ASSIGNMENT REPORT

Question 1)

Miss rate in L1 cache and L2 cache was calculated by varying

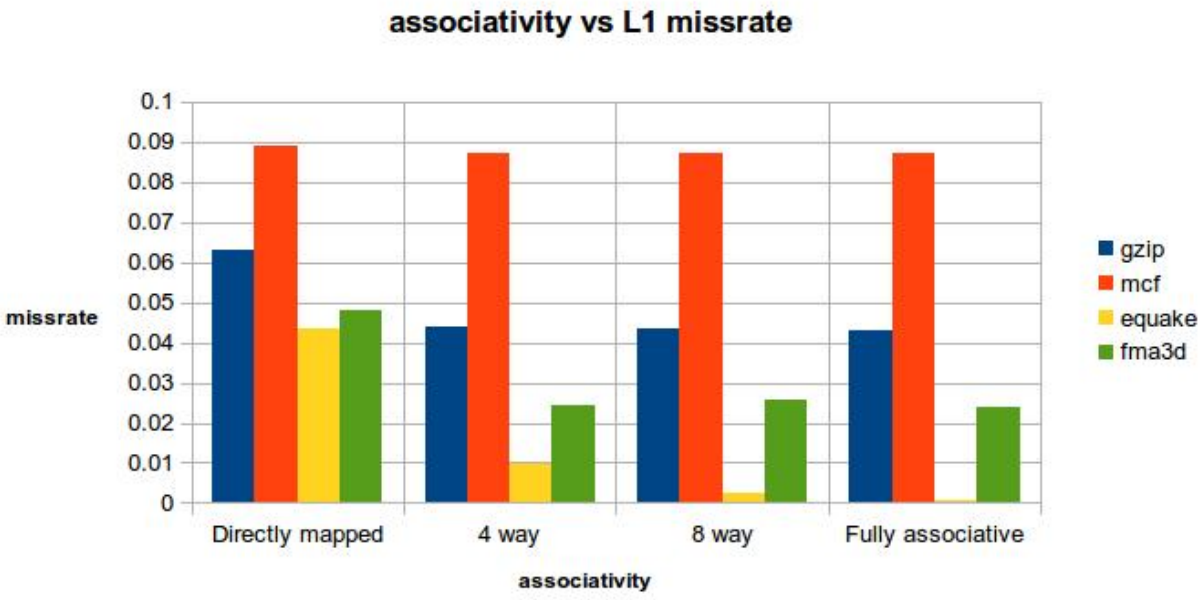
a) Associativity
b) Replacement Policies
c) Cache size
d) Block size

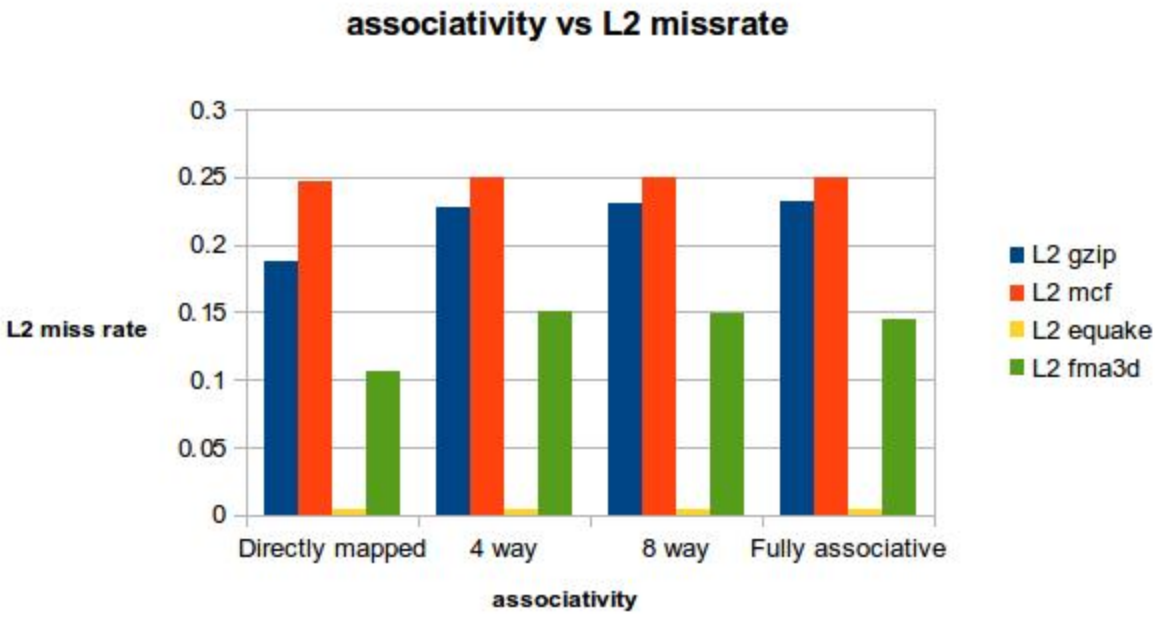
8 different graphs were plotted, 4 each for l1 and l2. The findings are as follows.

