TECHNISCHE UNIVERSITÄT DRESDEN

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

Leistungssteigerung der Speicherarchitektur des SHAP-Mehrkernprozessors

Vortrag zum Diplom

Andrej Olunczek
andrej@olunczek.de

Dresden, 02.11.2011

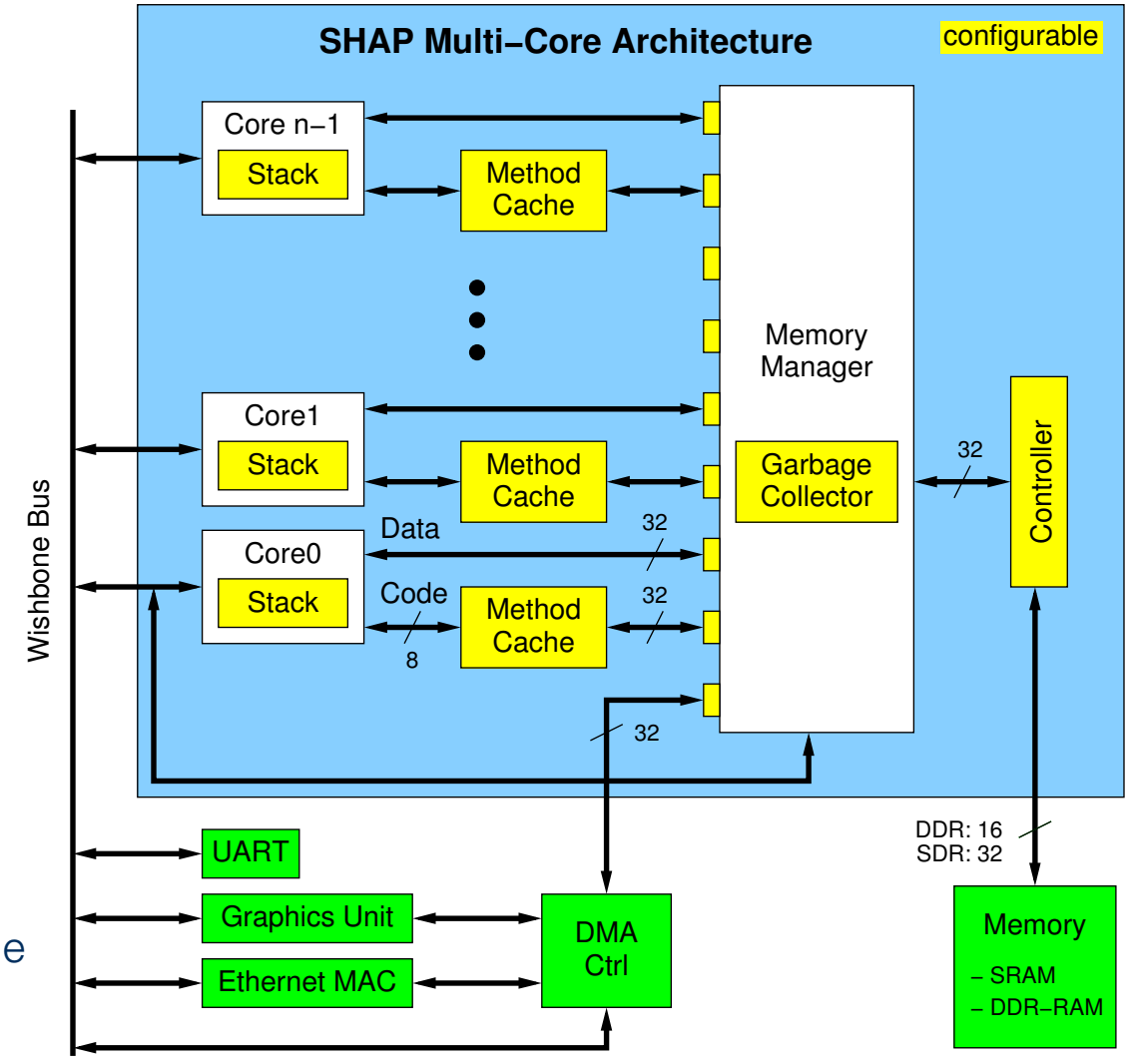
TECHNISCHE UNIVERSITÄT DRESDEN

Gliederung

Einführung

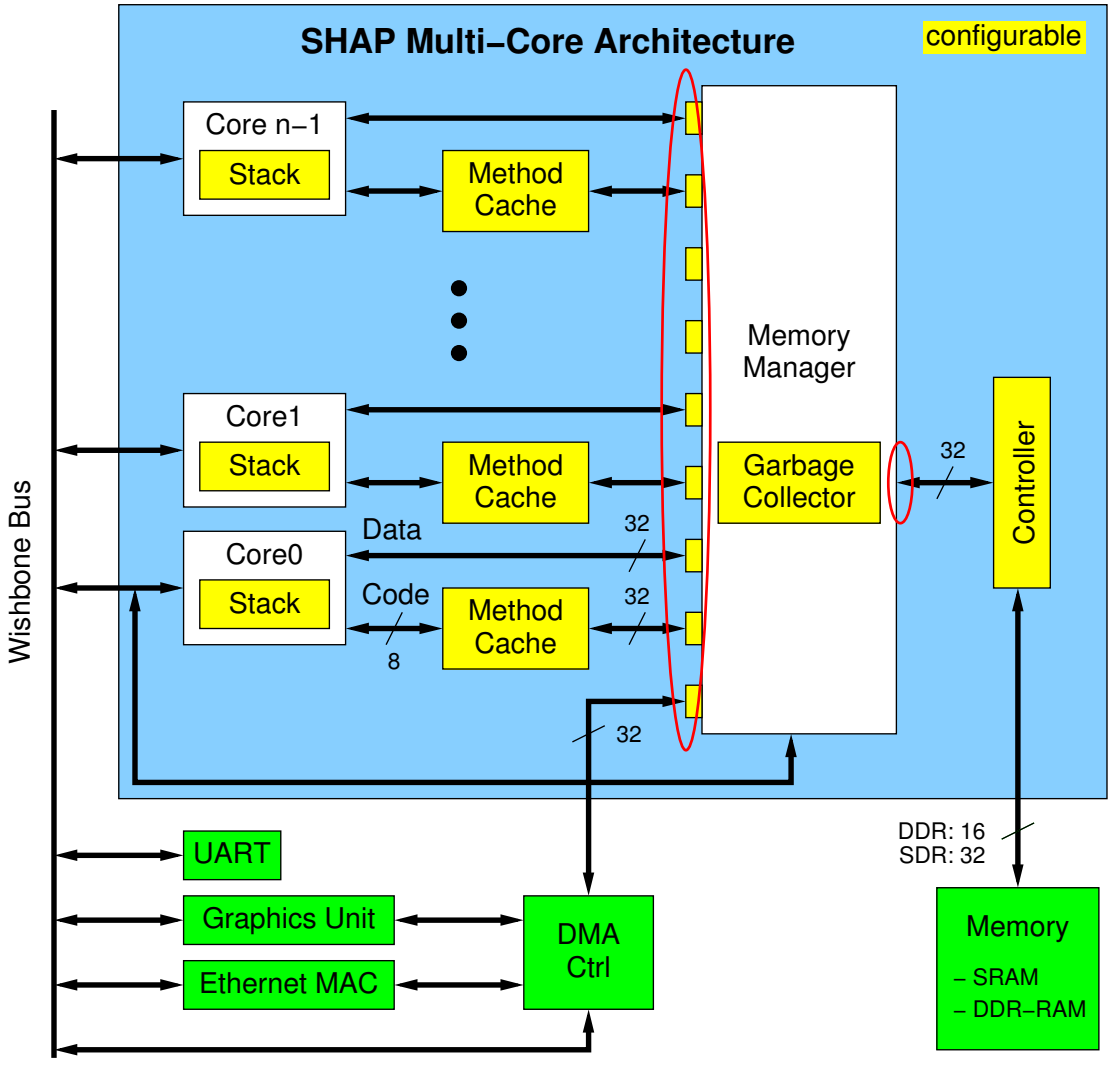
Der SHAP-Mehrkernprozessor

- führt nativ Java-Bytecode aus
- frei konfigurierbar, auch in der Anzahl der Kerne
- objektorientierte Speicherverwaltung mit nebenläufiger Garbagecollection
- getrennter Zugriff auf Daten und Befehle



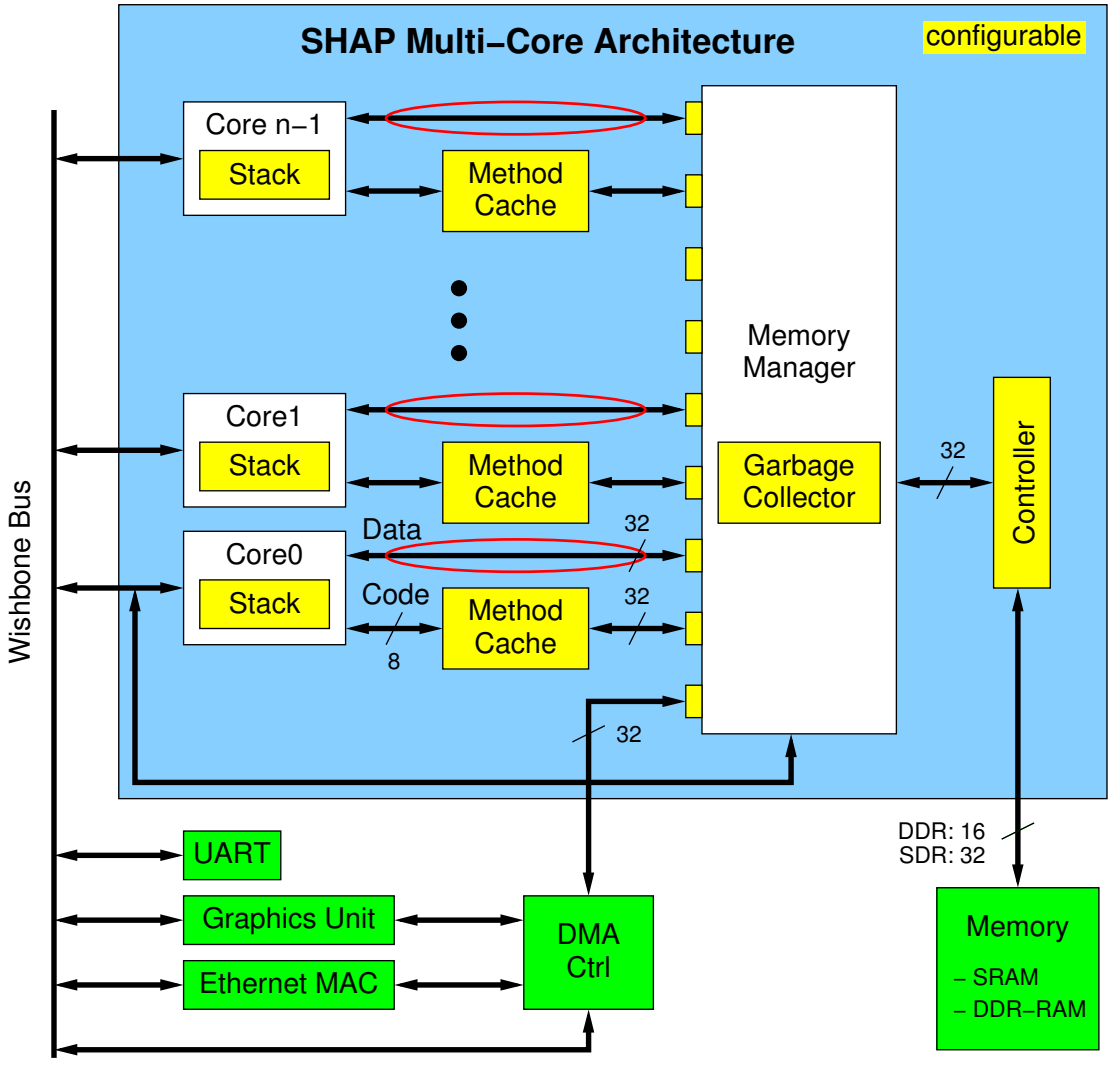
Einführung
Motivation

- viele Kerne greifen auf einen gemeinsamen Speicher zu
- je mehr Kerne, umso weniger Speicherbandbreite pro Kern
- Befehlszugriffe sind bereits über einen Methodencache optimiert



Einführung
Motivation

- viele Kerne greifen auf einen gemeinsamen Speicher zu
- je mehr Kerne, umso weniger Speicherbandbreite pro Kern
- Befehlszugriffe sind bereits über einen Methodencache optimiert
- ⇒ Konzentration auf Datenzugriffe



Einführung
Datenzugriffe

- Objekte werden über eine Objektreferenz (virtuelle Adresse) angesprochen
- Memory Manager verwaltet eine Tabelle mit den physikalischen Adressen aller Objekte
- Die physikalische Adresse ist durch den Garbage Collector veränderbar
- Für jeden Datenzugriff wird zuerst die Referenz aufgerufen und durch den Memory Manager die physikalische Basis-Adresse geladen
- danach folgen ein oder mehrere Zugriffe auf die per Offset definierten Datenwörter
- Die Zugehörigkeit von physikalischen Adressen zu Objektreferenzen kann leicht in einem Translation Lookaside Buffer (TLB) gecacht werden
- aufwändiger ist das Cachen der eigentlichen Datenwörter

Caching von Objekten
komplette Objekte

- jHISC: kleine kurzlebige Objekte nur im Cache
- + keine Speicherzugriffe nötig
- - Kohärenzprobleme
- - Garbage Collection

Caching von Objekten
Teile eines Objektes

- JOP: nur bestimmte Offsetbereiche im Cache
- + deutliche Reduzierung der Speicherzugriffe
- - Kohärenzprobleme
- - Platzbedarf vs Geschwindigkeitsvorteil

Caching von Objekten
Ein Cache für alle Cores

- großer Cache in der MMU
- + nur eine Verwaltungslogik
- - Flächenbedarf für viele Cores
- - Cores verdrängen sich gegenseitig die Daten

Caching von Objekten
Caching von Konstanten

- nur konstante Werte im Cache halten
- + kein Kohärenzproblem
- + kleiner Flächenbedarf
- - nur wenige Daten cachebar

Caching von Objekten
Cache-Architekturen

- Aufbau
  - Direct-mapped
  - n-Wege-assoziativ
  - Vollassoziativ
- Ersetzungsstrategie
  - First-In, First-Out
  - Least-Recently-Used(LRU)
  - Random
  - Not-Most-Recently-Used(NMRU)
- kein Cachen von Schreibzugriffen, um Garbage Collection im Cache zu vermeiden
- $\Rightarrow$ für kleine Caches vollassoziativ + LRU für bestmögliche Ausnutzung

