1. Purpose

This report depict how changes in the L1 data cache configuration affect the utilization of the L1 data cache.

The parameters changed for L1 data cache includes:

- Cache size,
- Line(Block) size, (Size of the data retrieved when the miss occurs)
- Associativity

2. Expectation

Before, I start to make experiment. I made some guesses to find how the cache configuration changes affect the utilization of L1 data cache. In these guesses I thought:

1. If we will change the associativity of L1 data cache for N = 500. I expected to see not significant change because we already fulling the cache with our data. I thought that increasing associativity not give any additional power to utilize the cache.

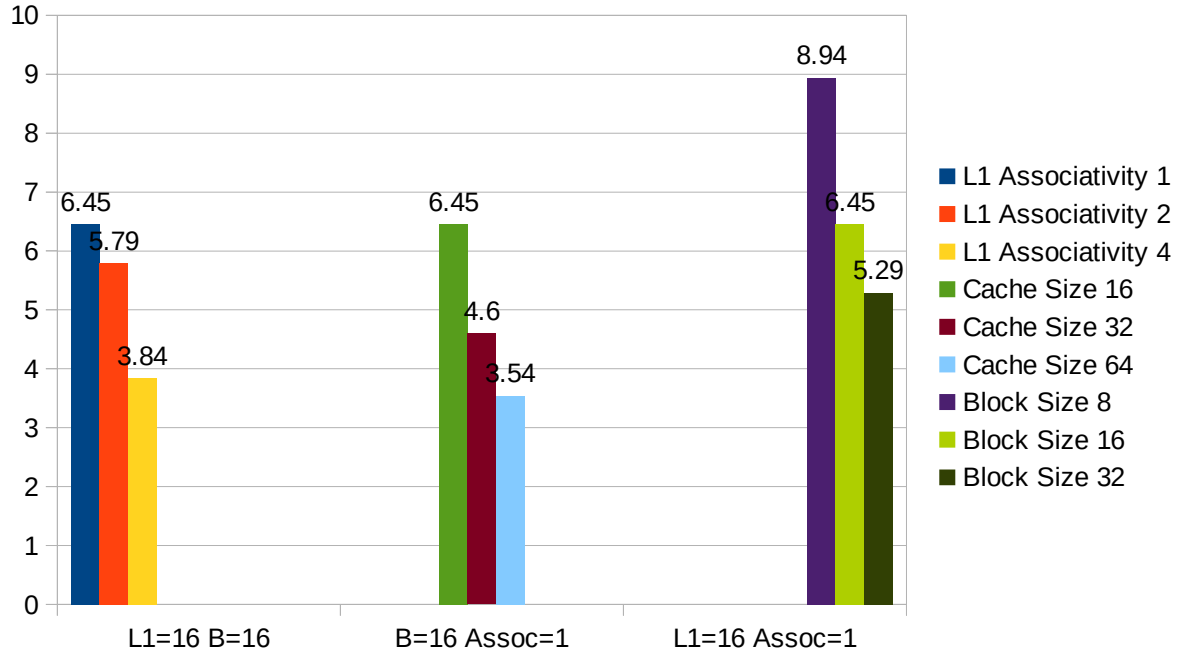
2. For the increasing cache size I expected to see increase in utilization because we need more cache space to put our NxN matrix multiplication entries.

3. For the cache in block size I expected to see increase with increasing block size but not a big one. Because when we increase the cache size for the left matrix we can ready more data(row entry) for multiplication before access request but for the right matrix we probably hold unnecessary data which we probably lose later for the N = 500 case.

4. For the N = 10 case I expected not any change with adjusting cache configuration according the given table values. Because N = 10 case is so small and we already comfortable with cache. I only think that maybe increasing associativity can help us a little.

3. Data Collection

When I ran the pin “allcache” tool with the given matrix multiplication code N = 500 as default. I obtained below results for miss rates in percentages:



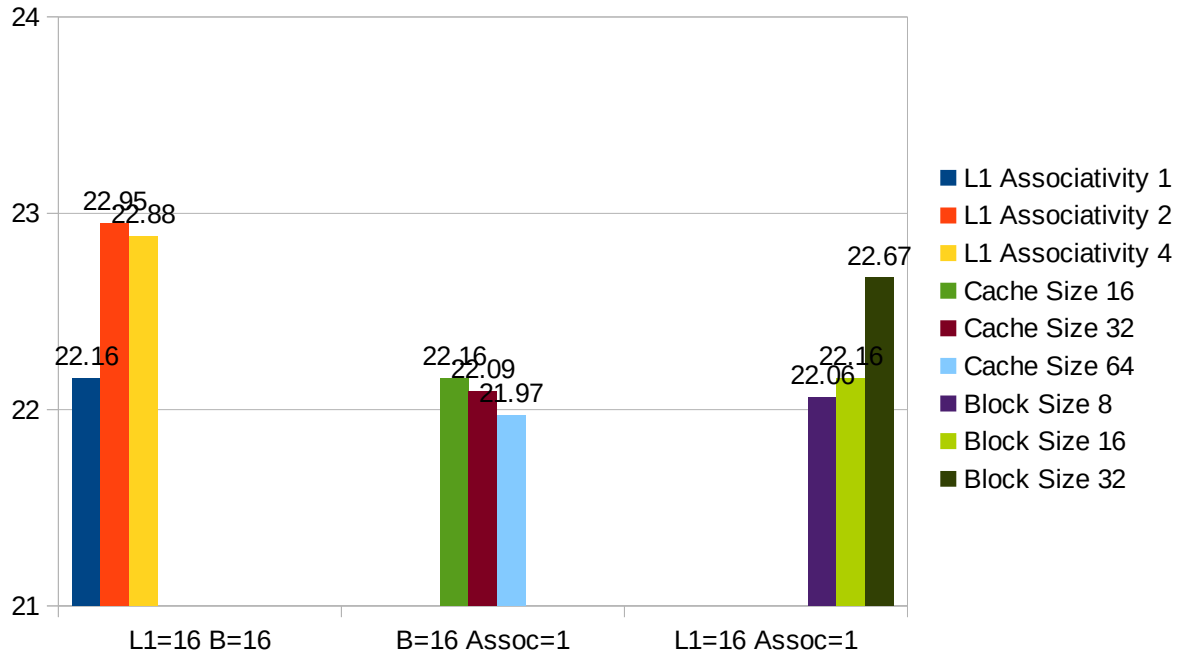
I want to also present some of the numeric result from my experiment to explain better in later part of the report for N = 500 case:

