NC State University

Department of Electrical and Computer Engineering

ECE 463/521: Spring 2016

Project #1: Cache Hierarchy

by

Ritesh Gajare

NCSU Honor Pledge: "I have neither given nor received unauthorized aid on this test or assignment."

Student's electronic signature: ____Ritesh Gajare__
(sign by typing your name)

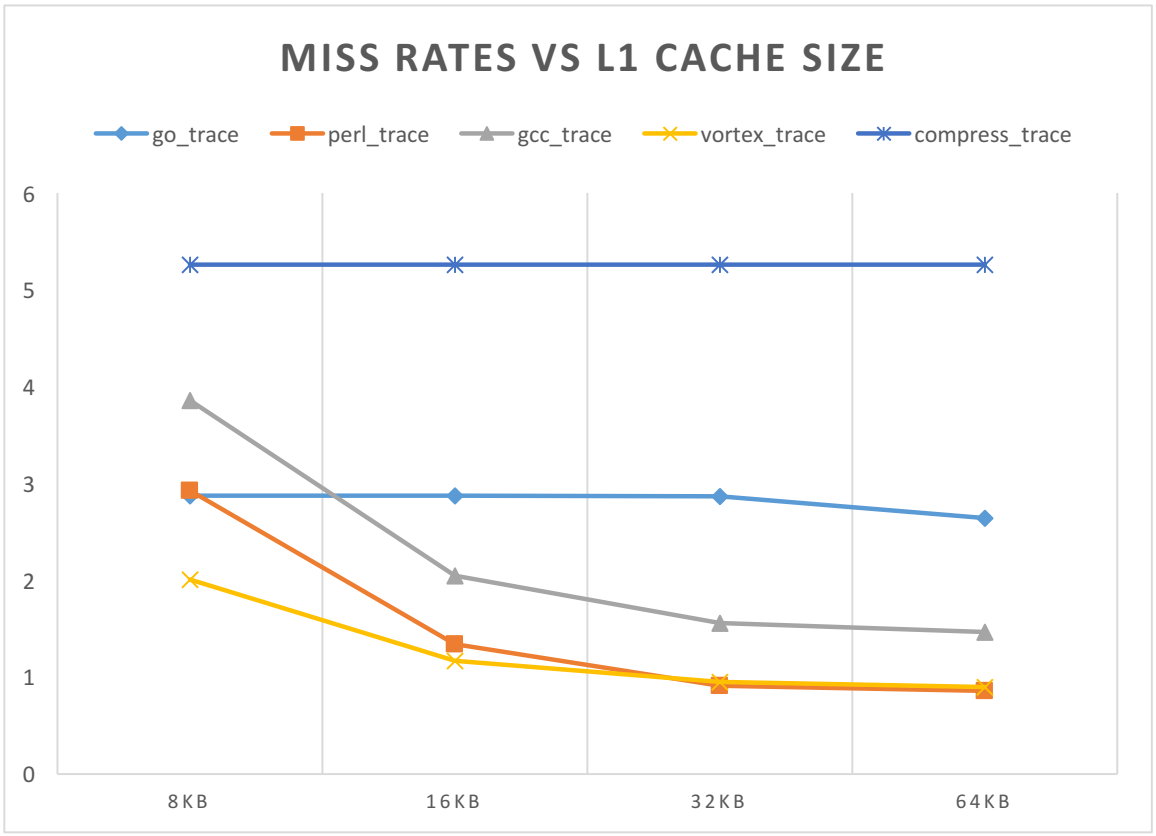
Course number: _____521___
(463 or 521 ?)

Plot #1

Assume that there is no L2 cache. Assume that LRU replacement policy is used for the L1 cache.

Fix the L1 block size at 64B and the L1 associativity at 4. Vary the cache size between 8KB, 16KB, 32KB and 64KB. Plot the L1 cache miss rates. The plot should have 4 data points per benchmark.

	go_trace	perl_trace	gcc_trace	vortex_trace	compress_trace
8KB	2.874	2.929	3.859	2.01	5.26
16KB	2.871	1.337	2.045	1.166	5.26
32KB	2.869	0.91	1.558	0.951	5.26
64KB	2.645	0.86	1.466	0.899	5.26



Observation: Miss rates decreases for most of the benchmarks as we double the cache size from 8KB to 16KB.

Compress & go benchmark doesn't benefit much for the cache size variations.

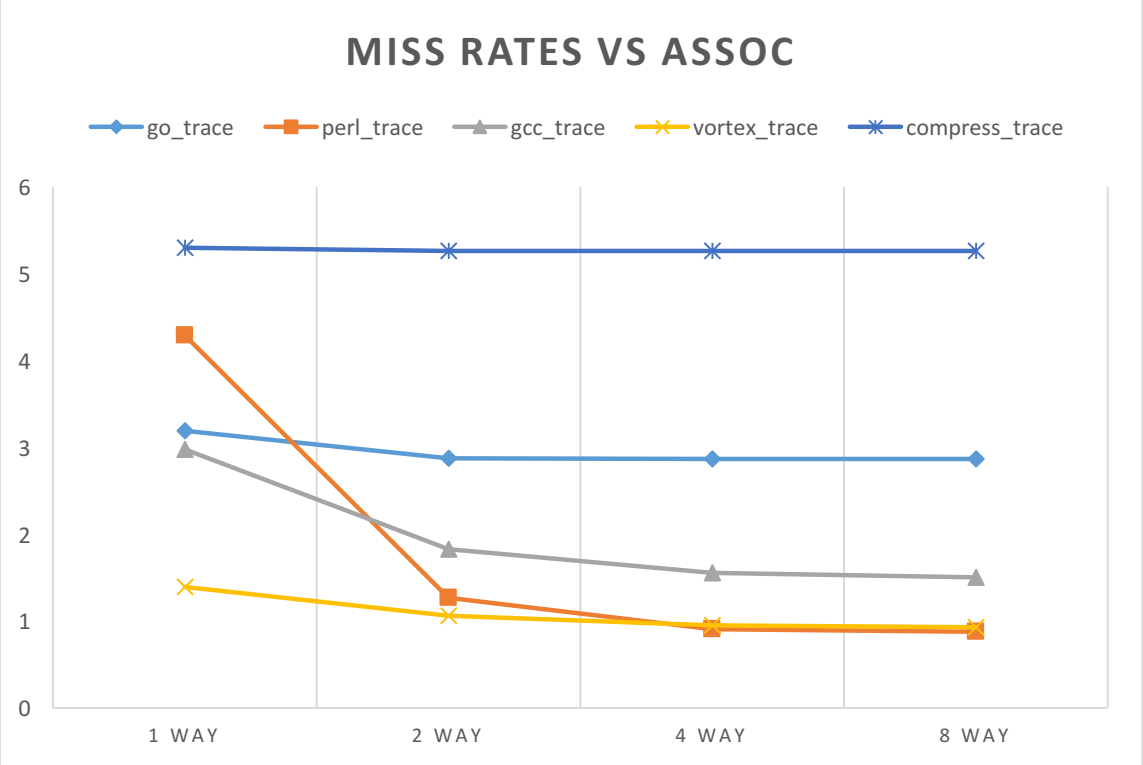
Discussion:

- Decrease in miss rate s significant in the case of perl & gcc benchmark.
- In the case of go benchmark there is no significant decrease in miss rate.
- The most important reason for the decrease in miss rates is increasing the cache size decreases the capacity misses. Now the cache is large enough that it can hold most the data and evicts.
- This decrease in miss rates has achieves saturation after some size, as there it cannot change the cold miss as well as conflict misses.

Plot #2:

Fix the L1 block size at 64B and the L1 cache size at 32KB. Vary the associativity between 1, 2, 4 and 8. Plot the L1 cache miss rates. The plot should have 4 data points per benchmark.

	go_trace	perl_trace	gcc_trace	vortex_trace	compress_trace
1 way	3.19	4.292	2.977	1.391	5.298
2 way	2.872	1.264	1.831	1.064	5.26
4 way	2.869	0.91	1.558	0.951	5.26
8 way	2.869	0.879	1.5	0.928	5.26



Observation: The miss rate goes on decreasing with the increase in the associativity and is applicable for all of the benchmarks

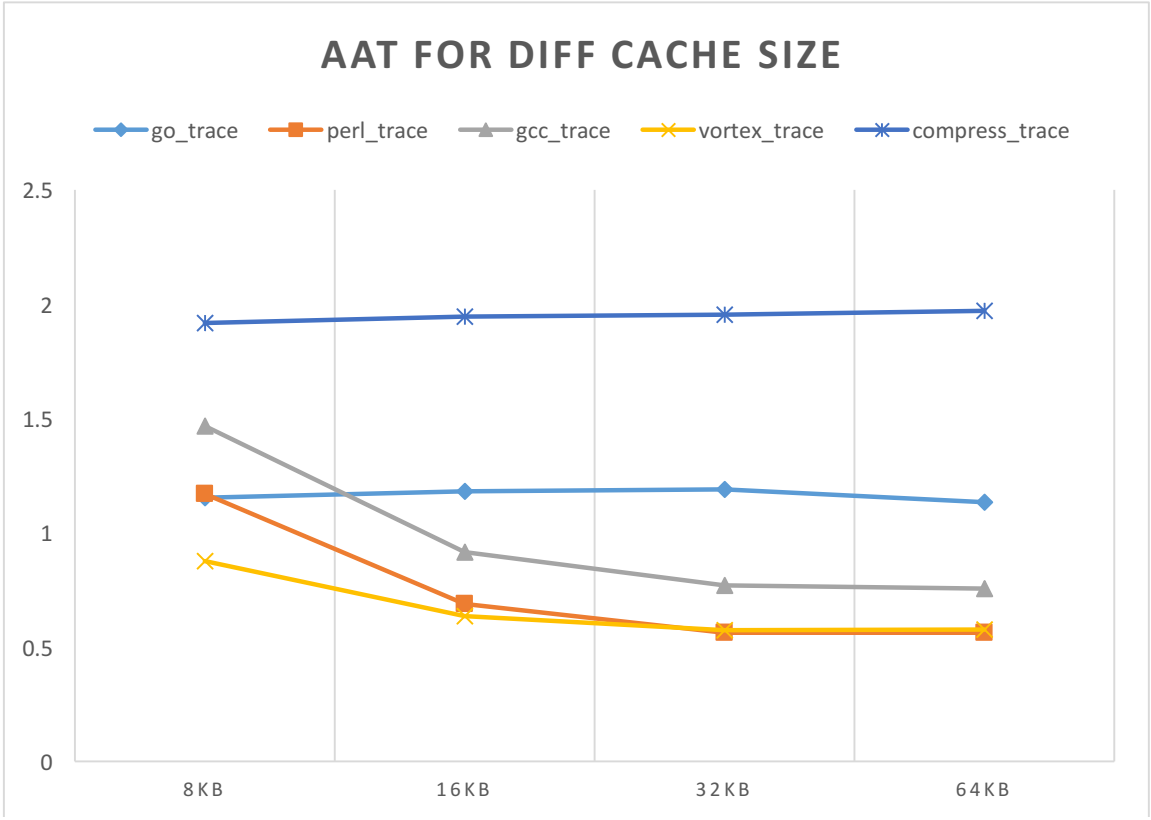
Discussions:

- Perl benchmark benefits the most with the increase in assoc from 1 to 2.
- Reason: Increasing the associativity decreases the conflict misses that otherwise happens due to two cachelines mapping to the same set, hence the existing cacheline is to be evicted.
- After 4 way assoc, we can also observe that increasing the assoc further doesn't affect much of the performance because, cold misses & capacity misses are not affected.

