Sinhgad Institutes

STES's

RMD Sinhgad School of Engineering, Warje-58

Laboratory Practice V

Department of Computer Engineering
BE (2019 Course)

SINHGAD TECHNICAL EDUCATION SOCIETY'S

RMD SINHGAD SCHOOL OF ENGINEERING
Warje, Pune 411058

Department of Computer Engineering

Sinhgad Institutes

LABORATORY MANUAL
AY: 2023-24

LABORATORY PRACTICE V

BE Computer Engineering
Semester –II
Subject Code - 410255

TEACHING SCHEME	CREDIT	EXAMINATION
Practical: 2 Hrs / Week	01	TW: 50 Marks
		PR: 50 Marks

Prepared By:

Mrs. Pradnya Kasture
Mrs. Sonal Fatangare
Mrs. Jyoti Raghatwan
(Assistant Professor, Department of Computer Engineering)

RMD SINHGAD SCHOOL OF ENGINEERING, WARJE, PUNE-58

NAAC accredited with "A" grade

<u>CERTIFICATE</u>

This is to certify that

Mr./Ms._____

of Class_____**Roll No.**_____**has completed all the practical work/term work in subject**

_____

satisfactorily in the department of Computer Engineering as prescribed by Savitribai Phule Pune

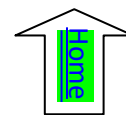**University during the academic year** _____

Staff In-charge   **Head of Department**   **Principal**

Date

Index

Sr. No.	Page No	Experiment Performed	Date	Sign	Remark
High Performance Computing					
1		Design and implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS and DFS .			
2		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.			
3		Implement Greedy search algorithm for Implement Min, Max, Sum and Average operations using Parallel Reduction			
4		Write a CUDA proram for 1. Addition of two large vectors 2. Matrix Multiplication using CUDA			
5		Implement HPC application for AI/ML domain.			
6		Implement any one Mini Project:			
7		Mini Project: Evaluate performance enhancement of parallel Quicksort Algorithm using MPI			
8		Mini Project: Implement Huffman Encoding on GPU			
9		Mini Project: Implement NonSerial Polyadic Dynamic Programming with GPU Parallelization			
Deep Learning					
1		Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.			
2		Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset			
3		Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition dataset https://archive.ics.uci.edu/ml/datasets/letter+recognition			
4		Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories			
5		Mini Project: Human Face Recognition			
6		Mini Project: Gender and Age Detection: predict if a person is a male or female and also their age			
7		Mini Project: Colorizing Old B&W Images: color old black and white images to colorful images			

Savitribai Phule Pune University **Fourth Year of Computer Engineering (2019 Course)** **410255: Laboratory Practice V**		
Teaching Scheme: **Practical: 2 Hours/Week**	**Credit** **01**	**Examination Scheme** **Term Work: 50 Marks** **Practical: 50 Marks**

Companion Course: High Performance Computing(410250), Deep Learning(410251)

Course Objectives:

- To understand and implement searching and sorting algorithms.
- To learn the fundamentals of GPU Computing in the CUDA environment.
- To illustrate the concepts of Artificial Intelligence/Machine Learning(AI/ML).
- To understand Hardware acceleration.
- To implement different deep learning models.

Course Outcomes:

CO1: Analyze and measure performance of sequential and parallel algorithms.
CO2: Design and Implement solutions for multicore/Distributed/parallel environment.
CO3: Identify and apply the suitable algorithms to solve AI/ML problems.
CO4: Apply the technique of Deep Neural network for implementing Linear regression andclassification.
CO5: Apply the technique of Convolution (CNN) for implementing Deep Learning models.
CO6: Design and develop Recurrent Neural Network (RNN) for prediction.

Guidelines for Instructor's Manual

Laboratory Practice V is for practical hands on for core courses High Performance Computing and Data Learning. The instructor's manual is to be developed as a hands-on resource and as ready reference. The instructor's manual need to include prologue (about University/program/ institute/ department/foreword/ preface etc), University syllabus, conduction and Assessment guidelines, topics under consideration-concept, objectives, outcomes, set of typical applications/assignments/ guidelines, references among others.

Guidelines for Student's Laboratory Journal

The laboratory assignments are to be submitted by student in the form of journal. Journal may

consists of prologue, Certificate, table of contents, and handwritten write-up of each assignment (Title, Objectives, Problem Statement, Outcomes, software and Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory- Concept in brief, Algorithm/Database design, test cases, conclusion/analysis). Program codes with sample output of allperformed assignments are to be submitted as softcopy.

Guidelines for Laboratory /Term Work Assessment

Continuous assessment of laboratory work is to be done based on overall performance and lab assignments performance of student. Each lab assignment assessment will assign grade/marks based on parameters with appropriate weightage. Suggested parameters for overall assessment as well as each lab assignment assessment include- timely completion, performance, innovation, efficient codes, punctuality and neatness reserving weightage for successful mini-project completion and related documentation.

Guidelines for Practical Examination

- Both internal and external examiners should jointly frame suitable problem statements for practical examination based on the term work completed.

- During practical assessment, the expert evaluator should give the maximum weightage to the satisfactory implementation of the problem statement.
- The supplementary and relevant questions may be asked at the time of evaluation to test the student's for advanced learning, understanding of the fundamentals, effective and efficient implementation.
- Encouraging efforts, transparent evaluation and fair approach of the evaluator will not create any uncertainty or doubt in the minds of the students. So adhering to these principles will consummate our team efforts to the promising boost to the student's academics.

Guidelines for Laboratory Conduction

- List of recommended programming assignments and sample mini-projects is provided for reference.
- Referring these, Course Teacher or Lab Instructor may frame the assignments/mini-project by understanding the prerequisites, technological aspects, utility and recent trends related tothe respective courses.
- Preferably there should be multiple sets of assignments/mini-project and distribute among batches of students.
- Real world problems/application based assignments/mini-projects create interest among learners serving as foundation for future research or startup of business projects.
- Mini-project can be completed in group of 2 to 3 students.

- Software Engineering approach with proper documentation is to be strictly followed.
- Use of open source software is to be encouraged.
- Instructor may also set one assignment or mini-project that is suitable to respective course beyond the scope of syllabus.

Operating System recommended :- 64-bit Open source Linux or its derivative

Programming Languages: Object Oriented Languages

C++/JAVA/PYTHON/R

Programming tools recommended: Front End: Java/Perl/PHP/Python/Ruby/.net, Backend : MongoDB/MYSQL/Oracle, Database Connectivity : ODBC/JDBC

Suggested List of Laboratory Experiments/Assignments

410250 : High Performance Computing

Any 4 Assignments and 1 Mini Project are Mandatory

Group 1

1.	Design and implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS and DFS .
2.	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.
3.	Implement Min, Max, Sum and Average operations using Parallel Reduction.
4.	Write a CUDA Program for : 1. Addition of two large vectors 2. Matrix Multiplication using CUDA C
5.	Implement HPC application for AI/ML domain.

Group 2

6.	Mini Project: Evaluate performance enhancement of parallel Quicksort Algorithm using MPI
7.	Mini Project: Implement Huffman Encoding on GPU
8.	Mini Project: Implement Parallelization of Database Query optimization
9.	Mini Project: Implement Non-Serial Polyadic Dynamic Programming with GPU Parallelization

410251 : Deep Learning

Any 3 Assignments and 1 Mini Project are Mandatory

Group 1

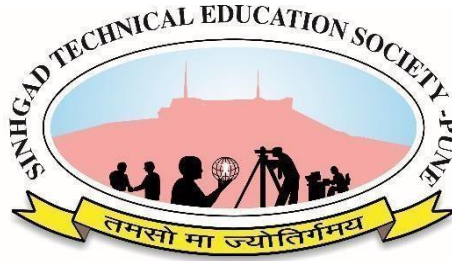|	
1.	**Linear regression by using Deep Neural network:** Implement Boston housing price predictionproblem by Linear regression using Deep Neural network. Use Boston House price predictiondataset.
2.	**Classification using Deep neural network** (Any One from the following) 1. Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition datasethttps://archive.ics.uci.edu/ml/datasets/letter+recognition 2. Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset
3.	**Convolutional neural network (CNN)** (Any One from the following) • Use any dataset of plant disease and design a plant disease detection system using CNN. • Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.
4.	**Recurrent neural network (RNN)** Use the Google stock prices dataset and design a time seriesanalysis and prediction system using RNN.
Group 2	
5.	**Mini Project:** Human Face Recognition
6.	**Mini Project:** Gender and Age Detection: predict if a person is a male or female and also their age
7.	**Mini Project:** Colorizing Old B&W Images: color old black and white images to colorful images

