HPC - Lab Assignment - 1

Name - Amit Kashinath Birajdar
Roll no - 19CS008
Class - BE Comp - B

Aim — Design and implement parallel Breadth first search and depth first search based on existing algorithm using OpenMP. Use a tree or undirected graph for BFS and DFS.

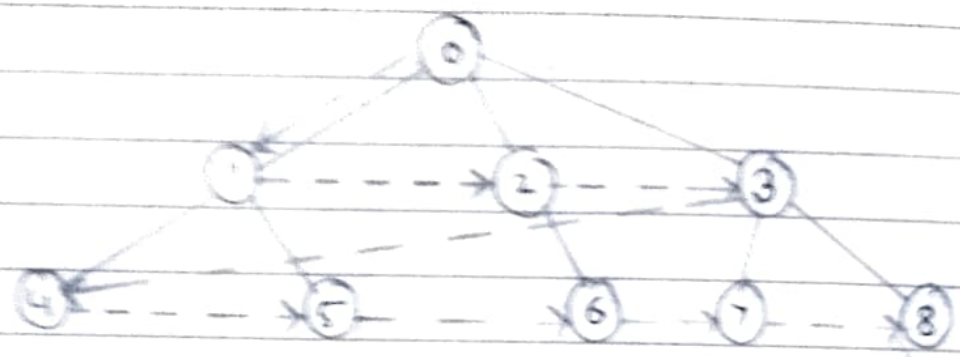
Requirement :— 64-bit Open Source linux or its derivative c/c++ programming language, openMP.

Theory :—

Breadth first Search :— There are many ways to traverse graph BFS is the most comm-only used approach. BFS is a traversing from algorithm where we should start traversing from a selected node (source node or starting node) and traverse the graph layerwise thus exploring the neighbours nodes (& nodes which are directly connected to source node.) we must then move towards the next-level neighbor nodes.

As the name BFS suggests. We are required to traverse the graph breadthwise as follows :—

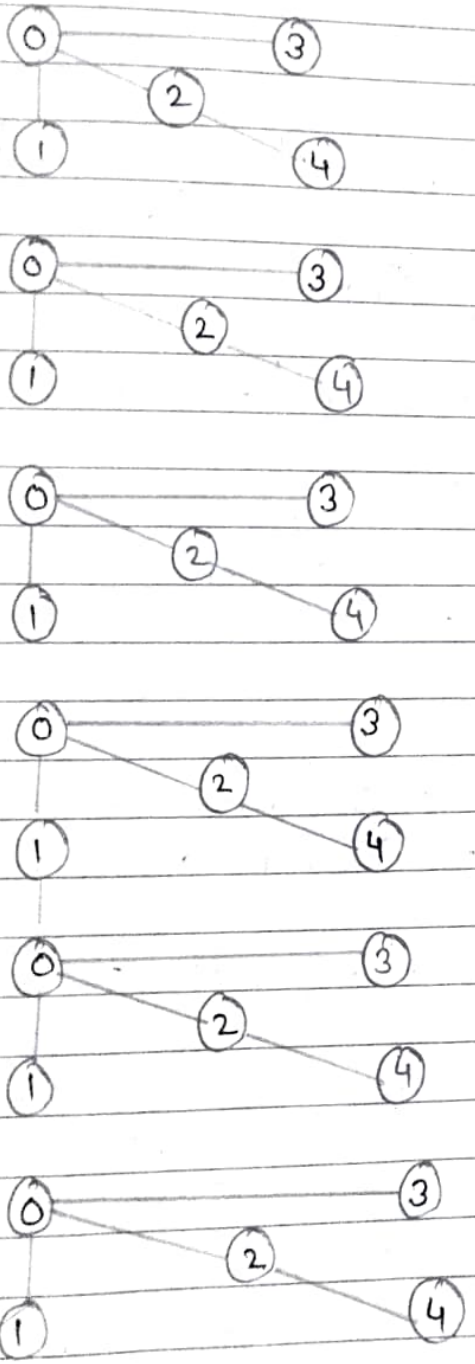
1) first move horizontaly and visit all the nodes of the current layer.



The distance between the nodes in the is comparatively lesser than the distance the node in layer 2. Therefore in BFS must traverse all the nodes in layer 1 we move to the nodes in layer 2.