a) Associativity

Constants -

i) Replacement Policy - LRU
ii) Cache size - 4KB
iii) Block size - 32B





OBSERVATIONS

1) miss rate is a non decreasing function of associativity. ie., as the number of blocks in each set increases, miss rate declines. This is understandable as there are more blocks into which common addresses can hash into. Hence, the reduced miss rate. The downside of this would be an increase in hit time. One exception is fma3d where the miss rate actually increases from 4 way to 8 way. This can be because of the nature of the instruction sets used by fma3d. although the increase is too negligible to be concerning.

2) In case of L2 cache though, changes in L1 associativity increases miss rates in certain cases (fma3d and gzip). This could be because, in case of increased hit in l1 cache, the number of access to l2 reduces proportionately. although, many of the reduced accesses were the hits. Misses in L2 barely changes since we have not modified its configuration. hence, the same miss count to the reduced instruction access count would project a higher miss rate.

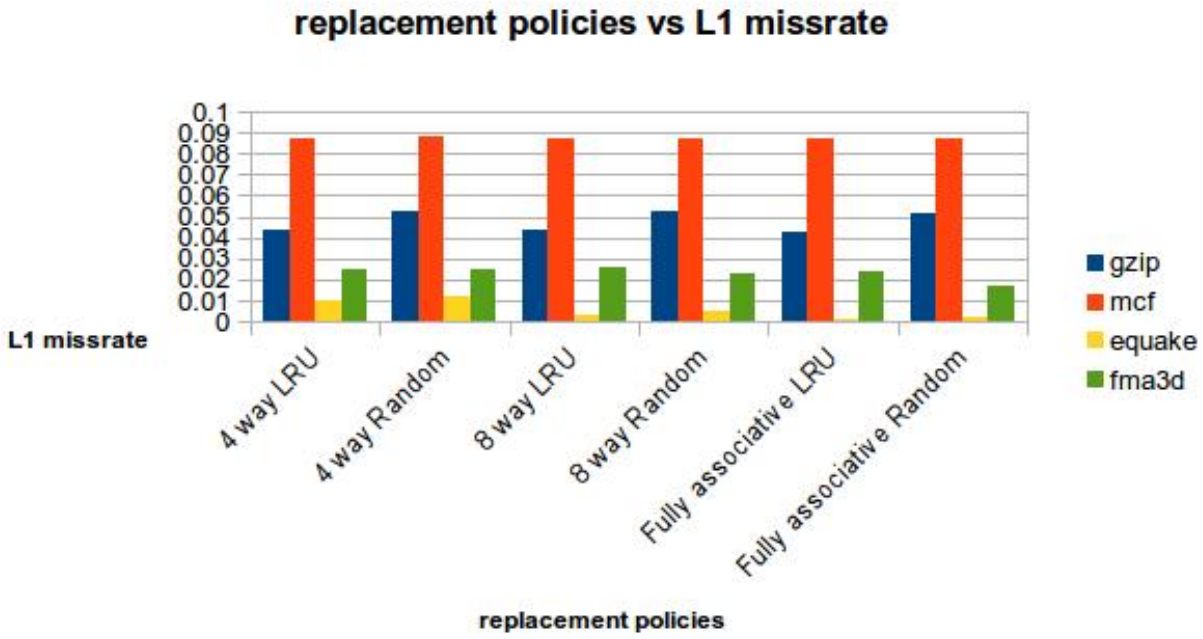
3) Miss rate is generally higher in L2 than in L1 cache.

