Entwurf und Implementierung verschiedener Garbage-Collector-Strategien für die Java-Plattform SHAP

Peter Reichel
peter.reichel@mailbox.tu-dresden.de

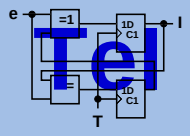
Technische Universität Dresden
Institut für Technische Informatik

Inhalt

Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Einleitung

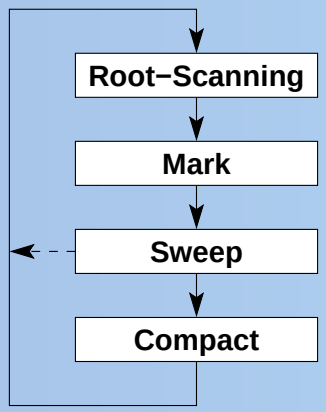
1 Einleitung

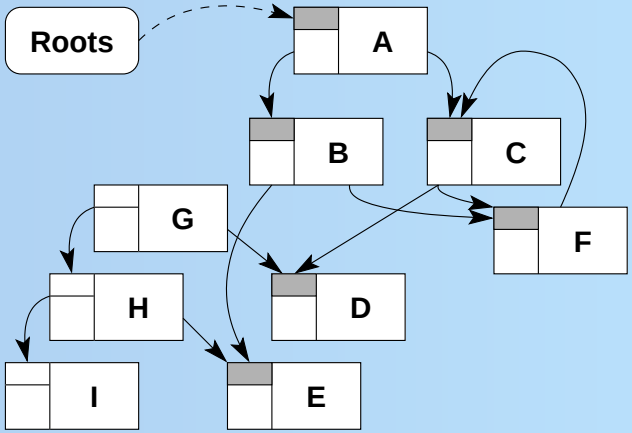


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Garbage-Collection

- **Ziel:** Automatische Freigabe *nicht länger benötigter* Speicherbereiche.
- **Ideal:** Unmittelbare Freigabe nach letzten Zugriff
  $\Rightarrow$ Blick in die Zukunft.
- **Real:** Freigabe *nicht referenzierter* Objekte
  - Zählung von Referenzen (*reference-counting*)
  - Traversieren des Objektgraphen (*tracing*)

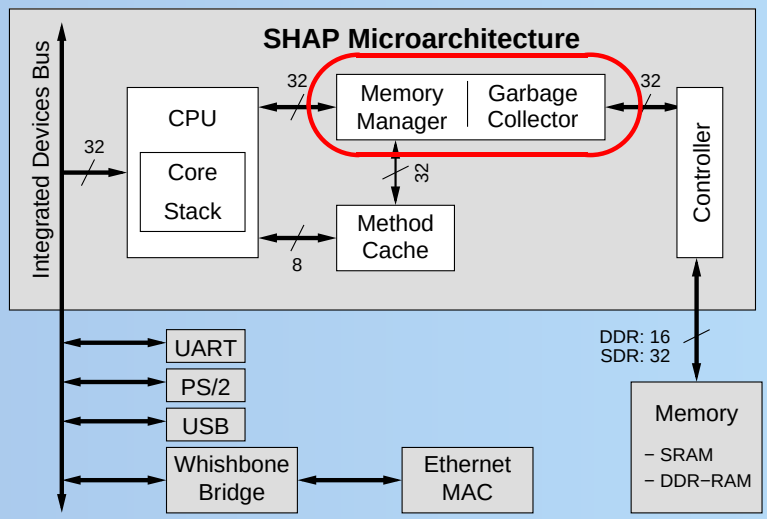




Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

SHAP-Mikroarchitektur:

- direkte Ausführung von Java-Bytecodes
- Stack und Heap physisch getrennt
  $\Rightarrow$ Objekte können nur auf dem Heap liegen!
  $\Rightarrow$ objektbasiertes Speichermodell
- *Worst-case*-Zeiten aller Operationen bekannt ($\rightarrow$ Echtzeit)



