B. Juurlink, W.Karl (Hrsg.)
Proceedings 28th PARS Workshop

# Comparing MPI Passive Target Synchronization Schemes on a Non-Cache-Coherent Shared-Memory Processor

Steffen C hristgau[1], Bettina Schnor[2]

**Abstract:** MPI passive target synchronisation offers exclusive and shared locks. These are the building blocks for the implementation of applications with Readers & Writers semantic, like for example distributed hash tables. This paper discusses the implementation of MPI passive target synchronisation on a non-cache-coherent multicore, the Intel Single-Chip Cloud Computer. The considered algorithms differ in their communication style, their data structures, and their semantics. It is shown that shared memory approaches scale very well and deliver good performance, even in absence of cache coherence.

**Keywords:** process synchronization; programming models and systems for manycores; MPI

## 1 Introduction

Distributed hash tables (DHTs) are a common approach for fast data access in big data and data analytics applications. However, DHTs imply dynamic communication which makes an implementation using two-sided communication, i. e. with `SEND` and `RECV` operations, cumbersome. In contrast, one-sided communication (OSC) with `PUT` and `GET` operations is a suited programming model for a DHT with its dynamic communication pattern.

Concerning the process coordination, a DHT application follows the Readers & Writers model [CS17a]: reads may occur concurrently while inserts have to be done exclusively. Hence, a resource has to be locked before it is updated. Typically, writers are given preference to avoid readers reading old data. This coordination scheme maps on MPI's *passive* target synchronization which offers *exclusive locks* (one writer) and *shared locks* (many readers). In addition, an MPI implementation has much freedom to implement the process synchronization for passive target OSC [Me15, p. 448].

This paper discusses different synchronisation algorithms on the experimental non-cache-coherent 48-core Intel Single-Chip Cloud Computer (SCC) [Ho10]. Figure 1 shows an architectural overview of the chip. While core counts steadily increase, the management of cache coherence becomes a more challenging task due to that high number of cores and high memory bandwidths [Mo15]. Although coherent high-end processors with 64 cores are currently available, non-coherent architectures provide an interesting research domain.

[1] Zuse Insitute Berlin, Supercomputing Department, christgau@zib.de
[2] University of Potsdam, Institute for Computer Science, schnor@cs.uni-potsdam.de

It has been shown in previous work that such nCC shared-memory systems can be easily programmed with well established technologies like, e. g., MPI [CS17b].

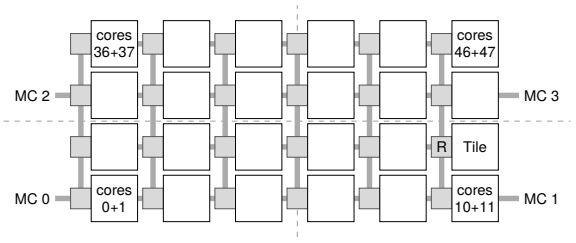



Fig. 1: Overview of the Intel SCC.

In this paper we compare the performance of three synchronization schemes for MPI passive target OSC: the message-based scheme from MPICH, the writer preference locks by Mellor-Crummey and Scott [MS91a] for shared memory systems (MCS-WP), and a best effort approach originally designed for RMA-capable distributed memory systems by Gerstenberger, Besta and Hoefler (GBH) [GBH14].

The next section gives an overview over related work. The different synchronization schemes and their data structures are described in Section 3, their implementation on the SCC is described in Section 4. Results from a micro-benchmarks are presented in Section 5, followed by a discussion. Section 6 concludes the paper.

## 2 Related Work

An early work on the topic of efficient MPI OSC implementations is the discussion for InfiniBand clusters [Ji04]. Recently, implementation schemes for NUMA-aware locks on cache-coherent multicore machines are gaining interest [DMS15, GLQ16, KMK17, CFMC15], but non-cache-coherent architectures are still neglected.

Concerning the SCC, the authors of [AMB12] investigate barrier synchronization on the SCC and use the Message Passing Buffer (MPB) to store the synchronization data. In [ASB14], they even exploit unused entries in the rare lookup tables of the chip's memory subsystem. The bottom line of this research is that synchronization data should be placed close to the spinning core. RCKMPI [UGT12] is a tuned message-based MPI implementation for the SCC and uses the fast on-chip MPB for message transport. One-sided communication is fully supported but is based on messages as well. In case of MPI's *general active synchronization*, we have already shown that an implementation using shared memory and uncached memory accesses outperforms the message-based approach significantly [CS17b]. Similar, Reble et al. discuss the active target *fence synchronization* style which they implement on top of an efficient barrier [RCL13].

