CS610 Assignment 2

Khushi Gupta

September 2025

Problem 1

Setup

- Implementations: **naive**, **input-blocked**, **output-blocked** 3D cross-correlation (no padding). Kernel: $3 \times 3 \times 3$.
- Input sizes tested: $N = \{128, 256, 512\}$ (i.e. $1 \ll 7, 1 \ll 8, 1 \ll 9$).
- Blocking configurable per-dimension (B_H, B_W, B_D). Each kernel timed 5 runs, reported time values are averages.
- PAPI events collected: `PAPI_L1_DCM`, `PAPI_L2_DCM`, `PAPI_TOT_CYC`.

Best block sizes

N	Block Sizes 1	Block Sizes 2
128	(4, 4, 64)	(1, 1, 64)
256	(1, 1, 128)	(1, 1, 256)
512	(1, 1, 256)	(1, 1, 512)

Table 1: Top two performing block sizes for each input size N.

Key Observed Trends

- **Input-blocked:** Slowest execution, with the fewest L1/L2 data cache misses but the highest cycle counts.
- **Output-blocked:** Fastest execution, with cycle counts and wall-clock time similar to the naïve version.

Possible reasons for input blocked to be slow -

1. **Many dirty evictions and stores:** Input-blocked repeatedly updates scattered output locations, causing cache lines to be reloaded and written back to memory multiple times. This leads to more evictions, extra stores, and higher cycles and time. In contrast, output-blocked writes to each cache line more contiguously, reducing these costs.
2. **Extra checks and indexing overhead:** input-blocked performs additional bound checks and index arithmetic (e.g., computing output indices and testing validity), increasing instruction count and branch overhead.

Results

```
Run 1
Naive Counters: L1 DCM=1040135, L2 DCM=2225319, CPU Cycles=61972692
Input Blocked:  L1 DCM=1021989, L2 DCM=1374416, CPU Cycles=199004444
Output Blocked: L1 DCM=1099683, L2 DCM=1448103, CPU Cycles=64822656

Run 2
Naive Counters: L1 DCM=1041427, L2 DCM=2236517, CPU Cycles=62198608
Input Blocked:  L1 DCM=1030505, L2 DCM=1379365, CPU Cycles=199951570
Output Blocked: L1 DCM=1087604, L2 DCM=1453411, CPU Cycles=63365114

Run 3
Naive Counters: L1 DCM=1038797, L2 DCM=2222812, CPU Cycles=61604289
Input Blocked:  L1 DCM=1021459, L2 DCM=1378983, CPU Cycles=198061179
Output Blocked: L1 DCM=1091328, L2 DCM=1452413, CPU Cycles=63684440

Run 4
Naive Counters: L1 DCM=1040845, L2 DCM=2229413, CPU Cycles=62094533
Input Blocked:  L1 DCM=1029037, L2 DCM=1379552, CPU Cycles=201065041
Output Blocked: L1 DCM=1099059, L2 DCM=1449344, CPU Cycles=64181221

Run 5
Naive Counters: L1 DCM=1038152, L2 DCM=2233796, CPU Cycles=60446471
Input Blocked:  L1 DCM=1032957, L2 DCM=1372772, CPU Cycles=199958286
Output Blocked: L1 DCM=1108754, L2 DCM=1446069, CPU Cycles=65238036

=== Timing Results (averaged over 5 runs) ===
Naive   : 23.0256 ms
Input Blocked : 66.3696 ms
Output Blocked : 23.5914 ms
```

