Assignment No-1(A)

Page No. : 1
Date. : / /

Aim:- Design and Implement 11[el] Breadth First search based on existing algorithms using openMP. Use a tree or an undirected graph for BFS.

Objective:- Students should be able to perform parallel Breadth search based on existing algorithms using OpenMP.

Prerequisite:- 1) Basic of programming language.
2) Concept of BFS
3) Concept of parallelism.

Theory

Breadth First search (BFS)

BFS stands for Breadth First search. It is a graph traversal algorithm used to explore all the nodes of a graph or tree systematically, starting from the root node or a specified starting point, and visiting all the neighboring nodes at the current depth level before moving on to the next depth level.

- The algorithm uses a queue data structure to keep track of the nodes that need to be visited, and marks each visited node to avoid processing it again.

The basic idea of the BFS algorithm is to visit all the nodes at a given level before moving on to the next level, which ensures that all the nodes are visited in breadth-first order.

BFS is commonly used in many applications, such as finding the shortest path betn two nodes, solving puzzle and searching through a tree or graph.

Example of BFS

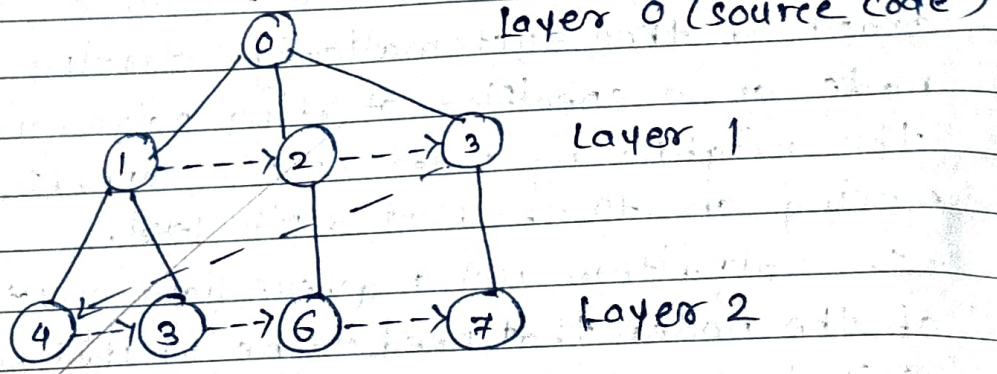
Now let's take a look at the steps involved in tra
-ng a graph by using Breadth-first-searchi-

Step 1:- Take an Empty queue.

Step 2:- Select a starting node(visiting a node) & inse
it into the Queue.

Step 3:- Provided that the Queue is not empty, extr
the node from the Queue and insert its child nodes
(exploring a node) into the Queue.

Step 4:- print the printed node.



How parallel BFS Works:-

- ||el BFS is an algorithm used to explore all the nodes of a graph or tree.
systematically in ||el. It is a popular ||el algori
used for graph traversal in distributed computi
shared-memory systems, and ||el clusters.

- The 11e1 BFS algorithm starts by selecting a root node or a specified starting point, and then assigning it to a thread or processor in the system.

- Each thread maintains a local queue of nodes to bevisited and marks each visited node to avoid processing it again.

- The algorithm then proceeds in levels, where each level represents a set of nodes that are at a certain distance from the root node.

- Each thread processes the nodes in its local queue at the current level with other threads or processors.

- The 11e1 algorithm uses two phases:- the computation phase and the communication phase. In the computation phase, each thread processes the nodes in its local queue while in the communication phase, the threads exchange the nodes that are adjacent to the current level with other threads.

Conclusion:- In this way we can achieve parallelism while implementing BFS.

Assignment No 1 (B)

Page No. : 1
Date. : / /

Aim:- Design and Implement parallel Depth first search based on existing algorithms using OpenMP. Use a tree or an undirected graph for DFS.

Objective:- students should be able to perform parallel Depth first search based on existing algorithms using OpenMP.

Prerequisite:- 1) Basic of programming language.
2) Concept of DFS.
3) Concept of parallelism.

