Everything you always wanted to know about synchronization but were afraid to ask

by Tudor David Rachid Guerraoui, Vasileios Trigonakis

Michał Niewiadomski

MIMUW

January 13, 2016

Synchronization analysis methods

depth	breadth
concurrent software hash table, Memcached, STM	**single-socket** uniform (Sun Niagara 2) non-uniform (Tilera TILE-Gx36)
primitives locks, message passing	
atomic operations CAS, FAI, TAS, SWAP	**multi-socket** directory-based (AMD Opteron) broadcast-based (Intel Xeon)
cache coherence loads, stores	

Figure 1: Analysis method.

Tested platforms - hardware

Name	Opteron	Xeon	Niagara	Tilera
System	AMD Magny Cours	Intel Westmere-EX	SUN SPARC-T5120	Tilera TILE-Gx36
Processors	4× AMD Opteron 6172	8× Intel Xeon E7-8867L	SUN UltraSPARC-T2	TILE-Gx CPU
# Cores	48	80 (no hyper-threading)	8 (64 hardware threads)	36
Core clock	2.1 GHz	2.13 GHz	1.2 GHz	1.2 GHz
L1 Cache	64/64 KiB I/D	32/32 KiB I/D	16/8 KiB I/D	32/32 KiB I/D
L2 Cache	512 KiB	256 KiB		256 KiB
Last-level Cache	2×6 MiB (shared per die)	30 MiB (shared)	4 MiB (shared)	9 MiB Distributed
Interconnect	6.4 GT/s HyperTransport (HT) 3.0	6.4 GT/s QuickPath Interconnect (QPI)	Niagara2 Crossbar	Tilera iMesh
Memory #Channels / #Nodes	128 GiB DDR3-1333 4 per socket / 8	192 GiB Sync DDR3-1067 4 per socket / 8	32 GiB FB-DIMM-400 8 / 1	16 GiB DDR3-800 4 / 2
OS	Ubuntu 12.04.2 / 3.4.2	Red Hat EL 6.3 / 2.6.32	Solaris 10 u7	Tilera EL 6.3 / 2.6.40

Table 1: The hardware and the OS characteristics of the target platforms.

General observations

1. Crossing sockets is a killer (2-7.5x increase in latency).
2. Sharing within a socket is necessary but not sufficient (problems with incomplete cache).
3. Intra-socket (non-)uniformity (distance from LLC) does matter (approx. 1.7x increase in latency).
4. LL/SC can be as expensive as atomic operations (due to fences).
5. Message passing shines when contention is very high.
6. None of the tested locking schemes we consider consistently outperforms any other one.
7. Simple locks (spin/ticket locks) perform best when contention is low.

Synchronization levels

Hardware based

Cache-coherence (MESI protocol, snooping/directory caching)

Software based

1. spin locks
2. queue locks
3. hierarchical locks
4. pthread mutex
5. message passing

Cache-coherence

MESI protocol

1. Modified: stale data in memory, no other cache has a copy
2. Exclusive: data are up-to-date in memory, no other cache has a copy
3. Shared: data are up-to-date in memory and other caches might have copies
4. Invalid: data are invalid

Coherence implementations

1. snooping: individual caches monitor any traffic on the addresses they hold in order to ensure coherence
2. directory-based: directory keep informations about caches and is responsible for coherence and updating data