(a) $N = 128$, Block = (4,4,64)

```
Run 1
Naive Counters: L1 DCM=1038217, L2 DCM=2232208, CPU Cycles=61556722
Input Blocked:  L1 DCM=1038693, L2 DCM=2204011, CPU Cycles=209721570
Output Blocked: L1 DCM=1039354, L2 DCM=2160848, CPU Cycles=64952180

Run 2
Naive Counters: L1 DCM=1037462, L2 DCM=2162460, CPU Cycles=61719967
Input Blocked:  L1 DCM=1022125, L2 DCM=2167493, CPU Cycles=207723167
Output Blocked: L1 DCM=1037131, L2 DCM=2166950, CPU Cycles=63385038

Run 3
Naive Counters: L1 DCM=1038302, L2 DCM=2164729, CPU Cycles=62106823
Input Blocked:  L1 DCM=1025452, L2 DCM=2164941, CPU Cycles=208976474
Output Blocked: L1 DCM=1038641, L2 DCM=2168081, CPU Cycles=64416759

Run 4
Naive Counters: L1 DCM=1039246, L2 DCM=2164135, CPU Cycles=61980789
Input Blocked:  L1 DCM=1025118, L2 DCM=2163451, CPU Cycles=210004177
Output Blocked: L1 DCM=1039941, L2 DCM=2170408, CPU Cycles=64870593

Run 5
Naive Counters: L1 DCM=1036759, L2 DCM=2171841, CPU Cycles=61085827
Input Blocked:  L1 DCM=1024061, L2 DCM=2164131, CPU Cycles=209355118
Output Blocked: L1 DCM=1038536, L2 DCM=2171470, CPU Cycles=63628792

=== Timing Results (averaged over 5 runs) ===
Naive   : 24.9344 ms
Input Blocked : 82.0566 ms
Output Blocked : 23.156 ms
```

(b) $N = 128$, Block = (1,1,64)

```
Run 1
Naive Counters: L1 DCM=10055148, L2 DCM=18782188, CPU Cycles=493233072
Input Blocked:  L1 DCM=8685185, L2 DCM=18343632, CPU Cycles=1681603832
Output Blocked: L1 DCM=9862265, L2 DCM=18732815, CPU Cycles=499292026

Run 2
Naive Counters: L1 DCM=10127968, L2 DCM=18797068, CPU Cycles=494052610
Input Blocked:  L1 DCM=8691065, L2 DCM=18362793, CPU Cycles=1680838163
Output Blocked: L1 DCM=9864683, L2 DCM=18788904, CPU Cycles=498470406

Run 3
Naive Counters: L1 DCM=10128717, L2 DCM=18810607, CPU Cycles=493353980
Input Blocked:  L1 DCM=8677670, L2 DCM=18357342, CPU Cycles=1681677834
Output Blocked: L1 DCM=9872581, L2 DCM=18791736, CPU Cycles=498331609

Run 4
Naive Counters: L1 DCM=10098900, L2 DCM=18791140, CPU Cycles=492025327
Input Blocked:  L1 DCM=8676893, L2 DCM=18313567, CPU Cycles=1682480877
Output Blocked: L1 DCM=9793146, L2 DCM=18718524, CPU Cycles=499203519

Run 5
Naive Counters: L1 DCM=10072017, L2 DCM=18740587, CPU Cycles=493214587
Input Blocked:  L1 DCM=8679107, L2 DCM=18318417, CPU Cycles=1681274455
Output Blocked: L1 DCM=9817931, L2 DCM=18725177, CPU Cycles=499152172

=== Timing Results (averaged over 5 runs) ===
Naive   : 174.138 ms
Input Blocked : 594.369 ms
Output Blocked : 159.135 ms
```

