Sperren und deren Einfluss auf die Leistung von Mehrkern-Rechnern

Eine Ausarbeitung im Rahmen des KvBK-Seminars

Markus Büttner
Friedrich-Alexander Universität Erlangen-Nürnberg
markus.buettner@fau.de

UMRISS

In dieser Arbeit wird eine vorliegende Untersuchung zum Thema Sperren auf NUMA Mehrkern-Rechnern behandelt. Dazu werden zunächst die Grundlagen der Arbeit und die Vorkehrungen, die nötig waren, um die erforderlichen Messungen durchführen zu können, zusammengefasst dargestellt und an entscheidenden Stellen mit einer weiteren umfangreichen Arbeit zu diesem Thema verglichen. Anschließend werden einige der Ergebnisse näher erläutert und Abweichungen von der vorangegangenen Arbeit gegenübergestellt.

1. EINFÜHRUNG

Um die sich seit Jahren auf dem Vormarsch befindenden NUMA[1] Mehrkern-Rechner effektiv nutzen zu können, muss bei der Entwicklung entsprechender Software mit besonderer Sorgfalt vorgegangen werden. Insbesondere durch Sperren (engl. Locks), die in kritischen Abschnitten zur Prozesssynchronisation verwendet werden, können dabei Leistungsverluste auftreten. Bisherige Arbeiten, die Vergleiche zwischen den verschiedenen Sperr-Algorithmen anstellen, beschränken sich dabei auf eine kleine Menge an verschiedenen Typen von Algorithmen, oder lassen weitere Faktoren, die den Vergleich beeinflussen können, außer Acht. So werden beispielsweise lastkontrollierende oder hierarchische Algorithmen nur dürftig behandelt, oder ihre Leistung lediglich anhand von Mikrobenchmarks festgestellt. Des Weiteren wurden neue Algorithmen hauptsächlich unter den Lastbedingungen getestet, für die sie ursprünglich konzipiert waren. Die untersuchte Arbeit[9] macht es sich deshalb zur Aufgaben, eine große Anzahl an Algorithmen unter realistischen Bedingungen zu testen und gegenüberzustellen. Dabei wurden insgesamt 27 verschiedene Algorithmen anhand 35 realistischer Anwendungen getestet. Die verwendeten Anwendungen stammen dabei aus den PARSEC[2], Phoenix 2 und SPLASH-2 Benchmark-Zusammenstellungen und werden noch um MySQL und einen SSL-Proxy ergänzt. Zur Durchführung wird sich dabei des Linux-Systemkerns und der x86-Architektur bedient.
Da eine manuelle Modifikation der verwendeten Anwendungen großen Aufwand mit sich gebracht hätte und bei der Verwendung bestehender Rahmenwerke wichtige Funktionen wie Pthread Zustandsvariablen nicht, oder nur eingeschränkt verfügbar sind, wurde die Untersuchung stattdessen mit der eigens implementierten LiTL[3]-Bibliothek durchgeführt. Damit lassen sich unter Aufrechterhaltung von Funktionen wie Zustandsvariablen, beliebige Sperren zusätzlich zu Pthread-Sperren zwischenschalten.
Während der Durchführung wurden unter anderem beobachtet, dass[9]:

1. ungefähr 60% der untersuchten Anwendungen stark von der verwendeten Sperre beeinflusst werden.
2. keine Sperre für alle Anwendungen die beste Leistung erbringt.
3. es keine eindeutige, globale Rangfolge bezüglich der Leistungsfähigkeit der Sperren gibt.
4. die beste Sperre für eine Anwendung sich häufig mit der Anzahl der beteiligten Knoten und der verwendeten Maschine ändert.
5. jede Sperre mehrere Anwendungen negativ beeinflusst.
6. Pthread-Sperren nicht aufgrund schlechter Leistung vermieden werden müssen[8].

Nachfolgend wird sich zunächst unter §2 mit den untersuchten Algorithmen beschäftigt. §3 befasst sich mit der verwendeten Hard- und Software und der Art und Weise, wie zur Ermittlung der Ergebnisse vorgegangen wurde. Unter §4 werden die Fragestellungen und die Ergebnisse der Untersuchung dargestellt. Nennenswerte Gemeinsamkeiten oder Unterschiede zu anderen Arbeiten, insbesondere Tudor et. al.[8], werden an den entsprechenden Stellen eingebracht und anschließend unter §5 in der Bewertung und dem Vergleich ergänzt. §6 schließt die Arbeit ab.

