TECHNISCHE UNIVERSITÄT DRESDEN

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

Anforderungsanalyse für Daten-Caches für die SHAP-Mikroarchitektur

Belegverteidigung – 04.02.2009

Stefan Alex
s2174321@inf.tu-dresden.de

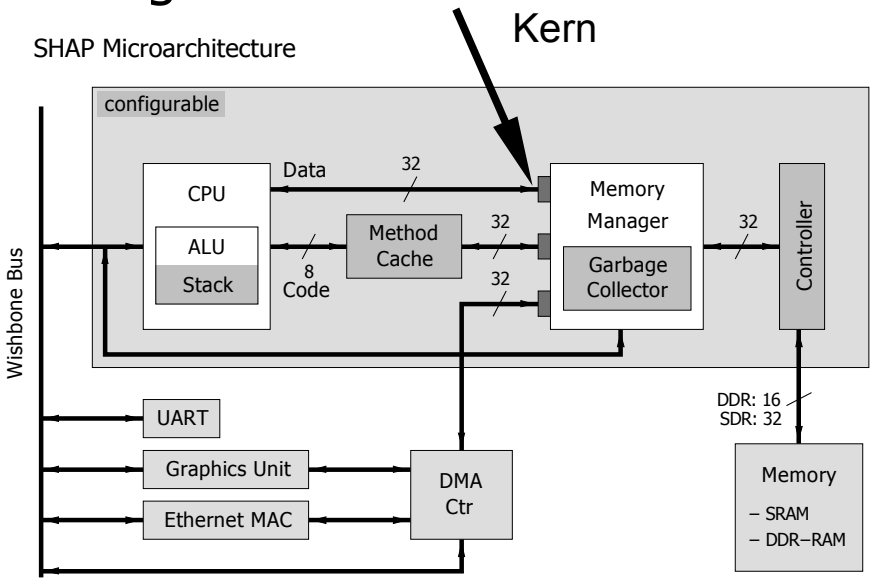
Gliederung

1. Aufgabenstellung
2. Cache-Varianten
3. Implementierungen
4. Testapplikationen
5. Ergebnisse
6. Ausblick

1. Aufgabestellung

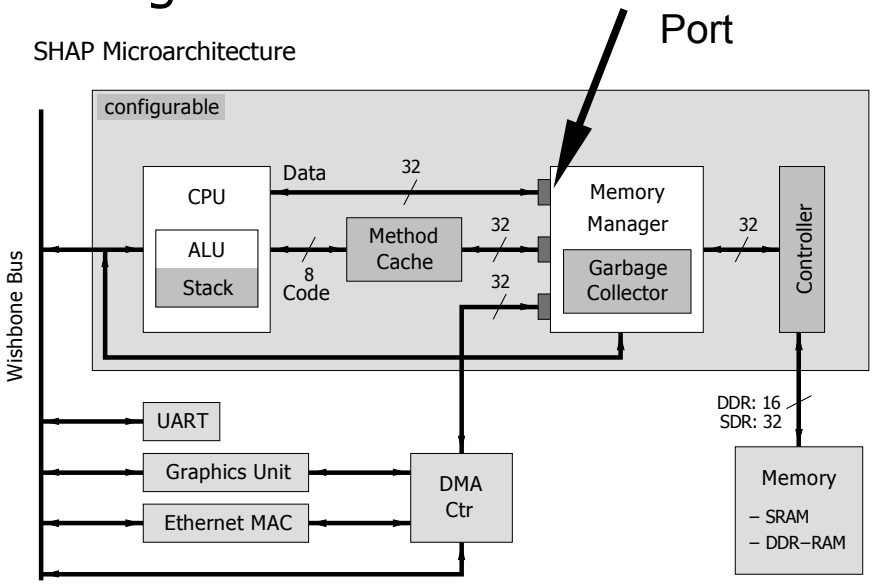
- Literaturstudium zu Strategien für Daten-Caches unter den Aspekten: Caching von Datenstrukturen (Objekte), Besonderheiten im Multi-Core-Bereich und praktischer Anwendung in eingebetteten Systemen.
- Entwurf von Protokollierungseinheiten für die Analyse der zeitlichen Lokalität von Objektaktivierungen einerseits und für die Analyse der zeitlichen und örtlichen Lokalität der Speicherzugriffe innerhalb der Objekte andererseits.
- Implementierung der Protokollierungseinheiten im Prozessorsimulator DITO als auch in Hardware (VHDL). Gegenseitiger Test anhand einer einfachen, dafür zugeschnittenen Testapplikation.
- Bestimmung der unter Punkt 2 genannten Kenndaten für mindestens 3 verschiedene, selbst gewählte Applikationen. Daraus ableitend, Formulierung von Anforderungen für den zukünftigen Entwurf eines TLB und eines Daten-Caches.
- Zusammenfassung und Dokumentation der Ergebnisse.

