TECHNISCHE UNIVERSITÄT DRESDEN

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

Simulative Verifikation und Evaluation des Speichermanagements einer Multi-Core-Prozessorarchitektur am Beispiel von SHAP

Christian Greth
christian.greth@mail.inf.tu-dresden.de

Dresden, 09.11.2010

Aufgabenstellung

1. Literaturstudium zu Verifikation und Evaluation von digitalen Systemen mittels Simulation und Trace.
2. Erarbeitung geeigneter Testszenarien für Verifikation der Funktionalität sowie Evaluation der Leistungsfähigkeit.
3. Identifikation von notwendigen Trace-Hardware- und Software-Komponenten unter Berücksichtigung der gegebenen Trace-Infrastruktur.
4. Implementierung und Test der identifizierten Trace-Hardware- und Software-Komponenten.
5. Durchführung und Auswertung der erarbeiteten Testszenarien.
6. Beurteilung des Speichermanagements hinsichtlich korrekter Funktion und Leistungsfähigkeit.
7. Zusammenfassung und Dokumentation der Ergebnisse.

Verifikation digitaler Systeme

Motivation

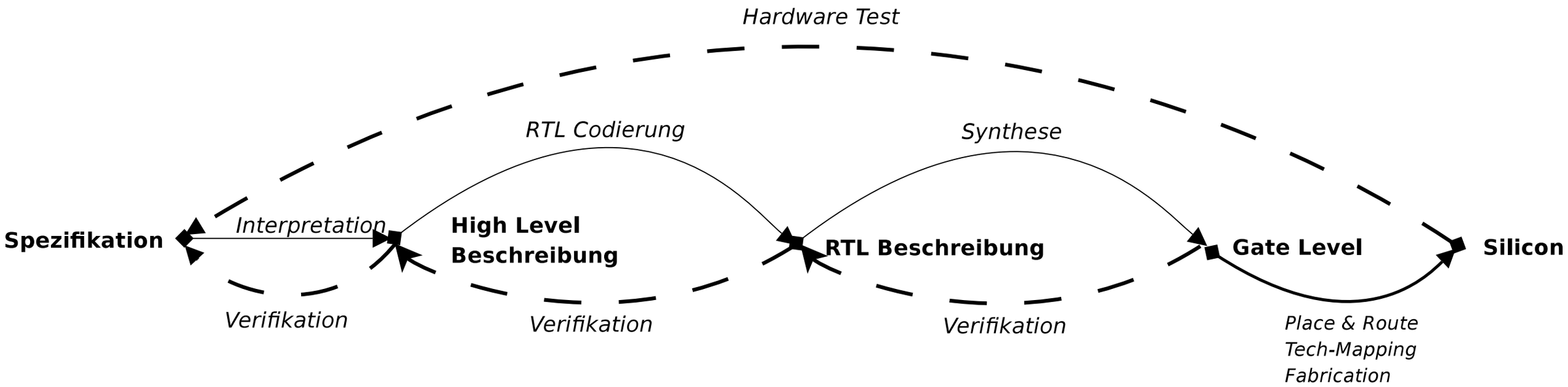
- Überprüfung der Einhaltung von Designregeln.
- Fehlverhalten eines Designs finden und eliminieren.
- Die Funktion/das Verhalten eines Designs mit der Spezifikation abgleichen.

Problem [1]

- 70% des Designprozesses werden für Verifikation/Test beansprucht [2].
- Verifikation bildet den kritischen Pfad.
- Verlust der Kontrolle über Low-Level-Funktionen durch Abstraktion.
  → Abstraktion reduziert jedoch die Dauer der Verifikation.

„Was soll verifiziert werden?“

Verifikation digitaler Systeme



- Verifikation ungleich Test [1]
- Verifikation gegen die Spezifikation [1]

Verifikation digitaler Systeme

Formale Verifikation

- Nutzung mathematischer Methoden zum Beweis der Funktionalität.
- Formale Verifikation kann Fehlerfreiheit eines Entwurfs beweisen.
- Eine formale Beschreibung des zu verifizierenden Systems ist nötig.

Funktionale Verifikation (Validation)

- Emulation
  - Prototyping (FPGA-based prototyping)
- Simulation

Funktionale Verifikation

Prototyping (FPGA-based prototyping)

- Ist schneller als eine Simulation.
- Bietet nur wenige Möglichkeiten für Debugging.
- Debugging kann sehr aufwendig werden.

Simulation

- Hardware-Beschreibung wird auf eine Zielarchitektur abgebildet.
- Hoher Grad an Parallelität muss auf eine sequentielle Maschine „projeziert“ werden.
- Signale können zu jeder Zeit beobachtet werden.
- Setzen von Breakpoints und Start/Stop/Resume der Simulation ist möglich.
- Bietet die Möglichkeit einer Überdeckungsanalyse.

Leistungsevaluation digitaler Systeme

Motivation

- Leistungsfähigkeit eines Systems ermitteln/zeigen und anschließendes Bewerten.
- Vergleichsmöglichkeiten zu anderen/ähnlichen Systemen schaffen.
- Eventuelle Engpässe (bottlenecks) finden und beheben.

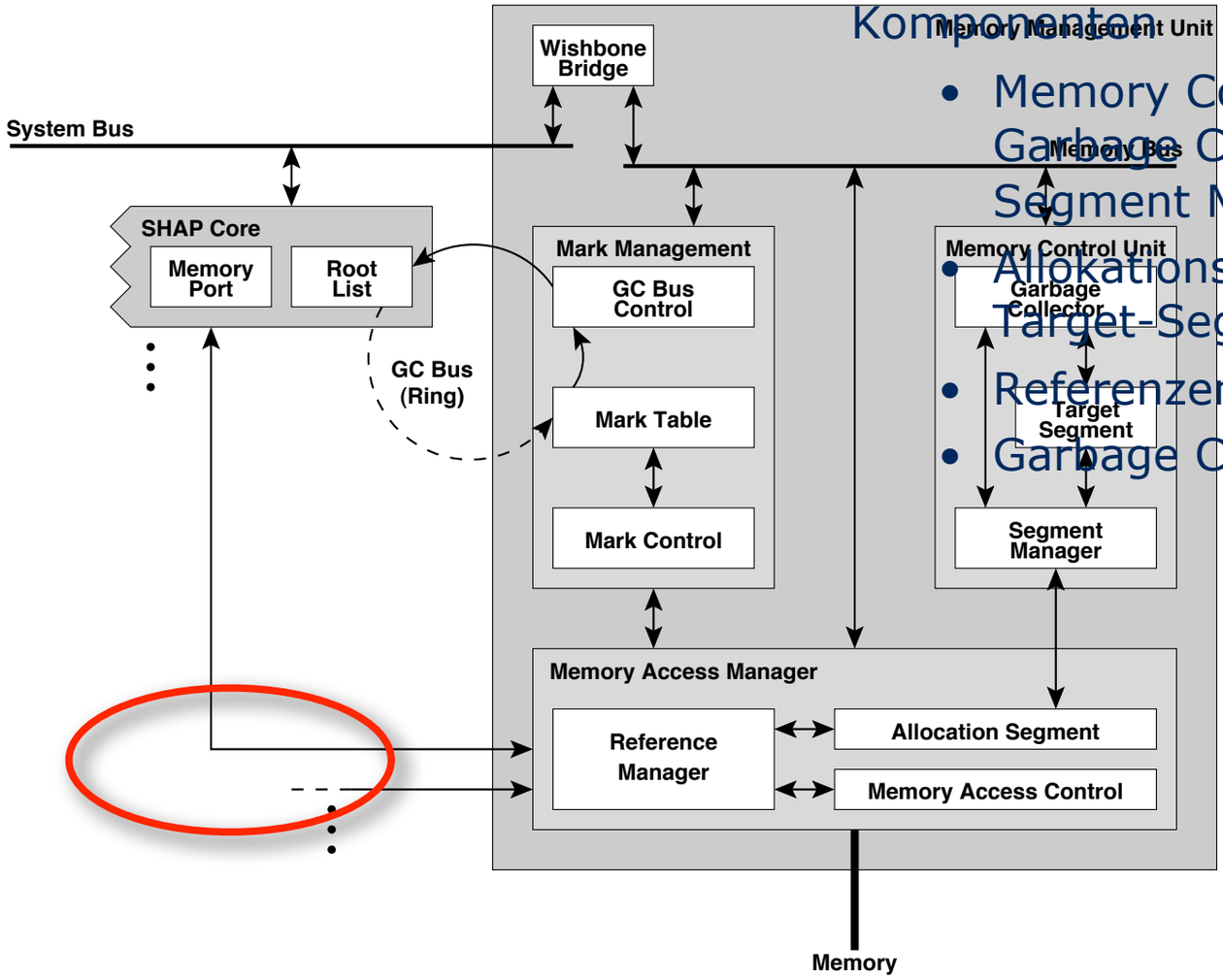
Metriken und Methoden

- Latenz
- Durchsatz
  - → Objektallokationen
  - → Objektzugriffe (Lese-/Schreibzugriffe)

Bewertungskriterien

- Maximum und Minimum
- Durchschnitt

SHAP - Speichermanagement



Überblick MMU

Komponenten

- Memory Control Unit mit Garbage Collector und Segment Manager
- Allokations-Segment und Target-Segment
- Referenzen Manager
- Garbage Collector Bus

Testumgebung und -szenarien

Testumgebung

- FPGA für Prototyp-Test (z.B. Evaluation Board ML505 von Xilinx) und
- SHAP mit integrierter Trace-Hardware.