To do further analysis on the cache performance, Average access times are computed based on the data used from CACTI simulations.

We plot the Average access time for plot #1.

	go_trace	perl_trace	gcc_trace	vortex_trace	compress_trace
8KB	1.130853	1.148453	1.446053	0.854373	1.894373
16KB	1.152656	0.661776	0.888336	0.607056	1.917136
32KB	1.18933	0.56245	0.76981	0.57557	1.95445
64KB	1.165881	0.594681	0.788601	0.607161	2.002681

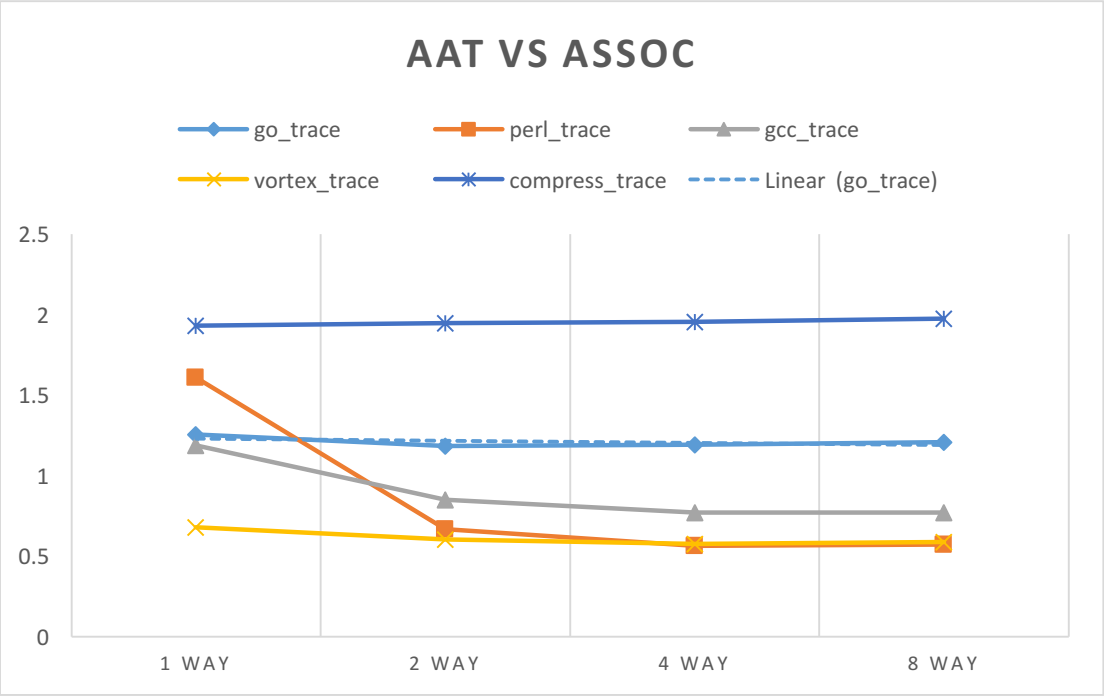


Observation:

- The key thing to notice is that as we increase the cache size, the access time goes on reducing approximately till the cache size of 32KB.
- With the increase of cache size from 32KB to 64KB, the access time increases a little bit.
- This trend is not followed by compress and go benchmarks.
- In both of the case, the average access time increases by a small amount.

We also computed the average access time required with the variation of associativity. Given below is the data-

	go_trace	perl_trace	gcc_trace	vortex_trace	compress_trace
1 way	1.254153	1.606793	1.185993	0.678473	1.928713
2 way	1.181486	0.666926	0.848366	0.602926	1.945646
4 way	1.18933	0.56245	0.76981	0.57557	1.95445
8 way	1.206591	0.569791	0.768511	0.585471	1.971711



Observation:

- The key thing to notice is that as we increase the cache size, the access time goes on reducing approximately till the cache size of 4-way.
- With the increase of cache size from 4-way to 8-way, the access time increases a little bit.

- In case of Compress benchmark, the average access time increases with the increase in assoc.

Implications:

- Since we know that there was a miss rate decrease for every increase in the cache size, but that doesn't imply that the cache average access time decreases. (Compress benchmark).

Area consideration:

	1	2	4	8
8	0.053293238	0.083756164	0.068434156	0.102584886
16	0.096748995	0.130107044	0.105941693	0.130444675
32	0.210543576	0.205554649	0.236647681	0.242170635
64	0.330469394	0.350242085	0.30228937	0.360317611

The area increases as we increase the size of the cache as well as if we increase the assoc of the cache. Taking this under account & considering the fact that smaller area (die area) results lower cost, we should consider cache structure such that it has as small as necessary for greater performance.

Recommendation:

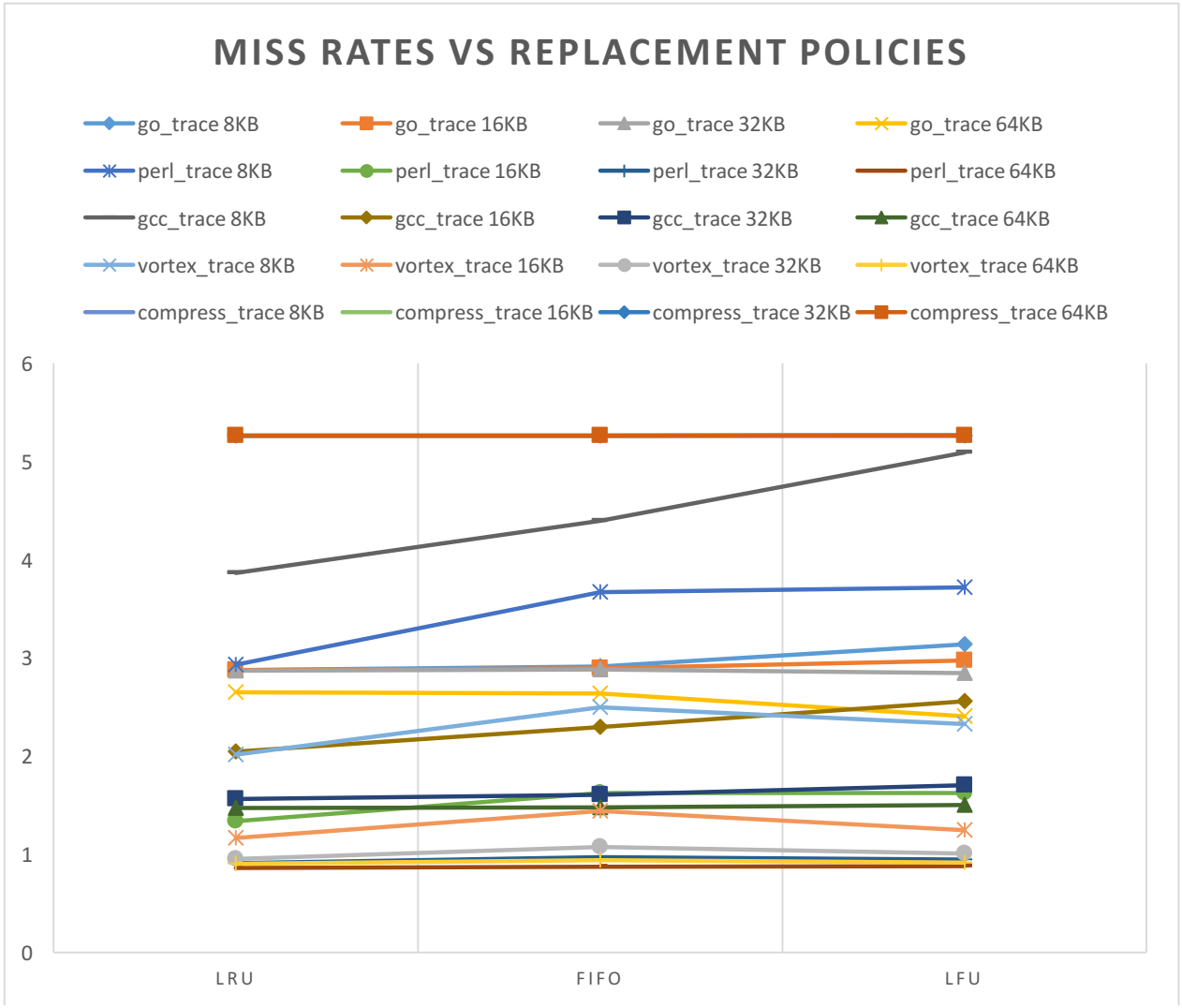
- Considering the area values given in CACTI table & the plots #1 & #2, we will recommend 32KB cache size with 2-way assoc.
- If we are running & considering cache architecture for processor based only on compress benchmark, then we should recommend 8KB 1-way cache as miss rates/AAT remains almost same for all the different configuration. And the fact that the area required is less by a significant amount (approx. 0.05).
- If we are running & considering cache architecture for processor based only on go benchmark, then we would recommend 8KB 4-way cache as miss rates/AAT remains almost same for all the different configuration. And the fact that the area required is less by a significant amount (approx. 0.07).
- If we are running & considering cache architecture for processor based only on perl, vortex, gcc benchmark, then we would recommend 32KB 4-way cache as miss rates/AAT remains almost same for all the different configuration.

Plot #3

Fix the L1 block size at 64B and the L1 associativity at 4. Vary cache size between 8KB, 16KB, 32KB and 64KB. For each configuration vary the replacement policy between LRU, FIFO and LFU. Plot the L1 Cache miss rates for each replacement policy. Note

that for each benchmark this plot will have 3 data series (one per replacement policy) and each data series will have 4 data points (one per cache size). That is, 12 data points per benchmark.

	go_trace			perl_trace			gcc_trace			vortex_trace			compress_trace		
	LRU	FIFO	LFU	LRU	FIFO	LFU	LRU	FIFO	LFU	LRU	FIFO	LFU	LRU	FIFO	LFU
8KB	2.874	2.908	3.136	2.929	3.664	3.714	3.859	4.394	5.087	2.01	2.494	2.324	5.26	5.26	5.265
16KB	2.871	2.889	2.973	1.337	1.623	1.624	2.045	2.293	2.556	1.166	1.438	1.241	5.26	5.26	5.264
32KB	2.869	2.878	2.842	0.91	0.968	0.944	1.558	1.601	1.702	0.951	1.075	1.008	5.26	5.26	5.26
64KB	2.645	2.637	2.404	0.86	0.869	0.879	1.466	1.476	1.497	0.899	0.935	0.913	5.26	5.26	5.26



Observation:

- The miss rate is maximum for LFU policy, for most of the benchmarks.
- LRU behave sufficiently good for a large number of benchmarks and cache configuration.
- Hardware implementation of LFU is relatively complex.
- For smaller cache size (typically 8KB), miss rates for LFU are higher than that of LRU & FIFO
- For larger cache size (typically 32-63KB), miss rates are comparable for all of the replacement policies.
- In case of go benchmark, 64KB L1 has lower miss rates for LFU than LRU/FIFO.