Tested platforms

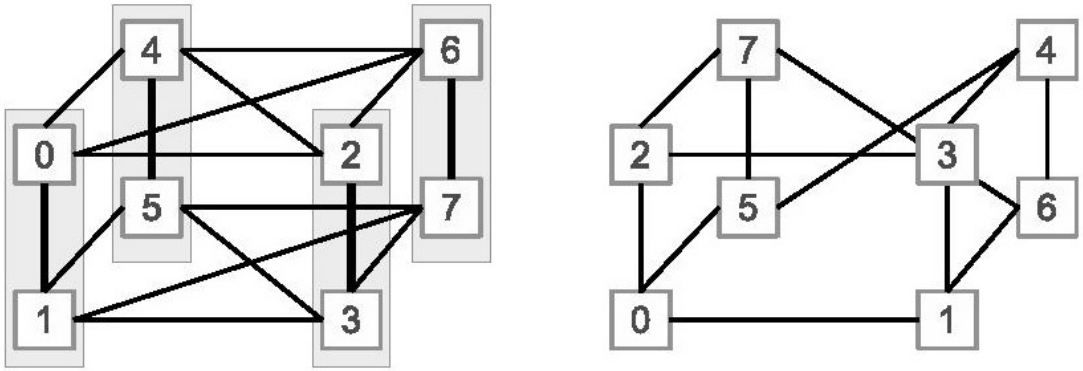
Opteron

48 cores/4 MCM (multi-chip modules). Each MCM has two 6-core dies with independent memory controllers, max 2-hops distance.
MOESI cache protocol (Owned: modified, other caches has copy).
Directory-based cache-coherence.

Xeon

80 cores/8 sockets, max 2-hops distance.
inclusive caches/write-back LLC
Snooping cache-coherence
MFESI cache protocol (Forward: shared, only one cache loads data).

Tested platforms - topologies



(a) AMD Opteron (b) Intel Xeon

Figure 2: The system topologies.

Tested platforms

Niagara

Uniform communication with cache.
2 CPU/8 cores/64 hardware threads.

Tilera

Non-uniform communication with cache.
Implements Dynamic Distributed Cache.
36 cores

Tested platforms - latencies

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

Table 3: Local caches and memory latencies (cycles).

Tested platforms - latencies

System	Opteron				Xeon			Niagara		Tilera	
Hops / State	same die	same MCM	one hop	two hops	same die	one hop	two hops	same core	other core	one hop	max hops
loads											
Modified	81	161	172	252	109	289	400	3	24	45	65
Owned	83	163	175	254	-	-	-	-	-	-	-
Exclusive	83	163	175	253	92	273	383	3	24	45	65
Shared	83	164	176	254	44	223	334	3	24	45	65
Invalid	136	237	247	327	355	492	601	176	176	118	162
stores											
Modified	83	172	191	273	115	320	431	24	24	57	77
Owned	244	255	286	291	-	-	-	-	-	-	-
Exclusive	83	171	191	271	115	315	425	24	24	57	77
Shared	246	255	286	296	116	318	428	24	24	86	106
atomic operations: CAS (C), FAI (F), TAS (T), SWAP (S)											
Operation	all	all	all	all	all	all	all	C/F/T/S	C/F/T/S	C/F/T/S	C/F/T/S
Modified	110	197	216	296	120	324	430	71/108/64/95	66/99/55/90	77/51/70/63	98/71/89/84
Shared	272	283	312	332	113	312	423	76/99/67/93	66/99/55/90	124/82/121/95	142/102/141/115

Load/store testing

1. cross socket synchronization is about 2-7x slower
2. snooping performs better than directory-based caching
3. uniformness matters

Opteron - enhance locality

1. using ticket locking with prefetchw x86 instruction
2. unsafe, but performs better

Opteron - locality

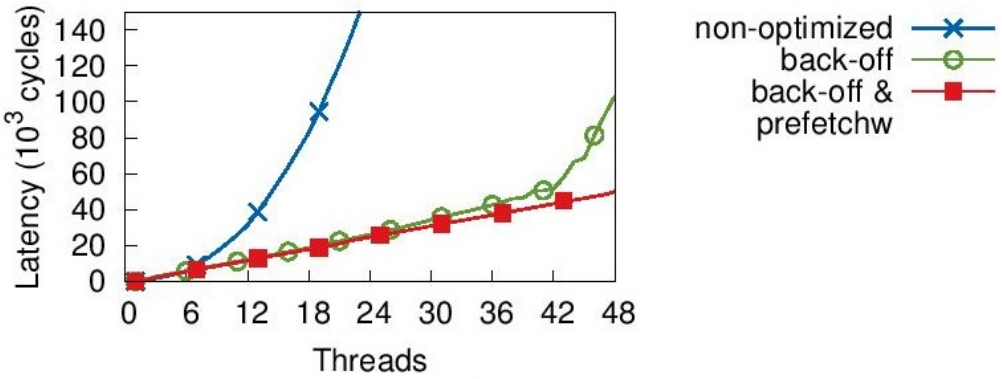


Figure 3: Latency of acquire and release using different implementations of a ticket lock on the Opteron.