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

2 Entwurf des Speichermanagers

Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Forderungen

Forderungen an die Speicherverwaltung:

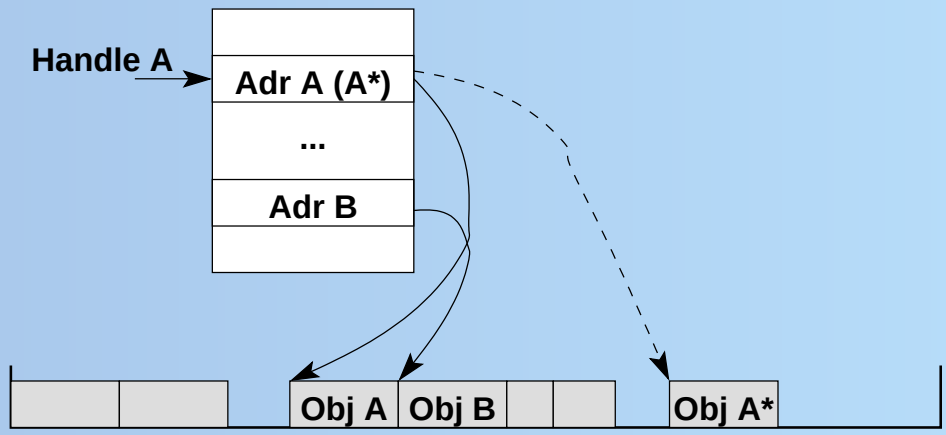
1. Unterstützung des *objektbasierten Speichermodells* von SHAP
   $\Rightarrow$ anlegen neuer Objekte
   $\Rightarrow$ Zugriff auf bestehende Objekte
   $\Rightarrow$ Freigabe nicht benötigter Objekte (GC)
2. Vermeidung langer Unterbrechungen
   $\Rightarrow$ vorhersagbare maximale Latenz-Zeiten
3. Speicherreservierung in konstanter Zeit
4. Bereitstellung von ausreichend freiem Speicher

Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Objekte und Handles

Objektbasiertes Speichermodell:

- Verwendung von Indirektionszeigern (Handles)
  ⇒ erlaubt leichte Verschiebung des Objekts im Speicher
- Referenz-Einträge in Tabelle gespeichert
  ⇒ Anzahl Referenz-Einträge und damit Objekte begrenzt!

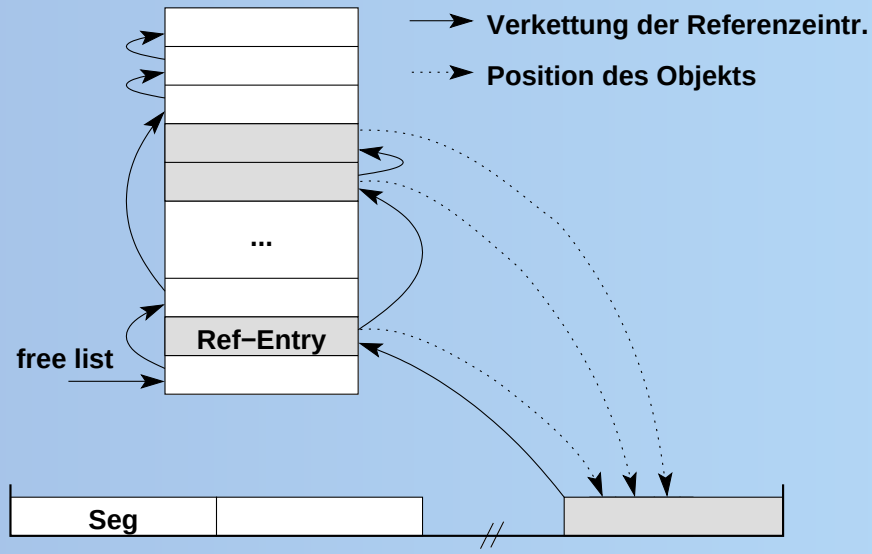


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

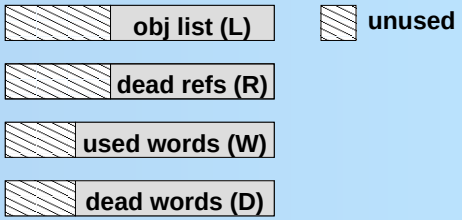
Segmente

Teilung des Speichers:

- Aufteilung des Speichers in $k = 2^n$ gleichgroße *Segmente*
- stets ein *Allokationssegment* gewählt
  $\Rightarrow$ neue Objekte anlegen, fortlaufende Speicherreservierung
  $\Rightarrow$ Austausch, wenn freier Speicher kleiner als max. Objektgröße
  $\Rightarrow$ Suche nach freiem Speicher nicht notwendig!



Segment-Eintrag:

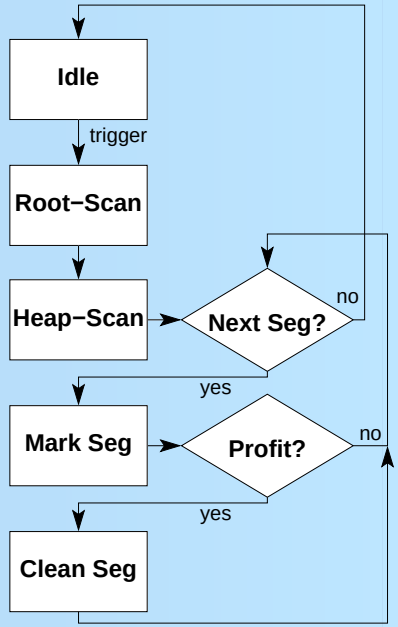


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Grundkonzept

Grundkonzept der Garbage Collection:

- Tracing-Verfahren (konservativ, Signatur)
- GC läuft parallel zur eigentlichen Applikation
  ⇒ Veränderung des Objektgraphen möglich!
  ⇒ Keine fälschliche Freigabe!
- Überwachung des Stacks
  ⇒ Objekt markieren wenn Referenz entdeckt
- Bereinigung eines Segments, wenn *Gewinn* an freiem Speicher erzielt werden kann
  ⇒ Verschieben erreichbarer Obj. in Zielsegment
  ⇒ Freigabe nicht erreichbarer Referenz-Einträge
  ⇒ Freigabe des gesamten Segments



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Variationen (1)

Voralterung von Segmenten:

- Objekte haben meist kurze Lebensdauer
  (→ *”most objects die young“* [Wils94])
- Allokationssegment enthält nur junge Objekte!
- Bereinigung: unnötigte Verschiebung junger Objekte

⇒ Segment wird für die Dauer eines Zyklus ignoriert

Frühere Freigabe der Referenz-Einträge:

- Referenzeinträge bereits in Markierungs-Phase freigeben
- erhöht Anzahl nutzbarer Referenzeinträge
- Beschränkung der Objektanzahl pro Segment nur noch für Allokationssegment

Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

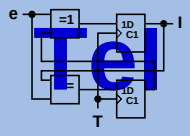
Variationen (2)

Objekte etwa gleichen Alters zusammenfassen (*Generationen*):

- Segmente werden zu *Gruppen* zusammengefasst
- jede Gruppe entspricht einer Generation
- Überlebt Objekt aus Gruppe $i \Rightarrow$ verschieben in Gruppe $i+1$
- Markierungs- und Freigabephase werden für ältere Gruppen seltener ausgeführt
  $\Rightarrow$ Verkürzung der Zyklenzeit

Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

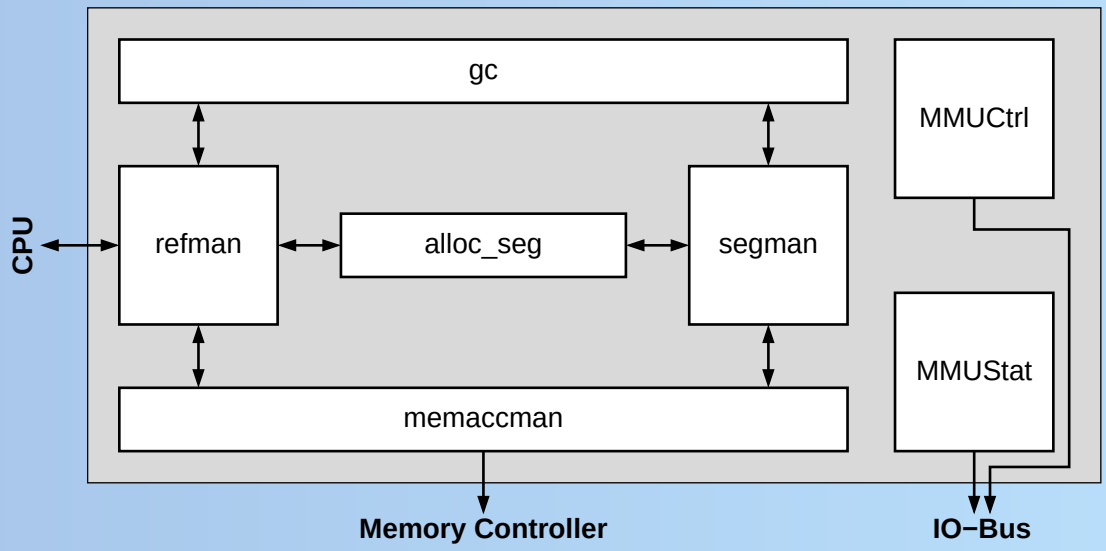
3 Implementierung



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Struktur der MMU

- MMU in VHDL implementiert
- saubere Trennung der einzelnen Komponenten
- Steuerung der MMU über `MMUCtrl`
- Statistik-Informationen über `MMUStat`