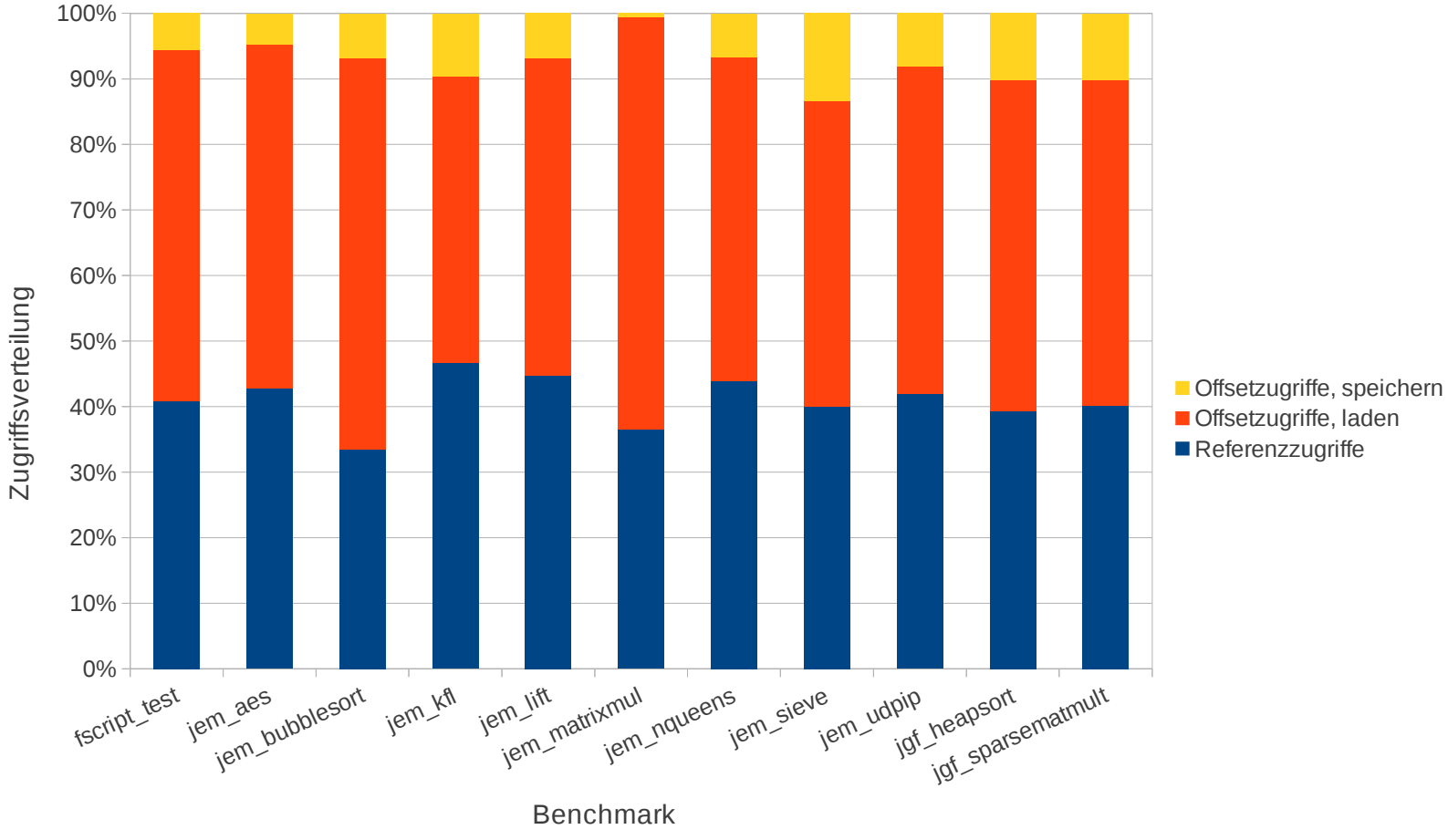
Analyse
Benchmarks

	mitgelesene Takte	ausgeführte Bytecodes	durchgeführte Speicherzugriffe
FScript-Test	13.731.709	2.540.511	1.179.728
JEM AES	10.574.368	2.173.511	761.898
JEM Bubblesort	10.749.533	2.450.909	1.175.378
JEM Kfl	38.059.086	8.947.838	2.052.515
JEM Lift	12.836.418	2.921.822	1.375.775
JEM Matrixmul	14.281.049	2.394.905	1.539.327
JEM NQueens	24.195.895	8.216.892	766. 117
JEM Sieve	10.603.322	2.633.788	1.075. 611
JEM Udpip	16.695.041	3.391.612	1.497.999
JGF Crypt	62.989.904	9.107.062	339.645
JGF Heapsort	12.453.892	1.957.153	889.337
JGF SparseMatmult	43.470.478	5.814.747	3.168.902

$\Rightarrow$ Tracing-Modul ermöglicht das Mitschreiben von komplexen Vorgängen im SHAP-Prozessor

Analyse

Verteilung der Speicherzugriffe



Analyse
Offset-Zugriffe - häufigste Offsets

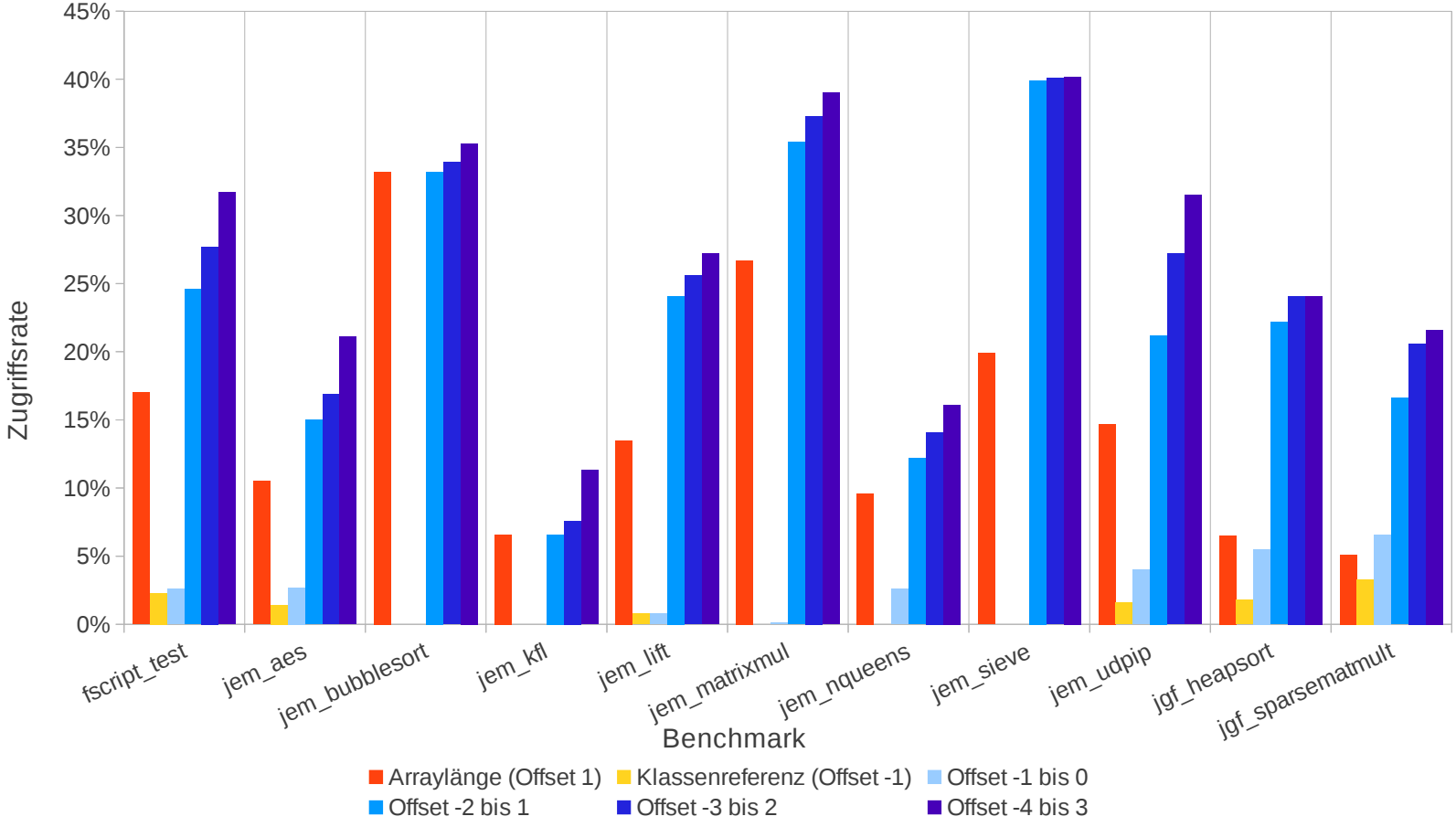
	Offset #1		Offset #2		Offset #3	
FScript-Test	0001	34,25%	0002	5,16%	0003	5,16%
JEM AES	0001	21,05%	000e	7,42%	3ffb	5,47%
JEM Bubblesort	0001	49,81%	0004	3,36%	0005	3,27%
JEM Kfl	0001	12,33%	000a	6,75%	3ffb	5,56%
JEM Lift	0001	25,50%	3ffe	17,00%	3ffb	16,62%
JEM Matrixmul	0001	42,31%	3ffe	13,39%	3ffd	1,61%
JEM NQueens	0089	40,40%	0001	17,11%	0004	11,88%
JEM Sieve	3ffe	33,23%	0001	33,23%	0035	0,54%
JEM UdpIp	0001	29,59%	3ffd	7,60%	0000	6,35%
JGF Heapsort	0001	19,84%	3ffe	10,70%	0000	9,12%
JGF SparseMatmult	0001	16,94%	0002	9,38%	0000	8,22%

Analyse
Offset-Zugriffe - häufigste verursachende Bytecodes

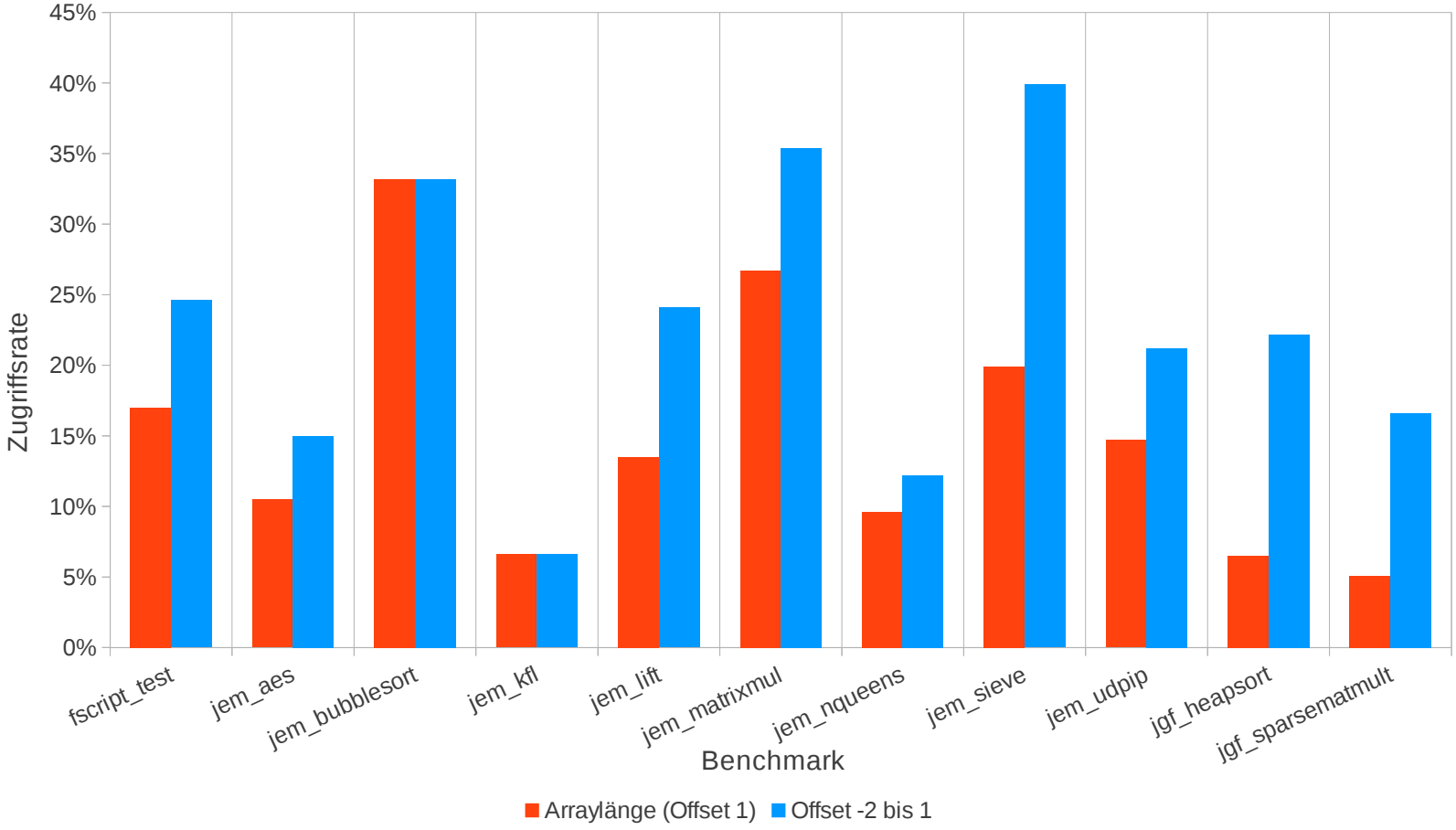
	Verursacher #1		Verursacher #2		Verursacher #3	
FScript-Test	*aload	53,78%	getfield	19,93%	inv.virtual	5,18%
JEM AES	*aload	30,75%	inv.special	14,71%	getstatic	10,74%
JEM Bubblesort	*aload	79,25%	*astore	20,38%	io_write	0,33%
JEM Kfl	getstatic	46,77%	inv.static	16,03%	*aload	13,49%
JEM Lift	getfield	44,71%	*astore	20,71%	*aload	17,00%
JEM Matrixmul	*aload	82,84%	getfield	14,85%	*astore	1,55%
JEM NQueens	inv.static	40,41%	*aload	21,94%	*astore	12,23%
JEM Sieve	*astore	44,75%	getfield	33,23%	*aload	21,71%
JEM UdpIp	*aload	36,78%	getstatic	14,16%	*astore	13,95%
JGF Heapsort	getfield	13,74%	*aload	12,22%	getstatic2	12,15%
JGF SparseMatmult	inv.virtual	16,43%	getstatic2	10,96%	*aload	10,57%

⇒ Arraylänge steht im Offset 0001 und wird vor jedem Zugriff kontrolliert

Analyse
Zugriffsraten (lesend)

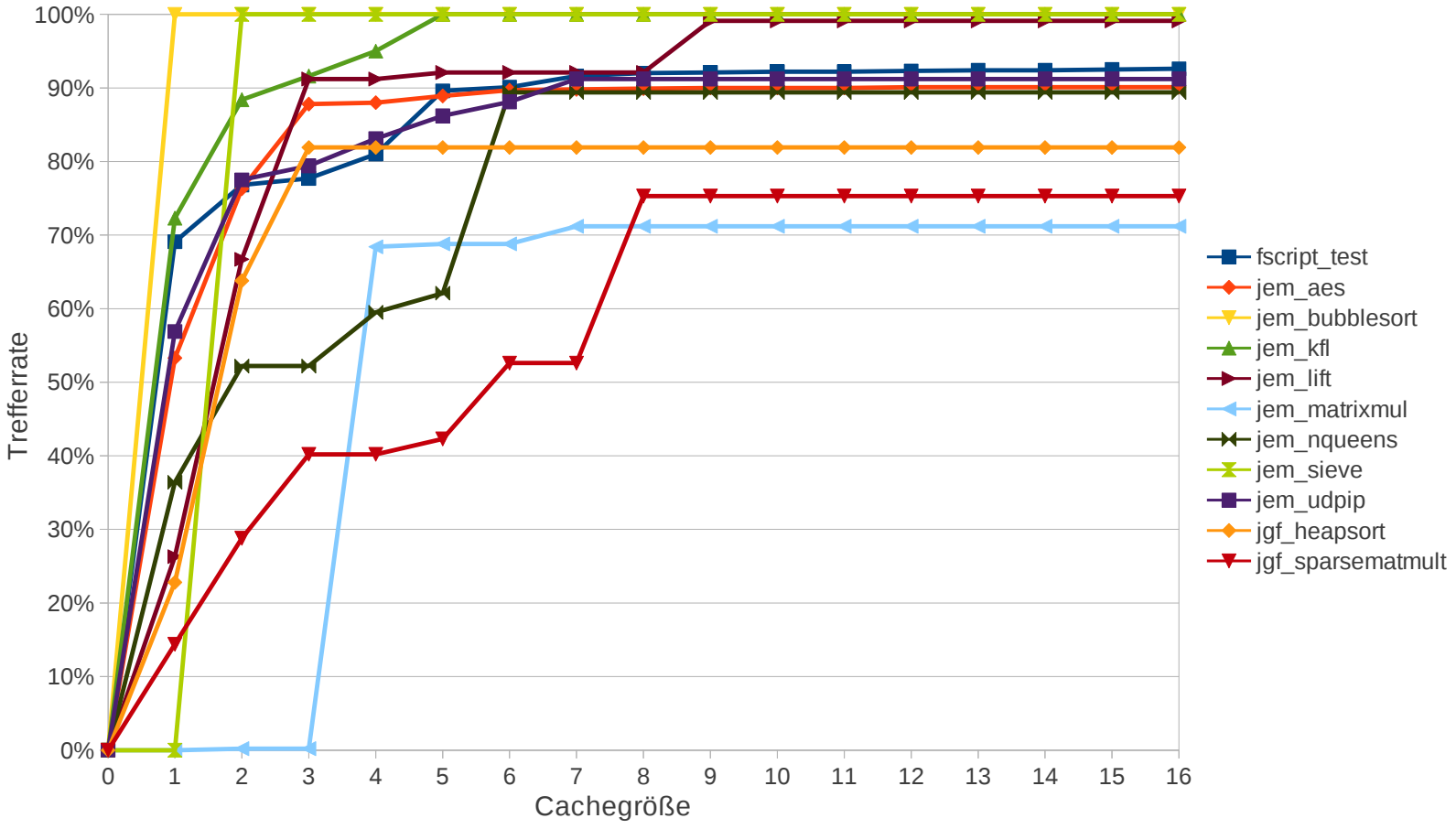


Analyse
Zugriffsraten (lesend)