Regarding Distributed Memory Architectures, Gerstenberger et al. have published performance numbers of a distributed hash table application running on up to 32k cores [GBH14].

They use their own MPI-3.0 RMA library implementation for Cray Gemini and Aries interconnects called foMPI (fast one-sided MPI). The presented synchronization scheme for passive synchronization is described in Section 3.1 and adapted for the SCC (see Section 4). Schmid et al. have proposed a scheme for Readers & Writers locking dedicated for distributed memory architectures with RMA capabilities like the Cray XC30 [SBH16]. The synchronization data structures are organized hierarchically in a distributed tree.

Subsuming the related work, there are no efforts in *passive target synchronization* for nCC many-core CPUs with shared memory like the SCC.

# 3 Synchronization Schemes for nCC Architectures

This section describes three implementation designs for MPI passive target synchronization. The first two are known from the literature. The third one describes the default implementation on the SCC. While [GBH14] presents a best-effort approach for a distributed-memory machine, the work from [MS91a] addresses scaling on shared-memory machines.

## 3.1 GBH Best Effort Synchronization

In [GBH14], Gerstenberger, Besta and Hoefler (GBH) present an implementation for MPI passive target synchronization for the Cray XC super-computers. It is based on atomic remote direct memory operations (RDMA) operations which are supported by the hardware. The design uses two stages of counters for each created window object: a single global counter and per-process local counters. All counters are allocated in memory close to the owning process. The global counter resides in the memory of a designated process (rank 0). All counters are accessible by RDMA operations.

The global counter tracks active `LOCKALL` operations and exclusive locks which are mutual exclusive. The per-process counter indicates the number of active exclusive and shared locks. As there can be only one exclusive access at a given time and process, a single bit is used to indicate such epochs (see Fig. 2).

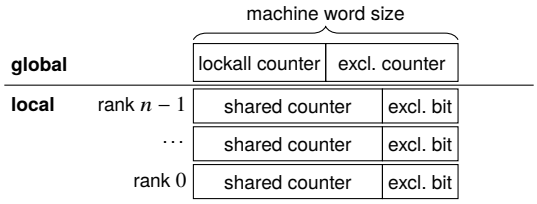



Fig. 2: Counters used by the GBH synchronization scheme.

Whenever a lock of either type is going to be acquired, the respective fields in the counters are incremented using atomic operations which return the previous value (fetch and add).

When a conflict is detected, e. g. shared locks are active at a process but an exclusive one should be acquired, the counter modifications are reverted and the process tries again at a later time using an exponentially growing back-off. The scheme does not distinguish between different process types, so any reader may overtake writers.

### 3.2 MCS Locks with Writer Preference

To avoid centralized spin objects which cause high interconnect traffic, Mellor-Crummey and Scott proposed MCS locks [MS91a]. Those are based on linked lists of lock objects that are allocated in shared memory. Each process that wants to enter a critical section by means of MCS locks appends a list entry which consists of a boolean flag `blocked` and a pointer to the next waiting process. The flag is initially set to `TRUE`. A process that wants to acquire the lock repeatedly polls the flag in its list entry until it is set to `FALSE` by a process which releases its lock.

The main advantage of using one list item per process is that spinning is done only on a local list item and not on a globally shared one like a single spin lock, for example.

Based on the original MCS locks, which do not differentiate between process types, Mellor-Crummey et al. present specialized locks that give precedence to either reading or writing processes [MS91b]. We have implemented MCS locks with writer preference (MCS-WP), since it fits best to the DHT use case where lots of readers and rare writers are expected.

Independent of the precedence, the proposed lock data structures contain lists for waiting reader and writer processes as shown in Figure 3. In addition to the lists, there is a `state` variable which is a single integer variable. For writer-precedence, the state tracks the number of active readers and provides flags for indicating presence of interested readers, interested writers, and active writers. Those are manipulated with atomic operations [MS91b, Sec. 3]. For usage with MPI passive target synchronization, every window $i$ is associated with a lock data structure $L_i$ as shown in Figure 3.

