Multicore Locks: The Case is not Closed Yet
(Technical report – Extended version of the USENIX ATC'16 article)
Version: May 23rd, 2016

Hugo Guiroux[†⋆] Renaud Lachaize[†⋆] Vivien Quéma[†‡⋆]
[†]*Université Grenoble Alpes* [‡]*Grenoble INP*
[⋆]*LIG (CNRS UMR 5217)*

Abstract

NUMA multicore machines are pervasive and many multithreaded applications are suffering from lock contention. To mitigate this issue, application and library developers can choose from the plethora of optimized mutex lock algorithms that have been designed over the past 25 years. Unfortunately, there is currently no broad study of the behavior of these optimized lock algorithms on realistic applications. In this paper, we attempt to fill this gap. We perform a performance study of 27 state-of-the-art mutex lock algorithms on 35 applications. Our study shows that the case is not yet closed regarding locking on multicore machines. Indeed, our conclusions include the following findings: (i) at its optimized contention level, no single lock is the best for more than 52% of the studied workloads; (ii) every lock is harmful for several applications, even if the application parallelism is properly tuned; (iii) for several applications, the best lock changes when varying the number of threads. These findings call for further research on optimized lock algorithms and dynamic adaptation of contention management.

1 Introduction

Today, multicore machines are pervasive and many multithreaded applications are suffering from bottlenecks related to critical sections and their corresponding locks. To mitigate these issues, application and library developers can choose from the plethora of optimized mutex lock algorithms that have been designed over the past 25 years but there is currently no clear study to guide this puzzling choice for realistic applications. In particular, the most recent and comprehensive empirical performance evaluation on multicore synchronization [9], due to its breadth (from hardware protocols to high-level data structures), only provides a partial coverage of locking algorithms. Indeed, the aforementioned study only considers 9 algorithms, does not consider hybrid spinning/blocking waiting policies, omits emerging approaches (e.g., load-control algorithms described in §2) and provides a modest coverage of hierarchical locks [14, 5, 6], a recent and efficient approach. Furthermore, most of the observations are based on microbenchmarks. Besides, in the case of papers that present a new lock algorithm, the empirical observations are often focused on the specific workload characteristics for which the lock was designed [21, 26], or mostly based on microbenchmarks [14, 12].

The present paper provides a broad performance study on Linux/x86 of 27 state-of-the-art mutex lock algorithms on a set of 35 realistic and diverse applications (the PARSEC, Phoenix, SPLASH2 suites, MySQL and an SSL proxy). We make a number of observations, several of which have not been previously mentioned: (*i*) about 60% of the studied applications are significantly impacted by lock performance; (*ii*) no single lock is systematically the best, even for a fixed number of contending cores; (*iii*) worse, at their optimized contention level (individually tuned for each application), the best locks never dominate for more than 52% of the lock-sensitive applications; (*iv*) any of the locks is harmful (i.e., significantly inefficient compared to the best one) for at least several workloads; (*v*) across all the lock-sensitive applications, there is no clear performance hierarchy among the locks, even at a fixed number of contending cores; (*vi*) for a given application, the best lock varies according to both the number of contending cores and the machine; (*vii*) unlike previous recommendations [9] advocating that standard Pthread mutex locks should be avoided for workloads using no more than one thread per core, we find that, with our studied workloads, the current Linux implementation of these locks actually yields good performance for many applications with this pattern. Moreover, we show that all these results hold even when each configuration, i.e., each *(application, lock)* pair, is tuned to its optimal degree of parallelism. From our performance study, we draw two main conclusions. First, specific lock algorithms should not be hardwired into the code of applications. Second, the observed trends call for further research both regarding lock algorithms and runtime support for parallel performance and contention management.

To conduct our study, manually modifying all the applications in order to retrofit the studied lock algorithms would have been a daunting task. Moreover, using a meta-library that allows plugging different lock algorithms under a common API (such as liblock [26] or libslock [9]) would not have solved the problem, as this would still have required a substantial re-engineering effort for each application. In addition, such meta-libraries provide no or limited support for important features like Pthread condition variables, used within many applications. Therefore, we implemented LiTL[1], a low-overhead library that allows transparent interposition of Pthread mutex lock operations and support for mainstream features like condition variables, without any restriction on the application-level locking discipline.

The remainder of the paper is organized as follows: §2 presents a taxonomy of existing lock designs and the list of algorithms covered by our study. §3 describes our experimental setup and the studied applications. §4 describes the LiTL library. §5 exposes the main results from our empirical observations. §6 discusses related works and §7 concludes the paper.

2 Lock algorithms

2.1 Background

The body of existing works on optimized lock algorithms for multicore architectures is rich and diverse and can be split into the following five categories:

1) Flat approaches correspond to simple algorithms (typically based on one or a few shared variables accessed by atomic instructions) such as: simple spinlock [33], backoff spinlock [2, 30], test and test-and-set (TTAS) lock [2], ticket lock [30], partitioned ticket lock [11], and standard Pthread mutex lock.

2) Queue-based approaches correspond to locks based on a waiting queue in order to improve fairness as well as the memory traffic, such as: MCS [30, 33] and CLH [7, 29, 33].

3) Hierarchical approaches are specifically aimed at providing scalable performance on large-scale NUMA machines, by attempting to reduce the rate of lock migrations among NUMA nodes. This category includes HBO [32], HCLH [28], FC-MCS [13], HMCS [5], AHMCS [6] and the algorithms that stem from the *lock cohorting* framework [14]. A cohort lock is based on a combination of two lock algorithms (similar or different): one used for the global lock and one used for the local locks (there is one local lock per NUMA node); in the usual C-L_A-L_B notation, L_A and L_B respectively correspond to the global and the node-level lock algorithms. The list includes C-BO-MCS, C-PTL-TKT and C-TKT-TKT (also known as Hticket [9]). The *BO*, *PTL* and *TKT* acronyms respectively correspond to backoff lock, partitioned ticket lock, and standard ticket lock.

4) Load-control approaches correspond to algorithms that aim at limiting the number of threads that concurrently attempt to acquire a lock, in order to prevent a performance collapse. These algorithms are derived from queue-based locks. This category includes MCS-TimePub[2] [19] and so-called *Malthusian algorithms* like Malth_Spin and Malth_STP[3] [12].

5) Delegation-based approaches correspond to algorithms in which it is (sometimes or always) necessary for a thread to delegate the execution of a critical section to another thread. The typical benefits expected from such approaches are improved cache locality and better resilience under high lock contention. This category includes Oyama [31], Hendler [20], RCL [26], CC-Synch [15] and DSM-Synch [15].

Another important design dimension is the *waiting policy* used when a thread cannot immediately obtain a requested lock [12]. There are three main approaches: (i) spinning on a memory address, (ii) immediate parking (i.e., blocking the thread) either for a fixed amount of time or until the thread gets a chance to obtain the lock, and (iii) spinning-then-parking (STP), a hybrid strategy using a fixed or adaptive threshold [22]. The choice of the waiting policy is mostly orthogonal to the lock design but, in practice, policies other than pure spinning are only considered for certain types of locks: the queue-based (from categories 2–4 above) and the standard Pthread mutex locks. Besides, note that the GNU C library for Linux provides two versions of Pthread mutex locks: the default one uses parking (via the `futex` syscall) and the second one uses an adaptive spin-then-park strategy. The latter version can be enabled with the `PTHREAD_MUTEX_ADAPTIVE_NP` option [23].

2.2 Studied algorithms

Our choice of studied locks is guided by the decision to focus on *portable* lock algorithms. We therefore exclude the following locks that require strong assumptions on

[1] LiTL: Library for Transparent Lock interposition.

[2] MCS-TimePub is mostly known as MCS-TP but we use MC-TimePub to avoid confusion with MCS_STP.

[3] Malth_Spin and Malth_STP correspond to MCSCR-S and MCSCR-STP, respectively, but we do not use the latter names to avoid confusion with other MCS locks.

Name Total #cores	**A-64** 64	**A-48** 48	**I-48** 48 (no hyperthreading)
Server model	Dell PE R815	Dell PE R815	SuperMicro SS 4048B-TR4FT
Processors	4× AMD Opteron 6272	4× AMD Opteron 6344	4× Intel Xeon E7-4830 v3
Microarchitecture	Bulldozer / Interlagos	Piledriver / Abu Dhabi	Haswell-EX
Core clock	2.1 GHz	2.6 GHz	2.1 GHz
Last-level cache (per node)	8 MB	8 MB	30 MB
Interconnect	HT3 - 6.4 GT/s per link	HT3 - 6.4 GT/s per link	QPI - 8 GT/s per link
Memory	256 GB DDR3 1600 MHz	64 GB DDR3 1600 MHz	256 GB DDR4 2133 MHz
#NUMA nodes (#cores/node)	8 (8)	8 (6)	4 (12)
Network interfaces (10 GbE)	2× 2-port Intel 82599	2× 2-port Intel 82599	2-port Intel X540-AT2

Table 1: Hardware characteristics of the testbed platforms.

the application/OS behavior, code modifications, or fragile performance tuning: HCLH, HBO, FC-MCS, and all the delegation-based locks (see Dice et al. [14] for detailed arguments).

Our study considers 27 mutex lock algorithms that are representative of both well-established and state-of-the-art approaches. We use the *_Spin* and *_STP* suffixes to differentiate variants of the same algorithm that only differ in their waiting policy. The *-LS* tag corresponds to optimized algorithms borrowed from libslock [9]. Our set includes ten flat locks (Backoff, Partitioned ticket, Phtread, Pthread adaptive, Spinlock, Spinlock-LS, Ticket, Ticket-LS, TTAS, TTAS-LS), seven queue-based locks (Alock-LS, CLH-LS, CLH_Spin, CLH_STP, MCS-LS, MCS_Spin, MCS_STP), seven hierarchical locks (C-BO-MCS_Spin, C-BO-MCS_STP, C-PTL-TKT, C-TKT-TKT, Hticket-LS, HMCS, AHMCS), and three load-control locks (Malth_Spin, Malth_STP, MCS-TimePub).

3 Experimental setup and methodology

3.1 Testbed and studied applications

Our experimental testbed consists of three Linux-based servers whose main characteristics are summarized in Table 1. All the machines run the Ubuntu 12.04 OS with a 3.17.6 Linux kernel (CFS scheduler), glibc 2.15 and gcc 4.6.3. For our comparative study of lock performance, we consider (i) the applications from the PARSEC benchmark suite (emerging workloads), (ii) the applications from the Phoenix 2 MapReduce benchmark suite, (iii) the applications from the SPLASH2 high-performance computing benchmark suite[4], (iv) the MySQL database running the Cloudstone workload, and (v) SSL proxy, an event-driven SSL endpoint that processes small messages. In order to evaluate the impact of workload changes on locking performance, we also consider so called "long-lived" variants of four of the above workloads denoted with a "_ll" suffix. Note that six of the applications cannot be evaluated on the two 48-core machines because, by design, they only accept a number of threads that correspond to a power of two: facesim, fluidanimate (from PARSEC), fft, ocean_cp, ocean_ncp, radix (from SPLASH2).

Most of these applications use a number of threads equal to the number of cores, except the three following ones: dedup (3× threads), ferret (4× threads) and MySQL (hundreds of threads). Two thirds of the applications use Pthread condition variables.

3.2 Tuning and experimental methodology

For the lock algorithms that rely on static thresholds, we use the recommended values from the original papers and implementations. The algorithms based on a spin-then-park waiting policy (e.g., Malth_STP [12]) rely on a fixed threshold for the spinning time that corresponds to the duration of a round-trip context switch [22] — in this case, we calibrate the duration using a microbenchmark on the testbed platform.

All the applications are run with memory interleaving (via the `numactl` utility) in order to avoid NUMA memory bottlenecks. Generally, in the experiments presented in this paper, we study the performance impact of a lock for a given contention level, i.e., the number of threads of the application. We vary the contention level at the granularity of a NUMA node (i.e., 8 cores for the A-64 machine, 6 cores for the A-48 machine, and 12 cores for the I-48 machine). For most of the experiments detailed in the paper, the application threads are not pinned to specific cores. The impact of pinning is nonetheless discussed in §5.3.

Finally, each experiment is run at least five times and we compute the average value. Overall, we observe little variability for most configurations. For all experiments, the considered application-level performance metric is the throughput (operations per time unit).

[4] We excluded the Cholesky application because of extremely short completion times.

4 The LiTL lock interposition library

In order to carry out the lock comparison study, we have developed LiTL, an interposition library for Linux/x86 allowing transparently replacing the lock algorithm used for Pthread mutexes. We describe its design, implementation, and assess its performance.

4.1 Design

The design of LiTL does not impose any restriction on the level of nested locking and is compatible with arbitrary locking disciplines (e.g., hand-over-hand locking [33]). The pseudo-code of the main wrapper functions of the LiTL library is depicted in Figure 1.

```
// return values and error checks
// omitted for simplification

pthread_mutex_lock(pthread_mutex_t *m) {
    optimized_mutex_t *om = get_optimized_mutex(m);
    if (om == null) {
        om = create_and_store_optimized_mutex(m);
    }
    optimized_mutex_lock(om);
    real_pthread_mutex_lock(m);
}

pthread_mutex_unlock(pthread_mutex_t *m) {
    optimized_mutex_t *om = get_optmized_mutex(m);
    optimized_mutex_unlock(om);
    real_pthread_mutex_unlock(m);
}

pthread_cond_wait(pthread_cond_t *c,
                  pthread_mutex_t *m) {
    optimized_mutex_t *om = get_optimized_mutex(m);
    optimized_mutex_unlock(om);
    real_pthread_cond_wait(c, m);
    real_pthread_mutex_unlock(m);
    optimized_mutex_lock(om);
    real_pthread_mutex_lock(m);
}

// Note that the pthread_cond_signal and
// pthread_cond_broadcast primitives
// do not need to be interposed
```

Figure 1: Overview of the pseudocode for the main wrapper functions of LiTL.

General principles The primary role of LiTL is to maintain a mapping structure between an instance of the standard Pthread lock (`pthread_mutex_t`) and an instance of the chosen optimized lock type (e.g., MCS_Spin). This implies that LiTL must keep track of the lifecycle of all the application's locks through interposition of the calls to `pthread_mutex_init()` and `pthread_mutex_destroy()`, and that each interposed call to `pthread_mutex_lock()` must trigger a lookup for the instance of the optimized lock. In addition, lock instances that are statically initialized can only be discovered and tracked upon the first invocation of `pthread_mutex_lock()` on them (i.e., a failed lookup leads to the creation of a new mapping).

The `lock`/`unlock` API of several lock algorithms requires an additional parameter (called "struct" hereafter) in addition to the lock pointer. For example, in the case of an MCS lock, this parameter corresponds to the record to be inserted in (or removed from) the lock's waiting queue. In the general case, a struct cannot be reused nor freed before the corresponding lock has been released. For instance, an application may rely on nested critical sections (i.e., a thread T must acquire a lock L_2 while holding another lock L_1). In this case, T must use a distinct struct for L_2 in order to preserve the integrity of L_1's struct. In order to gracefully support the most general cases, LiTL systematically allocates exactly one struct per lock instance and per thread.

Supporting condition variables Dealing with condition variables inside each optimized lock algorithm would be complex and tedious as most locks have not been designed with condition variables in mind. We therefore use the following strategy: our wrapper for `pthread_cond_wait()` internally calls the true `pthread_cond_wait()` function. To issue this call, we need to hold a real Pthread mutex lock (of type `pthread_mutex_t`). This strategy (depicted in the pseudocode of Figure 1) does not introduce high contention on the internal Pthread lock. Indeed, for workloads that do not use condition variables, the Pthread lock is only requested by the holder of the optimized lock associated with the critical section. Furthermore, workloads that use condition variables are unlikely to have more than two threads competing for the Pthread lock: the holder of the optimized lock and a notified thread. Note that the latter claim also holds for workloads that rely on `pthread_cond_broadcast()` because the Linux implementation of this call only wakes up a single thread from the wait queue of the condition variable and directly transfers the remaining threads to the wait queue of the Pthread lock.

Support for specific lock semantics The design of LiTL is compatible with specific lock semantics when the underlying lock algorithms offer the corresponding properties. For example, LiTL supports non-blocking lock requests (`pthread_mutex_trylock()`) for all the currently implemented locks except CLH-based locks and Hticket-LS, which are not compatible with such semantics. Although not yet implemented, LiTL could easily support blocking requests with timeouts for the so-called "abortable" locks (e.g., MCS-Try [34] and MCS-TimePub [19]). Moreover, support for optional Pthread

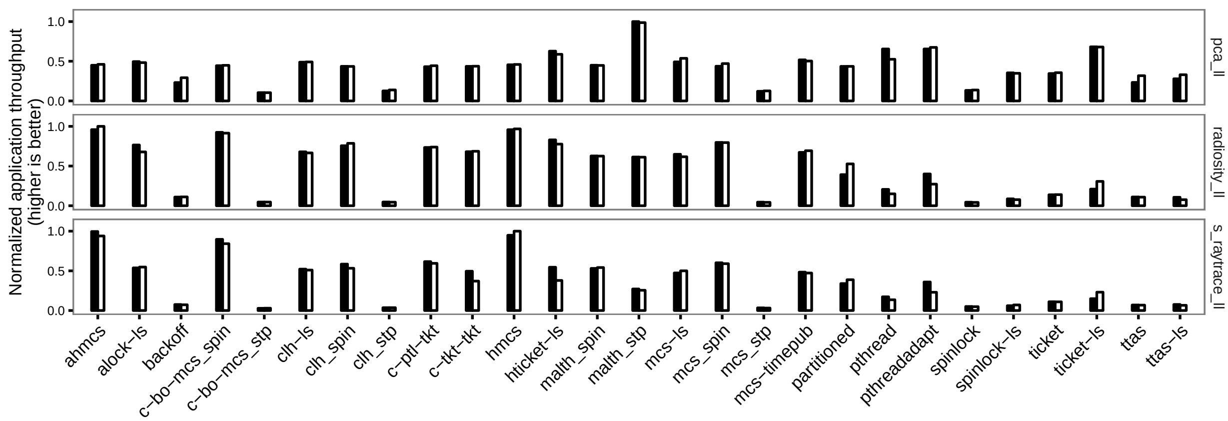

Figure 2: Performance comparison (throughput) of manually implemented locks (black bars) vs. transparently interposed locks using LiTL (white bars). The throughput is normalized with respect to the best performing configuration for a given application (**A-64 machine**).

mutex behavior like reentrance and error checks[5] could be easily integrated in the generic wrapper code by managing fields for the current owner and the lock acquisition counter.