@The CO-PO Mapping Matrix												
CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	1	-	1	1	-	2	1	-	-	-	-	-
CO2	1	2	1	-	-	1	-	-	-	-	-	1
CO3	-	1	1	1	1	1	-	-	-	-	-	-
CO4	3	3	3	-	3	-	-	-	-	-	-	-
CO5	3	3	3	3	3	-	-	-	-	-	-	-
CO6	3	3	3	3	3	-	-	-	-	-	-	-
CO7	3	3	3	3	3		-	-	-	-	-	-

Sinhgad Institutes

Laboratory Practice V

High Performance Computing

Group A

Assignment No: 1(A)

Title of the Assignment: Design and implement Parallel Breadth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS

Objective of the Assignment: Students should be able to perform Parallel Breadth First Search based on existing algorithms using OpenMP

Prerequisite:

1.   Basic of programming language

2.   Concept of BFS

3.   Concept of Parallelism

Contents for Theory:

1.   What is BFS?

2.   Example of BFS

3.   Concept of OpenMP

4.   How Parallel BFS Work

5.   Code Explanation with Output

What is 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 between two nodes, solving puzzles, and searching through a tree or graph.
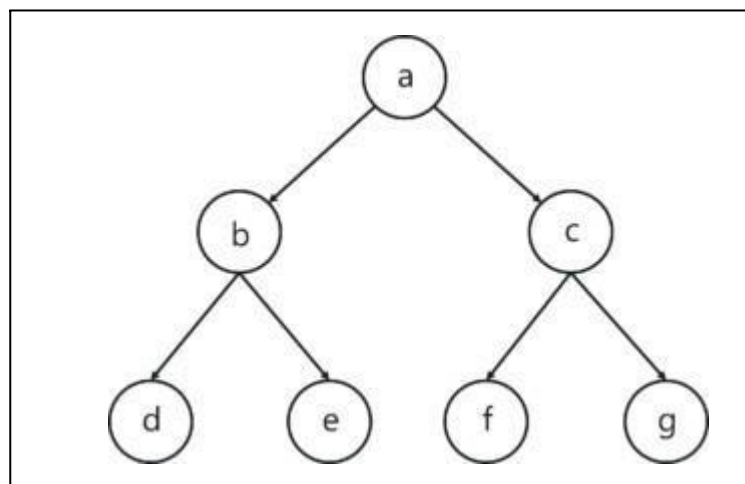
Example of BFS

Now let's take a look at the steps involved in traversing a graph by using Breadth-First Search:

Step 1: Take an Empty Queue.

Step 2: Select a starting node (visiting a node) and insert it into the Queue.

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

Step 4: Print the extracted node.

Concept of OpenMP

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.

- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.

- OpenMP programs are designed to take advantage of the shared-memory architecture of modern processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to execute a parallel region of the code, and then waits for all threads to complete before continuing with the sequential part of the code.

- OpenMP is widely used in scientific computing, engineering, and other fields that require high-performance computing. It is supported by most modern compilers and is available on a widerange of platforms, including desktops, servers, and supercomputers.

How Parallel BFS Work

- Parallel BFS (Breadth-First Search) is an algorithm used to explore all the nodes of a graph or tree

systematically in parallel. It is a popular parallel algorithm used for graph traversal in distributed computing, shared-memory systems, and parallel clusters.

● The parallel 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 be visited 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, and then exchanges the nodes that are adjacent to the current level with other threads or processors. This is done to ensure that the nodes at the next level are visited by the next iteration of the algorithm.

● The parallel BFS 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 or processors.

● The parallel BFS algorithm terminates when all nodes have been visited or when a specified node has been found. The result of the algorithm is the set of visited nodes or the shortest path from the root node to the target node.

● Parallel BFS can be implemented using different parallel programming models, such as OpenMP, MPI, CUDA, and others. The performance of the algorithm depends on the number of threads or processors used, the size of the graph, and the communication overhead between the threads or processors.

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

Assignment Question

1. **What if BFS?**
2. **What is OpenMP?What is its significance in parallel programming?**
3. **Write down applications of Parallel BFS**
4. **How can BFS be parallelized using OpenMP? Describe the parallel BFS algorithm using OpenMP.**
5. **Write Down Commands used in OpenMP?**

Reference link
● https://www.edureka.co/blog/breadth-first-search-algorithm/

Group A

Assignment No: 1(B)

Title of the Assignment: Design and implement Parallel Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for DFS

Objective of the Assignment: 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

Contents for Theory:

1.   What is DFS?

2.   Example of DFS

3.   Concept of OpenMP

4.   How Parallel DFS Work

What is 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 between two vertices or to traverse a graph in a systematic way. The algorithm starts at the root node and 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 very large graphs. The iterative approach uses a stack to keep track of nodes to be explored and is preferred for larger graphs.

DFS can also be used to detect cycles in a graph. If a cycle exists in a graph, the DFS algorithm will eventually reach a node that has already been visited, indicating that a cycle exists.

A standard DFS implementation puts each vertex of the graph into one of two categories:

1.   Visited
2.   Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
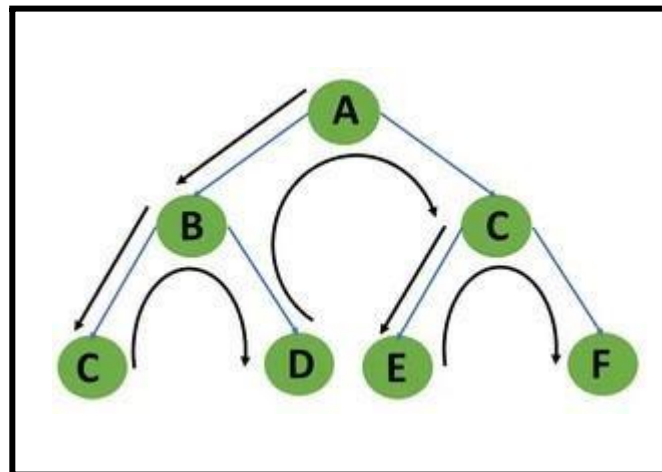
Example of DFS:

To implement DFS traversal, you need to take the following stages.

Step 1: Create a stack with the total number 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 and 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, and 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.

Consider the following graph as an example of how to use the dfs algorithm.
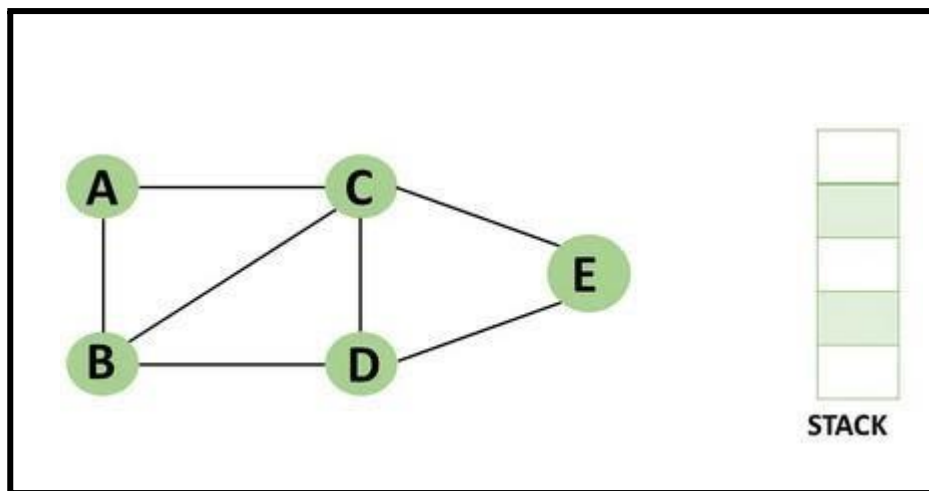
Step 1: Mark vertex A as a visited source node by selecting it as a source node.