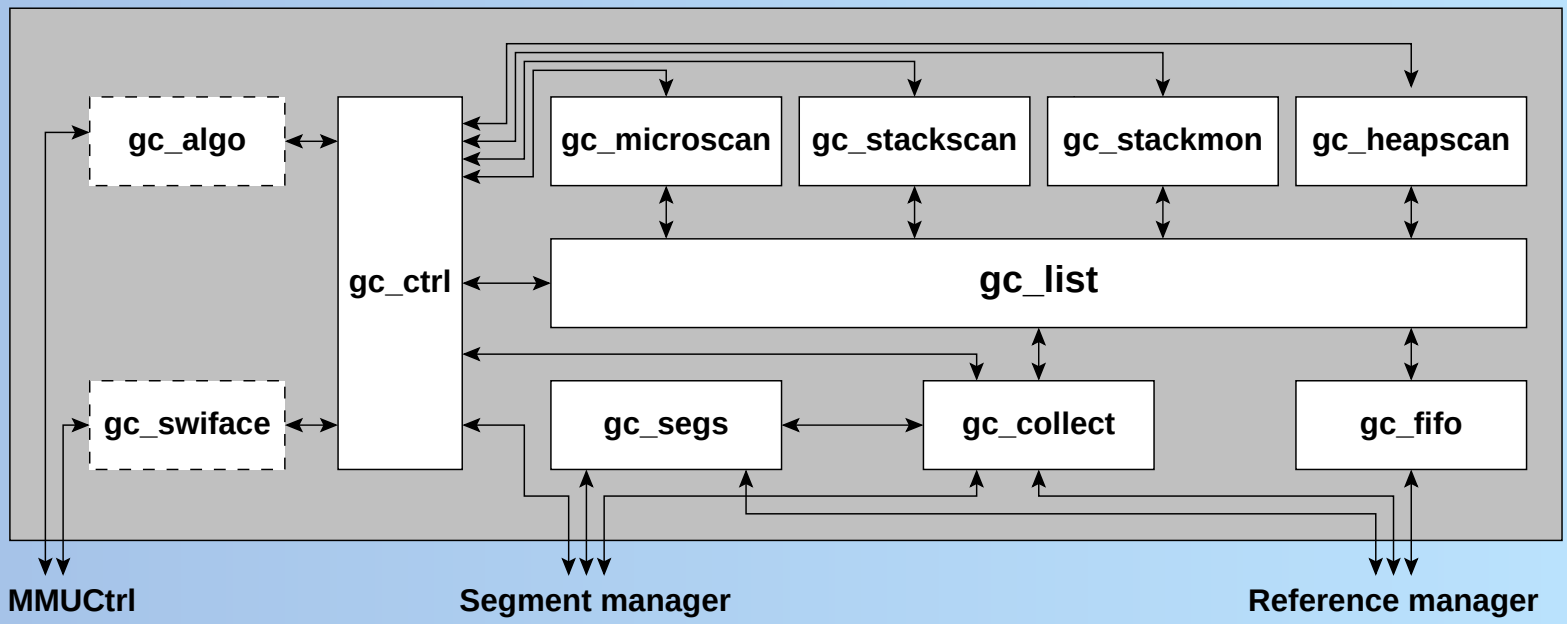


Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Struktur des GC

Übersicht der Komponenten des GC:

- Steuerung durch FSM (`gc_algo`) oder mittels Java-Programm (`gc_swiface`)



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Implementierte Varianten

- `gc_hw_base`:
  Grundkonzept
- `gc_hw_pm`:
  Voralterung von Segmenten
- `gc_hw_fri`:
  frühere Freigabe von Referenzeinträgen
- `gc_hw_pm_fri`:
  Kombination der Voralterung von Segmenten und der früheren Freigabe von Referenzeinträgen
- `gc_sw_fri`:
  Verwendung des Software-Interfaces `gc_swiface`, Bildung von Segmentgruppen (Generationen)

Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Syntheseergebnisse

Syntheseergebnisse für Spartan-3-FPGA XC3S1000:

- Taktfrequenz: 50 MHz
- 8 kByte Stack, 1 MByte externer SRAM
- 2048 Referenzeinträge, 32 Segmente (je 8kByte)

Variante:	gc_hw_base	gc_hw_pm	gc_hw_fri	gc_hw_pm_fri	gc_sw_fri
Flip-Flops	1697 (3139)	1697	1709	1709	1755
4 input LUTs	4337 (6588)	4345	4384	4392	4504
- used as logic	4071 (6328)	4079	4115	4123	4235
- used as route-through	180 (174)	180	183	183	183
- used for Dual Port RAMs	86 (86)	86	86	86	86

Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Vergleich implementierter Varianten

