Roll No : 41453

High Performance Computing : Assignment 4

Title :
Parallel Search Algorithm

Problem Statement :
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.

Learning Outcome:
Be able to :
- Decompose problem into sub problems, to learn how to use GPUs, to learn to solve sub problem using threads on GPU cores.

Software and Hardware Requirements :
- 64 bit CPU
- 4 GB RAM
- Ubuntu 18 OS
- G++ compiler
- OpenMP support
- MPI Compiler

Steps to run:
- Install G++ Compiler
- Verify OpenMP support
- Use G++ or MPI to compile the program (g++ filename.cu -o filename-fopenmp)
- Run program using generated output file (./filename)

Programmers Perspective:
S = {s; e; X; Y; F_{me}; F_f; DD; NDD}
s = start state (start of program)
e = end state (final required output obtained)
X = {X1} where X1 is element of Graph.
Y = Output Set (Searched Element)
F_{me} = function to perform parallel searching operations.
F_f = {f1,f2,f3}

where
f1 = decomposition function
f2 = function to generate graph
f3 = function to display results
DD = Graph of Elements.
NDD = No non deterministic data

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.

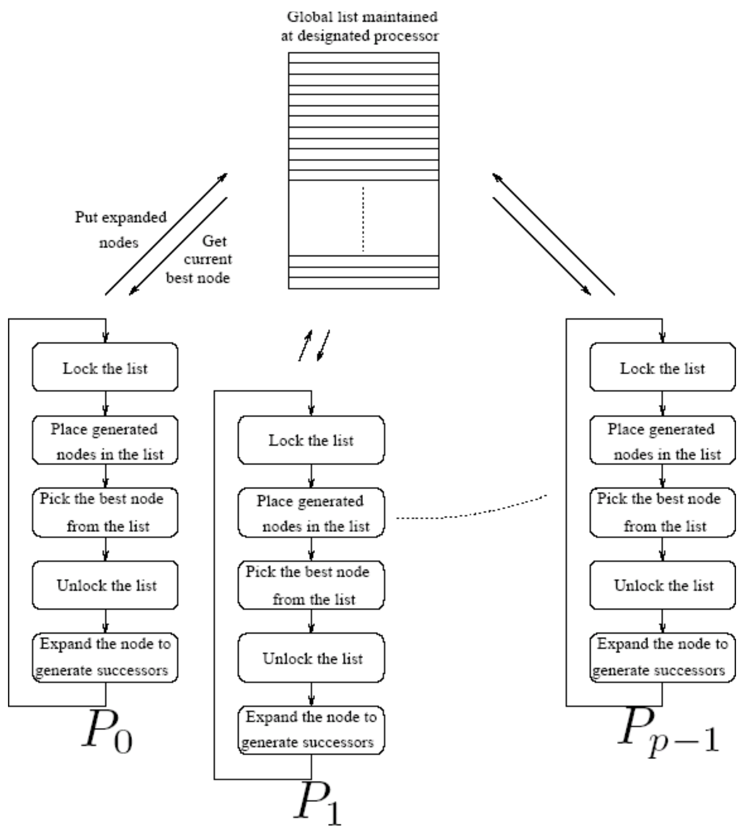
This algorithm contains two main components:
Open list , Close list
In most parallel formulations of BFS, different processors concurrently expand different nodes from the open list.

