TECHNISCHE UNIVERSITÄT DRESDEN

VLSI-EDA

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

Leistungssteigerung der Heap-Anbindung von Java-Bytecode-Mehrkernprozessoren

Martin Zabel,
Andrej Olunczek,
Rainer G. Spallek

DASS Dresden, 04.05.2012

Martin.Zabel@tu-dresden.de

http://vlsi-eda.inf.tu-dresden.de

Gliederung

1 Motivation
2 Bisherige Arbeiten
3 Analyse
4 Ergebnisse
5 Zusammenfassung

Motivation

Warum Java?

- Objektorientierung, Portabilität
- automatische Speicherverwaltung, Sicherheit
- Unterstützung für Parallelisierung

Warum Java-(Bytecode-)Prozessor?

- Native Ausführung von Java-Bytecode
  ➔ kein OS, keine Interpretation, keine Re-Kompilation
- Echtzeitfähigkeit
- Geeignet für eingebettete Systeme mit begrenzten Ressourcen

Warum Mehrkernprozessoren?

- Elektr. Verlustleistung steigt überproportional mit Taktfrequenz
- Stattdessen Nutzung von Parallelität auf Thread-Ebene

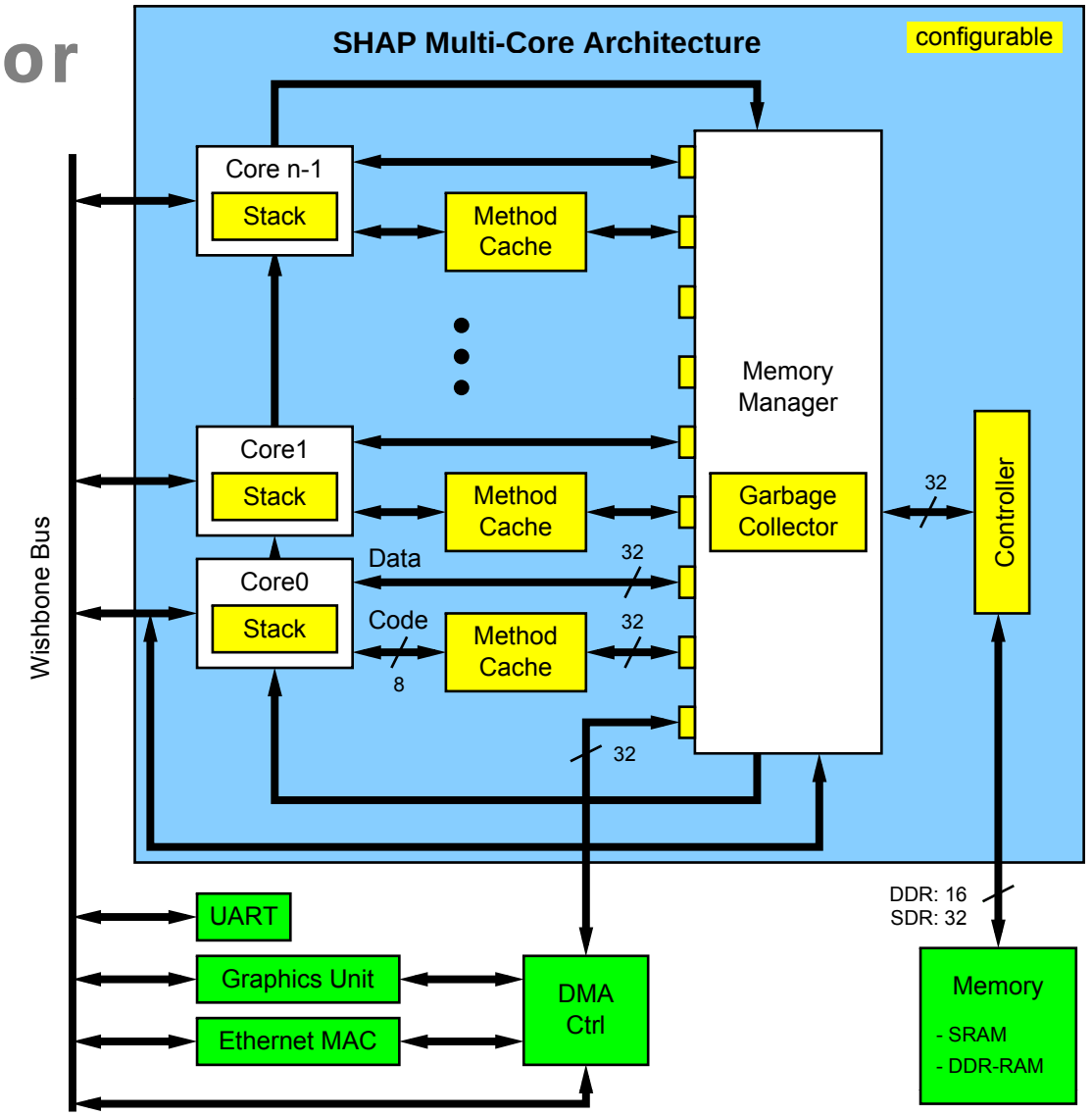
Java-Mehrkernprozessor

Beispiele: JopCMP, jamuth, REALJava und SHAP

Gemeinsamkeit: zentraler gemeinsamer Heap-Speicher für alle Kerne

SHAP-Mehrkernprozessor:

- Lokaler Stack-Speicher je Kern
- Methoden-Cache je Kern
- Pipelining der Heap-Zugriffe
- Maximaler Speed-Up von 8 für Programme mit überdurch-schnittlich vielen Speicherzugriffen [1]



Ziel

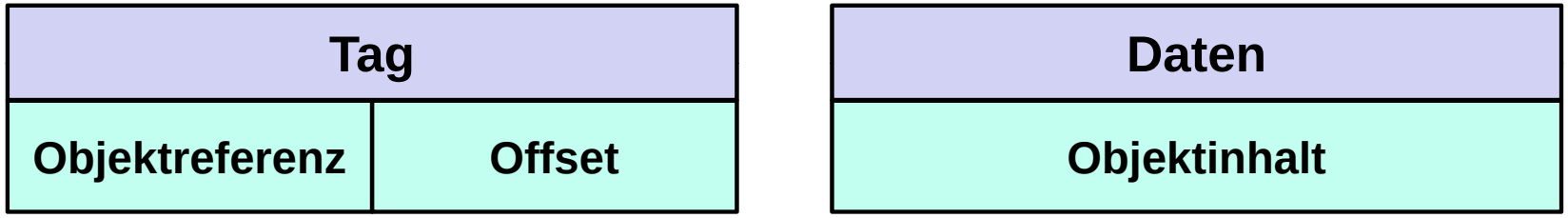
Weitere Entlastung der Heap-Speicherschnittstelle,
um einen höheren Grad an Parallelverarbeitung (Speed-Up) zu erreichen.

Gliederung

Bisherige Arbeiten

Allg. Lösung für objektorientierte Prozessoren:

Cache für Objekte analog eines Daten-Caches [2] [3]



Speziell für Echtzeitsysteme:

Separate Caches für verschiedene Daten-Bereiche [4]:

- Standard-Cache für Daten an statischen Adressen (z.B. Klasseninfos)
- Objekt-Cache für Daten an dynamischen Adressen (z.B. Objekte)