2. Cache-Varianten
Implementierungsstandort



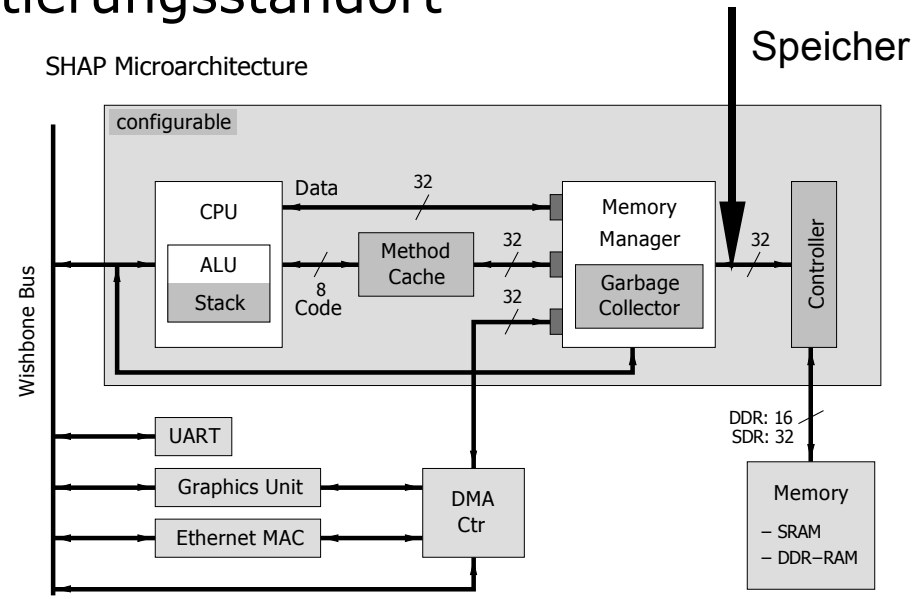
- Cachen der Daten-Anfragen des Prozessors
- virtuell adressiert, kurzer Anfrageweg durch CPU, langer Speicherzugriff
- Kohärenzprobleme bei Multi-Core-Systemen

2. Cache-Varianten
Implementierungsstandort



- Umrechnung des Offsets → andere Verteilung der Einträge über die Sets
- zusätzlicher Wartetakt bei Cache-Anfrage, kürzere Speicheranfrage

2. Cache-Varianten
Implementierungsstandort



- physikalisch adressiert, schneller Speicherzugriff
- kann Anfragen des CPU, Methoden-Caches, Speichermanager beantworten
- keine Kohärenzprobleme, dafür Eintragsinvalidierung durch Kopieroperationen

3. Implementierungen

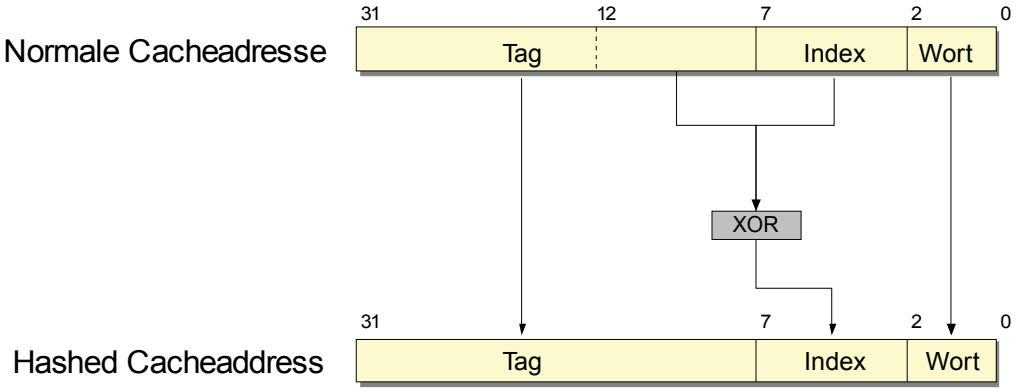
- Protokollierungseinheiten
  - Aufzeichnen von Adress-Traces
  - Kompression der Trace-Daten
  - Übertragung per USB
  - Ermitteln Durchschnittswerte für Zugriffszeiten
- Simulator
  - Lokalitätsanalyse
  - Simulation des Translation-Lookaside-Buffers
  - Cache
    - Analytische Modelle
    - Cache-Simulation

