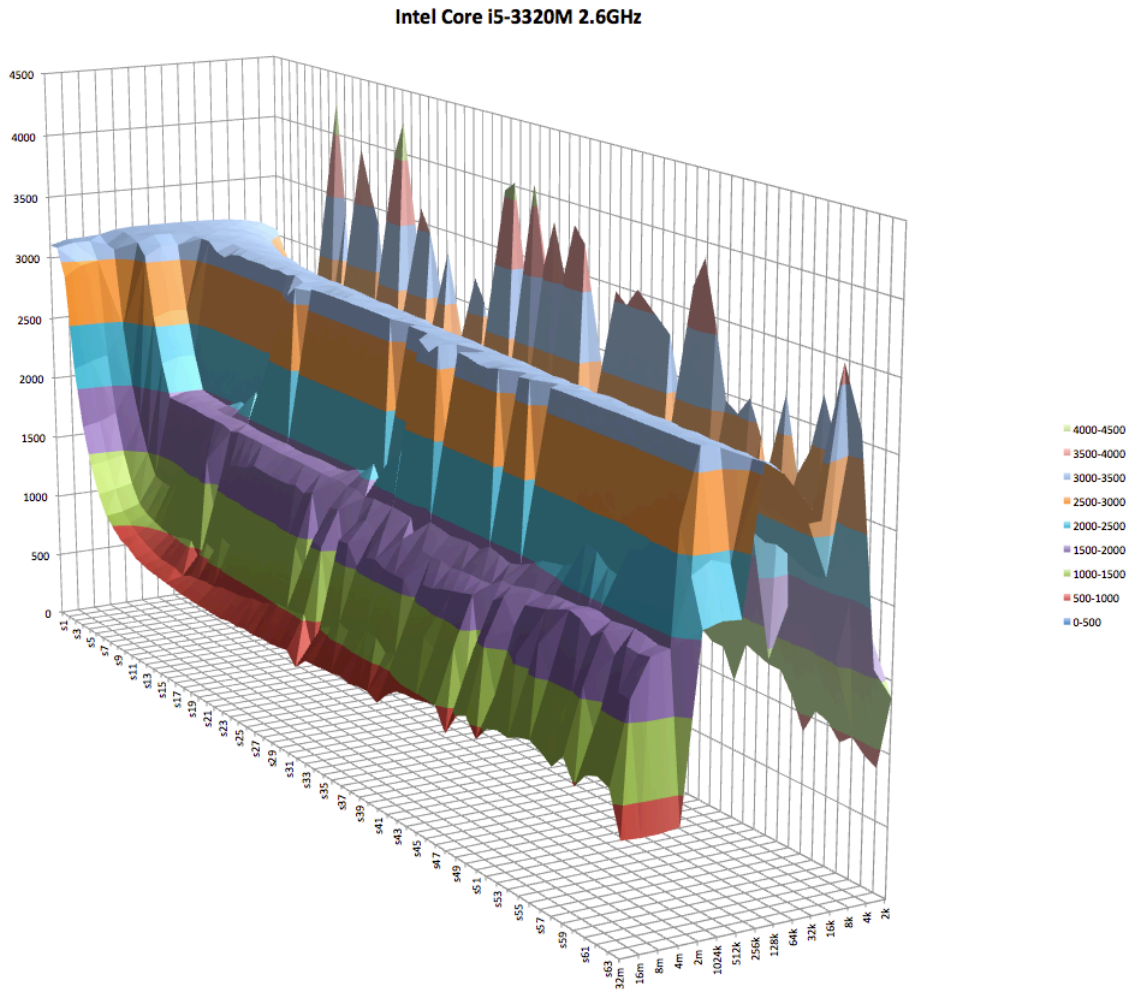
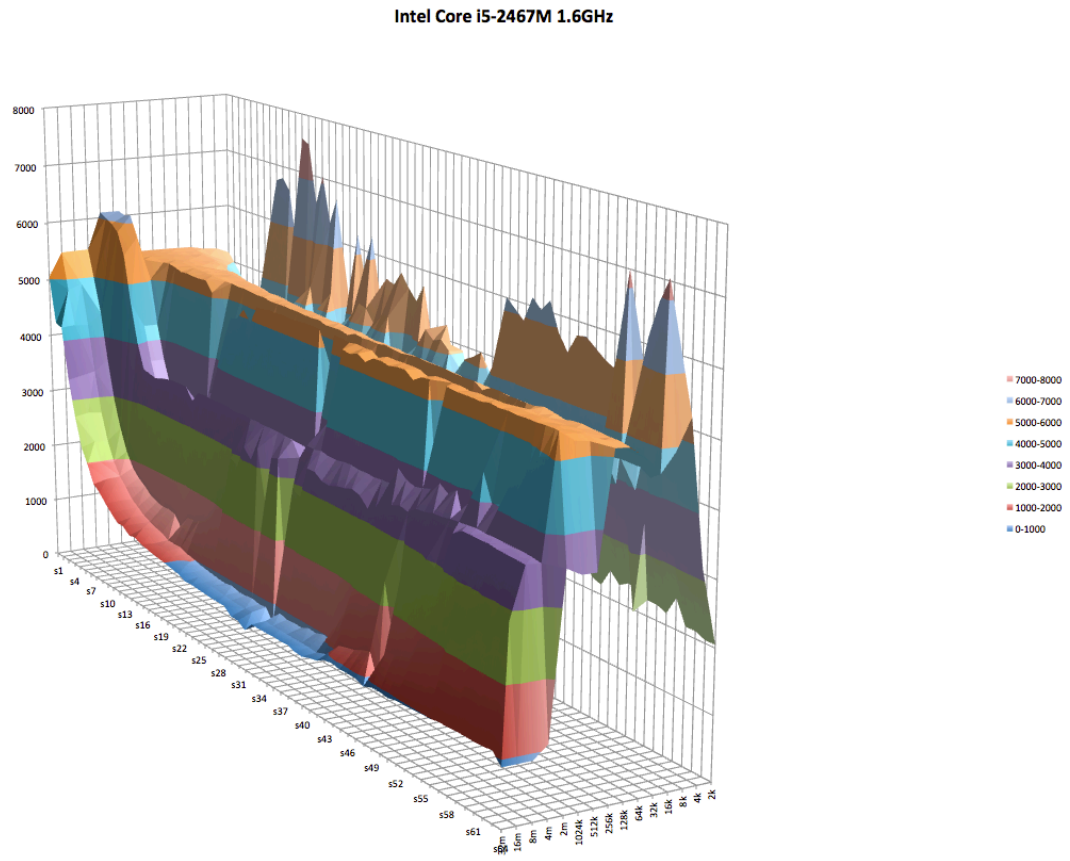
Wei You
r9e7
77610095