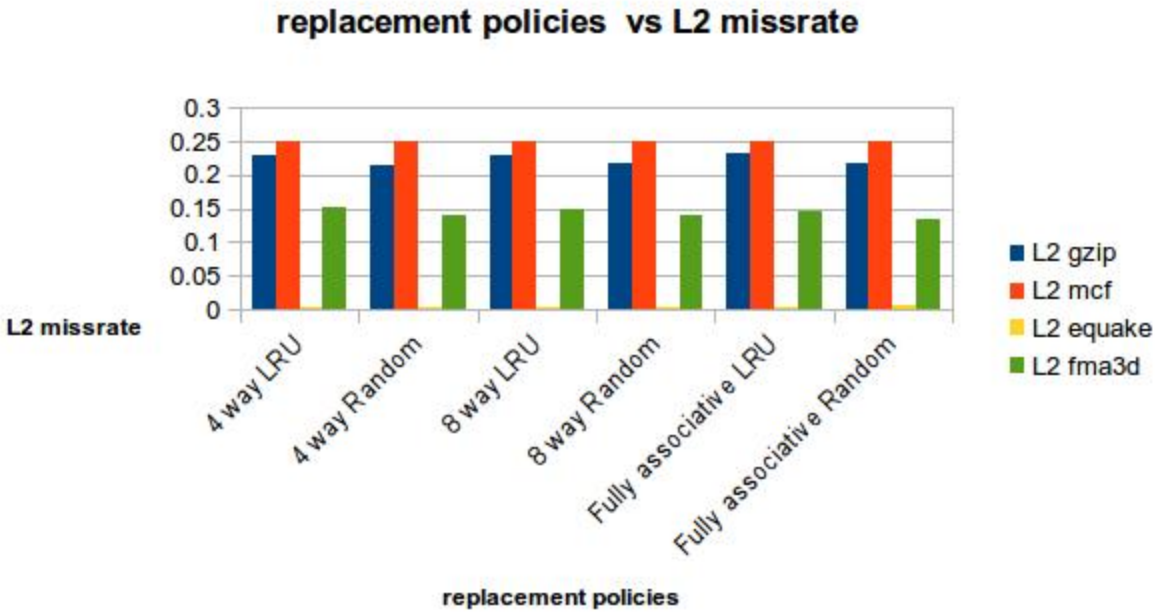
b) Replacement Policies with associativity (except directly mapped)

Constants -

ii) Cache size - 4KB

iii) Block size - 32B





OBSERVATIONS and INFERENCES

1) For Least Recently Used policy, the miss rate decreases as the associativity increases. This is because, bigger the set size, better the chances that the least recently used ones are not going to be referenced again. In a 2 way associativity though, LRU will kick out the block that could be only two cycles old. In equake, this effect is pronounced. Other benchmarks are all the same for LRU with all associativity. This can be because either there were not too many replacements / the instruction might not have repeated itself much. In general though, we can conclude that **LRU works well with higher associativity**

