HW4 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 : Tesla 4
Device : Device 0 - “Tesla T10 Processor” with Compute 1.3 capability
(Could not test on fermi in my alloted slot. There were issues in getting fermi node.
Bugziall ticket number is 5997)
1) In the original BFS implementation, we had a thread per vertex. However since any places within the large 64K array could be active at particular level of iteration, threads within a thread warp take different control paths(active take one and nonactive take other) for most of the thread warps, resulting in the branch diversion. Therefore when there is a branch diversion, all threads in the warp execute serially resulting in inefficiency. This is the **main reason** of the reduction in the performance. We mitigate this inefficiency by post-processing the array. We bring all the active threads together(contiguous) in the front of the array and keep all inactive threads together after active thread. Therefore there is only one thread warp where the branch divergence happens(at the boundary of active and non active threads). Even in the special case if number of the active nodes is multiple of the warp size, then we have no branch divergence. For all the other threads warps, threads within a thread warp only follow one branch path thus mitigating the branch divergence.

2) By putting the dummy instructions in the else part we spend some time in executing that instruction. In branch divergence, as explained in the part 1, when instructions are executed serially, we want to spend time on the both control paths, to get the idea of how performance difference is there when there is branch divergence and when it is not there. If we do not put dummy instruction, we would get almost same results when we have branch divergence and we don't have branch divergence. That is why we put the dummy statements in the else part.

3) 
   a) With Post-processing – tesla 0.161 ms on new BFS  
   b) Without Post-processing - tesla 0.173 ms on old BFS  
   c) As expected I got the improvement by 0.012 ms on average. We should get more improvement in case of arrays which are of size more than 64K. As explained in the part 1, we get less branch divergence(More specifically only in 1 thread warp in each iteration for BFS) in the thread warps because of the way we are post-processing the array in the each iterations.
4)
a) With Post-processing - Tesla 0.165 ms on new BFS

b) Without Post-processing - Tesla 0.185 ms on normal BFS

c) As expected I got the improvement by 0.02 ms on average which is more than the improvement of the part 2c. Since we have added more dummy instructions, we should be able to see more difference in the time required for the old and new BFS. This is because old one with branch divergence will take larger time when we have more dummy instructions.