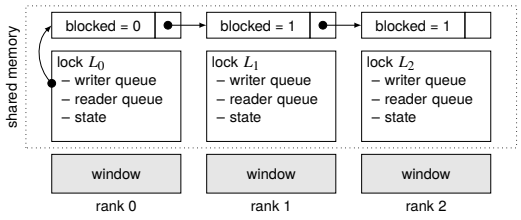



Fig. 3: MCS-lock based data structure for reader or writer precedence.

### 3.3 Message-Based Synchronization

RCKMPI, the MPI implementation for the SCC, uses messages to implement passive target synchronization. This behaviour is inherited from MPICH's CH3 device implementation but varies depending on the configuration. By default, the `LOCK` synchronization and subsequent communication operations are deferred until the end of the `UNLOCK` operation. The library then sends a control message from the origin to the target process, waits for a reply, i. e. *lock granted message*, issues the communication operations, and signals the unlock operation by setting an according field in the message header of the final communication operation. The unlock indicator can be piggy-backed in case of a single RMA operation. If no RMA operation is performed, no messages are sent. However, MPICH/RCKMPI can be configured to send a control message to the target for lock acquisition at the beginning of the access epoch. This also ensures transfer of control messages even in the absence of communication. Although this implements one-sided communication it actually requires participation of the target to process the synchronization messages.

Independent of the active configuration, the lock requests from different origins are serialized at the target process. Since the received messages are processed in the order at which they are received by the target, there is no preference of readers or writers (or lock type).

## 4 Implementation on the SCC

The SCC is not a product but a research vehicle [Ho10]. Each of the 48 cores has two integrated 16 KB L1 caches – one for data and instructions – as well as an external unified 256 KB L2 cache. There is no cache coherence between the caches of different cores, but every core can access all memory location. In addition to main memory, a fast 16 kB Message Passing Buffer (MPB) is placed on each tile.

All of the of the above synchronization schemes have been implemented in RCKMPI. Messages exchange is done by writing messages in the receiver's MPB who polls the buffer for new incoming messages. This implementation is considered as the baseline version.

The GBH and MCS-WP implementations do not use messages. Instead, the required data structures are allocated in shared off-chip DRAM memory. Due to the non-coherent architecture of the SCC, those data structures are polled using uncached memory accesses. While previous research proved that polling the on-tile MPB or even the Lookup Tables (see Section 2) reduces the traffic on the interconnect, both approaches are hardly feasible in our case due to a resource conflict (MPB) or a missing resource management (LUTs). Therefore, the external DRAM is used as a resource for the synchronization data. As shown in previous work [CS17a], synchronization data is allocated in a distributed fashion: Per-process data is stored in the DRAM memory close to the owning core.

# 5 Experimental Evaluation

We evaluate the different design schemes using a communication-free microbenchmark. The experiments were conducted on a SCC system with cores clocked at 533 MHz and 800 MHz for the mesh network and the memory controllers. A total of 32 GB of RAM was installed on the system. Each core runs Linux 3.1.4 with platform-relevant patches applied. Software is cross-compiled using GCC 4.4.6 with optimization (`-O2`), and MPICH 3.1.3 was used as the foundation MPI implementation.

## 5.1 Microbenchmark description

The employed microbenchmark measures the latency for a pair of LOCK/UNLOCK operations. No communication is performed between those two operations. The time for performing these operations is compared for the GBH and MCS-WP implementations as well as for the message-based but SCC-optimized RCKMPI. Because the default RCKMPI implementation defers the synchronization, we also measure RCKMPI with a forced message exchange for synchronization (cf. Section 3.3).

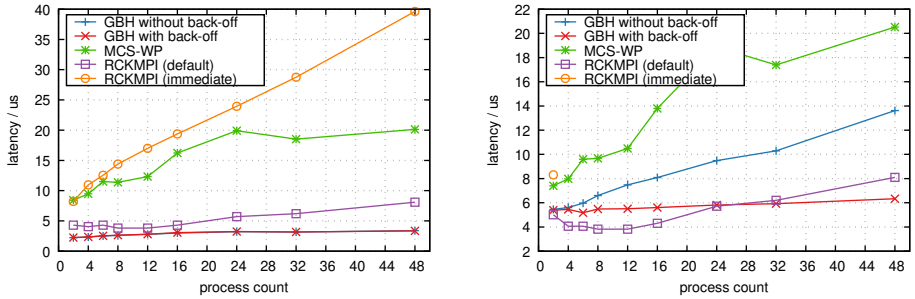
Each process of the microbenchmark performs 1000 pairs of `LOCK`/`UNLOCK` calls in a tight loop. The type of the employed lock is controlled by an input parameter that specifies the share of shared and exclusive locks each process shall issue. According to that parameter, every process randomly decides between the two lock types. The target process is chosen randomly as well and may include the origin process. The access mode (shared or exclusive) will obviously have an influence on the results. Therefore, three different ratios of shared and exclusive locks, i. e.. readers and writers, were measured: only shared locks (only readers), all accesses are made with exclusive locks (only writers) and a mixture of both where shared and exclusive accesses are equally distributed (see Figure 4a–5).