Hardware

- Tracer-Modul als Schnittstelle zwischen SHAP-Komponenten und Trace-Hardware und
- als Trigger zum Aufzeichnen der Trace-Daten.

Software

- Test-Applikation programmiert in Java und
- eventuell Script um gewonnene Trace-Daten auszuwerten.

Verifikation - Referenzen Manager

Funktion

- Verwaltung aller Referenzen und deren Informationen von
- Objekt-Allokation bis Objekt-Freigabe und
- Verwaltung des freien Speichers im Allokations-Segment.

Komponente	Kommando	Codierung	
		binär	hexadezimal
Core	NEWREF	1111	F
	VIVREF	1110	E
	CHKWEAK	1101	D
GC	GETDATA	0010	2
	SETDATA	0011	3
	SCANREF	0100	4
	MOVEREF	0101	5
	FREEREF	0110	6

Kommandos *refman*

	Port 1	Port 2	Port 3
Inhalt	Kommando	Referenz	Daten
Beschreibung	das aktuelle verarbeitete Kommando	die aktuell verarbeitete Referenz	frei definierbare Daten
Beispiel	0x5	0x0123	0x3123

Tracer-Modul *refman*

Verifikation - Referenzen Manager

codiertes Kommando

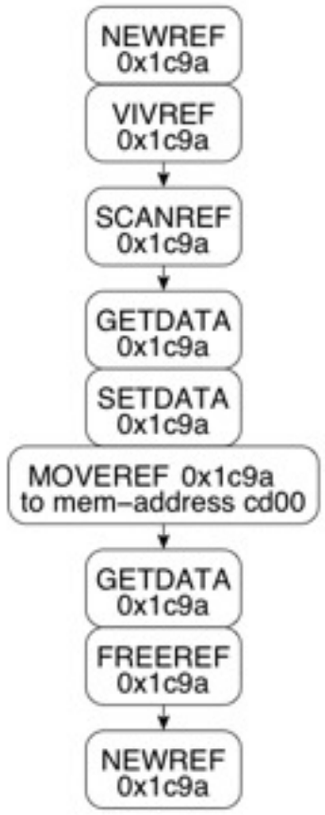
Referenz

Nachrichten-Nummer

4 Byte zusätzliche Daten

```
8635,f,1c9a,0000
8636,e,1c9a,0000
⋮
9678,4,1c9a,0000
⋮
10835,2,1c9a,0000
10836,3,1c9a,0000
10837,5,1c9a,cd00
⋮
27449,2,1c9a,0000
27450,6,1c9a,0000
⋮
27487,f,1c9a,0000
⋮
```

Auszug aus Trace-Daten *refman*



Interpretation der Trace-Daten

Verifikation - Referenzen Manager

Überprüfung durch Perl-Script

- Verlauf der Referenzen,
- Verwaltung der Referenzen-Listen und
- Einhaltung bestimmter Kommando-Abfolgen.

Komponente	Kommando	Codierung	
		binär	hexadezimal
Core	NEWREF	1111	F
	VIVREF	1110	E
	CHKWEAK	1101	D
GC	GETDATA	0010	2
	SETDATA	0011	3
	SCANREF	0100	4
	MOVEREF	0101	5
	FREEREF	0110	6

Kommandos *refman*

Verifikation weiterer Komponenten

Kombination aus Tracer-Modul und Perl-Script

- Allokations-Segment,
- Garbage Collector Bus und
- Zugriffe der Cores in Objekte

statische Codeanalyse

- Garbage Collector und
- Segment-Manager

Code-Coverage-Methoden

- VHDL-Quellcode von SHAP

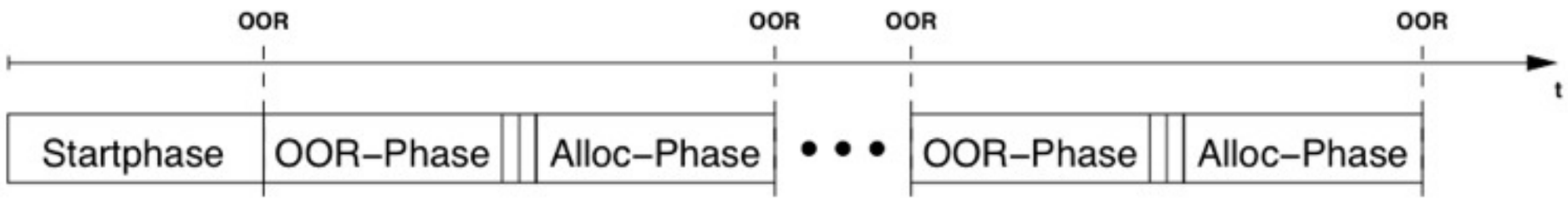
Evaluation - maximale Auslastung

Standardkonfiguration des Speichermanagements

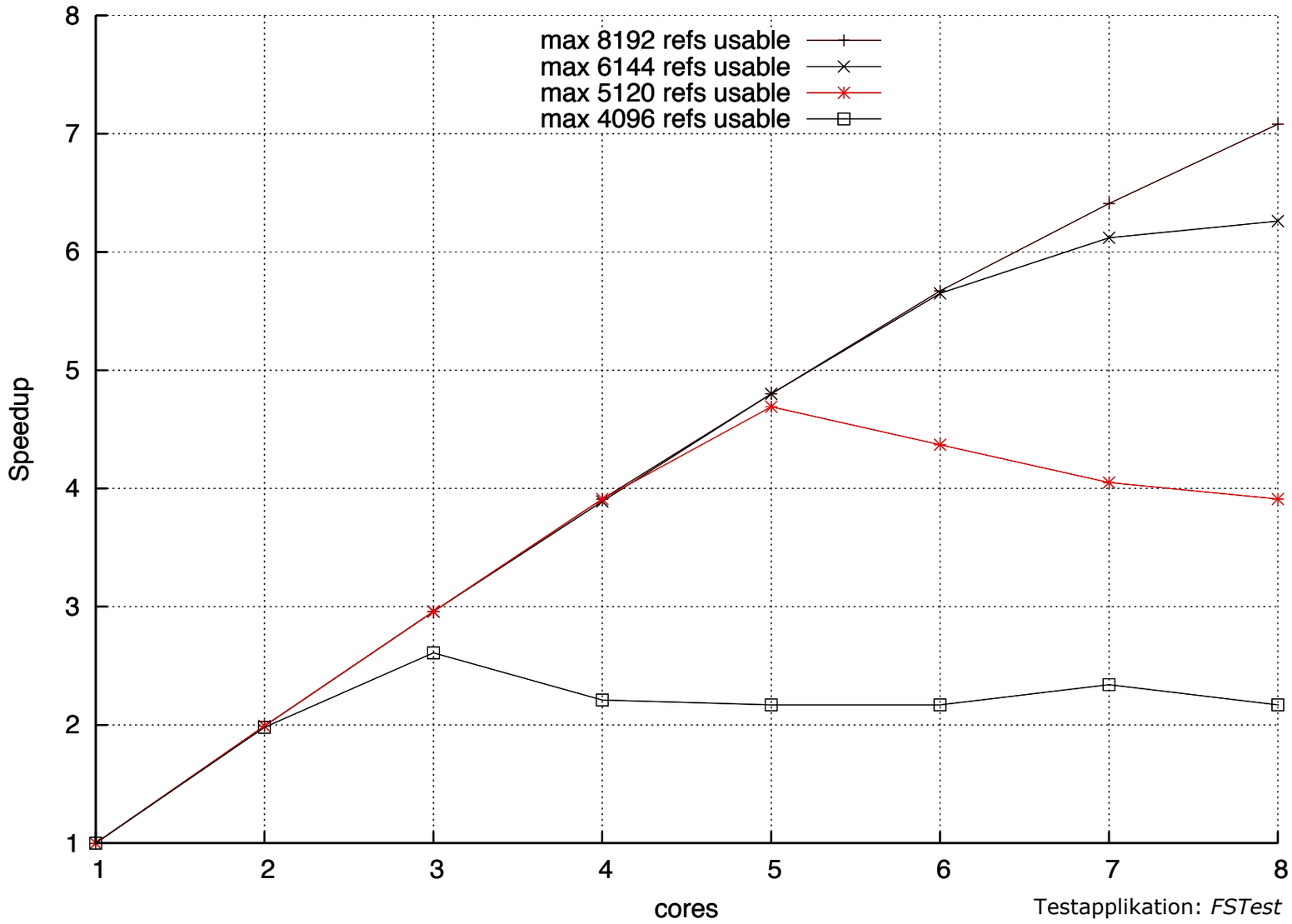
- maximal 8192 Referenzen gleichzeitig nutzbar
- 1MB Heap verteilt auf 16 Segmente mit je 64KByte
- Objekte sind organisiert in Speicherworten zu je 4Byte