4.2 Implementation

The library relies on a scalable concurrent hash table (CLHT [10]) in order to store, for each Pthread mutex instance used in the application, the corresponding optimized lock instance, and the associated per-thread structs. For well-established locking algorithms like MCS, the code of LiTL borrows from other libraries [9, 1, 26]. Other algorithms are implemented from scratch based on the description of the original papers. For algorithms that are based on a parking or on a spinning-then-parking waiting policy, our implementation directly relies on the `futex` Linux system call.

Finally, the source code of LiTL relies on preprocessor macros rather than function pointers. Indeed, we have observed that the use of function pointers in the critical path introduced a surprisingly high overhead. Moreover, all data structures are cache-aligned in order to mitigate the impact of false sharing.

4.3 Experimental validation

In this section, we assess the performance of LiTL using the A-64 machine. To that end, we compare the performance (throughput) of each lock on a set of applications running in two distinct configurations: manually modified applications and unmodified applications using interposition with LiTL. Clearly, one cannot expect to obtain exactly the same results in both configurations, as the setups differ in several ways, e.g., with respect to the exercised code paths, the process memory layout and the allocation of the locks (e.g., stack- vs. heap-based). However, we show that between both configurations: (i) the achieved performance is close and (ii) the general trends for the different locks remain stable.

We selected three applications: pca_ll, radiosity_ll and s_raytrace_ll. These three applications are particularly lock-intensive and the last two use Pthread condition variables. Therefore, all three represent an unfavorable case for LiTL. Moreover, we focus the discussion on the results under the highest contention level (i.e., when the application uses all the cores of the target machine), as this again represents an unfavorable case for LiTL.

Figure 2 shows the normalized performance (throughput) of both configurations (manual/interposed) for each *(application, lock)* pair: black bars correspond to manually implemented locks, whereas white bars correspond to transparently interposed locks using LiTL. In addition, Table 2 summarizes the performance differences for each application: number of locks for which each version performs better and, in each case, the average gain and the relative standard deviation.

We observe that, for all of the three applications, the results achieved by the two versions of the same lock are very close: the average performance difference is below 5%. Besides, Figure 2 highlights that the general trends observed with the manual versions are preserved with the interposed versions. We thus conclude that using LiTL to study the behavior of lock algorithms in an application yields only very modest differences with respect to the performance behavior of a manually modified version.

[5] Using respectively the `PTHREAD_MUTEX_RECURSIVE` and `PTHREAD_MUTEX_ERRORCHECK` attributes.

		pca_ll	radiosity_ll	s_raytrace_ll
Manual	Winners	10	17	19
	Average Gain	2%	3%	4%
	Rel. Dev.	4%	4%	5%
LiTL	Winners	17	10	8
	Average Gain	2%	3%	3%
	Rel. Dev.	2%	5%	3%

Table 2: Detailed statistics for the performance comparison of manually implemented locks vs. transparently interposed locks using LiTL (**A-64 machine**).

5 Performance study of lock algorithms

In this section, we use LiTL to compare the behavior of the different lock algorithms on different workloads and at different levels of contention. In the interest of space, we do not systematically report the observed standard deviations. However, in order to mitigate the impact of variability, when comparing the performance of two locks, we consider a margin of 5%: lock A is considered better than lock B if B's achieved performance is below 95% of A's. Besides, in order to make fair comparisons, the results presented for the Pthread locks are obtained using the same library interposition mechanism as with the other locks.

Note that some configurations are not tested because of specific restrictions. First, streamcluster, streamcluster_ll, and vips cannot use CLH-based locks or Hticket-LS as they do not support trylocks semantics. Second, we omit the results for most locks with MySQL: given the extremely large ratio of threads to cores, most locks yield performance close to zero. Third, some applications, e.g., dedup and fluidanimate, run out of memory for some configurations.

Finally, for the sake of space, we do not report all the results for the three studied machines. We rather focus on the A-64 machine and provide summaries of the results for the A-48 and I-48 machines. Nevertheless, the entire set of results can be found in the Appendices.

The section is structured as follows. §5.1 provides preliminary observations that drive the study. §5.2 answers the main questions of the study regarding the observed lock behavior. §5.3 discusses additional observations.

5.1 Preliminary observations

Before proceeding with the detailed study, we highlight some important characteristics of the applications.

5.1.1 Selection of lock-sensitive applications

Table 3 shows two metrics for each application and for different numbers of nodes on the A-64 machine: the performance gain of the best lock over the worst one, as well as the relative standard deviation for the performance of the different locks. For the moment, we only focus on the relative standard deviations at the maximum number of nodes (*max nodes*—highest contention) given in the 5th column (the detailed results from this table are discussed in §5.2.1).

We consider that an application is *lock-sensitive* if the relative standard deviation for the performance of the different locks at max nodes is higher than 10% (highlighted in bold font). We observe that about 60% of the applications are impacted by locks. We observe similar trends on the three studied machines (Tables 4 , 15, 16).

In the remainder of this study, we focus on lock-sensitive applications.

	Gain 1 node	R.Dev. 1 node	Gain max nodes	R.Dev. max nodes	Gain opt nodes	R.Dev. opt nodes
barnes	10%	2%	36%	8%	31%	7%
blackscholes	11%	2%	2%	1%	2%	1%
bodytrack	1%	0%	9%	2%	4%	1%
canneal	5%	1%	7%	2%	7%	2%
dedup	**683%**	**56%**	**970%**	**55%**	**683%**	**56%**
facesim	**10%**	**2%**	**771%**	**76%**	**14%**	**3%**
ferret	**1%**	**0%**	**349%**	**58%**	**107%**	**25%**
fft	8%	2%	11%	3%	9%	2%
fluidanimate	**48%**	**11%**	**302%**	**28%**	**133%**	**20%**
fmm	**26%**	**7%**	**42%**	**12%**	**42%**	**11%**
freqmine	7%	2%	6%	1%	6%	1%
histogram	7%	2%	20%	5%	12%	3%
kmeans	9%	3%	12%	2%	12%	2%
linear_regression	**9%**	**2%**	**228%**	**22%**	**49%**	**10%**
lu_cb	11%	2%	5%	1%	5%	1%
lu_ncb	17%	5%	8%	2%	8%	2%
matrix_multiply	**7%**	**3%**	**643%**	**51%**	**372%**	**38%**
mysqld	**30%**	**9%**	**174%**	**38%**	**122%**	**34%**
ocean_cp	**17%**	**4%**	**129%**	**15%**	**22%**	**5%**
ocean_ncp	**21%**	**5%**	**118%**	**14%**	**18%**	**4%**
pca	**12%**	**3%**	**358%**	**31%**	**47%**	**8%**
pca_ll	**19%**	**5%**	**665%**	**47%**	**100%**	**20%**
p_raytrace	2%	0%	1%	0%	2%	0%
radiosity	**3%**	**1%**	**91%**	**13%**	**13%**	**4%**
radiosity_ll	**8%**	**2%**	**2299%**	**71%**	**180%**	**29%**
radix	2%	1%	8%	2%	8%	2%
s_raytrace	**4%**	**1%**	**1929%**	**62%**	**126%**	**29%**
s_raytrace_ll	**4%**	**1%**	**3343%**	**79%**	**157%**	**26%**
ssl_proxy	**37%**	**6%**	**1309%**	**63%**	**58%**	**11%**
streamcluster	**13%**	**3%**	**1087%**	**56%**	**13%**	**3%**
streamcluster_ll	**23%**	**4%**	**1305%**	**55%**	**56%**	**12%**
string_match	5%	2%	11%	2%	11%	2%
swaptions	8%	2%	10%	2%	10%	2%
vips	**2%**	**1%**	**334%**	**32%**	**8%**	**2%**
volrend	**7%**	**1%**	**161%**	**21%**	**24%**	**5%**
water_nsquared	**10%**	**2%**	**94%**	**14%**	**94%**	**14%**
water_spatial	**24%**	**5%**	**98%**	**15%**	**96%**	**15%**
word_count	4%	1%	17%	3%	12%	2%
x264	4%	1%	6%	2%	5%	2%

Table 3: For each application, performance gain of the best vs. worst lock and relative standard deviation (**A-64 machine**).

	A-64	A-48	I-48
# tested applications	39	33	33
# lock-sensitive applications	23	19	17

Table 4: Number of tested applications and number of lock-sensitive applications (**all machines**).

Applications	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls
dedup	-	252	129	89	95	229	200	204	125	117	75	96	119	119	106	110	113	80	136	120	126	147	118	141	121	145	197
facesim	412	908	425	172	55	888	895	78	460	328	324	379	711	71	1k	948	87	26	895	91	67	726	35	919	462	489	530
ferret	134	176		46		170	174		109	63	100	108	57		194	192			173					182	34		7
fluidanimate	-	72			9	-	-	-				-	7	53	8	12	54	7				16		13	11	6	65
fmm							15	12																			
histogram	95	88	90	95	95	87	92	92	84	79	94	90	90	88	89	85	109	84	89	125	88	107	87	105	102	97	104
linear_regression	44	227	12	21	132	67	45	34	7	49	44	15	25	8	51	47	24		50	10	8	38	8	21			27
matrix_multiply		259								92	287	66			62				7				64		65		55
mysqld	-	-	-	-		-	-	-	-	-	-	-	-		-	-		25	-			-	-	-	-	-	-
ocean_cp	107	97	114	81	70	103	124	121	89	92	96	73	87	75	111	114	82	45	103	72	73	234	49	136	60	106	173
ocean_ncp	93	99	90	73	69	90	93	79	76	90	81	73	84	85	73	92	95	61	98	97	85	206	56	89	57	93	186
pca	77	79	163	42	370	69	44	148	40	34	68	49	37		49	55	134	19	50	97	36	229	80	116	35	160	130
pca_ll	91	81	219	14	582	74	41	321	23	16	88	31	7	21	58	41	403		21	195	114	513	168	108	51	206	476
radiosity																	69						21		10		53
radiosity_ll		12	413		1k	13	10	699	33	19			7		13	11	792	18	48	157	71	987	164	296	97	411	615
s_raytrace		18	185		1k		66	460		14	13	16		7			436		100	88	14	269	50	134	149	195	154
s_raytrace_ll	19	96	781	17	2k	110	107	1k	83	180	15	170	68	161	108	88	1k	118	178	371	185	1k	308	495	301	857	881
ssl_proxy	44	69	695	33	1k	107	61	1k	61	103	608	78	36	52	95	99	1k	73	87	268	195	2k	268	360	139	718	957
streamcluster	2k	2k	4k	2k	2k	-	-	-	1k	2k	1k	-	4k	16k	4k	3k	16k	1k	1k	2k	3k	9k	2k	5k	4k	4k	7k
streamcluster_ll	421	246	829	410	497	-	-	-	266	275	250	-	816	4k	774	590	4k	301	275	446	450	2k	585	1k	615	718	1k
vips	64	56	22	400	32	-	-	-	331	189	131	-	229	18	46	51	18	21	60	20	21	20	23	37	28	22	26
volrend	52	88	97	62	99	72	82	123	50	62	52	59	69	128	79	86	109	82	83	131	162	222	114	74	70	108	154
water_nsquared																											
water_spatial																											

Table 5: For each *(application, lock)* pair, performance gain (in %) of the optimized configuration over the max-node configuration. The background color of a cell indicates the number of nodes (1, 2, 4, 6, or 8 nodes) for the optimized configuration: 1 | 2 | 4 | 6 | 8. Dashes correspond to untested cases. (**A-64 machine**).

5.1.2 Selection of the number of nodes

In multicore applications, optimal performance is not always achieved at the maximum number of available nodes (abbreviated as *max nodes*) due to various kinds of scalability bottlenecks. Therefore, for each *(application, lock)* pair, we empirically determine the *optimized configuration* (abbreviated as *opt nodes*), i.e., the number of nodes that yields the best performance. For the A-64 and A-48 machines, we consider 1, 2, 4, 6, and 8 nodes. For the I-48 machines, we consider 1, 2, 3, and 4 nodes. Note that 6 nodes on A-64 and A-48 correspond to 3 nodes on I-48, i.e., 75% of the available cores.

The results are displayed in Tables 5, 18, 19 (resp. for the A-64, A-48 and I-48 machines). For each *(application, lock)* pair, the corresponding cell indicates the performance gain of the optimized configuration with respect to the max-node configuration. The background color of a cell indicates the number of nodes for the optimized configuration. In addition, Table 6 provides a breakdown of the *(application, lock)* pairs according to their optimized number of nodes for all machines.

We observe that, for many applications, the optimized number of nodes is lower than the max number of nodes. Moreover, we observe (Table 5) that the performance gain of the optimized configuration is often extremely large. This confirms that tuning the degree of parallelism has frequently a very strong impact on performance. We also notice that, for some applications, the optimized number of nodes varies according to the chosen lock.

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

	I-48
1 Node	33%
2 Nodes	14%
3 Nodes	8%
4 Nodes	45%

Table 6: Breakdown of the *(application, lock)* pairs according to their optimized number of nodes (**all machines**).

In light of the above observations, the main questions investigated in the study (§5.2) will be considered from two complementary angles: (i) comparing locks at a fixed number of nodes, and (ii) comparing locks at their optimized configurations (i.e., with possibly a different number of nodes for each). The first angle offers insight for situations in which the degree of parallelism cannot be adjusted, while the second is useful for scenarios in which more advanced application tuning is possible.

5.2 Main questions

5.2.1 How much do locks impact applications?

Table 3 shows, for each application, the performance gain of the best lock over the worst one at 1 node, max nodes, and opt nodes for the A-64 machine. The table also shows the relative standard deviation for the performance of the different locks.

We observe that the impact of locks on the performance of applications depends on the number of nodes. **At 1 node, the impact of locks on lock-sensitive applications is moderate**. More precisely, most applications exhibit a gain of the best lock over the worst one that is lower than 30%. In contrast, **at max nodes, the impact of locks is very high for all lock-sensitive applications**. More precisely, the gain brought by the best lock over the worst lock ranges from 42% to 3343%. Finally, **at the optimized number of nodes, the impact of locks is high, but noticeably lower than at max nodes.** We explain this difference by the fact that, at max nodes, some of the locks trigger a performance collapse for certain applications (as shown in Table 5), which considerably increases the observed performance gaps between locks. We observe the same trends on the A-48 and I-48 machines (Tables 15 and 16 in the Appendices).

5.2.2 Are some locks always among the best?

Table 7 shows the *coverage* of each lock, i.e., how often it stands as the best one (or is within 5% of the best) over all the studied applications for the A-64 machine. The results are shown for three configurations: 1 node, max nodes, and opt nodes. Besides, Table 8 displays, for each machine (at 1 node, max nodes and opt nodes) the following metrics aggregated over the different locks: the min and max coverage, the average coverage, and the relative standard deviation of the coverage.

	Number of nodes		
Locks	1	Max	Opt
ahmcs	67%	24%	52%
alock-ls	52%	4%	30%
backoff	83%	30%	26%
c-bo-mcs_spin	74%	22%	39%
c-bo-mcs_stp	62%	12%	29%
clh-ls	63%	5%	37%
clh_spin	68%	5%	37%
clh_stp	63%	16%	21%
c-ptl-tkt	57%	22%	35%
c-tkt-tkt	74%	22%	39%
hmcs	65%	22%	48%
hticket-ls	63%	16%	37%
malth_spin	61%	9%	26%
malth_stp	54%	29%	29%
mcs-ls	74%	4%	30%
mcs_spin	70%	22%	48%
mcs_stp	79%	21%	29%
mcs-timepub	54%	38%	29%
partitioned	70%	22%	39%
pthread	50%	21%	29%
pthreadadapt	58%	33%	29%
spinlock	65%	26%	30%
spinlock-ls	57%	30%	35%
ticket	74%	22%	39%
ticket-ls	74%	13%	35%
ttas	83%	26%	43%
ttas-ls	65%	0%	9%

Table 7: For each lock, fraction of the lock-sensitive applications for which the lock yields the best performance for three configurations: 1 node, max nodes, and opt nodes (**A-64 machine**).

# nodes	Coverage	A-64	A-48	I-48
1	[min; max]	[50%; 83%]	[27%; 83%]	[44%; 89%]
	Avg.	66%	66%	62%
	Rel. Dev.	9%	15%	12%
Max	[min; max]	[0%; 38%]	[0%; 42%]	[5%; 50%]
	Avg.	19%	17%	24%
	Rel. Dev.	10%	12%	11%
Opt	[min; max]	[9%; 52%]	[0%; 47%]	[5%; 50%]
	Avg.	34%	21%	28%
	Rel. Dev.	9%	13%	12%

Table 8: Statistics on the coverage of locks for three configurations: 1 node, max nodes, and opt nodes (**all machines**).

We make the following observations (Table 8). **No lock is among the best for more than 89% of the applications at 1 node and for more than 52% of the applications both at max nodes and at the optimal number of nodes**. We also observe that the average coverage is much higher at 1 node than at max nodes, and slightly higher at the optimized number of nodes than at max nodes. This is directly explained by the observations made in §5.2.1. First, at 1 node, locks have a much lower impact on applications than in other configurations and thus yield closer results, which increases their likelihood to be among the best ones. Second, at max nodes, all of the different locks cause, in turn, a performance collapse, which reduces their likelihood to be among the best locks. This latter phenomenon is not observed at the optimized number of nodes. We observe the same trends on the A-48 and I-48 machines (Tables 21 and 22 in the Appendices).