(c) $N = 256$, Block = (1,1,256)

```
Run 1
Naive Counters: L1 DCM=10004631, L2 DCM=18514015, CPU Cycles=495092978
Input Blocked:  L1 DCM=9533217, L2 DCM=18547138, CPU Cycles=1540114363
Output Blocked: L1 DCM=9739104, L2 DCM=18589716, CPU Cycles=505446426

Run 2
Naive Counters: L1 DCM=9979171, L2 DCM=18627493, CPU Cycles=494029689
Input Blocked:  L1 DCM=9531488, L2 DCM=18575600, CPU Cycles=1538023915
Output Blocked: L1 DCM=9750963, L2 DCM=18675099, CPU Cycles=504265558

Run 3
Naive Counters: L1 DCM=9988868, L2 DCM=18621892, CPU Cycles=494990266
Input Blocked:  L1 DCM=9526090, L2 DCM=18543659, CPU Cycles=1538048924
Output Blocked: L1 DCM=9759080, L2 DCM=18687306, CPU Cycles=503484594

Run 4
Naive Counters: L1 DCM=9999333, L2 DCM=18608926, CPU Cycles=493903960
Input Blocked:  L1 DCM=9532526, L2 DCM=18567210, CPU Cycles=1537714296
Output Blocked: L1 DCM=9751737, L2 DCM=18574721, CPU Cycles=505111768

Run 5
Naive Counters: L1 DCM=9975329, L2 DCM=18561692, CPU Cycles=494324159
Input Blocked:  L1 DCM=9533617, L2 DCM=18583488, CPU Cycles=1538505247
Output Blocked: L1 DCM=9761621, L2 DCM=18640070, CPU Cycles=505515538

=== Timing Results (averaged over 5 runs) ===
Naive   : 153.999 ms
Input Blocked : 564.778 ms
Output Blocked : 146.417 ms
```

(d) $N = 256$, Block = (1,1,128)

```
Run 1
Naive Counters: L1 DCM=363486505, L2 DCM=148975675, CPU Cycles=4797733980
Input Blocked:  L1 DCM=161110391, L2 DCM=147829819, CPU Cycles=10193290602
Output Blocked: L1 DCM=341258427, L2 DCM=152628224, CPU Cycles=3664664205

Run 2
Naive Counters: L1 DCM=333615507, L2 DCM=151958745, CPU Cycles=3690936586
Input Blocked:  L1 DCM=157168996, L2 DCM=149317818, CPU Cycles=9140007391
Output Blocked: L1 DCM=341772254, L2 DCM=153026261, CPU Cycles=3660284707

Run 3
Naive Counters: L1 DCM=333055183, L2 DCM=151043976, CPU Cycles=3682241411
Input Blocked:  L1 DCM=156561975, L2 DCM=148341371, CPU Cycles=9235542240
Output Blocked: L1 DCM=340911946, L2 DCM=152131567, CPU Cycles=3671892814

Run 4
Naive Counters: L1 DCM=332644603, L2 DCM=152172278, CPU Cycles=3665116513
Input Blocked:  L1 DCM=158011288, L2 DCM=149275406, CPU Cycles=9145798370
Output Blocked: L1 DCM=339271460, L2 DCM=153945272, CPU Cycles=3640499862

Run 5
Naive Counters: L1 DCM=331288713, L2 DCM=151439711, CPU Cycles=3685533240
Input Blocked:  L1 DCM=158076331, L2 DCM=149283973, CPU Cycles=9137406195
Output Blocked: L1 DCM=337966450, L2 DCM=152797618, CPU Cycles=3661923822

=== Timing Results (averaged over 5 runs) ===
Naive   : 969.785 ms
Input Blocked : 2298.24 ms
Output Blocked : 913.453 ms
```