Depth first Search :- We parallelize DFS by the work to be done among a number of processors. Each proc searches a disjoint part of search done in a depth first search pattern since each searches the space in a depth first se manner, the state space to be sear in efficiently represented by a stack. T depth of the stack is the depth is the node being currently explored test the -iveness of parallel DFS we have used solve the 15 - puzzle problem is a 4 square tray containining 15 square The remaining 16th squre is covered DFS is pervasive - algorithm often used building block for topological start -ity and planarity testing, among applications

Example: —



visited

stack

visited 0

stack 1, 2, 3

visited 0

stack 2, 3

visited 0, 1, 2

stack 4, 3

visited 0, 1, 2, 4

stack 3

visited 0, 1, 2, 4, 3

stack

Steps for Searching :—

In parallel BFS :

Step 1 : Start with the root node, ma
Step 2 : As the root node has no node
level, go to the next level.
Step 3 : Visit all adjacent node and
them visited.
Step 4 : Go to the next level and visi
the unvisited node.
Step 5 : Continue this process until a
nodes are visited or founded the
-ed element.

In parallel DFS :—

Step 1 : Consider a node (root node) the
not visited previously and mark
visited.
Step 2 : Visit the first adjacent successor
mark it.
Step 3 : Fall the successor node of the
-ed node are already visited or
doesn't have any more succor n

Conclusion :— Hence, we successfully studi
implemented parallel first sea
parallelel Depth first search

HPC Lab Assignment No - 2

Name :- Amit Kashinath Birajdar
Roll no :- 19CS008
Class :- BE Comp - B

Aim - Write a program to implement Parallel Bubble sort and merge Sort using OpenMP. Use existing algorithm and measure the performance of sequential and parallel algorithm.

Requirements :-

64 bit open source linux or its derivative, c/c++ programming, Open Mp.

Theory :-

Sorting :

Sorting is the process of arranging elements in a group of in a particular order if ascending order or in descending order, alphabetic order etc.

Parallel Sorting :- A sequential Sorting algorithm may not be efficient enough when we have to set sort a huge volume of data. Therefore, parallel algorithm are used in Sorting.

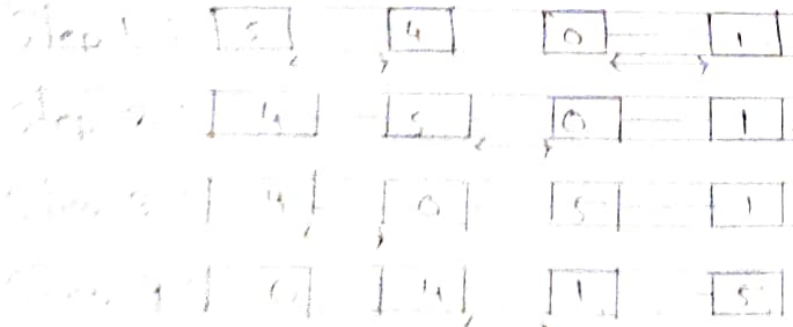
Bubble Sort:

The idea of bubble sort is to compare two adjacent elements and if they are not in right order, switch them. Do this comparing and switching (if necessary) until we reach to end of the array. Repeat this process from the begining of the array n times.

Parallel Bubble Sort:

It is implemented using pipeline. let localisze = n/proc. we divide the array in no-proc parts and each process executes the bubble sort on its part, including comparing the last element with the first are belonging to the next thread. Implement it with for loop (instead of (i<n)) for (j=0, j<n) and for every iteration of i each thread has to wait until the previous thread has finished.

Example for parallel bubble sorting:
[5, 4, 0, 1]

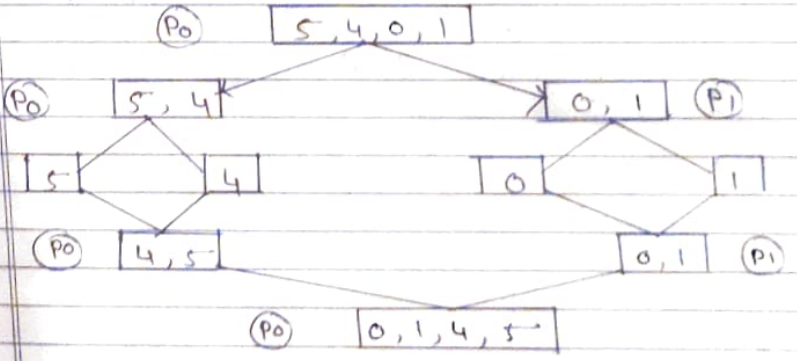


Merge Sort:

Merge sort is a sorting technique based on divide and conquer technique with worst-case time complexity (O(nlogn)). Merge sort divides the array into equal parts and then combines them in a sorted manner.

