

# LP1 Assignment HPC 4

Date: 24<sup>th</sup> September, 2020

Title: Parallel Searching Algorithms

## Problem Definition:

Design and implement parallel algorithm utilizing all resources available for

1. Binary Search for Sorted Array
2. Best-First Search that (traversal of graph to reach a target in the shortest possible path)

## Learning Objectives:

- Learn Parallel Decomposition of Problem
- Learn Parallel Computing using CUDA

## Learning Outcomes:

- I will be able to perform parallel computing using CUDA libraries
- I will be able to decompose a problem into sub problems, learn how to use GPUs, and solve problems using threads on GPU

## S/W & H/W Packages:

1. Operating System : 64-bit Open source Linux or its derivative
2. Programming Language: C/C++
3. NVidia CUDA-enabled GPU
4. OpenMP support
5. Google Colaboratory(uses Tesla K80 GPU)
6. MPI Compiler

## Related Mathematics:

Let S be the system set:

$S = \{s; e; X; Y; Fme; DD; NDD; Fc; Sc\}$

s=start state

e=end state

X=set of inputs

$X = \{X_1\}$

where

$X_1$  = Elements of Vector

Y= Output Set (Sum or Product of elements of Vector/Matrix)

Fme is the set of main functions

$Fme = \{f_1, f_2, f_3\}$

where

$f_1$  = decomposition function

$f_2$  = function to generate graph

$f_3$  = function to display results

DD= Deterministic Data Graph of Elements

NDD=Non-deterministic data No non deterministic data

Fc =failure case: No failure case identified for this application

## Concepts Related to Theory:

### **Best-First Search:**

Best-First Search is an algorithm that traverses a graph to reach a target in the shortest possible path. Unlike BFS and DFS, Best-First Search follows an evaluation function to determine which node is the most appropriate to traverse next.

Steps of Best-First Search:

- Start with the root node, mark it visited.
- Find the next appropriate node and mark it visited.
- Go to the next level and find the appropriate node and mark it visited.
- Continue this process until the target is reached.

This algorithm contains two main components: Open list, Close list in most parallel formulations of BFS, different processors concurrently expand different nodes from the open list.

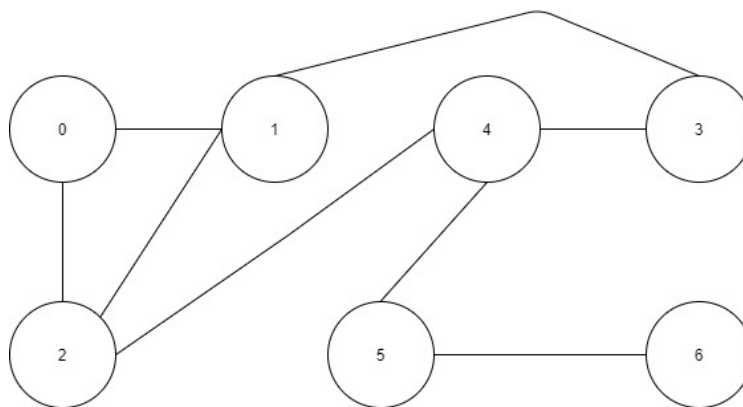
There are two problems with this approach

- The termination criterion of sequential BFS fails for parallel BFS
- Since the open list is accessed for each node expansion, it must be easily accessible to all processors, which can severely limit performance.

The core data structure is the Open list (typically implemented as a priority queue). Each processor locks this queue, extracts the best node, unlocks it. Successors of the node are generated, their heuristic functions estimated, and the nodes inserted into the open list as necessary after appropriate locking.

Termination signaled when we find a solution whose cost is better than the best heuristic value in the open list. Since we expand more than one node at a time, we may expand nodes that would not be expanded by a sequential algorithm.

A general schematic for parallel best-first search using a centralized strategy. The locking operation is used here to serialize queue access by various processors. The open list is a point of contention. Avoid contention by having multiple open lists. Initially, the search space is statically divided across these open lists. Processors concurrently operate on these open lists.



