metriken"

- Allokations - Rate
- Verhältnis allokierte / lebende Objekte
- HEAP Manipulationen

Anpassung von Jikes RVM (open source, relativ gute Performance) für Analysen

04 Statische Metriken
Code-Komplexität

- JVM unabhängig
- Architektur unabhängig
- SPEC bedingt "objektorientiert"
- DaCapo: intensivere Nutzung von OOP um Komplexität zu bewältigen

Benchmark	WMC	DIT	NOC	CBO	RFC	LCOM
SPEC						
_201_compress	154	19	0	55	426	780
_202_jess	614	97	1	632	1846	2032
_205_raytrace	330	33	3	117	743	1046
_209_db	152	12	0	42	454	789
_213_javac	1011	186	38	1175	3293	3753
_222_mpegaudio	367	40	0	167	796	1350
_227_mtrt	332	33	3	117	743	1046
_228_jack	375	46	0	163	860	6911
pseudojbb	541	35	0	254	1419	2487
min	152	12	0	42	426	780
max	1011	186	38	1175	3293	6911
geomean	**366**	**40**	**2**	**176**	**950**	**1701**
DaCapo						
antlr	1253	84	8	674	3094	8444
bloat	2265	206	21	1661	6232	6521
chart	1627	101	16	648	3979	29169
eclipse	10763	830	164	7277	26209	218199
fop	1433	148	17	998	3867	13041
hsqldb	2419	73	3	766	4676	47371
jython	3023	225	60	1398	5725	97111
luindex	494	50	0	246	1372	2260
lusearch	618	55	0	297	1441	3419
pmd	2348	215	4	1199	5384	126893
xalan	2433	161	24	971	5682	37394
min	494	50	0	246	1372	2260
max	10763	830	164	7277	26209	218199
geomean	**1857**	**138**	**10**	**935**	**4420**	**22561**

WMC	Weighted methods/class	CBO	Object class coupling
DIT	Depth Inheritance Tree	RFC	Response for a Class
NOC	Number of Children	LCOM	Lack of method cohesion

