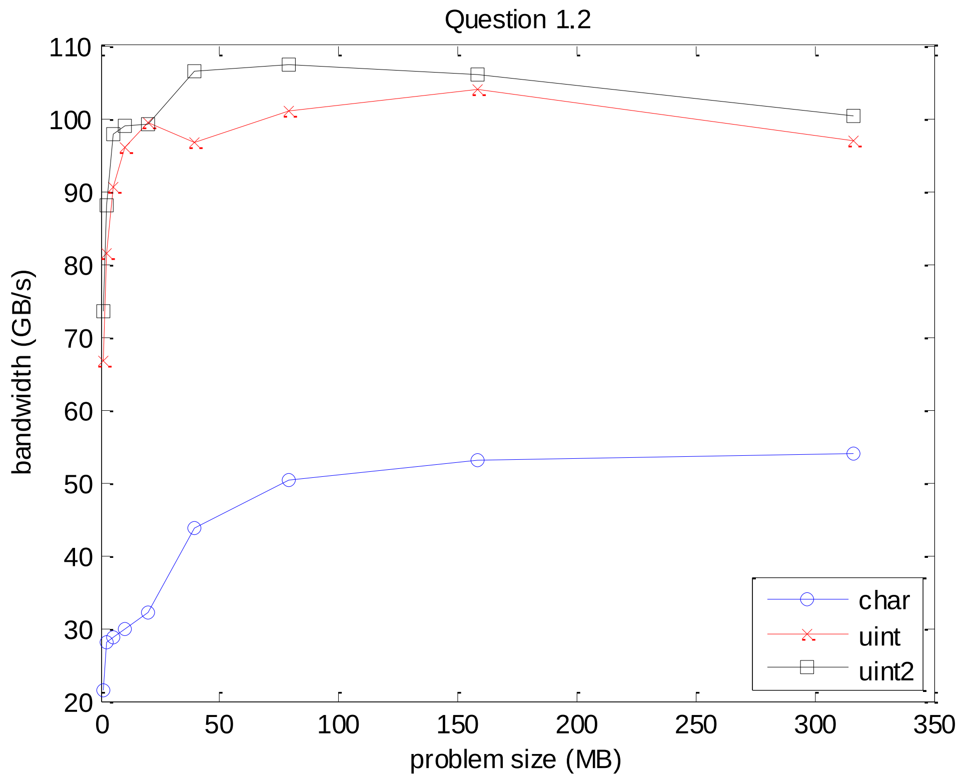
HW2 Writeup

Problem 1.2



Problem 1.3

The smallest unit of thread is warp which consists of 32 processors and the cache line of each warp has a width of 128 bytes. Therefore, in the case of *uint*, the warp will use up all the 128 bytes loaded into the cache since the size of *uint* is 4 bytes and 32 x 4 bytes = 128 bytes. However, for *char*, since each *char* variable has only a size of 1 byte, 3/4 of the data loaded into the cache is wasted in one execution of the warp. This explains why using *uint* format has higher bandwidth. However, the increase in bandwidth is lower than the expected value of 4 because the excessive data loaded into the cache in the case of *char* can be used in the next warp execution.

There is not much difference in bandwidth for both the *uint* and *uint2* cases because in both cases, there is no waste in memory. Any data loaded into the cache during each warp execution will all be used within that execution.

Problem 2.2

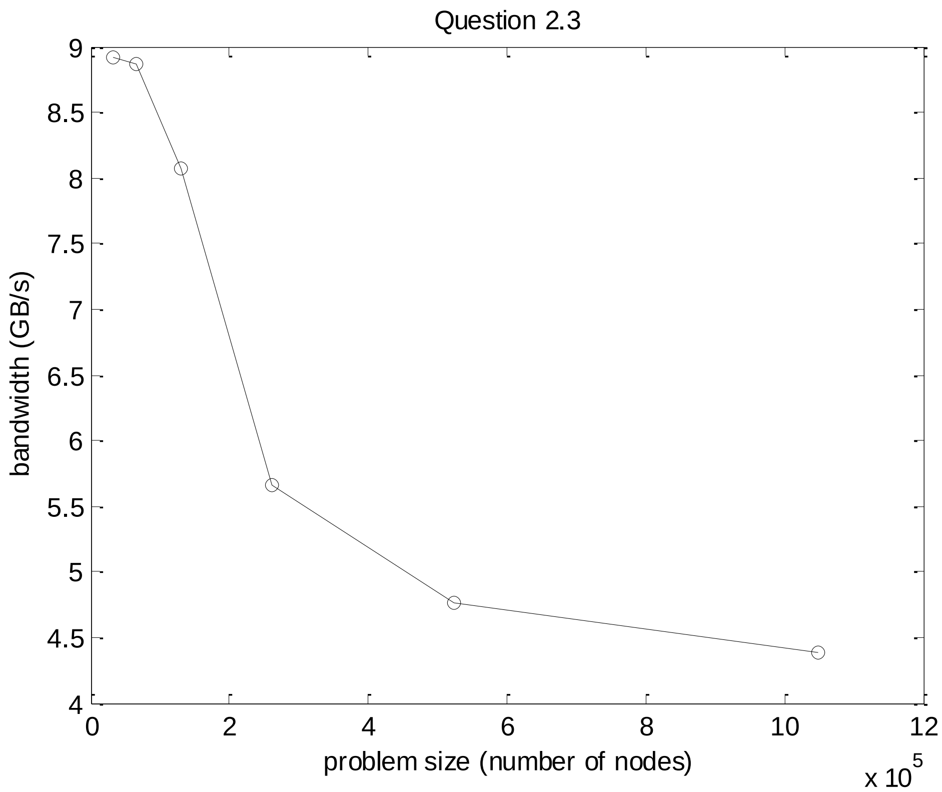
In the cuda function *device_graph_propagate()*, we can see that for each entry output of vector *graph_nodes_out*, there is one write operation.

As for the read operations, it can be seen that for each node, we need to read the range of neighbors *graph_indices[i]* and *graph_indices[i + 1]*. For each neighbor, we also have three read operators to get *graph_edges*, *inv_edges_per_node* and *graph_nodes_in*. Therefore, for each node, there are *2 + 3*avg_edges* read operations.

Since both data types *uint* and *float* have 4 bytes, the total number of read and write operations are:

*(3 + 3*avg_edges)* num_nodes*num_iterations*(4 bytes)*

Problem 2.3



Problem 2.4

The memory access in this problem is random due to the data structure *graph_indices* and *graph_edges*. Once we get the indices of neighboring nodes from *graph_edges*, we have to access the input vector *graph_nodes_in* and *inv_edges_per_node* randomly to calculate the output value. This data access pattern is bad since we cannot reuse data loaded into cache. As we can see, the maximum bandwidth in problem 1 is over 100 GB/s and even the smallest one is still above 20 GB/s. However, in problem 2, the highest bandwidth is only around 9 GB/s. In problem 1, contiguous memory is accessed when elements of arrays are read. This data accessing pattern is much better than the random data accessing pattern in problem 2 in caching and it explains why bandwidth in problem 1 is much higher than that in problem 2.