- You should push vertex A to the top of the stack.
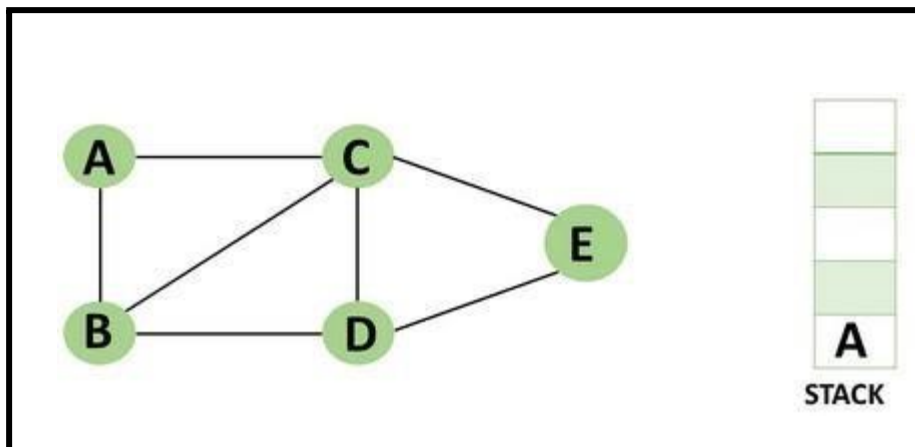
Step 2: Any nearby unvisited vertex of vertex A, say B, should be visited.

You should push vertex B to the top of the stack
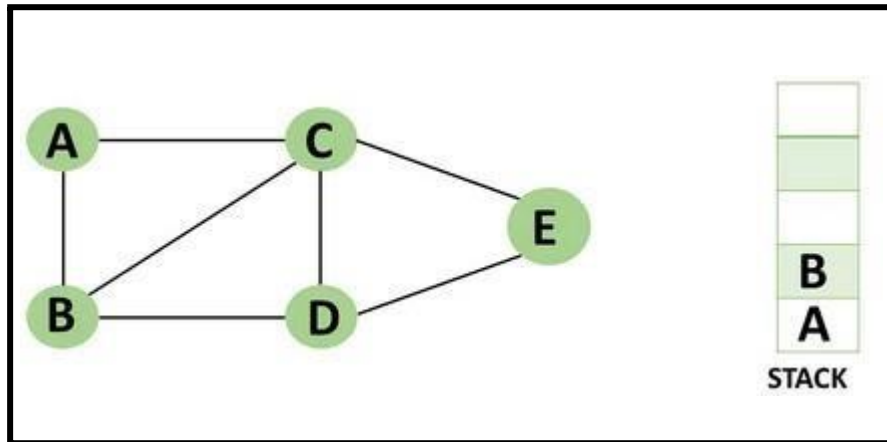
Step 3: From vertex C and D, visit any adjacent unvisited vertices of vertex B. Imagine you have chosen vertex C, and you want to make C a visited vertex.

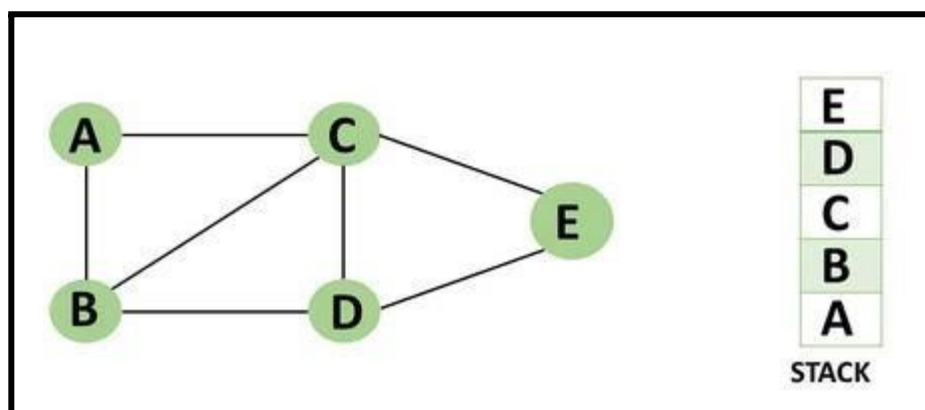- Vertex C is pushed to the top of the stack.
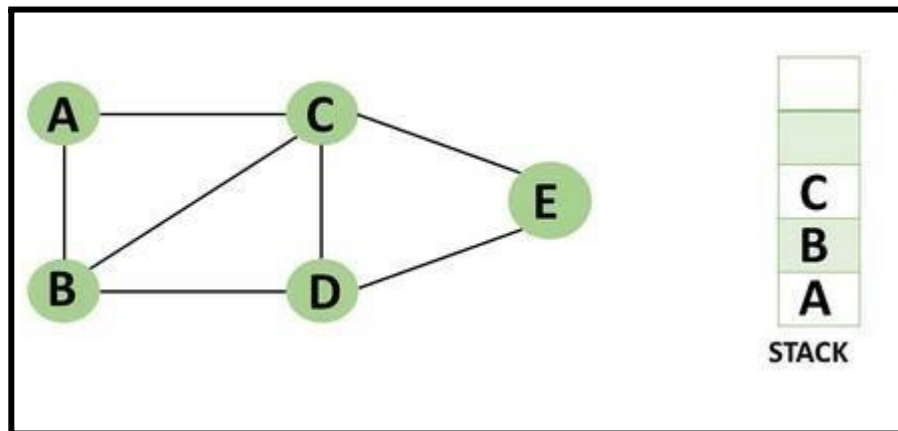
Step 4: You can visit any nearby unvisited vertices of vertex C, you need to select vertex D and designate it as a visited vertex.
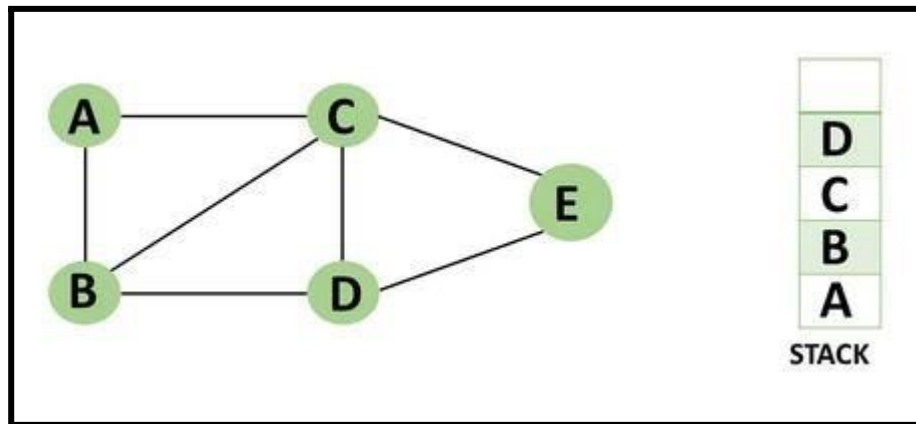
- Vertex D is pushed to the top of the stack.
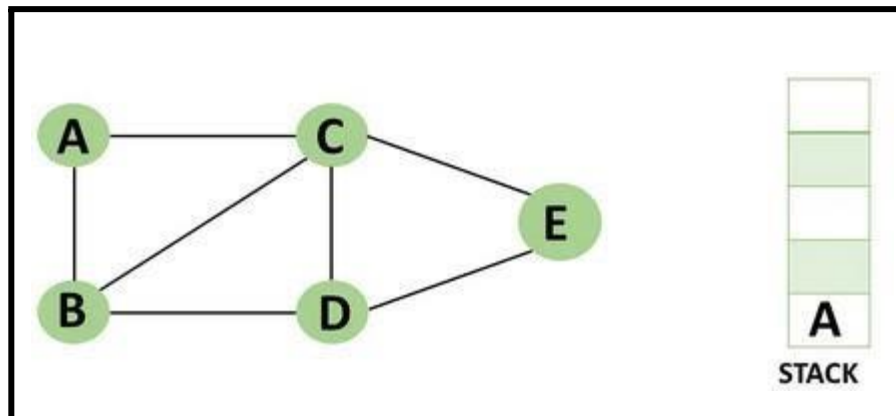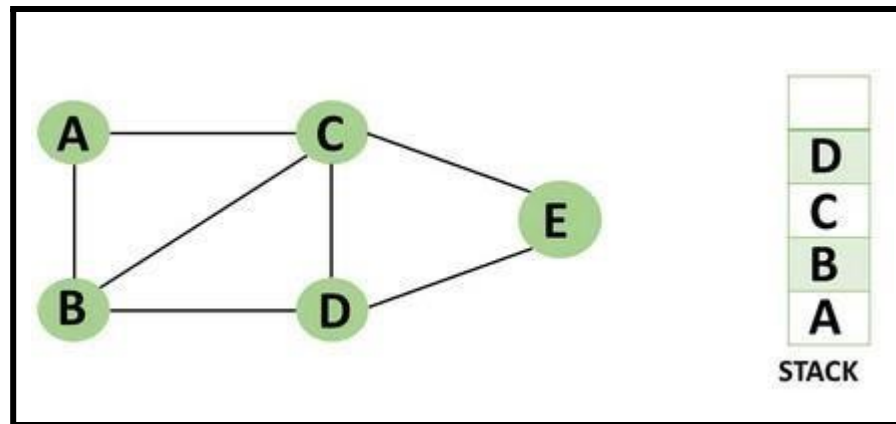
Step 5: Vertex E is the lone unvisited adjacent vertex of vertex D, thus marking it as visited.
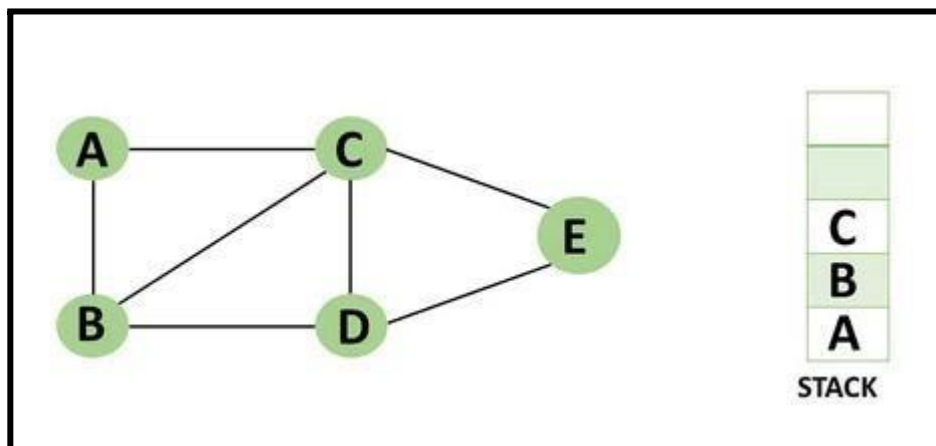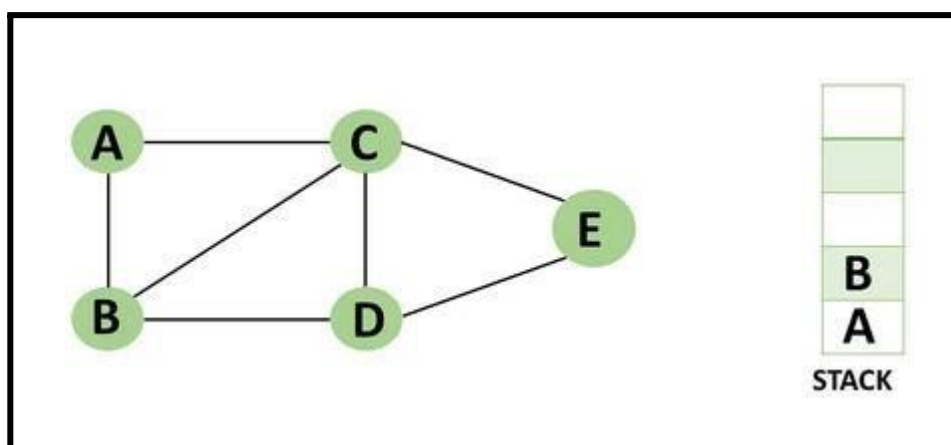
- Vertex E should be pushed to the top of the stack.