Parallel Merge Sort

eg 5, 4, 0, 1



Algorithm

for parallel Bubble Sort

Step 1 : for i=0 to n-2
Step 2 : if i is even then
Step 3 : for j=0 to (n+2)-1 in parallel
Step 4 : if A[2j] > A[2j+1] then

Step 5: Exchange $A[2i] \longleftrightarrow A[2i+1]$
Step 6: Else.
Step 7: for $j=0$ to $(n/2)-2$ do in par
Step 8: If $A[2j+1] > A[2j+2]$ then
Step 9: Exchange $A[2j+1] \longleftrightarrow A[2j+2]$
Step 10: Next i

Algorithm

Conclusion:-

Hence, we success fully st
and implemented parallel Bubble sort
parallel merge sort osing open mp.

HPC - Lab Assignment - 3

Name: Amit Kashinath Sirajdar
Roll no: 19CS008
Class: BE Comp-B

Aim:-

Implement Min, max, sum and average operation using parallel Reduction.

Requirements:-

64-bit open Source linux or its derivative. c/c++ programing, OpenMP.

Theory:

OpenMP:

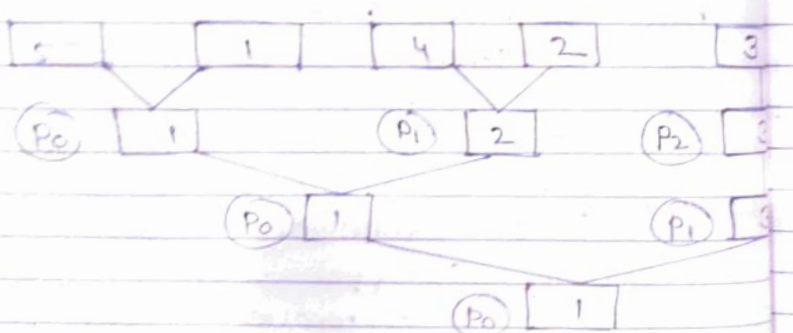
OpenMP is a parallel programming model that allows for the creations of parallel programs using a set of compiler directives, library routines and environment variables. OpenMP is particularly well suited for shared Memory architecture where multiple processors can access the same memory space. In OpenMP, parallelism is achieved through the use of threads, which are light weight that can be created and destroyed dynamically at runtime.

While implementation we used the "parallel for" construct to parallelize the for loop that iterates over the elements of the array. This construct splits the loop into smaller chunks, which are then executed in parallel by multiple threads.

Operation Minimum (min):

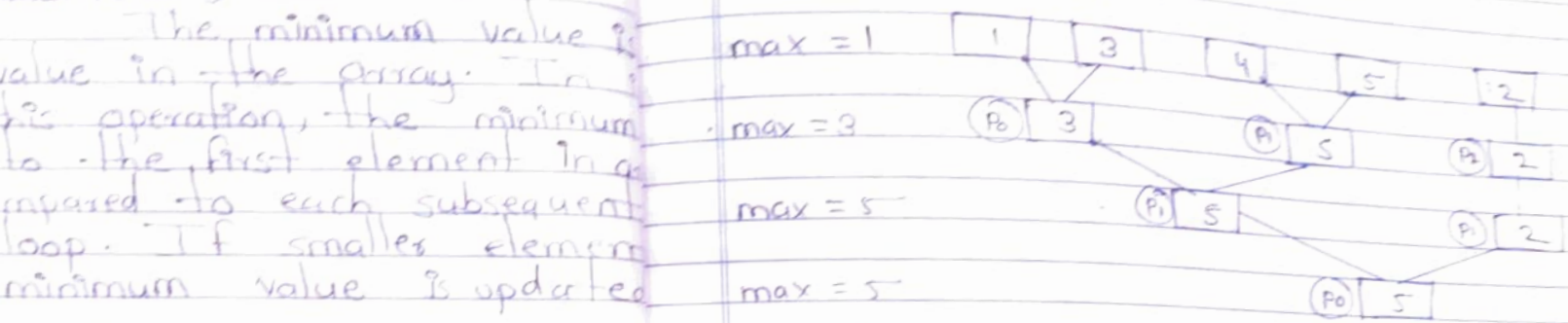
The minimum value is the smallest value in the array. In the execution of this operation, the minimum is initialized to the first element in the array. Then compared to each subsequent element in loop. If smaller element is found the minimum value is updated accordingly.