Since we are interested in the scaling of the different synchronization schemes, we run the benchmark with different numbers of processes. The processes are mapped according to the core with matching number. That is, the rows of the SCC's mesh network are filled before moving to the next row. In case for 24 processes, the chip's lower half (see Figure 1) is filled.

From each of the 1000 `LOCK`/`UNLOCK` cycles, the required time is measured. Finally, all samples from all processes are gathered and the median time from all synchronization operations is computed. This value is shown in the following diagrams for different core counts. We compare MCS-WP, RCKMPI with both immediate messaging (synchronization message upon method call) and default behaviour (no messages), and GBH. In addition to GBH, a version without back-off is included in the evaluation in order to analyze the impact of the back-off on the synchronization latency. For the version with back-off, the initial delay between two lock acquisition attempts is 1 µs. This value is doubled for each consecutive failed attempt. It has been shown for the SCC that the usage of back-offs can improve the performance of synchronization primitives [RCL13].

### 5.2 Results: Shared Locks Only

Figure 4a shows the latency of the different implementations when all accesses are shared. The RCKMPI implementation with immediate messaging has the highest latency due to overhead from sending and processing the control messages. The default RCKMPI implementation includes only library overhead but no message exchange and scales therefore well. It is slightly slower than both GBH versions due to additional management in the message-based code path that are not used for the GBH and MCS implementations.




(a) Shared locks only.

(b) 50% shared and 50% exclusive locks.

Fig. 4: Synchronization latencies for the shared-only and mixed cases.

Both GBH versions exhibit nearly constant and identical synchronization latencies because no conflicts occur in the shared-only use case and thus no back-off is required. Consequently, the two curves overlap in the plot. Similar, the reason for the constant time is that shared accesses are not mutual exclusive. In the GBH scheme, acquiring a shared lock only involves incrementing the shared counter in the target's local counter (see Fig. 2). Due to the distribution of the synchronization data and missing exclusive locks, which might cause more attempts to acquire the shared lock, no contention on these counters is observed on the SCC.

In case of the of the MCS-WP, the latency is generally higher than for GBH. The latter only involves incrementing a single per-process counter value, but for MCS the state variable needs to be checked and list data has to be changed. This causes the operations to take longer than for GBH.

From the data one can also note an increasing latency for up to 24 processes. After that, the latency remains nearly constant with a slight drop for 32 processes. This observation can be attributed to the distributed synchronization data. With up to 24 processes, the two lower memory controllers of the chip (see Figure 1) have to handle the polling requests of the 12 processes associated to each of them. Additional processes are then handled by the next memory controllers, but do not increase the load on the already utilized ones.

This is also the reason for the slight latency drop at 32 processes: Since the upper two memory controllers have to serve fewer processes than the lower two, the median latency

reduces. Similar behavior can be identified for the switch from 6 (only handled by MC 0) to 8 processes (MC 1 handles additional polling accesses).

Since for 24 processes the two lower memory controllers experience maximum usage and because of the distributed data, no further increase of the lock latency is observed when the number of MPI processes is raised. This is different from the statement in [SBH16, p. 11], that MCS locks that distinguish between readers and writers do not scale well under heavy read contention. We are not able to confirm this remark by our experiments on the SCC.

### 5.3 Results: Lock Type Mix

In Figure 4b, the results for the 50% mix of shared and exclusive locks is displayed. For this workload, no data — except for two processes — could be acquired for the immediate RCKMPI variant. The benchmark deadlocked in those cases. Our assumption is that required responses to control message are not sent when they are expected. This might be due to absent message processing and might be solved by triggering process through `MPI_Test` calls. However, a deeper investigation was out of the paper's scope.

The default variant of RCKMPI which does not send any message unsurprisingly performs as in case for shared lock.