Stressing atomic operations

1. each thread tries atomic operation and pause for some number cycles
2. preventing processor optimalizations and instruction pipelining

Atomic operations - performance

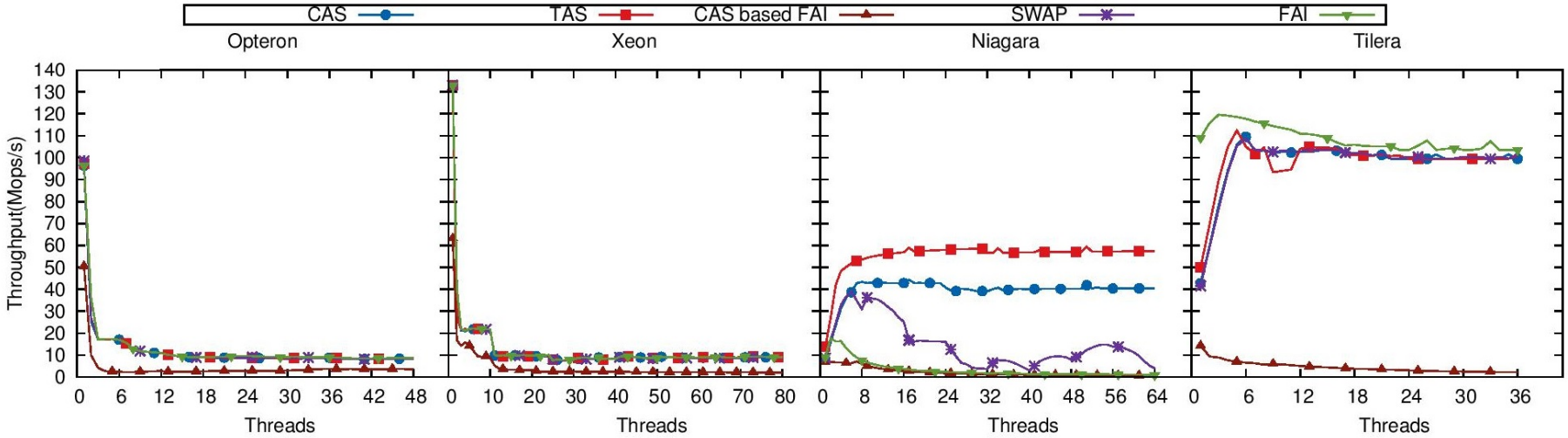


Figure 4: Throughput of different atomic operations on a single memory location.

Uncontested locking

1. measuring lock latency based on distance from previous holder
2. cross-socket communication is bottleneck