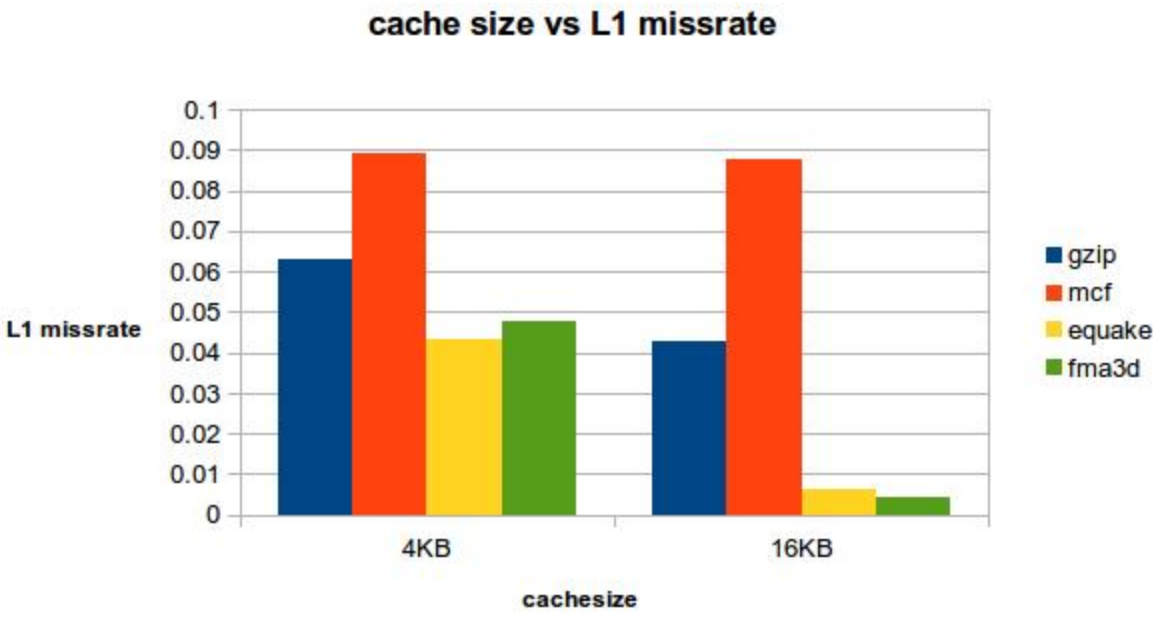
2) The progression of random replacement policy is also similar to LRU. ie., whenever LRU does bad from one associativity to another, random also does worse and whenever LRU does better, random does better. The observation here is the magnitude by which the replacement policy improves / degrades. while LRU moves up / down considerably, Random policy changes ever so slightly in most cases.

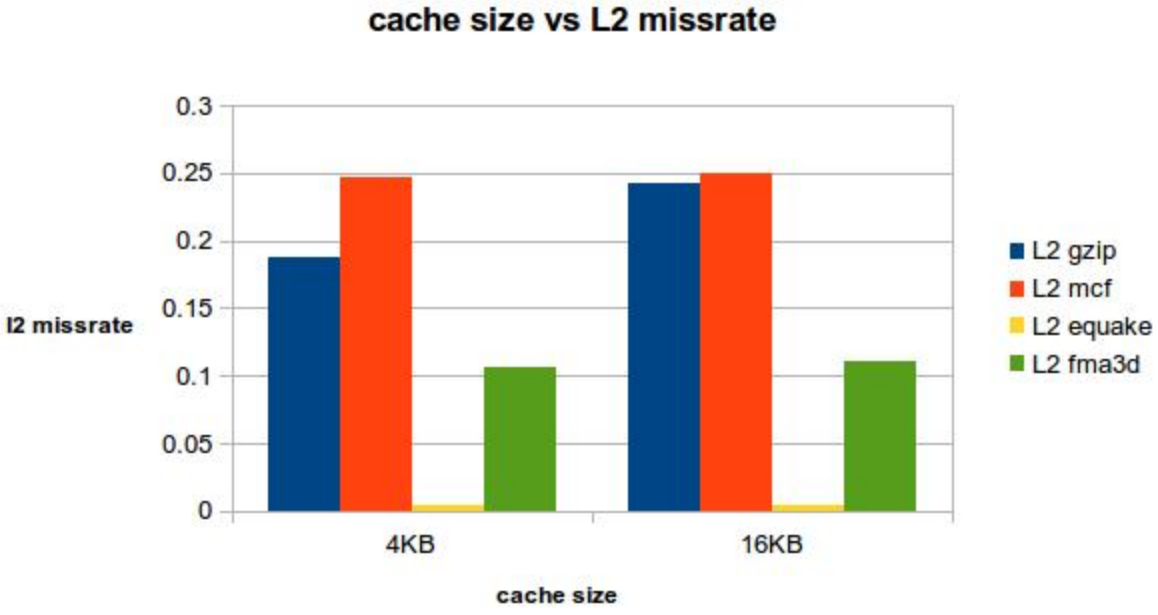
3) For gzip and equake, LRU seems to be a better fit, while random policy fits better for fma3d. mcf remains unmoved. Could be reasoned again by the placement and frequency of instructions in these programs

c) Cache size

Constants -

i) Replacement Policy - LRU
ii) Associativity - Directly Mapped
iii) Block size - 32B





OBSERVATIONS and INFERENCES

1) When you increase the cache size, you give a lot more extra space. and hence probability of miss tends to go down, which we can clearly see from the above graph. The downside again would be a reduced hit time.

