Part 1:

Group 1(Direct Mapping):

	Final hit rate:	Block utilization
8 blocks, cache block size: 2	50%	50%
8 blocks, cache block size: 4	75%	50%
8 blocks, cache block size: 8	94%	50%

Explanation:

For the direct mapping cache, When the number of blocks is 8, and cache block size decreases from 2 to 8, final hit rate increases and block utilization doesn't change. In the row-major traversal, matrix elements are accessed in the same order they are stored in memory. When the cache block size is 2, each cache miss is followed by 1 hits as the next 1 elements are found in the same cache block. So, the final hit is 50%. When the cache block size is 4, each cache miss is followed by 3 hits as the next 3 elements are found in the same cache block. So, the final hit is 75%. When the cache block size is 8, the cache size is equal to the total matrix elements size (64 words). Cache miss is followed by 7 hits as the next 7 elements are found in the same cache block at the first loop. For the second loop, all 8 elements are found in the cache block. So, the final hit is 94%(60/64). Because the the number of block does not change, the cache block size become bigger so that the hit rate will become bigger after miss. Because of the direct mapping and only using half matrix elements, all of block utilizations are 50%.

Group 2(Direct Mapping):

	Final hit rate:	Block utilization
4 blocks, cache block size: 4	75%	50%
8 blocks, cache block size: 4	75%	50%
16 blocks, cache block size: 4	88%	50%

Explanation:

For the direct mapping cache, When cache block size is 4, and the number of blocks increases from 4 to 16, final hit rate increases and block utilization doesn't change.

In the row-major traversal, matrix elements are accessed in the same order they are stored in memory. When the number of block are 4 and 8 and the cache block size is 4, each cache miss is followed by 3 hits as the next 3 elements are found in the same cache block. So, the final hit is 75%. When the number of block is 16, the cache size is equal to the total matrix elements size (64 words). Cache miss is followed by 3 hits as the next 3 elements are found in the same cache block at the first loop. For the second loop, all 4 elements are found in the cache block. So, the final hit is 88%(56/64). when the the number of block becomes bigger and the cache block size doesn't change, the final hit rate will keep 75% if total cache size is less 64 words, and the final hit rate will increase if total cache size is bigger and equal 64 words. Because of the direct mapping and only using half matrix elements, all of block utilizations are 50%.

Group 3(Direct Mapping):

	Final hit rate:	Block utilization
16 blocks, cache block size: 2	50%	50%
8 blocks, cache block size: 4	75%	50%
4 blocks, cache block size: 8	88%	50%

Explanation:

For the direct mapping cache, When cache block size increases from 2 to 8, and the number of blocks decreases from 16 to 4, final hit rate increases and block utilization doesn't change. In the row-major traversal, matrix elements are accessed in the same order they are stored in memory. When the number of block is 16 and the cache block size is 2, each cache miss is followed by 1 hits as the next 1 element is found in the same cache block. So, the final hit is 50%. When the number of block is 8 and the cache block size is 4, each cache miss is followed by 3 hits as the next 3 elements are found in the same cache block. So, the final hit is 75%. When the number of block is 4 and cache block size is 8, each cache miss is followed by 7 hits as the next 7 elements are found in the same cache block. So, the final hit is 88%(56/64). Even though the the number of block decreases and the cache block size increase, the total cache size doesn't change and keep 32 words. At that time, when the block size is bigger, the hit rate will become bigger after miss. Because of the direct mapping and only using half matrix elements, all of block utilizations are 50%.

Group 4, Fully associative:

	Number of Blocks: 8, Cache block size: 4 words, LRU	Number of Blocks: 8, Cache block size: 4 words, Random	Number of Blocks: 4, Cache block size: 4 words, LRU	Number of Blocks: 4, Cache block size: 4 words, Random
Final Hit rate	88%	88%	75%	78%
Non-empty blocks	8	8	4	4
Block utilization	100%	100%	100%	100%

Explanation:

Because each block is 4 words, each miss will be followed by 3 hits. With fully associative cache, when we have 8 blocks in the cache, the hit rate was higher than 4 blocks. The block utilization was the same between 8 blocks and 4 blocks. For eight 4-words blocks configuration, using LRU policy or Random policy achieve the same hit rate. The reason was with 8 blocks, we did not have to replace any blocks with new data. The program executed data[row][col] = value 64 times, but it really only access total of 32 words (integers). For eight 4-words blocks configuration, it can store 32 words. The cache size is perfect for how much data we need. Once all the blocks are filled in the first iteration, we will not have to evict anything on the second iteration, so will not have miss anymore, which gives 88% hit rate, and 100% block utilization. With four 4-words blocks though, the cache can only store 16 integers. Once all blocks are full, we will have to evict data in a block to replace it with new data (4 integers) from memory. This cause hit rate to decrease. And because we will replace blocks, replacement policy will matter here. Random policy achieves a better hit rate than LRU policy. With LRU policy, each time we will evict a different block. After all the blocks are filled with the first 16 integers, if we want to access the second 16 integers, all the data will be evicted because we need 4 blocks to store 16 integers. After that, when we start the second iteration, all the blocks will be replaced again because now we want the first 16 integers. For random policy, we have a chance to evict the same block, by doing this, not all blocks will have to be replaced. On the second iteration, we might have blocks that still store integers from the first 16 integers, this will increase the hit rate.

Group 5:

	Direct-mapping, Number of Blocks: 8, Cache block size: 4 words, LRU	Fully associative, Number of Blocks: 8, Cache block size: 4 words, LRU	N-way set associative, Number of Blocks: 8, Cache block size: 4 words, LRU, Set size: 1	N-way set associative, Number of Blocks: 8, Cache block size: 4 words, LRU, Set size: 2	N-way set associative, Number of Blocks: 8, Cache block size: 4 words, LRU, Set size: 4	N-way set associative, Number of Blocks: 8, Cache block size: 4 words, LRU, Set size: 8
Final Hit rate	75%	88%	75%	75%	88%	88%
Non-empty blocks	4	8	4	4	8	8
Block utilization	50%	100%	50%	50%	100%	100%

Explanation:

With direct mapping, on the first iteration, every miss brings 4 ints to the cache, so we have 3 hits after a miss. Only 4 blocks are used because data of the first row and the third row use the same 2 blocks in the cache (same index bits), and the second and the fourth row use the same 2 blocks in the cache. So on the second iteration, accessing first row will miss because we have to evict the data of third row from cache. Then it is just every 1 miss and 3 hits again, which gives 75% hit rate. For fully associative cache, it has been explained in group 4. N-way set associative cache with set size 1 is essentially just direct mapping cache, hence why it has the same hit rate and block utilization as direct mapping cache. For N-way set size 2, because the block size is still 4 words, it means we still have 3 hit after 1 miss on first iteration. Again the data of the first row and the third row have the same index (set) bits, and because each set stores only two blocks, the first row and the third row will have to use the same two blocks again. We have to evict the data of the first row (8 ints) from cache to store data of the third row, the same for second the fourth rows. On the second iteration, evicting third rows to store first row. Every 1 miss 3 hits again, which gives 75% hit rate, and 4 blocks used total. For set size 4, the first row and the third row still have the same index bits. But each set now can store 4 blocks, which means we don't have to evict data of the first row in cache to store the third row because there are still two empty blocks in the same set. Same for the second and the fourth row. On second iteration, nothing has been evicted, so no miss. Thus, 88% his rate, and 8 blocks used total. For set size 8, it is just a fully associative cache like the second configuration.