Problem

- Sind 8192 Objekte allokiert so kommt es zum „OutOfRef".
  - ➡ Cores können nicht weiter allokieren und werden blockiert.
  - ➡ Cores müssen warten bis GC Referenzen freigegeben hat.



Evaluation - maximale Auslastung

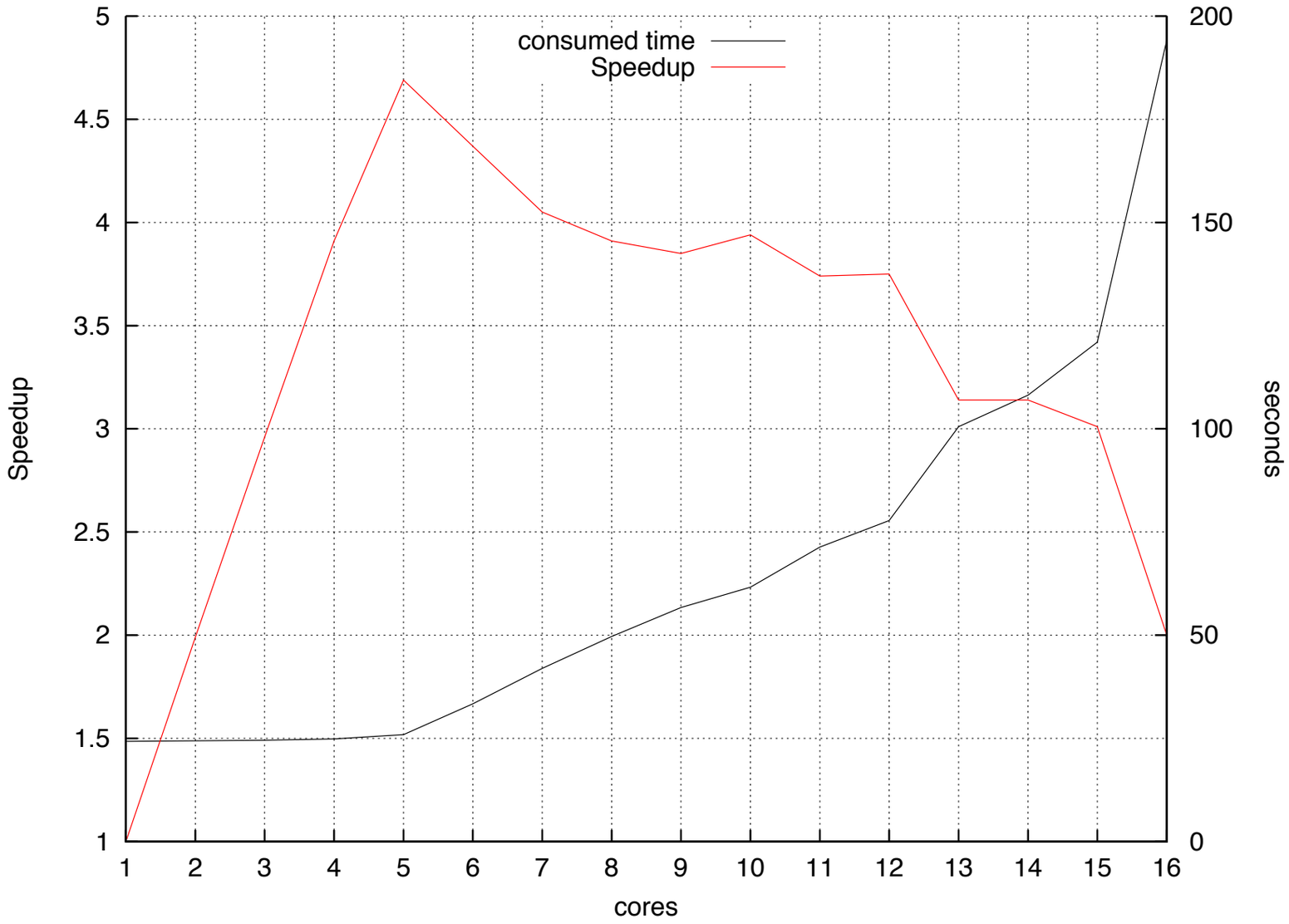


Testapplikation: *FSTest*

$$Sp = \frac{T_1}{T_n} * n = n * d$$

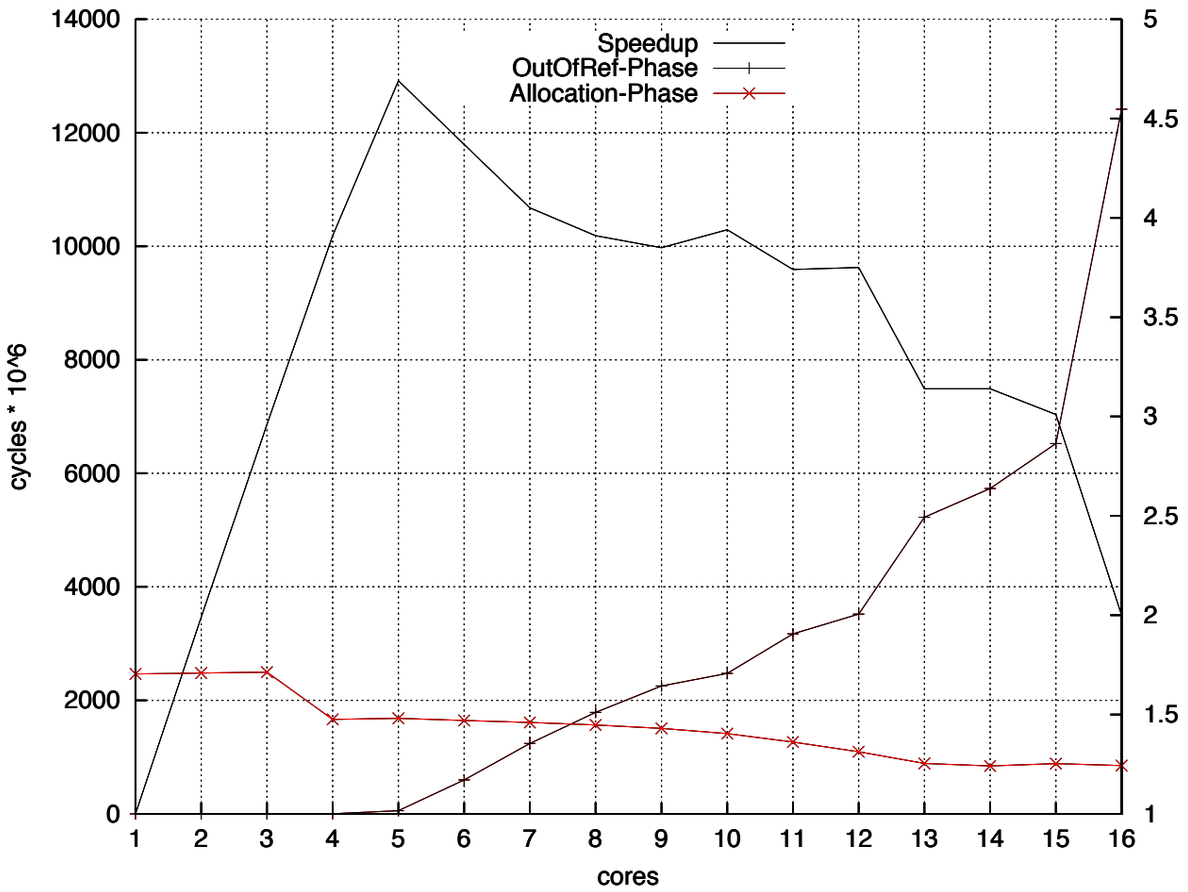
$$d = \min\left\{1, \frac{m}{a * n}\right\}$$

