LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

MNM TEAM

Entwicklung leistungsfähiger RMA-Locks durch Portierung und Optimierung von NUMA-Algorithmen

Abschlussvortrag Masterarbeit

Adrian Uffmann

adrian.uffmann@campus.lmu.de

LMU München

01.02.2022

Motivation

- Locks sind einer der wichtigsten Grundbausteine zur Synchronisierung in parallelen Programmen
- Es gibt viele Lock-Algorithmen für gemeinsamen Speicher
- Aber kaum welche für verteilten Speicher
  - `MPI_Win_lock`
  - `dash::Mutex` [1]
  - D-MCS und RMA-MCS [2]

Gemeinsamer und verteilter Speicher

- MPI bietet mit *Remote Memory Access* (RMA) einseitige Kommunikation für verteilten Speicher an, die sehr ähnlich zu gemeinsamem Speicher ist:
  - Lesender Zugriff (MPI_Get)
  - Schreibender Zugriff (MPI_Put)
  - Atomarer Zugriff (MPI_Accumulate, MPI_Get_accumulate, MPI_Compare_and_swap, ...)

Gemeinsamer und verteilter Speicher

- MPI bietet mit *Remote Memory Access* (RMA) einseitige Kommunikation für verteilten Speicher an, die sehr ähnlich zu gemeinsamem Speicher ist:
  - Lesender Zugriff (MPI_Get)
  - Schreibender Zugriff (MPI_Put)
  - Atomarer Zugriff (MPI_Accumulate, MPI_Get_accumulate, MPI_Compare_and_swap, ...)
- Zugriffe auf entfernten Speicher haben eine hohe Latenz

$\Rightarrow$ Zugriffe auf entfernten Speicher müssen vermieden werden

$\Rightarrow$ Zugriffe auf entfernten Speicher müssen vermieden werden

Das gibt es auch auf gemeinsamem Speicher ...

⇒ Zugriffe auf entfernten Speicher müssen vermieden werden

Das gibt es auch auf gemeinsamem Speicher ... bei NUMA

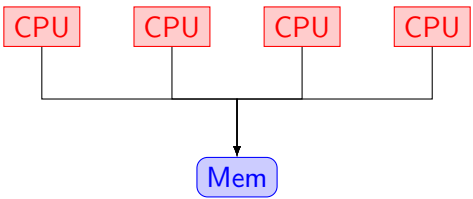

Abbildung 1: Uniform Memory Access

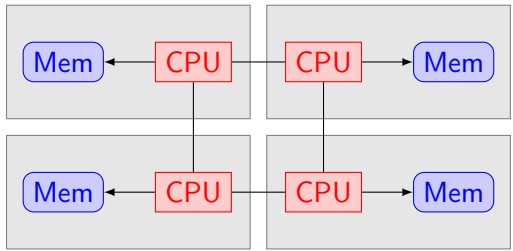

Abbildung 2: Non-Uniform Memory Access

Forschungsfrage

Können Lock-Algorithmen für NUMA-Systeme auf verteilten Speicher portiert werden, um eine bessere Performance als bestehende Implementierungen verteilter Locks zu erreichen?

Themenverwandte Arbeit

Es gibt hierzu bereits eine Arbeit von Schmid, Besta und Hoefler [2]:

- Stellt hierarchischen Lock für verteilten Speicher vor: RMA-MCS
- Basiert auf einem Lock-Algorithmus für NUMA: HMCS-Lock [3]
- Nutzt foMPI [4]

Themenverwandte Arbeit

Es gibt hierzu bereits eine Arbeit von Schmid, Besta und Hoefler [2]:

- Stellt hierarchischen Lock für verteilten Speicher vor: RMA-MCS
- Basiert auf einem Lock-Algorithmus für NUMA: HMCS-Lock [3]
- Nutzt foMPI [4]
  - foMPI funktioniert nur auf Cray Gemini (XK5, XE6) und Aries (XC30) Systemen

Themenverwandte Arbeit

Es gibt hierzu bereits eine Arbeit von Schmid, Besta und Hoefler [2]:

- Stellt hierarchischen Lock für verteilten Speicher vor: RMA-MCS
- Basiert auf einem Lock-Algorithmus für NUMA: HMCS-Lock [3]
- Nutzt foMPI [4]
  - foMPI funktioniert nur auf Cray Gemini (XK5, XE6) und Aries (XC30) Systemen
  - RMA-MCS bewegt sich außerhalb der Garantien von MPI 3.1 [5, Kapitel 11.7.3]

  $\Rightarrow$ Portierung auf Intel-MPI war schwierig

Übersicht über den weiteren Vortrag

- Benchmarks
- Optimierung der Basislocks
- Portierung von NUMA-Locks
- Fazit

Benchmarks

- Benchmarks aus themenverwandter Arbeit [2] fokussieren *Reader-Writer*-Locks
- Sonst keine Benchmarks verfügbar

Benchmarks

- Benchmarks aus themenverwandter Arbeit [2] fokussieren *Reader*-*Writer*-Locks
- Sonst keine Benchmarks verfügbar

$\Rightarrow$ Neue Benchmarksuite erforderlich

Kennzahlen

- Geschwindigkeit
- Konkurrenz

- Fairness

Kennzahlen

- Geschwindigkeit
- Konkurrenz
  - Definition: Der Anteil der Akquisitionen, bei denen ein Prozess den Lock nicht direkt akquirieren konnte, sondern auf einen Vorgänger warten musste.
- Fairness

Kennzahlen

- Geschwindigkeit
- Konkurrenz
  - Definition: Der Anteil der Akquisitionen, bei denen ein Prozess den Lock nicht direkt akquirieren konnte, sondern auf einen Vorgänger warten musste.
- Fairness
  - Nicht die theoretische Fairness des Algorithmus
  - Sondern die praktische Fairness der Implementierung (wie in [6])

  $\Rightarrow$ Variationskoeffizient (CV) des Prozessfortschritts

Benchmarksuite

- *Uncontested Performance Benchmark* (UPB)

  ⇒ 0 % Konkurrenz

- *Empty Critical Section Benchmark* (ECSB)

  ⇒ 100 % Konkurrenz

- *Wait Before Acquire Benchmark* (WBAB)

  ⇒ Variable Konkurrenz von 100 % bis 0 %

- *Changing Critical Work Benchmark* (CCWB)

  ⇒ Möglichst realistisch mit variabler Konkurrenz von 0 % bis 100 %

Optimierung der Basislocks

`dash::Mutex`

- Bestandteil der PGAS-Bibliothek DASH [1]
- Basiert auf dem MCS-Lock [7]
- Nutzt Punkt-zu-Punkt-Kommunikation für Lockübergaben (`MPI_Send` und `MPI_Recv`)

Optimierung von `dash::Mutex`

- Iterationsdauer ⇒ Geringer ist besser
- Viel kritische Arbeit = Hohe Konkurrenz
- `dash::Mutex` ließ sich stark optimieren