04 Dynamische Metriken

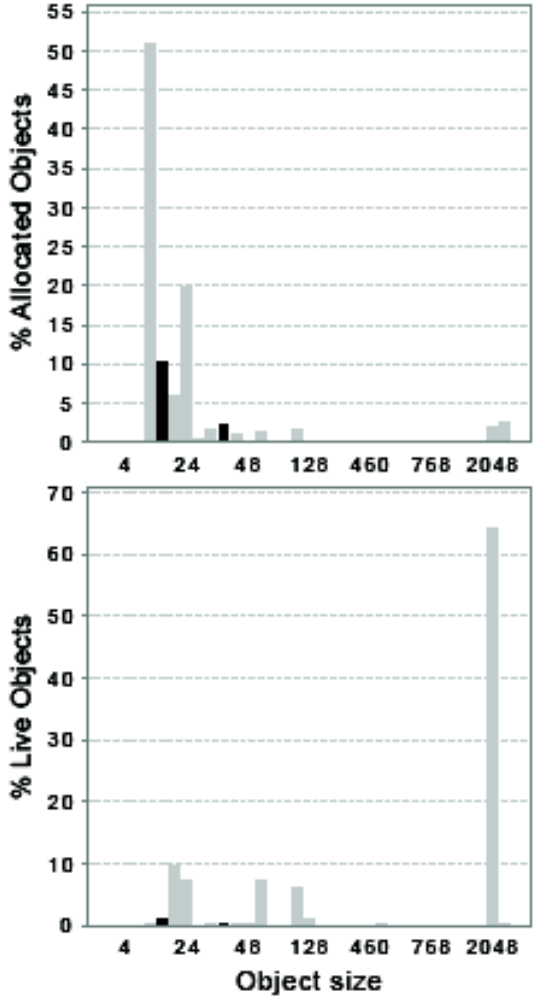
Datenstrukturen

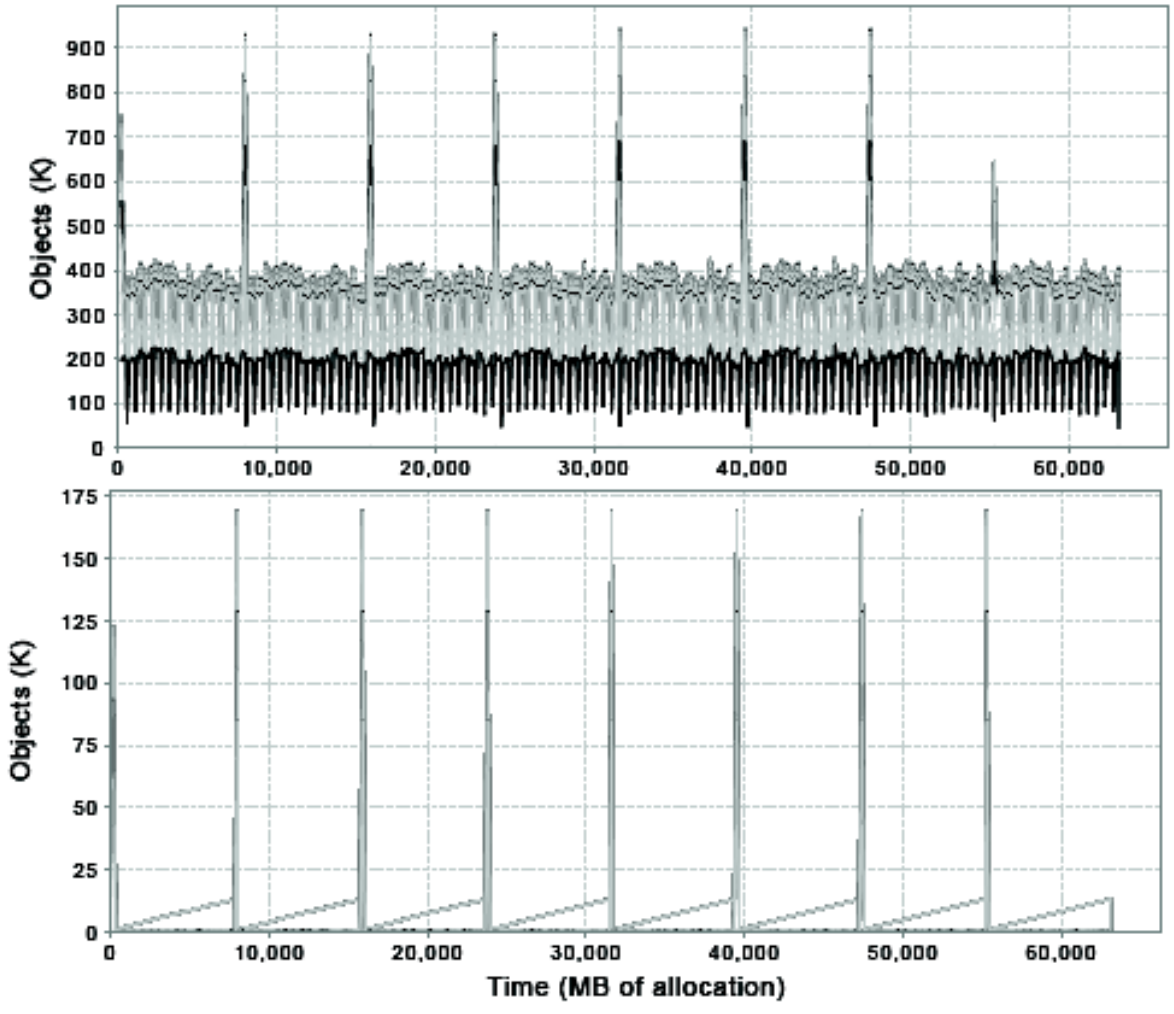
- Objekt Allokation
- Objekt-Lebenszeit (unterscheidet sich stark von Allokation!)
- Objekt-Größen
- Heap-Zusammensetzung
- Aktivität im Objektnetz