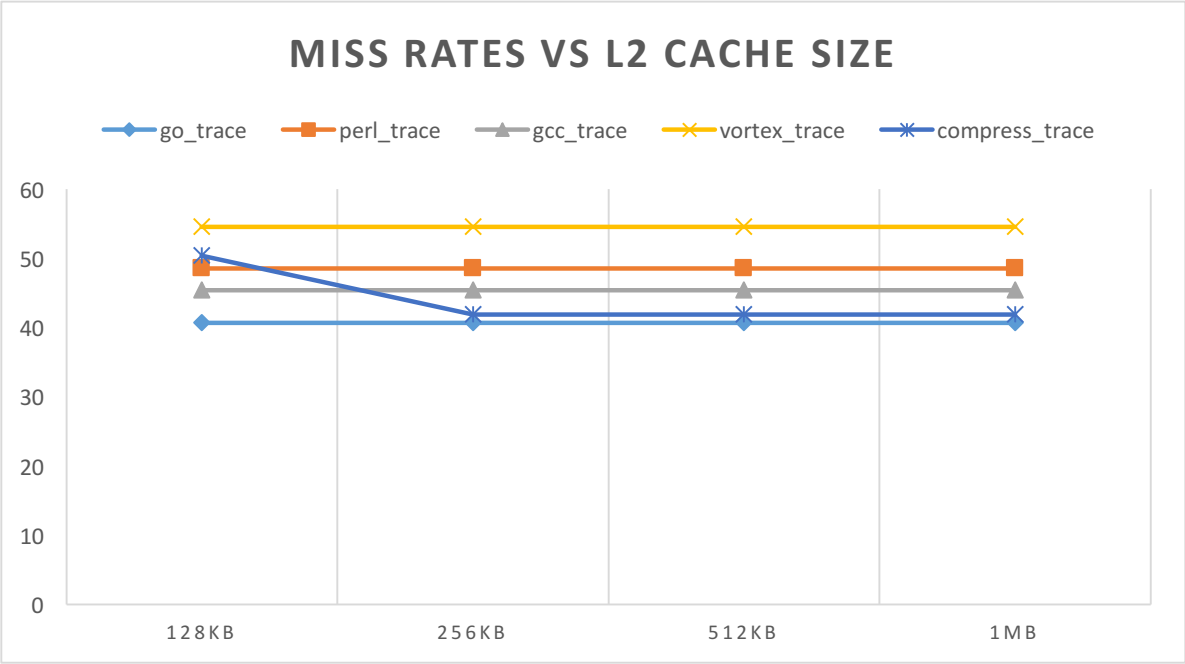
Recommendation: We should choose LRU as our replacement policy as it reduces the miss rates.

Plot #4

Fix block size at 64B, L1 cache size at 16KB, L1 associativity at 4 and the replacement policy to be LRU. Assume that the L2 Cache is always non-inclusive.
Fix L2 associativity at 8. Vary the L2 cache size between 128KB, 256KB, 512KB and 1MB. Plot the L2 cache miss rates. The plot should have 4 data points per benchmark.

Following is the data points based on the experiments:

	go_trace	perl_trace	gcc_trace	vortex_trace	compress_trace
128KB	40.7289	48.6056	45.446	54.562	50.4195
256KB	40.7289	48.6056	45.446	54.562	41.885
512KB	40.7289	48.6056	45.446	54.562	41.885
1MB	40.7289	48.6056	45.446	54.562	41.885

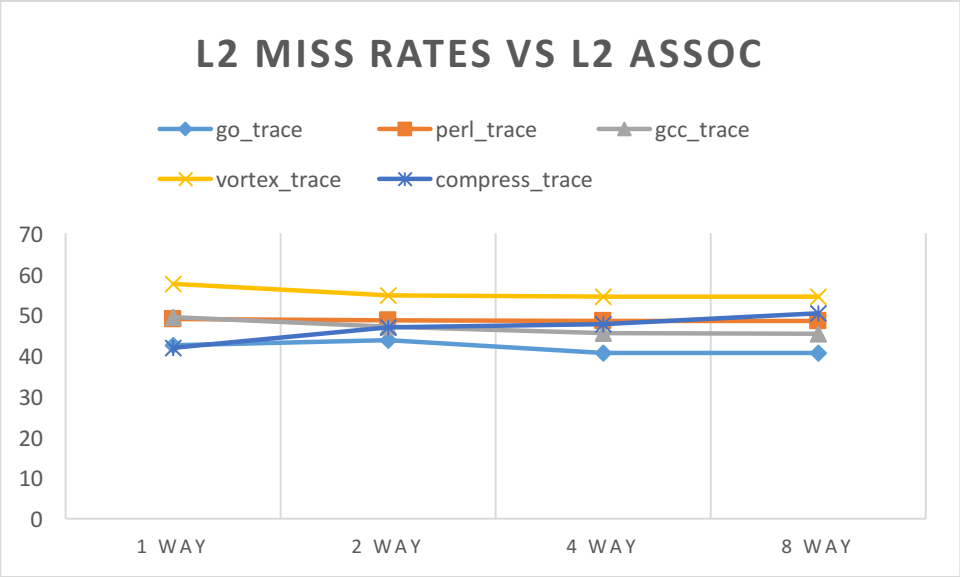


Observation:

- Miss rates usually remains the same for most of the benchmark.
- In case of compress benchmark, miss rate gets lowered with the increase in the cache size.

Plot #5

	go_trace	perl_trace	gcc_trace	vortex_trace	compress_trace
1 way	42.511	49.0609	49.4836	57.6642	41.9058
2 way	43.7725	48.7763	47.1674	54.8054	47.0119
4 way	40.7289	48.6056	45.4773	54.562	47.768
8 way	40.7289	48.6056	45.446	54.562	50.4195



Observation:

- Miss rates usually decreases as we increase the assoc for most of the benchmarks.
- In case of compress benchmark, miss rate increases with increase in the L2 assoc.

Average access times analysis:

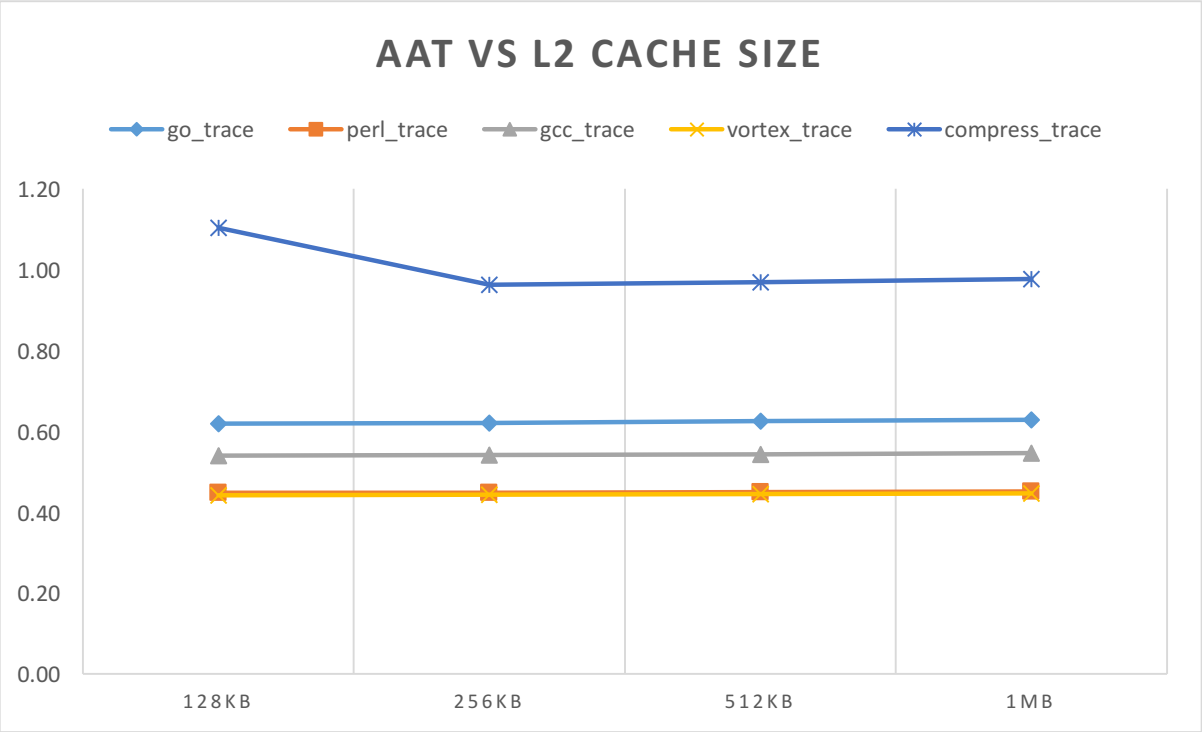
To do further analysis on the cache performance, Average access times are computed based on the data used from CACTI simulations.

AATL2 = HitTimeL1 + (MissRateL1 * HitTimeL2) + (MissRateL1 * MissRateL2 * MissPenalty)

All of the static values are taken from the CACTI simulator sheet.

The following is the computed AAT for different configuration taken for plot #4.

	go_trace	perl_trace	gcc_trace	vortex_trace	compress_trace
128KB	0.62	0.45	0.54	0.44	1.10
256KB	0.62	0.45	0.54	0.44	0.96
512KB	0.62	0.45	0.54	0.44	0.97
1MB	0.63	0.45	0.55	0.45	0.98

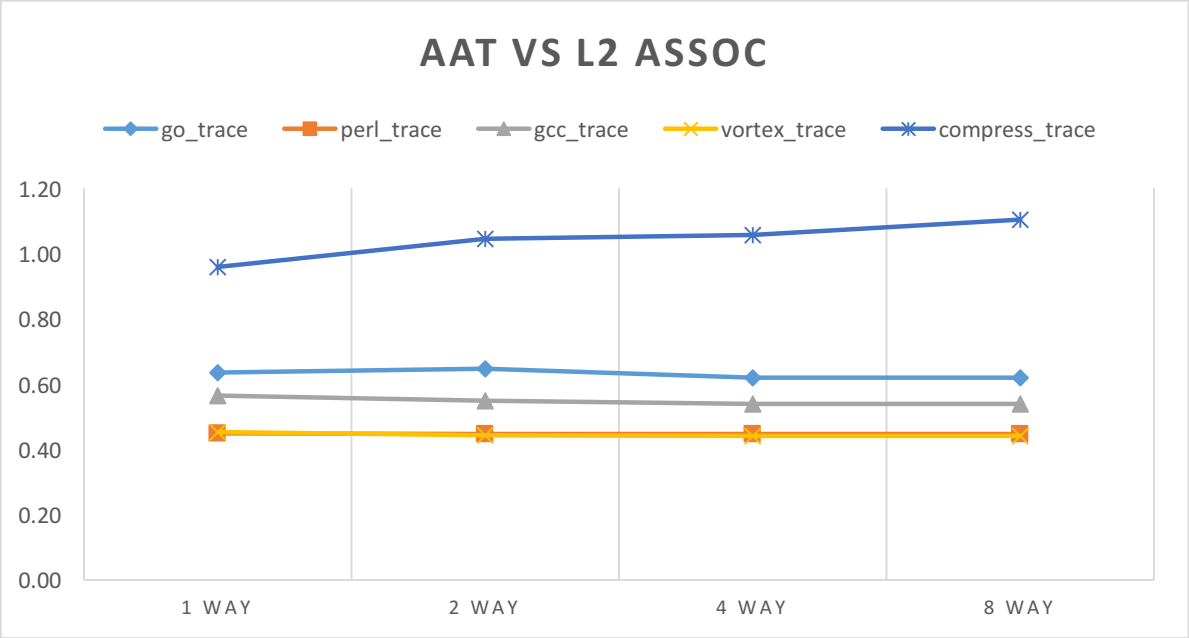


Observations:

- As we vary the size of L2 cache, there is no significant difference in the average access time for most of the benchmarks.
- In case of compress benchmark, we see a large drop in the AAT.