(e) $N = 512$, Block = (1,1,512)

```
Run 1
Naive Counters: L1 DCM=333497784, L2 DCM=152294598, CPU Cycles=3698261543
Input Blocked:  L1 DCM=161857138, L2 DCM=150800877, CPU Cycles=11399322230
Output Blocked: L1 DCM=344352449, L2 DCM=153256301, CPU Cycles=3697725651

Run 2
Naive Counters: L1 DCM=336096989, L2 DCM=150965295, CPU Cycles=3723294279
Input Blocked:  L1 DCM=161706068, L2 DCM=149900631, CPU Cycles=11395960149
Output Blocked: L1 DCM=343849619, L2 DCM=152613608, CPU Cycles=3712794856

Run 3
Naive Counters: L1 DCM=337868456, L2 DCM=152154101, CPU Cycles=3739269755
Input Blocked:  L1 DCM=161655973, L2 DCM=149936465, CPU Cycles=11395900480
Output Blocked: L1 DCM=343840127, L2 DCM=152907029, CPU Cycles=3695827906

Run 4
Naive Counters: L1 DCM=336963238, L2 DCM=152324914, CPU Cycles=3699605198
Input Blocked:  L1 DCM=161607723, L2 DCM=150758201, CPU Cycles=11396624506
Output Blocked: L1 DCM=343824529, L2 DCM=153041165, CPU Cycles=3701339823

Run 5
Naive Counters: L1 DCM=337132304, L2 DCM=152401482, CPU Cycles=3698879226
Input Blocked:  L1 DCM=161558264, L2 DCM=151184063, CPU Cycles=11394478687
Output Blocked: L1 DCM=343958707, L2 DCM=152801749, CPU Cycles=3707370355

=== Timing Results (averaged over 5 runs) ===
Naive   : 828.833 ms
Input Blocked : 2510.22 ms
Output Blocked : 815.854 ms
```

(f) $N = 512$, Block = (1,1,256)

Figure 1: Best two block configurations per input size N. Each subfigure shows the performance for one block size.

Problem 2

N threads read N files and count words & lines. The original code performed updates to shared counters inside inner loops, causing heavy lock contention and cache-coherence (false sharing) traffic.

Key performance issues in original code

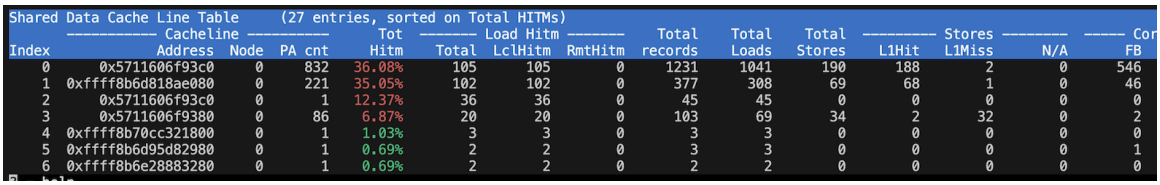
- Per-word `pthread_mutex_lock/unlock` on `tracker.total_words_processed`.
- Frequent writes to contiguous `tracker.word_count[]` entries from different threads - false sharing and many HITM cacheline transfers.
- Per-line locking for line counter (lock/unlock per line).

Optimizations applied

- Local aggregation of counters inside each thread (`word_count`, `line_count`).
- Single write to the per-thread slot (`tracker.word_count[tid]`) and one short critical section to add local totals to global counters at thread end.
- Removed per-word and per-line frequent locking.

Observed effect

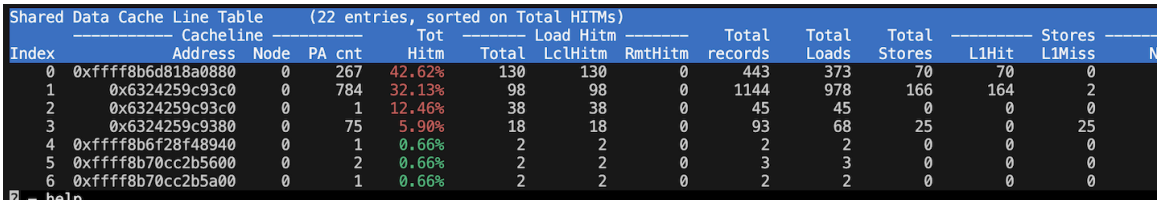
- **Before:** Shared cachelines showed many HITM events indicating frequent ownership transfers between cores. These events mean that one core attempted to read or write a cacheline owned by another core, triggering coherence traffic and stalls. The two snapshots below illustrate the high level of coherence activity and contention on the shared data cachelines:



Shared Data Cache Line Table (27 entries, sorted on Total HITMs)