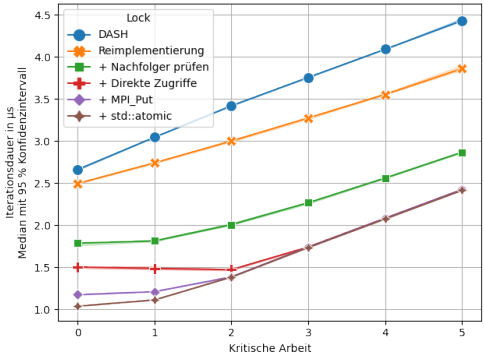

Abbildung 3: CCWB mit 28 Prozessen

Optimierte Basislocks

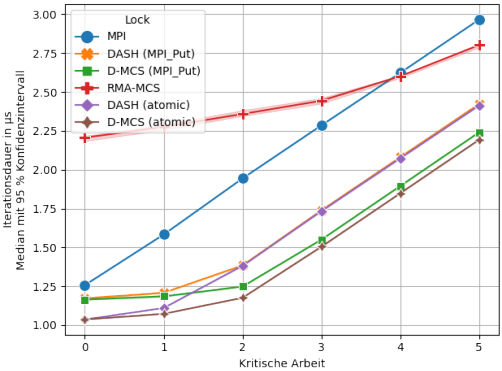

Abbildung 4: CCWB mit 28 Prozessen

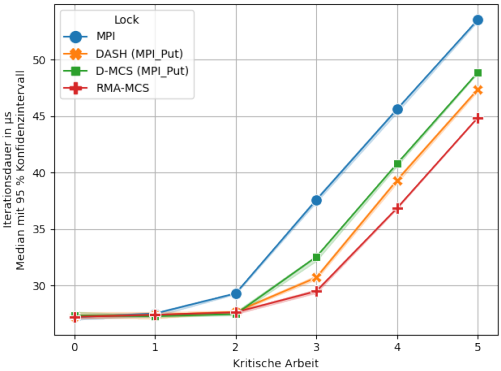

Abbildung 5: CCWB mit 112 Prozessen

Betrachtete NUMA-Locks

Algorithmus	Jahr	Geeignet
RH-Lock [8]	2002	✔
HCLH-Lock [9]	2006	✘
Cohort-Lock [6]	2012	✔
HMCS-Lock [3]	2015	✘
AHMCS-Lock [10]	2016	✘
CST-Lock [11]	2017	✘
SHFL-Lock [12]	2019	✔

Cohort-Lock

- Ein globaler Lock
- Ein lokaler Lock pro NUMA-Knoten
- Lokale Nachfolger ...
  - ... werden bevorzugt
  - ... erben globalen Lock automatisch
- Verhinderung von *Starvation* durch Zähler für lokale Lockübergaben

Cohort-Lock: Zähler-Optimierungen

- C-MCS-MCS = Cohort-Lock mit globalem und lokalen MCS-Lock
- Durchsatz ⇒ Höher ist besser
- Wenig Wartezeit = Hohe Konkurrenz
- Direkte Portierung: langsam
- Mit Optimierung:
  - bis zu 8x so schnell wie zuvor
  - bis zu 4x so schnell wie RMA-MCS

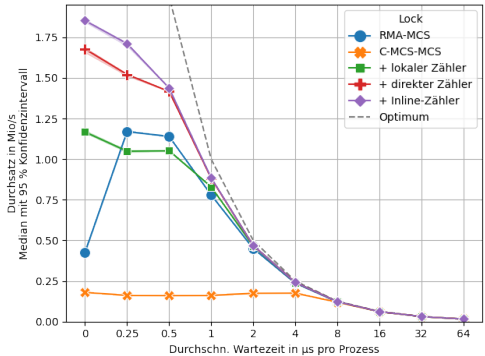

Abbildung 6: WBAB mit 112 Prozessen

Cohort-Lock: Verschiedene globale und lokale Locks

- Global:
  - TAS-Lock
  - MCS-Lock (optimierter `dash::Mutex`)
  - TKT-Lock [13]
- Lokal:
  - MCS-Lock (optimierter D-MCS)
  - TKT-Lock [13]
  - TTS-Lock

Cohort-Lock: Evaluation verschiedener globaler und lokaler Locks

- Overhead = Iterationsdauer - Optimum

  ⇒ Geringer ist besser

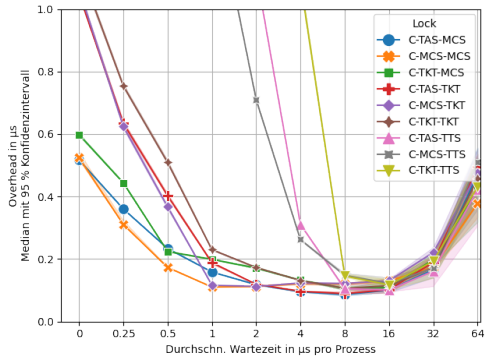

Abbildung 7: WBAB mit 112 Prozessen

Cohort-Lock: Evaluation verschiedener globaler und lokaler Locks

- Overhead = Iterationsdauer - Optimum

  ⇒ Geringer ist besser

- Hohe Konkurrenz (0 µs bis 0,5 µs):
  - Lokaler Lock entscheidend
  - MCS am besten

