Analyse von Synchronisation

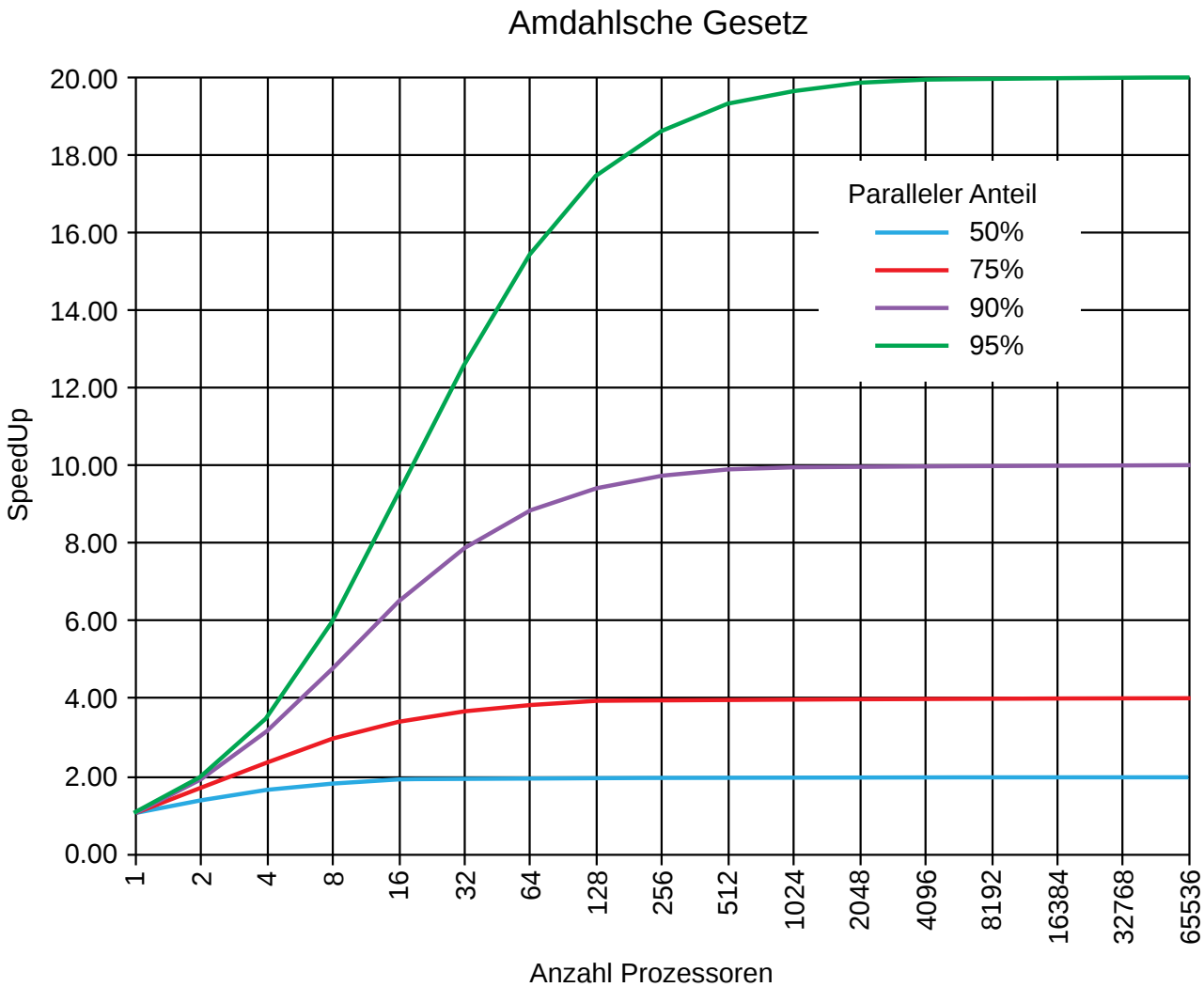
auf heutigen Prozessoren

Michael Panzlaff

Friedrich-Alexander Universität Erlangen-Nürnberg (FAU)

KVBK CS 4 - FAU

Amdahls Gesetz



[Com09]

Inhalt

- Überblick
- Testsysteme
- Messmethoden
- Speicherzugriffe
- Locks im Vergleich
- Nachrichten basierte Kommunikation
- Anwendungsebene

Synchronisation?

„Everything you always wanted to know about synchronization but were afraid to ask“ [DGT13]

Synchronisation?

„Everything you always wanted to know about synchronization but were afraid to ask“ [DGT13]

- Prozessoren bekommen immer mehr Kerne

Synchronisation?