Similarly, we compute the AAT by for different cache configuration taken under consideration in plot #5

	go_trace	perl_trace	gcc_trace	vortex_trace	compress_trace
1 way	0.64	0.45	0.57	0.45	0.96
2 way	0.65	0.45	0.55	0.44	1.04
4 way	0.62	0.45	0.54	0.44	1.06
8 way	0.62	0.45	0.54	0.44	1.10



Observations:

- As we vary the assoc of L2 cache, there is no significant difference in the average access time for most of the benchmarks.
- In case of compress benchmark, we see a gradual but significant increase in the average access time.

Area considerations:

The area increases as we increase the size of the cache as well as if we increase the assoc of the cache. Taking this under account & considering the fact that smaller area (die area) results lower cost, we should consider cache structure such that it has as small as necessary for greater performance. Following is the plot that specifies area for different cache configuration.
These area values will help in choosing/recommending cache structure.

	128KB	256KB	512KB	1MB
1	0.657687839	1.277806565	2.487864981	3.747960026
2	0.694245498	1.562127169	2.225827274	4.349252234
4	0.667017966	1.141294802	2.177361671	4.673162925
8	0.559933334	1.293540536	2.640142073	4.874201405

Recommendation:

- If we are running & considering cache architecture for processor, then we should recommend 128KB 8-way cache as miss rates/AAT remains almost same for all

the different configuration. And the fact that the area required is less by a significant amount (approx. 0.055).

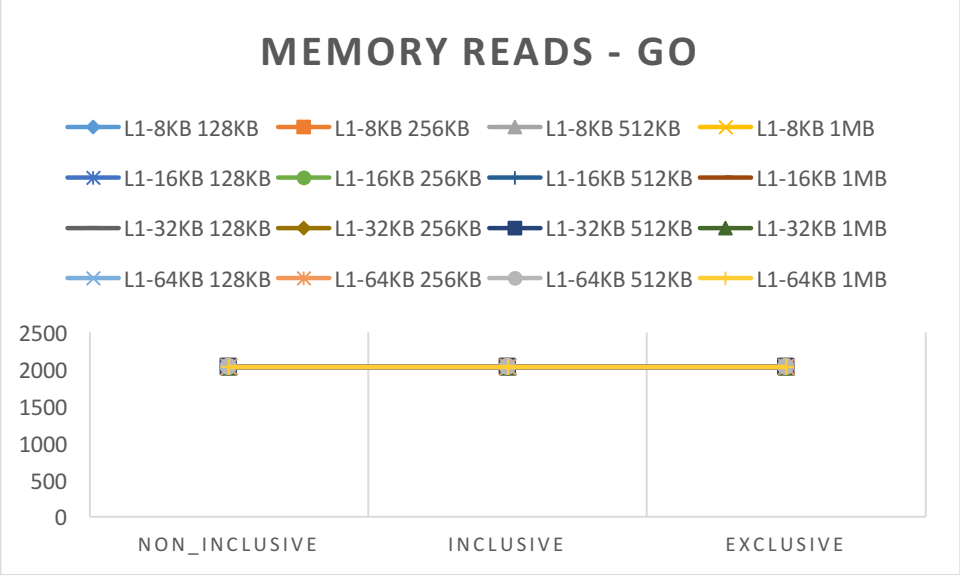
- Miss rate is of 48% for perl benchmark.
- AAT is of 0.45.

Plot #6

Plot the number of cache misses resulting in a memory read for each configuration for each inclusion policy (inclusive, exclusive and non-inclusive). Note that for each benchmark this plot with have 3 data series (one per inclusion policy) and each data series will have 16 points (one per configuration).

We study the read misses for L2 cache aize

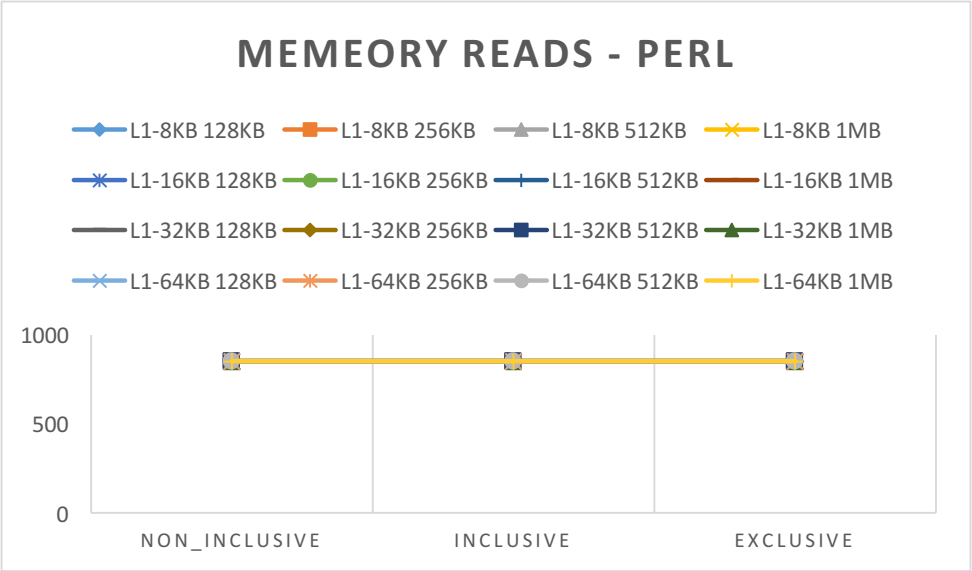
Plot #6(a) – Go Benchmark



Observation:

- The memory reads remain the same for all configuration.

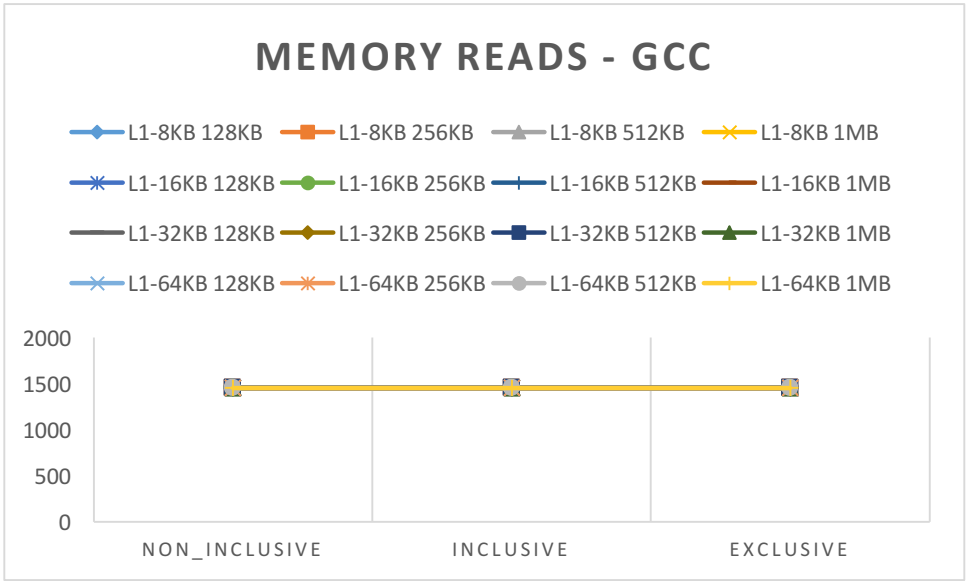
Plot #6(a) – Perl Benchmark



Observation:

- The memory reads remain the same for all configuration.

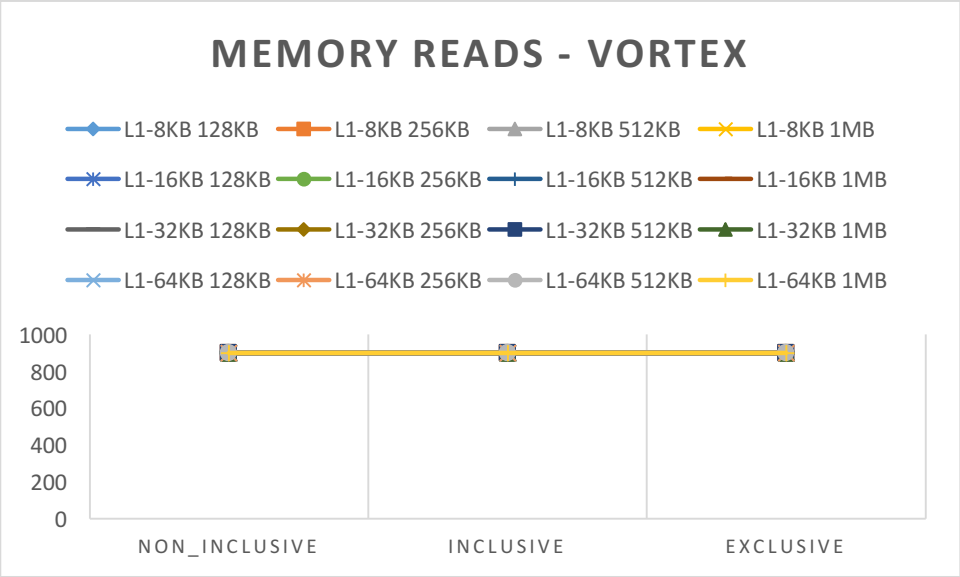
Plot #6(a) – GCC Benchmark



Observation:

- The memory reads remain the same for all configuration.