2) mcf remains steady with both the cache block sizes. There could be two reasons. The

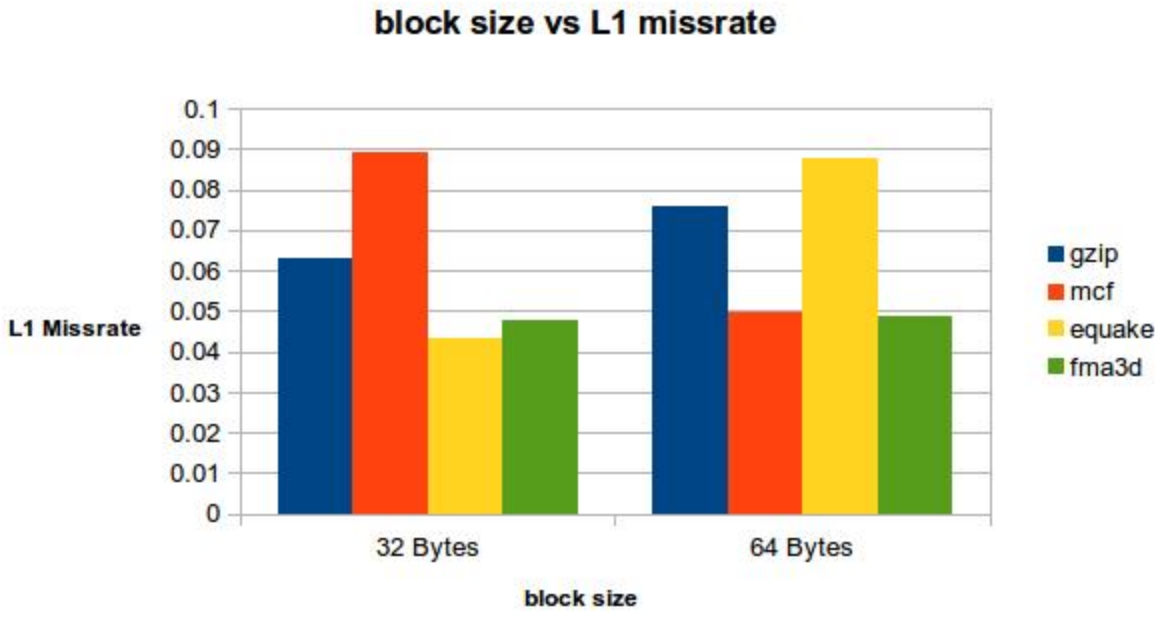
amount of data references is so frequent for so many instructions that no size of a cache could stop the misses from occurring. There's another scenario where, the instructions are well spread out and unique enough that a smaller cache does beautifully well to accommodate all the instructions as long as it is necessary. Hence additional time is not going to reduce miss rate since miss rate is already near zero. The former is more likely because of a higher miss rate exhibited by the benchmark

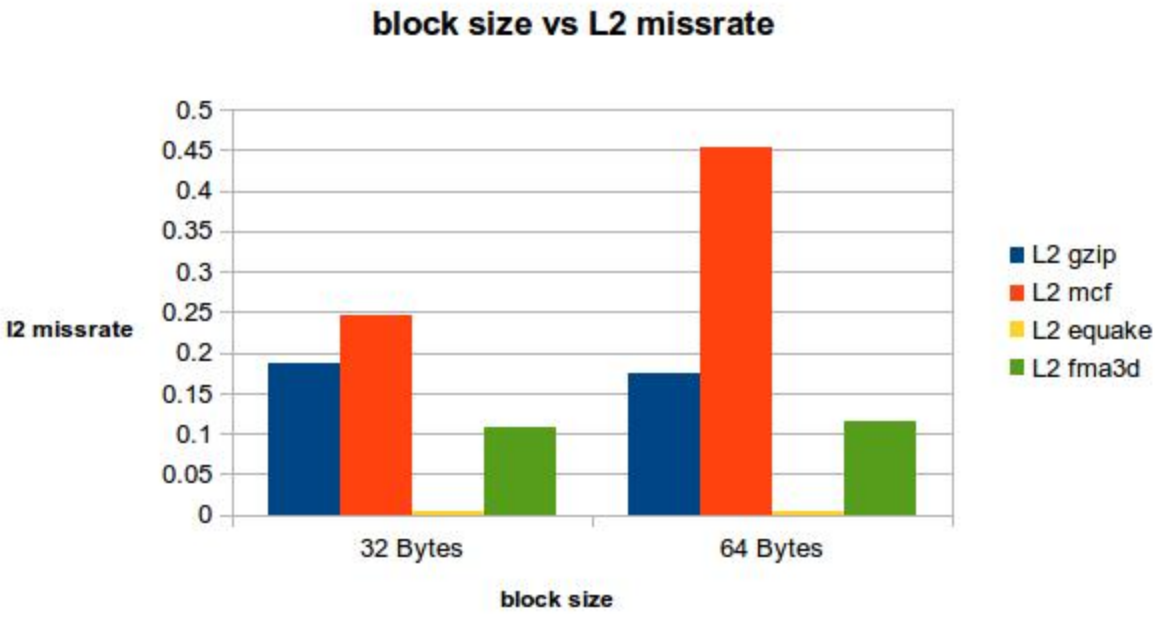
3) In L2 cache, for gzip alone, the miss rate increases with increase in cache block size. Note that the cache block size increased was not of L2, but of L1. L2's properties remains the same as ever. But since L1 is able to hold a lot more data now, L2 experience more misses than hits and that is why we see such a trend here.