„Everything you always wanted to know about synchronization but were afraid to ask“ [DGT13]

- Prozessoren bekommen immer mehr Kerne
- Speedup schwer zu erhalten mit Synchronisation (inbesondere Locks)

Synchronisation?

„Everything you always wanted to know about synchronization but were afraid to ask“ [DGT13]

- Prozessoren bekommen immer mehr Kerne
- Speedup schwer zu erhalten mit Synchronisation (inbesondere Locks)
- Wahl der Snychronisationsmethoden zunehmend bedeutender

Synchronisation?

„Everything you always wanted to know about synchronization but were afraid to ask“ [DGT13]

- Prozessoren bekommen immer mehr Kerne
- Speedup schwer zu erhalten mit Synchronisation (inbesondere Locks)
- Wahl der Snychronisationsmethoden zunehmend bedeutender
- Vergleich zwischen verschiedenen Anwendungsszenarien schwer [Boy+10]

Synchronisationsschichten

Tabelle: Synchronisationsebenen [DGT13, S. 33]

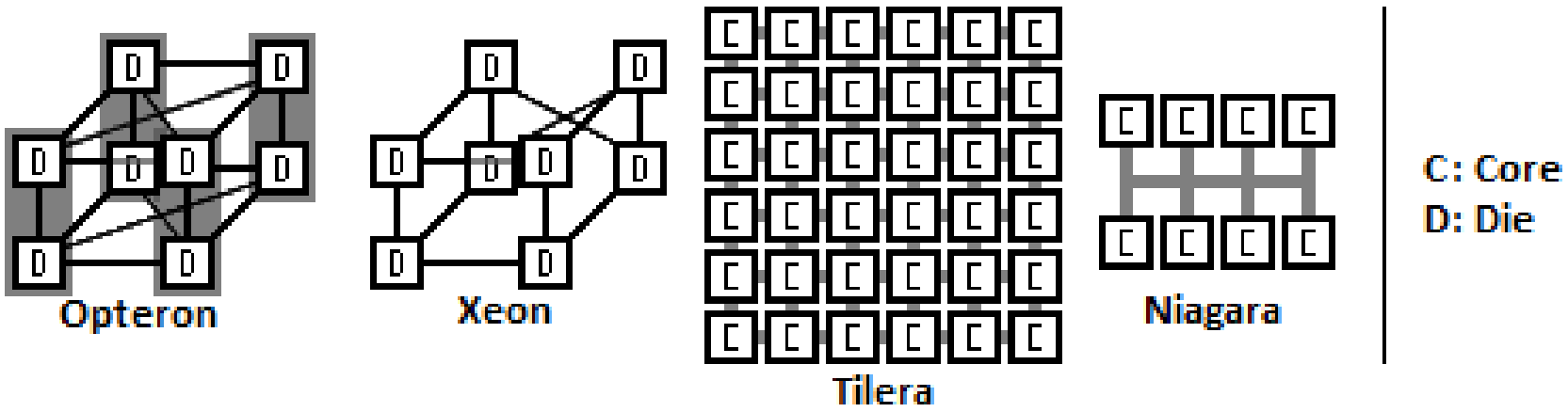
Ebene	**Beispiele**
Nebenläufige Software	Streutabelle, STM
Primitive	Locks, Message Passing
Atomare Operationen	CAS, FAI, TAS, SWAP
Cache Kohärenz	Laden, Speichern

Testsysteme

Systeme:

- 4x AMD Opteron (je 2x6 Kerne)
- 8x Intel Xeon (je 10 Kerne)
- Niagara (8 Kerne, je 8 Threads)
- Tilera (36 Kerne)

Testsysteme - Topologie



[DGT13] [Tec16] [Ora07]

Messaufbau

Messen der verschiedenen Ebenen:

Messaufbau

Messen der verschiedenen Ebenen:

- *ccbench* - Messung der Latenz von Cache-Kohärenz

Messaufbau

Messen der verschiedenen Ebenen:

- *ccbench* - Messung der Latenz von Cache-Kohärenz
- *libslock* - Schnittstelle für verschiedene Lock Implementierungen

Messaufbau

Messen der verschiedenen Ebenen:

- *ccbench* - Messung der Latenz von Cache-Kohärenz
- *libslock* - Schnittstelle für verschiedene Lock Implementierungen
- *libssmp* - Nachrichten basierte Kommunikation in Software

Messaufbau

Messen der verschiedenen Ebenen:

- *ccbench* - Messung der Latenz von Cache-Kohärenz
- *libslock* - Schnittstelle für verschiedene Lock Implementierungen
- *libssmp* - Nachrichten basierte Kommunikation in Software
- *ssht* - Threadsichere Streutabelle

Messaufbau

Messen der verschiedenen Ebenen:

- *ccbench* - Messung der Latenz von Cache-Kohärenz
- *libslock* - Schnittstelle für verschiedene Lock Implementierungen
- *libssmp* - Nachrichten basierte Kommunikation in Software
- *ssht* - Threadsichere Streutabelle
- *Memcached* - Cache System für z.B. Datenbanken

Speicherzugriffe

Tabelle: Cache und Speicher Latenzen [DGT13, S. 35]

	Opteron	**Xeon**	**Niagara**	**Tilera**
L1	3	5	3	2
L2	15	11	-	11
LLC	40	44	24	45
RAM	136	355	176	118

Speicherzugriffe

Tabelle: Cache und Speicher Latenzen [DGT13, S. 35]

	Opteron	**Xeon**	**Niagara**	**Tilera**
L1	3	5	3	2
L2	15	11	-	11
LLC	40	44	24	45
RAM	136	355	176	118

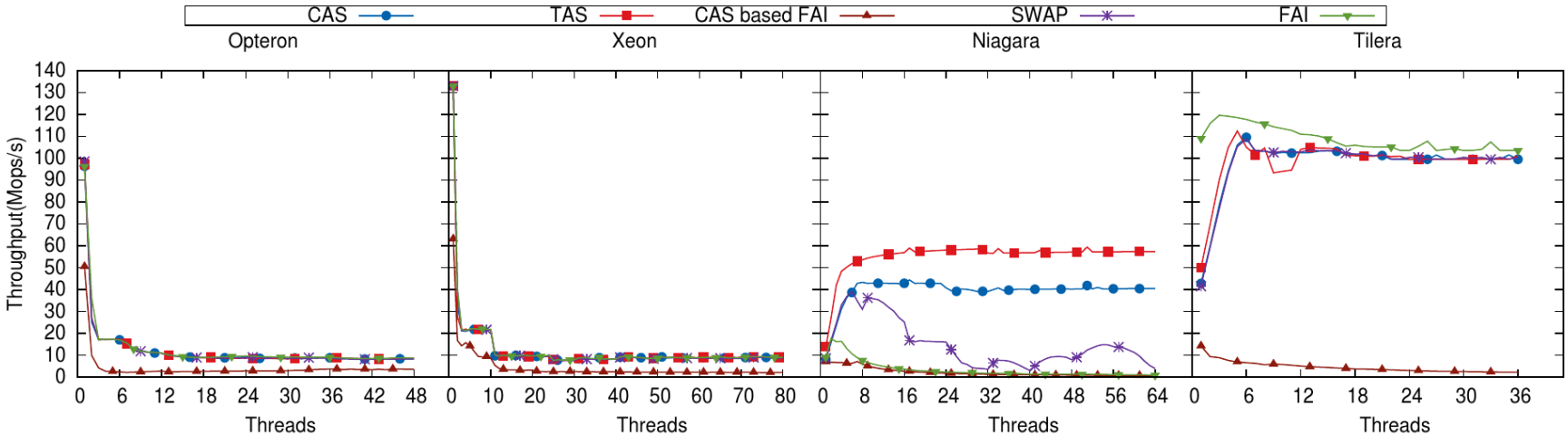
Niagara sticht durch seinen uniformen Speicherzugriff heraus.

Cache Kohärenz

Durchschnittliche Messungen:

- **Opteron:** Speicherzugriff über benachbarten *Die* verhält sich sehr ähnlich wie über einen benachbarten Sockel.
- **Xeon:** Ähnlich zum Opteron. Verhält sticht nicht viel schlechter als der Opteron trotz doppeleter Anzahl an Sockeln.
- **Niagara:** Von allen am schnellsten. Profitiert auch hier vom uniformen Speicherzugriff
- **Tilera:** Ist trotz seines Zugriffs über mehrere Knoten sehr schnell.