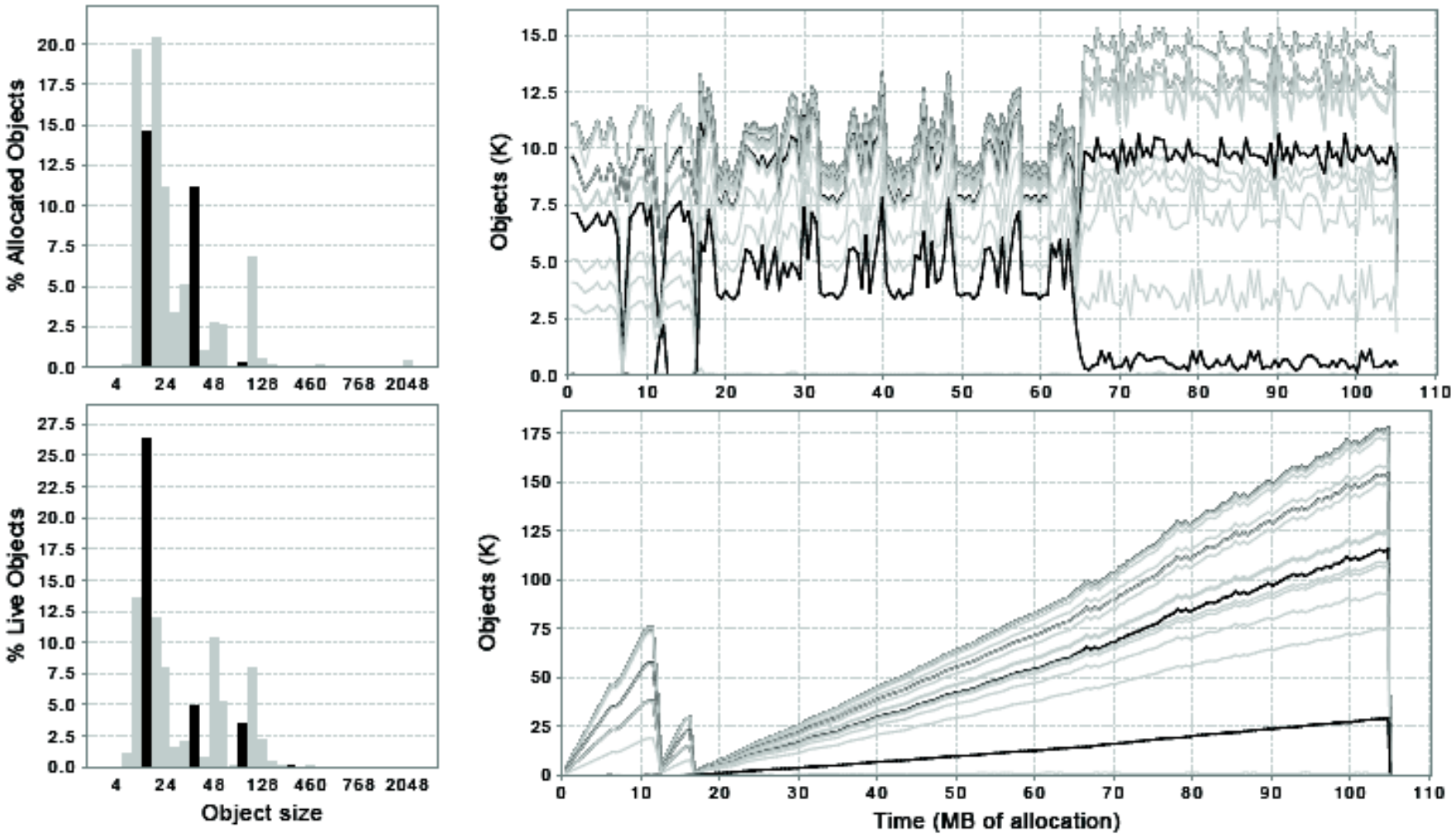
Jikes JIT in Java: separater Heap notwendig