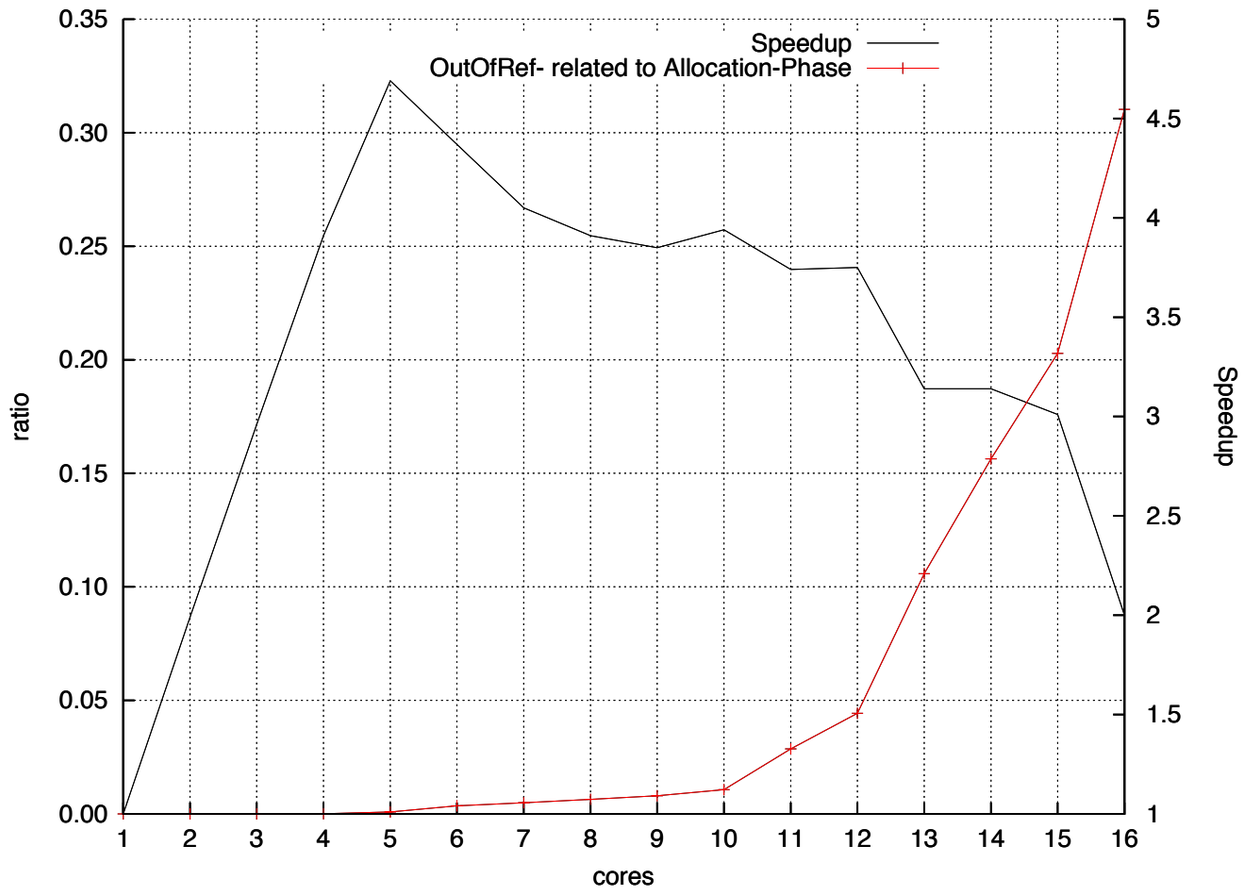
Evaluation - maximale Auslastung



Testapplikation: *FSTest; GC-Konfiguration: maximal 5120 Referenzen*

Evaluation - maximale Auslastung



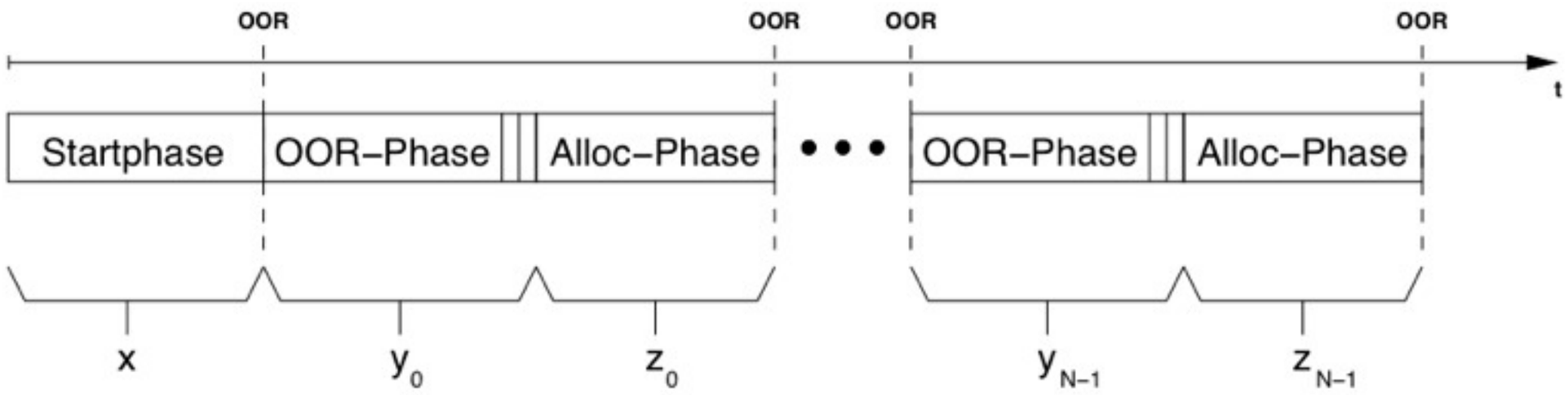


Testapplikation: *FSTest; GC-Konfiguration: maximal 5120 Referenzen*

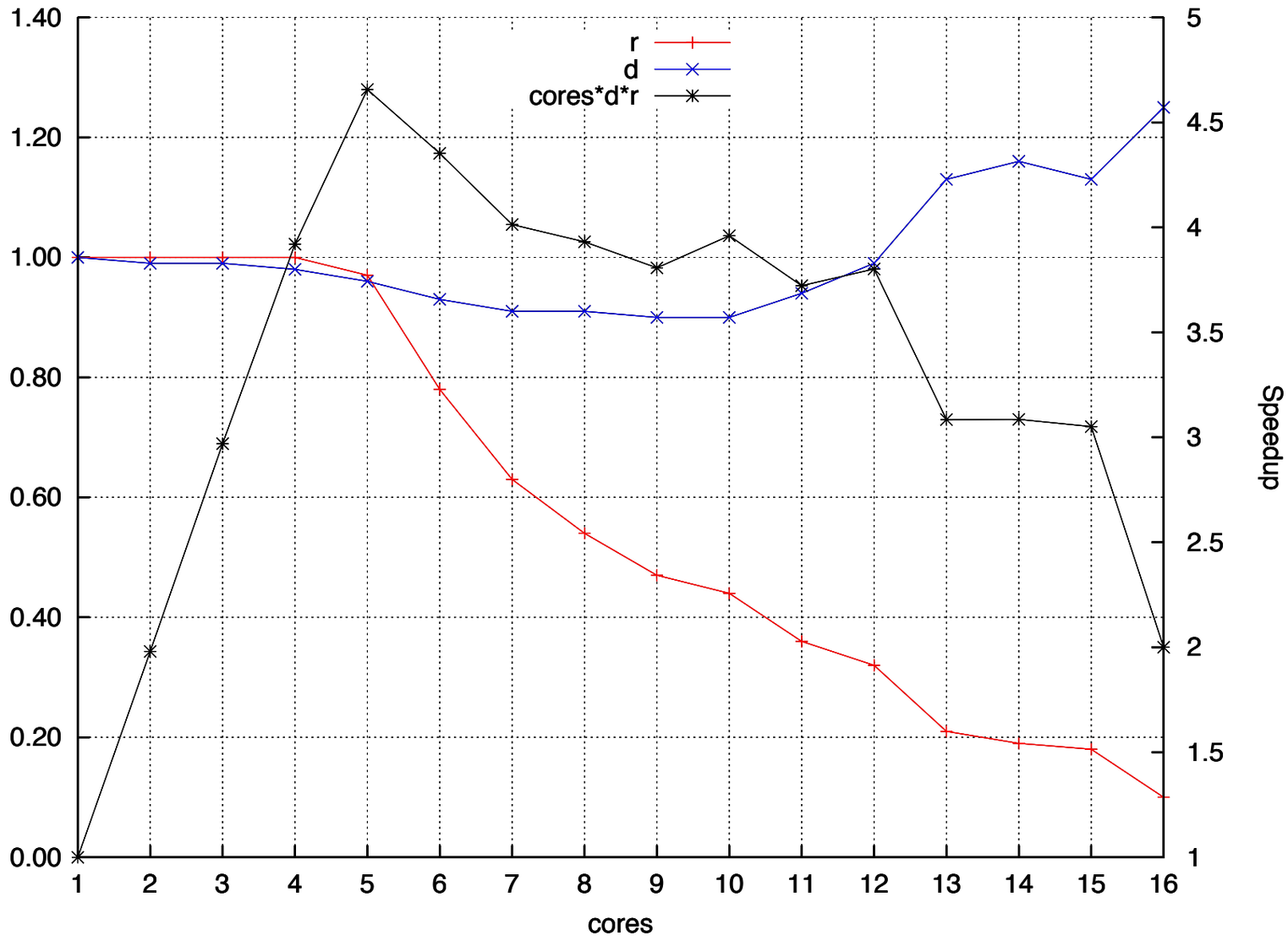
Evaluation - maximale Auslastung

$$Sp = n * d * r$$

$$r = \frac{x + \sum_{i=0}^{N-1} z_i}{x + \sum_{i=0}^{N-1} (y_i + z_i)}$$



Evaluation - maximale Auslastung



Testapplikation: *FSTest;* GC-Konfiguration: *maximal 5120 Referenzen*

Evaluation

Referenzen-Manager

- Häufigkeiten aller Kommandos unter verschiedenen Szenarien
- zeitlich gewichtete Häufigkeiten aller Kommandos