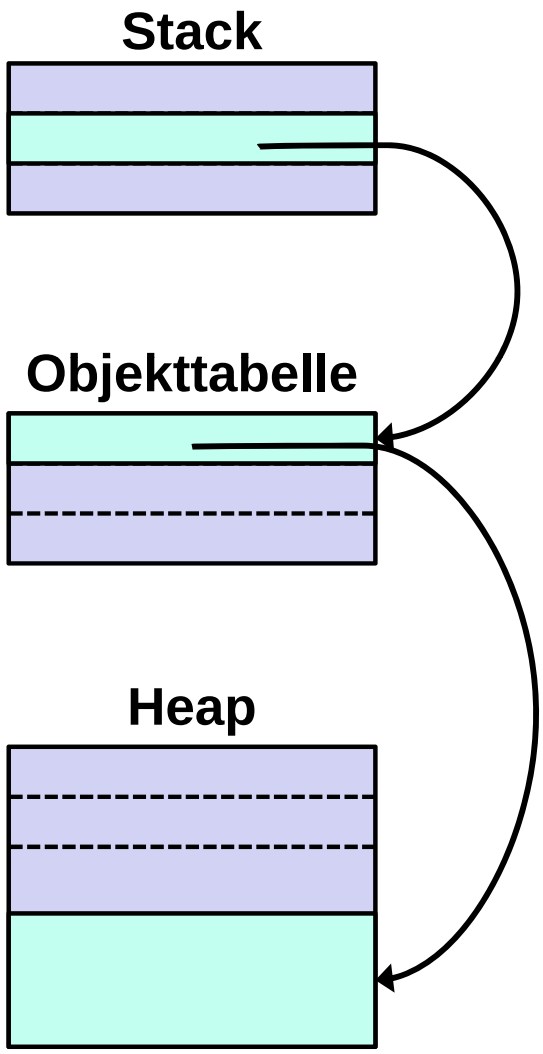
Indirekte Objektadressierung

Problem bei JopCMP und SHAP:

- Objekttabelle im externen Speicher
- Zusätzliche Latenz beim Heap-Zugriff
- Zusätzlicher Bedarf an Speicherbandbreite

Lösung:

- Translation-Lookaside-Buffer (TLB) [1]
- Virtuell indizierter Objekt-Cache [2]



Cache-Koheränz

Problem: Koheränz von verteilten Objekt-Caches

Vorteil der JVM: Synchronisation nur bei [5]

- Betreten eines kritischen Abschnitts
- Zugriff auf „volatile“-Variable

Gliederung

Analyse

Durchführung:

- Benchmark-Suite JemBench (Version 2.0) [9]
- SHAP-Mehrkernarchitektur mit **einem Kern** und Trace-Einheit [1]
- Aufzeichnung der ausgeführten Befehle und Speicherzugriffe

Ergebnisse:

1. Häufigste Objektzugriffe sind Lesezugriff auf Arrays[1] und Instanzvariablen.
2. 80% der Objektzugriffe konzentrieren sich auf 6 Objekte.
3. Häufiger Zugriff auf die ersten benutzerdefinierten Objektvariablen an den Offsets -2 und 1

➔ **Weitere Untersuchung**: kleiner vollassoziativer Cache je Kern

[1] (implizite Lesezugriffe auf Array-Länge bereits berücksichtigt)

Variantenuntersuchung

1. Bisheriger TLB mit 2 Einträgen
2. Array-Längen-Cache
3. Offset-Cache (-2 und 1): Write-Through-Cache, SRAM-Anbindung, 1 Valid-Bit/Wort, Ersetzung nach Least-Recently-Used (LRU)
4. Adress- und Offset-Cache (AOC) = Variante 3 mit integriertem TLB
5. Nur TLB mit LRU-Strategie

Ergebnisse für MatrixMul-Benchmark und 8 Cache-Zeilen (außer Variante 1)

Variante	1	2	3	4	5
Speicherbusbelegung u	**10.7%**	9.0%	8.1%	**5.2%**	7.9%
Max. Speed-Up $S=1/u$	**9.3**	11.1	12.3	**19.2**	12.7

Variantenuntersuchung

Weitere Varianten:

- 1 Valid-Bit pro Cache-Zeile (bei SRAM-Anbindung)
- DDR-RAM-Anbindung mit 1 Valid-Bit pro Cache-Zeile
- Größere Cache-Zeilen (mehr Offsets)
- Mehr Cache-Zeilen

➔ Keine nennenswerten Verbesserungen, teils Verschlechterungen.