Theory:-

Depth-First search (DFS)

DFS stands for depth-first search. It is a popular graph traversal algorithm that explores as far as possible along each branch before backtracking.
This algorithm can be used to find the shortest path bet^n two vertices or to traverse a graph in a systematic way. The algorithm starts at the root node & explores as far as possible along each branch before backtracking. The backtracking is done to explore the next branch that has not been explored yet.
DFS can be implemented using either a recursive or an iterative approach. The recursive approach is simpler to implement but can lead to a stack overflow error for every very large graphs.
-The iterative approach uses a stack to keep tracle of nodes to be explored and is preferred for larger graphs.

Date. : / /

DFS can also be used to detect cycles in a graph.
- A standard DFS implementation puts each vertex of the graph into one of two categories.
  - Visited
  - Not visited.

Example of DFS:-
To implement DFS traversal, you need to take the following stages.

Step 1:- Create a stack with the total no. of vertices in the graph as the size.

Step 2:- Choose any vertex as the traversal's beginning point. Push a visit to that vertex and add it to the stack.

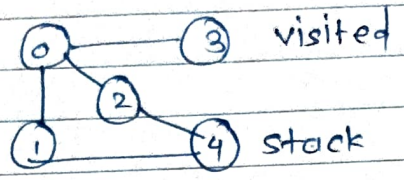
Step 3:- Push any non-visited adjacent vertices of a vertex at the top of the stack to the top of the stack.

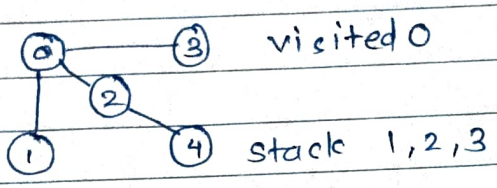
Step 4:- Repeat steps 3 & 4 until there are no more vertices to visit from the vertex at the top of the stack.

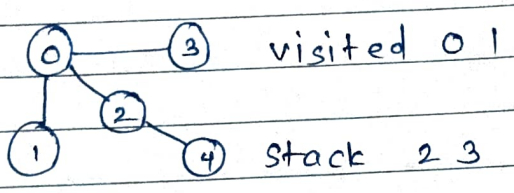
Step 5:- If there are no new vertices to visit, go back and pop one from the stack using backtracking

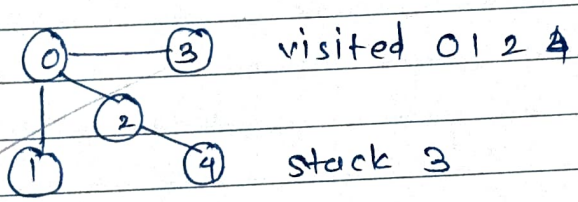
Step 6:- Continue using steps 3, 4, & 5 until the stack is empty.

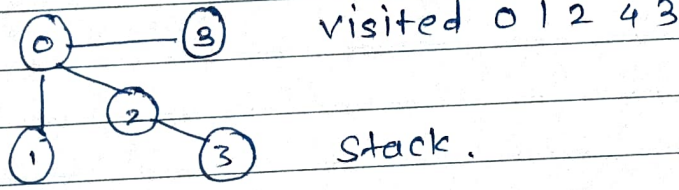
Step 7:- When the stack is entirely unoccupied, create the final spanning tree by deleting the graph's unused edges.











Conclusion :- In this way we can achieve parallism while implementing DFS.

Assignment No-2

Page No. : 1.
Date. : / /

Aim:- Write a program to implement parallel Bubble sort and Merge sort using OpenMP, use existing algorithms and measure the performance of sequential and parallel algorithms.

Objectives:- Study of parallel sorting algorithms like bubble sort and Merge sort

Prerequisites:- Student should know basic concepts of bubble sort and merge sort.

Theory:-

Merge sort

Merge sort is a sorting algorithm that uses a divide and conquer approach to sort an array or a list of elements. The algorithm works by recursively dividing the input array into two halves, sorting each half, and then merging the sorted halves to produce a sorted output.