There are two problems with this approach

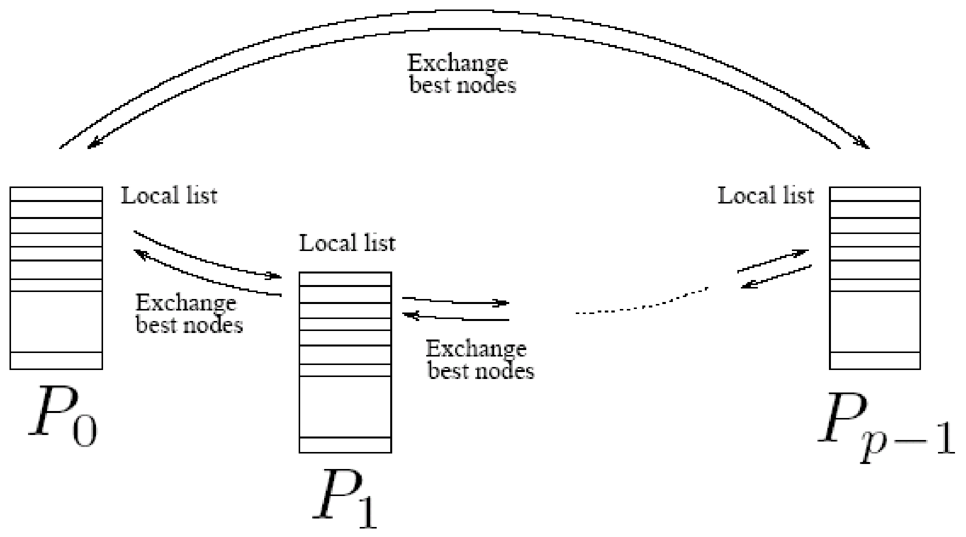
- The termination criterion of sequential BFS fails for parallel BFS
- Since the open list is accessed for each node expansion, it must be easily accessible to all processors, which can severely limit performance

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.


Global list maintained at designated processor
Put expanded nodes
Get current best node
Lock the list
Place generated nodes in the list
Pick the best node from the list
Unlock the list
Expand the node to generate successors
P_0
P_1
P_{p-1}

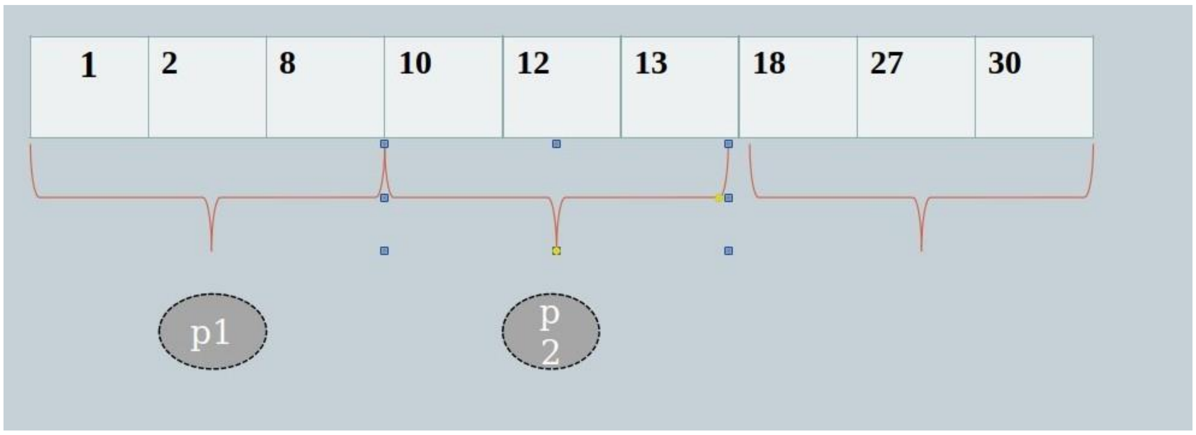
Since the heuristic values of nodes in these lists may diverge significantly, we must periodically balance the quality of nodes in each list.



A message-passing implementation of parallel best-first search using the ring communication strategy

Parallel Binary Search:

We have an ordered array ,we have two processors
We part our array to P+1 parts , where p is number of processors



For k is less than n processors, split the array into n/k groups and assign a processor to each group.
Run binary search on that group.
Now the time is log(n/k).

Conclusion :
Implemented Parallel best first search using OpenMP and implemented Parallel binary search using MPI successfully.