3. Implementierungen
Cache-Strategien

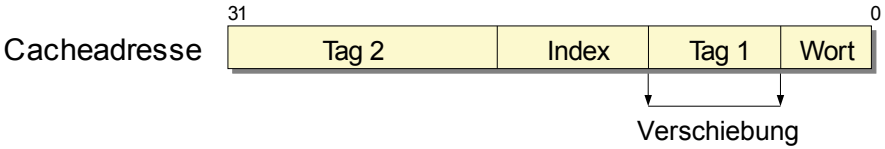
- Assoziativität
- Cache-Größe, Block-Größe, (Wort-Größe)
- Schreibstrategien
  - Write-Though (With Write-Allocation / With No-Write-Allocation)
  - Copy-Back
- Ersetzungsstrategien
  - LRU, NMRU, Random, FIFO
- Write-Buffer/Victim-Cache
- Adress-Dekodierung
- Kennzahlen: Hit-Raten, Speed-Up, Hardware-Score

3. Implementierungen
Adress-Dekodierung

- Index-Umrechnung:

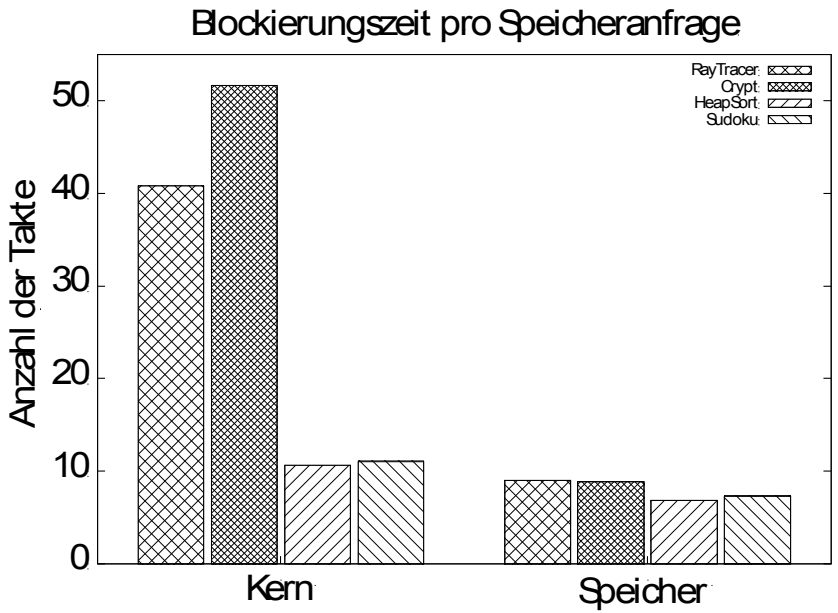


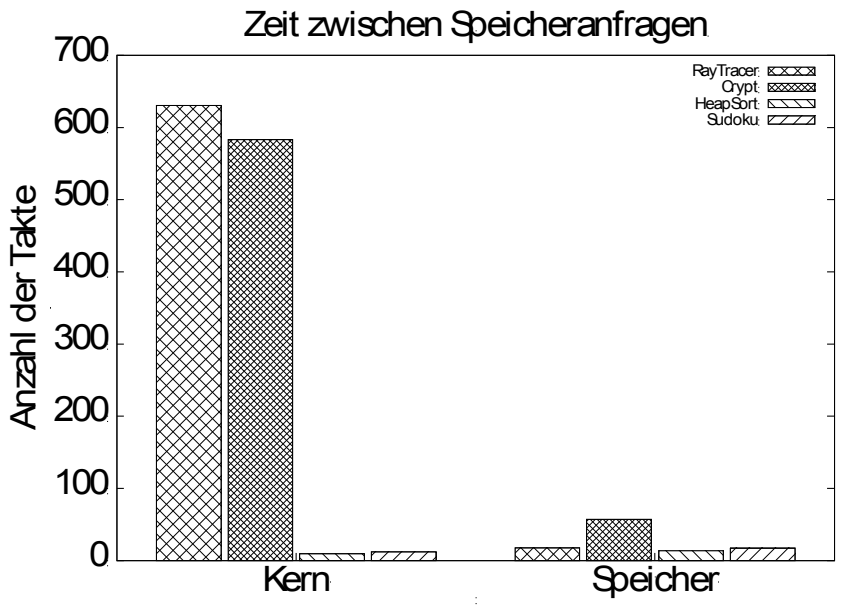
- Auswahl der Index-Bits:



4. Testapplikationen

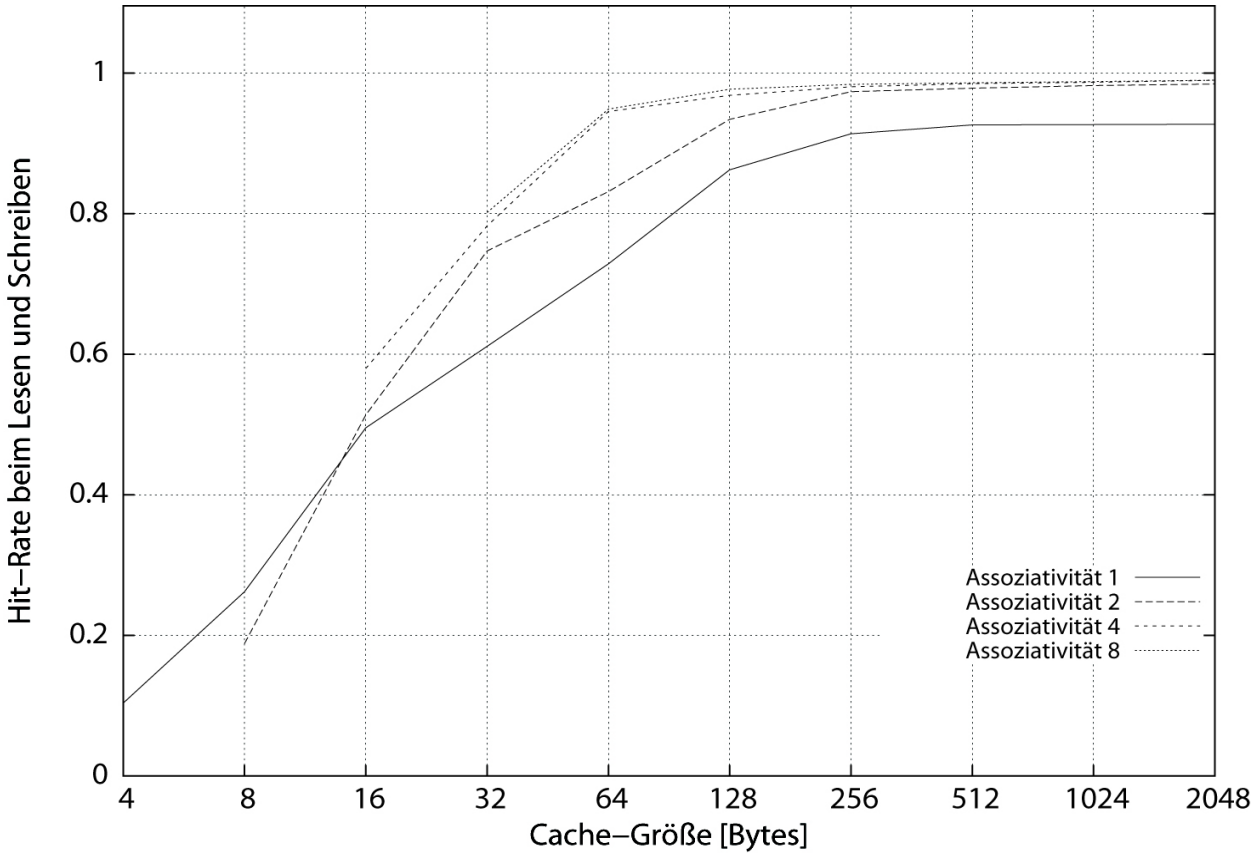
- Anwendungen mit unterschiedlichen Arbeitsmengen/Verhalten
- zusätzlich Multi-Thread-Anwendungen
- Wirkung des Caches abhängig von Anzahl und Dauer der Speicherzugriffe