The merge sort algorithm can be broken into the following steps:-

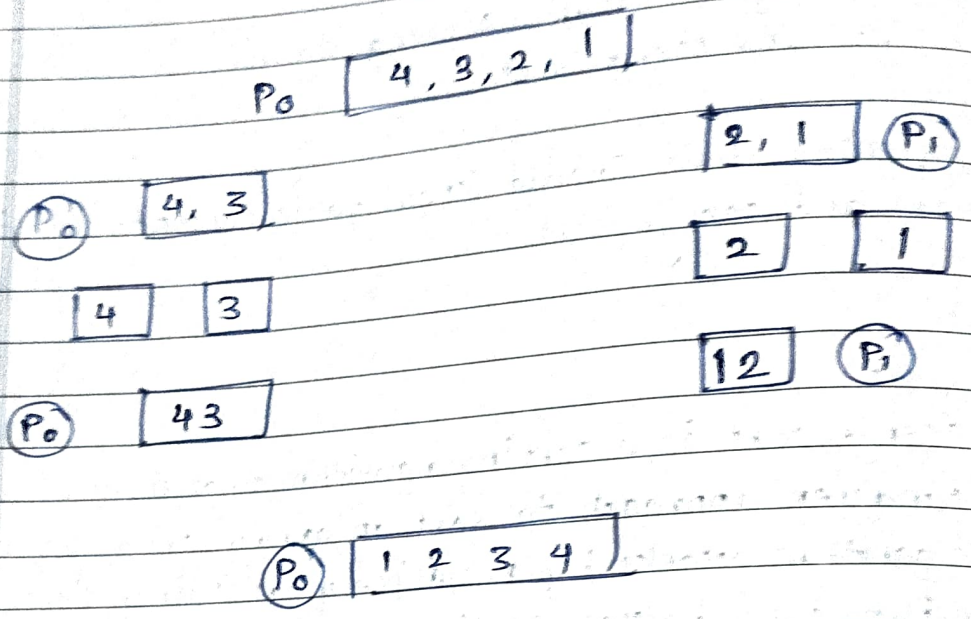
1) Divide the input array into two halves.
2) Recursively sort the left half of the array.
3) Recursively sort the right half of the array
4) Merge the two sorted halves into a single sorted output array.

Parallel Merge sort:-

- Parallelize processing of sub-problems.
- Max parallelization achieved with one processor per node (at each larger / weight)

Example:-

perform merge sort on the following elements given a processors P_0 & P_1 which processor is responsible for which comparison 4,3,2,1



Bubble sort:-

The idea of bubble sort is to compare the two adjacent elements. If they are not in the right cordes, switch them

Parallel bubblesort

- Implement as a pipeline.
- let local size = n/no_prac we divide the array in no.prac parts and each process connects the bubble sort on its part including comparing the last element with the first one belonging to the next thread

- Implement with the loop (instead of j<i) for (j=0; j<n ; j++)
  - For every iteration of i, each thread need to work until the precious thread has finishes that iteration before starting.
  - we will co-ordinate using barrier.

Example :- 4, 3, 1, 2.

Step 1 : 4 ⟷ 3 1 ⟷ 2

Step 2 :- 3 4 ⟷ 1 2

Step 3 :- 3 ⟷ 1 4 ⟷ 2

Step 4 :- 1 3 ⟷ 2 4

Step 5 :- 1 2 3 4

Conclusion :- Thus, we have studied paralled ~~in~~ bubble T and paralled merge sort implementation.

Assignment No-3

Page No. : 1.
Date. : / /

Aim:- Implement Min, MAX, sum and Average operations using parallel reduction.

Objectives:- To study and implementation of objectives based parallel programming model.

Prerequisites:- 64 bit open source linux or its derivatives
programming languages :- C/c++ / Java.