Step 6: Vertex E's nearby vertices, namely vertex C and D have been visited, pop vertex E from the stack.
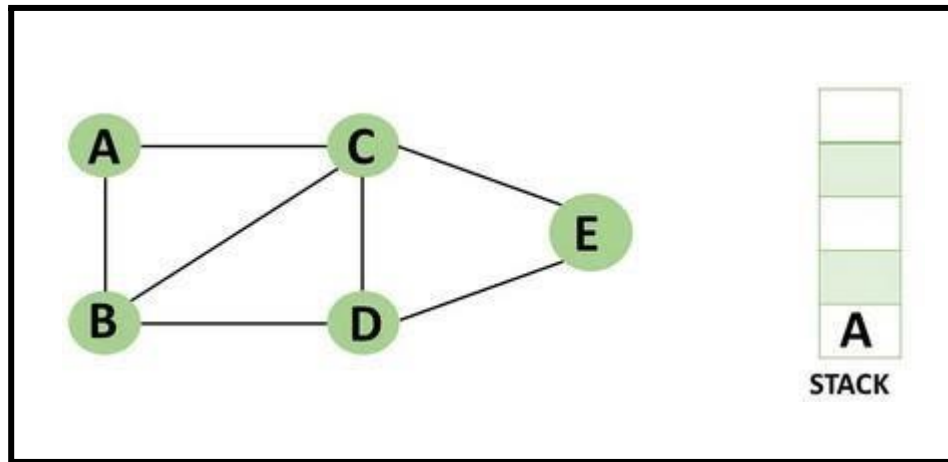
Step 7: Now that all of vertex D's nearby vertices, namely vertex B and C, have been visited, pop vertex D from the stack.
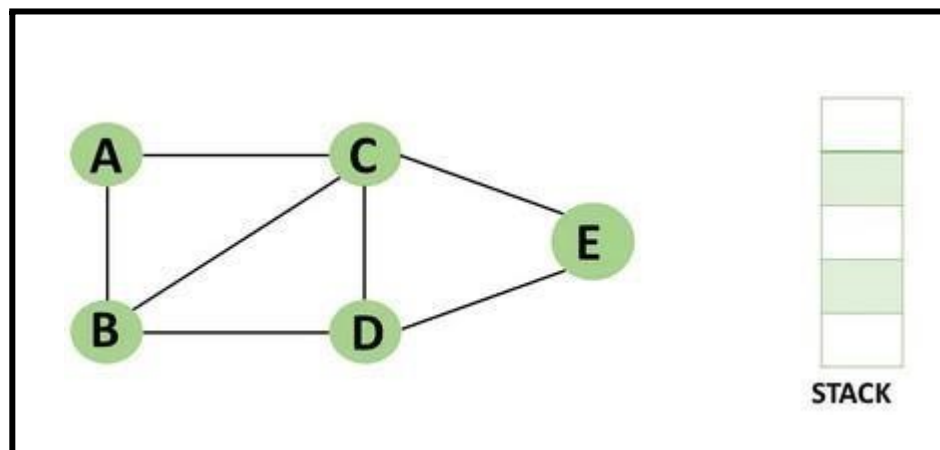

Step 8: Similarly, vertex C's adjacent vertices have already been visited; therefore, pop it from the stack.

Step 9: There is no more unvisited adjacent vertex of b, thus pop it from the stack.

Step 10: All of the nearby vertices of Vertex A, B, and C, have already been visited, so pop vertex A from the stack as well.

Concept of OpenMP

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.

- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.

- OpenMP programs are designed to take advantage of the shared-memory architecture of modern processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to

execute a parallel region of the code, and then waits for all threads to complete before continuing with the sequential part of the code.

How Parallel DFS Work

- Parallel Depth-First Search (DFS) is an algorithm that explores the depth of a graph structure to search for nodes. In contrast to a serial DFS algorithm that explores nodes in a sequential manner, parallel DFS algorithms explore nodes in a parallel manner, providing a significant speedup in large graphs.

- Parallel DFS works by dividing the graph into smaller subgraphs that are explored simultaneously. Each processor or thread is assigned a subgraph to explore, and they work independently to explore the subgraph using the standard DFS algorithm. During the exploration process, the nodes are marked as visited to avoid revisiting them.

- To explore the subgraph, the processors maintain a stack data structure that stores the nodes in the order of exploration. The top node is picked and explored, and its adjacent nodes are pushed onto the stack for further exploration. The stack is updated concurrently by the processors as they explore their subgraphs.

- Parallel DFS can be implemented using several parallel programming models such as OpenMP, MPI, and CUDA. In OpenMP, the #pragma omp parallel for directive is used to distribute the work among multiple threads. By using this directive, each thread operates on a different part of the graph, which increases the performance of the DFS algorithm.