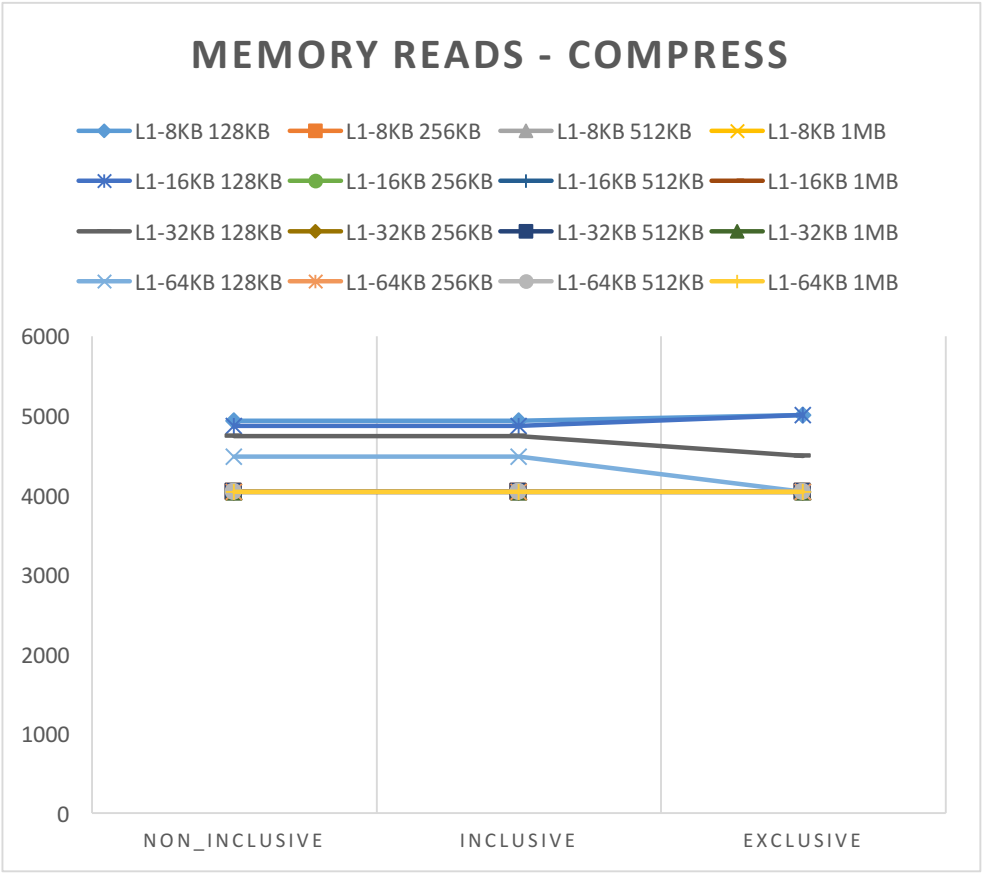
Plot #6(a) – Vortex Benchmark



Observation:

- The memory reads remain the same for all configuration.

Plot #6(a) – Compress Benchmark



Observation:

- In case of larger L1 (typically 32/64KB), and smaller L2 (typically 128KB), the memory reads are less in the case of Exclusive caches.
- In case of smaller L1 (typically of 8/16KB) and smaller L2 (128KB), the memory reads are more in case of Exclusive cache as compared to Inclusive or Non-Inclusive.
- The Memory reads remain the same in case of Inclusive & Non-Inclusive cache.

Implications:

- In case of Exclusive, since L1 content cannot be present in L2, L2 can have large amount of unique data.
- If the size of L1 is large and L2 is small, then in case of Non-Inclusive & Inclusive, much of the L2 will be taken by the data present in L1. This may result in eviction of some useful data from L2. (Useful meaning the data that may be referenced later).
- This results in higher number of memory reads for Non-Inclusive & Inclusive cache.

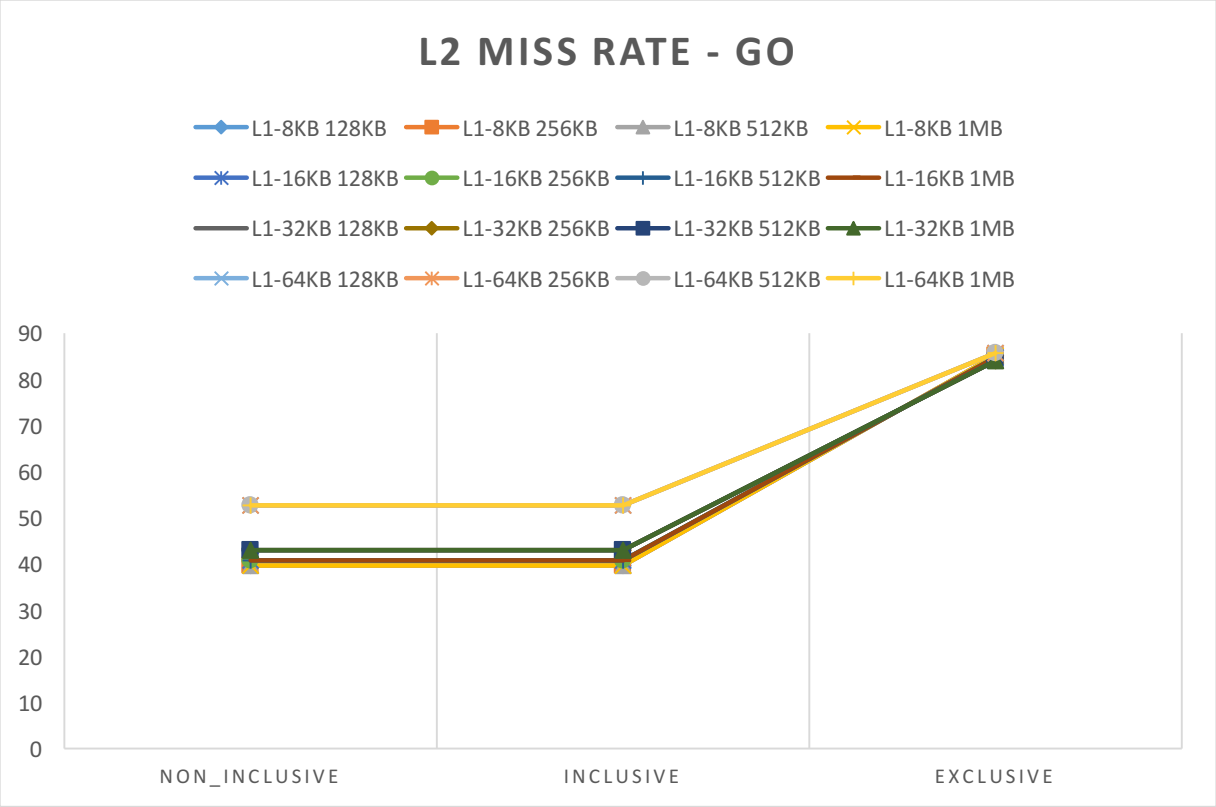
Plot #7

Assume that the AAT for Exclusive caches can be calculated similarly to the AAT calculation for non- inclusive caches. Even if blocks brought in from memory are directly installed in the L1 cache (no insertion time at L2 cache), we still encounter the hit latency of L2 cache as a result of checking the L2 cache before reading from memory. Plot the AAT when using non-inclusive, inclusive and exclusive L2 caches for each configuration. The number of data points will be similar to Plot 6.

For this experiment we consider different configuration given in the spec sheet. The miss rates for these configuration is tabulated as follows.

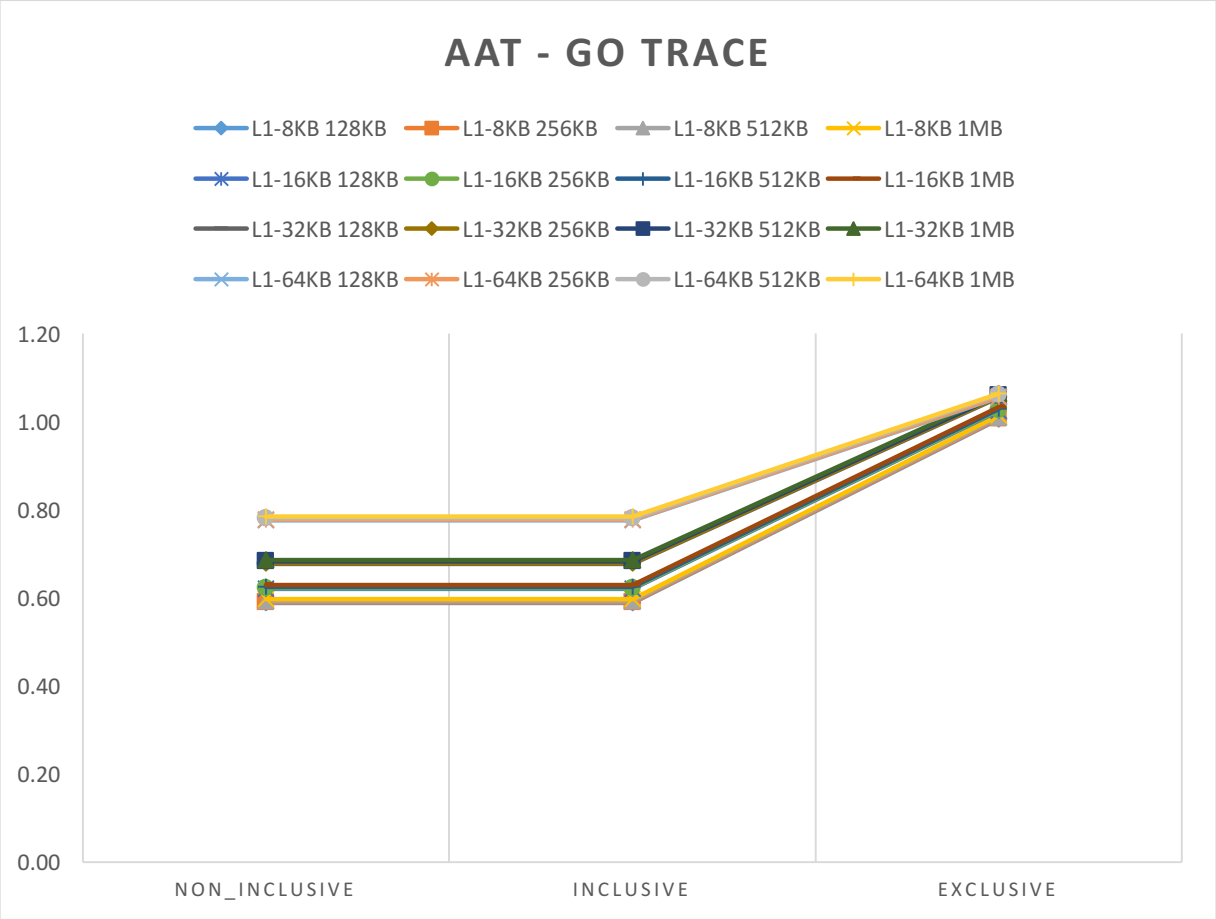
For each of the benchmark, plot #1 would refer to the miss rates for different configuration whereas, plot #2 would refer to the AAT required. For each of the benchmark we would make sufficient observation and finally we would recommend a cache configuration that will be better in terms of configuration. Note: we would not consider area required for the cache.

Plot #7(a) – Go benchmark



Observation:

- Miss rate remains the same for the case of Non-inclusive and Inclusive cache configuration.
- Exclusive cache configuration results in higher cache miss rate as compared to that of Non-inclusive & Inclusive caches.
- As we increase the size of L1, the L2 miss rates increases by a significant factor.