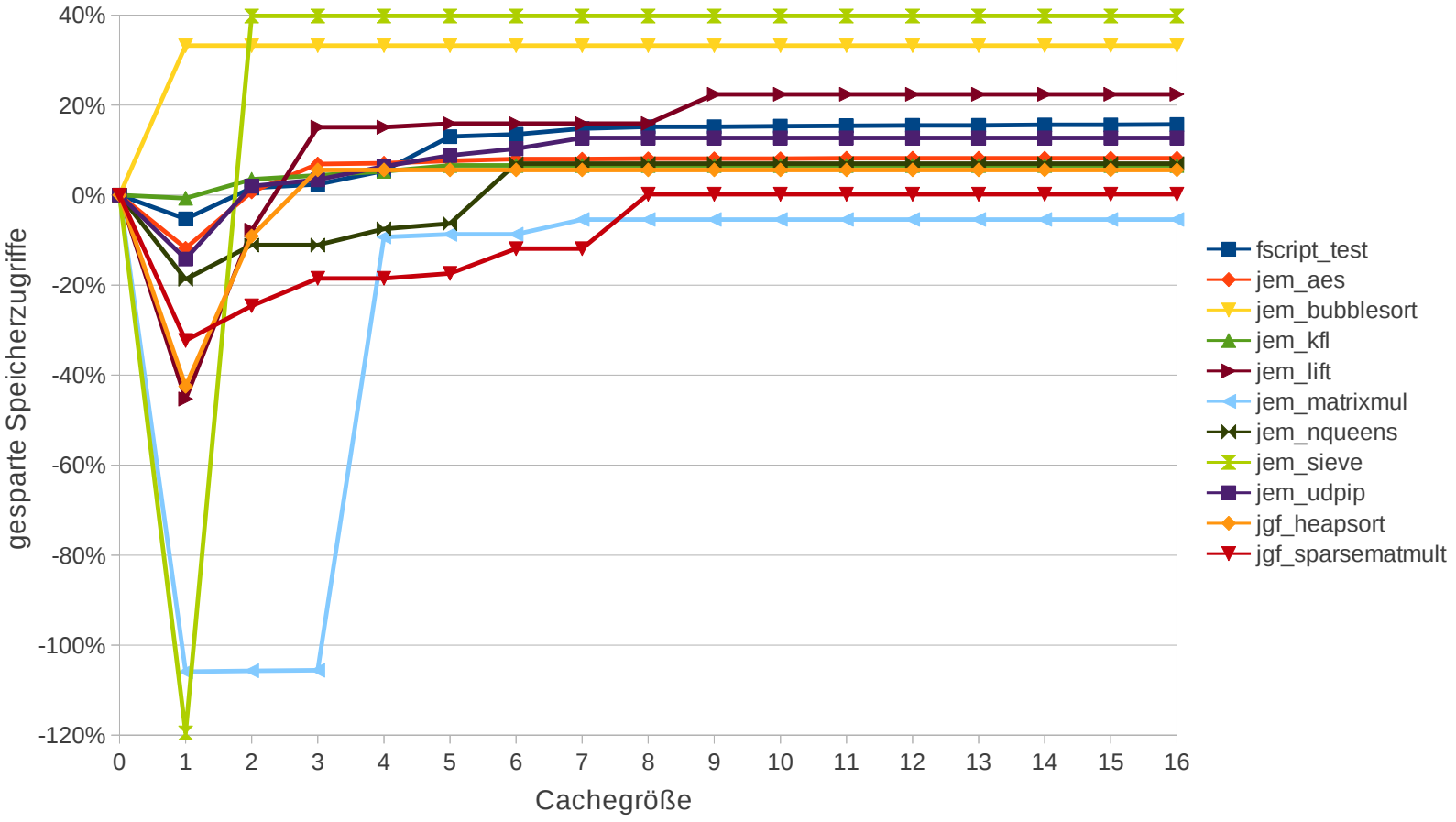
Simulation

Objectcache (Offset -2 bis 1) - Trefferrate



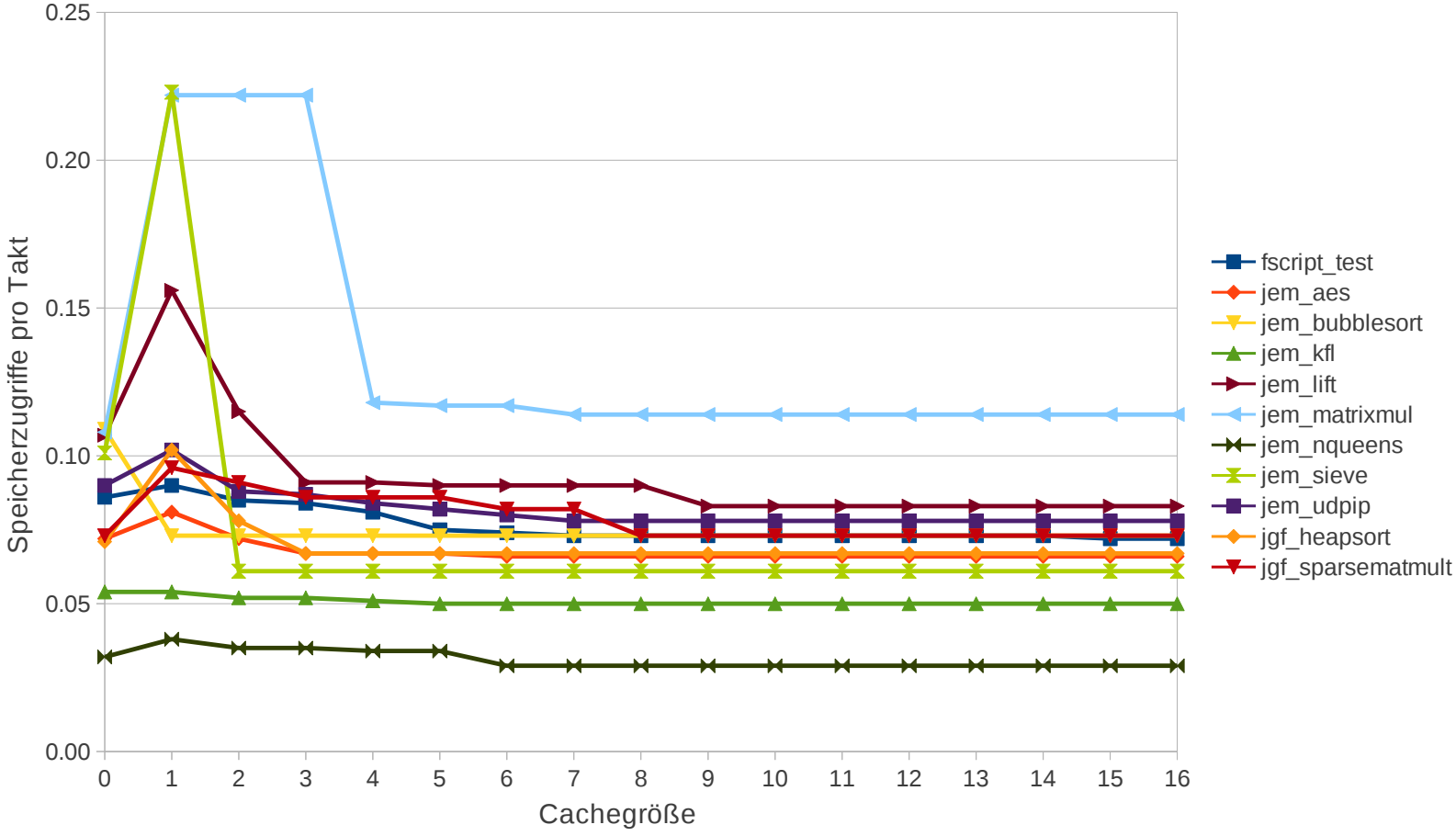
Simulation

Objectcache (Offset -2 bis 1) - gesparte Speicherzugriffe

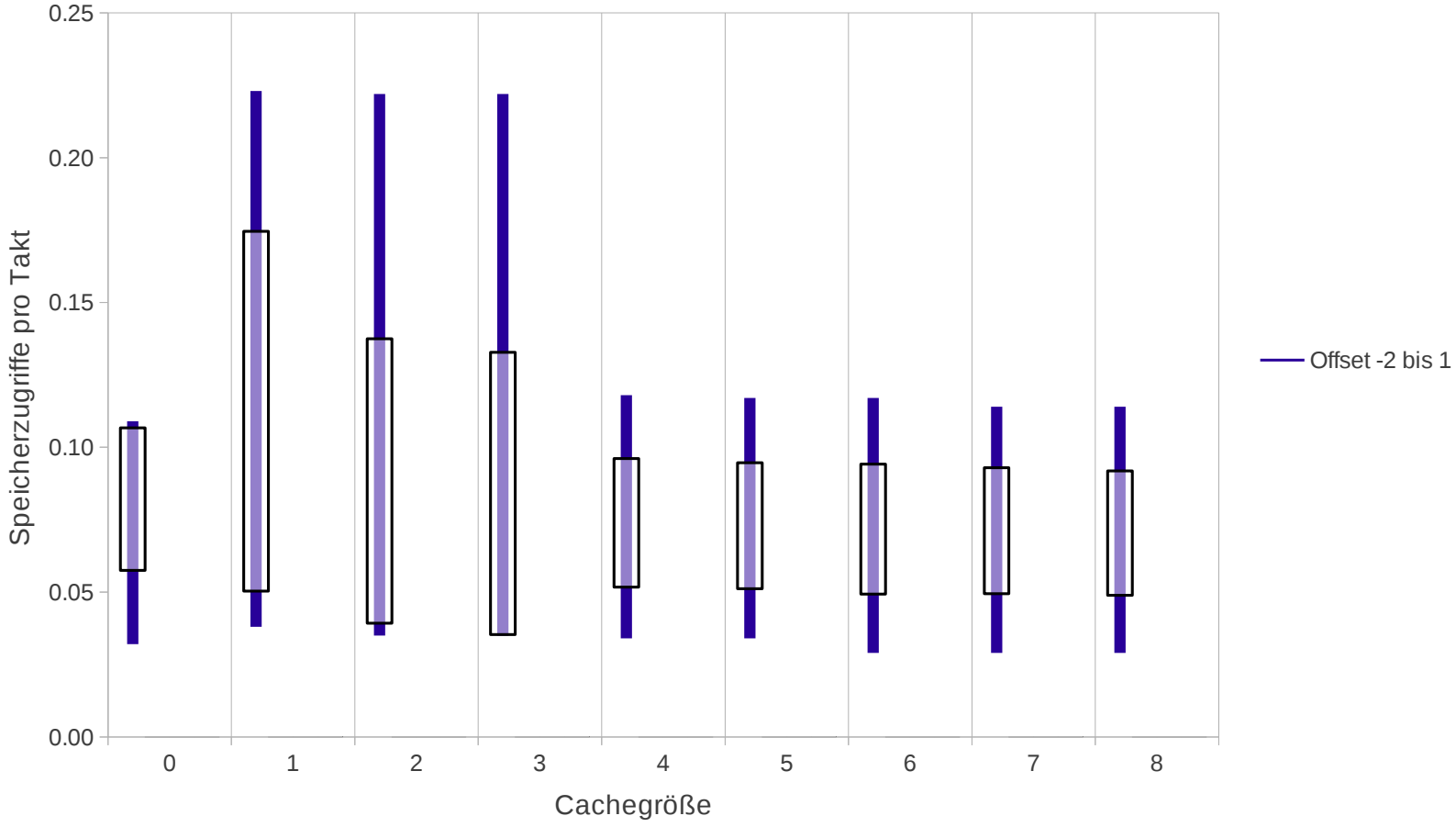


Simulation

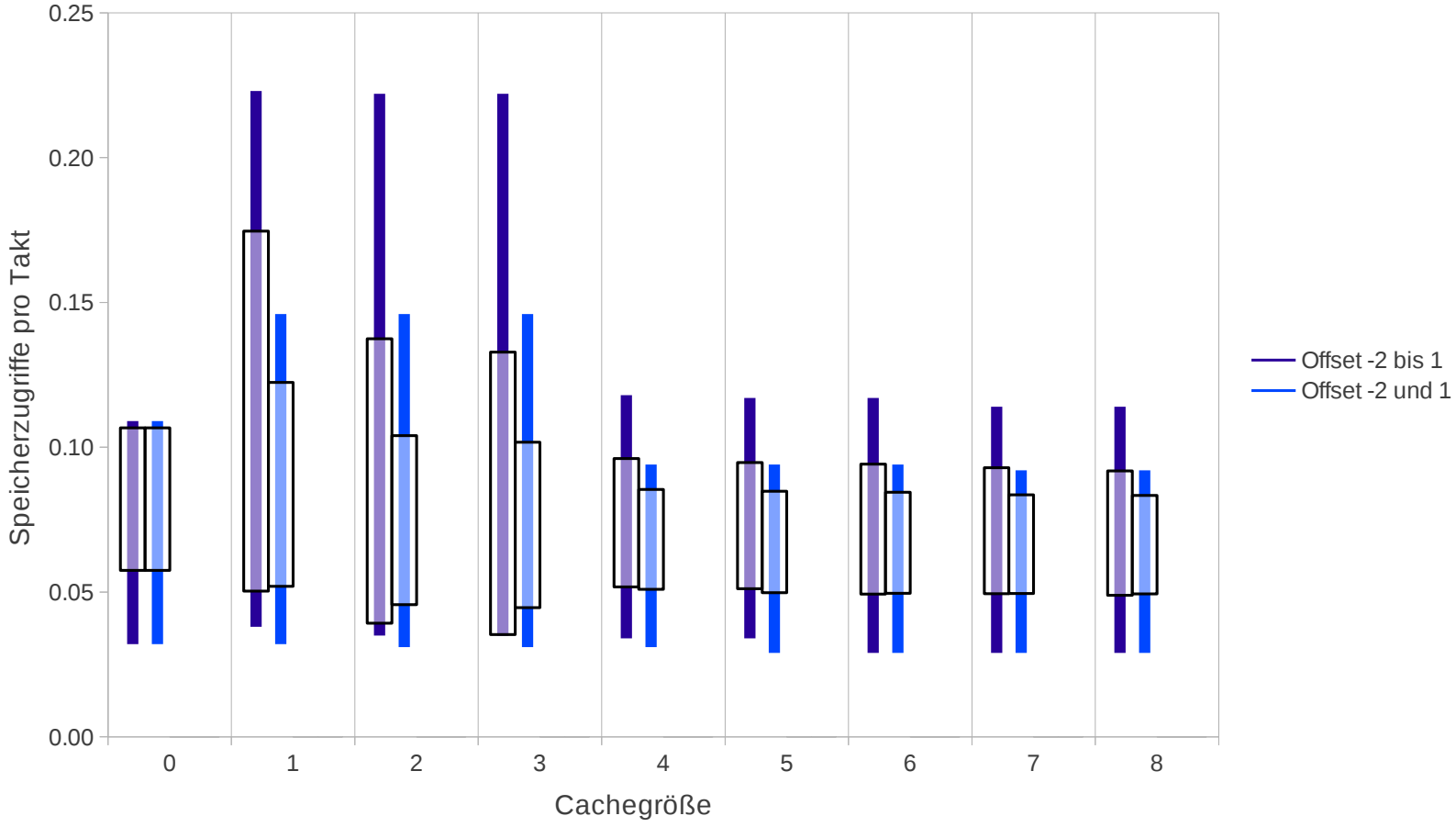
Objectcache (Offset -2 bis 1) - Speicherzugriffe pro Takt