For GBH, the latency is slightly increased compared to the previous results. The scaling, however, remains nearly identical and still shows a constant time for the synchronization for all process counts. The increased latency can be accounted to the higher probability for an unsuccessful attempt for lock acquisition. In such a case, the processes perform their back-off but are able to acquire the lock in a later attempt very soon, since the median latency only increases by about 3 µs.

Opposite to GBH with back-off, the version without this feature shows a latency that increases linear with the number of processes. The effect is due to the contention. This can be explained by a competition for both the global and the per-process counter variables. This reduces the chance of a lock acquisition for either process type. Especially, the global counter must be modified both at the beginning and at the end of the lock attempt — notably, this has to be done also in the unsuccessful case. Since the global counter is a centralized data structure, contention on the responsible memory controller is likely.

With the exception of the GBH without back-off and the dysfunctional immediate RCKMPI version, the overall performance and scaling is identical to the previous scenario.

For MCS-WP, an almost identical performance as in the previous experiment is observed. While two different process types are active, the same data structures are used and the same operations (state manipulation and list management) are performed. Thus, the overall performance stays the same.

### 5.4 Results: Exclusive Locks Only

Finally, Figure 5 shows the scaling where only exclusive locks are used. The GBH variant without back-off clearly suffers from the sole usage of exclusive locks and its aggressive best-effort approach. The effect of contention on the global counter from the previous experiment is amplified which causes increased latency.

Contrary to that observation, the other synchronization schemes still perform with identical scaling behavior and similar absolute latency. For GBH with back-off, the latency increases slightly and approaches MCS-WP. This might be caused by an increased number of attempts to acquire the lock.

For MCS-WP, the performance is still equivalent to the previous experiments. Because writers just queue up at the individual per-process queues (cf. Figure 3), the median time to acquire the lock does not increase. Further, the completely distributed data structures pays off as no contention occurs.

The immediate RCKMPI variant works without problems in this experiments. However, the latency is up to about four times higher than for the other implementations. Moreover, linear scaling can be observed.

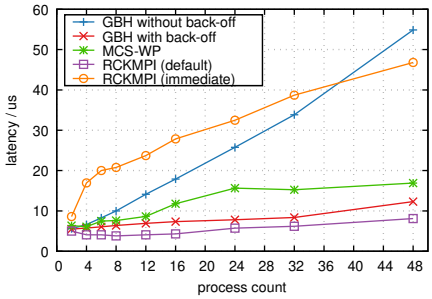



Fig. 5: Latency for exclusive locks.

### 5.5 Discussion

Although a tuned implementation for message transfer is available on the SCC, it does not pay off in case for MPI passive target synchronization. Besides issues with deadlocks, which may be fixable, the observed latency is much higher than for the presented memory-based approaches that use uncached-memory accesses due to the immanent data transfer and processing overhead.

Contrary, the memory-based schemes perform with low latencies and nearly constant scaling and are therefore a favorable choice for nCC shared memory architectures like the SCC. The implemented synchronization schemes take special care for the distribution of data

structures and their access pattern. The MCS-WP scheme avoids centralized data, and the GBH scheme with back-off uses a rate-limited access to its data structures.

## 6 Conclusion

In this paper, we discussed and evaluated three different synchronization schemes for non-cache-coherent shared memory architectures, like the SCC. Two memory-based schemes known from the literature have been implemented for that platform. The evaluation shows that such schemes are well-suited for nCC many-core architectures both in terms of absolute performance and scalability. Despite, they employ uncached memory operations, the approaches even outperform competitors that rely on SCC-optimized message passing.

The experiments also show that the MCS-WP scheme which gives precedence to writers can be used on nCC systems without scalability or severe performance degradations. The reason is the avoidance of centralized data structures. To achieve a comparable performance, the GBH scheme that also uses centralized data structures in addition to distributed ones, a back-off mechanism appears to be crucial for the median latency. Nevertheless, the algorithms have to be evaluated in the context of an application in subsequent work. Future work may also include an analysis how the presented approaches perform on contemporary processors built from multiple *chiplets* when taking their inherent NUMA-design and hardware support for cache coherence into consideration.

## Bibliography

[AMB12] Al-Khalissi, Hayder; Marongiu, Andrea; Berekovic, Mladen: Low-Overhead Barrier Synchronization for OpenMP-like Parallelism on the Single-Chip Cloud Computer. In: Many-core Applications Research Community (MARC) Symposium at RWTH Aachen University, November 29th-30th 2012, Aachen, Germany. 2012.