Memory Control Unit

- Dauer und Häufigkeit der GC-Zyklen durch verschiedene Szenarien
- Verhalten von Allokations- und Target-Segment
- Objektverschiebungen

Objektallokationen und -zugriffe

- maximale Allokationsraten, Schreiberaten und Leseraten
- Einfluss des GC auf Objektallokationen

Zusammenfassung

Verifikation

- zwei gefundene Fehlerverhalten wurden bereits korrigiert
- eine Verbesserung im Referenzenmanagement wurde implementiert
- alle Hardware-Komponenten sind unter den entwickelten Testszenarien fehlerfrei

Evaluation

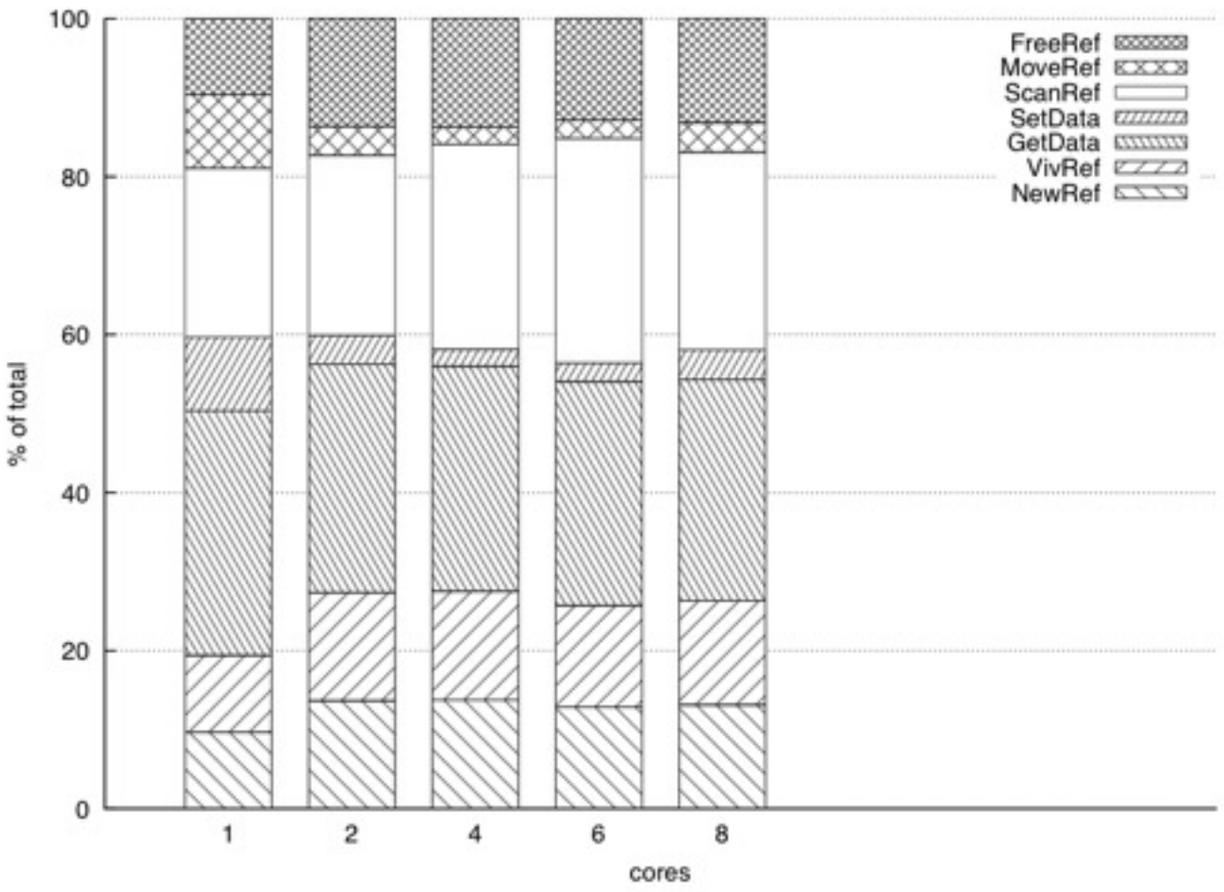
- Dimensionierung des Speichermanagement ist für aktuelle Szenarien ausreichend
- Szenarien am Maximum der Ressourcen werden stark ausgebremst