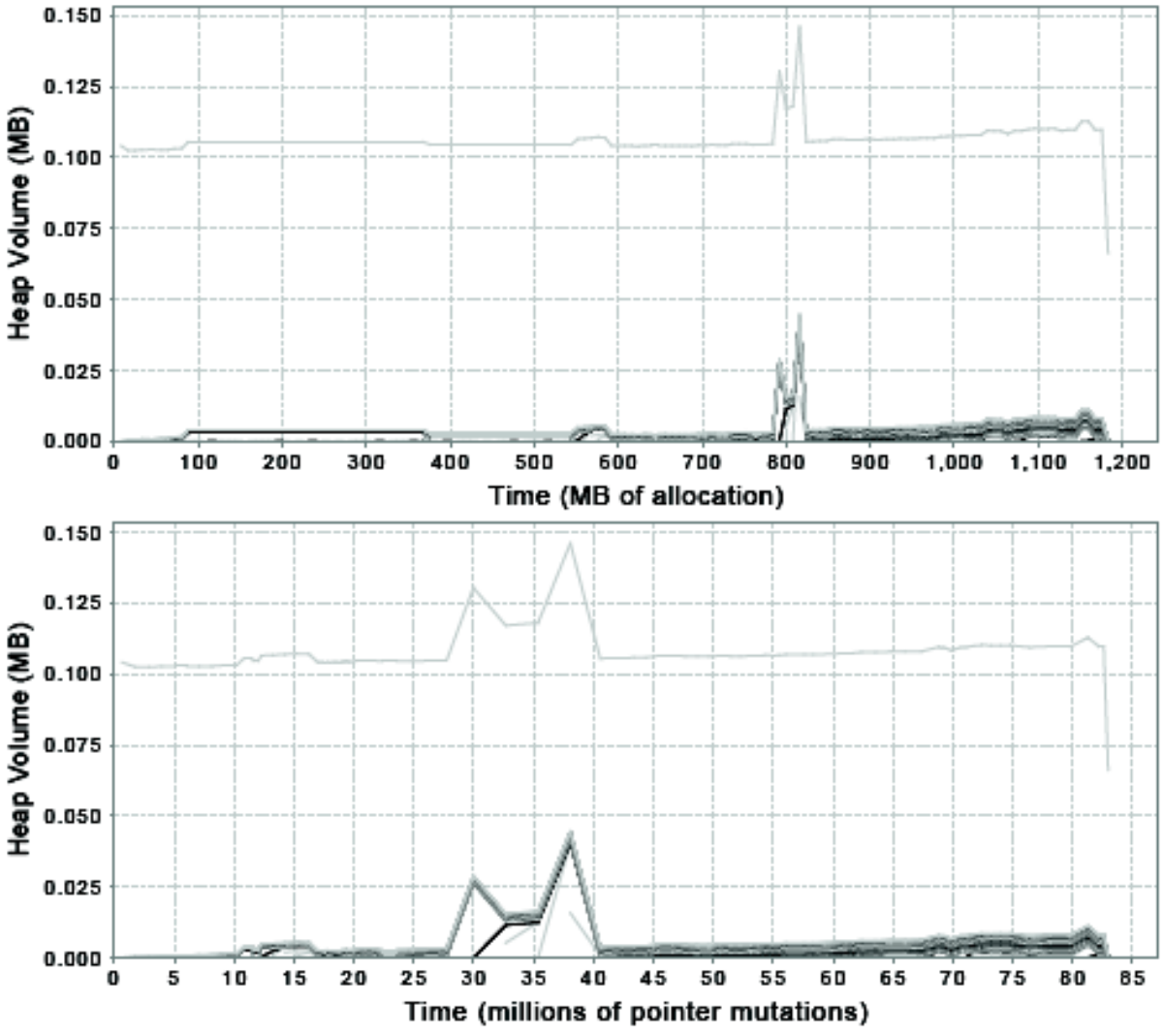
Benchmark	Classes Loaded	Methods Declared	Methods & Bytecodes Compiled						I-Cache Misses			
			All		Optimized		% Hot		L1 I-cache		ITLB	
			Methods	BC KB	Methods	BC KB	Methods	BC	/ms	norm	/ms	norm
SPEC												
_201_compress	157	1118	254	23.9	16	3.7	6.3	15.5	69	0.08	4	0.07
_202_jess	293	1777	655	42.4	46	4.3	7.0	10.1	383	0.45	31	0.56
_205_raytrace	177	1316	381	32.4	44	9.0	11.5	27.8	1826	2.12	191	3.42
_209_db	149	1108	249	23.7	11	1.5	4.4	6.3	34	0.04	2	0.04
_213_javac	302	2261	978	89.0	141	25.4	14.4	28.5	6356	7.39	672	12.04
_222_mpegaudio	200	1407	425	68.3	88	19.4	20.7	28.4	731	0.85	24	0.43
_227_mtrt	178	1318	379	32.4	39	7.6	10.3	23.5	1940	2.25	45	0.81
_228_jack	202	1392	488	53.6	33	5.4	6.8	10.1	3142	3.65	201	3.60
pseudojbb	238	2622	824	69.7	174	25.7	21.1	36.9	5556	6.46	759	13.60
min	149	1108	249	23.7	11	1.5	4.4	6.3	34	0.04	2	0.04
max	302	2622	978	89.0	174	25.7	21.1	36.9	6356	7.39	759	13.60
geomean	**204**	**1523**	**464**	**43.6**	**46**	**7.8**	**10.0**	**18.0**	**860**	**1.00**	**56**	**1.00**