Index	Cacheline Address	Node	PA cnt	Tot Hitm	Load Hitm Total	Load Hitm LclHitm	Load Hitm RmtHitm	Total records	Total Loads	Total Stores	Stores L1Hit	Stores L1Miss	Stores N/A	Cor FB
0	0x5711606f93c0	0	832	36.08%	105	105	0	1231	1041	190	188	2	0	546
1	0xffff8b6d818ae080	0	221	35.05%	102	102	0	377	308	69	68	1	0	46
2	0x5711606f93c0	0	1	12.37%	36	36	0	45	45	0	0	0	0	0
3	0x5711606f9380	0	86	6.87%	20	20	0	103	69	34	2	32	0	2
4	0xffff8b70cc321800	0	1	1.03%	3	3	0	3	3	0	0	0	0	0
5	0xffff8b6d95d82980	0	1	0.69%	2	2	0	3	3	0	0	0	0	1
6	0xffff8b6e28883280	0	1	0.69%	2	2	0	2	2	0	0	0	0	0

Figure 2: Before optimization - cacheline coherence records



Shared Data Cache Line Table (22 entries, sorted on Total HITMs)

Index	Cacheline Address	Node	PA cnt	Tot Hitm	Load Hitm Total	Load Hitm LclHitm	Load Hitm RmtHitm	Total records	Total Loads	Total Stores	Stores L1Hit	Stores L1Miss	N
0	0xffff8b6d818a0880	0	267	42.62%	130	130	0	443	373	70	70	0	
1	0x6324259c93c0	0	784	32.13%	98	98	0	1144	978	166	164	2	
2	0x6324259c93c0	0	1	12.46%	38	38	0	45	45	0	0	0	
3	0x6324259c9380	0	75	5.90%	18	18	0	93	68	25	0	25	
4	0xffff8b6f28f48940	0	1	0.66%	2	2	0	2	2	0	0	0	
5	0xffff8b70cc2b5600	0	2	0.66%	2	2	0	3	3	0	0	0	
6	0xffff8b70cc2b5a00	0	1	0.66%	2	2	0	2	2	0	0	0	

Figure 3: Before optimization - cacheline coherence records

- **After:** The Shared Data Cache Line Table shows that inter-core cacheline transfers were almost entirely eliminated. In the optimized run, the previously affected cachelines no longer

appeared as sources of contention. Only a single residual entry was recorded with just one HITM event. This negligible activity arises because different threads still update their own per-thread word counters, which may occasionally map to the same cacheline :

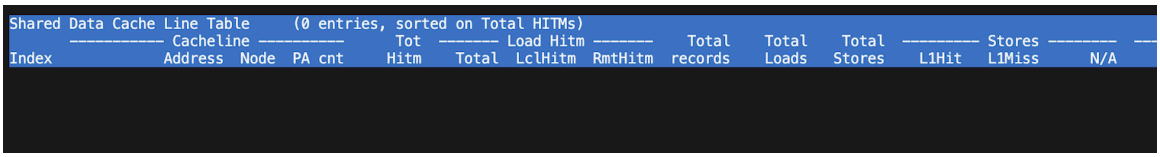


Figure 4: After optimization - cacheline coherence records.

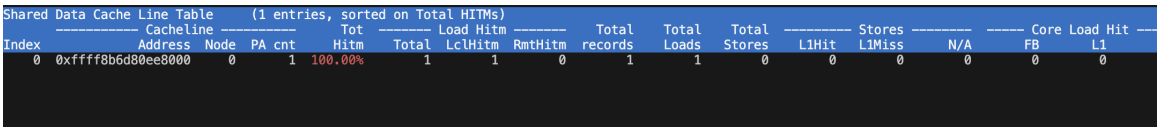


Figure 5: After optimization - cacheline coherence records.

Problem 3

Results

The following results were obtained on *image*1.*cse.iitk.ac.in*. For 64 threads the execution could not complete in reasonable time.