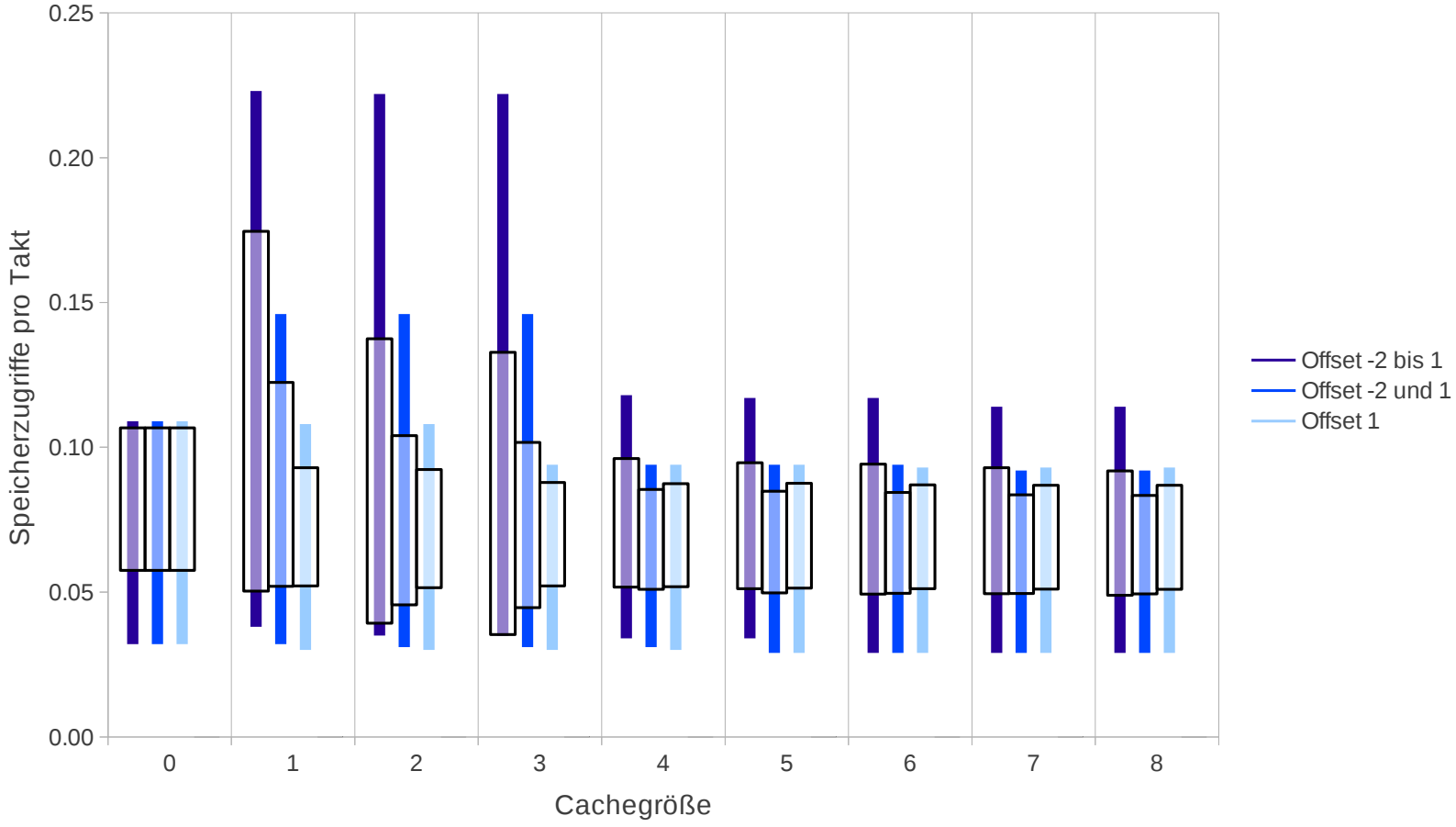
Simulation
Caching von Offsetzugriffen - Vergleich



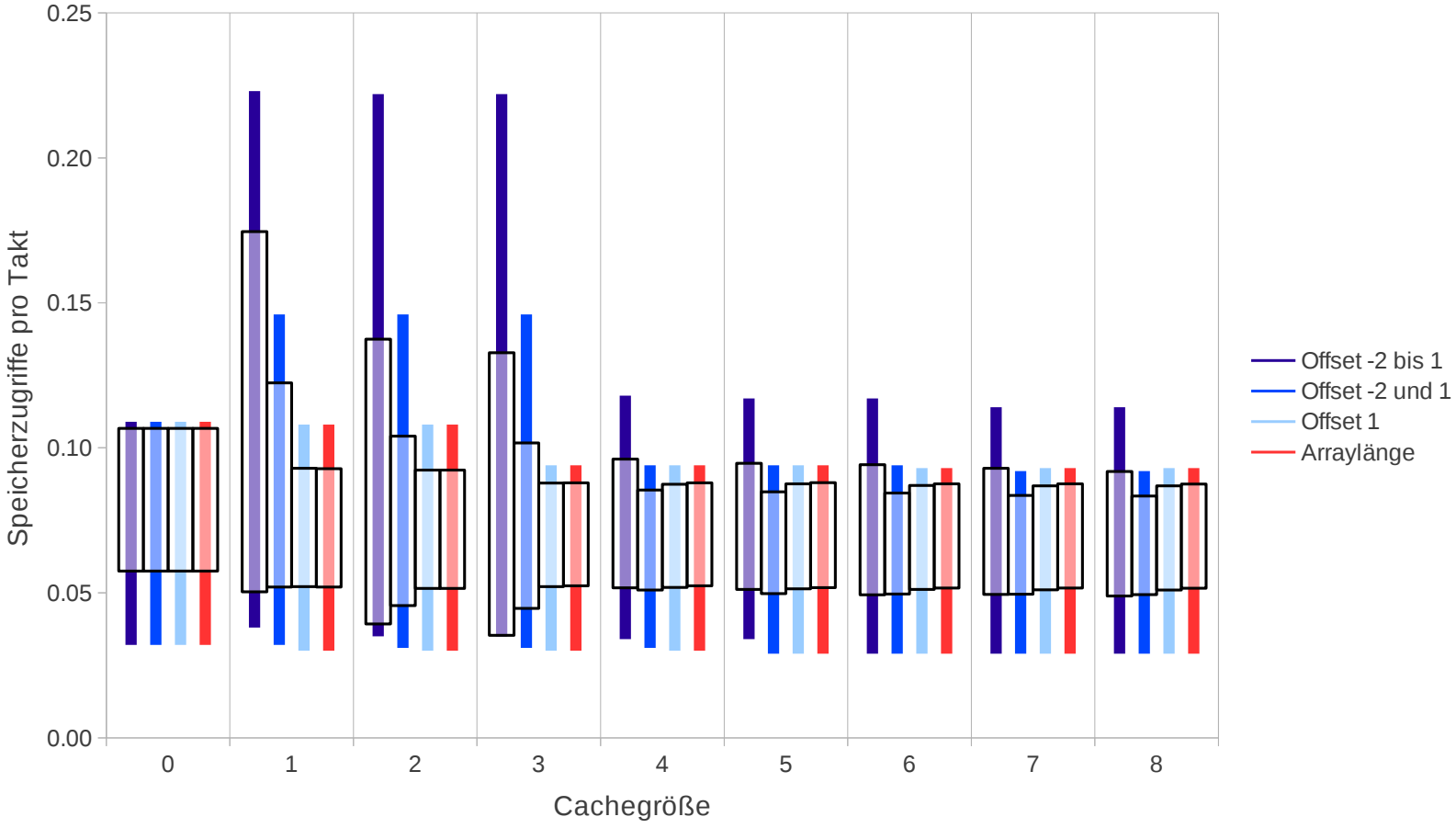
Simulation
Caching von Offsetzugriffen - Vergleich



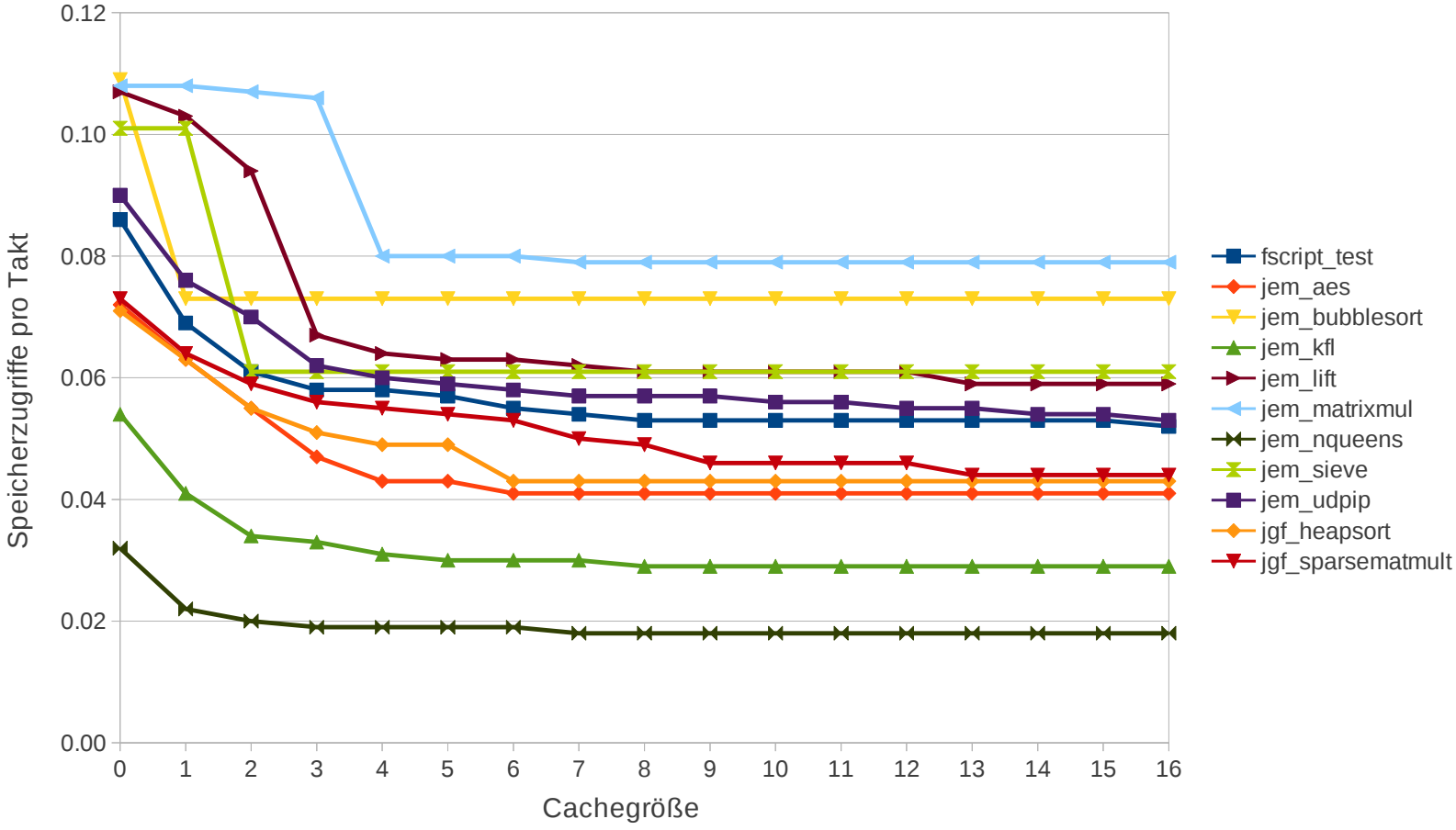
Simulation
Caching von Offsetzugriffen - Vergleich