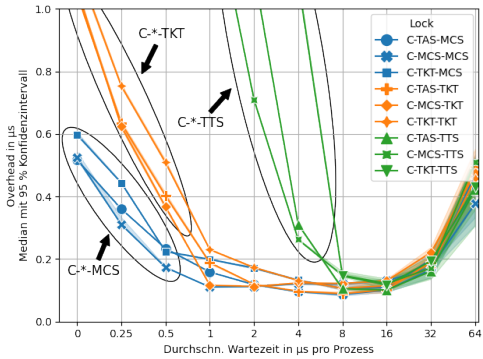

Abbildung 7: WBAB mit 112 Prozessen

Cohort-Lock: Evaluation verschiedener globaler und lokaler Locks

- Overhead = Iterationsdauer - Optimum
  - ⇒ Geringer ist besser
- Hohe Konkurrenz (0 µs bis 0,5 µs):
  - Lokaler Lock entscheidend
  - MCS am besten
- Mittlere Konkurrenz (1 µs bis 4 µs):
  - Globaler Lock entscheidend

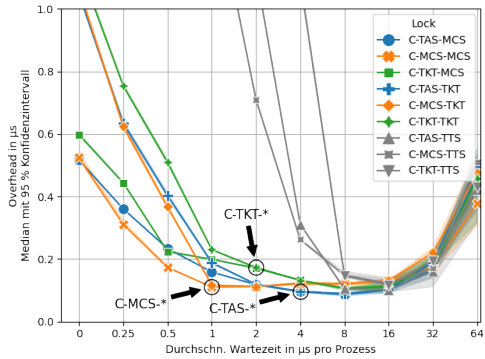

Abbildung 7: WBAB mit 112 Prozessen

Cohort-Lock: Lokale Warteschlangen-Locks

- Global:
  - MCS-Lock (optimierter `dash::Mutex`)
- Lokal:
  - MCS-Lock (optimierter D-MCS)
  - CLH-Lock [14] [15]
  - Hemlock [16]

Cohort-Lock: Evaluation lokaler Warteschlangen-Locks

- Optimierungen aus Hemlock Arbeit [16]:
  - Overlap
  - *Coherence Traffic Reduction* (CTR)
  - *Aggressive Hand-over* (AH)
- Optimierung aus HMCS Arbeit [3]:
  - Inline-Zähler

$\Rightarrow$ Lokaler Hemlock am schnellsten

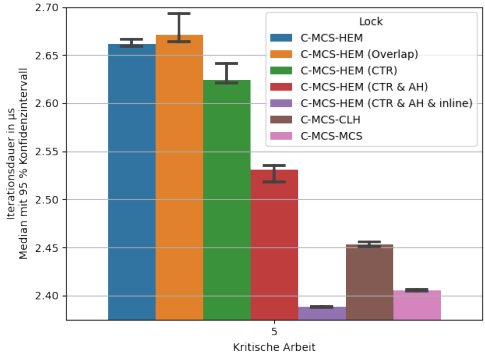

Abbildung 8: CCWB mit 28 Prozessen

Ergebnisse

- C-MCS-HEM bis zu vier Mal so schnell wie RMA-MCS
  - Bei geringerem Speicherverbrauch
- `dash::Mutex` konnte stark optimiert werden
  - Sollte trotzdem besser einen Algorithmus für NUMA nutzen
- Neue Benchmarksuite

  $\Rightarrow$ Weitere Locks können leicht evaluiert werden

Zukünftige Forschung

- Untersuchung weiterer Algorithmen
  - FC-MCS [17]
  - CNA [18]
  - Fissile-Locks [19]
  - CLoF [20]
- Untersuchung von *Reader-Writer*-Locks
  - Kann für Implementierung von `MPI_Win_lock` genutzt werden

  $\Rightarrow$ Viele Anwendungen könnten profitieren

References I

[1] H. Zhou, Y. Mhedheb, K. Idrees, C. W. Glass, J. Gracia und K. Fürlinger, "DART-MPI: An MPI-Based Implementation of a PGAS Runtime System," in *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, Ser. PGAS '14, Eugene, OR, USA: Association for Computing Machinery, 2014, ISBN: 9781450332477. DOI: 10.1145/2676870.2676875. Adresse: https://doi.org/10.1145/2676870.2676875.

References II

[2] P. Schmid, M. Besta und T. Hoefler, "High-Performance Distributed RMA Locks," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, Ser. HPDC '16, Kyoto, Japan: Association for Computing Machinery, 2016, S. 19–30, ISBN: 9781450343145. DOI: 10.1145/2907294.2907323. Adresse: https://doi.org/10.1145/2907294.2907323.