2. SPERR-ALGORITHMEN

2.1 Klassifizierung

Die behandelte Arbeit befasst sich mit einem großen Ausschnitt aus der Menge der heute zur Verfügung stehenden Sperren. Für die Leistungsermittelung ist die Kenntnis über die genaue Funktionsweise der einzelnen Sperren nicht nötig. Dennoch werden diese in fünf verschiedene Klassen eingeordnet:
Zu den **flachen** (engl. Flat) Ansätzen gehören beispielsweise die Pthread-Sperre oder TTAS[4]. Dabei handelt es sich um einfache Algorithmen, die sich lediglich einer oder mehrerer

[1]Non-uniform memory access
[2]Princeton Application Repository for Shared-Memory Computers
[3]Library for Transparent Lock inpterposition
[4]Test and test-and-set

gemeinsamer Variablen bedienen, die durch atomare Aktionen gelesen und geschrieben werden.

Warteschlangenbasierte (engl. Queue-based) Ansätze hingegen verwenden eine Warteschlange, um eine gerechtere Ordnung und ein effizienteres Warten der Fäden zu ermöglichen. Ein Vertreter dieser Klasse ist MCS[5], das die Warteschlange mit Hilfe einer verketteten Liste implementiert[7].

Die **hierarchischen** (engl. Hierarchical) Ansätze bedienen sich einer mehrstufigen Anordnung gleicher oder verschiedener Sperren und sind speziell für große NUMA-Rechner gedacht. Ein auf das bereits erwähnte MCS aufbauendes Beispiel für diese Kategorie ist HMCS. Entgegen dem üblicheren Vorgehen, eine Hierarchie aus einer Sperre pro NUMA-Knoten und einer globalen Sperre zu verwenden[12], wird hier eine N-stufige Hierarchie aus MCS-Sperren verwendet, wodurch einer Anwendung eine feingranularere Optimierung bezüglich der Lokalität ermöglicht wird[6].

Lastkontrollierende (engl. Load-control) Ansätze bauen auf den warteschlangenbasierten Ansätzen auf, versuchen aber die Anzahl der gleichzeitigen Zugriffe auf eine Sperre zu minimieren. Ein ebenfalls auf MCS basierender Vertreter dieser Art ist MCS-TP. MCS wird hierbei um Funktionalität erweitert, die ein Ausscheiden und einen Wiedereintritt in die Warteschlange aufgrund von Zeitüberschreitung ermöglichen[13].

Bei **delegationsbasierten** (engl. Delegation-based) Algorithmen wird die eigentliche Ausführung des kritischen Abschnittes einem anderen Faden überlassen. Eine Implementierung dieses Verfahrens stellt RCL dar[5].

Neben der Klassifizierung nach der Funktionsweise der Sperren, lassen sich diese zusätzlich anhand der Strategie, mit der auf das erfolgreiche Belegen einer Sperre gewartet wird, unterscheiden. **Umlaufsperren** (engl. Spinlock) rotieren dabei lediglich auf einer Speicheradresse. **Blockierende Sperren** hingegen pausieren den Faden bis er die Möglichkeit erhält, die Sperre zu belegen, oder bis eine gewisse Zeitspanne vergangen ist. **STP**[6] stellt eine Kombination aus beiden Verfahren dar.

2.2 Ausgewählte Algorithmen

Bei der Auswahl der Algorithmen wurden solche, die sehr spezielle Anforderungen an die Umgebung haben, oder sehr spezielle Einstellungen benötigen und damit nur sehr eingeschränkt portable sind, nicht berücksichtigt. Insgesamt wurden so 27 verschiedene Algorithmen getestet, wobei Algorithmen, die mit unterschiedlichen Wartestrategien und Optimierungen getestet wurden, jeweils separat gezählt werden. Bis auf die lastkontrollierenden Algorithmen sind alle erwähnten Klassen mehrfach vertreten. Das Spektrum der abgedeckten Algorithmen beschränkt sich also nicht nur auf eine kleine Menge an Algorithmen, die zu den am häufigsten verwendeten Algorithmen zählen[8], sondern befasst sich auch mit Algorithmen, die ansonsten weniger Beachtung finden.