Ausblick

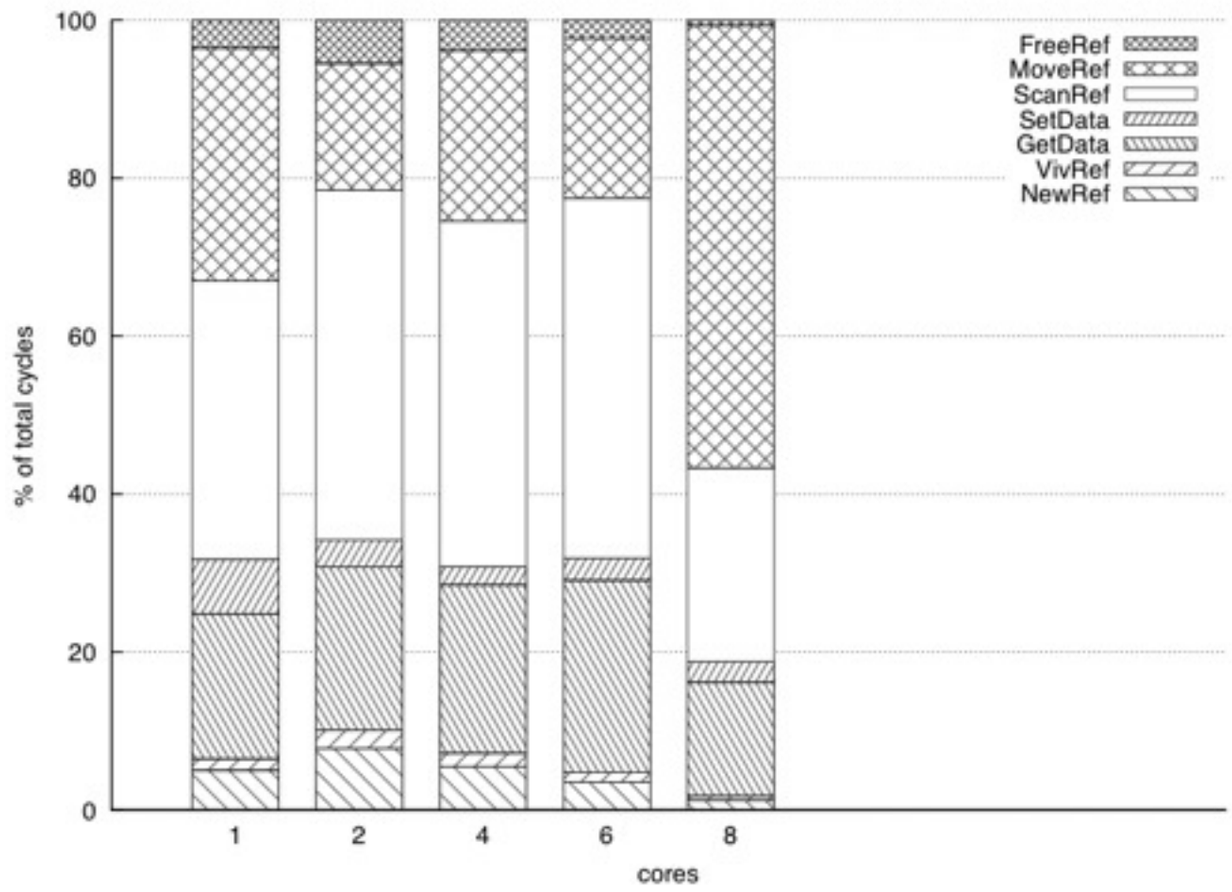
- detailliertere Verifikation der Hardware-Komponenten
- Erweiterung der Trace-Architektur (Komprimierung)
- weitere Betrachtungen am Maximum der Speichermanagement-Ressourcen

Anhang

Evaluation Referenzen Manager



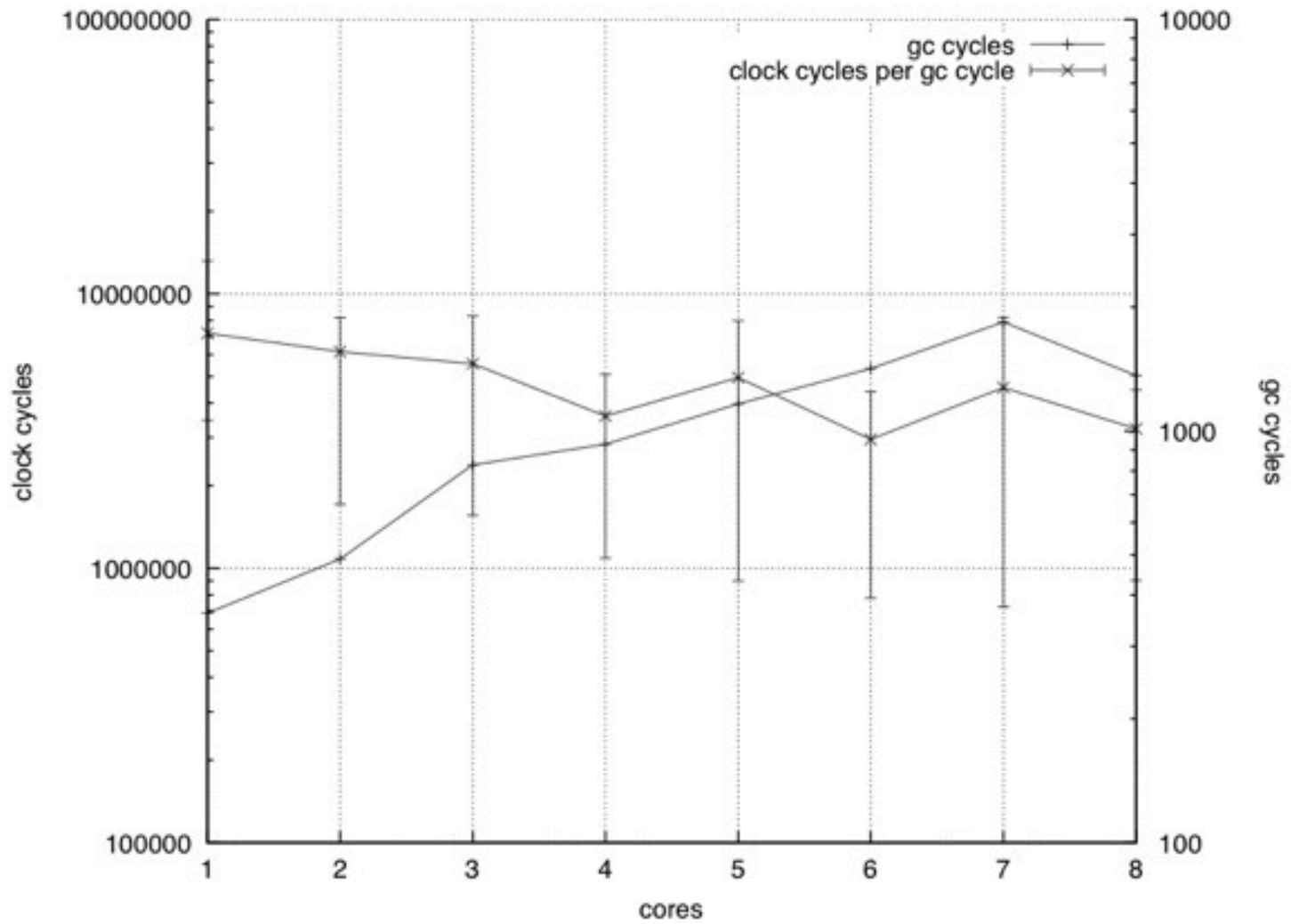
prozentuale Häufigkeit der Kommandos durch *FSTest*