[ASB14] Al-Khalissi, Hayder; Shah, Syed Abbas Ali; Berekovic, Mladen: An Efficient Barrier Implementation for OpenMP-Like Parallelism on the Intel SCC. In: 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2014, Torino, Italy, February 12-14, 2014. 2014.

[CFMC15] Chabbi, Milind; Fagan, Michael; Mellor-Crummey, John: High Performance Locks for Multi-level NUMA Systems. In: Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. PPoPP 2015, ACM, New York, NY, USA, 2015.

[CS17a] Christgau, Steffen; Schnor, Bettina: Design of MPI Passive Target Synchronization for a Non-Cache-Coherent Many-Core Processor. In: Parallel-Algorithmen, -Rechnerstrukturen und -Systemsoftware: 27. PARS Workshop. volume 34 of Mitteilungen. Gesellschaft für Informatik, 2017.

[CS17b] Christgau, Steffen; Schnor, Bettina: Exploring one-sided communication and synchronization on a non-cache-coherent many-core architecture. Concurrency and Computation: Practice and Experience, 29(15), 2017.

[DMS15] Dice, David; Marathe, Virendra J.; Shavit, Nir: Lock Cohorting: A General Technique for Designing NUMA Locks. ACM Trans. Parallel Comput., 1(2), February 2015.

[GBH14] Gerstenberger, Robert; Besta, Maciej; Hoefler, Torsten: Enabling highly-scalable remote memory access programming with MPI-3 One Sided. Scientific Programming, 22(2), 2014.

[GLQ16] Guiroux, Hugo; Lachaize, Renaud; Quéma, Vivien: Multicore Locks: The Case is Not Closed Yet. In: Proceedings of the 2016 USENIX Conference on Usenix Annual Technic al Conference. USENIX ATC '16, USENIX Association, Berkeley, CA, USA, 2016.

[Ho10] Howard, Jason et al.: A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS. In: Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International. February 2010.

[Ji04] Jiang, Weihang et al.: Efficient Implementation of MPI-2 Passive One-Sided Communication on InfiniBand Clusters. In (Kranzlmüller, Dieter; Kacsuk, Péter; Dongarra, Jack J., eds): 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary. volume 3241 of Lecture Notes in Computer Science. Springer, 2004.

[KMK17] Kashyap, Sanidhya; Min, Changwoo; Kim, Taesoo: Scalable NUMA-aware Blocking Synchronization Primitives. In: Proceedings of the 2017 USENIX Conference on Usenix Annual Techni cal Conference. USENIX ATC '17, USENIX Association, Berkeley, CA, USA, 2017.

[Me15] Message Passing Interface Forum: , MPI: A Message-Passing Interface Standard, Version 3.1. online, June 2015. http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[Mo15] Morgan, Timothy Prickett: , More Knights Landing Xeon Phi Secrets Unveiled, March 2015. http://www.nextplatform.com/2015/03/25/more-knights-landing-xeon-phi-secrets-unveiled/ accessed 2019-08-28.

[MS91a] Mellor-Crummey, John M.; Scott, Michael L.: Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors. ACM Trans. Comput. Syst., 9(1), 1991.

[MS91b] Mellor-Crummey, John M.; Scott, Michael L.: Scalable Reader-Writer Synchronization for Shared-Memory Multiprocessors. In (Wise, David S., ed.): Proceedings of the Third ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPoPP), Williamsburg, Virginia, USA, April 21-24, 1991. ACM, 1991.

[RCL13] Reble, Pablo; Clauss, Carsten; Lankes, Stefan: One-sided communication and synchronization for non-coherent memory-coupled cores. In: International Conference on High Performance Computing & Simulation, HPCS 2013. IEEE, 2013.

[SBH16] Schmid, Patrick; Besta, Maciej; Hoefler, Torsten: High-Performance Distributed RMA Locks. In (Nakashima, Hiroshi; Taura, Kenjiro; Lange, Jack, eds): Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing, HPDC 2016, Kyoto, Japan, May 31 - June 04, 2016. ACM, 2016.

[UGT12] Ureña, Isaías A. Comprés; Gerndt, Michael; Trinitis, Carsten: Wait-Free Message Passing Protocol for Non-coherent Shared Memory Architectures. In: 19th European MPI Users' Group Meeting, EuroMPI 2012, Vienna, Austria. 2012.