5.2.3 Is there a clear hierarchy between locks?

Table 9 shows pairwise comparisons for all locks, at max nodes on the A-64 machine. In each table, cell $(rowA, colB)$ contains the score of lock A vs. lock B, i.e., the percentage of applications for which lock A is at least 5% better than lock B. For example, Table 9 shows that for 38% of the applications, AHMCS performs at least 5% better than Backoff at the optimized number of nodes. Similarly, the table shows that Backoff is at least 5% better than AHMCS for 29% of the applications. From these two values, we can conclude that the two above mentioned locks perform very closely for 33% of the applications. At the end of each line (resp. column), the table also shows the mean of the fraction of applications for which a lock is better (resp. worse) than others. Besides, the latter two metrics are summarized for the three machines in Table 10.

We observe that **there is no clear global performance hierarchy between locks**. More precisely, for most pairs of locks (A, B), there are some applications for which A is better than B, and vice-versa (Table 9). The only marginal exceptions are the cells having 0% for value. This corresponds to pairs of locks (A, B) for which A

	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls	average
ahmcs		19	38	48	29	22	17	61	19	48	5	33	33	43	38	38	48	52	24	38	43	57	48	33	33	43	38	36
alock-ls	19		39	30	26	16	16	58	17	22	9	26	39	30	22	26	43	30	9	39	43	48	39	35	30	35	39	30
backoff	29	35		30	26	37	37	58	26	26	35	32	35	26	35	30	52	30	17	35	39	30	26	4	22	0	39	30
c-bo-mcs_spin	33	48	43		35	37	32	74	22	17	39	32	39	48	39	9	48	13	22	39	39	39	43	48	39	35	65	38
c-bo-mcs_stp	33	43	35	22		42	32	74	17	22	30	21	22	25	26	26	42	21	13	33	33	39	26	26	22	26	61	31
clh-ls	22	21	37	42	32		16	47	26	26	16	26	37	37	16	32	47	26	16	42	47	53	47	47	42	42	47	34
clh_spin	22	32	32	32	26	32		53	21	37	21	42	32	26	32	21	47	32	11	37	37	47	42	32	42	37	47	33
clh_stp	33	32	5	16	11	37	16		26	16	26	26	16	11	21	16	11	5	11	11	11	21	21	11	26	11	32	18
c-ptl-tkt	19	35	35	39	30	32	21	68		26	22	26	26	43	30	26	57	39	17	39	35	48	35	30	30	35	57	35
c-tkt-tkt	24	39	35	26	39	32	26	74	26		30	32	48	65	43	17	57	22	9	39	43	39	43	39	43	35	65	38
hmcs	14	30	39	35	22	42	32	74	17	39		32	39	35	35	26	52	39	26	39	39	48	39	30	30	30	52	36
hticket-ls	17	16	47	32	26	21	32	74	11	21	5		32	42	11	26	53	32	11	42	42	53	42	37	26	47	58	33
malth_spin	14	35	22	22	26	26	16	63	13	17	22	16		22	22	13	39	17	4	35	35	35	39	17	13	17	48	25
malth_stp	24	35	22	35	21	32	37	58	17	17	26	21	4		22	17	33	25	9	33	29	35	22	17	17	17	48	26
mcs-ls	24	17	35	35	35	21	26	63	13	17	17	16	35	26		17	39	17	4	39	43	43	35	30	17	35	48	29
mcs_spin	29	43	35	26	39	37	32	68	26	17	39	47	39	43	43		43	22	22	35	39	35	43	39	30	39	61	37
mcs_stp	29	35	9	22	21	32	32	42	22	9	30	26	17	17	26	9		12	17	21	25	17	17	13	17	13	39	22
mcs-timepub	33	39	35	22	33	42	37	68	17	9	30	32	39	29	22	9	38		13	29	33	30	35	30	30	30	57	32
partitioned	24	39	26	39	43	32	32	68	26	22	39	53	52	43	35	35	61	35		43	48	48	43	26	43	35	65	41
pthread	29	39	22	26	25	37	32	58	22	17	39	26	30	25	35	26	46	25	13		21	39	13	17	13	17	43	28
pthreadadapt	29	43	22	35	21	37	37	53	30	26	35	26	26	25	35	30	42	25	17	21		22	22	17	17	17	43	29
spinlock	29	39	9	26	17	37	32	53	35	13	39	32	43	35	35	22	39	17	22	26	30		26	13	30	9	35	29
spinlock-ls	29	39	26	30	35	26	26	63	26	30	35	16	30	30	30	30	48	30	22	43	30	48		26	13	26	57	33
ticket	29	35	9	26	26	32	32	63	26	22	35	32	30	26	30	26	48	22	13	26	39	30	26		22	0	39	29
ticket-ls	19	22	30	26	39	26	32	68	26	26	22	11	35	39	22	26	52	26	26	35	48	43	39	30		30	52	33
ttas	24	35	4	26	22	37	26	63	26	17	35	32	30	26	30	30	52	17	17	30	35	30	26	4	26		30	28
ttas-ls	19	17	9	17	13	21	16	42	13	13	4	5	22	22	9	22	30	9	13	17	22	30	17	13	4	9		17
average	25	33	27	29	28	32	28	62	22	22	26	28	32	32	29	23	45	25	15	33	36	39	33	26	26	26	49	

Table 9: For each pair of locks *(rowA, colB)* at the optimized number of nodes, score of lock A vs lock B: percentage of applications for which lock A performs at least 5% better than B (**A-64 machine**).

	Better			Worse		
Lock	A-64	A-48	I-48	A-64	A-48	I-48
ahmcs	36%	40%	52%	25%	28%	25%
alock-ls	30%	42%	37%	33%	25%	32%
backoff	30%	29%	23%	27%	33%	45%
c-bo-mcs_spin	38%	47%	46%	29%	25%	15%
c-bo-mcs_stp	31%	25%	38%	28%	44%	25%
clh-ls	34%	46%	32%	32%	32%	38%
clh_spin	33%	38%	33%	28%	34%	37%
clh_stp	18%	11%	8%	62%	72%	71%
c-ptl-tkt	35%	44%	54%	22%	26%	13%
c-tkt-tkt	38%	42%	51%	22%	27%	15%
hmcs	36%	50%	52%	26%	21%	17%
hticket-ls	33%	45%	42%	28%	25%	17%
malth_spin	25%	36%	31%	32%	37%	35%
malth_stp	26%	20%	28%	32%	53%	36%
mcs-ls	29%	43%	35%	29%	22%	26%
mcs_spin	37%	38%	36%	23%	33%	23%
mcs_stp	22%	23%	20%	45%	59%	52%
mcs-timepub	32%	38%	34%	25%	34%	29%
partitioned	41%	42%	38%	15%	32%	23%
pthread	28%	33%	34%	33%	43%	35%
pthreadadapt	29%	34%	34%	36%	38%	36%
spinlock	29%	35%	20%	39%	44%	49%
spinlock-ls	33%	41%	38%	33%	30%	31%
ticket	29%	23%	17%	26%	44%	53%
ticket-ls	33%	40%	28%	26%	24%	35%
ttas	28%	28%	24%	26%	34%	44%
ttas-ls	17%	27%	20%	49%	42%	52%

Table 10: For each lock, at the optimized number of nodes, mean of the fraction of applications for which the lock is better (resp. worse) than other locks (**all machines**).

never yields better performance than *B*. The results at max nodes (Table 24 in the Appendices) exhibit similar trends as the ones at opt nodes. Besides, we make the same observations (both at opt nodes and max nodes) on the A-48 and I-48 machines (Tables 25, 26, 27 and 28 in the Appendices).

5.2.4 Are all locks potentially harmful?

Our goal is to determine, for each lock, if there are applications for which it yields substantially lower performance than other locks and to quantify the magnitude of such performance gaps. Table 11 displays, for the A-64 machine, the performance gain brought by the best lock with respect to each of the other locks for each application at max nodes (top part) and at the optimized number of nodes for each lock (bottom part). For example, the top part of the table shows that for the dedup application, the best lock (0%, here Spinlock-LS) is 598% better than the Alock-LS lock. The gray cells highlight values greater than 15%. Thus, for each lock in a column, the number of grey cells corresponds to the number of applications for which the lock is beaten by a gap of 15% or more by the best lock(s) for this application. In addition, Table 12 displays, for each machine, the fraction of applications that are significantly hurt by a given lock.

On the three machines, we observe that, **both at max**

Applications	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls
Max nodes																											
dedup	-	598	4	135	137	970	575	576	27	11	145	130	130	129	123	127	128	105	14	6	2	2	0	4	0	5	579
facesim	298	701	323	107	25	680	687	52	333	224	234	273	531	40	771	710	52	0	685	56	44	572	6	719	340	368	409
ferret	329	297	10	84	0	261	312	0	286	228	255	291	196	0	349	317	0	4	314	0	1	10	0	331	84	9	11
fluidanimate	-	301	0	57	65	-	-	-	35	14	72	-	36	95	50	40	94	50	14	5	12	26	0	17	15	9	201
fmm	41	37	15	3	26	38	39	33	30	0	35	32	16	14	32	2	0	0	14	25	23	2	25	15	27	17	34
histogram	1	2	8	3	4	3	3	12	2	0	2	0	0	1	5	1	14	1	4	19	2	18	3	11	5	8	12
linear_regression	32	228	24	20	108	57	31	62	0	52	28	11	17	0	49	46	56	3	39	15	0	83	15	32	9	19	49
matrix_multiply	9	559	5	26	7	18	9	3	24	136	608	642	5	3	639	27	2	0	33	3	3	5	637	3	633	5	630
mysqld	-	-	-	-	30	-	-	-	-	-	-	-	-	0	-	-	7	173	-	97	102	-	-	-	-	-	-
ocean_cp	31	18	37	22	16	27	38	38	24	29	29	15	23	27	27	43	32	0	24	11	19	129	5	55	5	38	81
ocean_ncp	27	28	29	30	9	25	27	28	12	28	16	10	20	22	14	36	37	11	29	31	27	118	0	25	2	29	93
pca	65	69	155	46	357	61	48	220	40	38	59	39	38	0	43	58	214	23	45	110	39	252	75	110	23	157	112
pca_ll	47	38	251	24	664	25	51	511	30	24	41	0	18	36	17	50	526	15	27	206	68	584	128	128	17	241	338
radiosity	14	12	0	0	1	13	9	0	8	1	7	9	9	12	10	1	91	0	1	0	0	1	33	0	19	0	71
radiosity_ll	0	47	801	9	2k	50	16	2k	35	45	3	28	59	63	62	12	2k	44	76	567	267	2k	396	614	193	825	1k
s_raytrace	2	24	536	17	2k	9	75	1k	8	27	18	38	26	64	16	0	1k	13	122	230	122	714	118	412	225	554	471
s_raytrace_ll	6	82	1k	18	3k	96	87	3k	68	169	0	164	84	291	99	69	3k	111	157	639	335	2k	428	813	332	1k	1k
ssl_proxy	0	18	532	1	1k	47	16	879	9	41	379	20	16	35	43	47	900	29	36	293	153	1k	249	271	85	539	735
streamcluster	45	24	153	13	63	-	-	-	7	13	3	-	210	1k	183	118	979	6	0	90	133	505	33	290	166	177	395
streamcluster_ll	61	6	188	20	55	-	-	-	0	17	6	-	234	1k	202	133	1k	34	13	77	102	518	65	263	139	155	411
vips	41	38	4	333	17	-	-	-	267	145	101	-	177	0	28	28	1	3	37	0	2	3	1	16	8	4	10
volrend	2	28	41	9	34	16	25	58	1	9	0	6	17	63	22	26	47	24	24	78	104	161	58	24	16	51	92
water_nsquared	94	48	2	2	9	58	35	35	7	0	14	10	7	6	9	3	2	7	4	6	7	0	6	4	6	4	37
water_spatial	97	49	2	11	7	63	40	39	4	5	8	4	8	5	5	9	9	10	1	0	0	2	1	1	0	1	41
Opt nodes																											
dedup	-	378	10	199	193	682	443	436	36	23	237	183	153	152	161	160	158	174	16	16	9	0	10	3	10	3	451
facesim	2	4	6	0	6	4	4	12	1	0	4	2	2	8	3	1	7	4	3	7	13	7	3	5	3	4	6
ferret	88	47	6	29	0	37	53	0	89	106	82	92	93	0	56	46	0	3	55	0	0	7	0	56	41	6	7
fluidanimate	-	133	0	50	51	-	-	-	35	14	64	-	28	27	39	25	26	40	14	5	12	9	0	4	3	3	83
fmm	41	35	15	3	26	38	21	19	30	0	33	32	16	14	32	2	0	0	14	25	23	1	25	15	27	17	34
histogram	0	5	9	1	2	6	3	11	6	6	1	1	1	3	6	4	4	5	6	2	3	9	5	3	0	4	5
linear_regression	2	12	24	11	0	5	1	35	4	14	0	8	5	4	10	11	39	14	4	16	4	48	19	22	15	25	30
matrix_multiply	9	83	5	22	7	18	9	3	24	23	83	348	5	3	357	23	2	0	24	3	3	5	349	3	343	5	372
mysqld	-	-	-	-	31	-	-	-	-	-	-	-	-	0	-	-	8	121	-	96	96	-	-	-	-	-	-
ocean_cp	5	0	7	12	13	4	2	4	10	12	10	11	9	21	0	11	20	14	2	7	15	14	18	9	9	12	10
ocean_ncp	3	1	6	17	1	3	3	12	0	5	0	0	2	3	3	10	10	8	2	4	7	11	0	4	2	5	5
pca	2	4	6	13	6	4	12	41	10	12	4	3	11	7	5	12	47	13	6	17	12	17	7	7	0	8	1
pca_ll	6	5	51	49	54	0	48	100	46	48	3	5	53	55	3	46	71	51	45	43	8	53	17	51	7	53	5
radiosity	10	9	0	0	1	10	8	0	6	1	7	9	7	10	8	1	13	0	1	0	0	1	10	0	9	0	11
radiosity_ll	0	31	75	9	53	32	5	180	1	22	3	28	49	59	42	1	165	22	19	159	114	120	88	80	49	80	83
s_raytrace	2	5	123	16	74	9	5	123	5	11	5	19	26	53	14	0	117	12	10	75	94	120	45	119	30	121	125
s_raytrace_ll	2	6	79	16	74	7	4	157	5	10	0	11	25	72	9	3	150	11	6	79	74	75	48	75	23	76	78
ssl_proxy	3	4	17	12	23	5	7	30	0	3	0	0	26	31	9	9	23	11	7	57	27	20	40	19	15	15	16
streamcluster	11	9	6	0	4	-	-	-	8	1	7	-	10	10	9	1	2	5	7	12	7	2	2	8	8	7	9
streamcluster_ll	30	29	31	0	9	-	-	-	15	31	28	-	54	47	46	42	39	41	27	36	55	46	2	33	41	31	35
vips	4	7	3	4	7	-	-	-	3	3	5	-	2	2	5	2	3	3	3	0	1	4	0	2	2	3	5
volrend	2	4	9	2	2	3	4	8	3	2	0	1	5	8	4	3	7	4	3	17	18	23	12	8	4	10	15
water_nsquared	94	48	2	2	9	58	35	35	7	0	14	10	7	6	9	3	2	7	4	6	7	0	6	4	6	4	37
water_spatial	95	49	2	11	7	63	40	39	4	5	8	4	8	5	5	9	9	10	1	0	0	2	1	1	0	1	41

Table 11: For each application, at max nodes (top part) and at the optimized number of nodes (bottom part), performance gain (in %) obtained by the best lock(s) with respect to each of the other locks. The grey background highlights cells for which the performance gains are greater than 15%. A line with many gray cells corresponds to an application whose performance is hurt by many locks. A column with many gray cells corresponds to a lock that is outperformed by many other locks. Dashes correspond to untested cases. (**A-64 machine**).

nodes and at the optimal number of nodes, all locks are potentially harmful, yielding sub-optimal performance for a significant number of applications (Table 12). We also notice that locks are significantly less harmful at the optimized number of nodes than at max nodes. This is explained by the fact that several of the locks create performance collapses at max nodes, which does not occur at the optimized number of nodes. Moreover, we observe that, for each lock, the performance gap to the best lock can be significant (Tables 11, 31 and 32).

5.3 Additional observations

Impact of the number of nodes. Table 13 shows, for each application on the A-64 machine, the number of pairwise changes in the lock performance hierarchy when the number of nodes is modified. For example, in the case of the facesim application, there are 18% of the pairwise performance comparisons between locks that change when moving from a 1-node configuration to a 2-node configuration. Similarly, there are 95% of pairwise comparisons that change at least once when considering

Lock	A-64		A-48		I-48	
	Max	Opt	Max	Opt	Max	Opt
ahmcs	62%	24%	56%	39%	39%	33%
alock-ls	87%	39%	61%	39%	58%	58%
backoff	61%	35%	68%	53%	58%	53%
c-bo-mcs_spin	61%	35%	53%	58%	47%	32%
c-bo-mcs_stp	71%	38%	80%	65%	55%	45%
clh-ls	84%	37%	73%	40%	69%	62%
clh_spin	84%	32%	60%	47%	62%	56%
clh_stp	79%	58%	87%	87%	81%	75%
c-ptl-tkt	52%	30%	53%	42%	47%	26%
c-tkt-tkt	61%	26%	58%	42%	53%	26%
hmcs	61%	26%	37%	37%	37%	16%
hticket-ls	58%	32%	44%	38%	50%	50%
malth_spin	78%	43%	63%	53%	53%	53%
malth_stp	54%	38%	65%	60%	55%	55%
mcs-ls	78%	30%	63%	47%	58%	58%
mcs_spin	70%	26%	63%	53%	58%	58%
mcs_stp	67%	46%	70%	65%	70%	60%
mcs-timepub	42%	25%	65%	55%	50%	50%
partitioned	61%	26%	68%	47%	63%	47%
pthread	62%	50%	60%	55%	60%	55%
pthreadadapt	58%	38%	55%	50%	55%	50%
spinlock	65%	39%	68%	58%	63%	53%
spinlock-ls	57%	39%	58%	42%	58%	47%
ticket	74%	39%	79%	63%	74%	63%
ticket-ls	65%	39%	58%	47%	63%	47%
ttas	61%	35%	68%	53%	63%	58%
ttas-ls	87%	57%	78%	61%	74%	68%

Table 12: For each lock, at max nodes and at the optimized number of nodes, fraction of the applications for which the lock is harmful (**all machines**).

the 1-node, 2-node, 4-node and 8-node configurations.