4 Vergleich implementierter Varianten

Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

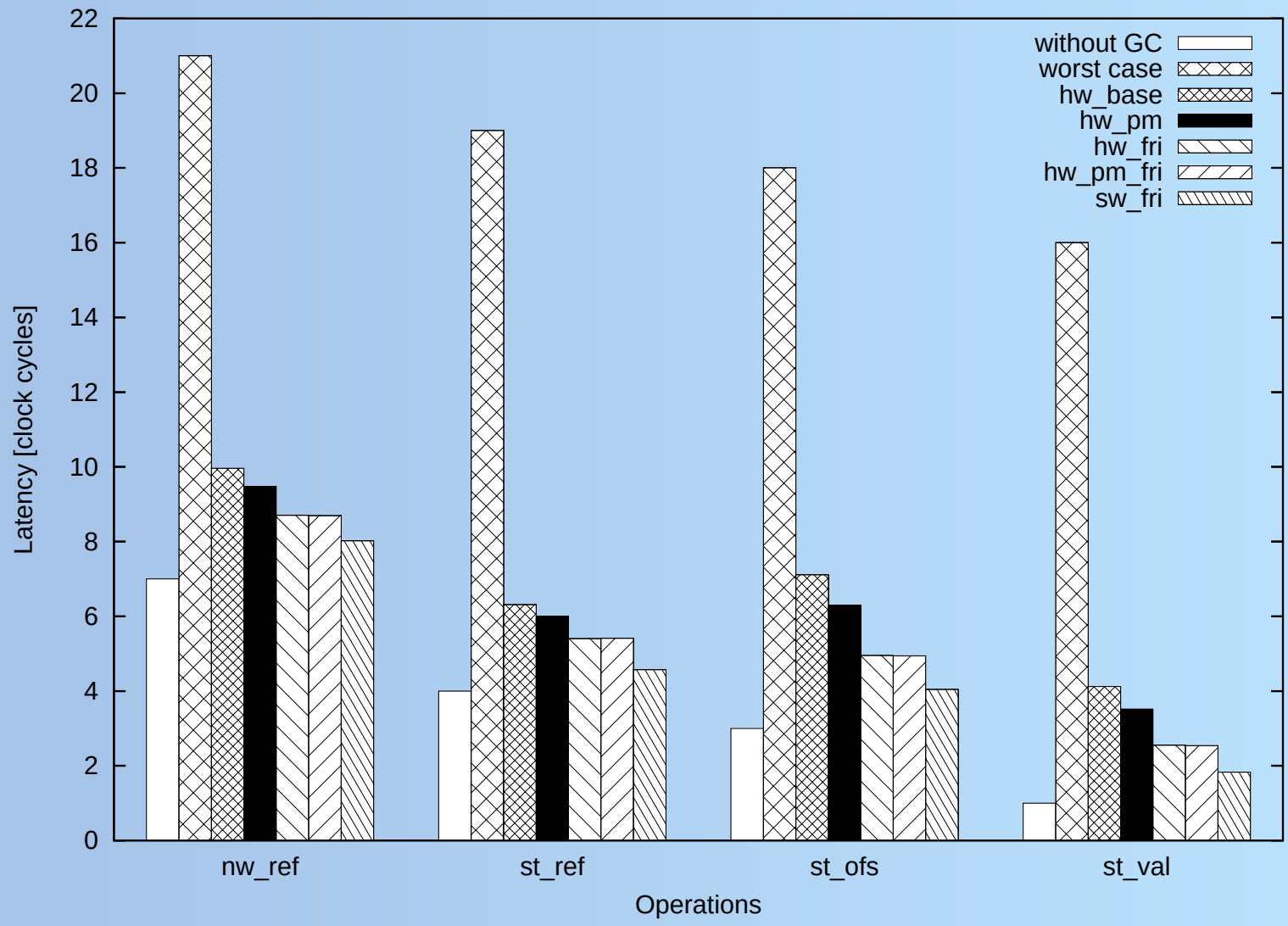
Test-Applikationen

Drei ausgewählte Test-Applikationen:

1. Caffeine Benchmark
   - synthetischer Benchmark
   - nur bei String-Test relativ viel Garbage
2. Sudoku-Löser
   - Finden *aller* Lösungen eines Sudoku-Puzzles mittels Backtracking
   - Ausgeben aller Lösungen in separatem Thread
3. FScript
   - Interpreter für einfache Script-Sprache
   - sehr speicherintensiv, viel Garbage