Conclusion-   In this way we can achieve parallelism while implementing DFS

Assignment Question

1. **What if DFS?**
2. **Write a parallel Depth First Search (DFS) algorithm using OpenMP**
3. **What is the advantage of using parallel programming in DFS?**
4. **How can you parallelize a DFS algorithm using OpenMP?**
5. **What is a race condition in parallel programming, and how can it be avoided in OpenMP?**

Reference link
- https://www.programiz.com/dsa/graph-dfs
- https://www.simplilearn.com/tutorials/data-structure-tutorial/dfs-algorithm

Group A

Assignment No: 2(B)

Title of the Assignment: Write a program to implement Parallel Bubble Sort. Use existing algorithms and measure the performance of sequential and parallel algorithms.

Objective of the Assignment: Students should be able to Write a program to implement Parallel Bubble Sort and can measure the performance of sequential and parallel algorithms.

Prerequisite:

1.   Basic of programming language

2.   Concept of Bubble Sort

3.   Concept of Parallelism

Contents for Theory:

1.   What is Bubble Sort? Use of Bubble Sort

2.   Example of Bubble sort?

3.   Concept of OpenMP

4.   How Parallel Bubble Sort Work

5.   How to measure the performance of sequential and parallel algorithms?

What is Bubble Sort?

Bubble Sort is a simple sorting algorithm that works by repeatedly swapping adjacent elements if they are in the wrong order. It is called "bubble" sort because the algorithm moves the larger elements towards the end of the array in a manner that resembles the rising of bubbles in a liquid.

The basic algorithm of Bubble Sort is as follows:

1. Start at the beginning of the array.
2. Compare the first two elements. If the first element is greater than the second element, swap them.
3. Move to the next pair of elements and repeat step 2.
4. Continue the process until the end of the array is reached.
5. If any swaps were made in step 2-4, repeat the process from step 1.

The time complexity of Bubble Sort is O(n^2), which makes it inefficient for large lists. However, it has the advantage of being easy to understand and implement, and it is useful for educational purposes and for sorting small datasets.

Bubble Sort has limited practical use in modern software development due to its inefficient time complexity of O(n^2) which makes it unsuitable for sorting large datasets. However, Bubble Sort has some advantages and use cases that make it a valuable algorithm to understand, such as:

1. Simplicity: Bubble Sort is one of the simplest sorting algorithms, and it is easy to understand and implement. It can be used to introduce the concept of sorting to beginners and as a basis for more complex sorting algorithms.

2. Educational purposes: Bubble Sort is often used in academic settings to teach the principles of sorting algorithms and to help students understand how algorithms work.

3. Small datasets: For very small datasets, Bubble Sort can be an efficient sorting algorithm, as its overhead is relatively low.

4. Partially sorted datasets: If a dataset is already partially sorted, Bubble Sort can be very efficient. Since Bubble Sort only swaps adjacent elements that are in the wrong order, it has a low number of operations for a partially sorted dataset.

5. Performance optimization: Although Bubble Sort itself is not suitable for sorting large datasets, some of its techniques can be used in combination with other sorting algorithms to optimize their performance. For example, Bubble Sort can be used to optimize the performance of Insertion Sort by reducing the number of comparisons needed.

Example of Bubble sort

Let's say we want to sort a series of numbers 5, 3, 4, 1, and 2 so that they are arranged in ascending order…

The sorting begins the first iteration by comparing the first two values. If the first value is greater than the second, the algorithm pushes the first value to the index of the second value.

First Iteration of the Sorting

Step 1: In the case of 5, 3, 4, 1, and 2, 5 is greater than 3. So 5 takes the position of 3 and the numbers become 3, 5, 4, 1, and 2.
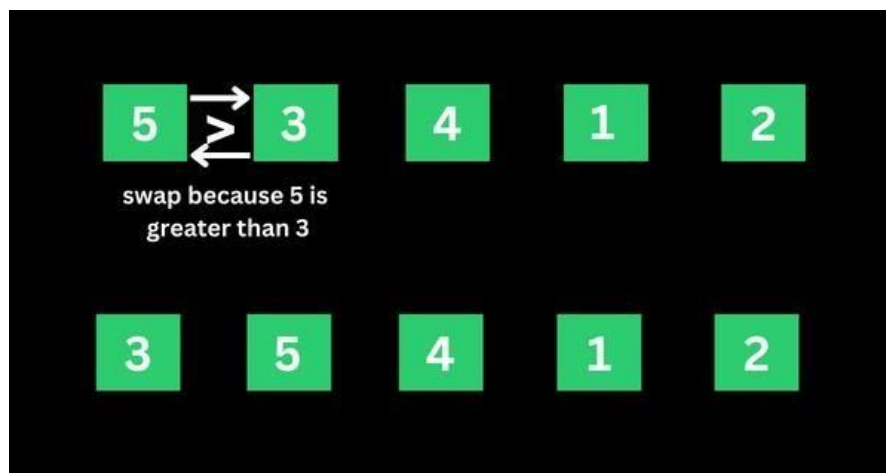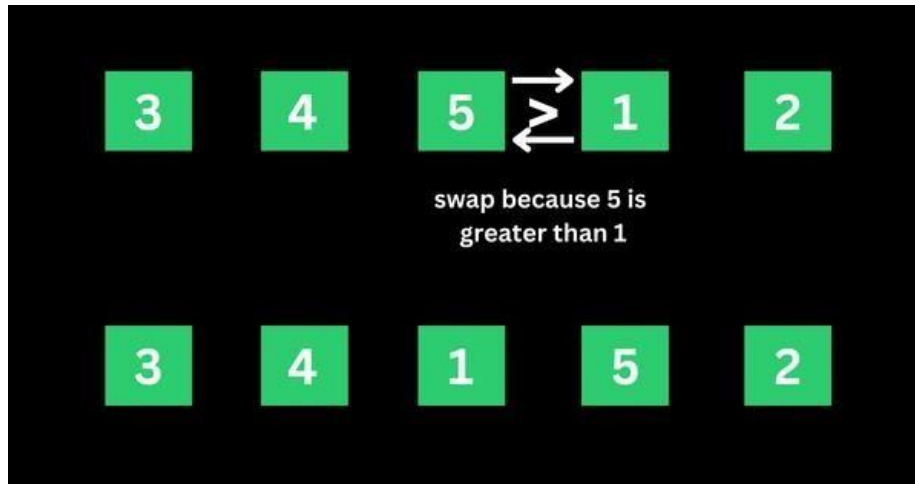

Step 2: The algorithm now has 3, 5, 4, 1, and 2 to compare, this time around, it compares the next two values, which are 5 and 4. 5 is greater than 4, so 5 takes the index of 4 and the values now become 3, 4, 5, 1, and 2.
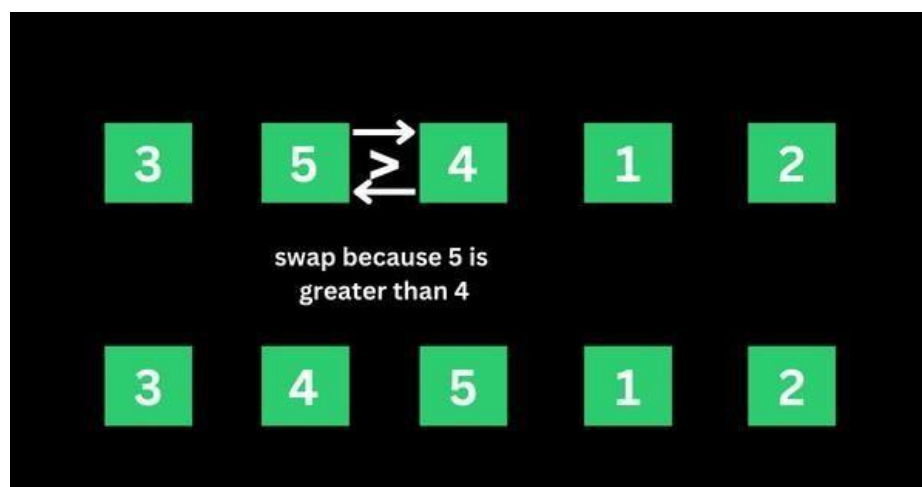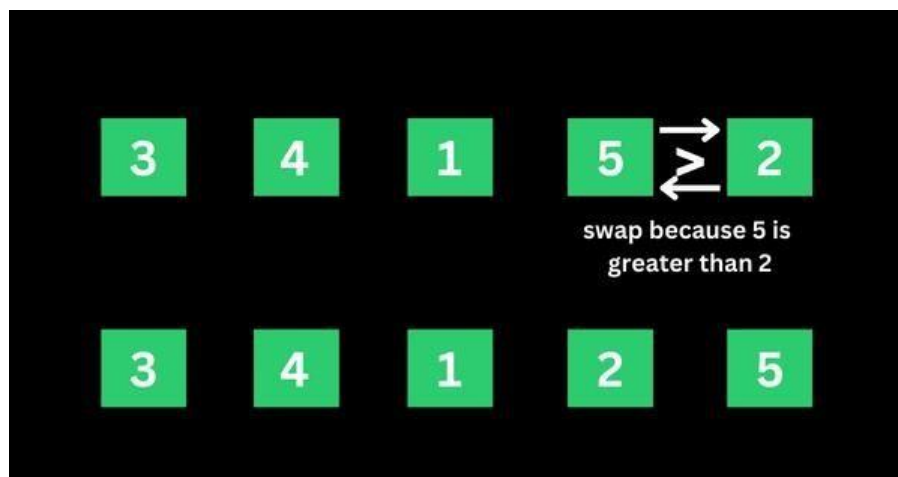

Step 3: The algorithm now has 3, 4, 5, 1, and 2 to compare. It compares the next two values, which are 5

and 1. 5 is greater than 1, so 5 takes the index of 1 and the numbers become 3, 4, 1, 5, and 2.


