Big Assignment Report

BFS

Team: 17

Name	Sec	BN
Abdelhamid Emad	1	34
Mahmoud Abdelhamid	2	22

Delivered to:

Dr/ Dina Tantawy

Eng/ Mohammad Abdallah

Vertex Centric TOP DOWN

- This approach will assign to each vertex a thread, and only the threads with a level equal to the current level will loop over all its neighbors and assign to them the new level (will be current level +1)
- Here the best graph representation will be a Compressed Sparse Row (CSR), because we need each vertex to loop over it is neighbors
- Better in low-degree graphs like Road graphs

Vertex Centric BOTTOM UP

- This approach will assign to each vertex a thread, and only the threads with any one of its neighbors in the current level (i.e visited) will update its level (will be current level +1)
- Here the best graph representation will be a CSC (Compressed Sparse Column) because we need each vertex to loop over the neighbors
- Here are faster than top-down, because this approach will loop over its neighbors until found only one visited rather than must loop over all its neighbors
- Better in high-degree graphs like network graphs

Edge Centric

- This approach will assign to each edge a thread, and only the threads that will live that is the source vertex in the current level, so will add the destination vertex to the current level +1
- Here the best graph representation will be a COO (Coordinate List) because we need to loop over all edges
- Better in high-degree graphs like network graphs

TOP DOWN Approach Frontier Based

- This is an optimization on the top-down vertex concentric approach that instead of generating a thread for each vertex will generate the number of threads equal to the number of vertices in the frontier
- Frontier will be updated with each kernel call and will add all the visited nodes in the current level that we processed
- Overhead: here multiple threads will need to access the same frontier so will need an atomic add operation

TOP-DOWN Approach Frontier Based with Privatization

- Here will make a shared memory for each block and add it to nodes of the current level then all threads at the end will share on adding to the global memory this reduces accessing the global memory

Analysis

CPU:

Number of nodes	Time elapsed
511	5.2614e-05s
15606	0.00232734s = 2327.34us
500000	0.10917s=109170us

TOP DOWN Vertex Centric:

Number of nodes	Time elapsed
511	39.200us
15606	517.01us
268495	1.9889ms
500000	2.1683ms

BOTTOM UP Vertex Centric:

Number of nodes	Time elapsed
511	39.168us
15606	592.53us
268495	3.9574ms
500000	2.6344ms

Edge Centric:

Number of nodes	Time elapsed
511	34.688us
15606	488.69us
268495	3.7401ms
500000	4.0495ms

TOP DOWN Frontier:

Number of nodes	Time elapsed
511	34.592us
15606	934.47us
268495	5.4708ms
500000	2.4561ms

TOP DOWN Forntier privatization:

Number of nodes	Time elapsed
511	62.687us
15606	1.0270ms
268495	5.209ms
500000	2.3255ms

Results:

TOP DOWN VS TOP DOWN Frontier

Number of nodes	Time elapsed (TOP DOWN)	Time elapsed (TOP DOWN Frontier)
511	39.200us	34.592us
15606	517.01us	934.47us
268495	1.9889ms	5.4708ms
500000	2.1683ms	2.4561ms

Comment:

When the number of nodes are small Top Down is better than Top Down Frontier but when the number of nodes become more bigger Top Down will better thus because Top down frontier access the global memory when adding an element to the frontier also there is atomic add at adding nodes to the frontier but in the top down there are waste of resources that number of the threads that diverge at each time

Top Down vs Bottom up

Number of nodes	Time elapsed (Top Down)	Time elapsed (Bottom up)
511	39.200us	39.168us
15606	517.01us	592.53us
268495	1.9889ms	3.9574ms
500000(SPRSE)	2.1683ms	2.6344ms

numberofedge=3*		
500000(DENSE) numberofedge=6*	3.5524ms	3.34ms

Comment:

Here we found that in case of sparse adjacency matrix (i.e the number of edges in the graph are small) Top Down are better than Bottom up , and when the number of edges in the graph become larger Bottom up will be better

Edge Centric Vs Top Down Vs Bottom Up

Number of nodes	Time elapsed (Edge Centric)	Time elapsed (Top Down)	Time elapsed (Bottom Up)
511	34.688us	34.592us	39.168us
15606	488.69us	934.47us	592.53us
268495	3.7401ms	5.4708ms	3.9574ms
500000	4.0495ms	2.4561ms	2.6344ms

Comment:

As the number of the nodes become small (i.e the number of edge also small) so the number of the threads needed in the edge centric will be small and as the number of nodes increases this will lead to large number of edges so number of the threads needed will be large this will lead to as the number of edges small edge will better otherwise Top down and Bottom Up will be better

TOP DOWN Frontier Vs TOP DOWN Forntier privatization:

Number of nodes	Time elapsed (TOP DOWN Frontier)	Time elapsed TOP DOWN Forntier privatization:
511	34.592us	62.687us
15606	934.47us	1.0270ms
268495	5.4708ms	5.209ms
500000	2.4561ms	2.3255ms

COMMENT :

in the Privatization we add the shared memory constant equal to 2048 bytes so in the smaller number of nodes the Top Down almost the same because the memory overhead but as the number of the nodes become bigger Top Down Privatization become the better than Top Down

CPU Vs TOP DOWN Forntier privatization

Number of nodes	Time elapsed (CPU)	Time elapsed (TOP DOWN Forntier privatization)
511	5.2614e-05s	
15606	0.00232734s	0.0030041s
268495	0.0688794s	0.00583322s
500000	0.10917s	0.00307382s

Comment:

here we calculate the elapsed time of Top down Frontier (time from when we start the kernel until we get the output) we found that at huge number of nodes the Top Down Frontier become much more Better !!