References III

[3] M. Chabbi, M. Fagan und J. 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*, Ser. PPoPP 2015, San Francisco, CA, USA: Association for Computing Machinery, 2015, S. 215–226, ISBN: 9781450332057. DOI: 10.1145/2688500.2688503. Adresse: https://doi.org/10.1145/2688500.2688503.

[4] R. Gerstenberger, M. Besta und T. Hoefler, "Enabling Highly Scalable Remote Memory Access Programming with MPI-3 One Sided," *Commun. ACM*, Jg. 61, Nr. 10, S. 106–113, Sep. 2018, ISSN: 0001-0782. DOI: 10.1145/3264413. Adresse: https://doi.org/10.1145/3264413.

References IV

[5] *MPI: A Message-Passing Interface Standard*, Message Passing Interface Forum, Juni 2015. Adresse: https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[6] D. Dice, V. J. Marathe und N. Shavit, "Lock Cohorting: A General Technique for Designing NUMA Locks," *SIGPLAN Not.*, Jg. 47, Nr. 8, S. 247–256, Feb. 2012, ISSN: 0362-1340. DOI: 10.1145/2370036.2145848. Adresse: https://doi.org/10.1145/2370036.2145848.

References V

[7] J. M. Mellor-Crummey und M. L. Scott, "Synchronization without Contention," in *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, Ser. ASPLOS IV, Santa Clara, California, USA: Association for Computing Machinery, 1991, S. 269–278, ISBN: 0897913809. DOI: 10.1145/106972.106999. Adresse: https://doi.org/10.1145/106972.106999.

References VI

[8] Z. Radović und E. Hagersten, "Efficient Synchronization for Nonuniform Communication Architectures," in *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, Ser. SC '02, Baltimore, Maryland: IEEE Computer Society Press, Nov. 2002, S. 1–13, ISBN: 076951524X. DOI: 10.1109/SC.2002.10038. Adresse: https://doi.org/10.1109/SC.2002.10038.

[9] V. Luchangco, D. Nussbaum und N. Shavit, "A Hierarchical CLH Queue Lock," in *Proceedings of the 12th International Conference on Parallel Processing*, Ser. Euro-Par'06, Dresden, Germany: Springer-Verlag, 2006, S. 801–810, ISBN: 3540377832. DOI: 10.1007/11823285_84. Adresse: https://doi.org/10.1007/11823285_84.

References VII

[10] M. Chabbi und J. Mellor-Crummey, "Contention-Conscious, Locality-Preserving Locks," *SIGPLAN Not.*, Jg. 51, Nr. 8, Feb. 2016, ISSN: 0362-1340. DOI: 10.1145/3016078.2851166. Adresse: https://doi.org/10.1145/3016078.2851166.

[11] S. Kashyap, C. Min und T. Kim, "Scalable NUMA-Aware Blocking Synchronization Primitives," in *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, Ser. USENIX ATC '17, Santa Clara, CA, USA: USENIX Association, 2017, S. 603–615, ISBN: 9781931971386.

References VIII

[12] S. Kashyap, I. Calciu, X. Cheng, C. Min und T. Kim, "Scalable and Practical Locking with Shuffling," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, Ser. SOSP '19, Huntsville, Ontario, Canada: Association for Computing Machinery, 2019, S. 586–599, ISBN: 9781450368735. DOI: 10.1145/3341301.3359629. Adresse: https://doi.org/10.1145/3341301.3359629.

[13] J. M. Mellor-Crummey und M. L. Scott, "Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors," *ACM Trans. Comput. Syst.*, Jg. 9, Nr. 1, S. 21–65, Feb. 1991, ISSN: 0734-2071. DOI: 10.1145/103727.103729. Adresse: https://doi.org/10.1145/103727.103729.

References IX

[14] T. Craig, "Building FIFO and priority queuing spin locks from atomic swap," University of Washington, Department of Computer Science und Engineering, FR-35, Seattle, WA 98195, Techn. Ber. 93-02-02, Feb. 1993.

[15] P. S. Magnusson, A. Landin und E. Hagersten, "Queue Locks on Cache Coherent Multiprocessors," in *Proceedings of the 8th International Symposium on Parallel Processing*, USA: IEEE Computer Society, 1994, S. 165–171, ISBN: 0818656026. DOI: 10.1109/IPPS.1994.288305. Adresse: https://doi.org/10.1109/IPPS.1994.288305.

