Q1. Use Cachegrind to determine the miss-rates of an 8KB, direct-mapped cache with the following block sizes: 32 bytes, 64 bytes, 128 bytes, and 256 bytes. To do so, use commands that look like this: valgrind --tool=cachegrind --D1=8192,1,{block} ./a.out where {block} ranges from 32 to 256. When the program runs correctly there will be lots of lines specifying the cache performance. The one you are interested in is D1 miss rate -- the miss rate for the L1 data cache. This value is rounded to 0.1%. If you want to see the result with more precision, you can divide the number of L1 data misses by the total number of L1 data accesses. After you have run Cachegrind for each block size, save the miss rate from each run. List the miss rate for each block size tested.

A1: 32bits: ==3580035== D1 miss rate: 3.1% ( 3.1% + 13.8% )

64bits: ==3580524== D1 miss rate: 1.6% ( 1.6% + 9.5% )

128bits: ==3580666== D1 miss rate: 0.8% ( 0.8% + 8.6% )

256bits: ==3580842== D1 miss rate: 0.4% ( 0.4% + 8.4% )

Q2. Based on your observations, determine a formula for the miss rate in terms of block size. The formula will not be exact, but it should track the miss rate quite closely.

A2: Let’s call block size ‘b’

We’ll call the miss rate ‘m’

m ̴ 1.024/b

Q3. Your formula above should have an intuitive explanation (i.e., it shoud not just be a line fit to data). Explain what is happening in the cache during each memory access to produce the results you observed.

A3: The reason that the miss rate decreases as the size of the cache increases is because the cache can grab longer blocks of memory from the RAM, which are more likely to have the values that we are looking for. This is due to principles of spatial locality, predicting that we are likely to use values in memory that are close to one another. Because we tested for this pattern using block sizes that are doubled from the previous run, we observe that the miss rate roughly halves due to the two-times increase in block size.