[5]Mellor-Crummney Scott
[6]Spinning-then-parking

3. UMSETZUNG

Zur Umsetzung wurde sich verschiedener Hardware und Software in verschiedenen Konfigurationen bedient. Diese werden im Folgenden näher erläutert.

3.1 Testumgebung

3.1.1 Hardware

Die verwendete Hardware setzte sich aus zwei AMD-basierten Maschinen mit 64 (A-64) und 48 (A-48) Kernen und einer Intel-basierten Maschine mit 48 (I-48) Kernen zusammen. Die präsentierten Ergebnisse beziehen sich größtenteils auf die A-64-Maschine, wurden jedoch ebenso mit den anderen Maschinen durchgeführt und in einem erweiterten Bericht bereitgestellt[10].

3.1.2 Software

Alle Konfigurationen wurden mit Ubuntu 12.04 in Kombination mit dem Linux Kernel in der Version 3.17.6 betrieben. Die für die Messungen benötigte Last soll realen Einsatzszenarien nachempfunden sein und wurde deshalb durch eine spezielle Auswahl an Software erzeugt:

Die **PARSEC** Benchmark-Zusammenstellung ist eine im Rahmen einer Dissertation an der Princeton Universität beschriebene Zusammenstellung von Anwendungen aus Bereichen wie der Medienverarbeitung oder der Animation physikalischer Prozesse. Durch die hohe Vielfalt an Anwendungen sollen möglichst viele Lastszenarien simuliert werden können[1].

Phoenix 2 stellt eine Version des von Google entwickelten MapReduce-Systems dar. Diese wurde jedoch für die Verwendung auf Mehrkern-Rechnern mit gemeinsamem Speicher optimiert[4].

SPLASH 2 ist eine Zusammenstellung die in der Zielsetzung PARSEC ähnelt, jedoch älter ist und andere Anwendungen enthält[11].

Bei der Kombination **MySQL** mit der Cloudstone-Auslastung handelt es sich um eine Zusammenstellung, die die Datenbank durch gleichzeitige Abfragen von mehreren simulierten Nutzern belastet[2].

Ein **SSL-Proxy** ist eine Anwendung, die per SSL verschlüsselte Nachrichten entgegennimmt und diese entschlüsselt und an das entsprechende Ziel hinter dem Proxy weiterleitet. Da hierbei gleichzeitig sehr viele Nachrichten anfallen können, ergibt sich eine besondere Eignung zum Test von Sperren.

Bis auf wenige Ausnahmen verwenden die Anwendungen standardmäßig eine Anzahl an Fäden, die gleich der Anzahl der verfügbaren Kerne ist. Ein kleiner Teil der Anwendungen ist nicht mit der A-48 und der I-48 Maschine kompatibel, da die Anzahl der verfügbaren Kerne einer Zweierpotenz entsprechen muss. Der Einfluss des Wettbewerbsniveaus (engl. contention level) auf die getesteten Sperren wurde ermittelt, indem die Anzahl der für die Anwendung verfügbaren Kerne, je nach Maschine, um die Anzahl der Kerne in einem NUMA-Knoten[7] variiert wurde.

[7]I-48: 12, A-48: 6, A-64: 8

Mit dieser großen Auswahl an Software muss sich die Arbeit also nicht auf die Ergebnisse von Mikrobenchmarks verlassen, deckt aber durch den Mangel an verschiedenen Plattformen wiederum nur einen eingeschränkten Bereich ab[8]. Es kann hier also davon ausgegangen werden, dass die Ergebnisse der Untersuchung womöglich nicht in allen Fällen ernstzunehmende Aussagen darüber sind, ob und unter welchen Bedingungen die jeweiligen Sperren in der realen Welt effektiv einsetzbar sind.