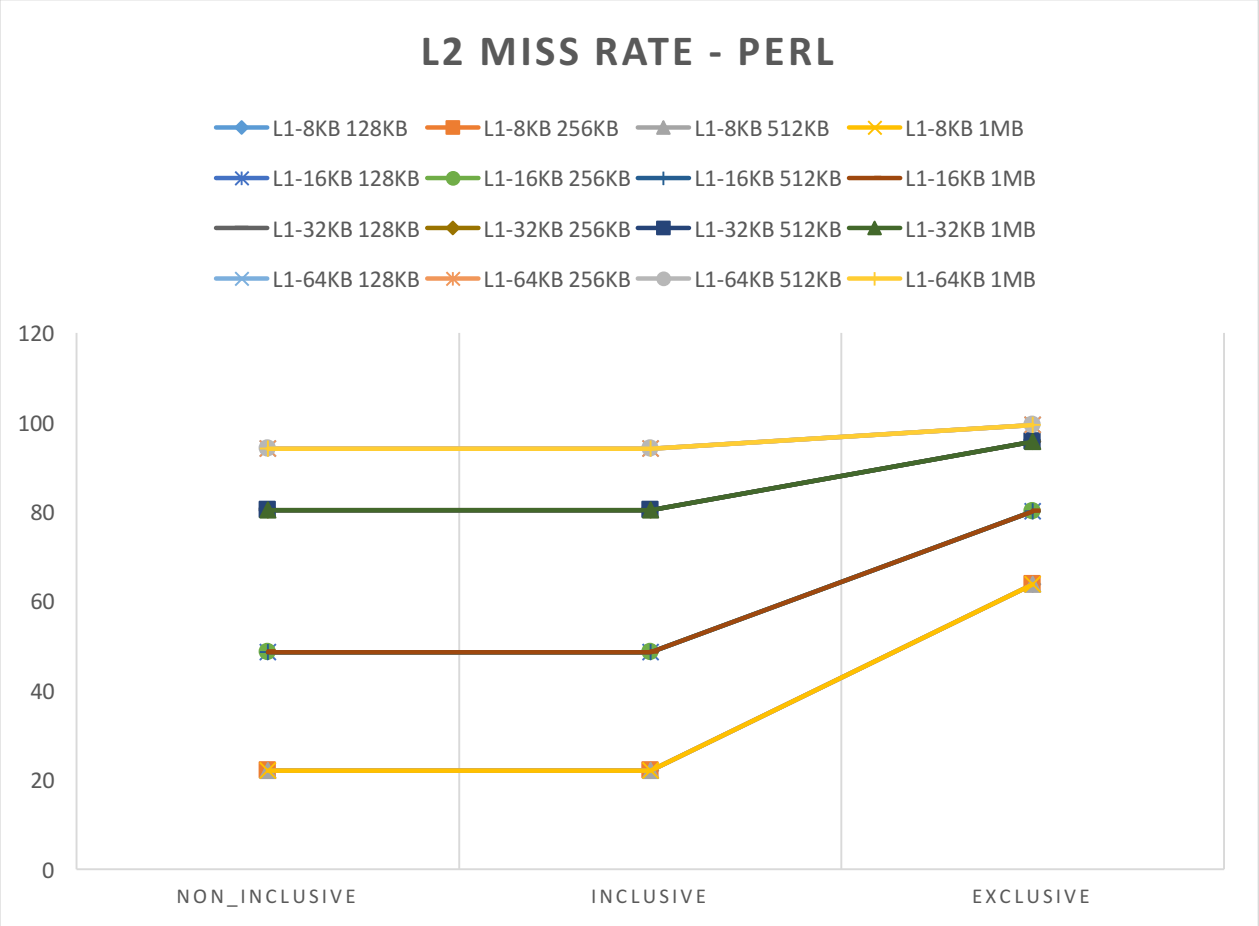
Observation:

- AAT remains the same for the case of Non-inclusive and Inclusive cache configuration mainly because both have same miss rates.
- Exclusive cache configuration results in higher cache miss rate as compared to that of Non-inclusive & Inclusive caches, hence the average access time is also greater.

Plot #7(b) – Perl benchmark

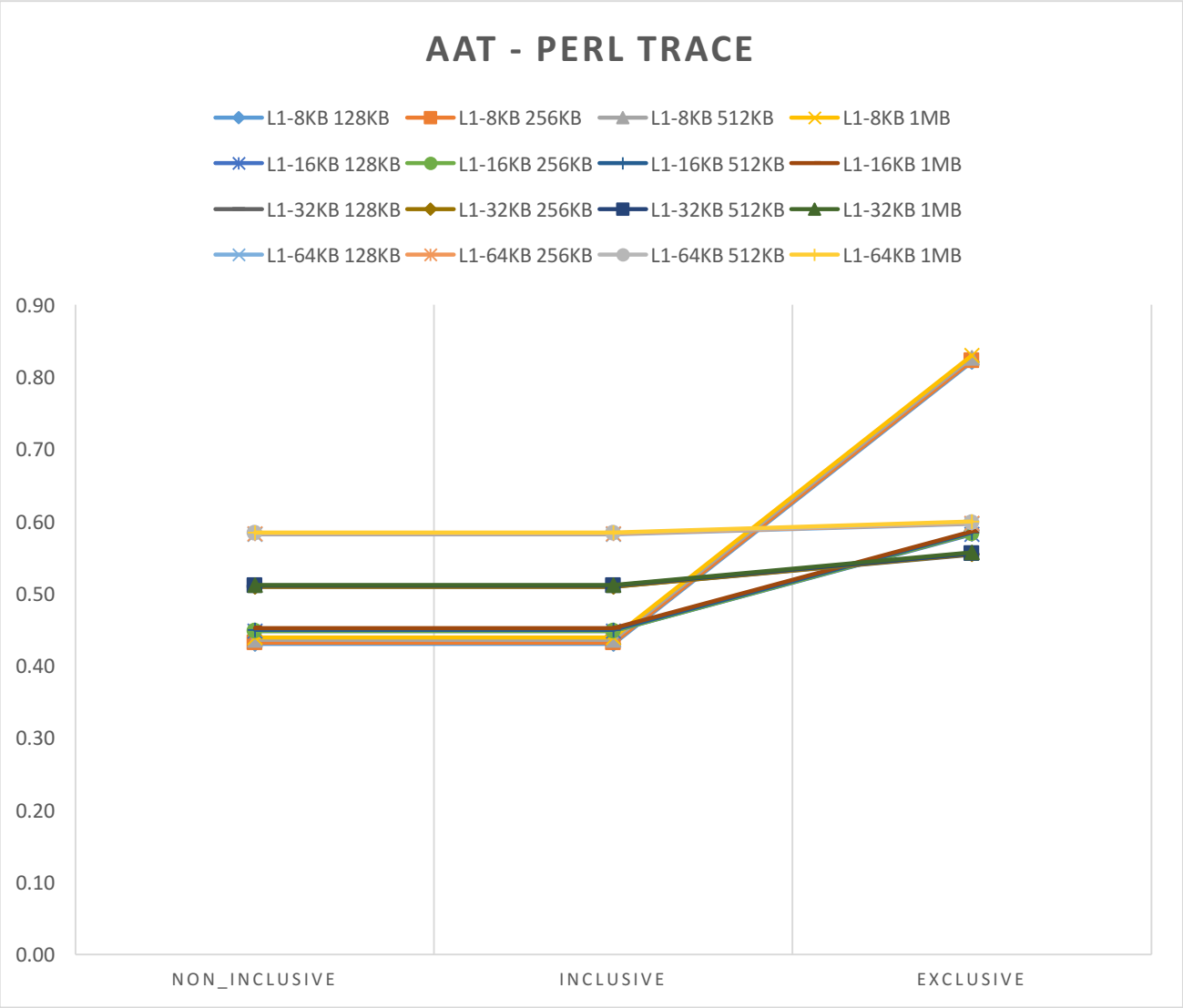
perl_trace	L1-8KB	L1-16KB	L1-32KB	L1-64KB

	128KB	256KB	512KB	1MB	128KB	256KB	512KB	1MB	128KB	256KB	512KB	1MB	128KB	256KB	512KB	1MB
NON_INC	22.10	22.10	22.10	22.10	48.61	48.61	48.61	48.61	80.34	80.34	80.34	80.34	94.05	94.05	94.05	94.05
INCL	22.10	22.10	22.10	22.10	48.61	48.61	48.61	48.61	80.34	80.34	80.34	80.34	94.05	94.05	94.05	94.05
EXCL	63.79	63.79	63.79	63.79	80.02	80.02	80.02	80.02	95.72	95.72	95.72	95.72	99.39	99.39	99.39	99.39



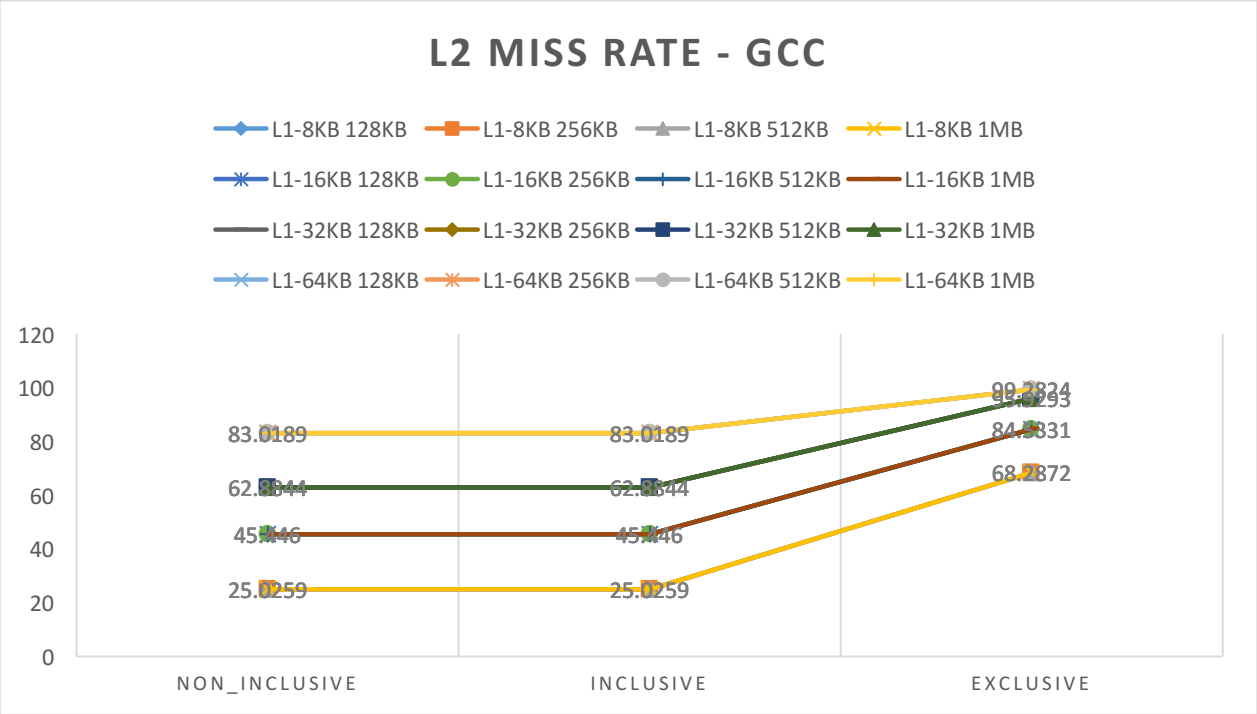
Observation:

- Miss rate remains the same for the case of Non-inclusive and Inclusive cache configuration.
- Exclusive cache configuration results in higher cache miss rate as compared to that of Non-inclusive & Inclusive caches.
- As we increase the size of L1, the L2 miss rates increases by a significant factor.



