Different approaches:

i. Global access base case (sobelEdgeDetection function)
ii. Storing the four corner points which are used in both sum1 and sum2 computation in local variables and then using the local variable. This seemed to slow down the execution time. (sobelEdgeDetectionWithRegisters)
iii. Storing 34 * 34 grid in shared memory with boundary threads loading the halo region and then computing (sobelEdgeDetectionSharedMem). This gives fluctuation computation times. But on an average performs worse than the base case due to thread divergence.
iv. Unrolling - Storing 4 * 4 grid in each thread in local array, and then computing the inner 2 * 2 values. (sobelEdgeDetectionSharedMem2).
v. Shared memory overlapping – here we over lap grids and compute the inner grid of values. This removes the additional checks required to load the halo region. This gives a lot better performance improvement compared to any of the above methods. (sobelEdgeDetectionSharedOverlap)
vi. Shared memory + unroll – here we do the same procedure as the above function but each thread computes 2 * 2 grid of values. This seems to be the best performing function with respect to the other methods.
vii. Shared Memory + unrolling + coalesced memory access – best performance optimization among all the above methods.

I have noticed a lot of fluctuation in estimated time. Even the default base case program, I have seen friends getting an estimate of 0.52 ms whereas I am getting 0.68ms, in spite of being similar code.

I have used block size of 32 * 32 over all the functions. But function V and VI above seems to perform slightly better with threads per block as 16 * 16.