DaCapo												
antlr	307	3517	1541	212.7	101	14.1	6.6	6.6	6198	7.20	597	10.70
bloat	471	5231	2023	169.1	100	9.5	4.9	5.6	6031	7.01	398	7.13
chart	706	8972	2299	204.1	113	20.8	4.9	10.2	11919	13.85	952	17.06
eclipse	1023	12450	3713	243.0	14	2.0	0.4	0.8	5053	5.87	702	12.58
fop	865	5761	2593	206.0	69	7.8	2.7	3.8	6603	7.68	532	9.53
hsqldb	355	5970	1411	130.2	122	18.9	8.6	14.5	4866	5.66	524	9.39
luindex	309	3118	940	74.3	168	29.3	17.9	39.4	1876	2.18	154	2.76
lusearch	295	2795	822	65.5	133	21.7	16.2	33.1	10183	11.84	1888	33.84
jython	886	9443	3242	462.5	297	28.5	9.2	6.2	2114	2.46	226	4.05
pmd	619	6163	2247	152.4	137	14.3	6.1	9.4	2819	3.28	223	4.00
xalan	552	6562	1747	126.2	194	36.0	11.1	28.5	3718	4.32	268	4.80
min	295	2795	822	65.5	14	2.0	0.4	0.8	1876	2.18	154	2.76
max	1023	12450	3713	462.5	297	36.0	17.9	39.4	11919	13.85	1888	33.84
geomean	**527**	**5768**	**1866**	**162.4**	**108**	**14.8**	**5.8**	**9.1**	**4792**	**5.57**	**455**	**8.16**

