Assignment No: 1

- Aim : Design and implement parallel Bfs and Dfs based on existing algorithms using openMP. use a Tree or an undirected graph for BFS and DFS.

- Software & Hardware Requirement : window os, C++ / Java / python Compiler, editor etc.

- Theory :

- DFS :

- Dfs is an algorithm for Searching a tree or an undirected graph data structure.

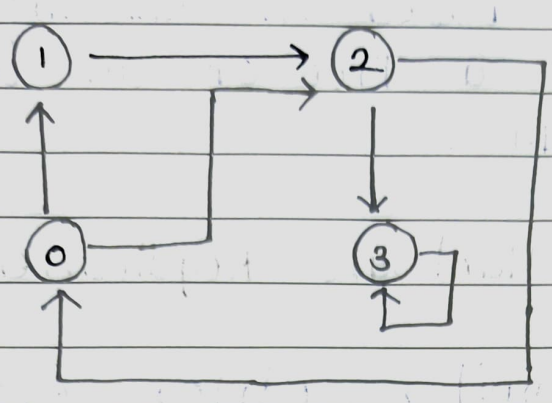
- Here, the Concept is to start jump from the the starting node known as root & traverse as for as possible in the same branch.

- If we get a node with no succesors node, we return & continue with the vertex which is yet to be visited.

Eg : Input n = 4 e = 6

0 → 1 , 0 → 2 , 0 → 2 , 1 → 2 , 2 → 0 , 2 → 3 , 3 → 3

- Graph :



Output : 1, 2, 0, 3

- BFS :

- BFS is an algorithm for Searching a tree or an undirected graph data structure.

- Here we start with a node and then visit all the adjacent nodes in the same level & then more to the adjacent successor node in the next level.

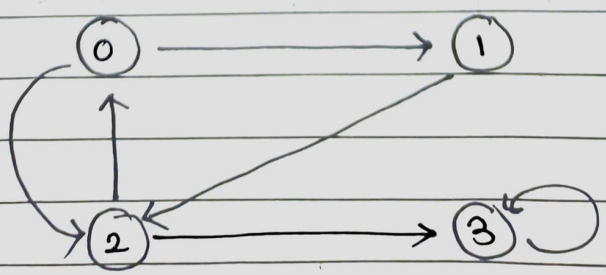
- It is also known as level by level search.

Eg :

I/P : n = 4 , n = 6.

1 → 2 , 2 → 0 , 2 → 3 , 3 → 3 , 0 → 1 , 0 → 2

Graph :



O/P : There are multiple ways to solve and soln for the graph.

way 1 → 2, 3, 0, 1

way 2 → 3, 0, 3, 1

- Algorithm :

- DFS :

1) Consider a node that is not visited previously & mark it visited
2) visit the first adjacent sucessor node and mark it visited.
3) If all the successors nodes of the considered node are already visited or it dosen't have any more Successor nodes return it to its present node.

- BFS :

1) Start with a root node & mark it visited.
2) As the root node has no node in the same level go to the next level.
3) visit all ajacent nodes and mark them visited.
4) Go to the next level and visit all the unvisited ajacent nodes.

- Conclusion :

After Completion of this practical we understand the Bfs and Dfs algorithm execution through parallel methods. We also get to known how to traverse a tree by Bfs and Dfs.

Assignment No: 2

- Aim : Write a program to implement paralled Bubble Sort and merge Sort using OpenMP. Use existing algorithms and measure the performance of Sequential and parallel algorithms.

- Software and Hardware Requirements : windows OS, C++ / Java / python Compiler / Editor.

- Theory :

1) Bubble Sort :

- It is the Simplest Sorting algorithm that works by repeatedly ~~supp~~ swapping the adjacent elements if they are in the wrong order.

- This algorithm is not suitable for large datasets as it is average & worst case time complexity is quite high.

- Algorithm :

i) Start with an array of unsorted.
ii) Define a function called "Bubble Sort" that takes in the array and the length of the array as parameters.
iii) In the function create variable called "Sorted" that is Set to true

iv) Create a for loop that iterates through the array starting at index '0' & ending at the length of the array.

v) within the for loop compare current element with the next element in the array.

vi) If the current is greater than the next swap their position & Sorted to false

vii) If Sorted is false call the bubble function again with the same array length as parameter

viii) If "Sorted" is true, the array is now Sorted & the function will return Sorted array call the buble Sort function with the unsorted array & its parameter to begin sorting process.