3.2 LiTL

Um die nötigen Modifikationen am Code der verwendeten Software gering zu halten, wurde die LiTL-Bibliothek zur Durchführung der Messungen implementiert. Sie erlaubt das Zwischenschalten von Sperren zu bestehenden Pthread-Sperren, ohne viel zusätzlichen Rechenaufwand hinzuzufügen oder die Funktionen der zugeschalteten Sperre einzuschränken.
Die LiTL-Bibliothek wird durch eine Hash-Tabelle gestützt, durch die die jeweilige Pthread-Sperre mit der einzusetzenden Sperre assoziiert wird. Um alle in der Anwendung verwendeten Sperren in die Tabelle eintragen zu können, muss die LiTL-Bibliothek die Erzeugung und Zerstörung von Pthread-Sperren ebenso durch Zwischenschaltung überwachen. Um

```
1  pthread_mutex_lock(pthread_mutex_t *m) {
2    optimized_mutex_t *om =
       get_optimized_mutex(m);
3    if (om == null) {
4      om = create_and_store_optimized_mutex(m
       );
5    }
6    optimized_mutex_lock(om);
7    real_pthread_mutex_lock(m);
8  }
9
10 pthread_mutex_unlock(pthread_mutex_t *m) {
11   optimized_mutex_t *om =
       get_optmized_mutex(m);
12   optimized_mutex_unlock(om);
13   real_pthread_mutex_unlock(m);
14 }
15
16 pthread_cond_wait(pthread_cond_t *c,
       pthread_mutex_t *m) {
17   optimized_mutex_t *om =
       get_optimized_mutex(m);
18   optimized_mutex_unlock(om);
19   real_pthread_cond_wait(c, m);
20   real_pthread_mutex_unlock(m);
21   optimized_mutex_lock(om);
22   real_pthread_mutex_lock(m);
23 }
```

Abbildung 1: Skizze der Funktionsweise der LiTL-Bibliothek (Pseudocode)[9]

mit Zustandsvariablen umgehen zu können, ohne diese in jeder Sperre, die diese nicht unterstützt, aufwändig behandeln zu müssen, bedient sich die LiTL-Bibliothek der Pthread Zustandsvariablen. Um diese nutzen zu können, muss jedoch zusätzlich zu jeder Sperre eine Pthread-Sperre gehalten werden. Dazu wird beim Belegen einer Sperre neben der eigentlichen Sperre auch jeweils noch die zugehörige Pthread-Sperre belegt (Abb.1 Z. 6f) und bei der Freigabe der Sperre ebenso wieder freigegeben (Abb.1 Z. 12f). Dieses Vorgehen führt zu einem geringen Mehraufwand, nicht jedoch zu einem ungünstigen Wettbewerb um die jeweilige Pthread-Sperre, da diese nur nach erfolgreichem Belegen der eigentlichen Sperre angefordert wird. Der Funktionsaufruf der originalen[8] `pthread_cond_wait()` findet in der entsprechenden Funktion der LiTL-Biblithek statt (Abb.1 Z. 18-22), wodurch Zustandsvariablen letztlich unabhängig von der Art der zwischengeschalteten Sperre nutzbar werden. Um die Effektivität der Zwischenschaltung mit der LiTL-Bibliothek zu verifizieren, wurden drei besonders ungünstige Anwendungen manuell mit allen ausgewählten Sperren modifiziert und anschließend mit den Ergebnissen der Zwischenschaltung verglichen. Die durchschnittliche Leistungsdifferenz lag dabei unter 5%, wobei dennoch mit wenigen Ausnahmen der erwartete Trend, dass die Zwischenschaltung schlechter abschneidet, bestätigt wurde.

4. LEISTUNGSMESSUNG

Zur Messung der Leistung der einzelnen Sperren wurde jeder Test fünfmal durchgeführt und der Durchschnitt gebildet. Die Maßeinheit der Messungen ist *Operationen pro Zeiteinheit*. Der folgende Abschnitt befasst sich mit den Ergebnissen der Messungen. Diese werden zusammen mit den Abweichungen von anderen Arbeiten präsentiert. Um die Ergebnisse einordnen zu können, müssen jedoch vorher noch einige Rahmenbedingungen der Autoren erwähnt werden[9]:

- Eine Sperre wurde im Vergleich als besser betrachtet, wenn sie mindestens 5% mehr Leistung erbringt als die andere Sperre.
- Auch die Pthread-Sperre wurde mit der LiTL-Bibliothek zwischengeschaltet, um den gleichen Mehraufwand zu erhalten.
- Aufgrund von Inkompatibilitäten wurde auf das Testen einiger Kombinationen aus Sperre und Anwendung verzichtet.

Anhand dieser Bedingungen wird sichergestellt, dass die Ergebnisse der Messungen nicht durch minimale Leistungsunterschiede zwischen zwei Sperren oder gänzlich ungeeignete Sperren unnötig verzerrt werden.