Lock / # threads	**1**	**2**	**4**	**8**	**16**	**32**
Pthread mutex	Run1: 30	Run1: 425	Run1: 1536	Run1: 8245	Run1: 45940	Run1: 210303
	Run2: 31	Run2: 262	Run2: 1121	Run2: 8460	Run2: 43667	Run2: 179210
	Run3: 28	Run3: 517	Run3: 1399	Run3: 6961	Run3: 42118	Run3: 175000
	Run4: 24	Run4: 379	Run4: 1113	Run4: 6904	Run4: 36090	Run4: 172615
	Run5: 32	Run5: 444	Run5: 1860	Run5: 7014	Run5: 42287	Run5: 178313
	Avg: 29	**Avg: 405**	**Avg: 1405**	**Avg: 7516**	**Avg: 42020**	**Avg: 183088**
Filter lock	Run1: 9	Run1: 722	Run1: 4597	Run1: 119103	Run1: 1360747	Run1: 11245362
	Run2: 5	Run2: 554	Run2: 4087	Run2: 122311	Run2: 1351788	Run2: 11025148
	Run3: 5	Run3: 810	Run3: 4395	Run3: 115104	Run3: 1351275	Run3: 11190998
	Run4: 5	Run4: 678	Run4: 4304	Run4: 139701	Run4: 1362924	Run4: 11167281
	Run5: 5	Run5: 722	Run5: 4590	Run5: 126059	Run5: 1372719	Run5: 11132490
	Avg: 6	**Avg: 697**	**Avg: 4395**	**Avg: 124456**	**Avg: 1359891**	**Avg: 11152256**
Bakery lock	Run1: 74	Run1: 864	Run1: 3760	Run1: 44536	Run1: 252431	Run1: 1384240
	Run2: 48	Run2: 720	Run2: 3600	Run2: 18384	Run2: 253678	Run2: 1369689
	Run3: 48	Run3: 904	Run3: 11148	Run3: 44568	Run3: 245631	Run3: 1406609
	Run4: 64	Run4: 946	Run4: 3788	Run4: 34160	Run4: 250302	Run4: 1455927
	Run5: 46	Run5: 852	Run5: 4576	Run5: 34000	Run5: 250706	Run5: 1388586
	Avg: 56	**Avg: 857.2**	**Avg: 5374**	**Avg: 35130**	**Avg: 250550**	**Avg: 1401010**
Spin lock	Run1: 18	Run1: 383	Run1: 3474	Run1: 15129	Run1: 111595	Run1: 687443
	Run2: 11	Run2: 390	Run2: 2115	Run2: 17856	Run2: 101857	Run2: 729216
	Run3: 13	Run3: 767	Run3: 2039	Run3: 15239	Run3: 120797	Run3: 868888
	Run4: 24	Run4: 286	Run4: 2557	Run4: 15393	Run4: 112310	Run4: 761947
	Run5: 11	Run5: 326	Run5: 2895	Run5: 17702	Run5: 107725	Run5: 694699
	Avg: 15	**Avg: 430**	**Avg: 2616**	**Avg: 16264**	**Avg: 110857**	**Avg: 748439**
Ticket lock	Run1: 31	Run1: 408	Run1: 6883	Run1: 25816	Run1: 111918	Run1: 507119
	Run2: 44	Run2: 406	Run2: 2428	Run2: 27266	Run2: 121398	Run2: 521938
	Run3: 39	Run3: 776	Run3: 2640	Run3: 25544	Run3: 119586	Run3: 508877
	Run4: 68	Run4: 488	Run4: 6503	Run4: 12692	Run4: 121899	Run4: 498921
	Run5: 40	Run5: 560	Run5: 2408	Run5: 27084	Run5: 65724	Run5: 516483
	Avg: 44	**Avg: 528**	**Avg: 4172**	**Avg: 23680**	**Avg: 108105**	**Avg: 510668**
Array Q lock	Run1: 11	Run1: 548	Run1: 3116	Run1: 23199	Run1: 50287	Run1: 549321
	Run2: 24	Run2: 618	Run2: 2592	Run2: 22811	Run2: 69180	Run2: 543002
	Run3: 13	Run3: 1878	Run3: 2692	Run3: 18767	Run3: 66194	Run3: 561290
	Run4: 22	Run4: 948	Run4: 2704	Run4: 28134	Run4: 73406	Run4: 541796
	Run5: 15	Run5: 748	Run5: 2840	Run5: 12776	Run5: 69052	Run5: 545946
	Avg: 17	**Avg: 948**	**Avg: 2789**	**Avg: 21137**	**Avg: 65624**	**Avg: 548271**