Step 4: The algorithm now has 3, 4, 1, 5, and 2 to compare. It compares the next two values, which are 5 and 2. 5 is greater than 2, so 5 takes the index of 2 and the numbers become 3, 4, 1, 2, and 5.

That's the first iteration. And the numbers are now arranged as 3, 4, 1, 2, and 5 – from the initial 5, 3, 4, 1, and 2. As you might realize, 5 should be the last number if the numbers are sorted in ascending order. This means the first iteration is really completed.

Second Iteration of the Sorting and the Rest

The algorithm starts the second iteration with the last result of 3, 4, 1, 2, and 5. This time around, 3 is smaller than 4, so no swapping happens. This means the numbers will remain the same.
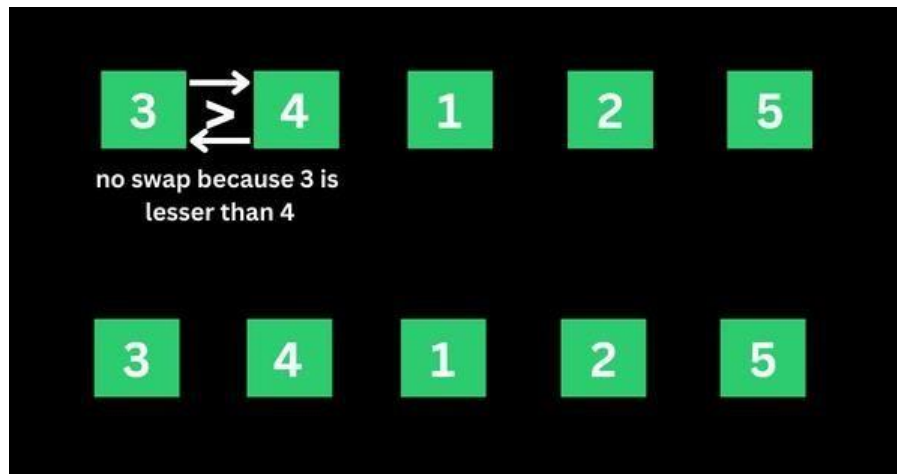
The algorithm proceeds to compare 4 and 1. 4 is greater than 1, so 4 is swapped for 1 and the numbers become 3, 1, 4, 2, and 5.
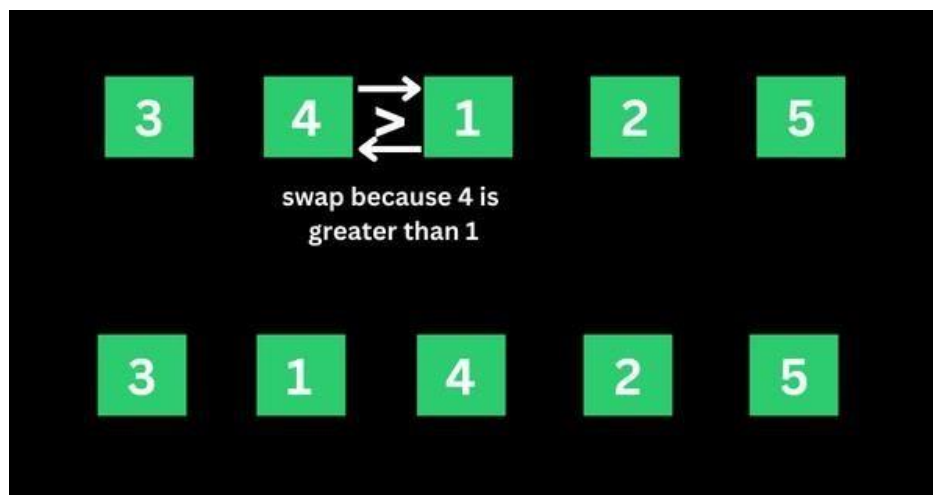
The algorithm now proceeds to compare 4 and 2. 4 is greater than 2, so 4 is swapped for 2 and the numbers become 3, 1, 2, 4, and 5.
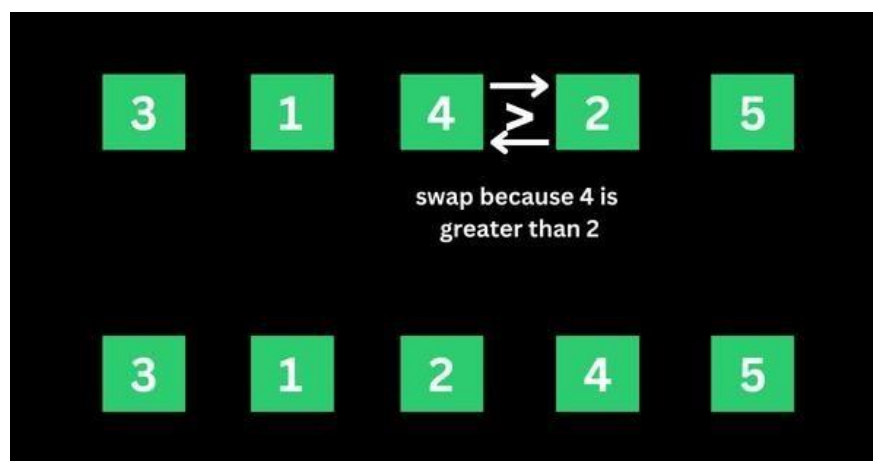
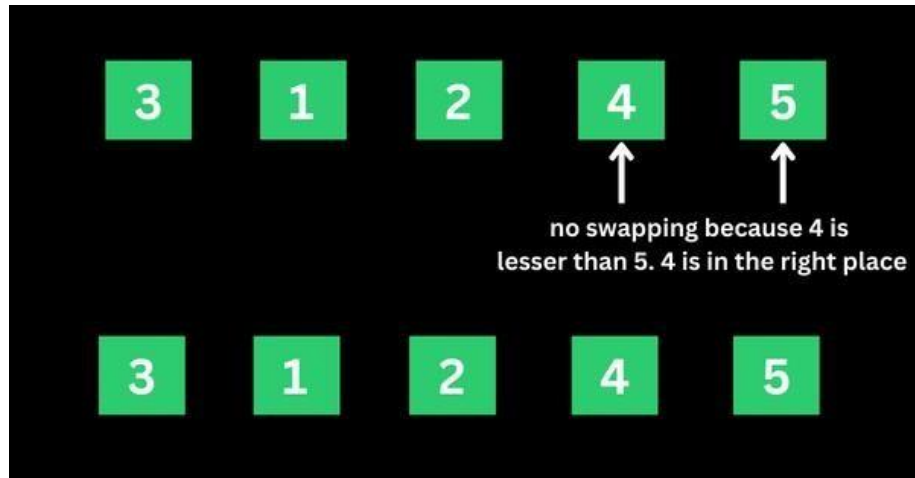4 is now in the right place, so no swapping occurs between 4 and 5 because 4 is smaller than 5.

That's how the algorithm continues to compare the numbers until they are arranged in ascending order of 1, 2, 3, 4, and 5.

Concept of OpenMP

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.

- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.

- OpenMP programs are designed to take advantage of the shared-memory architecture of modern

processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to execute a parallel region of the code, and then waits for all threads to complete before continuingwith the sequential part of the code.