Simulation
Caching von Offsetzugriffen - Vergleich

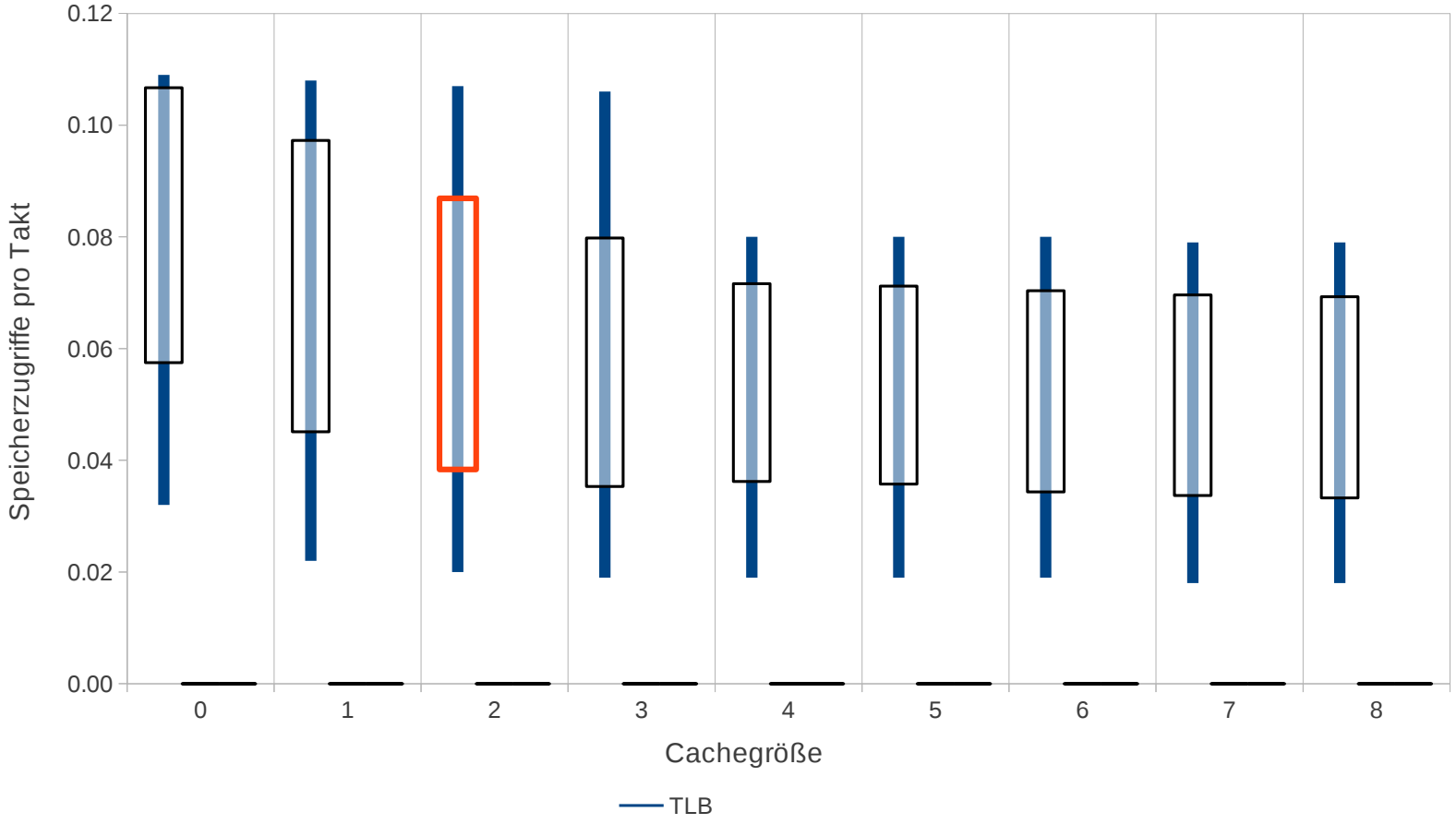


Simulation
TLB - Speicherzugriffe pro Takt



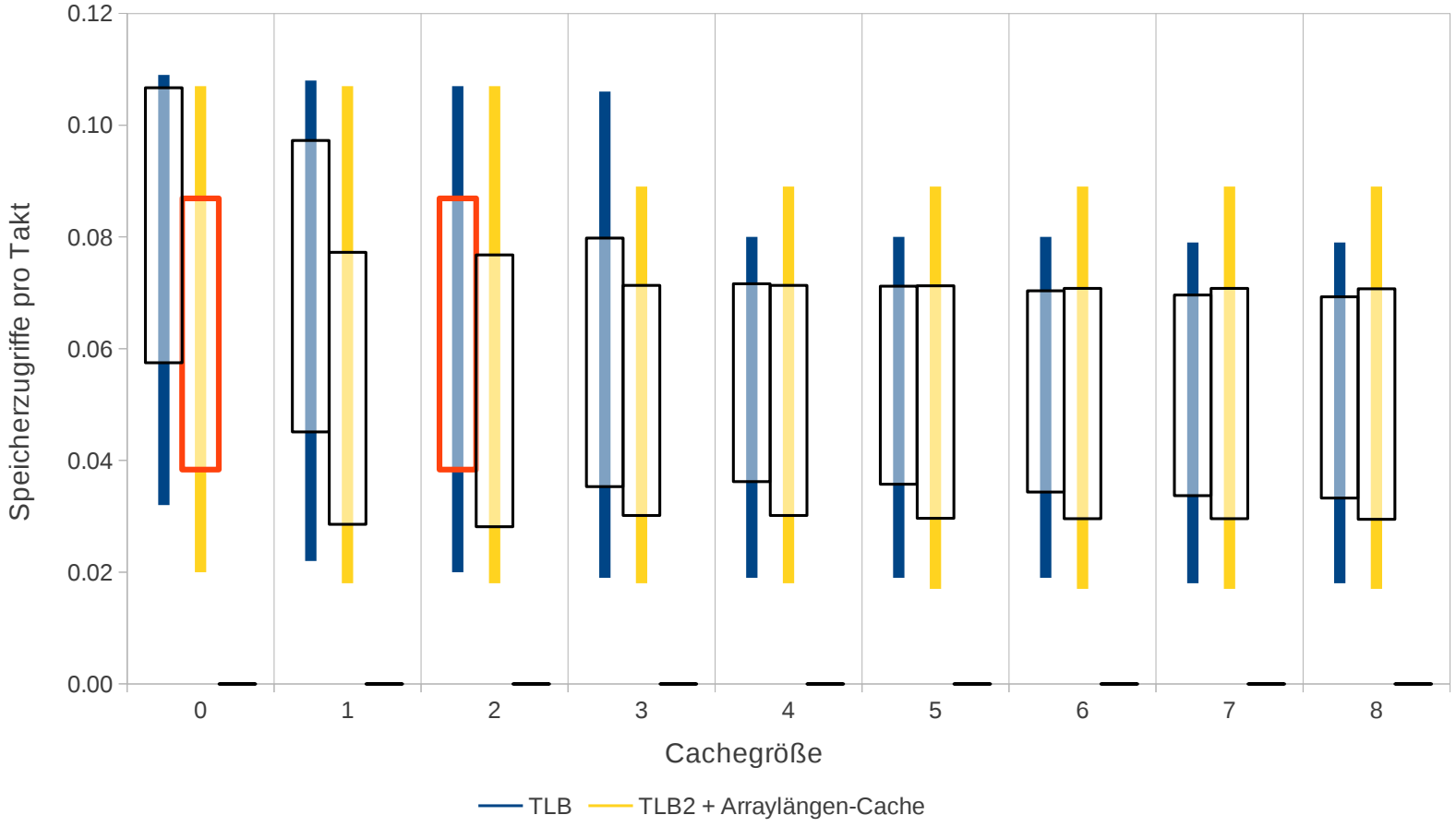
Bewertung und Ausblick

Vergleich TLB gegen Arraylängen-Cache



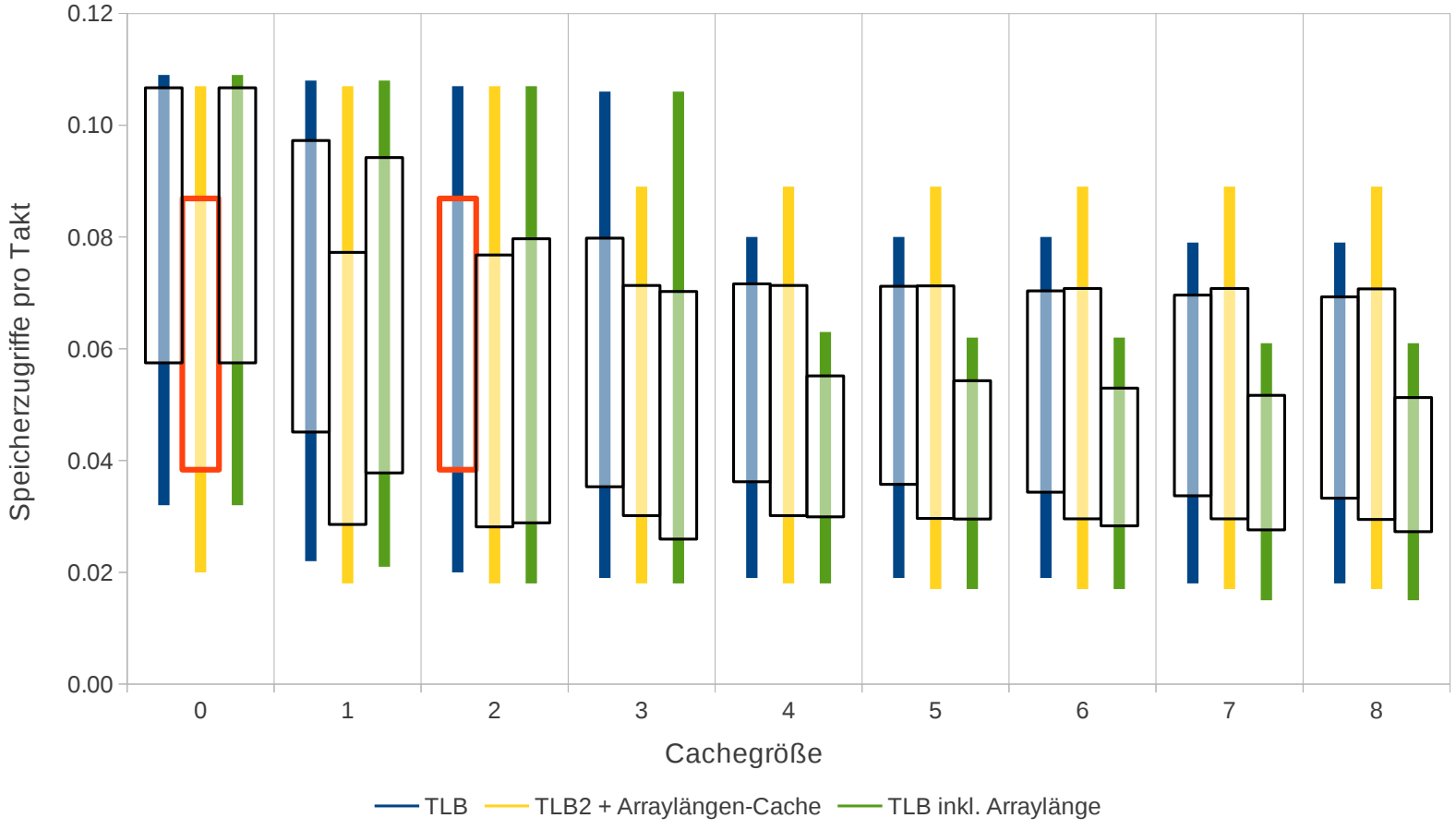
Bewertung und Ausblick

Vergleich TLB gegen Arraylängen-Cache



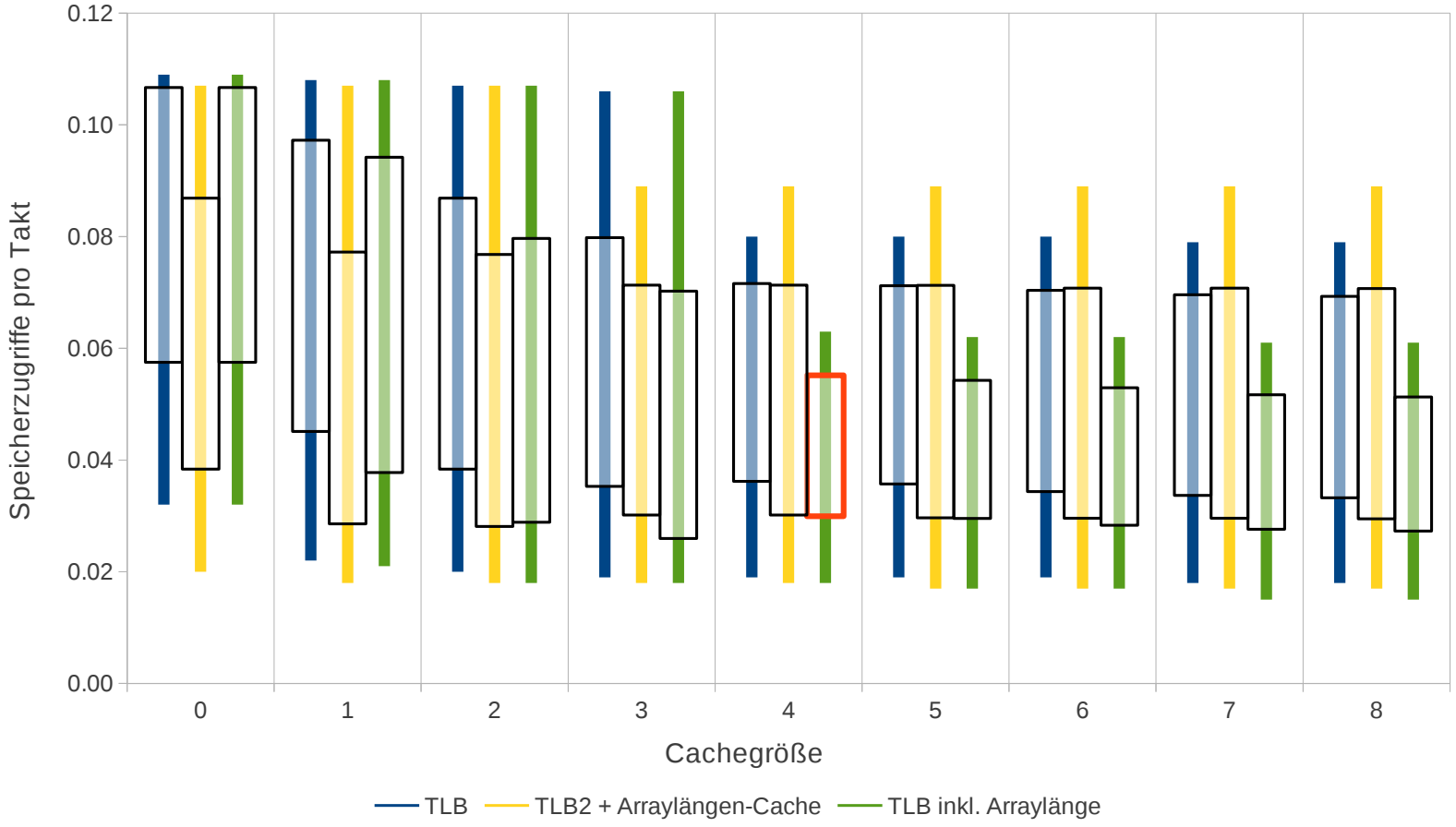
Bewertung und Ausblick

Vergleich TLB gegen Arraylängen-Cache



Bewertung und Ausblick

Zielstellung



Bewertung und Ausblick
Maßnahmen, Zusammenfassung

- TLB verdoppeln, bzw. vergrößern
- Arraylängen im TLB speichern
- wenn möglich, variable generische Implementation
- im Schnitt 30% der Datenzugriffe einsparbar
- nicht mit eingerechnet: Zugriffe durch den Methodencache
- Kompromiss zwischen Leistungssteigerung und Ressourcenverbrauch

Quellen

Towards Time-predictable Data Caches for Chip-Multiprocessors
Martin Schoeberl, Wolfgang Puffitsch, Benedikt Huber
http://www.jopdesign.com/doc/dcache_seus.pdf

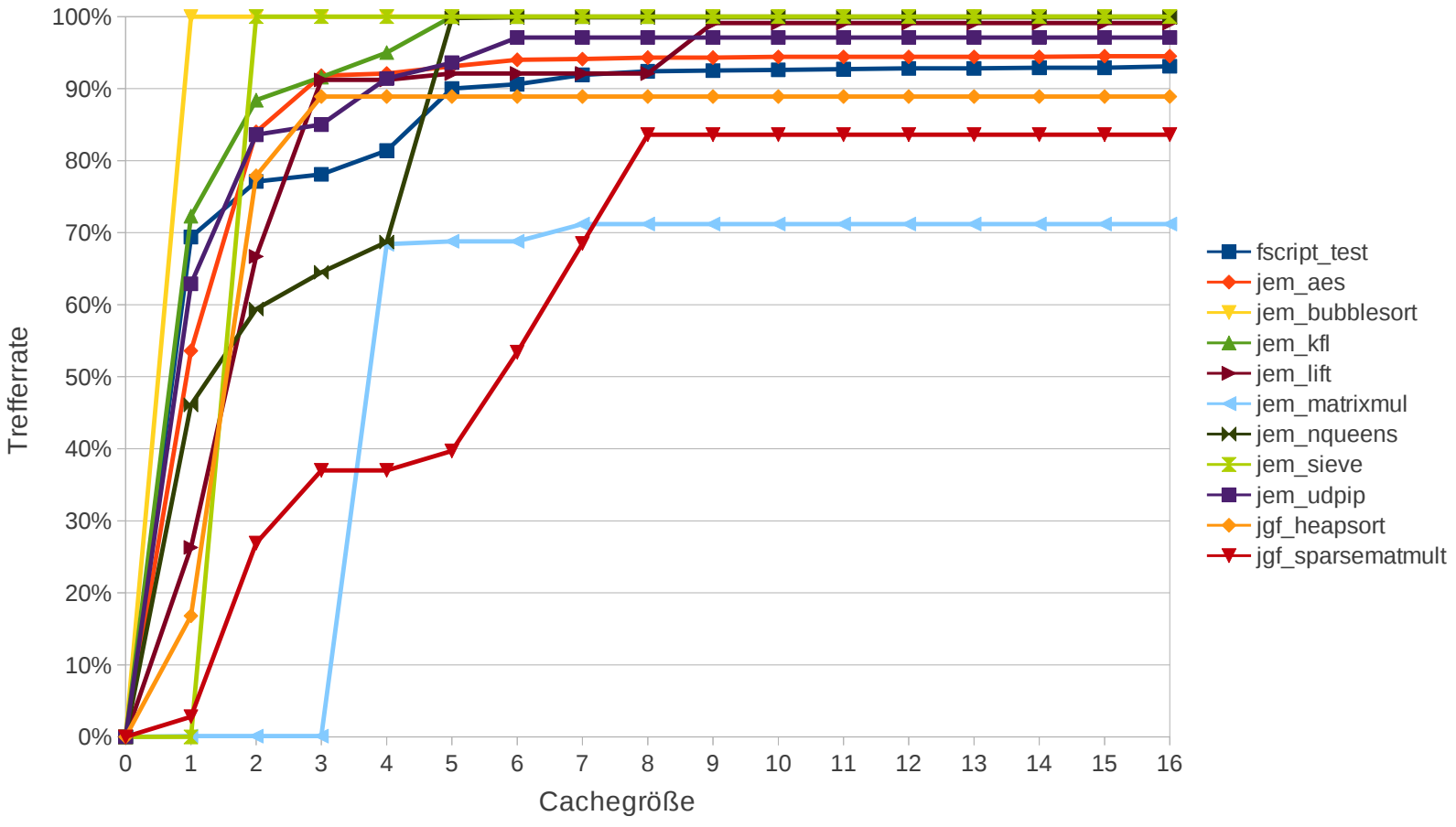
Hardware Concurrent Garbage Collection for Short-lived Objects in an Object-Oriented Processor
Yu Wing Shing, Richard Li, Anthony S. Fong
http://http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01374444