4.1 Vorausgehende Überlegungen

4.1.1 Relevante Anwendungen

Aus dem bereits präsentierten Spektrum an Anwendungen sind für die Messungen nur diejenigen interessant, deren Leistung sich in Abhängigkeit der gewählten Sperre merklich verändert. Als Maß dafür verwenden die Autoren die relative Standardabweichung der Leistung unter Verwendung der maximalen Anzahl an Knoten. Diese sollte mindestens 10% betragen, damit eine Anwendung als beeinflussbar angenommen wird. Nach diesem Kriterium sind nur 23 der Anwendungen für die weiteren Messungen mit A-64 relevant.

4.1.2 Optimale Knotenanzahl

[8]durch die glibc bereitgestellt

	A-64	A-48	I-48
1 Node	11%	9%	33%
2 Nodes	28%	24%	14%
3 Nodes	-	-	8%
4 Nodes	27%	21%	45%
6 Nodes	7%	9%	-
8 Nodes	27%	37%	-

Tabelle 1: Verteilung der optimalen Knotenanzahlen unter allen Kombinationen aus Anwendung und Sperre[9]

Da die Leistung paralleler Anwendungen meist nicht beliebig mit der Anzahl der verwendeten Knoten skaliert, bietet es sich an, neben der Messung mit einer festgesetzten Anzahl an Knoten[9], auch mit einer Variablen Anzahl an Knoten zu messen. Diese muss für jede Kombination aus Anwendung und Sperre so gewählt werden, dass die größte mögliche Leistung erreicht wird. Dabei zeigt sich (Tab. 1), dass in den meisten Fällen die höchste Leistung nicht mit der maximalen Anzahl an Knoten erreicht wird[9]. Eine Erklärung dafür wäre der Mehraufwand, der bei der parallelen Arbeit durch die Kommunikation und den Datenaustausch zwischen den einzelnen Knoten entsteht und die Zugewinne an Rechenleistung dadurch wieder relativiert[8].

4.2 Ergebnisse

4.2.1 Optimierungspotential

Der Aufwand, die beste Sperre für eine Anwendung zu finden, lohnt sich insbesondere, wenn davon ausgegangen werden kann, dass die richtige Wahl überhaupt einen messbaren Einfluss auf die Leistung der Anwendung hat. Die ermittelten Werte zeigen, dass besonders bei der Ausführung mit maximaler und optimierter Knotenanzahl ein gravierender Leistungszuwachs möglich ist, wenn die richtige Sperre gewählt wird. In manchen Fällen erbringt die besten Sperre mehr als die zehnfache Leistung der schlechtesten Sperre[9, 10]. Es zeigt sich also deutlich, dass der Einfluss durch die Wahl der Sperre keineswegs zu unterschätzen ist. Daraus und aus dem hohen Anteil an beeinflussbaren Anwendungen ergibt sich die erste Beobachtung: **Ungefähr 60% der untersuchten Anwendungen werden stark von der verwendeten Sperre beeinflusst.**

4.2.2 Beste Sperre

Eine Sperre, deren Leistung grundsätzlich für alle Anwendungen gleichbleibend hoch ist, wäre bei der Entwicklung paralleler Anwendungen ein großer Vorteil. Aufwändige Untersuchungen zur Ermittlung der optimalen Sperre wären damit zumindest während der anfänglichen Entwicklung weniger wichtig und könnten zugunsten anderer Arbeiten aufgeschoben werden. Wie sich aber zeigt, kann sich auch bei optimaler Anzahl an Knoten keine Sperre derartig durchsetzen. AHMCS erzielt bei 52% der getesteten Anwendungen in diesem Fall eine der besten Leistungen und setzt sich dabei nicht merklich von den erzielten 48% mehrerer anderer Sperren ab[9]. Diese Messungen decken sich mit den Aussagen von Tudor et. al., wobei dort noch zusätzlich darauf hingewiesen wird, dass auch einfach aufgebaute Sperren sehr gute Ergebnisse erzielen können[8]. Daraus folgt die zweite Beobachtung: **Keine Sperre erbringt für alle Anwendungen die beste Leistung.**