References X

[16] D. Dice und A. Kogan, "Hemlock : Compact and Scalable Mutual Exclusion," *CoRR*, Jg. abs/2102.03863v4, Jan. 2022. Adresse: https://arxiv.org/abs/2102.03863v4.

[17] D. Dice, V. J. Marathe und N. Shavit, "Flat-Combining NUMA Locks," in *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, Ser. SPAA '11, San Jose, California, USA: Association for Computing Machinery, 2011, S. 65–74, ISBN: 9781450307437. DOI: 10.1145/1989493.1989502. Adresse: https://doi.org/10.1145/1989493.1989502.

References XI

[18] D. Dice und A. Kogan, "Compact NUMA-Aware Locks," in *Proceedings of the Fourteenth EuroSys Conference 2019*, Ser. EuroSys '19, Dresden, Germany: Association for Computing Machinery, 2019, ISBN: 9781450362818. DOI: 10.1145/3302424.3303984. Adresse: https://doi.org/10.1145/3302424.3303984.

[19] ——,"Fissile Locks," in *Networked Systems*, C. Georgiou und R. Majumdar, Hrsg., Cham: Springer International Publishing, 2021, S. 192–208, ISBN: 978-3-030-67087-0. DOI: 10.1007/978-3-030-67087-0_13. Adresse: https://doi.org/10.1007/978-3-030-67087-0_13.

References XII

[20] R. L. de Lima Chehab, A. Paolillo, D. Behrens, M. Fu, H. Härtig und H. Chen, "CLoF: A Compositional Lock Framework for Multi-Level NUMA Systems," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, Ser. SOSP '21, Virtual Event, Germany: Association for Computing Machinery, 2021, S. 851–865, ISBN: 9781450387095. DOI: 10.1145/3477132.3483557. Adresse: https://doi.org/10.1145/3477132.3483557.