Atomare Operationen - Durchsatz



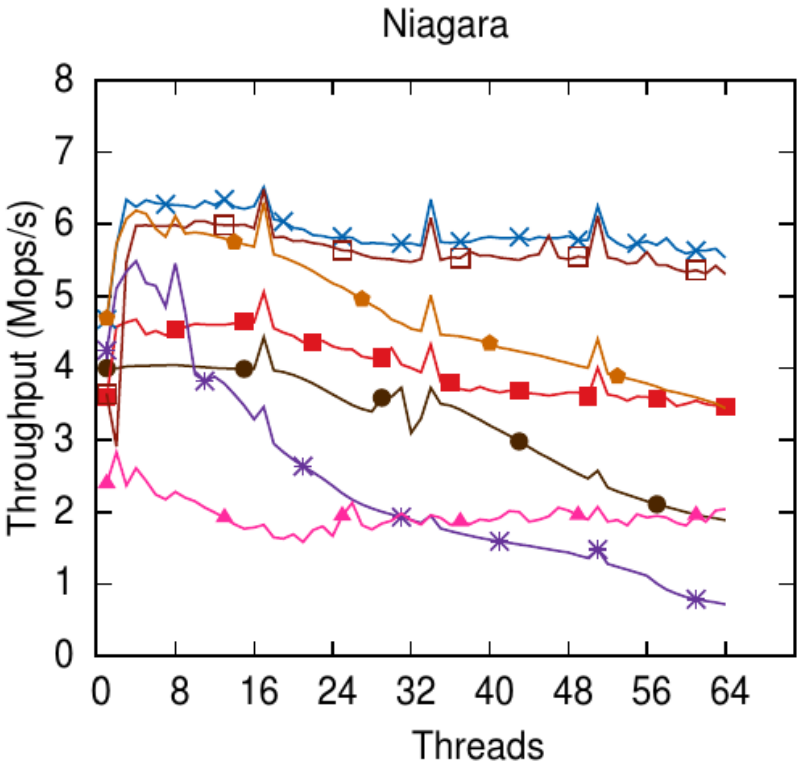
[DGT13, S. 40]

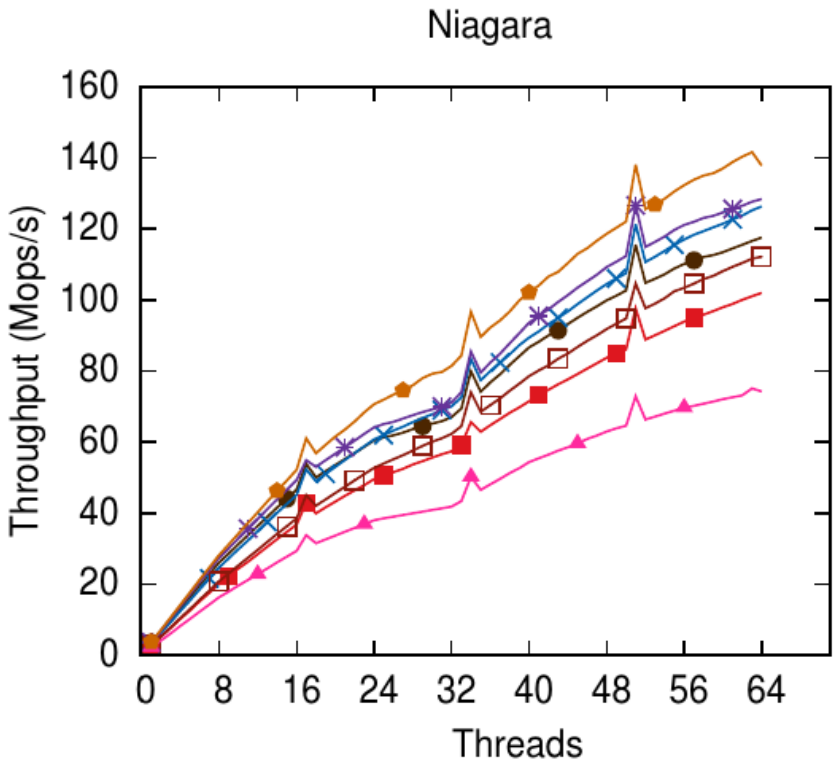
Atomare Operationen

- **Opteron:** Hoher Ein-Thread Durchsatz. Fällt zudem auffällig nach 6 Threads, da mehrere *Dies* kommunizieren müssen.
- **Xeon:** Ähnliches Verhalten wie beim Opteron. Hier Abfall sobald mehrere Sockel im Spiel sind.
- **Niagara:** Profitiert von direktem Speicherzugriff.
- **Tilera:** Hebt sich von allen am meisten ab trotz seiner aufwändig vermaschten Struktur.

Locks im Vergleich

links: Durchsatz auf einem Lock, **rechts:** Durchsatz auf 512 Locks





[DGT13, S. 41-42]

Locks im Vergleich

$\Rightarrow$ Blick auf die Skala sagt viel!