We observe that, **for all applications, the lock performance hierarchy changes significantly according to the chosen number of nodes**. Moreover, we observe the same trends on the A-48 and I-48 machines (Tables 34 and 35 in the Appendices).

Applications	% of pairwise changes between configurations			
	1/2	2/4	4/8	1/2/4/8
dedup	16%	6%	12%	19%
facesim	18%	38%	81%	95%
ferret	0%	74%	26%	87%
fluidanimate	5%	6%	24%	32%
fmm	33%	10%	19%	45%
histogram	19%	32%	24%	55%
linear_regression	58%	40%	57%	95%
matrix_multiply	16%	27%	45%	54%
mysqld	33%	20%	7%	40%
ocean_cp	54%	53%	72%	94%
ocean_ncp	52%	54%	56%	86%
pca	44%	60%	29%	89%
pca_ll	31%	38%	23%	73%
radiosity	11%	49%	65%	83%
radiosity_ll	66%	28%	14%	92%
s_raytrace	1%	70%	32%	96%
s_raytrace_ll	21%	69%	24%	99%
ssl_proxy	62%	12%	21%	78%
streamcluster	68%	21%	32%	88%
streamcluster_ll	60%	28%	31%	90%
vips	2%	3%	82%	82%
volrend	16%	27%	44%	85%
water_nsquared	23%	24%	13%	52%
water_spatial	12%	10%	10%	29%

Table 13: For each application, percentage of pairwise changes in the lock performance hierarchy when changing the number of nodes (**A-64 machine**).

Impact of the machine. Table 14 shows the number of pairwise lock inversions observed between the machines (both at max nodes and at the optimized number of nodes). More precisely, for a given application at a given node configuration, we check whether two locks are in the same order or not on the target machines.

We observe that **the lock performance hierarchy changes significantly according to the chosen machine**. Interestingly, we observe that there is approximately the same number of inversions between each pair of machines.

# nodes	A-64 vs. A-48	A-48 vs. I-48	A-64 vs. I-48
Max	38%	36%	38%
Opt	30%	29%	31%

Table 14: For each pair of machines, at max nodes and at opt nodes, percentage of pairwise changes in the lock performance hierarchy (**all machines**).

A note on Phtread locks. The various results presented in this paper show that the current Linux **Pthread locks perform well (i.e., are among the best locks) for a significant share of the studied applications**, thus providing a different insight than recent results, which were mostly based on synthetic workloads [9]. Beyond the changes of workloads, these differences may also be explained by the continuous refinement of the Linux Pthread implementation. It is nevertheless important to note that on each machine, some locks stand out as the best ones for a higher fraction of the applications than Pthread locks. Finally, we note that Pthread adaptive locks perform slightly better than standard Pthread locks.

Impact of thread pinning. As explained in §3.2, all the above-described experiments were run without any restriction on the placement of threads, leaving the corresponding decisions to the Linux scheduler. However, in order to better control CPU allocation and improve locality, some developers and system administrators use pinning to explicitly restrict the placement of each thread to one or several core(s). The impact of thread pinning may vary greatly according to workloads and can yield both positive and negative effects [9, 27]. In order to assess the generality of our observations, we also performed the complete set of experiments with an alternative configuration in which each thread is pinned to a given node, leaving the scheduler free to place the thread among the cores of the node. Note that for an experiment with a N-node configuration, the complete application runs on exactly first N nodes of the machine. We chose thread-to-node pinning rather than thread-to-core pinning because we observed that the former generally provided better performance for our studied applications, especially the ones using more threads than cores. The detailed results of our experiments with thread-to-node pinning are available in the companion technical report [18]. Overall, we observe that **all the conclusions presented in the paper still hold with per-node thread pinning**.

6 Related work

The design and implementation of the LiTL lock library borrows code and ideas from previous open-source toolkits that provide application developers with a set of optimized implementations for some of the most-established lock algorithms: Concurrency Kit [1], liblock [25, 24, 26], and libslock [9]. All of these toolkits require potentially tedious source code modifications in the target applications, even in the case of algorithms that have been specifically designed to lower this burden [3, 33, 36]. Moreover, among the above works, none of them provides a simple and generic solution for supporting Pthread condition variables. The authors of liblock [26] have proposed an approach but we discovered that it suffers from liveness hazards due to a race condition. Indeed, when a thread *T* calls `pthread_cond_wait()`, it is not guaranteed that the two steps (releasing the lock and blocking the thread) are always executed atomically. Thus, a wake-up notification issued by another thread may get interleaved between the two steps and *T* may remain indefinitely blocked.

Several research works have leveraged library interposition to compare different locking algorithms on legacy applications (e.g., Johnson et al. [21] and Dice et al. [14]) but, to the best of our knowledge, they have not publicly documented the design challenges to support arbitrary application patterns, nor disclosed the corresponding source code and the overhead of their interposition library has not been discussed.

Several studies have compared the performance of different multicore lock algorithms, either from a theoretical angle or based on experimental results [4, 33, 9, 24, 14]. In comparison, our study encompasses significantly more lock algorithms and waiting policies. Moreover, the bulk of these studies is mainly focused on characterization microbenchmarks while we focus instead on workloads designed to mimic real applications. Two noticeable exceptions are the work from Boyd-Wickizer et al. [4] and Lozi et al. [26] but they do not consider the same context as our study. The former is focused on kernel-level locking bottlenecks, and the latter is focused on applications in which only one or a few heavily contended critical sections have been optimized (after a profiling phase). For all these reasons, we make observations that are significantly different from the ones based on all the above-mentioned studies. Other synchronization-related studies like the one from Gramoli [16] have a different scope and focus on concurrent data structures, possibly based on other facilities than locks.

Finally, some tools have been proposed to facilitate the identification of locking bottlenecks in applications [35, 8, 26]. These publications are orthogonal to our work. We note that, among them, the profilers based on library interposition can be stacked on top of LiTL.

7 Conclusion and future work

Optimized lock algorithms for multicore machines are abundant. However, there are currently no clear guidelines and methodologies helping developers to select the right lock for their workloads. In this paper, we have presented a broad study of 27 locks algorithms with 35 applications on Linux/x86. To perform that study, we have implemented LiTL, an interposition library allowing the transparent replacement of lock algorithms used for Pthread mutex locks. From our study, we draw several conclusions, including the following ones: at its optimized contention level, no single lock dominates for more than 52% of the lock-sensitive applications; any of the locks is harmful for at least several applications; for a given application, the best lock varies according to both the number of contending cores and the machine that executes the application. These observations call for further research on optimized lock algorithms, as well as tools and dynamic approaches to better understand and control their behavior.

The source code of LiTL and the data sets of our experimental results are available online [17].

Acknowledgments

We thank the anonymous reviewers and our shepherd, Tim Harris, for their insightful comments on ealier drafts of this paper. Dave Dice provided detailed answers for our questions on Malthusian locks. Baptiste Lepers provided valuable insights for some of the case studies. Pierre Neyron provided his help to set up experiments on the I-48 machine. Finally, this work has been partially supported by: LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01), EmSoc Replicanos and AGIR CAEC projects of Université Grenoble-Alpes and GrenobleINP, and the INRIA/LIG Digitalis project.

References

[1] AL BAHRA, S. Concurrency Kit, 2015. `http://concurrencykit.org`.

[2] ANDERSON, T. E. The Performance of Spin Lock Alternatives for Shared-Memory Multiprocessors. *IEEE Transaction on Parallel and Distributed Systems* (Jan. 1990), 6–16.

[3] AUSLANDER, M., EDELSOHN, D., KRIEGER, O., ROSENBURG, B., AND WISNIEWSKI, R. Enhancement to the MCS Lock for Increased Functionality and Improved Programmability. U.S. Patent Application Number 20030200457 (abandoned), October 2003.

[4] BOYD-WICKIZER, S., KAASHOEK, M. F., MORRIS, R., AND ZELDOVICH, N. Non-scalable Locks are Dangerous. In *Proceedings of the Linux Symposium* (Ottawa, Canada, July 2012).

[5] CHABBI, M., FAGAN, M., AND MELLOR-CRUMMEY, J. High Performance Locks for Multi-level NUMA Systems. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'15)* (2015), ACM.

[6] CHABBI, M., AND MELLOR-CRUMMEY, J. Contention-conscious, Locality-preserving Locks. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'16)* (2016), ACM.

[7] CRAIG, T. S. Building FIFO and Priority-Queuing Spin Locks from Atomic Swap. Tech. Rep. TR 93-02-02, University of Washington, 1993.

[8] DAVID, F., THOMAS, G., LAWALL, J., AND MULLER, G. Continuously Measuring Critical Section Pressure with the Free-lunch Profiler. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications* (2014), OOPSLA '14, ACM.

[9] DAVID, T., GUERRAOUI, R., AND TRIGONAKIS, V. 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 (SOSP'13)* (2013), ACM.

[10] DAVID, T., GUERRAOUI, R., AND TRIGONAKIS, V. Asynchronized Concurrency: The Secret to Scaling Concurrent Search Data Structures. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'15)* (2015), ACM.

[11] DICE, D. Brief Announcement: A Partitioned Ticket Lock. In *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'11)* (2011), ACM.

[12] DICE, D. Malthusian Locks, november 2015. `http://arxiv.org/abs/1511.06035`.

[13] DICE, D., MARATHE, V. J., AND SHAVIT, N. Flat-Combining NUMA Locks. In *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'11)* (2011), ACM.

[14] DICE, D., MARATHE, V. J., AND SHAVIT, N. Lock Cohorting: A General Technique for Designing NUMA Locks. *ACM Transactions on Parallel Computing 1*, 2 (Feb. 2015), 13:1–13:42.

[15] FATOUROU, P., AND KALLIMANIS, N. D. Revisiting the Combining Synchronization Technique. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'12)* (2012), ACM.

[16] GRAMOLI, V. More Than You Ever Wanted to Know About Synchronization: Synchrobench, Measuring the Impact of the Synchronization on Concurrent Algorithms. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'15)* (2015), ACM.

[17] GUIROUX, H., LACHAIZE, R., AND QUÉMA, V. LiTL source code and data sets, 2016. `https://github.com/multicore-locks`.

[18] GUIROUX, H., LACHAIZE, R., AND QUÉMA, V. Multicore Locks: the Case is not Closed Yet. Technical report, 2016. Available from `https://github.com/multicore-locks`.

[19] HE, B., SCHERER, W. N., AND SCOTT, M. L. Preemption Adaptivity in Time-published Queue-based Spin Locks. In *Proceedings of the 12th International Conference on High Performance Computing (HiPC'05)* (2005), Springer-Verlag.

[20] HENDLER, D., INCZE, I., SHAVIT, N., AND TZAFRIR, M. Flat Combining and the Synchronization-Parallelism Tradeoff. In *Proceedings of the Twenty-second Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'10)* (2010), ACM.

[21] JOHNSON, F. R., STOICA, R., AILAMAKI, A., AND MOWRY, T. C. Decoupling Contention Management from Scheduling. In *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'10)* (2010), ACM.

[22] KARLIN, A. R., LI, K., MANASSE, M. S., AND OWICKI, S. Empirical Studies of Competitve Spinning for a Shared-memory Multiprocessor. In *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles (SOSP'91)* (1991), ACM.

[23] KYLHEKU, K. What is PTHREAD_MUTEX_ADAPTIVE_NP?, 2014. `http://stackoverflow.com/a/25168942`.

[24] LOZI, J.-P. *Towards More Scalable Mutual Exclusion for Multicore Architectures*. PhD thesis, UPMC, Paris, July 2014. `http://www.i3s.unice.fr/~jplozi/documents/lozi-phd-thesis.pdf`.

[25] LOZI, J.-P., DAVID, F., THOMAS, G., LAWALL, J., AND MULLER, G. Remote Core Locking: Migrating Critical-Section Execution to Improve the Performance of Multithreaded Applications. In *Proceedings of the 2012 USENIX Annual Technical Conference* (2012), USENIX Association.

[26] LOZI, J.-P., DAVID, F., THOMAS, G., LAWALL, J., AND MULLER, G. Fast and Portable Locking for Multicore Architectures. *ACM Transactions on Computer Systems 33*, 4 (Jan. 2016), 13:1–13:62.

[27] LOZI, J.-P., LEPERS, B., FUNSTON, J., GAUD, F., QUÉMA, V., AND FEDOROVA, A. The Linux Scheduler: A Decade of Wasted Cores. In *Proceedings of the 11th European Conference on Computer Systems (EuroSys'16)* (2016), ACM.

[28] LUCHANGCO, V., NUSSBAUM, D., AND SHAVIT, N. A Hierarchical CLH Queue Lock. In *Proceedings of the 12th International Conference on Parallel Processing (Euro-Par'06)* (2006), Springer-Verlag.

[29] MAGNUSSON, P. S., LANDIN, A., AND HAGERSTEN, E. Queue Locks on Cache Coherent Multiprocessors. In *Proceedings of the 8th International Symposium on Parallel Processing* (1994), IEEE Computer Society.

[30] MELLOR-CRUMMEY, J. M., AND SCOTT, M. L. Algorithms for Scalable Synchronization on Shared-memory Multiprocessors. *ACM Transactions on Computer Systems 9*, 1 (Feb. 1991), 21–65.

[31] OYAMA, Y., TAURA, K., AND YONEZAWA, A. Executing Parallel Programs with Synchronization Bottlenecks Efficiently. In *Proceedings of the International Workshop on Parallel and Distributed Computing For Symbolic And Irregular Applications (PDSIA'99)* (1999), World Scientific.

[32] RADOVIC, Z., AND HAGERSTEN, E. Hierarchical Backoff Locks for Nonuniform Communication Architectures. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA'03)* (2003), IEEE Computer Society.

[33] SCOTT, M. L. *Shared-Memory Synchronization*. Morgan & Claypool Publishers, 2013.

[34] SCOTT, M. L., AND SCHERER, W. N. Scalable Queue-based Spin Locks with Timeout. In *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming (PPoPP'01)* (2001), ACM.

[35] TALLENT, N. R., MELLOR-CRUMMEY, J. M., AND PORTERFIELD, A. Analyzing Lock Contention in Multithreaded Applications. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'10)* (2010), ACM.

[36] WANG, T., CHABBI, M., AND KIMURA, H. Be My Guest — MCS Lock Now Welcomes Guests. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'16)* (2016), ACM.

A Selection of the lock-sensitive applications

	Gain 1 node	R.Dev. 1 node	Gain max nodes	R.Dev. max nodes	Gain opt nodes	R.Dev. opt nodes
barnes	7%	2%	18%	5%	18%	5%
blackscholes	5%	1%	5%	1%	5%	1%
bodytrack	2%	1%	26%	6%	19%	4%
canneal	7%	1%	9%	2%	8%	2%
dedup	**155%**	**37%**	**224%**	**45%**	**155%**	**37%**
ferret	**1%**	**0%**	**478%**	**72%**	**137%**	**30%**
fmm	**16%**	**4%**	**53%**	**13%**	**48%**	**13%**
freqmine	12%	2%	5%	1%	5%	1%
histogram	**22%**	**5%**	**55%**	**11%**	**46%**	**9%**
kmeans	4%	1%	14%	3%	14%	3%
linear_regression	**38%**	**7%**	**216%**	**22%**	**109%**	**15%**
lu_cb	4%	1%	3%	1%	3%	1%
lu_ncb	19%	4%	37%	9%	33%	8%
matrix_multiply	6%	2%	25%	5%	12%	3%
mysqld	**57%**	**17%**	**54%**	**16%**	**53%**	**16%**
pca	**41%**	**6%**	**239%**	**28%**	**122%**	**15%**
pca_ll	**272%**	**16%**	**761%**	**47%**	**786%**	**34%**
p_raytrace	3%	0%	3%	0%	3%	0%
radiosity	**35%**	**9%**	**828%**	**34%**	**42%**	**10%**
radiosity_ll	**53%**	**7%**	**3064%**	**74%**	**349%**	**32%**
s_raytrace	**9%**	**2%**	**1543%**	**59%**	**344%**	**31%**
s_raytrace_ll	**6%**	**1%**	**3189%**	**72%**	**382%**	**40%**
ssl_proxy	**821%**	**30%**	**1309%**	**64%**	**1241%**	**34%**
streamcluster	**1342%**	**55%**	**1986%**	**50%**	**955%**	**48%**
streamcluster_ll	**17%**	**3%**	**4185%**	**76%**	**92%**	**18%**
string_match	6%	1%	18%	5%	18%	5%
swaptions	1%	0%	6%	1%	6%	1%
vips	**2%**	**0%**	**1616%**	**50%**	**17%**	**6%**
volrend	**9%**	**2%**	**175%**	**26%**	**30%**	**6%**
water_nsquared	**10%**	**2%**	**79%**	**12%**	**79%**	**12%**
water_spatial	**18%**	**4%**	**70%**	**12%**	**70%**	**12%**
word_count	7%	2%	35%	9%	25%	6%
x264	3%	1%	5%	1%	4%	1%

Table 15: For each application, performance gain of the best vs. worst lock and relative standard deviation (**A-48 machine**).