Uncontested locking - performance

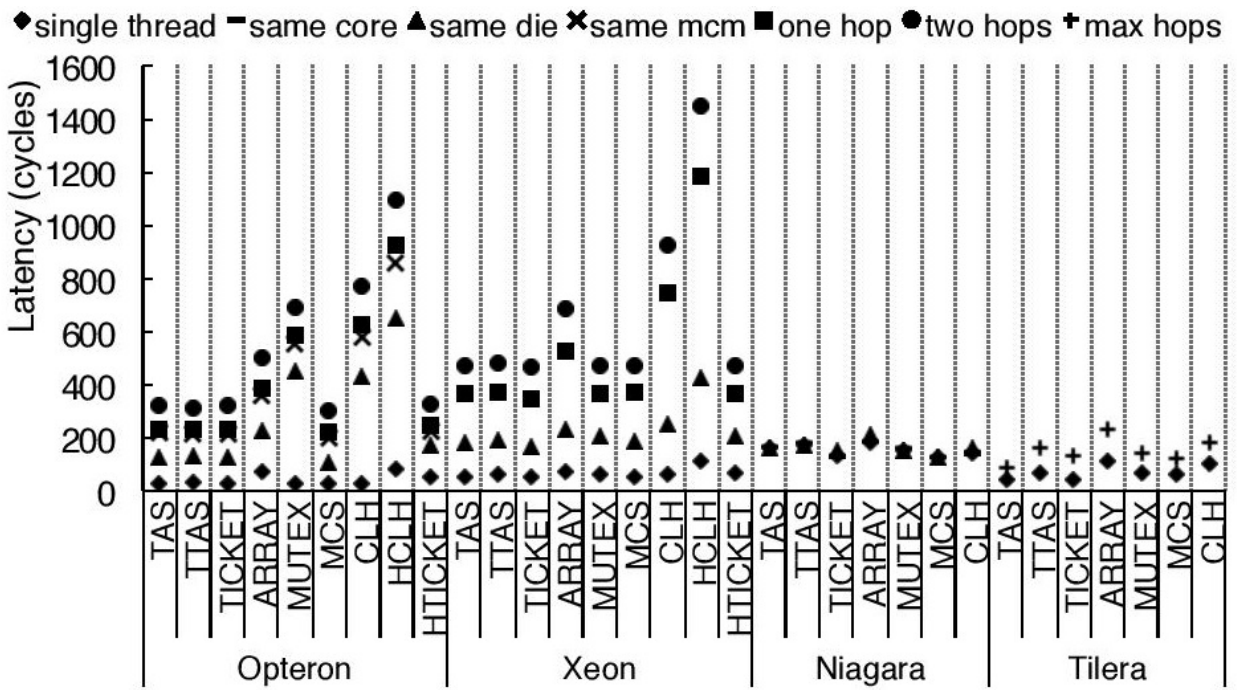


Figure 6: Uncontested lock acquisition latency based on the location of the previous owner of the lock.

Low/high contention locking

1. same methodology as for atomic operations
2. pthread mutexes are rather slow
3. simple locks perform better in low contention, but complex in high

Low/high contention locking

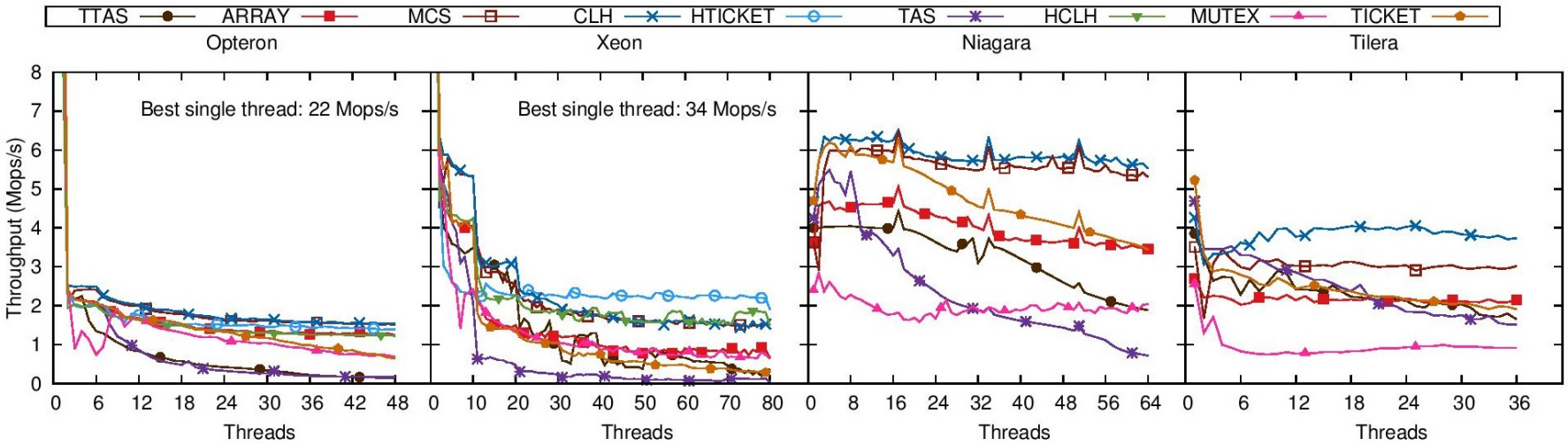


Figure 5: Throughput of different lock algorithms using a single lock.