- Bei weniger Konflikt steigt der Durchsatz viel.
- **Opteron** und **Xeon** zeigen wieder „Stufe“ für hohen Wettbewerb.
- **Niagara** profitiert von schneller Cache Kohärenz bei hohem Konflikt.
- **Tilera** ist aufgrund seines nicht-uniformen Speicherzugriffs nicht ganz so schnell wie der **Niagara**.
- Für wenig Konflikt ist der **Niagara** und **Tilera** bei gleich Threadanzahl sehr ähnlich schnell.
- Teilweise ähnliche Ergebnisse zu Prozessoren aus gleicher Generation [SBH15]

Nachrichtenbasierte Kommunikation

- Implementiert über Nachrichten gepackt in eine Cache-Zeilen.
- Flag bestimmt ob Nachricht gültig ist.
- Zwei Arten von Kommunikation wurden getestet:
  - **Punkt-zu-Punkt Kommunikation**
  - **Client-Server Kommunikation**

Nachrichtenbasierte Kommunikation

- Leistung ähnlich für Punkt zu Punkt sowie für Client-Server Kommunikation. Ausnahme hier der Xeon bei Client-Server Nachrichten auf dem gleichen *Die*.
- **Opteron**, **Xeon** und **Niagara** liegen etwa gleichauf für Latenzen innerhalb eines Prozessors.
- Schlecht schneidet der **Opteron** und der **Xeon** sobald zu anderen Sockeln kommuniziert wird.
- Der **Tilera** ist aufgrund seiner vermaschten Struktur mit Abstand am schnellsten.

Anwendungsebene

- Messung mittels *ssht*:
  - Testlauf mit 12 und 512 Behältern. Dies jeweils mit 12 und 48 Einträgen pro Behälter.
    - Für viele Threads ist skaliert die Leistung mit 512 besser, für weniger Threads eher mit 12.
    - 12 Einträge pro Behälter sind tendentiell bei allen Systemen schneller als 48 (geringerer Verwaltungsaufwand)
- Messung mittels *Memcached*:
  - **Get-Only-Test:** Nicht sonderlich interessant. Flaschenhals ist nicht die Synchronisation.
  - **Set-Only-Test:** Beanspruchung der Synchronisation (Aquirierung eines globalen Locks).
    Skalierung wie zu erwarten wieder schlecht bei zu vielen Threads auf den Multi-Sockel-Plattformen.
    Durchsatz bei den Ein-Sockel-Systemen eher gering (nicht optimiert?)

Fazit

- Kommunikation über mehrere Sockel ist langsam.
- Vermaschte Architekturen zeigen ihre Vorteile bei nachrichtenbasierter Kommunikation.
- Hoher Konflikt wirkt sich in der Regel schlecht auf den Durchsatz aus.

Fragen?
Fragen!

Literatur I

[Boy+10] Silas Boyd-Wickizer, Austin T Clements, Yandong Mao, Aleksey Pesterev, M Frans Kaashoek, Robert Morris, Nickolai Zeldovich u. a. „An Analysis of Linux Scalability to Many Cores.“ In: *OSDI*. Band 10. 13. 2010, Seiten 86–93.

[Com09] Wikimedia Commons. *Der Geschwindigkeitsgewinn bei der Verwendung von parallel arbeitenden Prozessoren bei der Bearbeitung eines parallelen Problems*. File: AmdahlsLaw.svg. 2009. URL: https://de.wikipedia.org/wiki/Amdahlsches_Gesetz#/media/File:AmdahlsLaw_german.svg.

[DGT13] Tudor David, Rachid Guerraoui und Vasileios Trigonakis. „Everything You Always Wanted to Know About Synchronization but Were Afraid to Ask“. In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, Seiten 33–48. DOI: 10.1145/2517349.2522714. URL: http://doi.acm.org/10.1145/2517349.2522714.

[Ora07] Oracle. *UltraSPARC T2: A Highly-Threaded, PowerEfficient, SPARC SOC*. 2007. URL: http://www.oracle.com/technetwork/systems/opensparc/02-t2-a-sscc2007-1530395.pdf.

Literatur II

[SBH15] H. Schweizer, M. Besta und T. Hoefler. „Evaluating the Cost of Atomic Operations on Modern Architectures“. In: *2015 International Conference on Parallel Architecture and Compilation (PACT)*. 2015, Seiten 445–456. DOI: 10.1109/PACT.2015.24.

[Tec16] Mellanox Technologies. *TILE-Gx36 Processor*. 2016. URL: https://www.mellanox.com/related-docs/prod_multi_core/PB_TILE-Gx36.pdf.