	Gain 1 node	R.Dev. 1 node	Gain max nodes	R.Dev. max nodes	Gain opt nodes	R.Dev. opt nodes
barnes	4%	1%	16%	4%	16%	4%
blackscholes	0%	0%	1%	0%	1%	0%
bodytrack	3%	1%	5%	1%	5%	1%
canneal	1%	0%	1%	0%	1%	0%
dedup	**612%**	**56%**	**879%**	**60%**	**612%**	**56%**
ferret	**0%**	**0%**	**700%**	**81%**	**58%**	**19%**
fmm	6%	1%	19%	5%	19%	4%
freqmine	20%	4%	2%	0%	2%	0%
histogram	16%	4%	22%	6%	16%	4%
kmeans	6%	2%	41%	9%	41%	9%
linear_regression	**10%**	**3%**	**111%**	**20%**	**90%**	**15%**
lu_cb	0%	0%	2%	1%	2%	1%
lu_ncb	7%	2%	31%	7%	31%	7%
matrix_multiply	3%	1%	8%	2%	8%	2%
mysqld	**166%**	**33%**	**132%**	**25%**	**166%**	**32%**
pca	**265%**	**20%**	**282%**	**33%**	**265%**	**20%**
pca_ll	**615%**	**26%**	**1101%**	**53%**	**1021%**	**35%**
p_raytrace	3%	1%	5%	1%	2%	1%
radiosity	**82%**	**9%**	**160%**	**25%**	**91%**	**10%**
radiosity_ll	**989%**	**33%**	**2240%**	**72%**	**1950%**	**53%**
s_raytrace	**3%**	**1%**	**1373%**	**57%**	**203%**	**34%**
s_raytrace_ll	**8%**	**1%**	**2387%**	**69%**	**238%**	**41%**
ssl_proxy	**1543%**	**43%**	**1659%**	**59%**	**1610%**	**49%**
streamcluster	**44%**	**11%**	**634%**	**69%**	**44%**	**11%**
streamcluster_ll	**63%**	**14%**	**677%**	**71%**	**162%**	**34%**
string_match	1%	0%	19%	4%	19%	4%
swaptions	0%	0%	3%	1%	3%	1%
vips	**1%**	**0%**	**848%**	**52%**	**27%**	**9%**
volrend	**8%**	**2%**	**44%**	**10%**	**23%**	**7%**
water_nsquared	**13%**	**3%**	**93%**	**14%**	**93%**	**14%**
water_spatial	**24%**	**5%**	**98%**	**16%**	**92%**	**16%**
word_count	3%	1%	11%	2%	3%	1%
x264	1%	0%	3%	0%	2%	0%

Table 16: For each application, performance gain of the best vs. worst lock and relative standard deviation (**I-48 machine**).

	Gain 1 node	R.Dev. 1 node	Gain max nodes	R.Dev. max nodes	Gain opt nodes	R.Dev. opt nodes
barnes	3%	1%	23%	5%	23%	5%
blackscholes	1%	0%	2%	0%	2%	0%
bodytrack	0%	0%	11%	3%	5%	2%
canneal	2%	0%	4%	1%	4%	1%
dedup	**535%**	**50%**	**968%**	**56%**	**535%**	**53%**
facesim	**1%**	**0%**	**301%**	**24%**	**20%**	**5%**
ferret	**8%**	**3%**	**387%**	**63%**	**356%**	**62%**
fft	8%	2%	10%	2%	10%	2%
fluidanimate	**42%**	**10%**	**305%**	**27%**	**187%**	**23%**
fmm	4%	1%	11%	3%	11%	3%
freqmine	4%	1%	3%	1%	3%	1%
histogram	5%	1%	21%	5%	16%	4%
kmeans	7%	2%	5%	1%	5%	1%
linear_regression	**3%**	**1%**	**96%**	**17%**	**73%**	**13%**
lu_cb	0%	0%	4%	1%	4%	1%
lu_ncb	6%	1%	5%	1%	5%	1%
matrix_multiply	0%	0%	5%	1%	5%	1%
mysqld	**30%**	**9%**	**174%**	**38%**	**122%**	**34%**
ocean_cp	**4%**	**1%**	**131%**	**19%**	**13%**	**4%**
ocean_ncp	**4%**	**1%**	**111%**	**16%**	**9%**	**3%**
pca	**2%**	**0%**	**350%**	**32%**	**62%**	**8%**
pca_ll	**3%**	**1%**	**739%**	**46%**	**159%**	**21%**
p_raytrace	1%	0%	2%	0%	1%	0%
radiosity	**3%**	**1%**	**115%**	**18%**	**7%**	**2%**
radiosity_ll	**10%**	**2%**	**2261%**	**69%**	**267%**	**29%**
radix	0%	0%	15%	3%	15%	3%
s_raytrace	**3%**	**1%**	**1219%**	**59%**	**211%**	**27%**
s_raytrace_ll	**1%**	**0%**	**2894%**	**78%**	**105%**	**26%**
ssl_proxy	**29%**	**5%**	**1256%**	**60%**	**69%**	**14%**
streamcluster	**12%**	**4%**	**728%**	**55%**	**42%**	**10%**
streamcluster_ll	**23%**	**5%**	**860%**	**57%**	**93%**	**23%**
string_match	8%	3%	9%	2%	9%	2%
swaptions	0%	0%	1%	0%	1%	0%
vips	**131%**	**23%**	**327%**	**33%**	**345%**	**37%**
volrend	**5%**	**1%**	**108%**	**16%**	**29%**	**6%**
water_nsquared	**7%**	**2%**	**89%**	**15%**	**89%**	**15%**
water_spatial	**16%**	**4%**	**87%**	**16%**	**87%**	**16%**
word_count	2%	0%	5%	1%	1%	0%
x264	0%	0%	1%	0%	1%	0%

Table 17: For each application, performance gain of the best vs. worst lock and relative standard deviation (**A-64 machine with thread-to-node pinning**).

B Selection of the number of nodes

Applications	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls
dedup	-	-	55	76	79	-	-	-	58	43	72	80	93	83	86	83	84	86	50	48	45	47	47	51	54	47	-
ferret	172	188		236		191	216		150	157	132	165	193		209	192			207					187	149		
fmm																											
histogram	27	38	52	37	52	34	36	48	38	22	39	37	42	42	25	40	46	35	30	41	26	53	39	52	45	42	45
linear_regression	11	52		7	176	60	10	27	12	14	46	9	43	51	80		15		24	12		26	11		10		14
matrix_multiply						12													10								
mysqld	-	-	-	-		-	-	-	-	-	-	-	-		-	-			-			-	-	-	-	-	-
pca	45	32	79	25	206	56	50	70	24	62	14	21	9	41	19	57	85	28	39	54	29	97	53	65	27	72	104
pca_ll	58	79	101	56	366	160		86	41	47	18	29		83	27	27	31	12		91	42	292	87	74	60	96	172
radiosity			42		79		18	593		10		13	12	36			523		17	56	13	172	38	41	19	45	61
radiosity_ll			311		715			438						451		6	605	7	32	245	91	888	253	238	66	301	409
s_raytrace			190		582			118					64	126	21		201		7	77	24	226	26	150	33	188	177
s_raytrace_ll			252	9	807	8	8	332	14			20	9	300			622	24	51	156	62	280	98	153	149	240	271
ssl_proxy	39	49	347	18	683	47	85	104	37	45	42	42	48		45	78	46	52	85	247	114	813	220	369	216	378	508
streamcluster	619	695	851	18		-	-	-		179	495	-	2k	7k	2k	2k	12k	1k	797	71	97	1k	1k	2k	942	1k	1k
streamcluster_ll	113	147	257	13	108	-	-	-	64	110	141	-	459	3k	462	389	3k	351	183	300	389	385	466	403	210	201	368
vips	68	97		842		-	-	-	1k	736	132	-	701		24	52			64					31			
volrend	30	49	43	43	68	36	30	100	41	41	32	31	26	155	49	89	178	87	32	117	115	162	72	41	32	47	69
water_nsquared																											
water_spatial																											

Table 18: For each *(application, lock)* pair, performance gain (in %) of the optimized configuration over the max-node configuration. The background color of a cell indicates the number of nodes (1, 2, 4, 6, or 8 nodes) for the optimized configuration: 1 | 2 | 4 | 6 | 8. Dashes correspond to untested cases. (**A-48 machine**).

Applications	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls
dedup	-	127	72	84	77	136	122	120	77	74	68	84	87	82	78	81	80	87	80	71	72	73	71	71	75	75	125
ferret	420	342		411		347	353		452	384	400	369	399		331	333			349					314	227		
fmm																											
histogram	61	60	62	52	56	59	45	65	65	81	56	57	56	50	56	65	59	47	67	62	59	70	53	75	56	73	56
linear_regression			7									33		32	59	30		7				12		13	19		
matrix_multiply																											
mysqld	-	-	-	-		-	-	-	-	-	-	-	-	10	-	-	17	15	-	82	72	-	-	-	-	-	-
pca	17	22	165	17	117	30	25	19	15	11	28	21	14		24	21	23	13	26	19	18	172	20	47	33	159	153
pca_ll		7	277	62	312	7	9	12	34	43		22		29		10	16		22	14	26	370	11	45	22	261	270
radiosity			65		50			36									136		9			85		10	9	62	58
radiosity_ll			309		472			12	11								28		34			374	11	48	35	320	302
s_raytrace			194		279		11	379					8				395		32		10	346		31	26	192	194
s_raytrace_ll		9	350		260	22	23	643					7	16	6	7	640	15	57	29	19	576	28	60	66	350	290
ssl_proxy	60	52	204	45	1k	48	37	22	44	54	58	34	27	34	52	45	52	54	84	37	25	390	31	34	33	238	313
streamcluster	311	130	949	419	485	-	-	-	524	469	383	-	951	1k	710	882	1k	828	370	1k	1k	1k	1k	1k	1k	877	853
streamcluster_ll	93		164	56	53	-	-	-	128	92	137	-	154	196	109	143	173	118	77	268	165	168	406	210	178	206	180
vips	84	93		591		-	-	-	652	356	182	-	316		99	69			80					76			
volrend			16					7		9						6			8		11	18	9	7	12	14	12
water_nsquared																											
water_spatial																											

Table 19: For each *(application, lock)* pair, performance gain (in %) of the optimized configuration over the max-node configuration. The background color of a cell indicates the number of nodes (1, 2, 3, or 4 nodes) for the optimized configuration: 1 | 2 | 3 | 4. Dashes correspond to untested cases. (**I-48 machine**).

Applications	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls
dedup	-	176	52	41	49	163	166	174	46	48	26	49	45	48	43	43	54	51	50	60	61	55	57	52	48	54	138
facesim	18	20	80	20	45	20	20	60	19	18	18	18	14	56	19	20	61	22	19	42	59	282	28	44	17	80	164
ferret											10		9														
fluidanimate	-	41				-	-	-				-															36
fmm																											
histogram	45	36	50	42	45	43	45	54	38	40	33	39	43	44	46	46	54	41	42	44	41	50	42	45	40	47	47
linear_regression			6			9		21									22					13		8			13
matrix_multiply																											
mysqld	-	-	-	-		-	-	-	-	-	-	-	-		-	-		25	-			-	-	-	-	-	-
ocean_cp	22	19	62	24	48	19	19	61	18	18	18	27	21	58	24	17	60	33	19	38	56	147	35	29	23	62	104
ocean_ncp	20	10	37	15	32	6	14	44	9	14	14	15	13	38	11	11	41	17	12	27	36	114	23	23	13	44	80
pca	29	23	144	28	67	22	28	140	28	29	26	25	25		21	27	143	23	28	88	26	279	53	65		150	134
pca_ll	33	23	147	35	151	30	20	193	33	24	32		34		40	27	197	28	12	92	16	408	65	74	19	155	233
radiosity			37		8			66									68			25	15	106	22	28	10	37	60
radiosity_ll			420		288		8	485	16	21					9	6	538	14	34	157	114	990	169	301	87	428	626
s_raytrace			298		178			323									333			83	29	411	39	192	28	312	274
s_raytrace_ll	7	77	613	20	579	87	74	1k	64	64	7	113	36	137	86	30	1k	92	152	328	153	1k	296	431	216	660	853
ssl_proxy	67	68	683	64	465	71	79	1k	42	40	55	58	52	57	86	73	1k	72	95	278	199	1k	272	336	154	721	933
streamcluster	1k	989	2k	1k	1k	-	-	-	974	998	813	-	3k	6k	2k	2k	5k	2k	1k	2k	3k	5k	2k	3k	2k	2k	4k
streamcluster_ll	203	179	562	275	392	-	-	-	198	234	197	-	466	2k	378	371	1k	233	229	445	492	1k	568	879	626	581	958
vips	23		22		22	-	-	-	15			-		19		24	20	20		21	21	18	22		17	22	23
volrend	14	10	38	6	18	16	11	20	7	7	8	9	14	19	12	13	19	15	12	27	34	74	27	30	19	39	69
water_nsquared																											
water_spatial																											

Table 20: For each *(application, lock)* pair, performance gain (in %) of the optimized configuration over the max-node configuration. The background color of a cell indicates the number of nodes (1, 2, 4, 6, or 8 nodes) for the optimized configuration: 1 | 2 | 4 | 6 | 8. Dashes correspond to untested cases. (**A-64 machine with thread-to-node pinning**).

C Are some locks always among the best?

Locks	Number of nodes 1	Max	Opt
ahmcs	72%	33%	39%
alock-ls	72%	11%	28%
backoff	74%	11%	16%
cbomcs_spin	79%	32%	26%
cbomcs_stp	65%	10%	15%
clh-ls	73%	7%	27%
clh_spin	40%	13%	7%
clh_stp	27%	7%	7%
c-ptl-tkt	58%	11%	26%
c-tkt-tkt	74%	16%	16%
hmcs	79%	42%	47%
hticket-ls	69%	25%	12%
malth_spin	68%	16%	0%
malth_stp	30%	10%	10%
mcs-ls	79%	16%	21%
mcs_spin	63%	26%	37%
mcs_stp	50%	10%	10%
mcs-timepub	60%	25%	30%
partitioned	68%	0%	5%
pthread	55%	30%	30%
pthreadadapt	80%	40%	30%
spinlock	68%	21%	26%
spinlock-ls	74%	26%	37%
ticket	68%	0%	5%
ticket-ls	79%	11%	26%
ttas	74%	21%	26%
ttas-ls	83%	0%	0%

Table 21: For each lock, fraction of the lock-sensitive applications for which the lock yields the best performance for three configurations: 1 node, max nodes, and opt nodes (**A-48 machine**).

Locks	Number of nodes 1	Max	Opt
ahmcs	72%	50%	50%
alock-ls	74%	21%	32%
backoff	47%	26%	26%
cbomcs_spin	74%	37%	47%
cbomcs_stp	70%	25%	25%
clh-ls	56%	12%	25%
clh_spin	56%	12%	6%
clh_stp	44%	12%	12%
c-ptl-tkt	74%	32%	42%
c-tkt-tkt	63%	21%	32%
hmcs	89%	32%	47%
hticket-ls	88%	19%	31%
malth_spin	68%	11%	11%
malth_stp	50%	40%	30%
mcs-ls	74%	16%	26%
mcs_spin	74%	11%	21%
mcs_stp	50%	20%	20%
mcs-timepub	60%	15%	15%
partitioned	68%	16%	26%
pthread	55%	35%	45%
pthreadadapt	60%	35%	40%
spinlock	47%	21%	21%
spinlock-ls	53%	42%	37%
ticket	58%	21%	21%
ticket-ls	58%	26%	21%
ttas	47%	26%	32%
ttas-ls	47%	5%	5%

Table 22: For each lock, fraction of the lock-sensitive applications for which the lock yields the best performance for three configurations: 1 node, max nodes, and opt nodes (**I-48 machine**).

Locks	Number of nodes 1	Max	Opt
ahmcs	71%	33%	48%
alock-ls	74%	22%	26%
backoff	96%	35%	48%
cbomcs_spin	70%	43%	52%
cbomcs_stp	78%	35%	57%
clh-ls	79%	11%	26%
clh_spin	84%	37%	47%
clh_stp	79%	11%	16%
c-ptl-tkt	70%	39%	39%
c-tkt-tkt	70%	48%	48%
hmcs	70%	70%	52%
hticket-ls	89%	53%	53%
malth_spin	74%	43%	48%
malth_stp	78%	43%	43%
mcs-ls	74%	35%	43%
mcs_spin	74%	43%	57%
mcs_stp	83%	26%	30%
mcs-timepub	78%	30%	52%
partitioned	78%	35%	43%
pthread	87%	30%	39%
pthreadadapt	87%	30%	39%
spinlock	83%	22%	22%
spinlock-ls	91%	35%	61%
ticket	83%	22%	39%
ticket-ls	87%	35%	39%
ttas	96%	35%	52%
ttas-ls	87%	4%	13%

Table 23: For each lock, fraction of the lock-sensitive applications for which the lock yields the best performance for three configurations: 1 node, max nodes, and opt nodes (**A-64 machine with thread-to-node pinning**).

D Is there a clear hierarchy between locks?