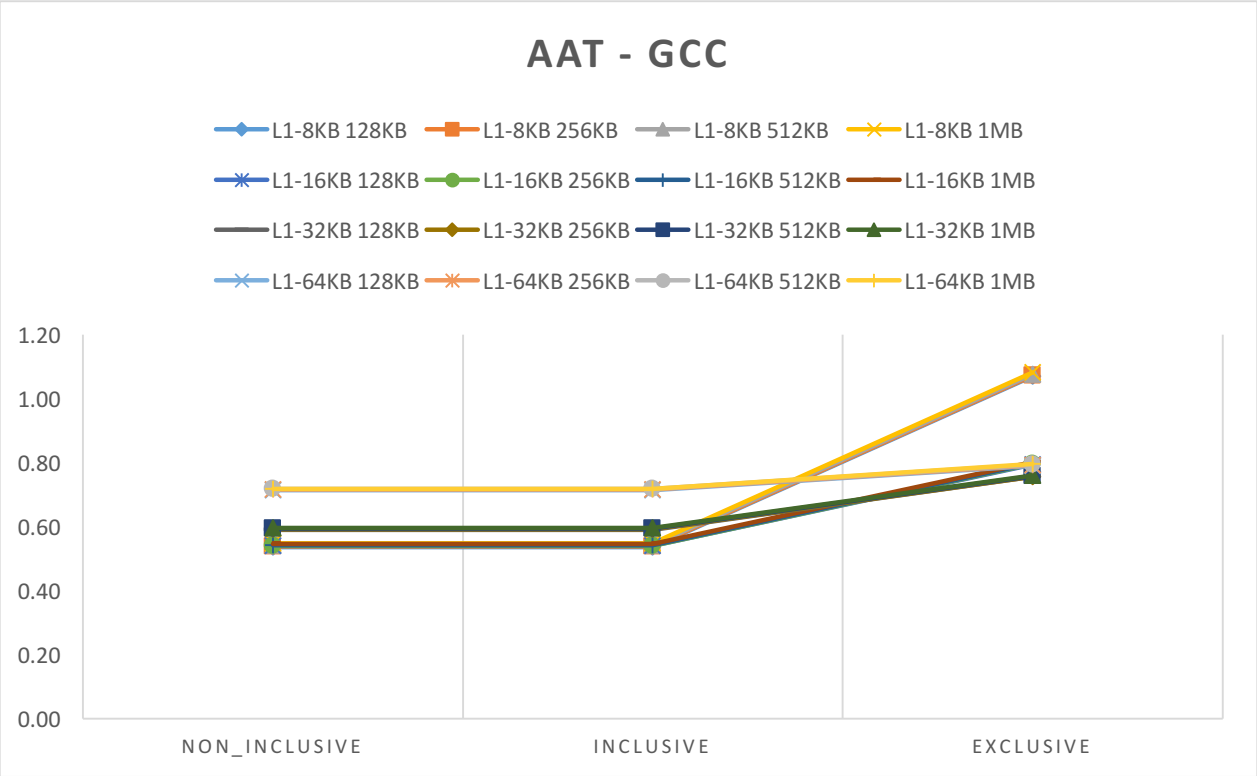
Plot #7(c) – GCC benchmark

gcc_trace	L1-8KB				L1-16KB				L1-32KB				L1-64KB			
	128 KB	256 KB	512 KB	1M B	128 KB	256 KB	512 KB	1M B	128 KB	256 KB	512 KB	1M B	128 KB	256 KB	512 KB	1M B
NON_INCLUSIVE	25.025259	25.025259	25.025259	25.025259	45.446	45.446	45.446	45.446	62.844	62.844	62.844	62.844	83.0189	83.0189	83.0189	83.0189
INCLUSIVE	25.025259	25.025259	25.025259	25.025259	45.446	45.446	45.446	45.446	62.844	62.844	62.844	62.844	83.0189	83.0189	83.0189	83.0189
EXCLUSIVE	68.2872	68.2872	68.2872	68.2872	84.5331	84.5331	84.5331	84.5331	95.9293	95.9293	95.9293	95.9293	99.2824	99.2824	99.2824	99.2824



Observation:

- Miss rate remains the same for the case of Non-inclusive and Inclusive cache configuration.
- Exclusive cache configuration results in higher cache miss rate as compared to that of Non-inclusive & Inclusive caches.
- In case of 8KB L1 non-inclusive/inclusive cache outperform exclusive caches by a factor of approx. 2
- As we increase the size of L1, the L2 miss rates increases by a significant factor.



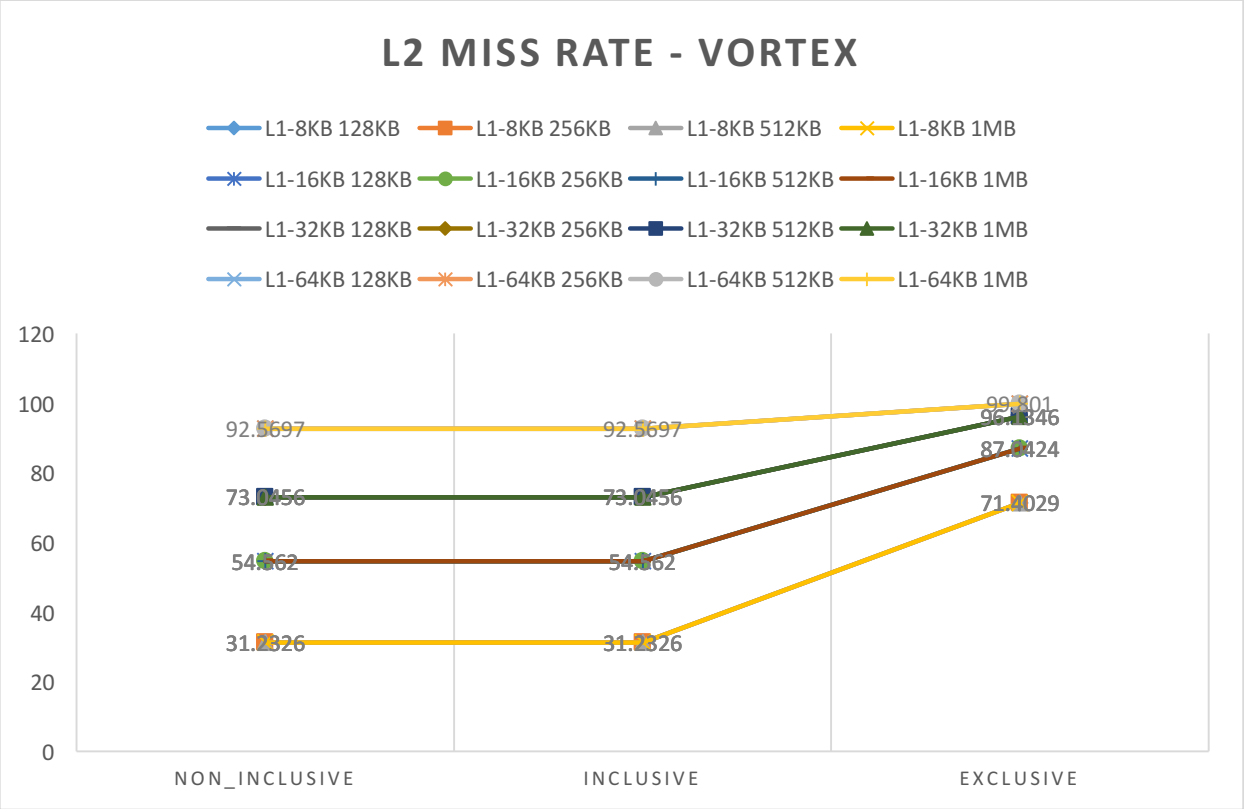
Observation:

- AAT remains the same for the case of Non-inclusive and Inclusive cache configuration mainly because both have same miss rates.

Exclusive cache configuration results in higher cache miss rate as compared to that of Non-inclusive & Inclusive caches, hence the average access time is also greater.

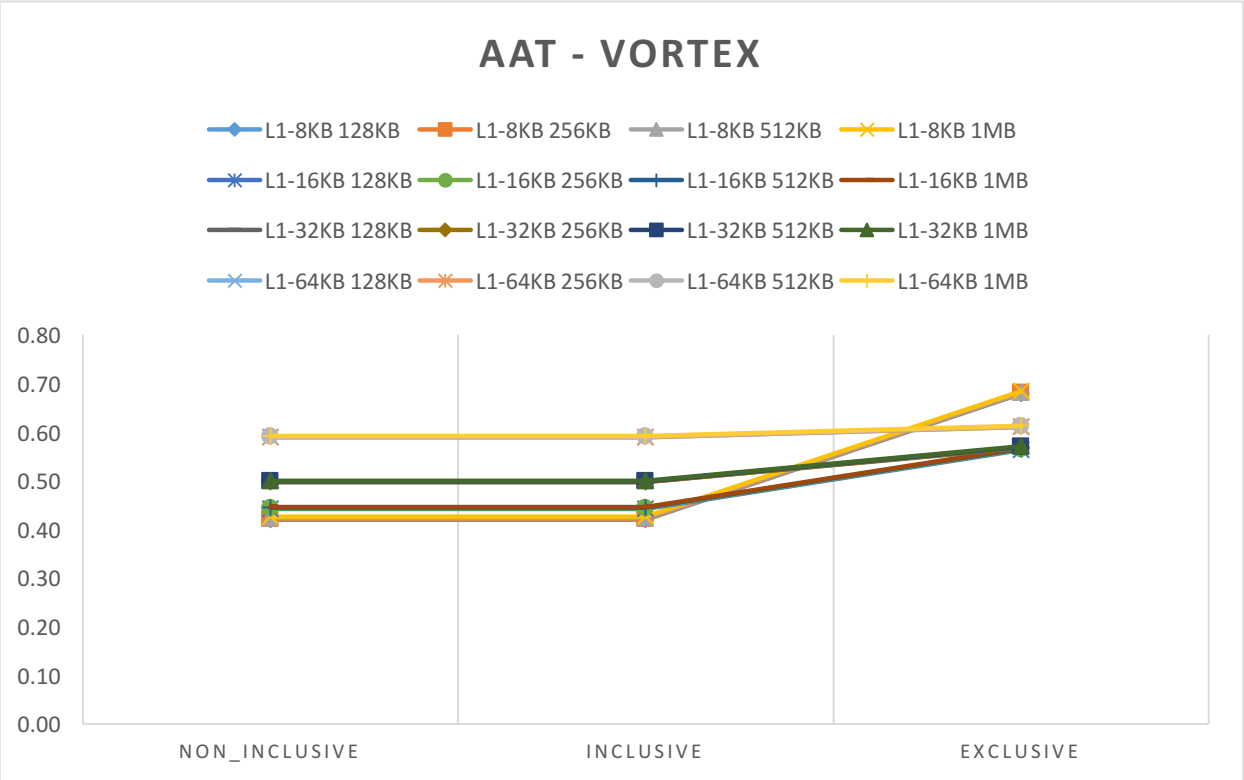
Plot #6(d) & #7(d) – Vortex benchmark

vortex_trace	L1-8KB				L1-16KB				L1-32KB				L1-64KB			
	128KB	256KB	512KB	1MB	128KB	256KB	512KB	1MB	128KB	256KB	512KB	1MB	128KB	256KB	512KB	1MB
NON_INCLUSIVE	31.232326	31.232326	31.232326	31.232326	54.562	54.562	54.562	54.562	73.0456	73.0456	73.0456	73.0456	92.5697	92.5697	92.5697	92.5697
INCLUSIVE	31.232326	31.232326	31.232326	31.232326	54.562	54.562	54.562	54.562	73.0456	73.0456	73.0456	73.0456	92.5697	92.5697	92.5697	92.5697
EXCLUSIVE	71.4029	71.4029	71.4029	71.4029	87.0424	87.0424	87.0424	87.0424	96.1346	96.1346	96.1346	96.1346	99.801	99.801	99.801	99.801



Observation:

- Miss rate remains the same for the case of Non-inclusive and Inclusive cache configuration.
- Exclusive cache configuration results in higher cache miss rate as compared to that of Non-inclusive & Inclusive caches.
- In case of 8KB L1 & 1MB L2, non-inclusive/inclusive cache outperform exclusive caches by a factor of approx. 2
- As we increase the size of L1, the L2 miss rates increases by a significant factor.
- For 64 KB L1, the miss rate of L2 is close to 90%.