Ex - [5, 1, 4, 2, 3]



Operation Maximum (max): The maximum value is the largest value in the array. In implementation of this operation, the maximum value is initialized to the first element in the array, and then compared to each subsequent element in the loop. If a larger element is found, the maximum value is updated accordingly.

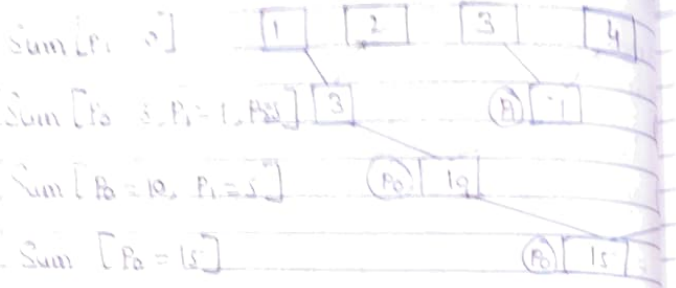
ex: [1, 3, 4, 5, 2]



Operation Sum of values (Sum) :-

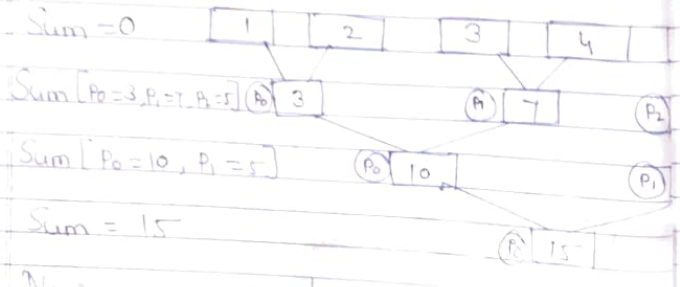
The sum of values is the total of the values in the array. In implementation of this operation, the sum is initialized to zero, and they by using parallel threads we do summation of two array elements in one thread. After that the summation value of all threads are added into pairs.

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



Average operation (avg): The average operation value is the mean value of all elements in the array. In implementation of this operation, the mean of two elements in an array is calculated and stored in one of the threads.

eg [1, 2, 3, 4, 5]



Now, avg = sum/n

where, avg → average value

Sum → Sumation of all elements in an array

n → number of elements in array

∴ average avg = Sum / n

= 15 / 5

avg = 3

∴ Average = 3

All of the above operations are commonly used in data analysis and processing tasks. By using Open MP to parallelize the loop that performs these operations, the code can be executed much faster than if were executed Serially. This can be especially useful when working with large datasets or computationally - intensive tasks.

Conclusion :-

Hence we successfully studied and implemented min, max, sum and average operations using parallel reduction using OpenMP.

HPC - Lab Assignment - 4

Name :- Amit Kashinath Birajdar
Roll no :- 19CS008
Class : BE Comp B

Aim :

Implement HPC application for AI/ML domain

Requirements :-

64-bit openSource linux or its derivatives
c/c++ programming
open MP

Theory :-

Here, 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 a smaller components called tokens. Tokens can be words, phrases, Sentences or any other meaningful unit of text.

Tokenization is a crucial step in many Natural language processing (NLP) applications such as text classification, named entity recognition and machine

translation there are many libraries for tokenizations Here are some

Open MP can be used because a set of compiler directives and library routines for parallel programming in other languages OpenMP can be used parallelize the tokenization process can improve the speed of process dealing with large terms. The idea behind parallel tokenization u OpenMP is to split the text into chunks and process each chunk in pa using multiple threads.

While openMP can be a powerful for parallelizing tokenization it's importe to note that the performance gain will depend on the specific use case and the size of the text being to In some cases the overhead of divi the text into chunks and comb the results may outweight the b of parallelization As with any optimiz technique, its important to measure performance.

The code reads lines of text from a file called 'example.txt' and use openMP to tokenize each line in parallel Here's a breakdown of how it works:

1) The 'tokenize' function takes a single line of text as i/p and tokenizes it using 'strtok function from the 'string h' library The function uses the 'amp-get thread-num' functio to get the in the current thread and uses a critical Section to print the line being processed and the tokens generated by each thread.

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

3) The '#pragma amp parallel' directive creates a parallel region and specifies the number of threads to use.

4) The while(fgetline MAX_LINE_LENGTH file) != NULL) loops reads each line of text from the file and pass it to be 'tokenize' function to be tokenized.

5) Once all threads have finished processing their chunks of text, the program closes the file and returns 0.

Conclusion :- Hence, we sucessfully de
an application using HPC openMP for
domain.