Answers to lab 4 (Caches)

Author: Yueyuan Huang, Shiqi Shu

Part 1

1. The hit rate is 0. This is because although each element is fetched into cache when it is accessed, the whole array cannot fit into the cache (the cache can fit exactly half of the array), which means the latter half will replace the former half, thus none of them could be found in the cache during the next loop.
2. There are 256 compulsory misses because the data have 256 words each fetch put a word into cache, we need 256 fetches to cache all of the data.
3. If the cache size is unchanged and the block size is changed to 8 bytes, there would be 128 compulsory misses because each fetch now fills 8 bytes instead of 4.
4. Only changing from direct mapping to 2-ways set associative doesn't affect the hit rate here because as long as we cache a single element each time and our cache cannot hold the whole array, nothing could be reused in cache.
5. Increasing the cache size to 1024 bytes improves the hit rate to 90% because the cache can now hold the whole array, so each element could be found in the later 9 loops after heating up the cache during the first loop. Whether direct mapping or 2-ways set associative here doesn't affect the hit rate as well.
6. At 1024-bytes cache size and direct mapping, the hit rate is 2304/2560 = 90% for 4-bytes block, 2432/2560 = 95% for 8-bytes block and 2496/2560 = 98% for 16-bytes block.
7. Note that our cache is now large enough to avoid conflict during iteration of a column so we have hits even during the first loop.
   - For 4-bytes block, the block has the same size as an element, so there are 256 cold misses to fetch the whole array into cache.
   - For 8-bytes block, each fetch puts 2 elements into cache, so 128 cold misses.
   - For 16-bytes block, each fetch puts 4 elements into cache, so 64 cold misses.

Part 2

1. No, the hit rate is still 0, for the same reason as the first question of part 1. This is not affected by the order we iterate through the array.
2. The hit rate is improved to 50% when we change the block size to 8 bytes because we can get a hit for every other element since we are now iterating in sequential order, and each fetch puts an element and the element right after it into cache.
3. The hit rate is improved to 75% for 16-bytes block size because now the blocks has the same size as rows of the array, so miss only occurs for the first element of a row.
4. The hit rate is 90% for the same reason as the fifth question of part 1. The cache can now hold the whole array, so each element could be found in the later 9 loops after heating up the cache during the first loop, and in the first loop hit rate is still 0 because we only cache single element.
5. The hit rate is 95%. In the first loop, we get a miss once every other element, so there are 128 misses; in the later loops, all elements are already in the cache, so 0 miss. The hit rate is then 1 - 128/2560 = 95%.
6. The hit rate is 98%. In the first loop, we get a miss once every 4 elements (the first element of each row), so there are 64 misses; in the later loops, all elements are already in the cache, so 0 miss. The hit rate is then 1 - 64/2560 = 98%.
7. No, because there is always at least 1 compulsory miss since the cache is initially empty. However we may achieve nearly 1.0 hit rate (2559/2560) if we configure the block size to 1024 bytes (size of the array) so that we only need a single fetch to cache the whole array.

Bonus

1. The hit rate is 0 because the elements of A and B of the same index happen to be mapped to the same set (typically as a result of array size of 2's power), they keep replacing each other in the cache and nothing in cache is shared between loops although the cache is capable of both arrays.
2. No, although the next element is cached each time, the element of another array will replace that block before it is used, so still no hit at all.
3. Yes, the hit rate is changed to 90% because now each set in cache can store both elements of the same index in two arrays, both array is cached after the first loop, the only misses are the cold misses in the first loop.

4. No, because we can still cache both arrays (Now each set contains 2 elements of A and 2 elements of B after the first loop), and the amount of cold misses is still the number of total elements.
5. It is to avoid conflict misses arise from special address patterns, and as we have multiple lines in a set, there is space for us to implement clever replacement policies to further improve the cache performance.