Time Complexity =  $O(|V|+|E|)$

---

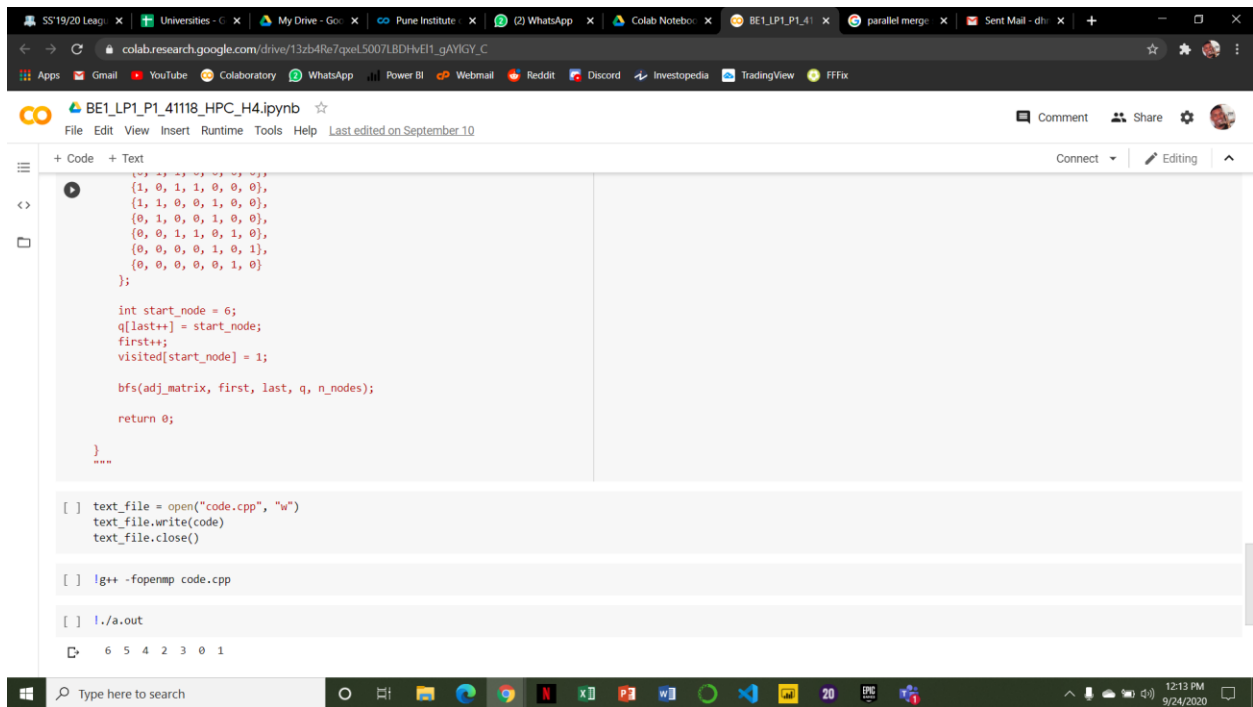
## References:

1. CUDA Documentation: <https://docs.nvidia.com/cuda/>

## Steps for Execution:

1. Open Google Colaboratory
2. Change Runtime type from CPU to GPU
3. Get nvcc plugin for Jupyter Notebook
4. Write parallel sorting algorithms.
5. Use `!g++ -fopenmp filename.cpp` to compile the code.
6. Use `!./a.out` to give output.

## Output:



```
BE1_LP1_P1_41118_HPC_H4.ipynb
File Edit View Insert Runtime Tools Help Last edited on September 10

+ Code + Text
Connect Editing

{1, 0, 1, 1, 0, 0, 0},
{1, 1, 0, 0, 1, 0, 0},
{0, 1, 0, 0, 1, 0, 0},
{0, 0, 1, 1, 0, 1, 0},
{0, 0, 0, 0, 1, 0, 1},
{0, 0, 0, 0, 0, 1, 0}
};

int start_node = 6;
q[last++] = start_node;
first++;
visited[start_node] = 1;

bfs(adj_matrix, first, last, q, n_nodes);

return 0;
}
---
```

```
[ ] text_file = open("code.cpp", "w")
text_file.write(code)
text_file.close()

[ ] !g++ -fopenmp code.cpp

[ ] !./a.out

6 5 4 2 3 0 1
```

## Notebook Link:

[https://colab.research.google.com/drive/13zb4Re7qxeL5007LBDHvEI1\\_gAYIGY\\_C?usp=sharing](https://colab.research.google.com/drive/13zb4Re7qxeL5007LBDHvEI1_gAYIGY_C?usp=sharing)

## Conclusion:

I have successfully performed parallel searching algorithms using openmp interface.