WCET Driven Design Space Exploration of an Object Cache
Benedikt Huber, Wolfgang Puffitsch, Martin Schoeberl
http://www.jopdesign.com/doc/ocwcet.pdf

A Self-Maintained Memory Module Supporting DMM
Weixing Ji, Feng Shi, Baojun Qiao
http://dl.acm.org/ft_gateway.cfm?id=1289916&type=pdf&CFID=40183499&CFTOKEN=20880542

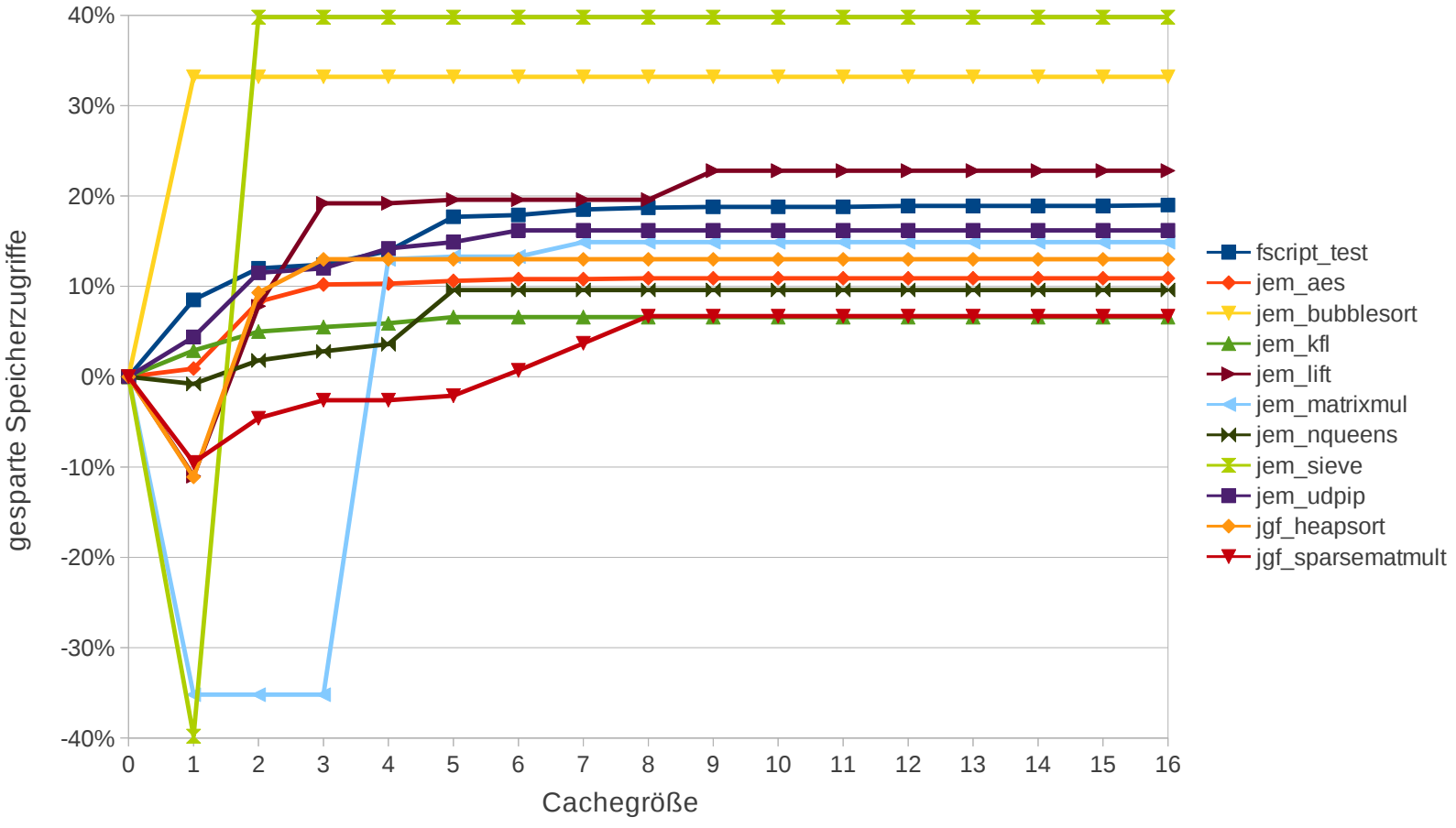
Anhang

Objectcache (Offset -2 und 1) - Trefferrate



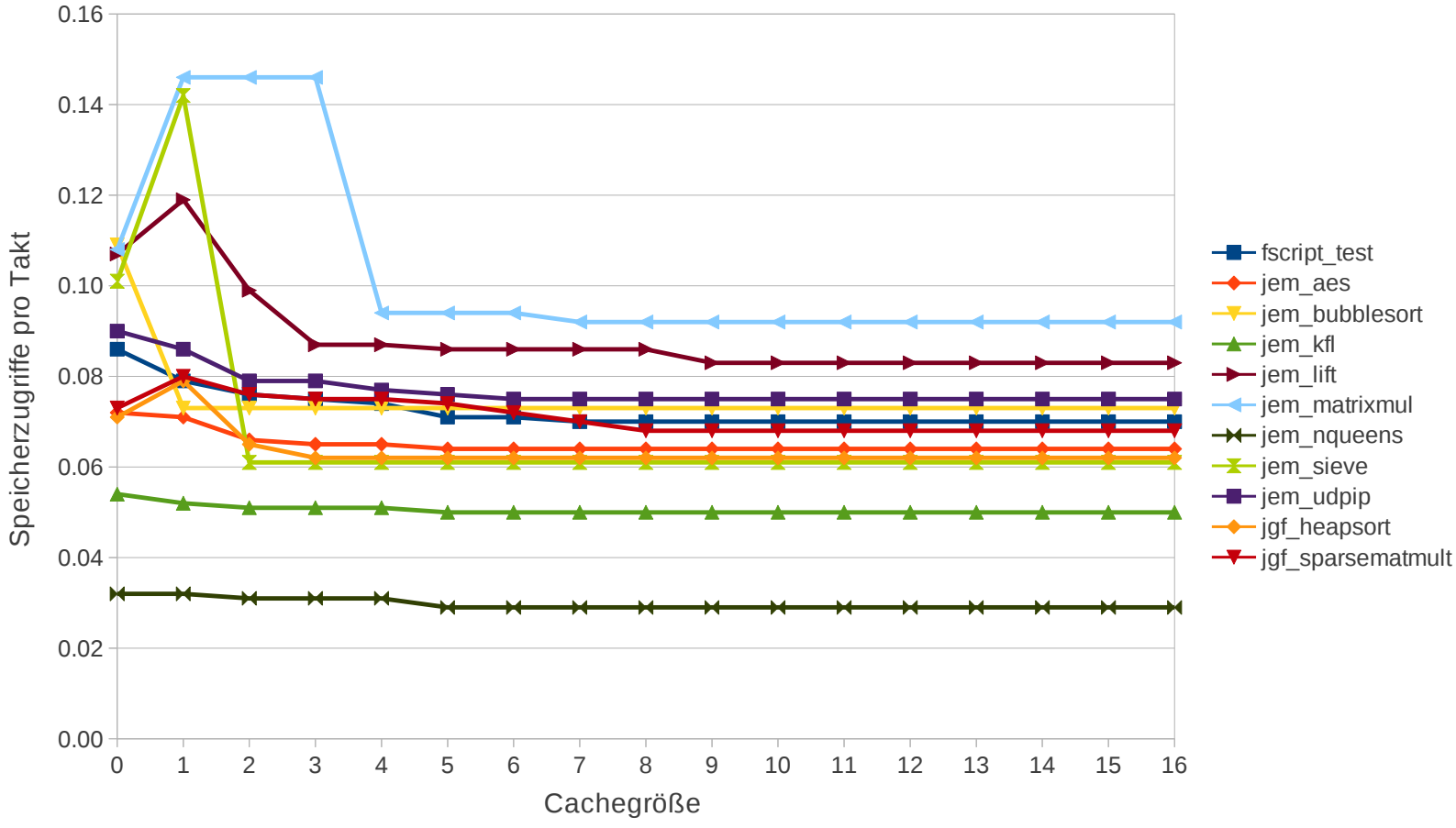
Anhang

Objectcache (Offset -2 und 1) - gesparte Speicherzugriffe