zeitlich gewichtete prozentuale Häufigkeit der Kommandos durch *FSTest*

Anhang

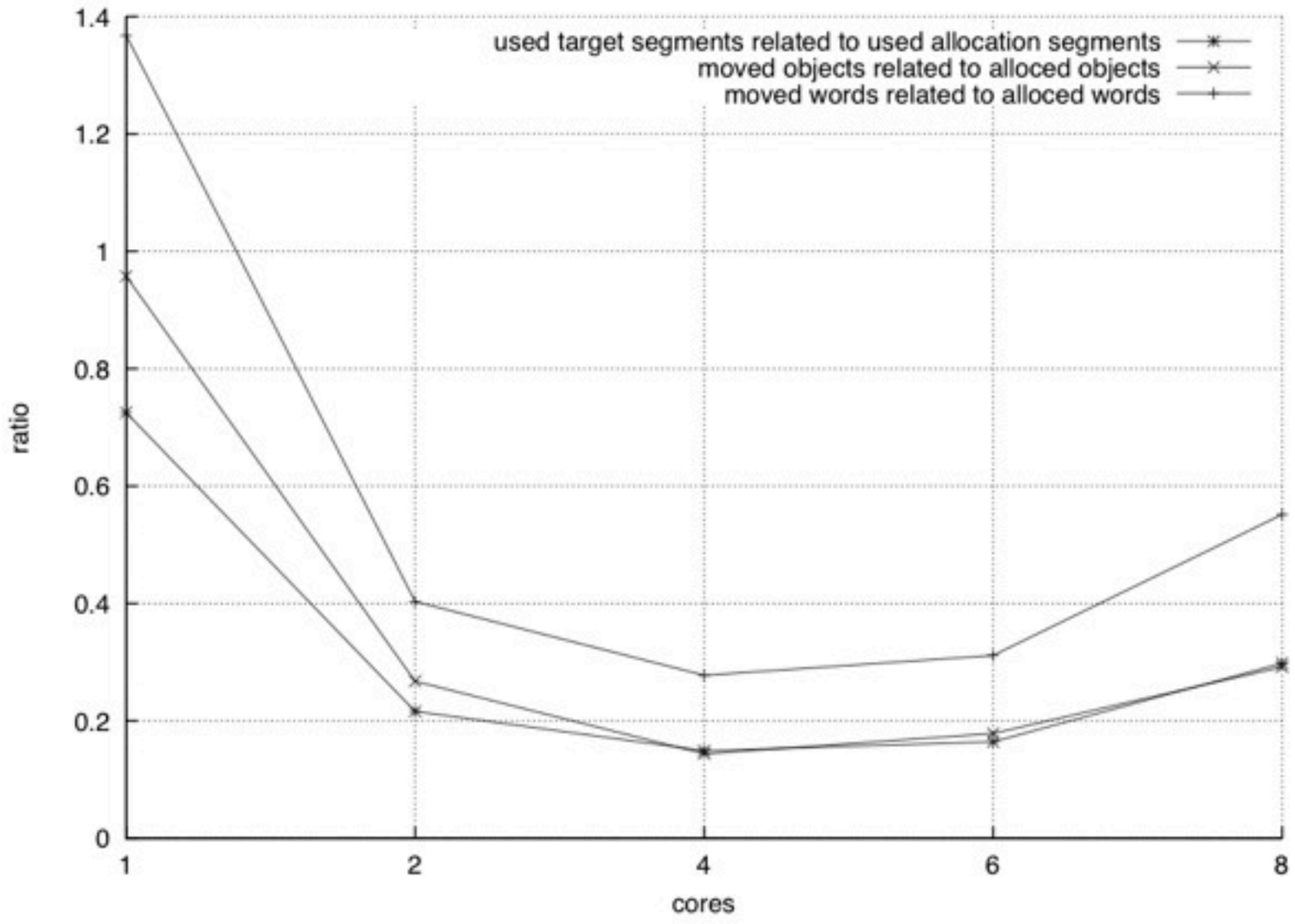
Evaluation GC-Zyklen



Anzahl und Dauer GC-Zyklen durch *FSTest*

Anhang

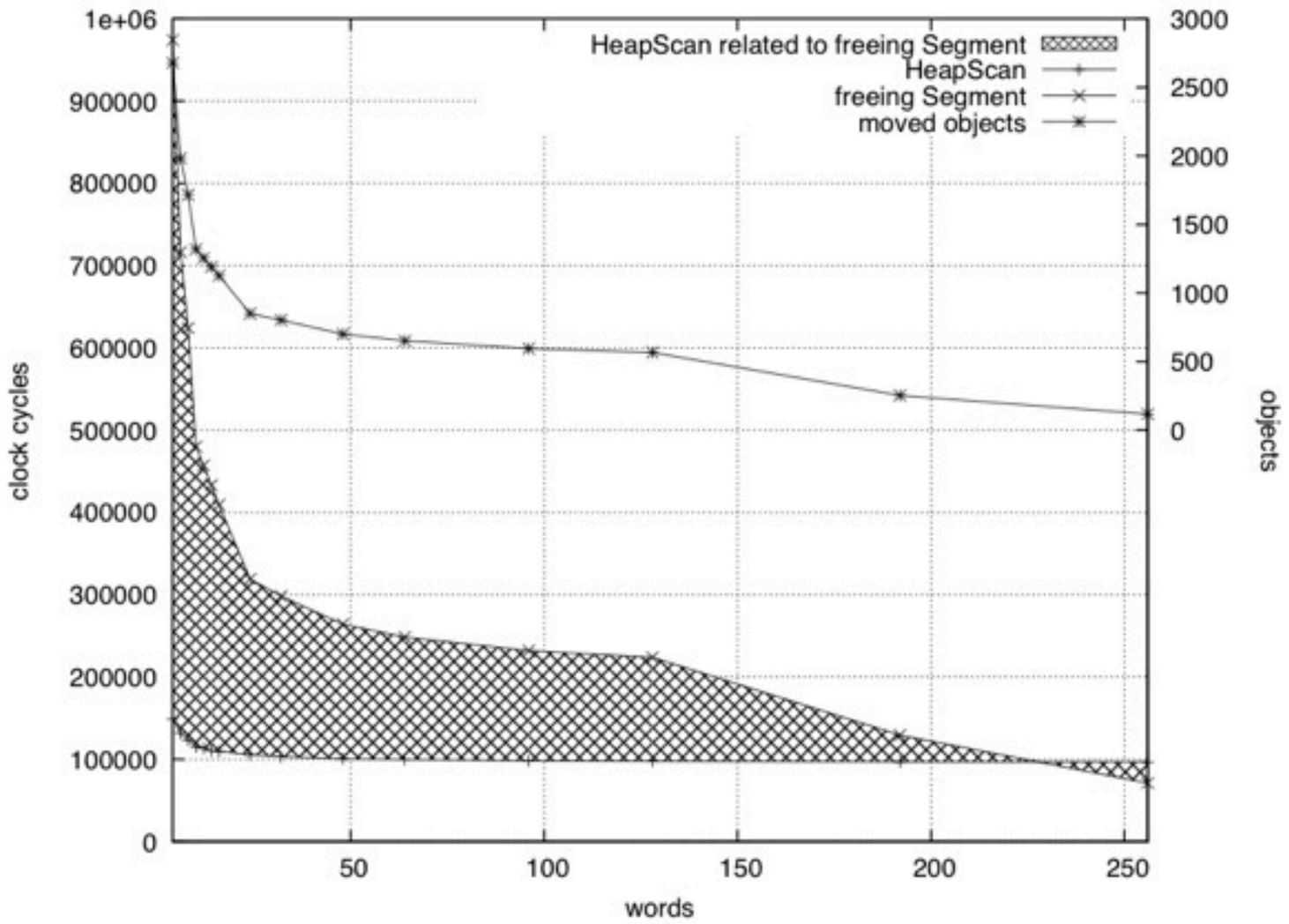
Allokations- und Target-Segment



Verhalten von Allokations- und Target-Segment durch *FSTest*

Anhang

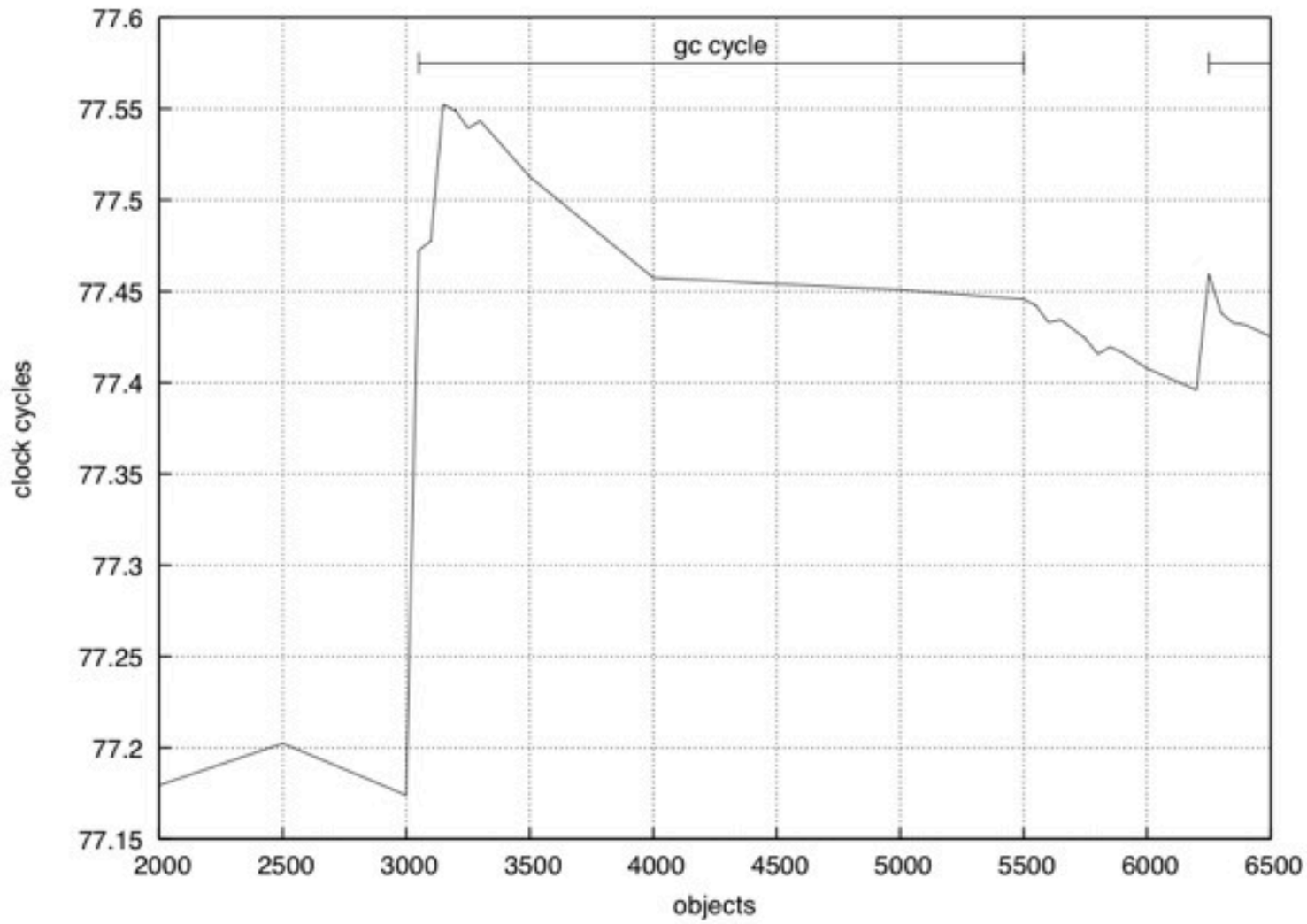
HeapScan und Segmentfreigabe



HeapScan und Segmentfreigabe im zeitlichen Vergleich

Anhang