[9]ein Knoten oder alle auf der Maschine verfügbaren Knoten

4.2.3 Leistungshierarchie

Auch wenn es nicht möglich ist, eine generell beste Sperre zu finden, könnte durch eine klare Rangfolge unter den Sperren zumindest die Auswahl anhand der Platzierung in der Rangfolge und dem jeweiligen Aufwand bei der Implementierung erleichtert werden. Hier zeigen die Messungen aber ebenso, dass keine klaren Aussagen getroffen werden können. Im paarweisen Vergleich ist der Anteil der Anwendungen für die eine Sperre bessere Ergebnisse liefert ähnlich dem Anteil in dem sie unterliegt[9]. Da für jede Anwendung auch Sperren existieren, die verhältnismäßig gut geeignet sind, kann auf dieses Problem mit der individuellen Auswahl der Sperre anhand der Bedingungen in der Anwendung reagiert werden[8]. Eine weitere Überlegung hierzu wäre reaktive Synchronisation, bei der diese Auswahl zur Laufzeit geschieht[3]. Das führt zur dritten Beobachtung: **Es ist keine eindeutige, globale Rangfolge bezüglich der Leistungsfähigkeit der Sperren erkennbar.**

4.2.4 Einfluss von Hardware und Konfiguration

Ebenso wie potentielle Gewinne durch die richtige Sperre muss in Betracht gezogen werden, dass die unter den falschen Bedingungen durch ungünstige Auswahl der Sperre viel Leistung verloren gehen kann. Das stellt insbesondere ein Problem dar, wenn die Bedingungen, unter denen eine Sperre verwendet wird, variabel sind. Das wird deutlich, wenn man betrachtet, dass sich die Rangliste der Sperren für eine Anwendung mit der Anzahl der verwendeten Knoten oder auch bei Ausführung auf einer anderen Maschine stark verändert. Daraus folgt die vierte Beobachtung: **Die beste Sperre für eine Anwendung ändert sich häufig mit der Anzahl der beteiligten Knoten und der verwendeten Maschine.**

4.2.5 Negativer Einfluss von Sperren

Von der vorherigen Beobachtung ausgehend, liegt die Vermutung nahe, dass diese starken Schwankungen in der Leistung auch sehr ungünstige Anwendungs-Sperren-Kombinationen mit sich bringen. Die Messungen zeigen, dass jede Sperre für jede Knotenanzahl mehrere Anwendungen in ihrer Leistungsfähigkeit im Vergleich zu anderen Sperren merklich beeinträchtigt. So ist die bereits erwähnte AHMCS-Sperre bei optimaler Knotenanzahl je nach Maschine für 24 bis 39% der Anwendungen ungünstig. Bei maximaler Knotenanzahl steigt dieser Anteil auf 62%. Diese Tendenz ist für alle betrachteten Sperren zu erkennen[9]. Das führt zur fünften Beobachtung: **Jede Sperre beeinflusst mehrere Anwendungen negativ.**

4.2.6 Leistungsfähigkeit von Pthread-Sperren

Die Ergebnisse der Messungen zeigen, dass sich die häufig verwendete Pthread-Sperre in den meisten Fällen im Mittelfeld der getesteten Sperren befindet. Zusätzlich erreichen manche Anwendungen ihre höchste Leistung unter deren Verwendung[9, 10]. Die von Tudor et. al.[8] gemachte Beobachtung, dass Pthread-Sperren keine gute Leistung erbringen, wird also nicht direkt bestätigt. Da die Fäden bei dieser Beobachtung jeweils auf einen Kern festgelegt waren (engl. thread-to-core pinning), ist die zweite Messreihe von Gui-

rox et. al. damit vergleichbar. Diese Konfiguration unterscheidet sich zwar von der bei Tudor et. al. verwendeten, da die Fäden stattdessen auf einen Knoten festgelegt waren (engl. thread-to-node pinning), was grundsätzlich etwas bessere Leistungen erzielt, liefert aber potenziell auch realistischere Ergebnisse, da nicht mit künstlichen Arbeitslasten getestet wurde. Daraus ergibt sich die sechste Beobachtung: **Pthread-Sperren müssen nicht aufgrund schlechter Leistung vermieden werden.**