Optimierung von `dash::Mutex`

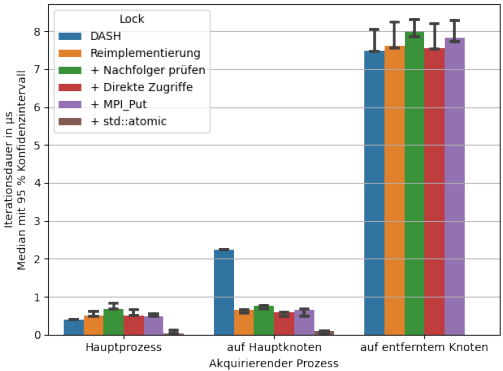

Abbildung 9: UPB

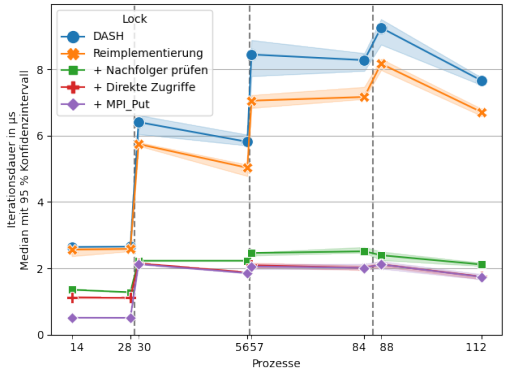

Abbildung 10: ECSB

Optimierung von `dash::Mutex`

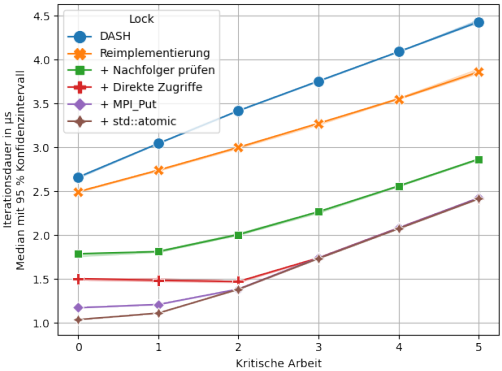

Abbildung 11: CCWB mit 28 Prozessen

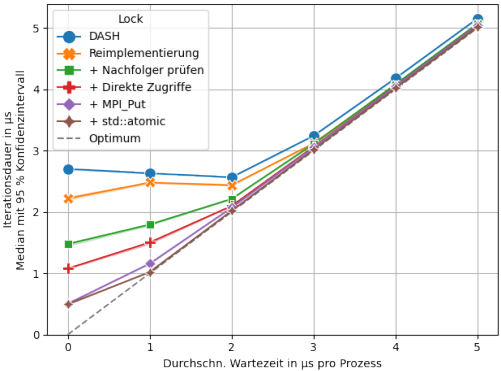

Abbildung 12: WBAB mit 28 Prozessen

Optimierung von `dash::Mutex`

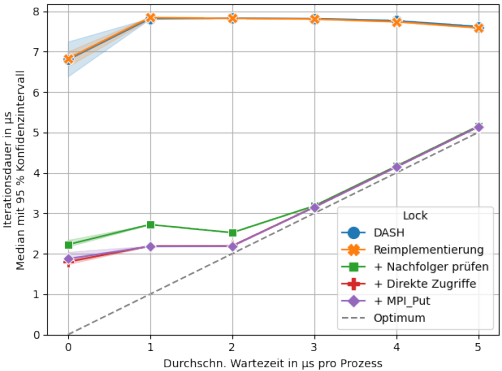

Abbildung 13: WBAB mit 112 Prozessen

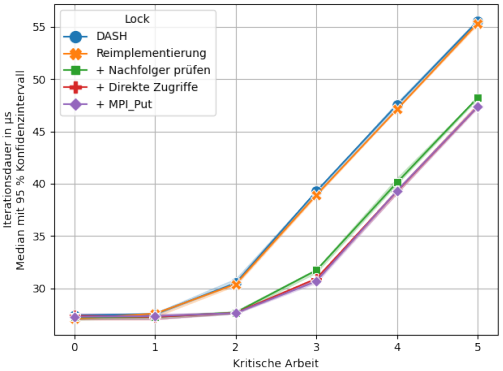

Abbildung 14: CCWB mit 112 Prozessen

Cohort-Lock: Zähler-Optimierungen

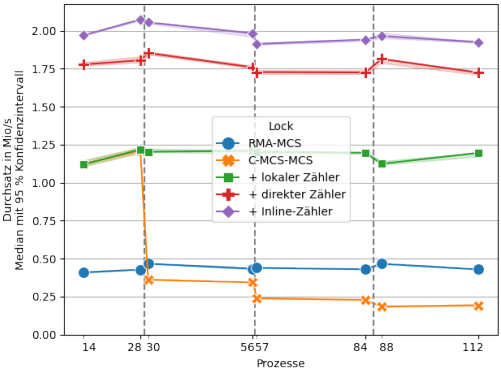

Abbildung 15: ECSB

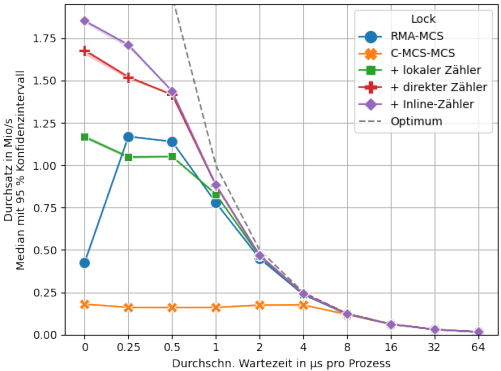

Abbildung 16: WBAB mit 112 Prozessen

Cohort-Lock: Fairness verschiedener globaler und lokaler Locks

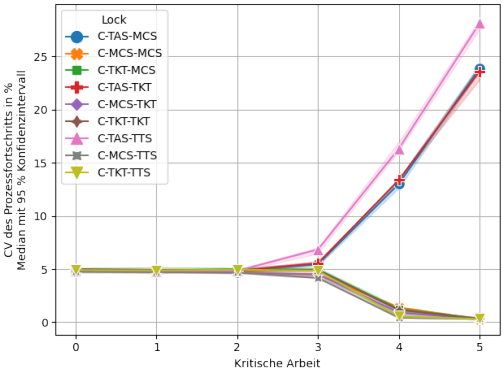

Abbildung 17: CCWB Fairness

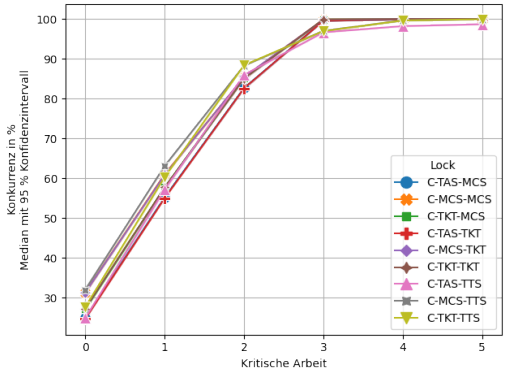

Abbildung 18: CCWB Konkurrenz