Question 2



The above graph is produced on a Lenovo ThinkPad with CPU Intel Core i5- 3320M, 2.6GHz. The ridges of temporal locality are created by continuously hitting the same pieces of data. It represents the cache size because it shows how many disjoint items can fit in cache. Looking at the ridges of temporal locality, L1 cache size is between 16KB and 64KB, L2 cache size is around 256KB and L3 cache size is between 2MB to 8MB. There is fall off in spatial locality because there is a limit in block size in cache. It flatten out

because once it reaches the maximum number of elements it can store, no more words can be put in and the spatial locality performance cannot increase anymore. We cannot identify block size of L1and L2 from graph. By looking at the end points of slopes of spatial locality, we can identify the block sizes of L3: around 8 strides of 4 bytes each, 32 bytes in total.



The above graph is produced on a 2011 MacBook Air with CPU Intel Core i5- 2467M, 1.6GHz. The ridges of temporal locality are created by continuously hitting the same pieces of data. It represents the cache size because it shows how many disjoint items can fit in cache. Looking at the ridges of temporal locality, L1 cache size is around 32KB, L2 cache size is around 256KB and L3 cache size is between 2MB to

8MB. There is fall off in spatial locality because there is a limit in block size in cache. It flatten out because once it reaches the maximum number of elements it can store, no more local words can be put in and the spatial locality performance cannot increase anymore. We cannot identify the block size of L1 and L2 from graph. By looking at the end points of slopes of spatial locality, we can identify the block size of L3: around 8 strides of 4 bytes each, 32 bytes in total.

Question 3

a)

Miss Rate: 0.1250

The arrays are stored in row-major order, so if the 2D array is [[1,2,3], [4,5,6]...], it is stored as [1,2,3,4,5,6...] in memory. Because the reading order of the nested loop is across columns of each row, once it reads the first number of the row, it will load the block of 32 bytes / 4 bytes = 8 integers into cache. The next 7 cache_read will be hits because they are in the same block, then it reads the 9^{th} number and cache hits the 10^{th} to 16^{th} numbers, etc. Hence, a miss happens every 8 numbers. Miss rate is 1 / 8 = 0.1250.

b)

Miss Rate = 0.5000The input 2D array looks like

|1 2 3 4 5 6 7 8 ... 128 |
|2 4 6 8 |
|3 6 9 ... |
|4 8 16 ... |

sumB's access pattern is pulling out blocks of 2x2 sub matrices and sum them. i.e. [1,2,2,4], then [3,6,9,8], [3,4,6,8]... When it access the first item of the first row, it adds the block to, say, set 0, and the second access of 2 is a hit. But this block is overwritten when accessing 3 in the third

row, because the address of 3 is differ by 256 * 4 = 1024 bytes from 1. Because the set index and offset are both 5 bits each, difference in 1024 in address will only affect the tag but not set index, it will use 3,6,9... to replace 1,2,3... in cache. Same thing happens with row 2 and 4. But since the access of the next element right after the replacement is a hit, the miss rate is 0.5.

c)
Miss rate = 1.0
As mentioned in part (b), row 3 and row 1 intend to replace each other, same with row 2 and 4 due to the difference of 1024 in address. sumC is accessing the data in column-major fashion, i.e. row1, row2, row3, row4, row1, row2,... By the time it access the second element in row 1, the cache block is already replaced the data from row 3, so all cache_reads will be misses. Hence miss rate = 1.0

d)
Miss rate = 1.0
Because the cache now is two-way set-associative, set index is 1 bit less than before, unlike part (c), now blocks that align with each other in row 1 and row 2 will go into same set, but blocks that align with them in row 3 and row 4 will also go into the same set and compete for space with them. The same alternating scheme still appears. The four blocks of data that align with other are replacing other for the 2 positions. Hence miss rate is still 1.0.

e)
Miss rate = 0.1250
The cache is now four-way set-associative, set index is 1 bit less than part (d). The four blocks that align with other still tend to go into same set, but now the set can fit them all, so

for each data block, the initial cache_read will be a miss but the later 7 reads will all hit. Hence miss rate = 1 / 8 = 0.1250.

f)
Miss rate = 0.1250
Unlike part (c), difference in address for blocks that align in row 1 and row 3 is now 120 * 2 * 4 = 960 (was 1024 before). This change will cause the blocks to map into different sets because adding 960 to the address will affect the lower 10 bits. Again, cache_read of the first element in the block of 8 is a miss and the later 7 are all hits. Hence the miss rate is 1 / 8 = 0.1250.

I spent 20 hours on this assignment.