5. BEWERTUNG UND VERGLEICH

Insgesamt sind die Erkenntnisse von Guirox et. al.[9] gemischt zu betrachten. Der große Umfang an verschiedenen Anwendungen und Sperren erscheint zunächst sehr sinnvoll, wird aber durch die geringe und insbesondere nicht sehr abwechslungsreiche Auswahl der Hardware in ihrer Aussagefähigkeit eingeschränkt. Außerdem werden zwar ausführliche Messungen durchgeführt, diese werden jedoch wenig bis nicht gedeutet. So werden die verschiedenen Arten von Sperren erläutert, diese Information aber nicht weiter genutzt, um beispielsweise zu erklären, warum Sperren wie AHMCS auf den verwendeten Maschinen einen Vorteil haben könnten. Auch wenn die Arbeit den Fokus klar auf die Softwareseite der Problematik legt, wären zusätzliche Hinweise auf die möglichen hardwareseitigen Gründe für die Ergebnisse wünschenswert gewesen. Hier liegt der Vorteil der älteren Arbeit von Tudor et. al.[8]. Diese behandelt zwar nur einige der bekanntesten Sperren, verwendet aber ein breiteres Spektrum an Hardware und beleuchtet weitere Faktoren wie Cache-Kohärenz und die Kommunikation zwischen den Knoten der Maschinen. So lässt sich hier ein besserer Überblick über den Einfluss der Hardware jenseits von veränderten Knotenzahlen gewinnen. Die Erkenntnisse der Arbeiten stimmen zwar ansonsten in einigen Punkte überein und bieten einen breiten Überblick über die Thematik, können aber aufgrund der trotzdem vorhandenen Schwächen bei der Abdeckung verschiedener Szenarien nicht als vollständiger Überblick angesehen werden.

6. AUSBLICK

Der Vormarsch von NUMA Mehrkern-Rechnern ist weiterhin nicht zu stoppen. Umfangreiche Unterstützung für Entwickler bei der Auswahl der Methoden zur softwareseitigen Synchronisation in der Form von ausführlichen Studien ist nach wie vor sehr dünn gesät. Insbesondere wie in den letzten Kapiteln ersichtlich wurden, können auch die Studien, die sich selbst als sehr umfangreich und vollständig wahrnehmen, nur eine Teilmenge der möglichen Szenarien hinsichtlich der Synchronisation im Allgemeinen und bezüglich Sperren im Besonderen abdecken. Es wurden zwar allgemeine Empfehlungen und auch Empfehlungen für spezielle Anwendungsfälle gegeben, diese sind aber durchaus nicht sehr portabel und müssen mit Vorsicht betrachtet werden. Das zeigt, wie umfangreich diese Domäne ist und wie viel Raum für neue Erkenntnisse hier noch vorhanden ist.

7. LITERATUR

[1] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[2] N. Grozev. Automated cloudstone setup in ubuntu vms. http://nikgrozev.com/2014/05/10/automated-cloudstone-setup-in-ubuntu-vms/, 2014. Abgerufen am 26.11.2016.

[3] B.-H. L. und Anant Agarwal. Reactive synchronization algorithms for multiprocessors. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VI, pages 25–35, New York, NY, USA, 1994. ACM.

[4] R. M. Y. und Anthony Romano und Christos Kozyrakis. Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, IISWC '09, pages 198–207, Washington, DC, USA, 2009. IEEE Computer Society.

[5] J.-P. L. und Florian David und Gaël Thomas und Julia Lawall und Gilles Muller. Remote core locking: Migrating critical-section execution to improve the performance of multithreaded applications. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, Berkeley, CA, USA, 2012. USENIX Association.

[6] M. C. und Michael Fagan und John Mellor-Crummey. High performance locks for multi-level numa systems. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 215–226. ACM, Februar 2007.

[7] J. M. M.-C. und Michael L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. In *Transactions on Computer Systems 9*, pages 21–65. ACM, Februar 1991.

[8] D. T. und 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*, pages 33–48. ACM, November 2013.

[9] H. G. und Renaud Lachaize und Vivien Quéma. Multicore locks: The case is not closed yet. In *Proceedings of the 2016 USENIX Annual Technical Conference*, pages 649–662. USENIX, Juni 2016.

[10] H. G. und Renaud Lachaize und Vivien Quéma. Multicore locks: The case is not closed yet (technical report – extended and updated version of the usenix atc'16 article) abgerufen von https://github.com/multicore-locks/litl/blob/master/paper/multicore_locks/tech-rep.pdf. Juni 2016.

[11] C. B. und Sanjeev Kumar und Kai Li. Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors. In *2008 IEEE International Symposium on Workload Characterization*, pages 47–56, Sept 2008.

[12] D. D. und Virendra J. Marathe und Nir Shavit. Lock cohorting: A general technique for designing numa locks. *ACM Transactions on Parallel Computing*, 1(2), Januar 2015.

[13] B. H. und William N. Scherer und Michael L. Scott. *Preemption Adaptivity in Time-Published Queue-Based Spin Locks*, pages 7–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.