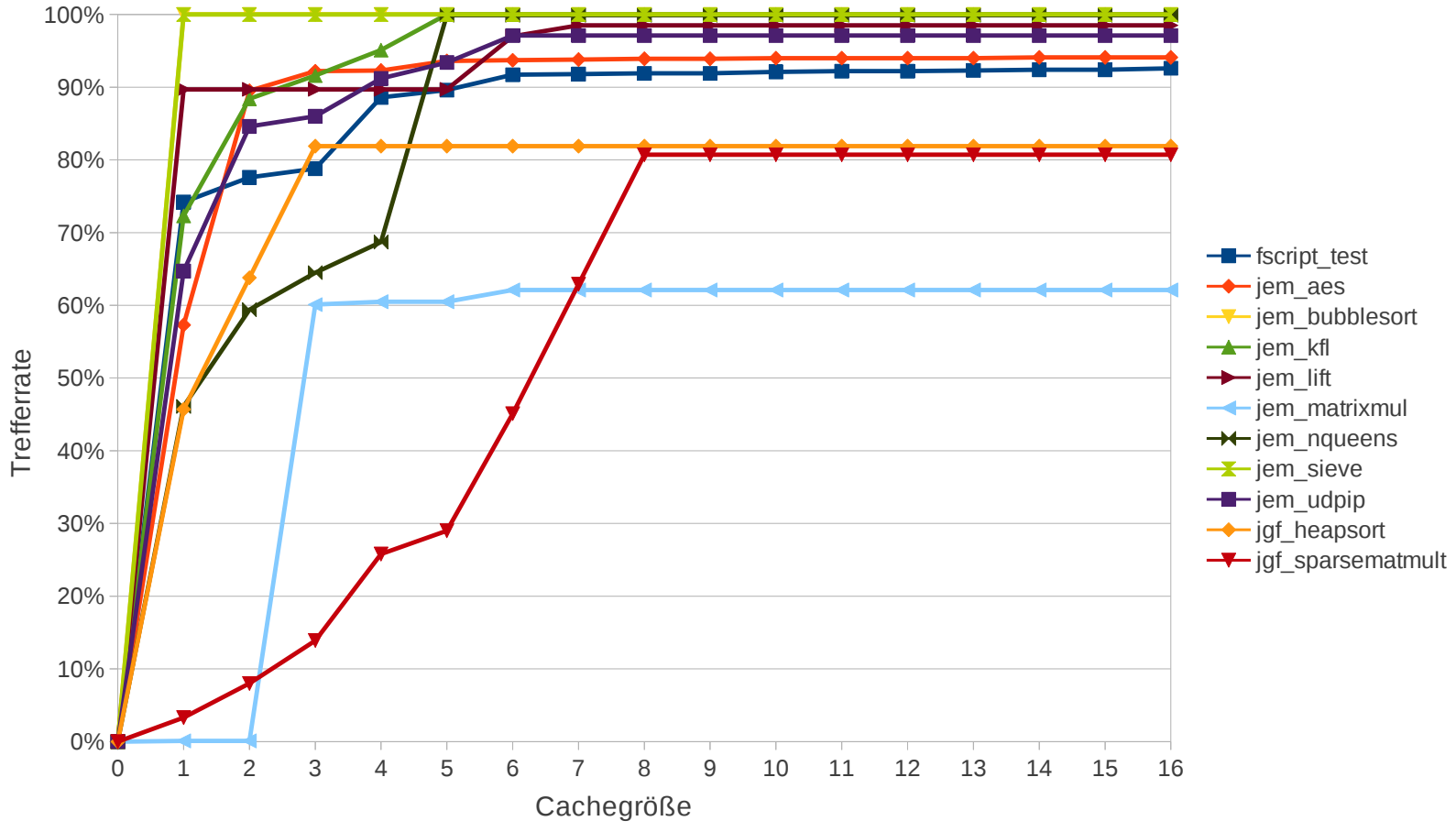
Anhang

Objectcache (Offset -2 und 1) - Speicherzugriffe pro Takt



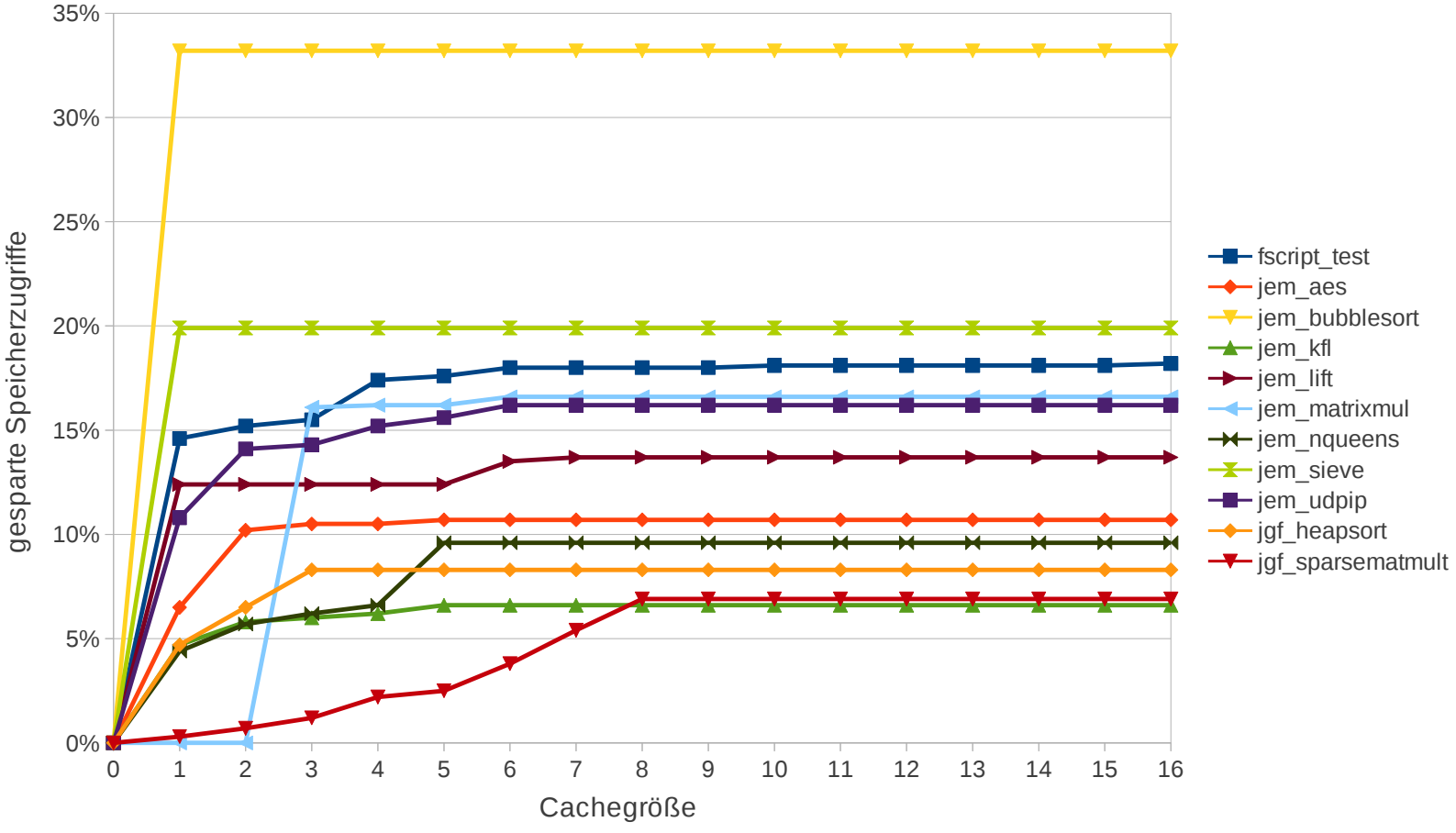
Anhang

Objectcache (Offset 1) - Trefferrate



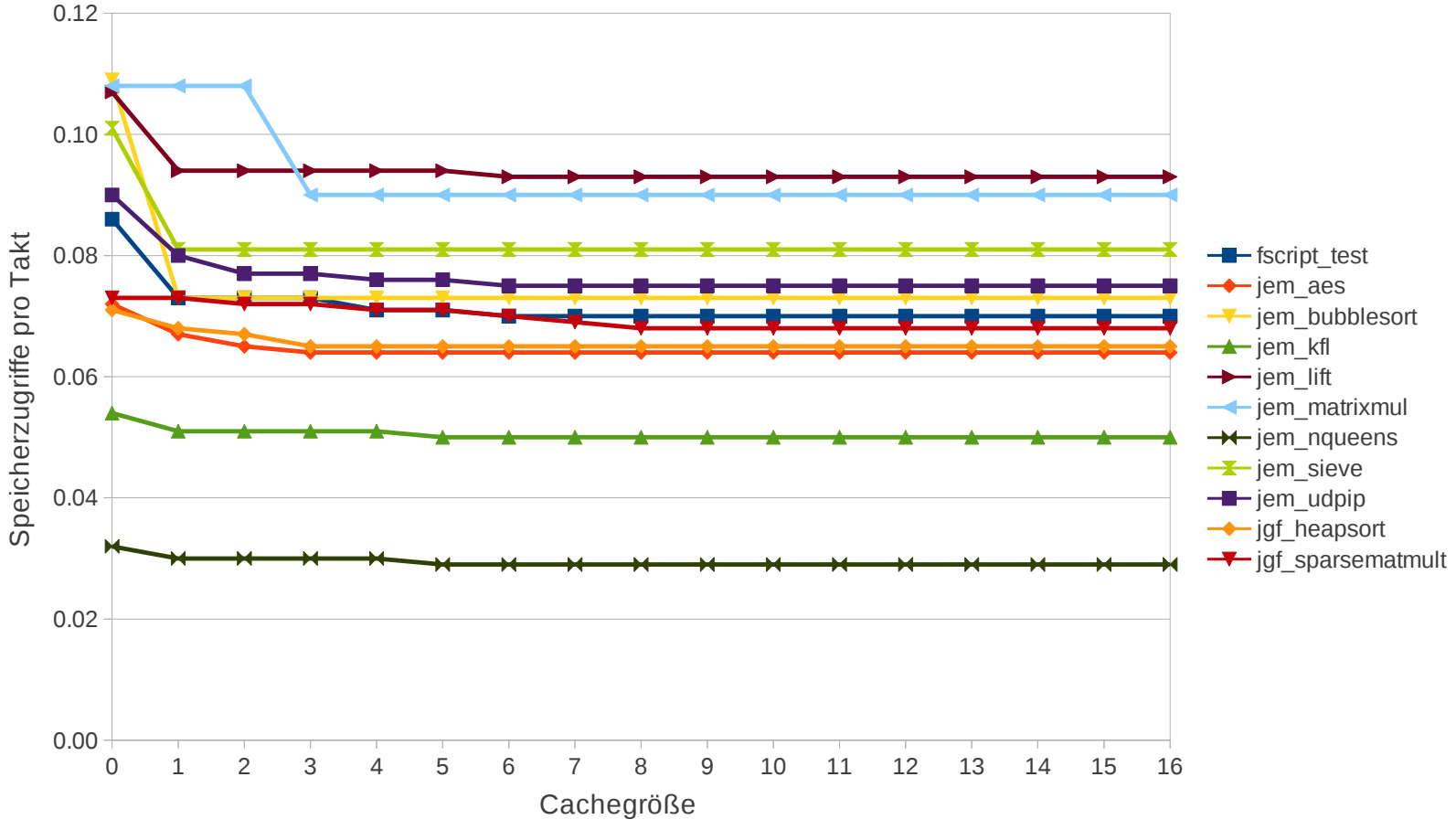
Anhang

Objectcache (Offset 1) - gesparte Speicherzugriffe



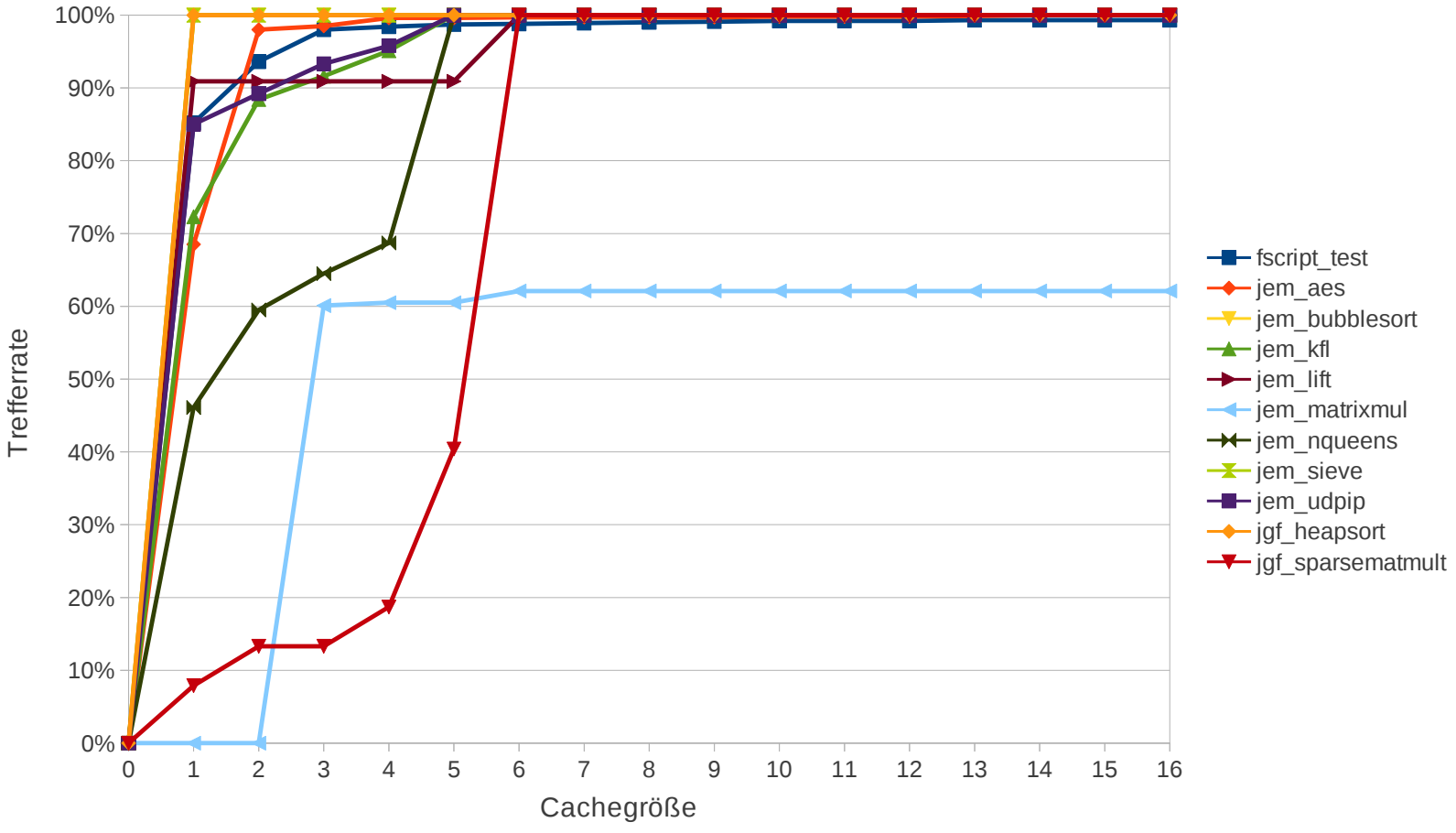
Anhang

Objectcache (Offset 1) - Speicherzugriffe pro Takt



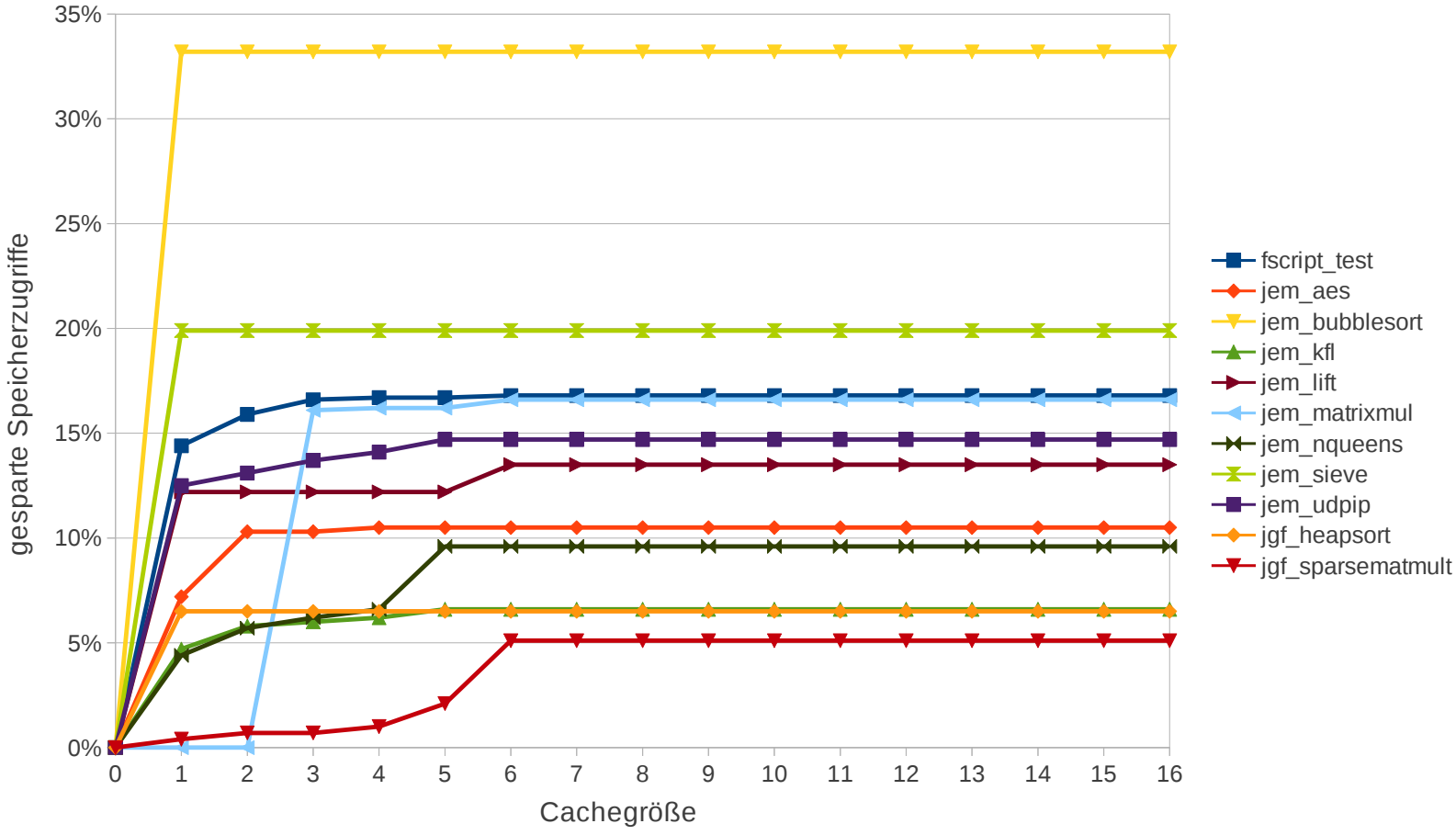
Anhang

Arraylängencache - Trefferrate



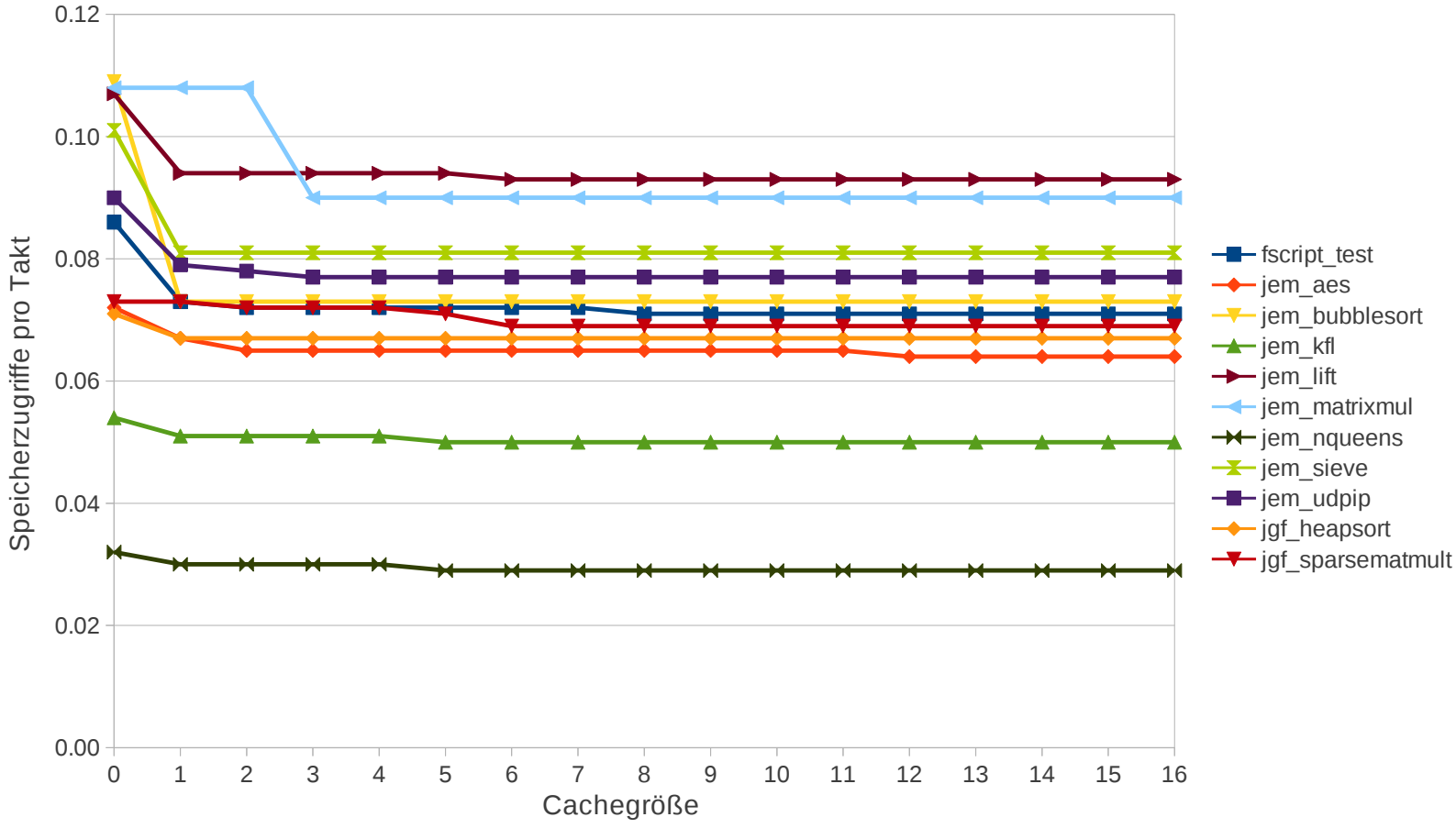