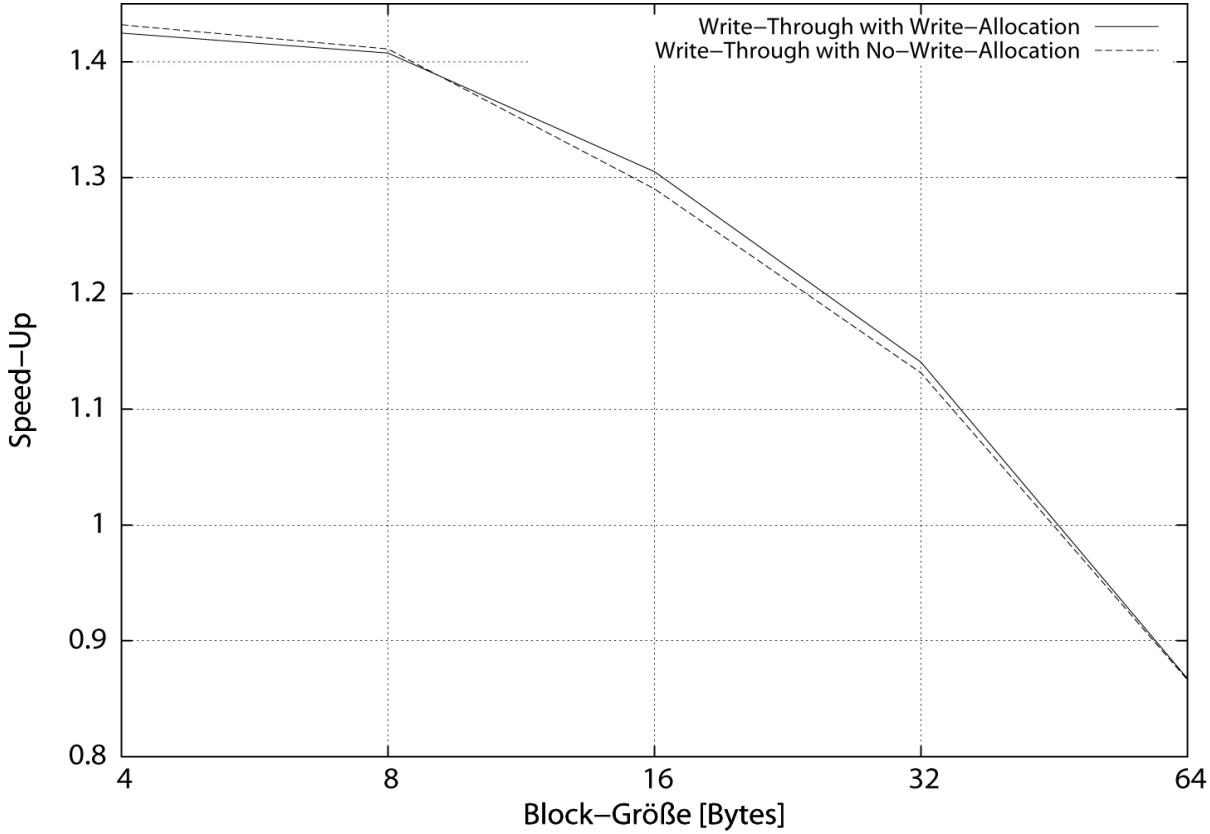


5. Ergebnisse

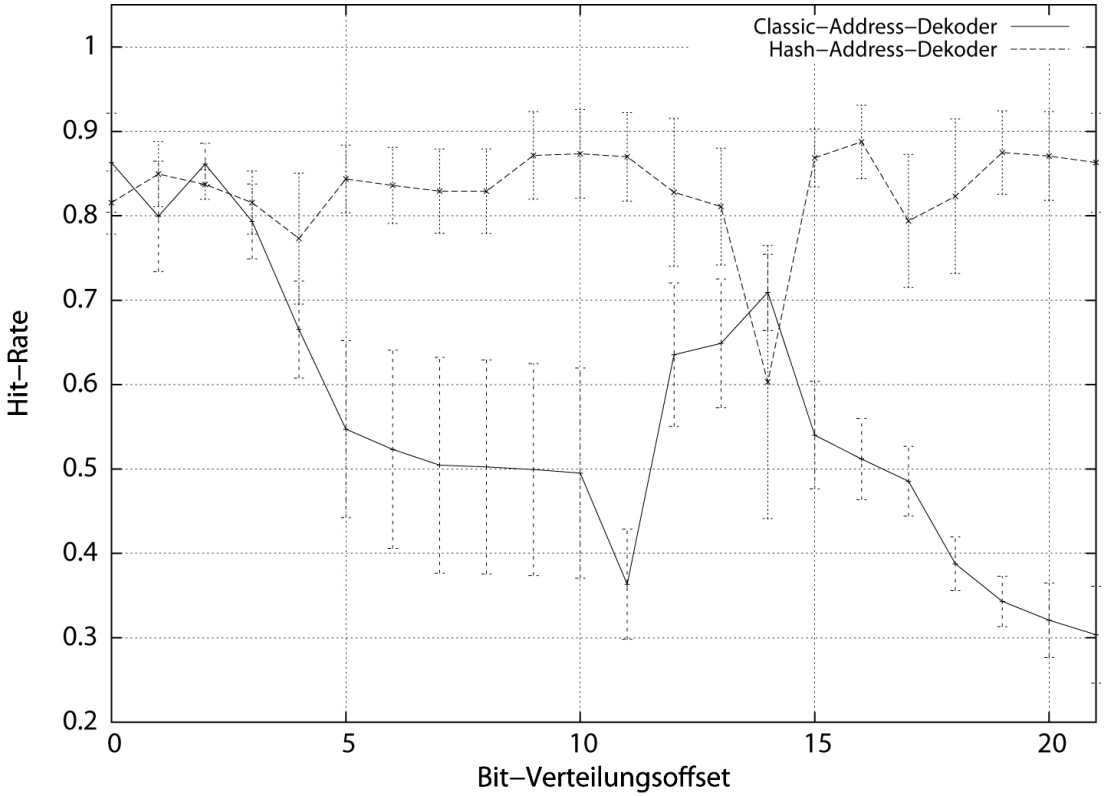
Kern – Cache-Größe und Assoziativität



5. Ergebnisse
Kern – Block-Größe und Schreibstrategie



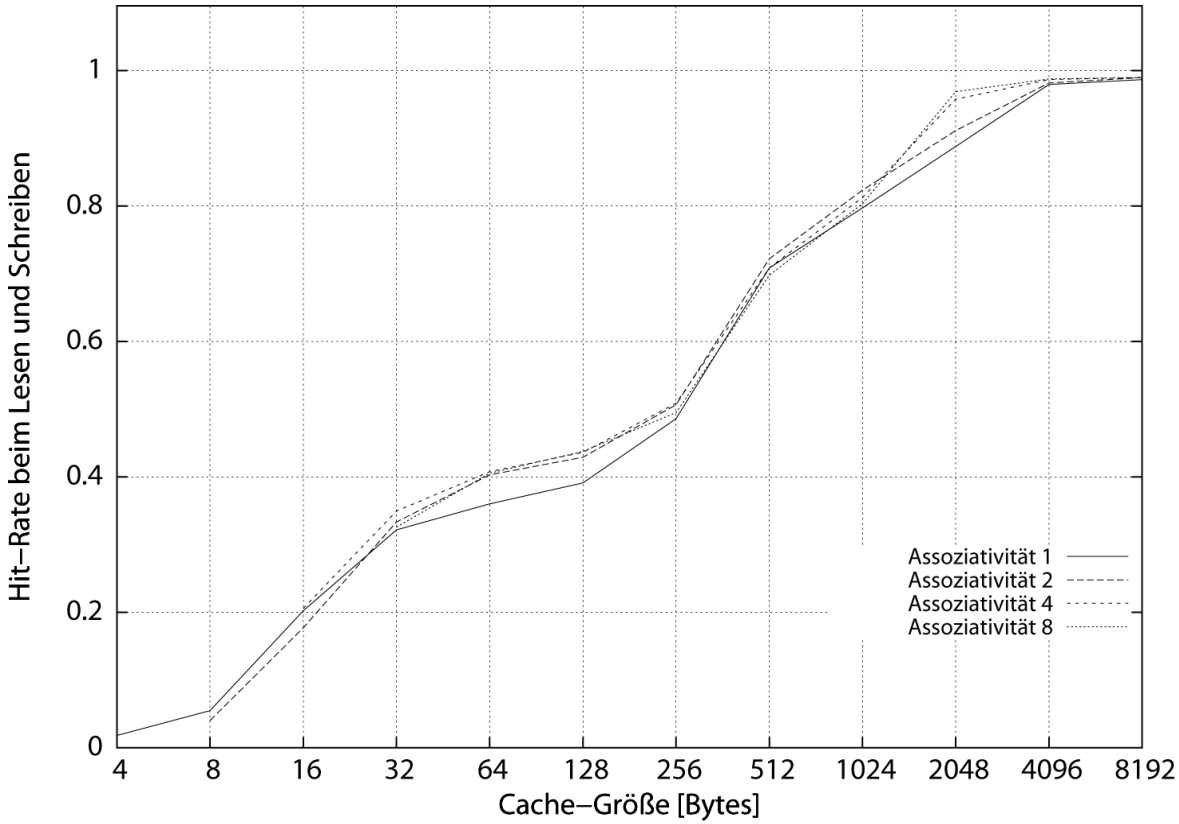