d) Block size

Constants -

i) Replacement Policy - LRU
ii) Associativity - Directly Mapped
iii) Cache size - 4 KB





OBSERVATIONS and INFERENCES

1) The miss rate is more fluctuating between benchmarks in correspondence to the block size. This can be attributed to locality. Some programs might be heavily sequential which gets benefitted when a large block from contiguous locations are brought into the cache. other programs might be data driven which does not go sequentially in its way. In this case, bring a large chunk of data is polluting the cache. This is why we see equake and gzip shoot up while mcf attains some level of advantage from the increased block size.

2) Although, what is consistent is that, when the L1 miss rate decreases, L2's miss rate shoots up, as with the case of equake. This sort of behavior has been consistent throughout our simulation so far.

SIMULATION RESULTS

Here are few tabular results from simulation which is used in the graphs depicted above.

a) Variations due to changes in associativity. (cache size, replacement policies and block size are kept constant)

	gzip	mcf	equake	fma3d
Directly mapped	0.0632	0.089	0.0434	0.0478
4 way	0.0441	0.087	0.0098	0.0245
8 way	0.0434	0.087	0.0027	0.0257
Fully associative	0.0431	0.087	0.0006	0.024

b) Same as above (for L2 cache)

	L2 gzip	L2 mcf	L2 equake	L2 fma3d
Directly mapped	0.1881	0.2462	0.004	0.1068
4 way	0.2281	0.25	0.0038	0.151
8 way	0.2301	0.25	0.0041	0.1492
Fully associative	0.2318	0.25	0.0039	0.1449

c) Variations due to changes in replacement policies for various associativity. (cache size and block size are kept constant)

	gzip	mcf	equake	fma3d
4 way LRU	0.0441	0.087	0.0098	0.0245
4 way Random	0.0522	0.0879	0.0115	0.0252
8 way LRU	0.0434	0.087	0.0027	0.0257
8 way Random	0.0521	0.0877	0.0055	0.0231
Fully associative LRU	0.0431	0.087	0.0006	0.024
Fully associative Random	0.0518	0.0876	0.0019	0.0174

d) same as above (for L2 cache)

	L2 gzip	L2 mcf	L2 equake	L2 fma3d
4 way LRU	0.2281	0.25	0.0038	0.151
4 way Random	0.2149	0.2485	0.0044	0.1397
8 way LRU	0.2301	0.25	0.0041	0.1492
8 way Random	0.2161	0.2488	0.0047	0.1407
Fully associative LRU	0.2318	0.25	0.0039	0.1449
Fully associative Random	0.2158	0.249	0.0051	0.1347

e) Variations due to changes in cache size. (associativity, replacement policies and block size are kept constant)

	gzip	mcf	equake	fma3d
4KB	0.0632	0.089	0.0434	0.0478
16KB	0.043	0.0875	0.0063	0.0045

f) same as above (for L2 cache)