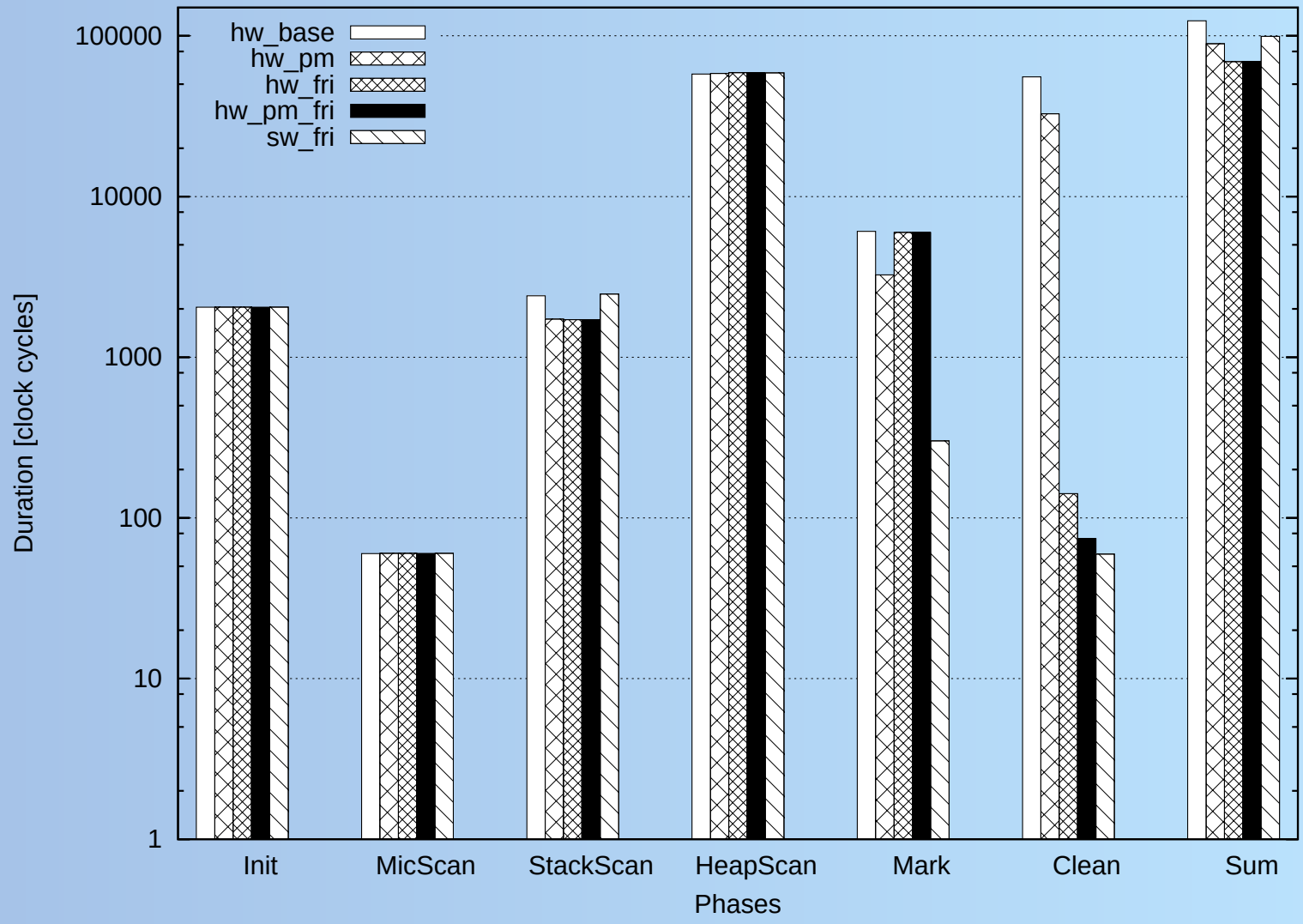
Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Vergleich der Latenzzeiten