	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls	average
ahmcs		38	52	29	43	44	33	56	33	38	19	28	43	33	48	52	62	24	38	48	38	67	43	57	43	52	76	44
alock-ls	33		48	9	35	32	16	47	9	22	17	16	17	35	39	30	57	13	26	43	35	52	43	48	35	43	61	33
backoff	33	48		26	57	47	42	63	30	26	39	32	43	26	48	39	61	17	30	17	13	65	17	35	26	22	83	38
cbomcs_spin	57	78	57		57	68	79	74	35	39	52	53	48	48	57	48	65	22	39	48	39	61	52	65	48	52	91	55
cbomcs_stp	43	52	30	26		53	47	47	26	26	35	21	35	33	35	35	42	4	26	29	25	43	17	30	26	30	65	34
clh-ls	22	37	42	11	42		26	47	11	16	21	21	11	26	26	32	53	11	32	42	32	58	47	53	32	42	63	33
clh_spin	17	32	37	5	42	42		47	11	26	16	16	5	21	37	26	58	16	32	42	32	58	47	42	32	37	68	32
clh_stp	39	37	11	16	32	37	21		21	16	21	21	16	5	21	21	21	0	16	11	5	58	11	16	21	11	37	21
c-ptl-tkt	57	78	57	43	52	63	79	74		43	43	32	48	43	57	57	70	35	48	52	39	61	52	65	43	61	87	55
c-tkt-tkt	48	61	57	22	61	47	63	74	35		43	42	48	52	57	48	61	22	30	52	43	65	57	61	57	57	87	52
hmcs	43	61	57	30	43	53	63	74	22	39		32	35	30	35	48	61	17	39	43	35	61	39	57	35	57	83	46
hticket-ls	56	53	63	26	42	63	68	74	21	32	37		37	42	42	53	63	26	53	47	37	63	42	68	42	63	84	50
malth_spin	43	57	48	22	43	63	68	79	22	30	43	26		35	35	43	65	17	52	43	35	61	43	61	35	43	87	46
malth_stp	48	52	52	35	42	53	63	79	26	30	43	32	22		35	43	50	8	39	54	38	57	43	52	52	52	91	46
mcs-ls	33	48	39	22	43	42	42	74	9	22	26	11	22	35		17	57	4	26	39	30	57	35	48	22	39	74	35
mcs_spin	24	57	48	13	48	47	37	68	22	22	39	37	35	39	52		57	17	22	39	35	57	43	48	35	48	83	41
mcs_stp	33	43	13	17	42	37	37	32	22	17	30	26	17	12	26	22		4	22	17	12	39	13	22	22	13	52	25
mcs-timepub	71	74	70	48	71	74	74	84	39	57	61	58	65	58	61	57	71		61	62	54	70	65	70	57	70	100	65
partitioned	43	48	43	26	61	42	37	74	26	17	48	26	30	35	48	39	74	22		48	35	61	48	43	39	43	87	44
pthread	48	52	52	35	46	53	53	63	35	30	57	32	43	25	52	48	58	17	35		21	65	9	48	35	48	91	44
pthreadadapt	52	52	57	39	46	58	58	74	35	35	61	37	43	29	57	52	71	17	26	38		70	35	61	39	57	96	50
spinlock	29	48	4	26	43	42	37	32	35	17	39	32	35	30	39	30	35	17	26	9	9		17	17	22	4	35	27
spinlock-ls	48	48	61	35	65	47	47	79	30	35	52	32	43	30	48	48	74	17	35	65	30	70		65	30	61	96	50
ticket	29	39	30	22	57	37	32	63	26	17	39	26	22	26	39	30	65	13	17	22	9	61	17		13	30	78	33
ticket-ls	48	61	48	39	61	58	63	74	35	35	57	26	52	35	48	52	74	22	43	43	30	65	35	65		52	87	50
ttas	33	48	4	26	52	47	42	58	30	22	43	32	39	26	43	39	65	13	26	13	4	65	13	35	17		78	35
ttas-ls	24	30	9	9	35	32	11	37	9	9	9	5	13	9	13	13	39	0	13	4	4	61	0	13	4	9		16
average	40	51	42	25	48	49	48	63	25	28	38	29	33	32	42	39	59	15	33	37	28	60	34	48	33	42	78	

Table 24: For each pair of locks *(rowA, colB)* at the maximum number of nodes, score of lock A vs lock B: percentage of applications for which lock A performs at least 5% better than B (**A-64 machine**).

	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls	average
ahmcs		6	50	28	50	13	40	60	33	33	17	27	50	50	11	33	50	50	44	50	61	50	50	56	33	50	44	40
alock-ls	22		39	33	56	33	33	67	39	44	17	40	44	50	28	33	50	44	44	56	61	50	50	50	28	44	39	42
backoff	28	28		26	37	33	33	73	26	26	21	19	26	42	26	26	63	16	26	26	26	37	11	32	11	5	22	29
cbomcs_spin	33	39	58		53	40	53	80	21	26	26	25	53	58	21	47	63	47	47	63	63	58	53	63	26	47	67	47
cbomcs_stp	28	28	16	11		33	33	67	11	11	11	6	16	40	16	42	60	40	21	25	15	37	0	26	11	21	22	25
clh-ls	27	0	60	27	60		33	67	20	33	20	40	53	60	27	40	60	53	53	60	60	60	60	67	33	60	53	46
clh_spin	27	20	40	27	47	33		67	27	33	13	27	40	60	13	20	60	40	27	60	47	53	47	47	27	40	40	38
clh_stp	20	13	7	7	0	33	7		13	7	7	7	13	13	7	7	53	7	13	0	0	13	0	7	7	7	7	11
c-ptl-tkt	28	22	47	32	53	20	40	80		16	21	31	53	68	32	47	63	58	32	58	58	58	47	47	32	53	61	44
c-tkt-tkt	28	17	47	21	58	27	33	80	21		26	12	53	63	26	42	63	47	42	58	58	58	47	47	26	42	56	42
hmcs	33	33	58	32	53	47	47	80	37	42		31	58	63	26	42	63	58	53	63	58	63	47	63	32	53	67	50
hticket-ls	20	20	56	25	56	27	40	80	12	19	31		44	62	19	50	69	50	44	56	62	62	56	56	19	62	73	45
malth_spin	28	22	37	16	53	27	33	87	16	16	21	19		53	16	32	58	32	16	58	58	47	42	32	21	37	56	36
malth_stp	28	28	5	16	15	33	33	47	21	21	16	19	16		16	26	40	20	21	5	5	21	0	21	16	21	17	20
mcs-ls	33	28	47	26	58	27	40	80	21	26	16	25	63	58		42	58	47	32	53	53	53	47	58	26	42	50	43
mcs_spin	28	22	32	37	42	47	27	80	42	42	21	44	42	53	26		53	16	32	47	42	37	32	47	37	26	44	38
mcs_stp	33	33	16	32	30	40	27	27	26	26	26	19	21	20	26	16		5	21	15	20	5	11	21	32	11	28	23
mcs-timepub	33	33	26	37	40	40	33	80	37	37	21	38	47	45	32	11	50		26	45	40	37	32	47	37	26	50	38
partitioned	28	33	47	37	58	27	27	80	26	32	26	50	37	63	21	37	58	42		58	47	53	37	53	32	47	50	42
pthread	28	33	21	21	35	33	40	80	26	26	26	38	26	60	26	32	70	25	32		10	32	0	42	21	26	39	33
pthreadadapt	28	33	26	16	40	33	40	80	26	16	26	31	26	60	21	42	70	35	32	25		42	0	37	21	32	39	34
spinlock	33	33	16	37	47	40	33	67	42	32	32	31	37	63	32	21	63	11	32	42	26		21	37	32	11	28	35
spinlock-ls	33	39	37	26	37	33	40	67	37	32	32	19	37	68	32	47	68	42	37	42	26	58		53	26	47	50	41
ticket	28	22	5	21	32	27	20	80	16	21	21	19	11	47	16	16	58	16	11	37	11	32	5		11	5	17	23
ticket-ls	22	22	37	16	58	33	47	80	21	26	26	12	42	63	16	42	58	37	47	53	53	53	47	47		42	50	40
ttas	22	22	5	26	42	27	20	80	26	26	21	19	32	47	21	26	58	16	26	32	21	37	16	32	21		17	28
ttas-ls	22	17	22	17	44	33	20	67	22	22	11	7	11	56	11	33	50	22	22	39	17	33	11	50	17	28		27
average	28	25	33	25	44	32	34	72	26	27	21	25	37	53	22	33	59	34	32	43	38	44	30	44	24	34	42	

Table 25: For each pair of locks *(rowA, colB)* at the optimized number of nodes, score of lock A vs lock B: percentage of applications for which lock A performs at least 5% better than B (**A-48 machine**).

	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls	average
ahmcs		44	67	33	56	47	47	67	39	39	11	27	56	67	44	50	67	50	56	67	50	67	67	72	56	67	67	53
alock-ls	28		56	28	56	47	33	67	33	44	11	27	50	61	33	44	67	50	50	56	50	67	56	56	39	56	56	47
backoff	28	33		21	63	40	27	80	21	11	21	19	37	58	37	42	79	32	21	26	16	68	11	37	16	11	61	35
cbomcs_spin	39	56	63		58	53	53	80	32	37	26	25	58	63	32	53	74	63	42	63	53	68	63	68	53	63	78	54
cbomcs_stp	33	39	16	16		33	27	40	16	16	21	6	26	25	21	42	60	40	26	15	15	47	11	32	16	16	28	26
clh-ls	20	7	60	20	67		33	67	20	33	7	27	40	67	27	27	67	33	27	53	40	67	53	60	33	60	60	41
clh_spin	20	20	60	20	60	47		67	47	33	13	27	40	60	27	33	67	40	40	60	47	67	53	67	40	60	60	45
clh_stp	20	13	7	7	33	33	13		13	7	7	7	13	20	7	7	60	7	13	0	0	27	0	7	7	7	7	13
c-ptl-tkt	33	50	63	37	63	47	53	80		37	21	25	58	74	42	53	74	63	47	63	53	68	53	58	42	63	78	54
c-tkt-tkt	22	44	58	32	58	40	40	80	16		21	19	58	74	32	53	74	53	53	63	47	68	63	63	47	63	78	51
hmcs	33	56	58	37	58	60	60	80	42	47		19	53	63	37	58	74	58	53	58	53	68	58	63	47	58	72	55
hticket-ls	33	40	62	25	62	40	47	80	25	31	12		44	69	25	50	75	50	50	56	50	69	56	62	44	62	80	50
malth_spin	28	39	53	21	53	47	40	87	26	32	11	19		63	26	37	74	32	32	47	47	58	42	53	37	47	61	43
malth_stp	28	28	11	11	40	33	27	47	16	11	11	19	16		16	26	70	25	16	5	0	32	0	26	16	16	22	22
mcs-ls	39	39	42	16	53	40	47	80	21	32	16	12	58	68		47	74	47	37	47	47	58	47	53	37	47	61	45
mcs_spin	33	33	42	37	53	67	27	80	42	42	26	44	53	63	37		63	11	42	47	42	53	37	47	37	42	56	44
mcs_stp	28	28	11	21	30	33	27	27	21	16	21	19	21	5	21	16		5	21	5	5	5	5	16	21	11	22	18
mcs-timepub	33	39	42	26	50	60	47	80	32	32	26	31	53	60	47	37	60		47	50	50	58	42	68	42	42	61	47
partitioned	28	39	58	32	63	33	33	87	21	21	26	25	37	68	42	42	74	37		58	53	68	58	63	47	58	72	48
pthread	28	39	32	21	65	40	40	87	26	21	32	38	42	70	37	37	85	30	26		10	74	0	58	21	32	72	41
pthreadadapt	39	39	58	16	60	53	47	87	32	21	32	31	42	80	26	37	85	35	42	45		68	32	68	37	63	83	48
spinlock	28	28	11	26	47	33	27	53	32	21	26	25	42	53	37	26	74	11	26	16	21		11	37	21	5	22	29
spinlock-ls	28	33	58	16	58	40	33	80	26	16	32	19	47	68	37	53	79	42	37	58	21	79		68	32	58	78	46
ticket	22	22	32	16	58	27	20	80	21	11	21	19	26	63	16	26	74	16	16	21	11	63	16		5	26	56	30
ticket-ls	28	33	58	16	58	47	40	80	21	21	26	12	53	68	37	58	74	47	42	58	37	74	42	74		53	78	47
ttas	22	28	5	21	63	33	27	80	21	16	21	19	42	58	37	42	74	32	21	32	16	74	16	37	21		56	35
ttas-ls	22	22	11	11	50	40	13	67	17	11	17	7	33	50	28	33	67	17	22	11	17	61	6	33	11	17		27
average	29	34	42	22	55	43	36	73	26	25	20	22	42	59	31	40	72	36	35	42	33	61	34	52	32	42	59	

Table 26: For each pair of locks *(rowA, colB)* at the maximum number of nodes, score of lock A vs lock B: percentage of applications for which lock A performs at least 5% better than B (**A-48 machine**).

	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls	average
ahmcs		33	67	28	50	40	53	67	28	39	11	47	61	67	50	56	67	56	39	56	56	67	56	72	61	67	67	52
alock-ls	22		58	11	32	50	38	62	11	11	5	12	42	53	11	21	58	32	32	53	47	63	47	58	37	58	47	37
backoff	28	32		21	16	31	31	56	21	16	32	19	32	16	26	26	42	21	21	11	16	16	5	21	21	5	16	23
cbomcs_spin	33	53	58		37	50	56	75	21	21	5	31	53	47	37	32	63	37	37	53	53	58	53	63	47	58	68	46
cbomcs_stp	39	37	63	16		31	38	75	21	16	26	6	32	30	32	26	50	30	32	35	35	58	37	68	32	58	63	38
clh-ls	13	6	62	12	25		12	62	0	6	0	12	31	56	6	12	56	19	25	50	44	62	50	62	44	56	50	32
clh_spin	20	25	56	6	31	25		56	0	12	6	12	38	44	6	12	56	6	19	56	50	56	50	62	38	56	50	33
clh_stp	27	25	6	6	0	31	6		6	6	6	6	6	0	6	6	6	6	6	0	0	6	0	6	6	6	6	8
c-ptl-tkt	17	42	63	26	53	56	69	81		16	11	44	63	74	53	58	68	58	37	53	63	63	58	68	63	63	74	54
c-tkt-tkt	39	47	63	16	47	62	69	81	16		16	31	58	58	47	42	74	47	37	53	53	63	47	63	53	58	74	51
hmcs	33	53	63	26	42	56	62	81	26	37		38	53	63	42	42	63	58	42	53	53	63	58	68	53	58	74	52
hticket-ls	27	50	62	6	44	50	50	81	0	6	6		50	50	31	12	62	25	25	50	50	62	50	62	44	56	69	42
malth_spin	17	21	53	5	16	25	31	81	0	0	11	6		47	11	11	42	26	11	42	47	53	37	58	47	47	58	31
malth_stp	28	37	42	16	10	31	31	69	16	16	26	6	16		26	16	40	20	16	15	25	42	16	63	21	47	47	28
mcs-ls	17	21	53	0	32	56	44	81	0	5	11	12	37	47		5	47	37	16	42	53	53	42	58	42	47	58	35
mcs_spin	17	21	53	5	26	50	50	81	0	5	11	12	37	42	11		47	32	21	42	53	53	47	68	47	47	63	36
mcs_stp	28	32	16	16	5	31	31	44	16	16	26	12	26	5	21	16		25	16	10	15	16	0	32	21	16	37	20
mcs-timepub	28	37	42	11	25	44	44	75	16	11	21	12	32	30	16	11	45		32	40	40	47	42	53	37	47	58	34
partitioned	11	21	58	11	32	19	31	81	0	5	16	12	47	53	21	21	68	26		53	53	63	47	63	42	63	74	38
pthread	28	37	37	21	25	31	38	81	21	21	32	19	32	40	37	37	65	45	26		20	53	0	47	32	26	42	34
pthreadadapt	28	37	47	26	25	31	38	81	21	21	32	19	26	20	32	21	55	40	26	20		47	11	53	26	37	68	34
spinlock	28	32	5	21	11	31	31	56	21	21	32	12	26	5	21	16	32	21	16	11	16		5	26	16	5	16	20
spinlock-ls	28	37	47	21	21	31	31	75	21	26	32	19	37	32	42	37	63	32	21	42	26	58		58	42	47	63	38
ticket	17	21	32	5	11	25	19	69	5	5	11	6	5	5	5	5	47	11	0	16	5	37	5		0	37	37	17
ticket-ls	22	26	42	11	16	25	19	81	16	21	21	6	16	26	21	11	47	11	16	21	32	47	21	53		47	47	28
ttas	28	32	5	26	16	31	31	62	21	16	32	19	37	16	37	26	42	21	16	11	11	26	5	21	21		16	24
ttas-ls	28	26	26	11	5	31	6	44	11	11	11	6	21	16	16	21	37	11	16	16	21	32	11	37	16	26		20
average	25	32	45	15	25	38	37	71	13	15	17	17	35	36	26	23	52	29	23	35	36	49	31	53	35	44	52	

Table 27: For each pair of locks *(rowA, colB)* at the optimized number of nodes, score of lock A vs lock B: percentage of applications for which lock A performs at least 5% better than B (**I-48 machine**).