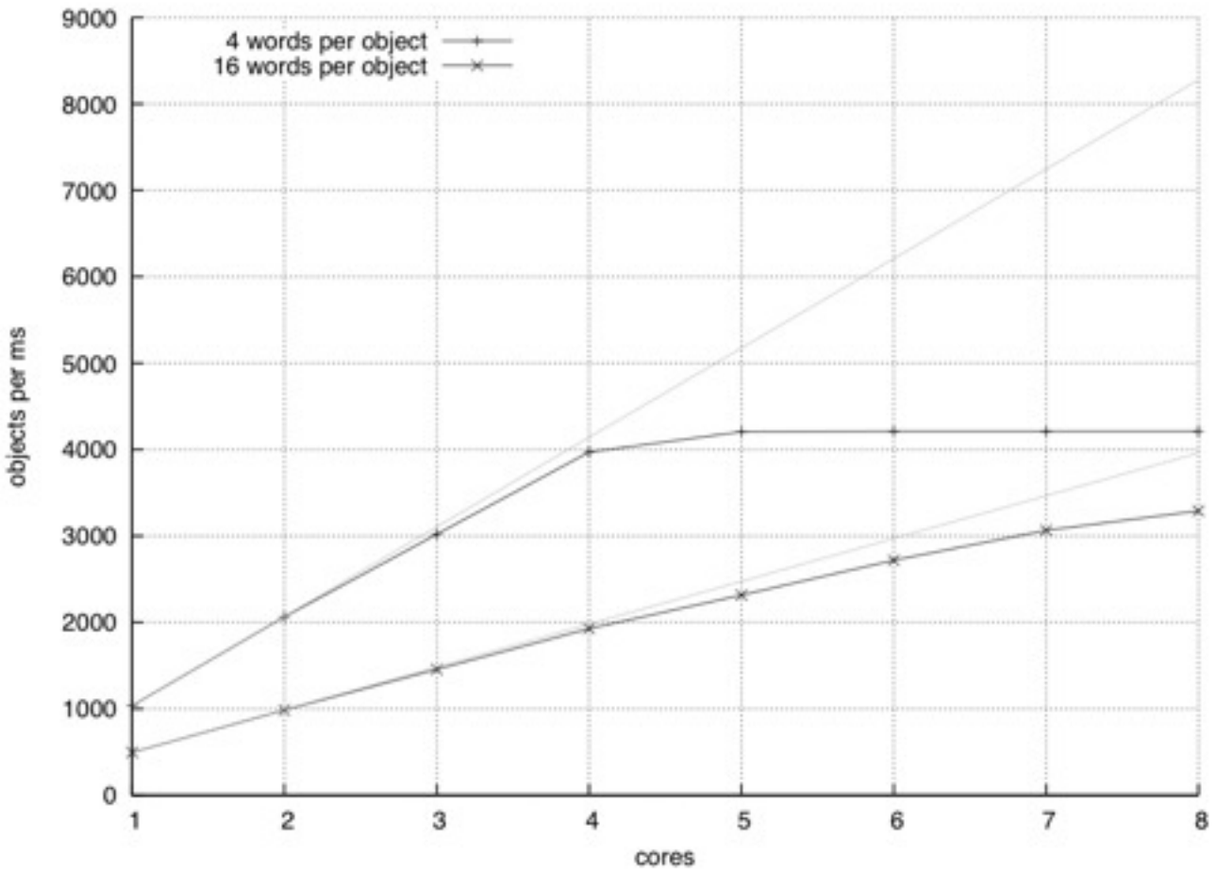
GC-Zyklus und Allokationen



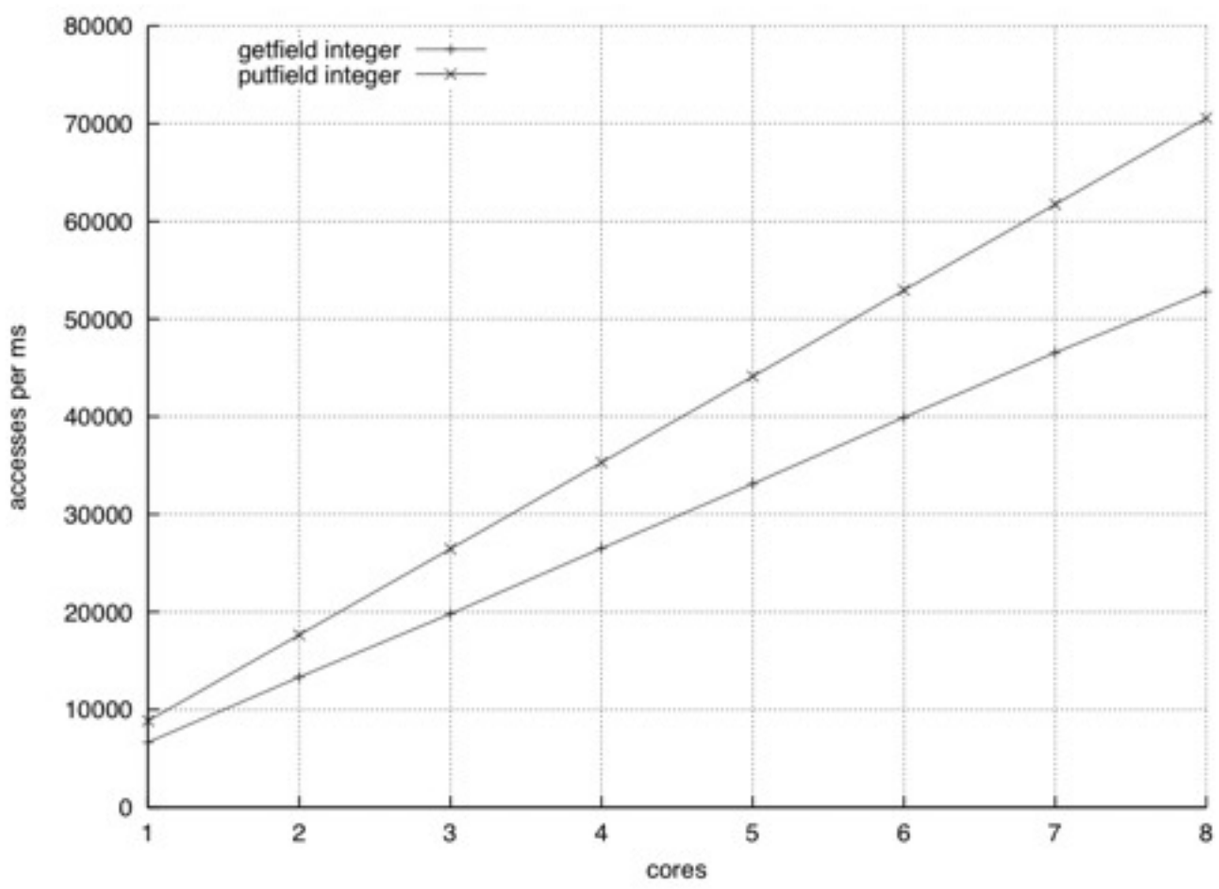
Einfluss eines GC-Zyklus auf Allokation

Anhang

maximale Allokations-, Schreibe- und Leseraten



maximale Allokationsrate



maximale Schreibe- und Leserate

Quellenangaben

[1] Bergeron, J.: Writing Testbenches. Functional Verification of HDL Models. Boston [u.a.]: Kluwer Academic, 2000.

[2] Wiemann, A.: Standardized Functional Verification. New York: Springer, 2008.