HW3 REPORT  
(CIS 6930: Spring 2012)  
I have neither given nor received any unauthorized aid on this assignment

NAME                  : Tushar Athawale  
UFID                  : 09781050  
HPC Account Username : athawale  
HPC Client            : Cobra  
Device                : Device 1 -“Tesla M2070” with Compute 2.0 capability
1) No. of the kernel functions used in the program = 2

Kernel function for **breadth first search**.

No. of threads per block = 256

No. of the code blocks = number of vertices / number of threads per block

Kernel function for **data copy**

No. of threads per block = 256

No. of the code blocks = number of vertices / number of threads per block

These numbers gave 100% occupancy on multiprocessor on the cuda occupancy calculator.

2) Synchronization Method Used –
   a) cudaThreadSynchronize()
   b) __syncthreads()

3) No. Shared memory is not used to for the bfs problem. The main reason is as follows. We can pull the vertex data into shared memory and then read data from the shared memory. However, we will have to do all remaining reads such as reading the neighbors and all the writes such as updating frontier and visited arrays into the global memory since neighbors can seat anywhere and not necessarily in the shared memory. Therefore pulling vertex data into shared and reading from the shared would not increase any efficiency much. When tried on the 256 nodes case there was not much rise in the efficiency when used shared memory.

4) Since shared memory is not used, L1 cache configuration best suits this application. Because of spatial locality, L1 cache accesses would be faster. e.g consider the cost array. When cost array for the neighbor is updated, it looks at its parent. It will search for the parent in the L1 cache. If not found in L1, it will search in L2 and if not found in L2 it will access the global memory. Then it will bring piece in spatial locality of the parent in L1. So the further accesses of the neighbors to their parent would be from L1 rather than global memory and would be lot faster.
5) In this assignment, handling for the small and the big graphs was similar. However if the graph size gets too much large, we will need to consider data types of larger range such as long int for large graphs.

**Execution Times -**

**256 Nodes -**

L1 cache disable – 0.000678s

L1 cache : 16 KB, Shared Memory: 48KB – 0.000491s

L1 cache : 48 KB, Shared Memory: 16KB – 0.000283s
64K Nodes -

L1 cache disable – 0.003812s

L1 cache : 16 KB, Shared Memory: 48KB – 0.003415s

L1 cache : 48 KB, Shared Memory: 16KB - 0.002978s

Disabled cache using -

-Xptxas -dlcm=cg