Low/high contention locking

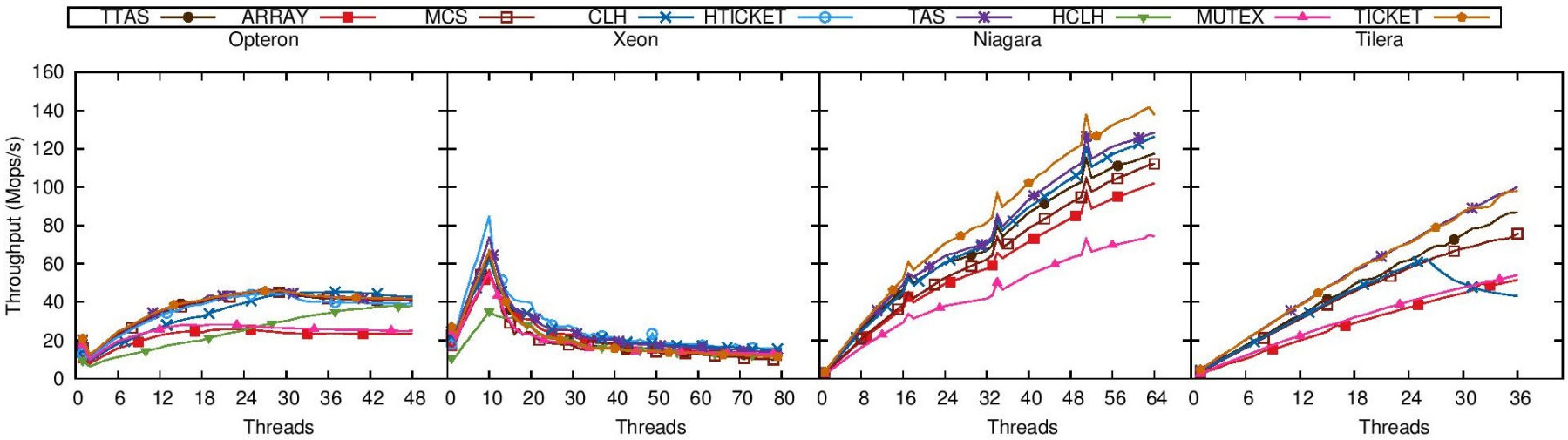


Figure 7: Throughput of different lock algorithms using 512 locks.