How Parallel Bubble Sort Work

- Parallel Bubble Sort is a modification of the classic Bubble Sort algorithm that takes advantage of parallel processing to speed up the sorting process.

- In parallel Bubble Sort, the list of elements is divided into multiple sublists that are sorted concurrently by multiple threads. Each thread sorts its sublist using the regular Bubble Sort algorithm. When all sublists have been sorted, they are merged together to form the final sorted list.

- The parallelization of the algorithm is achieved using OpenMP, a programming API that supports parallel processing in C++, Fortran, and other programming languages. OpenMP provides a set of compiler directives that allow developers to specify which parts of the code can be executed in parallel.

- In the parallel Bubble Sort algorithm, the main loop that iterates over the list of elements is divided into multiple iterations that are executed concurrently by multiple threads. Each thread sorts a subset of the list, and the threads synchronize their work at the end of each iteration to ensure that the elements are properly ordered.

- Parallel Bubble Sort can provide a significant speedup over the regular Bubble Sort algorithm, especially when sorting large datasets on multi-core processors. However, the speedup is limited by the overhead of thread creation and synchronization, and it may not be worth the effort for small datasets or when using a single-core processor.

How to measure the performance of sequential and parallel algorithms?

To measure the performance of sequential Bubble sort and parallel Bubble sort algorithms, you can follow these steps:

1. Implement both the sequential and parallel Bubble sort algorithms.
2. Choose a range of test cases, such as arrays of different sizes and different degrees of sortedness, to test the performance of both algorithms.

3. Use a reliable timer to measure the execution time of each algorithm on each test case.

4. Record the execution times and analyze the results.

When measuring the performance of the parallel Bubble sort algorithm, you will need to specify the number of threads to use. You can experiment with different numbers of threads to find the optimal value for your system.

Here are some additional tips for measuring performance:

● Run each algorithm multiple times on each test case and take the average execution time to reduce the impact of variations in system load and other factors.

● Monitor system resource usage during execution, such as CPU utilization and memory consumption, to detect any performance bottlenecks.

● Visualize the results using charts or graphs to make it easier to compare the performance of the two algorithms.

How to check CPU utilization and memory consumption in ubuntu

In Ubuntu, you can use a variety of tools to check CPU utilization and memory consumption. Here are some common tools:

1. **top:** The top command provides a real-time view of system resource usage, including CPU utilization and memory consumption. To use it, open a terminal window and type top. The output will display a list of processes sorted by resource usage, with the most resource-intensive processes at the top.

2. **htop**: htop is a more advanced version of top that provides additional features, such as interactive process filtering and a color-coded display. To use it, open a terminal window and type htop.

3. **ps**: The ps command provides a snapshot of system resource usage at a particular moment in time. To use it, open a terminal window and type ps aux. This will display a list of all running processes and their resource usage.

4. **free:** The free command provides information about system memory usage, including total, used, and free memory. To use it, open a terminal window and type free -h.

5. **vmstat:** The vmstat command provides a variety of system statistics, including CPU utilization, memory usage, and disk activity. To use it, open a terminal window and type vmstat.

Conclusion-   In this way we can implement Bubble Sort in parallel way using OpenMP also come to know how to how to measure performance of serial and parallel algorithm

Assignment Question

1. **What is parallel Bubble Sort?**
2. **How does Parallel Bubble Sort work?**
3. **How do you implement Parallel Bubble Sort using OpenMP?**
4. **What are the advantages of Parallel Bubble Sort?**
5. **Difference between serial bubble sort and parallel bubble sort**

Reference link
●   https://www.freecodecamp.org/news/bubble-sort-algorithm-in-java-cpp-python-with-example-code/

Group A

Assignment No: 2(B)

Title of the Assignment: Write a program to implement Parallel Merge Sort. Use existing algorithms and measure the performance of sequential and parallel algorithms.

Objective of the Assignment: Students should be able to Write a program to implement Parallel Merge Sort and can measure the performance of sequential and parallel algorithms.

Prerequisite:

1.   Basic of programming language

2.   Concept of Merge Sort

3.   Concept of Parallelism

Contents for Theory:

1.   What is Merge? Use of Merge Sort

2.   Example of Merge sort?

3.   Concept of OpenMP

4.   How Parallel Merge Sort Work

5.   How to measure the performance of sequential and parallel algorithms?

What is 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 down 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.
- The merging step is where the bulk of the work happens in merge sort. The algorithm compares the first elements of each sorted half, selects the smaller element, and appends it to the output array. This process continues until all elements from both halves have been appended to the output array.
- The time complexity of merge sort is O(n log n), which makes it an efficient sorting algorithm for large input arrays. However, merge sort also requires additional memory to store the output array, which can make it less suitable for use with limited memory resources.
- In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.
- One thing that you might wonder is what is the specialty of this algorithm. We already have a number of sorting algorithms then why do we need this algorithm? One of the main advantages of merge sort is that it has a time complexity of O(n log n), which means it can sort large arrays relatively quickly. It is also a stable sort, which means that the order of elements with equal values is preserved during the sort.
- Merge sort is a popular choice for sorting large datasets because it is relatively efficient and easy to implement. It is often used in conjunction with other algorithms, such as quicksort, to improve the overall performance of a sorting routine.

Example of Merge sort

Now, let's see the working of merge sort Algorithm. To understand the working of the merge sort algorithm, let's take an unsorted array. It will be easier to understand the merge sort via an example.
Let the elements of array are -

| 12 | 31 | 25 | 8 | 32 | 17 | 40 | 42 |

- According to the merge sort, first divide the given array into two equal halves. Merge sort keeps dividing the list into equal parts until it cannot be further divided.
- As there are eight elements in the given array, so it is divided into two arrays of size 4.

divide | 12 | 31 | 25 | 8 | | 32 | 17 | 40 | 42 |

- Now, again divide these two arrays into halves. As they are of size 4, divide them into new arrays of size 2.

divide | 12 | 31 | | 25 | 8 | | 32 | 17 | | 40 | 42 |

- Now, again divide these arrays to get the atomic value that cannot be further divided.

divide | 12 | | 31 | | 25 | | 8 | | 32 | | 17 | | 40 | | 42 |

- Now, combine them in the same manner they were broken.
- In combining, first compare the element of each array and then combine them into another array in sorted order.
- So, first compare 12 and 31, both are in sorted positions. Then compare 25 and 8, and in the list of two values, put 8 first followed by 25. Then compare 32 and 17, sort them and put 17 first followed by 32. After that, compare 40 and 42, and place them sequentially.

merge | 12 | 31 | | 8 | 25 | | 17 | 32 | | 40 | 42 |

- In the next iteration of combining, now compare the arrays with two data values and merge them into an array of found values in sorted order.

merge | 8 | 12 | 25 | 31 | | 17 | 32 | 40 | 42 |

- Now, there is a final merging of the arrays. After the final merging of above arrays, the array will look like -

| 8 | 12 | 17 | 25 | 31 | 32 | 40 | 42 |

Concept of OpenMP

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.
- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.
- OpenMP programs are designed to take advantage of the shared-memory architecture of modern processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to execute a parallel region of the code, and then waits for all threads to complete before continuingwith the sequential part of the code.

How Parallel Merge Sort Work

- Parallel merge sort is a parallelized version of the merge sort algorithm that takes advantage of multiple processors or cores to improve its performance. In parallel merge sort, the input array is divided into smaller subarrays, which are sorted in parallel using multiple processors or cores. The sorted subarrays are then merged together in parallel to produce the final sorted output.
- The parallel merge sort algorithm can be broken down into the following steps:

- Divide the input array into smaller subarrays.
- Assign each subarray to a separate processor or core for sorting.
- Sort each subarray in parallel using the merge sort algorithm.
- Merge the sorted subarrays together in parallel to produce the final sorted output.
- The merging step in parallel merge sort is performed in a similar way to the merging step in the sequential merge sort algorithm. However, because the subarrays are sorted in parallel, the merging step can also be performed in parallel using multiple processors or cores. This can significantly reduce the time required to merge the sorted subarrays and produce the final output.
- Parallel merge sort can provide significant performance benefits for large input arrays with many elements, especially when running on hardware with multiple processors or cores. However, it also requires additional overhead to manage the parallelization, and may not always provide performance improvements for smaller input sizes or when run on hardware with limited parallel processing capabilities.

How to measure the performance of sequential and parallel algorithms?