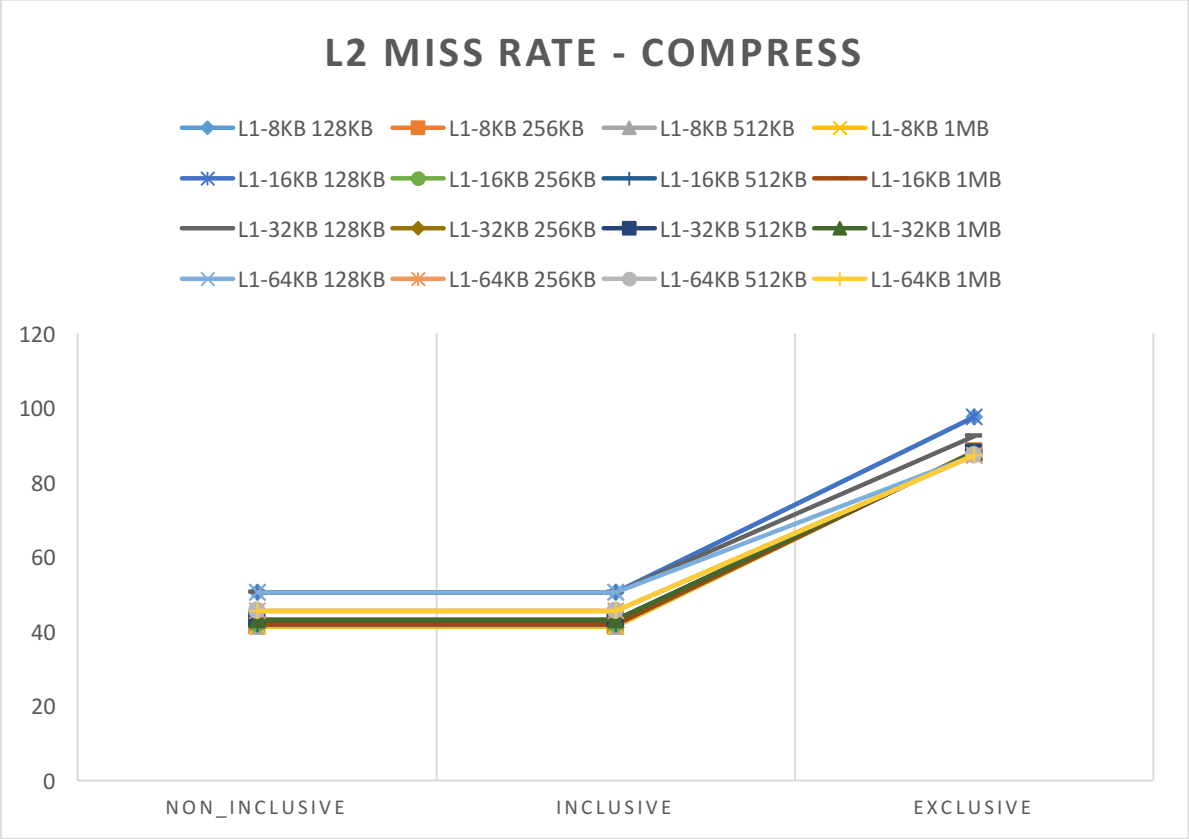
Observation:

- AAT remains the same for the case of Non-inclusive and Inclusive cache configuration mainly because both have same miss rates.

Exclusive cache configuration results in higher cache miss rate as compared to that of Non-inclusive & Inclusive caches, hence the average access time is also greater.

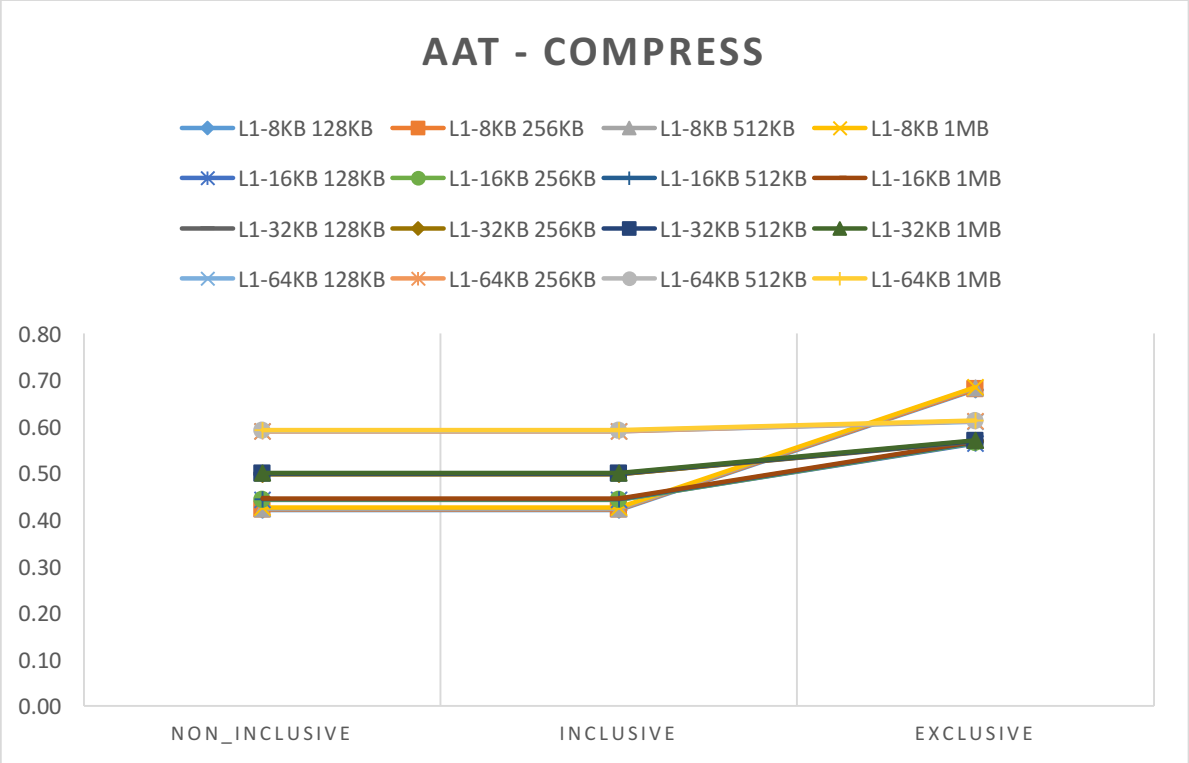
Plot #6(e) & #7(e) – Compress benchmark

compress	L1-8KB				L1-16KB				L1-32KB				L1-64KB			
	128 KB	256 KB	512 KB	1MB	`	256 KB	512 KB	1MB	128 KB	256 KB	512 KB	1MB	128 KB	256 KB	512 KB	1MB
NON_INCLUSIVE	50.414	41.337	41.337	41.337	50.4195	41.885	41.885	41.885	50.4309	43.0259	43.0259	43.0259	50.4557	45.5047	45.5047	45.5047
INCLUSIVE	50.414	41.337	41.337	41.337	50.4195	41.885	41.885	41.885	50.4309	43.0259	43.0259	43.0259	50.4557	45.5047	45.5047	45.5047
EXCLUSIVE	97.5558	88.2987	88.2987	88.2987	97.5253	88.1528	88.1528	88.1528	92.3461	87.8497	87.8497	87.8497	87.1946	87.1946	87.1946	87.1946



Observation:

- Miss rate remains the same for the case of Non-inclusive and Inclusive cache configuration.
- Exclusive cache configuration results in higher cache miss rate as compared to that of Non-inclusive & Inclusive caches.
- Miss rates becomes almost double in case of Exclusive cache.



Observation:

- AAT remains the same for the case of Non-inclusive and Inclusive cache configuration mainly because both have same miss rates.

Exclusive cache configuration results in higher cache miss rate as compared to that of Non-inclusive & Inclusive caches, hence the average access time is also greater.

Implications:

- The general observation is that the miss rate is less in the case of non inclusive caches & inclusive caches for wide range of benchmarks.
- Miss rate is same for inclusive as well as non-inclusive cache.
- In case of exclusive caches, the miss rate is higher.
- Average access time required is also greater in case of exclusive caches.
- As we increase the size of L1, miss rates in L1 cache decreases, but miss rate in L2 increase.
- This may be explained by the fact that L1 is able to address most of the request except for the cold misses. These accesses are compulsory miss and will also miss in L2.
- It can be argued that misses in L2 may have significant amount of cold misses in case of 64KB L1.

Non Inclusive vs Inclusive:

- Inclusive cache requires back invalidations, which in-turn put pressure on L1

- Non-Inclusive cache on the other hand doesn't require any back-invalidation, hence there is no extra traffic from L2 to L1.
- Inclusive caches will be better in the case where there are multiple processors and each of them have their own local cache (Not shared). If there is cache coherence problem, it just has to look into the lower level of the cache.
- Since we are considering multiprocessor configuration, this deduction is out of scope.
- In case of single processor configuration, as we have seen based on our experiments, the miss rates are same.
- Consider a case where L1 cache is cache is large and L2 cache is small.
- Inclusion property specifies that content of L1 should be always be present in L2. So effectively L2 can hold less unique data. In case of smaller L2 cache, this usually creates a problem. So we might see a larger L2 cache miss rates. We should recommend in using Non-Inclusive cache instead.
- Or in other words: Non-Inclusive cache will outperform Inclusive cache for this case.

Exclusive vs Non-Inclusive:

- Consider the case where the L1 cache size is large enough (typically of 64KB) while the size of L2 is small (typically of 128 KB). In the case of Non-Inclusive, any read in L1 would result in bringing the block in L1 as well as L2.
- In case of Exclusive cache, any request in L1 would only result in bringing the block in L1 and not in L2.
- Since, L2 doesn't have the blocks that were present in L1, L2 in turn has large space for unique data.
- Since L2 is small & L2 may contain blocks that were present in L1, any eviction in L1 would result in allotting a space in L2. Now if there is a capacity miss in L2 on the block that was evicted this would result in memory read.
- This capacity miss would less probably occur in case of Exclusive as L2 can hold whole 128 KB of unique data. This means there is more space in L2.
- In summary, we should recommend using Exclusive cache in the case of smaller size of L2 with relatively larger L1.