2) Merge Sort :

- It is defined as a Sorting algorithm works dividing an array into smaller sub array Sorting each sub array & then merging the Sorted Sub array back together to form the final Sorted array

- In simple we can say that the process of merge sort is to divide the array into 2 sub array.

- Sort each half & then merge the sorted array back together.

- This process is repeated until entire array is sorted.

- Algorithm :

1) Start.
2) Declare array & left, right, mid, variable.
3) perform merge function :

```
if left > right
    return.
mid > (left + right)/2
merge sort (array, left, mid).
merge sort (array, mid, right)
merge sort (array, left, mid, right);
```

4) Stop.

- Conclusion :

After completion of this practical we understand the bubble sort and merge sort execution using parallel programming.

Assignment No: 3

- Aim : Implement min, max, Sum and Average operation using parallel Reduction.

- Software and Hardware Requirement :
  - Parallel Computing architecture
  - Parallel programming models.
  - proficiency in programming language.

- Theory :

1) min - Reduction function.

- The function takes in a vector of integers as i/p & finds the minimum value in as the vector using parallel reduction.

- 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.

- The final minimum value is painted to the console.

2) max - Reduction function :

- The function takes in a vector of integers as i/p & finds the max value in the vector using parallel reduction.

- The open mp reduction clause is used with the max operator is find the max value across are threads.

- The max value found by each thread is reduced to the overall max value of the entire array.

3) Sum Reduction function :

- The function takes in vector of integers as i/p & finds the sum of all values in vector using parallel reduction.

- The OpenMP reduction clause is used with the '+' operator to the sum across all threads.

- The sum found by each thread is reduced to overall Sum of entire array.

4) Average Reduction function :

- The sum found by each thread is reduced to the overall sum of the entire array.

- The find sum is divided by the size of array to find Average

5) main function :

- The main function initialize a vector of integers with some values.

- The function calls the min reduction mark reduction Sum reduction & average reduction

6) Compiling and Running the program :

- You need to use C++ Compiler that Supports openmp such as g++ or lang open a terminal & navigator to the directory where program is saved.

- G++ openmp program Cpp -o program This command Compiles program & creates an executable file named "program" The "-fopenmp" flag tells compiler to enable openmp.

- Conclusion:

Parallel reduction is a powerful technique that allows us to perform these operation on large arrange efficiently. by dividing work among multiple threads running in parallel.

Assignment No : 4

- Aim : Write a CUDA program for addition of two large vectors.
- Prerequisite :

1) CUDA Concept.
2) Vector addition
3) How to execute program in CUDA program.

- What is CUDA :

1) CUDA (Compute unified Device Architecture) is a parallel computing platform & programming model developed by NVIDIA.

2) It allows develops to use the power of NVIDIA graphics processing units to accelerate computations tasks in various applications including Scientific computing ML & computer vision provides a set of programming APIs, libraries & tools that enable developers to write & execute code of NVIDIA GPUs

3) It supports popular programming model languages like c++, A python provides a simple programming model that abstracts array much of the Low level details of GPU architecture

- Steps for addition of two large vectors using CUDA.

1) Determine the size of the vectors:

In this step you need to define the size of vectors you need add.

2) Allocate memory of host:

In this step you need to allocate memory on the host for the two vectors that you want to add for result

3) Initialize the vectors:

In this step you need to initialize the how vectors that you want to add on the host.

4) Allocate memory on device:

In this step you need to allocate memory on device.

5) Copy the input vector from host to devices:

In this step, you need to copy the two i/p vectors from the host to device memory

6) Launch the kernel :

In this step you need to launch the CUDA kernel that will perform the addn opn.

7) Copy the result :

In this step you need the copy the result vector from me device memory to host memory.

8) Free memory on device :

In this step you need to free the memory that was allocated on device.

9) Free memory on host :

In this step you need to free the memory that was allocated on the host.

- Execution of program over CUDA environment :

1) Install CUDA Toolkit :

first you need to install me CUDA toolkit on your system.

2) Set your CUDA Environment :

Once the CUDA toolkit is installed you need to set up the CUDA environment on your System.

3) Write the CUDA program :

- You need to write a CUDA program that performs the addition of two large vectors.

4) Compile the CUDA program :

You need to compile the CUDA program the nucc compiler that comes with CUDA toolkit.
command :
nucc -o program_name program_name.cu.

5) Execute the file :

This will generate executable file. and this file will execute the program & perform addition of 2 vectors.

- Conclusion ! After execution of this practical we understand the CUDA program.