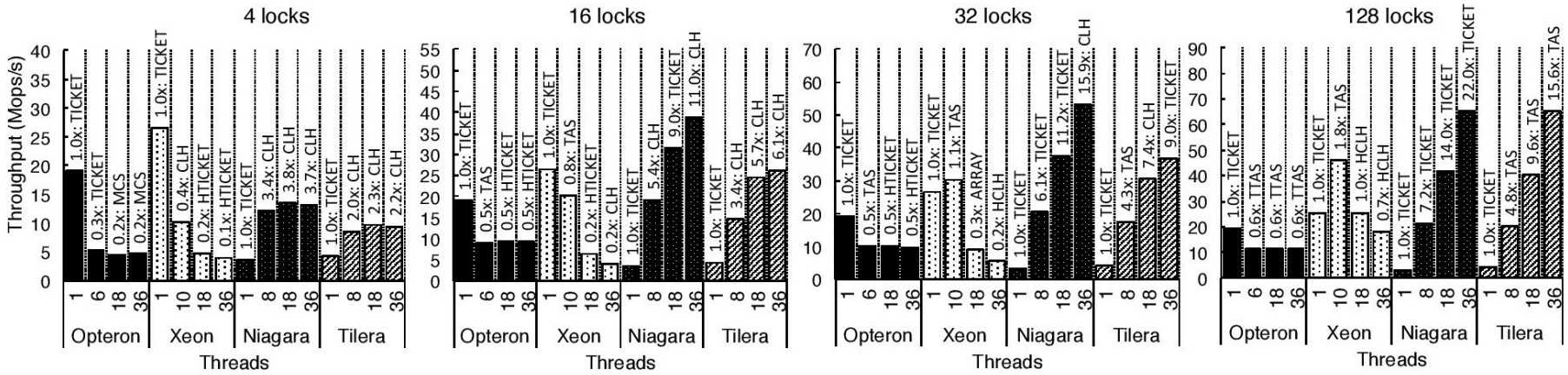
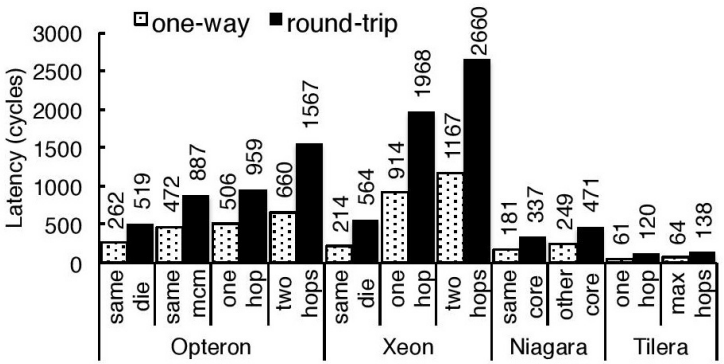


Figure 8: Throughput and scalability of locks depending on the number of locks. The "X : Y" labels on top of each bar indicate the best-performing lock (Y) and the scalability over the single-thread execution (X).

Message passing

1. one-to-one communication: Tilera performs best (message passing hardware support)
2. client-server communication: request-response more effective, message passing perfoms best for high contention

Message passing performance



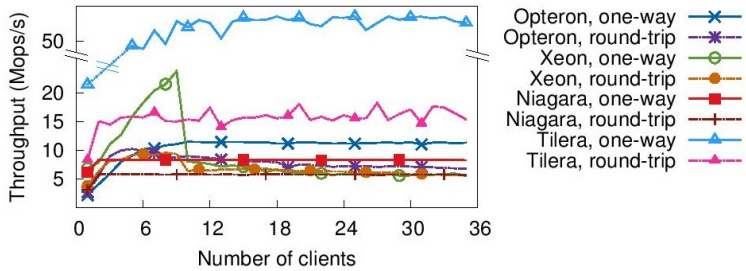


Figure 10: Total throughput of client-server communication.

Hashtable performance

1. operations: 0.8 get, 0.1 put, 0.1 remove
2. lock-based hashtables performs better for low contention (Tilera unperforms)
3. message-passing-based hashtables performs better for high contention