Theory:-

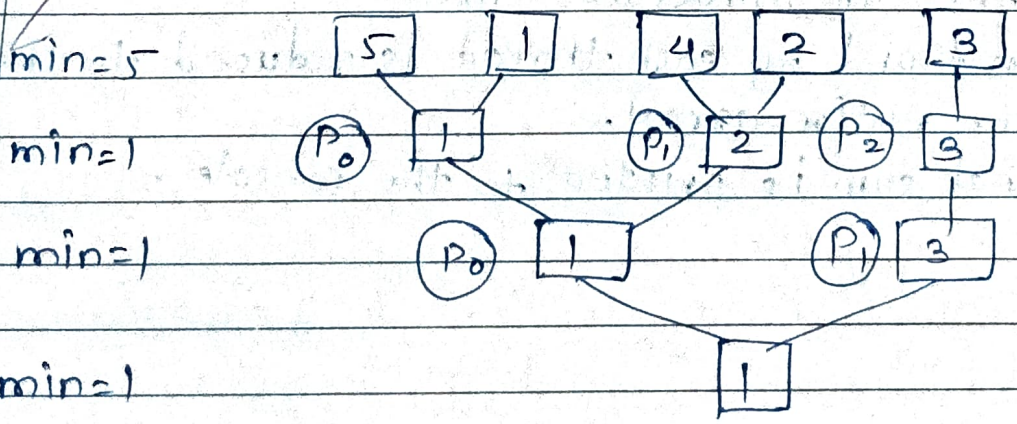
Parallel Reduction

Here's a function-wise manual on how to understand and run the sample c++ program that demonstrates how to implement Min, Max, Sum and Average operations using ||el reduction.

1) Min- Reduction Function:-

- The function takes in a vector of integers as input and finds the minimum value in the vector using parallel red^n.
- The openMP reduction clause is used with the 'min' operator to find the minimum value across all threads.
- The minimum value found by each thread is reduced to the overall minimum value of the entire array.

Example :- [5, 1, 4, 2, 3]



2) Max_Reduction function:-
- The function takes in a vector of integers as input and finds the maximum value in the vector using parallel reduction.
- The OpenMP reduction clause is used with the 'max' operator to find the maximum value across all threads.
- The maximum value found by each thread is reduced to the overall maximum value of the entire array.

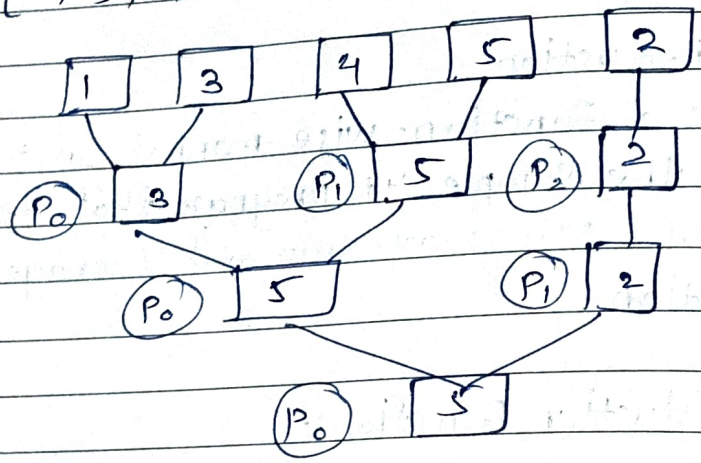
Example:- [1, 3, 4, 5, 2]