	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls	average
ahmcs		39	67	39	61	47	53	67	50	44	28	47	61	61	56	61	72	56	56	61	61	67	61	67	61	67	61	56
alock-ls	33		63	26	53	56	38	62	21	26	21	19	42	47	16	21	63	32	42	53	47	63	53	58	53	63	53	43
backoff	28	32		21	21	31	31	69	21	16	32	19	32	16	26	26	63	16	21	11	11	53	11	21	21	11	16	26
cbomcs_spin	28	42	63		47	62	62	81	37	32	16	31	47	53	37	37	63	42	47	53	53	63	53	63	58	63	74	50
cbomcs_stp	33	37	58	16		31	31	81	26	26	32	12	26	25	32	32	60	35	26	20	20	68	16	42	26	58	63	36
clh-ls	20	6	62	19	44		6	62	6	19	6	19	25	44	6	12	62	12	38	50	44	62	50	62	56	62	56	35
clh_spin	27	25	62	19	50	25		62	6	19	12	6	25	44	6	19	62	12	44	62	50	62	56	62	56	62	62	39
clh_stp	27	25	12	12	0	31	6		12	6	12	6	6	0	12	12	6	6	6	0	6	25	0	6	6	12	12	10
c-ptl-tkt	11	53	63	32	53	62	62	81		26	26	50	47	58	53	63	68	58	47	58	63	63	58	63	58	63	74	54
c-tkt-tkt	28	47	58	21	53	62	62	75	37		21	31	47	58	58	47	68	42	42	53	53	58	53	58	58	58	74	51
hmcs	17	53	63	21	47	56	56	81	42	42		38	53	53	42	47	63	47	42	58	53	63	53	63	58	63	79	52
hticket-ls	33	38	56	19	50	56	56	81	25	12	19		44	44	19	19	62	38	50	50	50	62	50	56	50	56	69	45
malth_spin	17	26	53	21	37	38	31	81	5	16	11	12		53	11	11	47	21	37	53	53	58	53	63	58	53	63	38
malth_stp	28	37	53	21	40	38	38	75	16	21	32	12	16		26	21	45	25	26	30	30	58	16	63	42	53	63	36
mcs-ls	22	21	58	16	42	50	56	75	16	21	16	19	42	47		11	58	37	37	53	53	58	53	58	53	58	68	42
mcs_spin	22	26	47	16	37	50	50	81	11	21	21	19	42	47	11		63	37	42	53	53	63	53	58	53	53	58	42
mcs_stp	28	32	11	16	5	31	31	31	16	16	26	12	16	10	21	16		20	16	15	15	16	11	26	21	11	32	19
mcs-timepub	28	37	53	21	40	50	50	81	16	16	21	19	32	50	21	26	60		58	55	50	58	53	74	63	58	68	44
partitioned	22	26	63	16	47	19	19	81	21	21	32	12	32	47	26	26	68	21		47	32	63	32	63	53	63	74	40
pthread	28	37	63	21	45	31	31	81	21	26	32	25	21	25	26	32	55	35	26		20	68	0	68	47	63	74	39
pthreadadapt	28	37	58	21	50	31	31	81	21	26	32	25	26	15	32	26	55	35	26	25		63	11	68	42	58	74	38
spinlock	28	32	0	21	11	31	31	56	21	26	32	12	21	16	26	21	42	11	21	5	11		5	21	16	0	16	20
spinlock-ls	28	37	58	21	47	31	31	81	21	32	32	25	32	32	32	32	58	26	37	32	21	58		74	53	63	74	41
ticket	22	21	47	16	42	25	19	69	16	16	21	12	16	11	16	11	47	11	5	0	5	47	0		0	47	58	23
ticket-ls	28	26	53	21	42	31	25	75	21	26	32	19	16	21	26	26	53	11	21	16	11	53	5	58		53	58	32
ttas	28	32	5	26	21	31	31	69	21	21	32	19	32	16	32	21	53	16	21	11	11	47	11	21	21		16	25
ttas-ls	28	26	47	16	16	31	6	56	16	16	16	12	21	16	16	32	58	11	16	11	11	63	11	26	16	42		24
average	26	33	50	21	38	40	36	72	21	23	23	21	32	35	26	27	57	27	33	36	34	57	32	52	42	51	57	

Table 28: For each pair of locks *(rowA, colB)* at the maximum number of nodes, score of lock A vs lock B: percentage of applications for which lock A performs at least 5% better than B (**I-48 machine**).

	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls	average
ahmcs		19	38	19	24	22	22	61	24	24	10	22	29	43	24	14	57	24	24	52	48	62	38	33	24	33	52	32
alock-ls	24		35	9	22	21	26	74	13	17	9	11	30	39	9	13	57	22	13	48	43	52	30	26	17	30	52	29
backoff	33	39		26	17	26	26	63	22	22	22	11	26	26	26	26	39	22	26	35	35	43	13	22	17	0	43	27
cbomcs_spin	38	48	43		26	47	26	79	30	17	26	26	30	48	22	17	61	17	26	57	48	61	35	43	39	39	74	39
cbomcs_stp	43	48	17	13		37	26	68	22	9	22	11	17	13	17	17	43	9	13	39	35	48	22	26	26	17	43	27
clh-ls	22	5	32	5	26		5	53	5	16	11	5	21	26	0	0	47	5	11	42	47	47	42	32	21	32	47	23
clh_spin	17	16	37	5	26	21		63	11	11	5	16	21	47	11	0	63	16	16	53	47	58	37	37	32	42	58	29
clh_stp	17	5	0	5	0	11	5		5	5	5	5	5	0	5	5	0	0	5	11	5	26	0	5	5	0	5	5
c-ptl-tkt	29	30	26	13	30	32	26	74		13	13	11	30	48	22	13	61	30	13	52	39	61	35	35	26	26	74	33
c-tkt-tkt	24	39	35	13	30	42	26	79	22		17	11	30	57	26	17	61	30	13	52	43	65	35	35	30	30	70	36
hmcs	29	43	35	17	22	42	47	79	35	22		21	30	43	30	13	57	30	26	48	43	57	30	39	30	35	65	37
hticket-ls	28	32	42	11	32	37	26	79	11	16	5		16	47	5	0	58	16	11	53	42	58	42	32	21	37	74	32
malth_spin	24	35	30	17	30	37	32	79	13	17	22	11		39	0	4	48	4	9	43	35	48	35	26	17	26	65	29
malth_stp	38	43	26	22	17	37	37	63	17	17	30	21	17		22	17	30	13	17	39	35	43	26	30	22	22	57	29
mcs-ls	29	35	30	13	26	42	32	84	22	22	26	16	26	35		9	48	9	17	48	39	48	35	26	22	30	65	32
mcs_spin	38	48	39	26	30	47	32	79	26	30	26	21	26	48	26		57	22	22	52	48	57	35	39	26	35	65	38
mcs_stp	24	30	0	13	4	26	26	32	9	9	17	5	17	4	17	13		4	9	13	13	30	4	9	9	0	26	14
mcs-timepub	29	35	26	17	22	32	32	74	17	17	26	16	26	35	17	13	48		17	48	35	52	35	35	30	26	61	32
partitioned	33	35	30	17	30	26	21	84	22	13	22	5	35	48	17	17	61	30		52	48	65	35	39	30	26	74	35
pthread	24	30	4	22	9	26	26	53	17	17	22	11	22	13	22	22	35	17	13		9	30	0	9	13	0	35	19
pthreadadapt	24	30	4	22	9	26	26	58	17	13	22	11	17	9	22	17	35	13	13	17		35	4	13	9	4	30	19
spinlock	24	26	0	17	9	21	21	37	17	13	22	11	17	9	22	17	26	13	13	17	4		4	9	9	0	17	15
spinlock-ls	33	39	26	26	22	26	26	63	26	22	26	11	26	22	26	26	52	22	26	39	35	52		26	22	22	52	31
ticket	33	26	13	13	17	26	26	74	17	17	22	11	26	26	22	22	57	22	9	39	35	57	13		0	13	35	26
ticket-ls	38	35	26	22	22	32	32	79	22	26	26	11	30	43	26	26	61	26	17	48	48	57	30	35		22	57	34
ttas	33	39	0	22	13	26	26	63	22	17	26	11	26	30	26	26	52	22	22	39	35	48	9	22	17		48	28
ttas-ls	33	22	4	9	9	21	21	47	9	9	9	5	17	9	17	17	26	9	9	30	22	26	9	13	9	9		16
average	29	32	23	16	20	30	26	67	18	17	19	12	24	31	18	15	48	17	16	41	35	49	24	27	20	21	52	

Table 29: For each pair of locks *(rowA, colB)* at the optimized number of nodes, score of lock A vs lock B: percentage of applications for which lock A performs at least 5% better than B (**A-64 machine with thread-to-node pinning**).

	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls	average
ahmcs		19	67	19	52	28	22	72	24	24	5	22	33	48	29	24	71	33	19	67	62	76	67	62	33	67	71	43
alock-ls	43		65	17	48	37	26	84	9	17	13	21	26	43	22	13	65	39	26	61	61	65	65	61	35	65	65	42
backoff	24	30		22	9	26	26	68	17	17	22	11	26	22	26	26	57	13	17	13	17	70	0	22	13	9	70	26
cbomcs_spin	48	43	65		52	58	37	84	26	17	9	26	35	52	35	26	65	30	30	65	61	65	65	70	48	65	83	49
cbomcs_stp	38	35	52	13		32	26	89	13	9	13	5	17	26	17	17	65	9	9	30	30	74	26	39	22	61	83	33
clh-ls	33	0	68	5	47		11	68	11	16	11	16	16	42	5	0	68	21	26	63	58	68	63	63	21	63	68	36
clh_spin	22	11	68	5	53	26		68	11	16	5	11	11	47	16	0	68	37	21	68	58	68	68	68	42	68	68	39
clh_stp	17	5	5	5	0	5	5		5	5	5	5	5	0	5	5	0	0	5	5	5	58	5	11	5	5	26	8
c-ptl-tkt	38	35	65	22	61	47	32	89		13	17	26	30	52	35	17	74	48	30	65	52	65	65	65	39	65	87	48
c-tkt-tkt	38	30	65	26	65	37	32	89	17		13	21	30	57	30	17	74	48	30	65	57	70	65	65	43	65	83	47
hmcs	38	52	65	30	57	58	47	89	35	30		21	35	48	35	26	65	39	35	65	57	65	65	65	43	65	83	51
hticket-ls	33	37	68	11	63	42	26	89	16	11	11		21	47	16	11	68	32	21	68	58	68	68	68	47	68	84	44
malth_spin	29	39	57	22	52	42	37	89	22	17	13	11		48	13	9	70	30	22	57	52	65	57	65	30	57	83	42
malth_stp	38	39	52	22	35	42	37	79	17	17	22	16	17		17	17	43	13	17	43	39	65	43	52	39	48	87	37
mcs-ls	38	26	57	17	57	32	37	89	22	22	26	16	26	48		13	70	26	26	52	61	65	57	61	35	52	83	43
mcs_spin	48	35	57	30	57	53	37	84	26	26	22	21	30	52	35		70	35	17	61	57	65	57	65	43	57	83	47
mcs_stp	24	30	4	13	4	26	26	21	9	9	9	5	9	9	9	9		4	9	4	4	52	4	13	9	4	39	14
mcs-timepub	33	30	65	13	43	37	32	89	17	17	17	16	26	43	22	22	65		26	65	61	70	57	65	43	65	83	43
partitioned	43	22	65	22	61	32	26	89	22	17	26	11	22	48	17	17	74	30		65	57	74	65	65	43	65	83	45
pthread	24	30	52	22	26	26	26	84	17	17	22	11	26	30	26	17	70	13	13		26	74	0	30	17	43	91	32
pthreadadapt	24	30	39	22	35	26	26	79	17	13	22	11	17	22	17	17	65	13	13	35		74	30	52	9	35	91	32
spinlock	24	26	0	17	9	21	21	26	17	13	22	11	17	17	17	17	22	13	13	0	0		0	9	9	0	22	14
spinlock-ls	24	30	61	22	48	26	26	84	17	17	22	11	26	30	26	26	65	13	17	57	30	74		48	22	57	87	37
ticket	24	30	48	13	30	26	26	84	17	17	22	11	17	30	13	17	70	13	13	26	17	70	0		0	48	83	29
ticket-ls	38	39	52	26	48	42	37	89	26	26	30	16	30	30	26	26	70	26	22	52	61	70	52	70		52	87	44
ttas	24	30	0	22	9	26	26	79	17	17	26	11	26	17	26	22	61	13	17	13	17	70	0	22	13		74	26
ttas-ls	24	22	4	9	0	21	21	53	9	9	9	5	9	9	9	9	39	0	9	0	0	61	0	9	4	9		13
average	32	29	49	18	39	34	28	77	18	17	17	14	23	35	21	16	61	23	19	45	41	68	40	49	27	48	75	

Table 30: For each pair of locks *(rowA, colB)* at the maximum number of nodes, score of lock A vs lock B: percentage of applications for which lock A performs at least 5% better than B (**A-64 machine with thread-to-node pinning**).

E Are all locks potentially harmful?

Applications	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls	
dedup	-	-	9	165	166	-	-	-	38	10	188	156	179	175	155	215	210	223	17	3	0	13	0	7	5	5	-	Max nodes
ferret	455	392	10	385	0	375	402	0	478	475	447	454	441	0	395	387	2	7	397	0	0	14	0	395	274	10	12	
fmm	47	46	37	39	39	48	53	40	41	40	40	42	24	34	39	5	0	3	26	33	33	3	33	41	42	45	40	
histogram	0	7	18	0	15	4	14	33	3	0	0	4	12	11	1	30	54	29	14	8	3	46	8	25	6	17	12	
linear_regression	8	45	25	0	216	49	4	76	10	6	27	2	35	75	62	25	109	21	26	21	3	80	14	26	5	27	19	
matrix_multiply	2	6	3	4	3	25	6	5	16	4	2	5	13	2	4	1	5	1	22	0	0	4	4	10	5	3	5	
mysqld	-	-	-	-	29	-	-	-	-	-	-	-	-	13	-	-	9	54	-	1	0	-	-	-	-	-	-	
pca	24	13	74	7	188	31	38	147	6	35	4	0	2	93	6	57	238	28	26	66	18	109	44	62	9	66	89	
pca_ll	21	32	161	23	472	96	29	434	8	15	11	8	0	271	12	65	760	48	10	170	68	422	123	131	36	150	228	
radiosity	29	26	87	26	140	27	34	827	20	33	28	32	29	84	29	0	608	0	31	109	56	179	90	62	49	81	101	
radiosity_ll	0	8	600	28	1k	18	8	2k	19	45	1	37	59	2k	31	10	3k	40	64	658	227	2k	533	502	173	578	753	
s_raytrace	2	0	352	22	2k	7	2	775	12	10	3	15	99	551	29	0	1k	0	13	183	93	393	66	278	83	354	311	
s_raytrace_ll	6	0	809	40	2k	17	15	2k	27	28	4	55	49	2k	14	11	3k	48	72	389	196	871	191	547	227	773	716	
ssl_proxy	1	6	313	0	766	6	52	765	7	18	5	9	43	863	10	53	1k	38	65	297	125	816	237	377	143	318	537	
streamcluster	28	44	71	57	23	-	-	-	23	31	0	-	231	1k	215	247	2k	166	60	189	248	169	133	223	94	97	151	
streamcluster_ll	159	214	356	0	119	-	-	-	70	117	130	-	632	4k	608	523	4k	487	242	410	578	510	308	576	312	288	490	
vips	94	130	2	985	16	-	-	-	2k	877	171	-	819	6	42	76	7	4	88	0	0	3	1	52	19	4	9	
volrend	0	20	18	9	45	5	6	80	12	13	3	2	3	151	16	58	174	59	9	87	86	135	48	17	7	22	40	
water_nsquared	78	43	6	13	12	50	29	29	9	10	13	14	8	9	16	5	6	8	6	8	8	0	9	5	11	4	31	
water_spatial	69	34	5	5	5	45	34	33	2	1	6	5	9	7	4	19	21	19	6	0	1	15	1	5	1	4	28	
dedup	-	-	3	121	118	-	-	-	28	13	146	108	112	120	101	152	148	155	14	2	1	13	0	4	0	5	-	Opt nodes
ferret	105	71	6	45	0	63	59	0	132	125	136	110	85	0	60	67	1	4	63	0	0	9	0	73	50	6	6	
fmm	47	46	37	39	39	48	47	40	41	40	40	42	24	34	39	5	0	3	26	33	33	3	33	38	42	41	40	
histogram	8	7	6	0	4	7	15	24	2	13	0	5	8	7	12	27	46	32	20	6	12	32	6	13	1	13	7	
linear_regression	11	9	37	7	31	6	8	59	12	7	0	8	8	33	3	39	108	38	16	24	17	63	18	37	10	42	20	
matrix_multiply	2	6	3	4	3	11	6	5	11	4	2	5	11	2	4	1	5	1	11	0	0	4	4	10	5	3	5	
mysqld	-	-	-	-	29	-	-	-	-	-	-	-	-	13	-	-	9	52	-	0	0	-	-	-	-	-	-	
pca	3	3	17	4	14	2	12	77	3	1	11	0	14	67	8	21	121	21	11	30	11	29	14	19	4	17	12	
pca_ll	3	0	75	6	65	2	73	287	3	6	27	13	35	173	20	75	785	79	49	91	59	79	61	79	15	72	63	
radiosity	32	29	34	26	37	27	16	36	21	24	28	20	18	38	25	0	16	1	14	37	41	4	41	18	29	28	27	
radiosity_ll	0	6	70	28	85	16	6	238	16	41	1	37	56	248	26	4	348	31	24	119	71	71	79	78	65	68	67	
s_raytrace	2	0	56	19	140	5	2	302	6	10	3	11	21	188	6	0	344	0	5	60	56	51	32	51	37	57	48	
s_raytrace_ll	6	0	158	29	148	8	6	366	12	22	4	28	37	382	10	6	355	19	14	91	82	155	47	156	31	156	119	
ssl_proxy	2	0	28	17	53	0	14	489	9	13	3	7	35	1k	5	19	1k	26	24	59	46	39	46	41	7	21	45	
streamcluster	6	8	7	695	636	-	-	-	635	180	0	-	8	8	8	3	6	8	6	907	954	7	1	10	10	4	7	
streamcluster_ll	68	76	77	22	45	-	-	-	43	43	32	-	81	84	74	76	74	80	67	76	92	74	0	86	84	78	74	
vips	15	17	2	15	15	-	-	-	16	16	17	-	14	6	15	16	7	4	15	0	0	3	1	15	16	4	9	
volrend	1	5	8	0	13	1	7	17	4	5	2	2	7	29	2	10	29	11	8	13	13	18	13	9	6	9	9	
water_nsquared	78	43	6	13	12	50	29	29	9	10	13	14	8	9	16	5	6	8	6	8	8	0	9	5	11	4	31	
water_spatial	69	34	5	5	5	45	34	33	2	1	6	5	9	7	4	19	21	19	6	0	1	15	1	5	1	4	28	

Table 31: For each application, at max nodes (top part) and at the optimized number of nodes (bottom part), performance gain (in %) obtained by the best lock(s) with respect to each of the other locks. The grey background highlights cells for which the performance gains are greater than 15%. A line with many gray cells corresponds to an application whose performance is hurt by many locks. A column with many gray cells corresponds to a lock that is outperformed by many other locks. Dashes correspond to untested cases. (**A-48 machine**).