L1 Data Cache	L1 16 – B 16 – Assoc 1	L1 16 – B 16 – Assoc 2	L1 16 – B 16 – Assoc 4
Load Hits	2459731417	2477213984	2528282095
Load Misses	169617893	152135326	101067215
Load Accesses	2629349310	2629349310	2629349310
Load Miss Rate	6.45%	5.79%	3.84%

L1 Data Cache	L1 16 – B 16 – Assoc 1	L1 32 – B 16 – Assoc 1	L1 64 – B 16 – Assoc 1
Load Hits	2459731417	2508498635	2536288087
Load Misses	169617893	120850675	93061223
Load Accesses	2629349310	2629349310	2629349310
Load Miss Rate	6.45%	4.60%	3.54%

L1 Data Cache	L1 16 – B 8 – Assoc 1	L1 16 – B 16 – Assoc 1	L1 64 – B 32 – Assoc 1
Load Hits	2394273696	2459731417	2536288087
Load Misses	235075614	169617893	93061223
Load Accesses	2629349310	2629349310	2629349310
Load Miss Rate	8.94%	6.45%	3.54%

When I ran the pin “allcache” tool with the given matrix multiplication code N = 10. I obtained below results for miss rates in percentages:



For N = 10 case I will not introduce detailed numerical values because they are nearly same for all the configuration.

4. Discussion

After all the data collection and initial commend on the experiment. Now we can discuss our finding for the experiment.

When I start to take first result of the operation. I started to think about how much space we need to complete the matrix multiplication in our L1 data cache and I used this calculation to approximate it:

- A double is 64 bit which equal to 8 byte and computation is in double precision in mul.c
- 500 x 500 our entry for one side of the multiplication which is equal to 250000
- If we multipy our size with our precision which is 8 byte * 250000 = 2000000 = 2MB

so we need 2MB cache size to hold one side of the matrix and we also need other 2 MB to hold right side of the matrix.

With these calculation at hand we sure that we need all the cache space to maximize our utilization. Because we have more data than our L1 data cache size.

Now lets start to talk about on our finding:

For the first case we change our associativity value which allow us to hold multiple values with same index value. We saw that increase in the associativity decreases the our miss rate. At first look, I really have hard times to understand it. However, later I believe that this happened because probably we are able to hold some of the column values for the right matrix in cache when we increase our associativity which with lower associativity we are not able to.

For the second case, we increase our cache size and we see that our miss rate decreases as we increase the cache size, because for some data when we need to access it again probably exist in the cache with big cache sizes.

For the third case, we changed the block sizes which means the chunk of data we need to retrieve when we need a data. For example when we need row one first entry we get row one second entry as well if the block size is big enough. For the block size 8(byte) we only retrieve a double, but if we increase our block size to 32 we can get 4 double at once which is useful for row operations because we sure that for the left side of the matrix we need all row entry at some time. Therefore, If we can be able fit them in cache before they are needed we can decrease our miss rate. When we consider right matrix which do operation with its columns retrieving second column value with first column value not useful for the N = 500 I guess. For the end result however we are able to decrease our miss rate by increasing our latency a little probably.

For the N = 10 case , I think there is no reasonable difference because we can fit cache. I run all the configuration multiple times and result nearly same for all the configuration. What I mean by that for a configuration if I run multiple times I get different result with small standard deviation and when I look that data I can’t distinguish them from each other.

5. Summary

As a result we can see that configuration of the L1 data cache can change the utilization for the large data operation. However the operations we are making also determines our utilization. If we care the cache configuration and write code with that in mind. We can increase our utilization without changing cache configuration as well.

For the matrix multiplication case we know that what we are actually doing is get left matrix row and get right matrix column, dot product them. For this operation we accessing same row values at different times which can cause cache misses. We can improve matrix multiplication with designing an algorithm which allow doing all the computation with accessed values and may never access them later. We may divide our matrices to small sub-matrices and work on them and sum up result when we done with all of them.