max = 1

max = 3

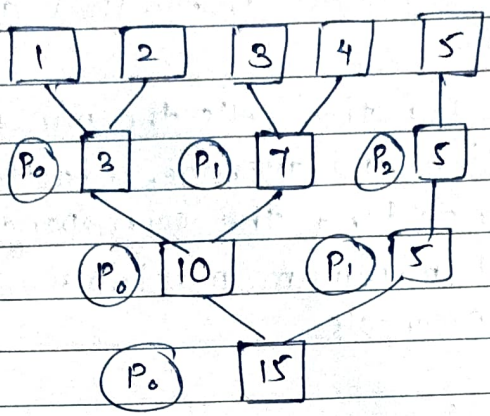
max = 5

max = 5



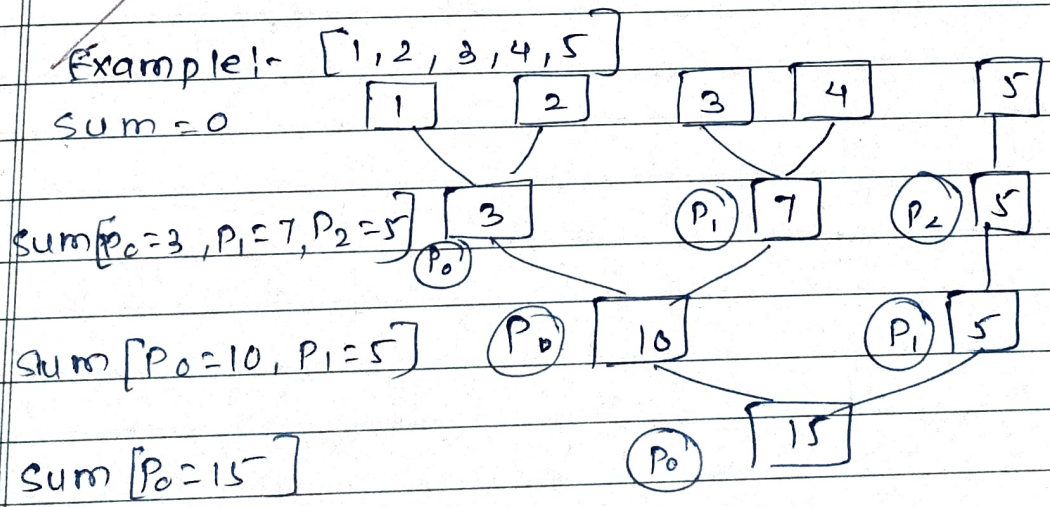
3) Sum_Reduction function:-
- The function takes in a vector of integers as input and finds the sum of all the values in the vector using reduction.
- The openMP reduction clause is used with the '+' operator to find the sum across all threads.
- The sum found by each thread is reduced to the overall sum of the entire array.
- The final sum is printed to the console.

example:- [1, 2, 3, 4, 5]



4) Average Reduction function:-

- The function takes in a vector of integers as input and finds the average of all the values in the vector using parallel reduction.
- The OpenMP reduction clause is used with the "+" operator to find the sum across all threads.
- The sum founded by each thread is reduced to the overall sum of the entire array.
- The final sum is divided by the size of the array to find the average.
- The final average value is printed to the console.



Page No. : 4
Date. : / /

Main function:-
- The function initializes a vector of integers with some values.
- The function calls the min_reduction, max_reduction, sum_reduction, and average_reduction functions on the input vector to find the corresponding values.
- final minimum, maximum, sum & average values are printed to the console

Conclusion:- Hence, we successfully studied and implemented min, max, sum & Avg operations using parallel reduction using openMP.

Assignment No:-4

Page No. 1.
Date

Aim:- Implement HPC application for AI/ML domain.

Prerequisites:- 64 bit open source linux or its derivative, c/c++ programming, openmp.

Theory

Hence we will develop an application using openmp libraries and developing an application which will perform tokenization in c language.

Tokenization:- Tokenization is the process of breaking down a text into smaller components called tokens. Tokens can be words, phrases, sentences or any other meaningful unit of text. Tokenization is a crucial step in any natural language processing applications such as text classification, named entity recognition and machine translation.

Open M.P can be used because openmp is a set of computer directives and library outlines for parallel programming in c and other languages. openmp can be used to parallelize the tokenization process which can improve the speed of the process when dealing with large texts. The basic idea behind parallel tokenization using openmp is to split the text into smaller chunks & process each chunk in parallel using multiple threads.

To tokenize a text using openmp we can split the text into smaller chunks.

Page No. 2
Date

thread can then process its chunk of the text independently and generate a set of tok
To ensure that each thread process an equal amount of text we can use 'omp' get 'thread num' function to get the ID of each Thread.

While openMP can be a powerful tool for parallelizing openmp tokenization, its importance to note that the performance gain will depe upon the specific use case and the size of t text being tokenized in some cases the over head of dividing the text into chunks and comb the results may out weight the benefits of tokenization.

The codes reads the line of text from a file called 'example.txt' and uses openMP to tokenize each line in it. Here's a breakdown of how it works,

1) The 'tokenize' fun takes a single line of text as input and tokenizes it using string fun from the 'string.h' library.

- The fun uses the 'omp get thread num' fun to get the ID of the current thread

2) The "main" function initializes the no. of threads to use and open the file for readin

Page No. 3.
Date

2) The 'main' fun initializes the number of threads to use and open the file for reading.

3) The #pragma omp parallel directive creates a parallel region.

4) The While (fgets(line, MAXLINELENGTH, fil)!=NULL) loop reads each line of text from the file

5) once all threads have finished processing their chunks of text the program closes the file & return 0.

Conclusion :- Hence, we successfully developed an application using HPC & using openMP for AI/ML domain.