Anhang

Arraylängencache - gesparte Speicherzugriffe



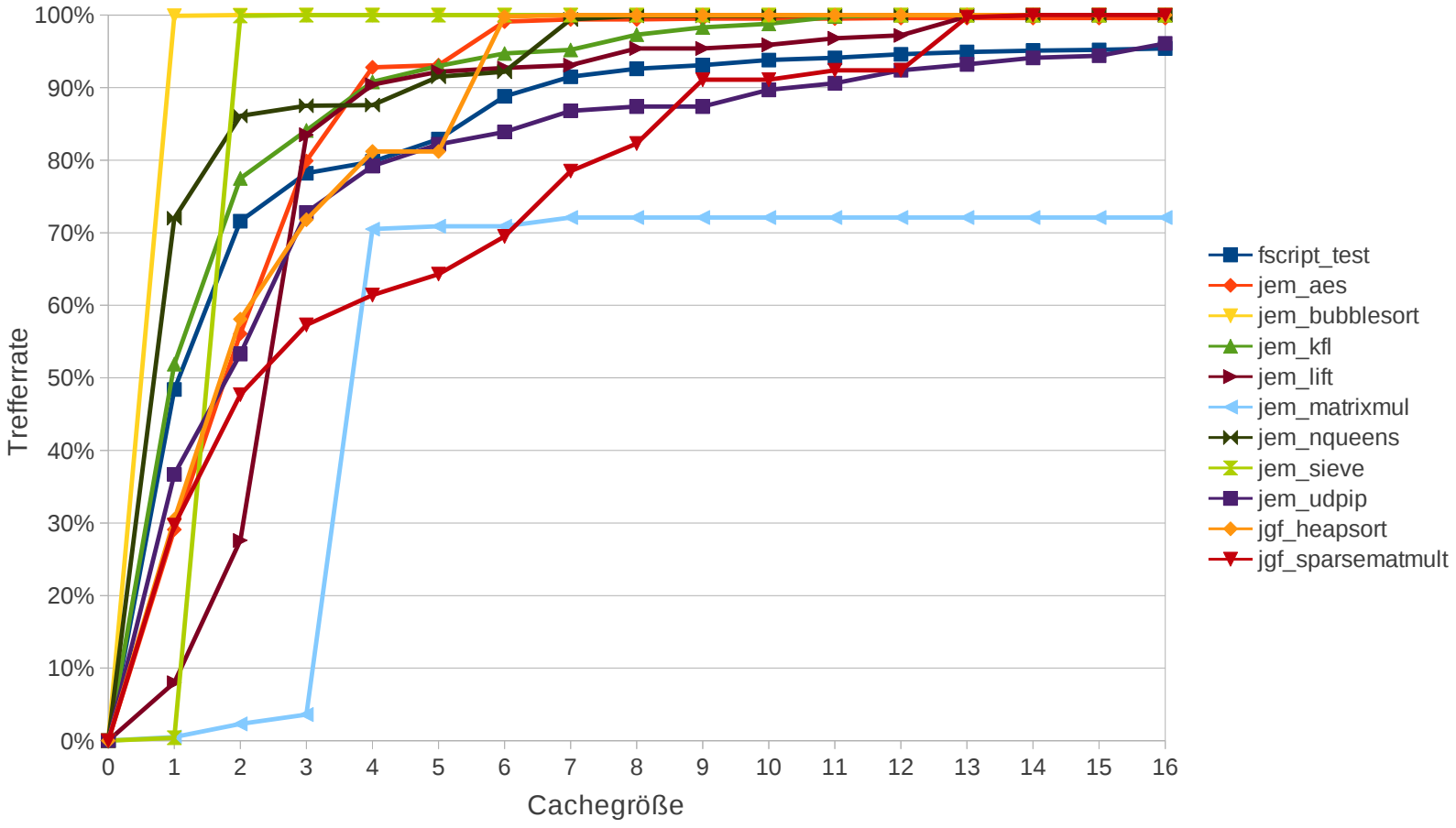
Anhang

Arraylängencache - Speicherzugriffe pro Takt



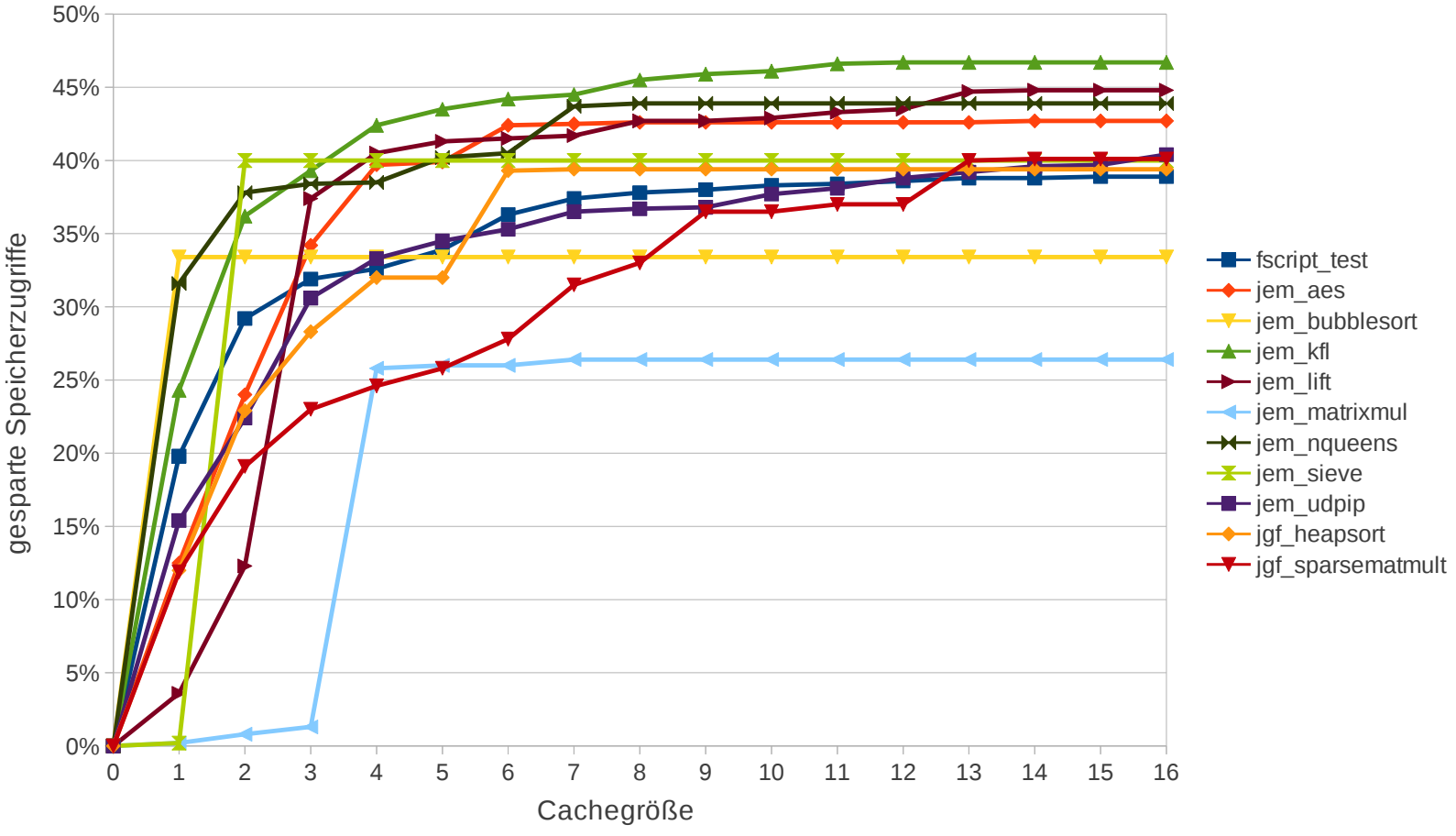
Anhang

TLB - Trefferrate



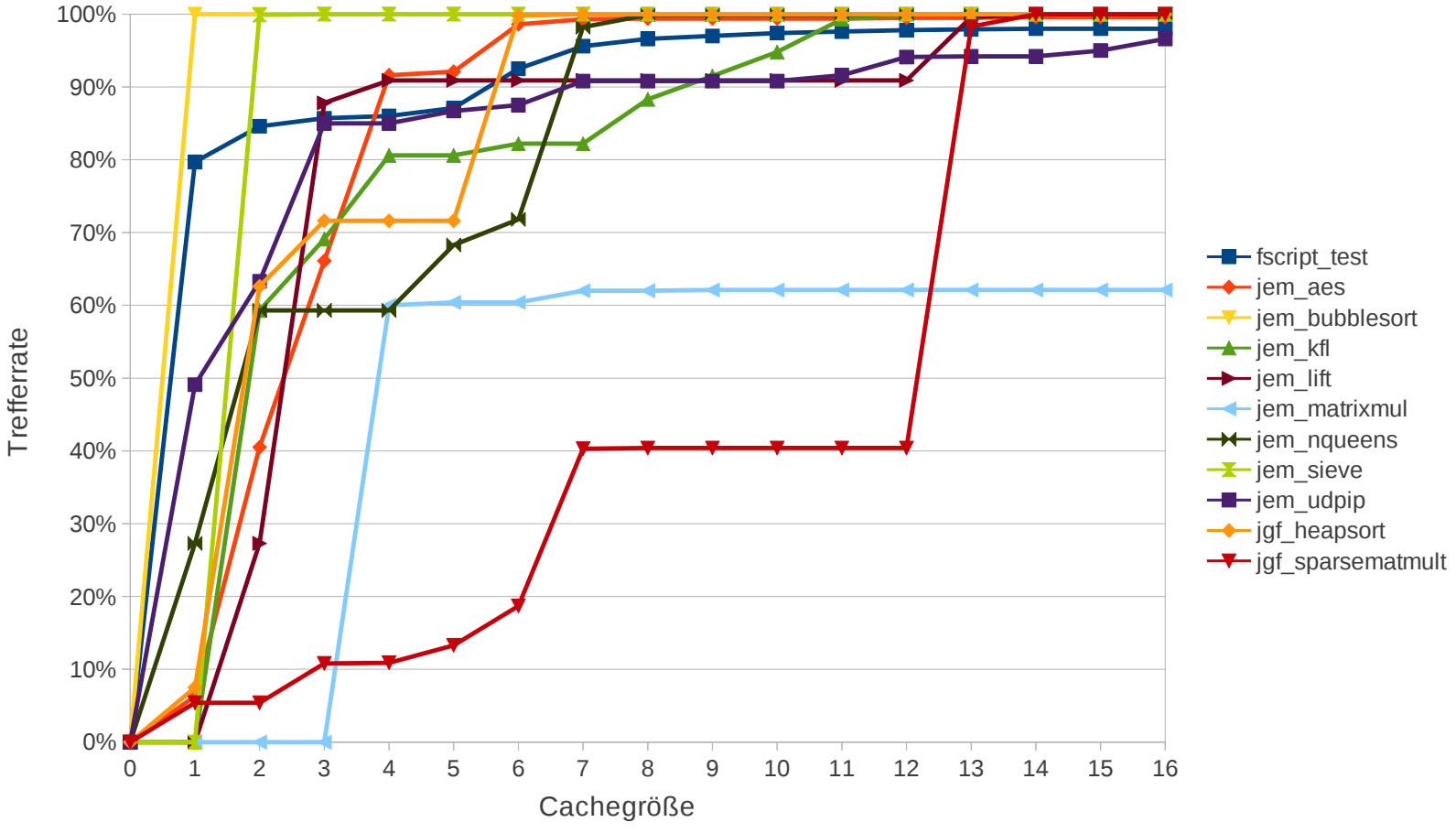
Anhang

TLB - gesparte Speicherzugriffe



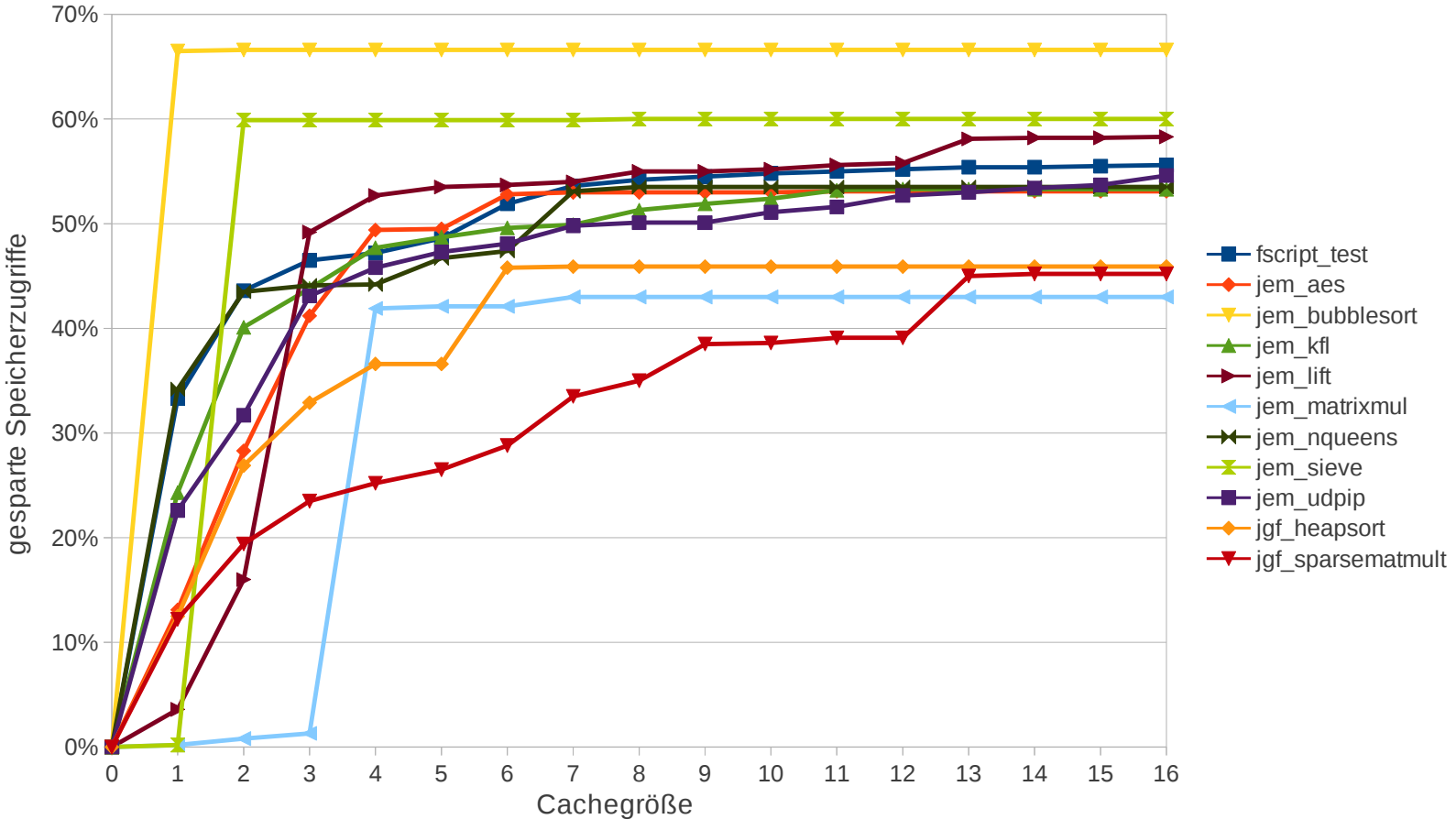
Anhang

Arraylängen-TLB - Trefferrate, AL-Teil



Anhang

Arraylängen-TLB - gesparte Speicherzugriffe



Anhang

Arraylängen-TLB - Speicherzugriffe pro Takt