5. Ergebnisse
Kern – Adress-Dekodierung



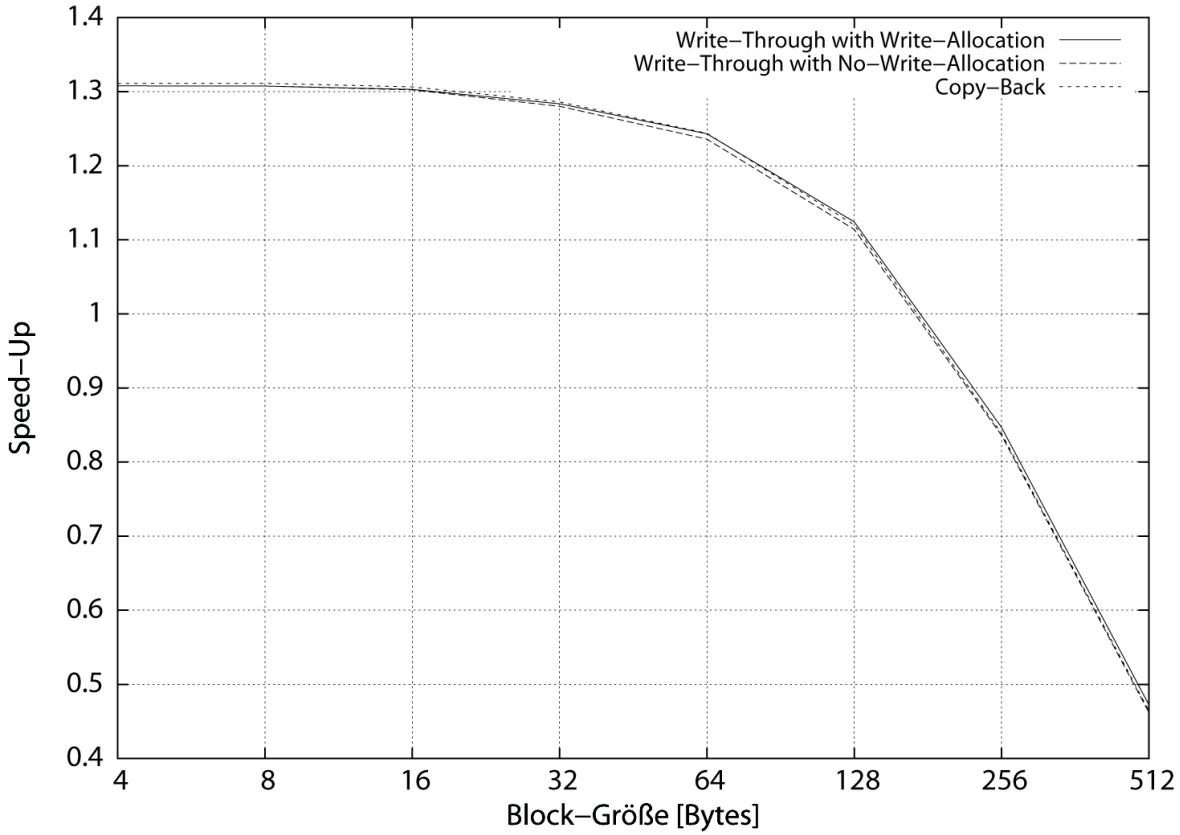
5. Ergebnisse
Kern

- **Ersetzungsstrategie**
  - minimaler Geschwindigkeitsgewinn durch Least-Recently-Used
- **Write-Buffer** → Kapazität 4-8 Einträge
  - Ausgleich niedriger Assoziativität von Direct-Mapped-Cache
  - genauere Komplexitätsabschätzung nötig
- durchschnittlicher Speed-Up von 1.41
- stark abhängig von Speicherzugriffsdauer/-anzahl