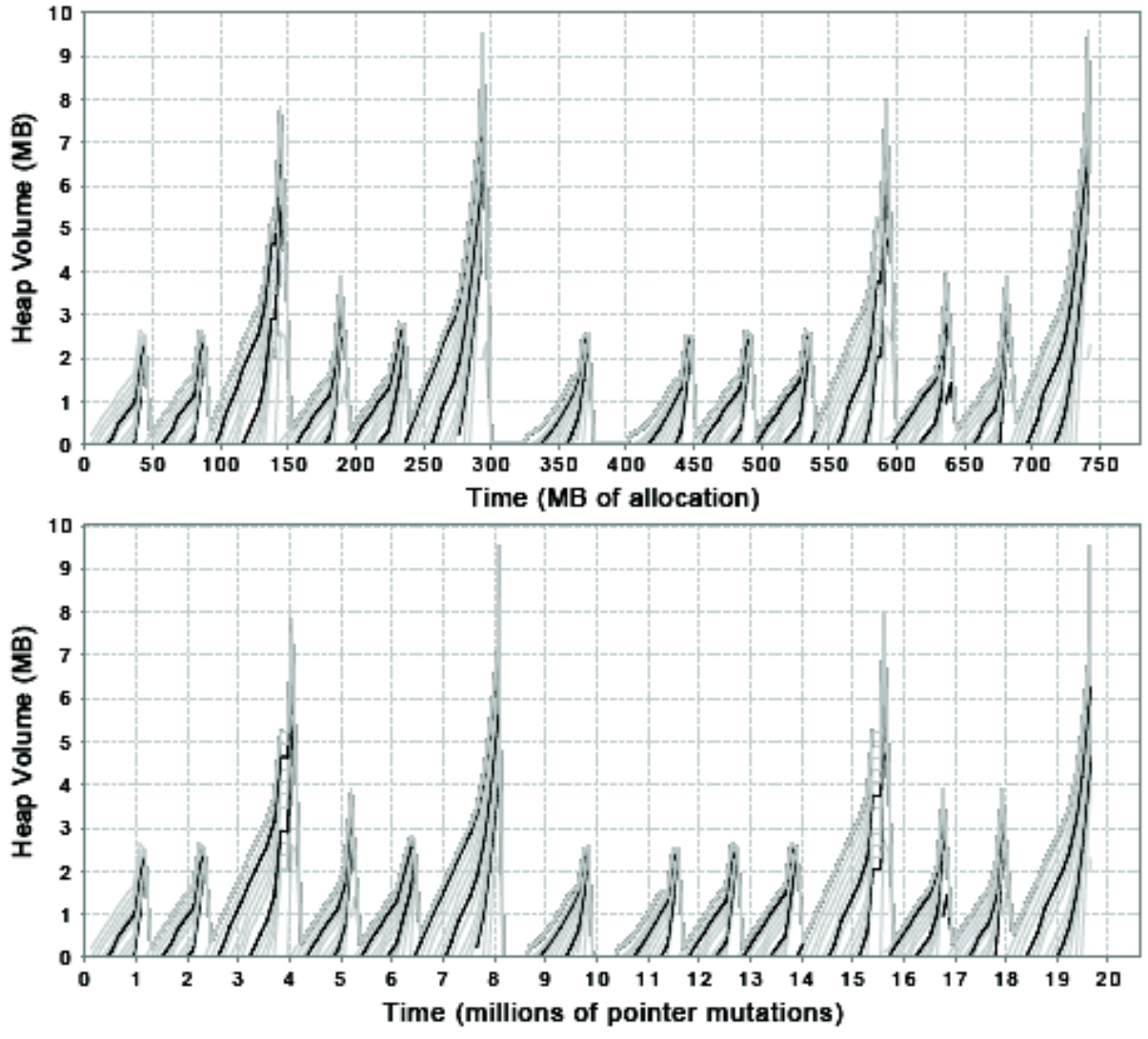
Benchmark	Heap Volume (MB)			Heap Objects			Mean Object Size		4MB Nursery Survival %
	Alloc	Live	Alloc/ Live	Alloc	Live	Alloc/ Live	Alloc	Live	
SPEC									
_201_compress	105.4	6.3	16.8	3,942	270	14.6	28,031	24,425	6.6
_202_jess	262.0	1.2	221.3	7,955,141	22,150	359.1	35	56	1.1
_205_raytrace	133.5	3.8	35.1	6,397,943	153,555	41.7	22	26	3.6
_209_db	74.6	8.5	8.8	3,218,642	291,681	11.0	24	31	14.6
_213_javac	178.3	7.2	24.8	5,911,991	263,383	22.4	32	29	25.8
_222_mpegaudio	0.7	0.6	1.1	3,022	1,623	1.9	245	406	50.5
_227_mtrt	140.5	7.2	19.5	6,689,424	307,043	21.8	22	25	6.6
_228_jack	270.7	0.9	292.7	9,393,097	11,949	786.1	30	81	2.8
pseudojbb	207.1	21.1	9.8	6,158,131	234,968	26.2	35	94	31.3
min	0.7	0.6	1.1	3,022	270	1.9	22	25	1.1
max	270.7	21.1	292.7	9,393,097	307,043	786.1	28,031	24,425	50.5
geomean	**86.5**	**3.8**	**23.0**	**1,180,850**	**35,886**	**32.9**	**77**	**110**	**8.7**
DaCapo									
antlr	237.9	1.0	248.8	4,208,403	15,566	270.4	59	64	8.2
bloat	1,222.5	6.2	195.6	33,487,434	149,395	224.2	38	44	6.0
chart	742.8	9.5	77.9	26,661,848	190,184	140.2	29	53	6.3
eclipse	5,582.0	30.0	186.0	104,162,353	470,333	221.5	56	67	23.8
fop	100.3	6.9	14.5	2,402,403	177,718	13.5	44	41	14.2
hsqldb	142.7	72.0	2.0	4,514,965	3,223,276	1.4	33	23	63.4
jython	1,183.4	0.1	8,104.0	25,940,819	2,788	9,304.5	48	55	1.6
luindex	201.4	1.0	201.7	7,202,623	18,566	387.9	29	56	23.7
lusearch	1,780.8	10.9	162.8	15,780,651	34,792	453.6	118	330	1.1
pmd	779.7	13.7	56.8	34,137,722	419,789	81.3	24	34	14.0
xalan	60,235.6	25.5	2,364.0	161,069,019	168,921	953.5	392	158	3.8
min	100.3	0.1	2.0	2,402,403	2,788	1.4	24	23	1.1
max	60,235.6	72.0	8,104.0	161,069,019	3,223,276	9,304.5	392	330	63.4
geomean	**907.5**	**6.2**	**147.6**	**18,112,439**	**103,890**	**174.3**	**53**	**62**	**8.4**