Table 2: All these were recorded for input size $N = 1e6$

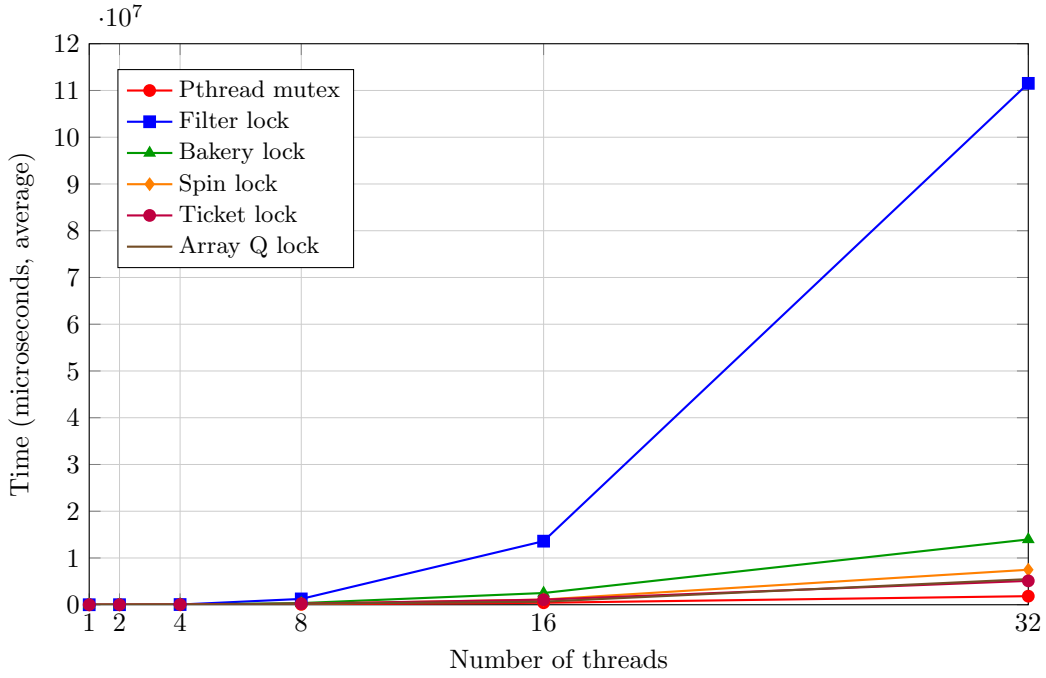


Figure 6: Average time (µs) vs number of threads for each lock type.

Optimizations

- Use of `pause` instruction inside spin-waits to reduce pipeline contention and busy-loop aggressiveness.
- Explicit `mfence` calls where needed to enforce ordering between reads/writes (Filter, Bakery, Ticket).
- Array-based queue uses cache-line alignment/padding (`alignas(64)`/padding) so each waiter spins on its own cache line to avoid sharing.

Trends

- At high thread counts, FilterLock and BakeryLock degrade noticeably due to their O(N) scanning/booking behavior.
- TicketLock and ArrayQLock provide better stability. TicketLock gives orderly FIFO handoff, while ArrayQLock reduces coherence by having each waiter spin on a separate, padded slot. Both perform better compared with a spinlock at high number of threads.
- Adding pause instruction in loops and cache-line padding for ArrayQLock improved performance.