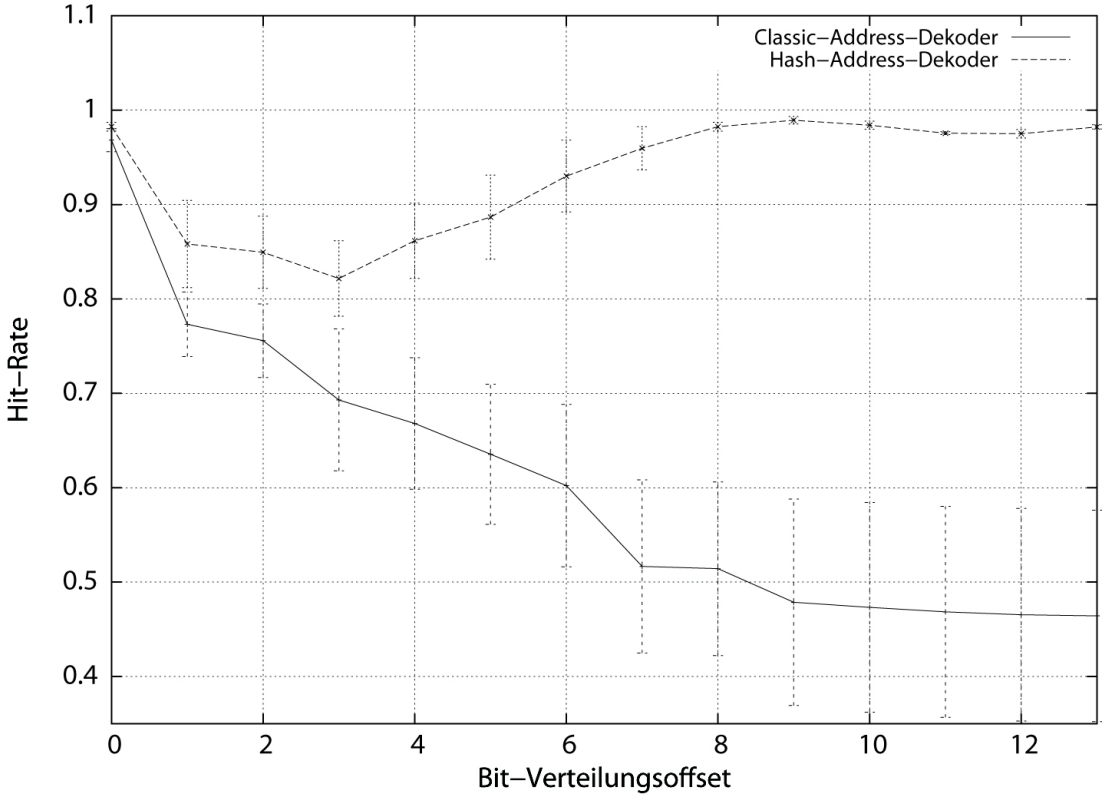
5. Ergebnisse
Speicher – Cache-Größe und Assoziativität



5. Ergebnisse
Speicher – Block-Größe und Schreibstrategie



5. Ergebnisse
Speicher – Adress-Dekodierung



5. Ergebnisse
Speicher

- **Ersetzungsstrategie**
  - kaum Unterschiede
  - teilweise Zufallsstrategie mit höheren Hit-Raten
- **Write-Buffer** → 8 Einträge
  - Steigerung des Geschwindigkeitsgewinns um 0.5 Prozentpunkte
- Cachen von CPU-, Methoden-Cache- und Memory-Manager-Anfragen sinnvoll
- Speed-Up von 1.26 - 1.29
- Beschleunigung aller Anwendungen

6. Ausblick

- Cache am Kern
  - Beschleunigung stark abhängig von den Anwendungen
  - geringer Aufwand
  - Kohärenzproblematik
- Cache am Port
  - ineffektiv aufgrund längerer Anfragezeit
- Cache am Speicher
  - hoher Aufwand
  - simple Cache-Strategien anwendbar
  - geringere Durchschnittsbeschleunigung
  - geringere Abweichung

Fragen ???

Zusatz: Empfehlungen

- Cache am Kern
  - Cache-Größe: 128 Byte
  - Block-Größe: 8 Byte (2 Speicherwörter)
  - Assoziativität: 2
  - Ersetzungsstrategie: Zufall
  - Schreibstrategie: Write Through with No-Write-Allocation
  - Adress-Dekoder: Hash-Address-Decoder mit 15 Bit-Verschiebung
  - Write-Buffer-Größe: 1
  - Hardware-Score: 38582

Zusatz: Empfehlungen

- Cache am Speicher
  - Cache-Größe: 2-4 KByte
  - Block-Größe: 16 Byte (4 Speicherwörter)
  - Assoziativität: 1
  - Ersetzungsstrategie: -
  - Schreibstrategie: Write Through with No-Write-Allocation
  - Adress-Dekoder: Hash-Address-Decoder mit 9 Bit-Verschiebung
  - Write-Buffer-Größe: 1
  - Cache-Anfrager: CPU, Methoden-Cache, Speicher-Manager
  - Hardware-Score: 642868-1341306