Gliederung

Implementierung

Merkmale:

- AOC mit Write-Through, LRU-Strategie, 1 Valid-Bit pro Wort
- Anhand SHAP-Mehrkernarchitektur
- Konfigurierbar: Anzahl Cache-Zeilen, zwischengespeicherte Offsets
- Zusätzliche Latenz von einem Takt beim Speicherzugriff nötig

Syntheseergebnis:

- Bis zu 17 Kerne bei Beibehaltung der Taktfrequenz von 80 MHz möglich
- Zusätzlicher HW-Aufwand für n ≤ 15 Kerne ggü. bisher:

Cache-Zeilen	Register	LUTs
4	+4%	+6%
8	+9%	+8%

Leistungssteigerung auf 1 Kern

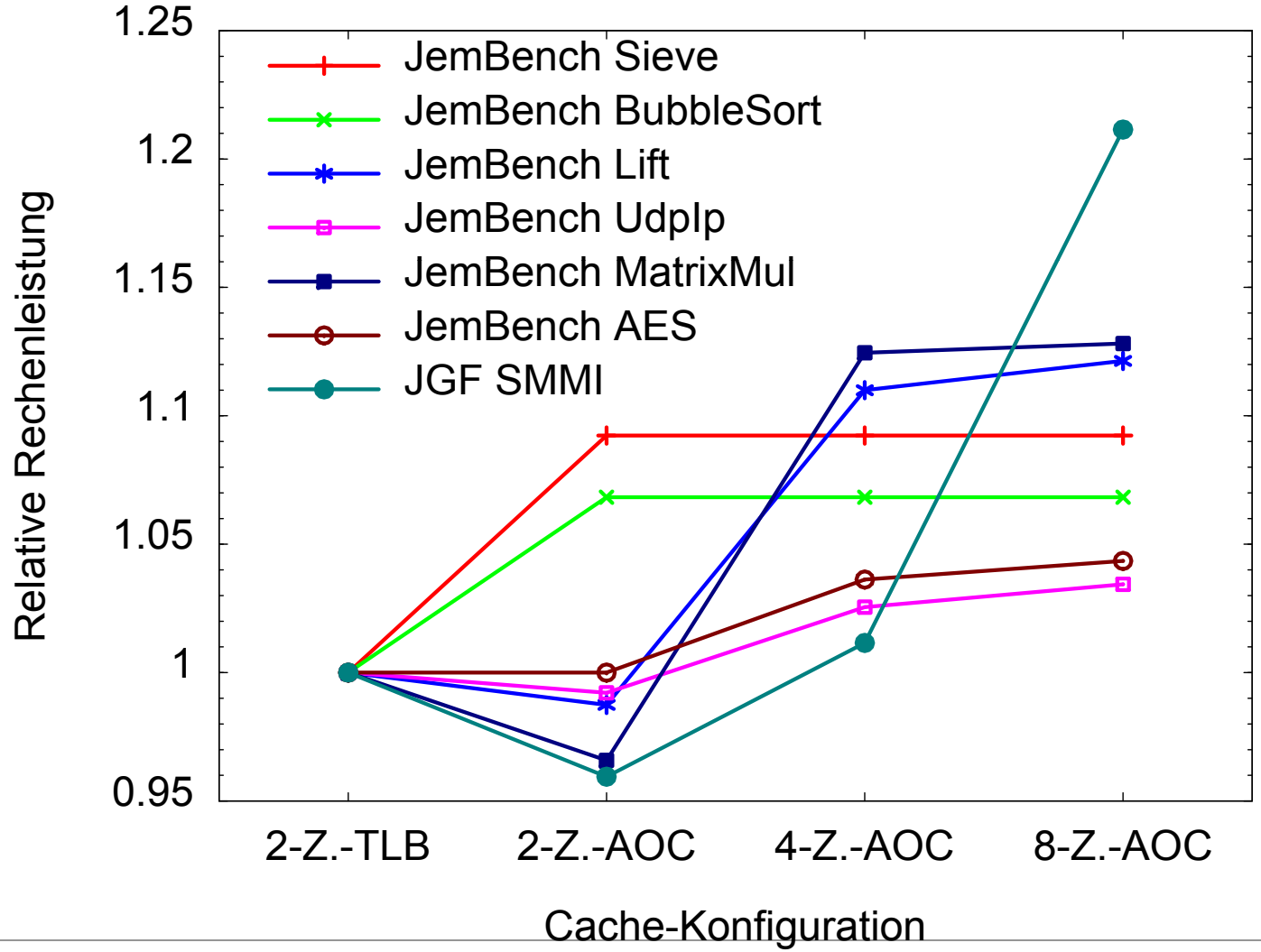
Basis:

bisherige Variante mit 2-Zeilen-TLB

Benchmarks:

- JemBench
- JavaGrande Framework [7], SparseMatMult m. Integer

Zusätzliche Latenz ab 4 Zeilen armortisiert.



Speed-Up der JemBench-MatrixMultiplikation

Matrixmultipl.:

Standard mit 20x20-Matrix

Basis:

Jeweilige Cache-Konfiguration mit einem Kern

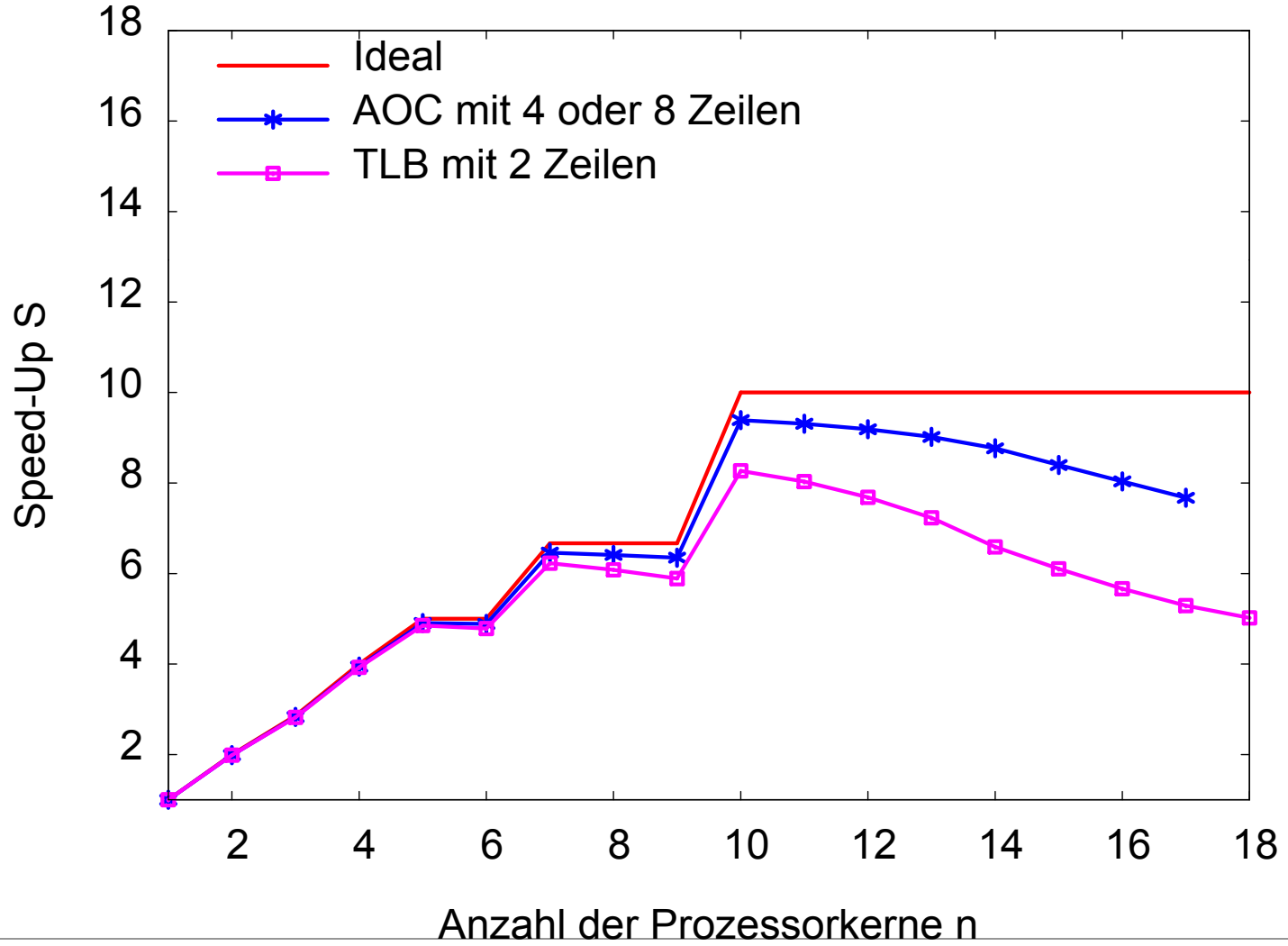
Idealer Speed-Up:

10 für ≥ 10 Kerne

Speed-Up-Max.:

8,3 ➔ 9,4

@ 10 Kerne



Speed-Up der JemBench-MatrixMultiplikation

Matrixmultipl.:

Erweitert mit 90x90-Matrix

Basis:

Jeweilige Cache-Konfiguration mit einem Kern

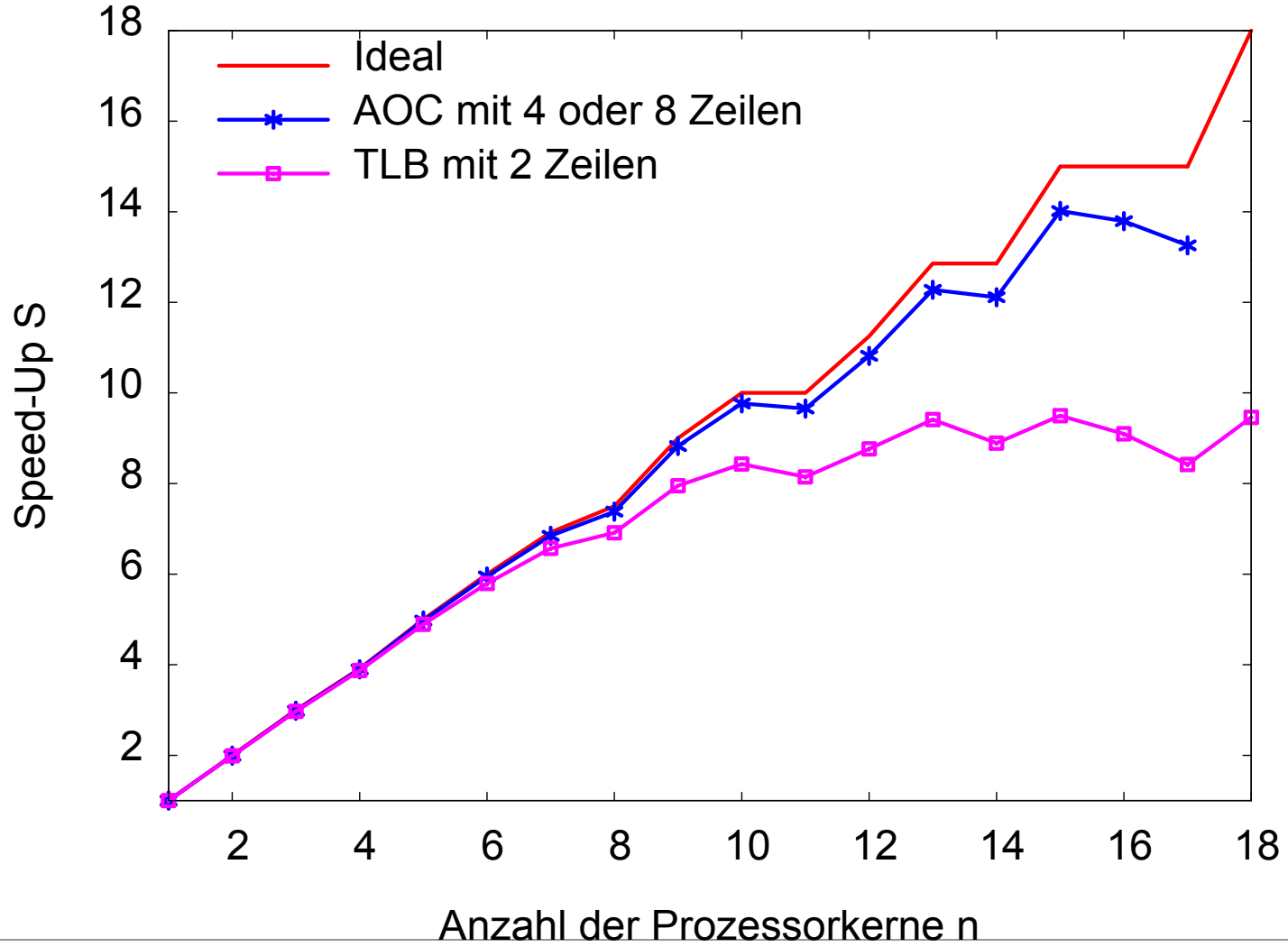
Idealer Speed-Up:

15 für 15--17 Kerne

Speed-Up-Max.:

9,5 ➔ 14,0

@ 15 Kerne



Speed-Up der JGF SparseMatMult mit Integer

Matrixmultipl.:

Reduziert auf 4.000x4.000-Matrix mit 16.000 besetzten Zellen, Integer

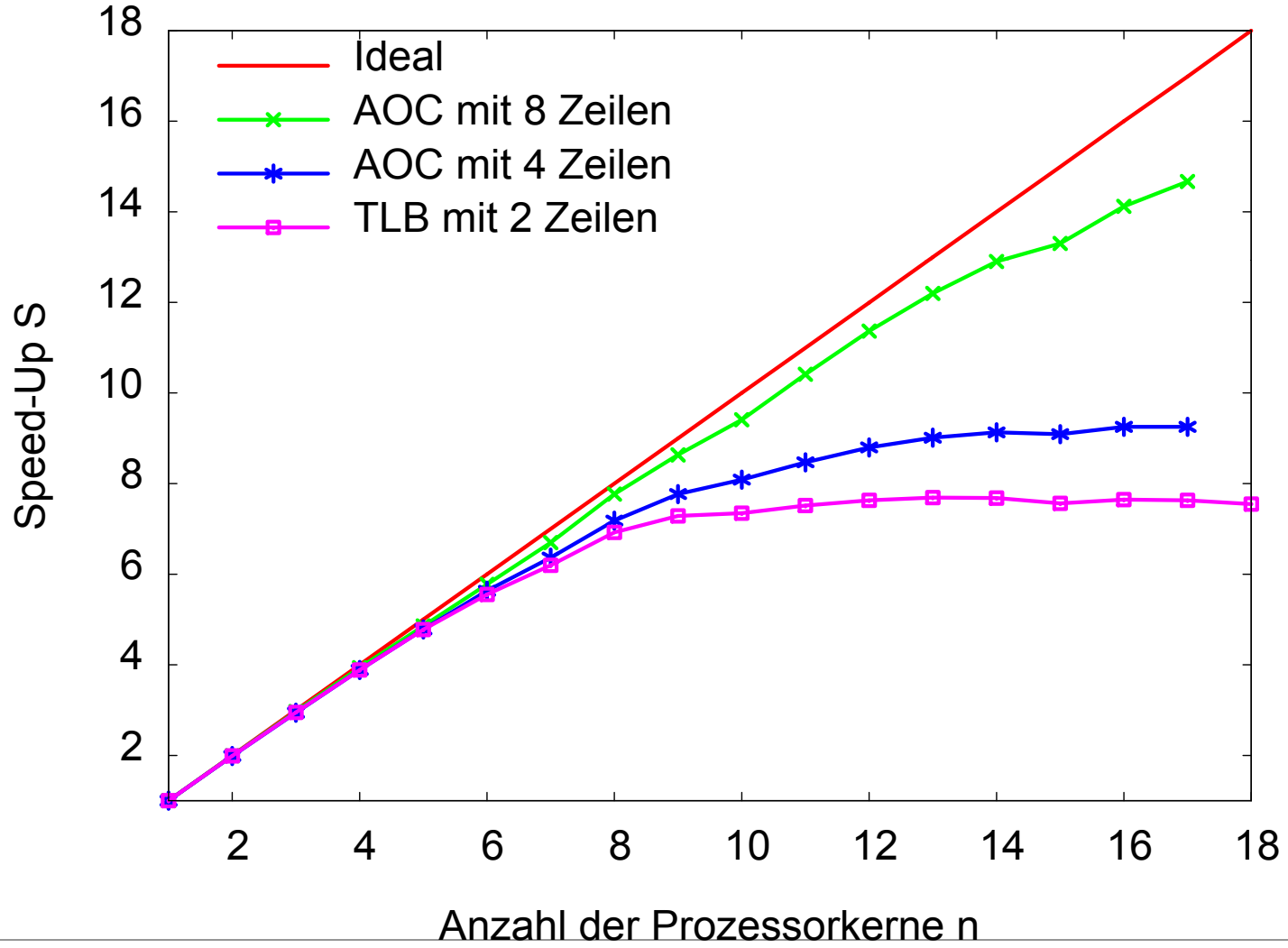
Basis:

Jeweilige Cache-Konfiguration mit einem Kern

Speed-Up-Max.:

7,6 ➔ 14,7

@ 17 Kerne



Gliederung

Zusammenfassung

Ausgangspunkt: SHAP-Mehrkernprozessor mit 2-Zeilen-TLB

Realisierter Objekt-Cache:

- Vollassoziativer kombinierter Adress- und Offset-Cache
- Konfigurierbar in Zeilenanzahl und Zeilengröße

Ergebnisse:

- Mit 4 Cache-Zeilen: Amortisierung der zusätzlichen Latenz
- Mit 8 Cache-Zeilen:
  - Maximaler Speed-Up gesteigert von ehemals 8--9 auf 14.
  - Beschleunigung des Einkernprozessors um 12 bis 21%.

➔Verdopplung der absoluten Verarbeitungsleistung von SparseMatMult.

➔Dafür nur 9% mehr Hardware-Ressourcen benötigt.

Vielen Dank für Ihre Aufmerksamkeit!

Fragen?

Ausgewählte Literatur

1. Zabel, Martin: Effiziente Mehrkernarchitektur für eingebettete Java-Bytecode-Prozessoren, Dresden, Technische Universität, Diss., 2012
2. Vijaykrishnan, N.; Ranganathan, N.: Supporting object accesses in a Java processor. In: *IE Proc. Computers and Digital Techniques* 147 (2000), Nr. 6, S. 435–443. – ISSN 1350–2387
3. Wright, G.; Seidl, M. L.; Wolczko, M.: An object-aware memory architecture. In: *Sci. Comput. Program.* 62 (2006), Nr. 2, S. 145–163. – ISSN 0167–6423
4. Huber, B.; Puffitsch, W.; Schoeberl, M.: WCET driven design space exploration of an object cache. In: *Proc. 8th Int'lWorkshop on Java Technologies for Real-Time and Embedded Systems (JTRES'10)*. ACM, 2010, S. 26–35
5. Lindholm, T.; Yellin, F.: The Java(TM) Virtual Machine Specification. 2nd edition. Amsterdam : Addison-Wesley Longman, 1999. – ISBN 978–0201432947
6. SCHOEBERL, Martin ; PREUSSER, Thomas B. ; UHRIG, Sascha: The embedded Java benchmark suite JemBench. In: *Proc. 8th Int'lWorkshop on Java Technologies for Real-Time and Embedded Systems (JTRES'10)*, S. 120–127
7. Java Grande Forum Benchmark-Suite. – http://www2.epcc.ed.ac.uk/computing/research_activities/java_grande/index_1.html