	L2 gzip	L2 mcf	L2 equake	L2 fma3d
4KB	0.1881	0.2462	0.004	0.1068
16KB	0.2418	0.2495	0.0043	0.1109

g) Variations due to changes in block size. (associativity, replacement policies and cache size are kept constant)

	gzip	mcf	equake	fma3d
32 Bytes	0.0632	0.089	0.0434	0.0478
64 Bytes	0.0757	0.0496	0.0878	0.0489

h) same as above (for L2 cache)

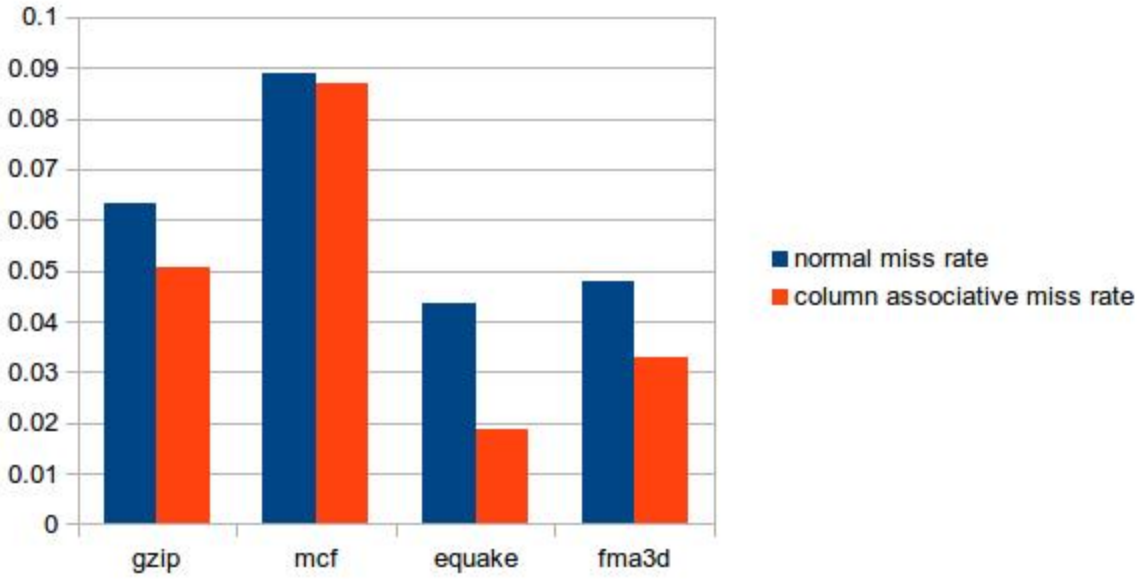
	L2 gzip	L2 mcf	L2 equake	L2 fma3d
32 Bytes	0.1881	0.2462	0.004	0.1068
64 Bytes	0.1737	0.4539	0.0036	0.1164

The actual log files are attached with the zipped attachment in the mail.

Question 2

In this question, we are going to consider miss rates only to compare column associative and regular caches. column associativity strives to improve the miss rate. it does not focus much on anything else. lets see how well it does.

Here's a simple graph depicting the whole scenario.



OBSERVATIONS and INFERENCES

1) Miss rates tend to decrease in all cases. which means the algorithm attains what it aims to. we see a drastic improvement in three of the benchmarks and a small improvement in mcf. This could be because of one of the following reasons.

a) Column associative cache tries to accommodate two instructions in the same set by rehashing and saving it in the same location. But if the instruction set continuously churns out three instructions in a loop, there's no way other than doing a memory access. mcf could be one such function where **more than 2 instructions loop of the same set around**.

b) Lack of **temporal locality** between memory locations that share same space. For eg, a and b could share same location, but they are not close enough to be effectively use the actual set and the rehashed set. lets say x actually belongs to the rehashed location of a. an access like a x b would introduce memory accesses and hence defeats the purpose, causing computational overhead.

SIMULATION RESULTS

	normal miss rate	column associative miss rate	improvement in missrate
gzip	0.0632	0.0505	20.0949367089
mcf	0.089	0.087	2.2471910112
equake	0.0434	0.0187	56.9124423963
fma3d	0.0478	0.033	30.9623430962

These are the simulation results in tabular format. The original log files are attached with the zipped code snippet. On a quick observation, 3 out of 4 benchmarks improve more than 20% in respect to miss rate!