Hashtables performance

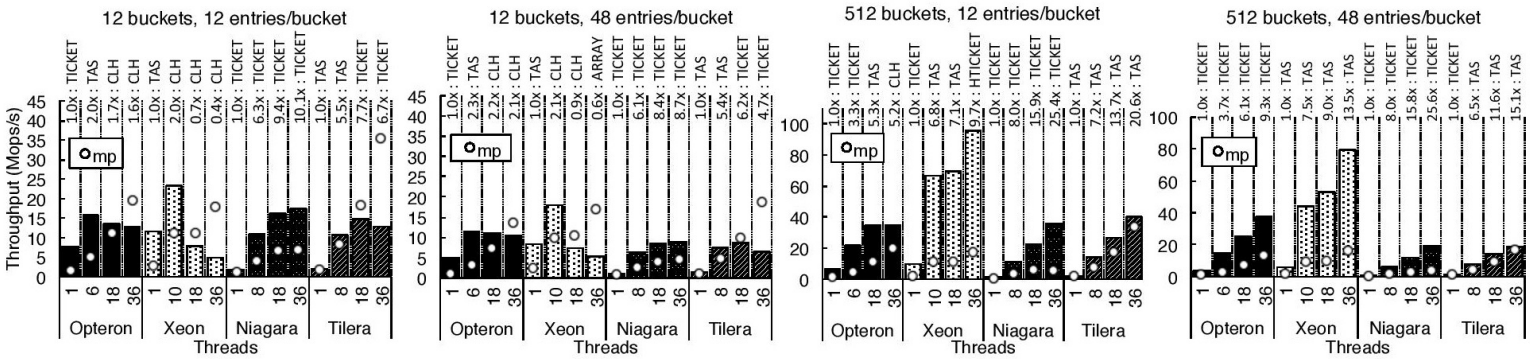


Figure 11: Throughput and scalability of the hash table (`ssht`) on different configurations. The "X : Y" labels on top of each bar indicate the best-performing lock (Y) and the scalability over the single-thread execution (X).

Memcached (key/value store) performance

1. using libslock instead of pthread mutexes
2. get - little effect in test, bottlenecks do not concern synchronization
3. set - atomic-operations-based locks performs better than mutexes on all platforms

Memcached (key/value store) performance

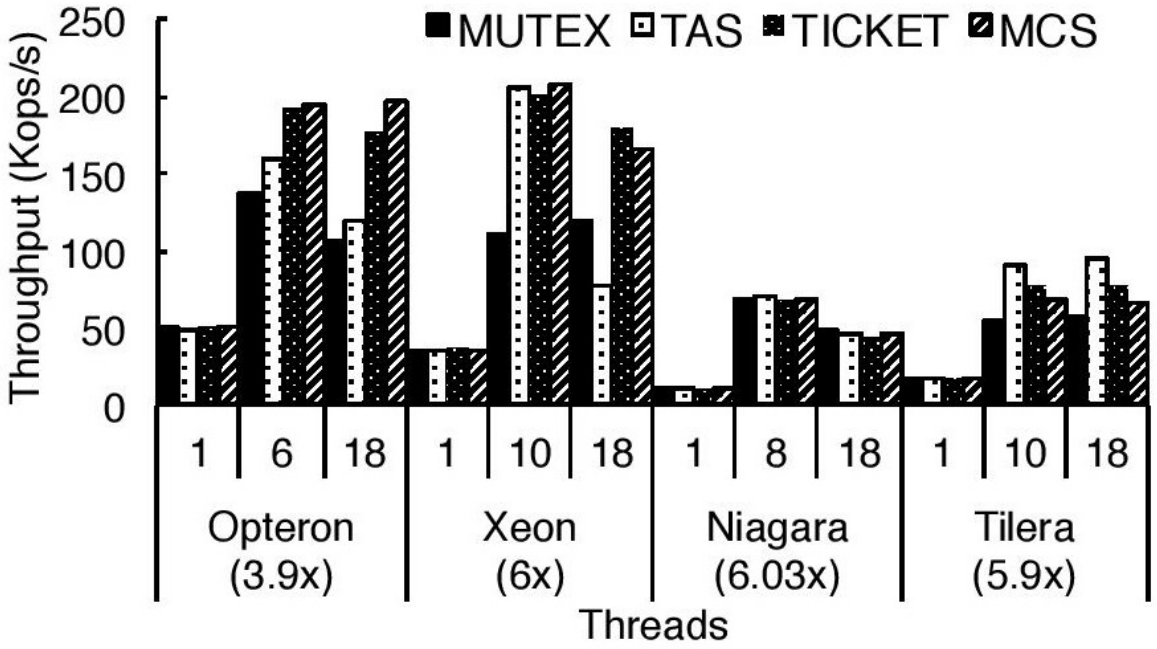


Figure 12: Throughput of Memcached using a set-only test. The maximum speed-up vs. single thread is indicated under the platform names.