RH-Lock

- Basiert auf *Test-and-Set* (TAS)-Locks für lokale Lockübergaben
- und *Test-and-Test-and-Set* (TTS)-Locks für entfernte Lockübergaben
- Beides sind sehr unfaire Locks

RH-Lock

- Basiert auf *Test-and-Set* (TAS)-Locks für lokale Lockübergaben
- und *Test-and-Test-and-Set* (TTS)-Locks für entfernte Lockübergaben
- Beides sind sehr unfaire Locks

$\Rightarrow$ Auch der RH-Lock ist sehr unfair

RH-Lock Evaluation

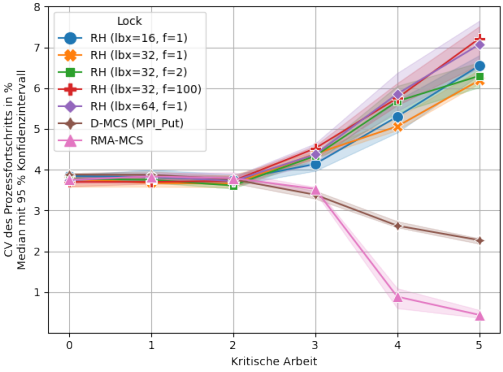

Abbildung 19: CCWB Fairness

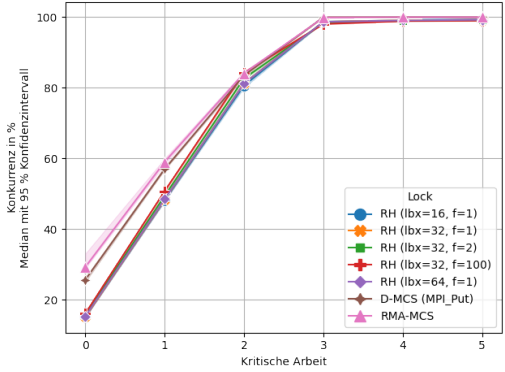

Abbildung 20: CCWB Konkurrenz

SHFL-Lock

- Nutzt einen TTS-Lock für den schnellen Pfad
- und einen MCS-Lock [7] für den langsamen Pfad
- Prozesse in der MCS-Warteschlange werden nach NUMA-Knoten gruppiert

SHFL-Lock Evaluation

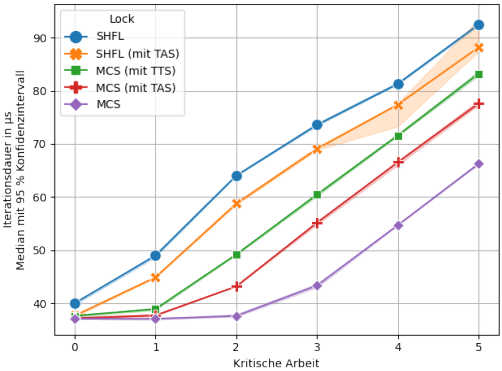

Abbildung 21: CCWB mit 112 Prozessen

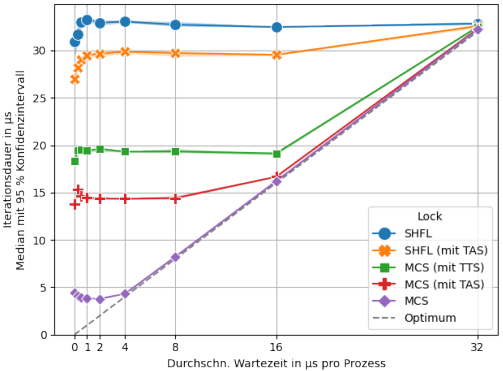

Abbildung 22: WBAB mit 112 Prozessen