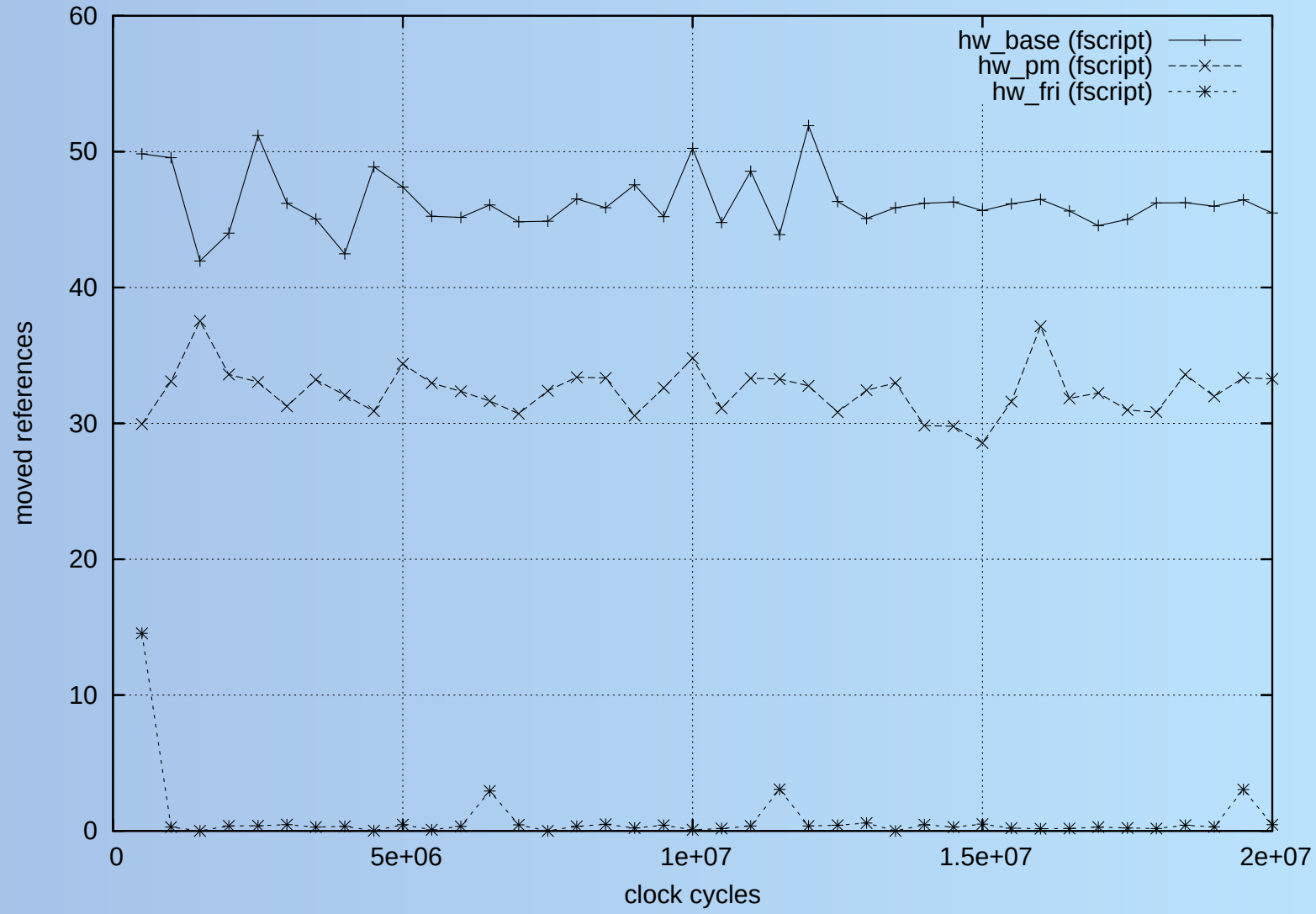
Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Dauer der Phasen



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Anzahl verschobener Objekte



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Zusammenfassung

5 Zusammenfassung

Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Zusammenfassung

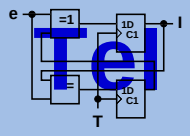
Zusammenfassung und Ausblick:

- Speichermanager erfüllt gestellte Forderungen
- alle Komponenten der MMU vollständig in HW implementiert
- Steuerung des GC durch FSM oder Java-Programm möglich
- Funktionsnachweis durch drei Test-Applikationen
- Vergleich unterschiedlicher Varianten
- größtes Optimierungspotential bei Heap-Scan
- konservativer Ansatz muss durch exakten ausgetauscht werden
  $\Rightarrow$ nicht erreichbare Objekte *müssen* freigegeben werden
- Vergleich durch Verwendung allgemeiner Benchmarks ermöglichen

Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de

Danke

Vielen Dank für Ihre Aufmerksamkeit!



Peter Reichel
Technische Universität Dresden
Institut für Technische Informatik
peter.reichel@mailbox.tu-dresden.de