Applications	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls	
dedup	-	608	1	170	163	879	541	543	15	12	178	163	165	162	173	162	161	178	13	0	1	4	1	2	5	6	542	Max nodes
ferret	657	540	12	642	0	547	556	0	700	660	626	580	688	0	533	528	0	11	553	0	0	14	0	555	387	13	12	
fmm	19	13	6	14	3	17	12	8	15	10	14	8	10	4	11	14	2	7	13	0	0	5	1	9	8	4	8	
histogram	1	4	12	4	3	3	0	15	3	16	0	2	2	3	3	5	15	0	6	5	2	22	3	18	4	15	6	
linear_regression	0	45	62	20	17	13	6	85	9	20	18	58	13	58	110	56	89	32	26	15	15	93	19	44	48	57	49	
matrix_multiply	0	2	1	3	5	2	2	3	2	2	4	4	6	5	3	8	7	3	1	2	3	4	2	4	4	3	2	
mysqld	-	-	-	-	0	-	-	-	-	-	-	-	-	28	-	-	34	132	-	38	30	-	-	-	-	-	-	
pca	3	11	164	8	104	17	18	282	3	4	16	12	1	0	12	13	274	8	20	33	24	173	25	50	27	171	135	
pca_ll	0	28	554	57	526	39	45	1k	30	33	1	32	33	100	20	25	1k	39	67	145	118	639	121	140	90	531	473	
radiosity	7	10	96	0	66	12	11	159	7	0	0	1	6	12	5	4	155	7	15	20	14	120	17	29	19	93	84	
radiosity_ll	3	68	1k	0	1k	94	99	2k	40	19	2	23	133	155	70	65	2k	91	187	274	176	2k	215	393	249	1k	1k	
s_raytrace	0	19	766	21	644	29	36	1k	12	16	9	41	65	112	24	22	1k	17	65	77	117	1k	53	186	122	761	733	
s_raytrace_ll	2	50	1k	27	1k	85	89	2k	29	12	0	53	97	210	48	47	2k	65	157	185	228	2k	140	389	262	1k	1k	
ssl_proxy	9	46	773	0	1k	56	56	1k	7	15	6	17	57	148	50	52	2k	64	88	130	85	1k	97	192	120	760	619	
streamcluster	91	0	422	114	146	-	-	-	187	151	106	-	461	538	321	402	479	411	112	518	625	499	504	634	607	380	374	
streamcluster_ll	144	0	417	74	79	-	-	-	174	118	150	-	477	550	335	415	486	451	120	588	573	473	556	677	627	481	412	
vips	129	140	4	768	13	-	-	-	847	477	246	-	420	3	149	111	7	8	126	1	0	6	1	119	10	4	5	
volrend	3	6	39	2	13	8	10	24	0	7	7	6	11	21	7	8	16	8	11	25	34	44	25	22	20	37	33	
water_nsquared	93	48	3	6	4	57	37	40	0	8	11	6	5	5	4	5	3	11	4	4	5	2	1	4	2	3	33	
water_spatial	98	55	0	5	5	65	41	41	2	0	7	5	6	4	6	4	4	4	2	2	0	1	0	4	1	0	42	
dedup	-	435	0	151	155	611	394	400	11	10	183	145	143	147	163	148	149	156	7	0	1	3	1	2	2	4	389	Opt nodes
ferret	45	44	9	45	0	44	45	0	44	57	45	45	58	0	46	45	0	8	45	0	0	8	0	58	49	7	7	
fmm	18	13	6	11	3	17	12	8	13	10	13	8	10	4	11	10	2	7	13	0	0	5	1	9	8	4	8	
histogram	0	4	10	9	5	3	10	11	0	2	1	3	4	9	5	1	15	8	0	4	2	14	7	7	6	6	8	
linear_regression	0	45	52	20	17	13	6	85	9	18	12	19	13	19	32	20	89	23	24	15	15	73	19	27	24	56	49	
matrix_multiply	0	2	1	3	5	2	2	3	2	2	4	4	6	5	3	8	7	3	1	2	3	4	2	4	4	3	2	
mysqld	-	-	-	-	32	-	-	-	-	-	-	-	-	54	-	-	52	165	-	0	0	-	-	-	-	-	-	
pca	0	3	13	5	6	2	6	264	1	6	3	5	1	8	3	6	243	8	8	26	19	14	18	15	7	18	5	
pca_ll	6	28	86	4	63	39	44	1k	4	0	8	17	43	67	28	22	1k	43	46	130	86	68	114	78	67	87	66	
radiosity	6	6	19	0	11	9	7	91	1	0	0	0	6	9	3	2	7	6	5	15	11	18	12	18	10	19	17	
radiosity_ll	3	64	263	0	138	92	91	2k	27	19	2	23	133	155	66	65	2k	90	113	254	176	270	183	232	159	259	230	
s_raytrace	0	15	194	21	96	23	23	203	12	16	9	34	52	104	20	18	197	17	25	70	97	196	48	119	77	195	183	
s_raytrace_ll	2	38	236	27	236	52	53	222	29	12	0	47	84	167	39	37	236	43	63	120	175	235	88	206	118	235	237	
ssl_proxy	1	42	324	2	57	56	68	1k	10	10	0	30	84	173	46	55	2k	58	51	149	119	373	123	224	145	276	157	
streamcluster	12	5	20	0	2	-	-	-	11	7	3	-	29	28	25	24	21	33	9	15	44	25	7	38	40	19	20	
streamcluster_ll	26	0	95	11	16	-	-	-	20	13	5	-	127	119	108	112	114	152	24	87	154	114	29	150	161	89	82	
vips	24	24	4	25	13	-	-	-	25	26	22	-	25	3	25	25	7	8	25	1	0	6	1	24	10	4	5	
volrend	1	3	21	0	8	4	5	17	0	0	2	0	8	16	3	3	11	4	4	19	22	22	15	14	8	21	20	
water_nsquared	93	48	3	6	4	57	37	40	0	8	11	6	5	5	4	5	3	11	4	4	5	2	1	4	2	3	33	
water_spatial	92	55	0	5	5	65	41	41	2	0	7	5	6	4	6	4	4	4	2	2	0	1	0	4	1	0	42	

Table 32: For each application, at max nodes (top part) and at the optimized number of nodes (bottom part), performance gain (in %) obtained by the best lock(s) with respect to each of the other locks. The grey background highlights cells for which the performance gains are greater than 15%. A line with many gray cells corresponds to an application whose performance is hurt by many locks. A column with many gray cells corresponds to a lock that is outperformed by many other locks. Dashes correspond to untested cases. (**I-48 machine**).

Applications	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls
dedup	-	578	1	134	133	967	954	954	22	11	127	116	116	111	115	112	117	136	11	2	3	3	0	0	0	3	560
facesim	3	5	61	5	32	6	5	56	5	4	4	3	0	47	4	5	55	6	5	30	56	301	12	28	2	62	169
ferret	382	289	5	109	0	308	327	0	322	368	386	347	340	0	254	314	0	1	328	0	0	6	0	232	153	3	5
fluidanimate	-	305	0	50	52	-	-	-	28	14	62	-	37	58	44	33	56	52	7	7	11	21	0	7	5	1	209
fmm	11	5	0	3	0	8	7	7	0	2	2	0	0	0	1	2	0	2	0	0	0	3	0	0	0	0	5
histogram	7	5	11	0	2	6	3	17	2	2	3	3	3	1	5	1	16	1	3	10	6	21	11	9	9	10	16
linear_regression	7	10	42	1	2	20	5	74	13	3	0	1	4	11	5	2	71	6	6	34	16	95	22	35	12	38	61
matrix_multiply	0	0	1	0	0	0	0	3	0	0	0	0	0	0	0	0	3	0	0	1	0	5	0	0	0	1	2
mysqld	-	-	-	-	30	-	-	-	-	-	-	-	-	0	-	-	7	173	-	97	102	-	-	-	-	-	-
ocean_cp	4	1	45	5	30	5	3	52	1	1	3	7	2	44	5	0	49	14	0	26	38	130	18	13	4	40	86
ocean_ncp	9	0	27	4	23	0	2	39	1	2	4	4	2	34	1	0	37	8	1	20	28	111	14	11	2	31	75
pca	50	42	186	49	97	42	49	289	49	51	47	46	44	0	40	47	289	43	49	134	53	349	85	92	21	192	173
pca_ll	69	57	298	77	299	55	73	659	63	64	68	23	45	0	54	53	661	35	55	217	88	739	148	167	40	310	431
radiosity	6	5	38	0	9	6	4	71	1	1	0	2	2	3	3	1	70	3	3	32	18	115	24	29	11	38	63
radiosity_ll	0	44	785	12	571	48	26	2k	37	48	1	24	69	77	55	20	2k	67	83	547	311	2k	405	602	212	796	1k
s_raytrace	4	10	601	18	363	20	11	1k	13	21	2	34	32	70	15	0	1k	17	33	270	144	789	139	403	109	630	530
s_raytrace_ll	1	94	1k	33	1k	110	92	3k	77	74	0	134	63	260	108	22	3k	119	177	703	377	2k	481	850	271	1k	1k
ssl_proxy	2	10	529	0	397	12	12	973	9	12	0	8	14	33	27	13	983	27	29	290	154	1k	246	254	76	554	758
streamcluster	50	28	142	37	28	-	-	-	23	21	0	-	261	728	207	180	566	115	34	170	275	603	113	304	201	147	357
streamcluster_ll	44	14	135	18	64	-	-	-	6	11	0	-	237	860	192	154	739	85	23	174	238	597	119	316	207	147	347
vips	67	34	3	235	3	-	-	-	326	208	88	-	147	0	22	35	1	2	36	1	1	1	2	27	7	3	6
volrend	6	4	37	0	13	11	4	21	0	0	0	2	9	18	6	5	19	10	6	36	48	108	32	28	14	38	78
water_nsquared	89	41	0	5	4	53	54	53	1	1	7	3	3	3	4	3	3	8	0	0	0	1	0	1	0	0	31
water_spatial	87	43	0	4	4	58	57	58	2	1	5	3	2	3	3	3	3	4	0	0	0	0	0	0	0	0	35

Max nodes

Applications	ahmcs	alock-ls	backoff	c-bo-mcs_spin	c-bo-mcs_stp	clh-ls	clh_spin	clh_stp	c-ptl-tkt	c-tkt-tkt	hmcs	hticket-ls	malth_spin	malth_stp	mcs-ls	mcs_spin	mcs_stp	mcs-timepub	partitioned	pthread	pthreadadapt	spinlock	spinlock-ls	ticket	ticket-ls	ttas	ttas-ls
dedup	-	284	3	159	145	534	519	501	30	18	181	127	132	122	135	132	120	144	16	0	0	3	0	3	5	4	333
facesim	0	1	3	0	4	1	1	12	1	1	1	1	0	8	0	1	10	0	1	5	12	20	1	2	0	3	17
ferret	355	289	5	109	0	308	327	0	322	347	342	325	303	0	254	314	0	1	328	0	0	6	0	232	153	3	5
fluidanimate	-	187	0	50	52	-	-	-	28	14	62	-	37	53	44	33	51	52	7	7	11	21	0	7	5	1	127
fmm	11	5	0	3	0	8	7	7	0	2	2	0	0	0	1	2	0	2	0	0	0	3	0	0	0	0	5
histogram	7	11	7	1	1	6	2	9	6	4	11	7	3	1	4	0	8	3	4	9	7	15	12	8	12	8	13
linear_regression	7	10	33	1	2	10	5	44	12	3	0	1	4	10	5	2	40	6	6	34	16	73	22	25	12	38	43
matrix_multiply	0	0	1	0	0	0	0	3	0	0	0	0	0	0	0	0	3	0	0	1	0	5	0	0	0	1	2
mysqld	-	-	-	-	31	-	-	-	-	-	-	-	-	0	-	-	8	121	-	96	96	-	-	-	-	-	-
ocean_cp	1	1	7	0	4	6	3	12	2	1	4	0	0	9	0	1	11	2	0	9	5	10	4	5	0	3	8
ocean_ncp	0	0	2	0	3	5	0	6	3	0	0	0	0	7	1	0	7	2	0	4	4	9	3	0	0	1	7
pca	16	16	17	16	17	16	16	62	17	17	17	17	16	0	16	16	60	16	16	24	22	18	20	16	15	16	16
pca_ll	27	28	61	32	59	19	44	158	23	32	27	23	9	0	10	20	156	5	38	65	62	65	50	53	18	61	59
radiosity	3	3	2	0	2	4	2	4	0	0	0	1	2	3	1	0	2	2	1	6	4	5	3	2	2	2	3
radiosity_ll	0	38	70	12	73	43	17	267	18	23	1	24	61	77	42	13	237	46	36	151	92	116	87	75	66	69	73
s_raytrace	4	10	76	18	66	20	11	211	13	21	2	28	32	70	15	0	205	17	31	102	89	73	72	72	63	77	68
s_raytrace_ll	1	17	101	18	102	19	17	105	15	13	0	17	28	62	19	0	101	21	17	99	101	102	56	90	25	102	74
ssl_proxy	0	8	31	0	44	8	3	48	26	32	6	12	23	39	12	7	37	21	8	69	39	41	52	33	13	30	36
streamcluster	32	19	6	12	12	-	-	-	17	12	11	-	36	35	34	32	31	35	13	41	40	39	0	23	24	7	25
streamcluster_ll	51	29	12	0	6	-	-	-	13	5	6	-	88	86	93	70	69	76	18	59	80	80	4	34	34	15	33
vips	62	57	1	301	0	-	-	-	345	269	125	-	196	0	46	31	0	2	54	0	0	3	0	52	10	1	4
volrend	0	1	7	1	3	2	1	8	0	1	0	1	3	7	2	1	7	4	2	15	18	28	11	6	2	7	13
water_nsquared	89	41	0	5	4	53	54	53	1	1	7	3	3	3	4	3	3	8	0	0	0	1	0	1	0	0	31
water_spatial	87	43	0	4	4	58	57	58	2	1	5	3	2	3	3	3	3	4	0	0	0	0	0	0	0	0	35

Opt nodes

Table 33: For each application, at max nodes (top part) and at the optimized number of nodes (bottom part), performance gain (in %) obtained by the best lock(s) with respect to each of the other locks. The grey background highlights cells for which the performance gains are greater than 15%. A line with many gray cells corresponds to an application whose performance is hurt by many locks. A column with many gray cells corresponds to a lock that is outperformed by many other locks. Dashes correspond to untested cases. (**A-64 machine with thread-to-node pinning**).

F Impact of the number of nodes

Applications	% of pairwise changes between configurations			
	1/2	2/4	4/8	1/2/4/8
dedup	7%	4%	11%	17%
ferret	0%	76%	18%	87%
fmm	23%	21%	37%	51%
histogram	40%	35%	25%	63%
linear_regression	26%	34%	44%	66%
matrix_multiply	33%	38%	47%	68%
mysqld	27%	0%	7%	33%
pca	31%	31%	29%	69%
pca_ll	50%	29%	35%	77%
radiosity	53%	42%	19%	82%
radiosity_ll	29%	48%	9%	77%
s_raytrace	27%	44%	30%	93%
s_raytrace_ll	23%	52%	25%	94%
ssl_proxy	45%	20%	11%	54%
streamcluster	15%	39%	45%	88%
streamcluster_ll	53%	28%	33%	89%
vips	0%	1%	85%	85%
volrend	16%	19%	45%	79%
water_nsquared	28%	32%	23%	63%
water_spatial	15%	18%	10%	33%

Table 34: For each application, percentage of pairwise changes in the lock performance hierarchy when changing the number of nodes (**A-48 machine**).

Applications	% of pairwise changes between configurations			
	1/2	2/3	3/4	1/2/3/4
dedup	4%	5%	5%	10%
ferret	21%	67%	15%	86%
fmm	11%	17%	26%	50%
histogram	35%	21%	25%	49%
linear_regression	36%	31%	39%	80%
matrix_multiply	0%	0%	4%	4%
mysqld	20%	27%	27%	53%
pca	35%	17%	12%	50%
pca_ll	49%	11%	11%	54%
radiosity	29%	11%	11%	44%
radiosity_ll	16%	8%	2%	21%
s_raytrace	74%	13%	11%	95%
s_raytrace_ll	78%	15%	12%	98%
ssl_proxy	15%	6%	8%	21%
streamcluster	14%	14%	22%	35%
streamcluster_ll	14%	16%	26%	38%
vips	0%	0%	81%	81%
volrend	15%	30%	10%	53%
water_nsquared	20%	0%	12%	32%
water_spatial	0%	1%	5%	7%

Table 35: For each application, percentage of pairwise changes in the lock performance hierarchy when changing the number of nodes (**I-48 machine**).

Applications	% of pairwise changes between configurations			
	1/2	2/4	4/8	1/2/4/8
dedup	11%	9%	2%	18%
facesim	0%	37%	35%	70%
ferret	21%	15%	19%	42%
fluidanimate	9%	4%	17%	28%
fmm	6%	12%	8%	26%
histogram	11%	35%	29%	62%
linear_regression	15%	52%	31%	82%
matrix_multiply	0%	0%	0%	0%
mysqld	33%	20%	7%	40%
ocean_cp	18%	45%	42%	79%
ocean_ncp	12%	20%	50%	72%
pca	21%	46%	15%	77%
pca_ll	17%	71%	19%	97%
radiosity	0%	55%	13%	68%
radiosity_ll	40%	50%	12%	92%
s_raytrace	0%	48%	48%	93%
s_raytrace_ll	0%	74%	30%	98%
ssl_proxy	66%	12%	12%	77%
streamcluster	65%	18%	25%	84%
streamcluster_ll	61%	21%	26%	83%
vips	12%	7%	9%	21%
volrend	20%	20%	39%	78%
water_nsquared	22%	9%	9%	42%
water_spatial	2%	1%	0%	4%

Table 36: For each application, percentage of pairwise changes in the lock performance hierarchy when changing the number of nodes (**A-64 machine with thread-to-node pinning**).