% Allocated Objects
55
50
45
40
35
30
25
20
15
10
5
0
4
24
48
128
460
768
2048
% Live Objects
70
60
50
40
30
20
10
0
4
24
48
128
460
768
2048
Object size


Objects (K)
900
800
700
600
500
400
300
200
100
0
0
10,000
20,000
30,000
40,000
50,000
60,000
Objects (K)
175
150
125
100
75
50
25
0
0
10,000
20,000
30,000
40,000
50,000
60,000
Time (MB of allocation)


% Allocated Objects
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
4
24
48
128
460
768
2048
% Live Objects
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
4
24
48
128
460
768
2048
Object size
Objects (K)
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
10
20
30
40
50
60
70
80
90
100
110
Objects (K)
175
150
125
100
75
50
25
0
0
10
20
30
40
50
60
70
80
90
100
110
Time (MB of allocation)


Heap Volume (MB)
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
100
200
300
400
500
600
700
800
900
1,000
1,100
1,200
Time (MB of allocation)
Heap Volume (MB)
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
Time (millions of pointer mutations)


Heap Volume (MB)
0
1
2
3
4
5
6
7
8
9
10
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
Time (MB of allocation)
Heap Volume (MB)
0
1
2
3
4
5
6
7
8
9
10
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Time (millions of pointer mutations)

04 Aktuelle Versionen

DaCapo Benchmark Suite

- Anpassung etwa alle 2-3 Jahre
- ursprünglich 11, inzwischen 14 Applikationen
- letztes Release: 2009-12-24
- 167MB

SPECjvm2008

- wesentlich komplexer
- Client- und Server
- Multi-threading
- 11 Applikationen
- Base und Peak Score
- Quellcode verfügbar
- detailierte Auswertung
- 64MB

SPECjbb2005

- komplexere Transaktionen
- weiterhin durchsatzba-siert

Literatur

- The DaCapo Benchmarks: Java Benchmarking Development and Analysis (Extended Version)
- Java Performance Evaluation through Rigorous Replay Compilation
- Wake Up and Smell the Coffee: Evaluation Methodology for the 21st Century
- SPEC Benchmark Workshop 2009: Computer Performance Evaluation and Benchmarking
- SPECjbb2005 - A Year in the Life of a Benchmark
- http://www.dacapobench.org/ (letztes Release: 24.12.2009)
- http://www.spec.org/
- http://www-plan.cs.colorado.edu/henkel/projects/colorado_bench/
- http://www.dhpc.adelaide.edu.au/projects/javagrande/benchmarks/
- http://jikesrvm.org/