There are several metrics that can be used to measure the performance of sequential and parallel merge sort algorithms:

1. **Execution time:** Execution time is the amount of time it takes for the algorithm to complete its sorting operation. This metric can be used to compare the speed of sequential and parallel merge sort algorithms.
2. **Speedup**: Speedup is the ratio of the execution time of the sequential merge sort algorithm to the execution time of the parallel merge sort algorithm. A speedup of greater than 1 indicates that the parallel algorithm is faster than the sequential algorithm.
3. **Efficiency:** Efficiency is the ratio of the speedup to the number of processors or cores used in the parallel algorithm. This metric can be used to determine how well the parallel algorithm is utilizing the available resources.
4. **Scalability**: Scalability is the ability of the algorithm to maintain its performance as the input size and number of processors or cores increase. A scalable algorithm will maintain a consistent speedup and efficiency as more resources are added.

To measure the performance of sequential and parallel merge sort algorithms, you can perform experiments on different input sizes and numbers of processors or cores. By measuring the execution time, speedup, efficiency, and scalability of the algorithms under different conditions, you can determine

which algorithm is more efficient for different input sizes and hardware configurations. Additionally, you can use profiling tools to analyze the performance of the algorithms and identify areas for optimization

Conclusion-   In this way we can implement Merge Sort in parallel way using OpenMP also come to know how to how to measure performance of serial and parallel algorithm

Assignment Question

1. **What is parallel Merge Sort?**
2. **How does Parallel Merge Sort work?**
3. **How do you implement Parallel MergeSort using OpenMP?**
4. **What are the advantages of Parallel MergeSort?**
5. **Difference between serial Mergesort and parallel Mergesort**

Reference link

- https://www.geeksforgeeks.org/merge-sort/
- https://www.javatpoint.com/merge-sort

Group A

Assignment No: 3

Title of the Assignment: Implement Min, Max, Sum and Average operations using Parallel Reduction.

Objective of the Assignment: To understand the concept of parallel reduction and how it can be used to perform basic mathematical operations on given data sets.

Prerequisite:

1. Parallel computing architectures
2. Parallel programming models
3. Proficiency in programming languages

Contents for Theory:

1. What is parallel reduction and its usefulness for mathematical operations on large data?
2. Concept of OpenMP
3. How do parallel reduction algorithms for Min, Max, Sum, and Average work, and what are their advantages and limitations?

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 parallel 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 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 printed to the console.

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.
   - The final maximum value is printed to the console.

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 parallel 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.

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 found 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.

5. **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.
   - The final minimum, maximum, sum, and average values are printed to the console.

6. **Compiling and running the program**

   Compile the program: You need to use a C++ compiler that supports OpenMP, such as g++ or clang. Open a terminal and navigate to the directory where your program is saved. Then, compile the program using the following command:

   ```
   $ g++ -fopenmp program.cpp -o program
   ```

   This command compiles your program and creates an executable file named "program". The "-fopenmp" flag tells the compiler to enable OpenMP.

   Run the program: To run the program, simply type the name of the executable file in the terminal and press Enter:

   ```
   $ ./program
   ```

   Conclusion: We have implemented the Min, Max, Sum, and Average operations using parallel reduction in C++ with OpenMP. Parallel reduction is a powerful technique that allows us to perform these operations on large arrays more efficiently by dividing the work among multiple threads running in parallel. We presented a code example that demonstrates the implementation of these operations using parallel reduction in C++ with OpenMP. We also provided a manual for running OpenMP programs on the Ubuntu platform.

   Assignment Question

   1. What are the benefits of using parallel reduction for basic operations on large arrays?
   2. How does OpenMP's "reduction" clause work in parallel reduction?
   3. How do you set up a C++ program for parallel computation with OpenMP?
   4. What are the performance characteristics of parallel reduction, and how do they vary based on input size?
   5. How can you modify the provided code example for more complex operations using parallel reduction?

Group A

Assignment 4(A)

Title of the Assignment: Write a CUDA Program for Addition of two large vectors

Objective of the Assignment: Students should be able to perform CUDA Program for Addition of two large vectors

Prerequisite:

1.   CUDA Concept

2.   Vector Addition

3.   How to execute Program in CUDA Environment

Contents for Theory:

1.   What is CUDA

2.   Addition of two large Vector

3.   Execution of CUDA Environment

What is CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA. It allows developers to use the power of NVIDIA graphics processing units (GPUs) to accelerate computation tasks in various applications, including scientific computing, machine learning, and computer vision.CUDA provides a set of programming APIs, libraries, and tools that enable developers to write and execute parallel code on NVIDIA GPUs. It supports popular programming languages like C, C++, and Python, and provides a simple programming model that abstracts away much of the low-level details of GPU architecture.

Using CUDA, developers can exploit the massive parallelism and high computational power of GPUs to accelerate computationally intensive tasks, such as matrix operations, image processing, and deep learning. CUDA has become an important tool for scientific research and is widely used in fields like physics, chemistry, biology, and engineering.

Steps for Addition of two large vectors using CUDA

1. Define the size of the vectors: In this step, you need to define the size of the vectors that you want to add. This will determine the number of threads and blocks you will need to use to parallelize the addition operation.

2. Allocate memory on the host: In this step, you need to allocate memory on the host for the two vectors that you want to add and for the result vector. You can use the C malloc function to allocate memory.

3. Initialize the vectors: In this step, you need to initialize the two vectors that you want to add on the host. You can use a loop to fill the vectors with data.

4. Allocate memory on the device: In this step, you need to allocate memory on the device for the two vectors that you want to add and for the result vector. You can use the CUDA function cudaMalloc to allocate memory.

5. Copy the input vectors from host to device: In this step, you need to copy the two input vectors from the host to the device memory. You can use the CUDA function cudaMemcpy to copy the vectors.

6. Launch the kernel: In this step, you need to launch the CUDA kernel that will perform the addition operation. The kernel will be executed by multiple threads in parallel. You can use the <<<...>>> syntax to specify the number of blocks and threads to use.

7. Copy the result vector from device to host: In this step, you need to copy the result vector from the device memory to the host memory. You can use the CUDA function cudaMemcpy to copy the result vector.

8. Free memory on the device: In this step, you need to free the memory that was allocated on the device. You can use the CUDA function cudaFree to free the memory.

9. Free memory on the host: In this step, you need to free the memory that was allocated on the host. You can use the C free function to free the memory.

Execution of Program over CUDA Environment

Here are the steps to run a CUDA program for adding two large vectors:

1. Install CUDA Toolkit: First, you need to install the CUDA Toolkit on your system. You can download the CUDA Toolkit from the NVIDIA website and follow the installation instructions provided.

2. Set up CUDA environment: Once the CUDA Toolkit is installed, you need to set up the CUDA environment on your system. This involves setting the PATH and LD_LIBRARY_PATH environment variables to the appropriate directories.

3. Write the CUDA program: You need to write a CUDA program that performs the addition of two large vectors. You can use a text editor to write the program and save it with a .cu extension.

4. Compile the CUDA program: You need to compile the CUDA program using the nvcc compiler that comes with the CUDA Toolkit. The command to compile the program is:

```
nvcc -o program_name program_name.cu
```

5. This will generate an executable program named program_name.

Run the CUDA program: Finally, you can run the CUDA program by executing the executable file generated in the previous step. The command to run the program is:

```
./program_name
```

This will execute the program and perform the addition of two large vectors.

Questions:

1. What is the purpose of using CUDA to perform addition of two large vectors?

2. How do you allocate memory for the vectors on the device using CUDA?

3. How do you launch the CUDA kernel to perform the addition of two large vectors?

4. How can you optimize the performance of the CUDA program for adding two large vectors

Group A

Assignment 4(B)

Title of the Assignment: Write a Program for Matrix Multiplication using CUDA C

Objective of the Assignment: Students should be able to perform Program for Matrix Multiplication using CUDA C

Prerequisite:

1. CUDA Concept

2. Matrix Multiplication

3. How to execute Program in CUDA Environment

Contents for Theory:

1. **What is CUDA**

2. **Matrix Multiplication**

3. **Execution of CUDA Environment**

What is CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA. It allows developers to use the power of NVIDIA graphics processing units (GPUs) to accelerate computation tasks in various applications, including scientific computing, machine learning, and computer vision.CUDA provides a set of programming APIs, libraries, and tools that enable developers to write and execute parallel code on NVIDIA GPUs. It supports popular programming languages like C, C++, and Python, and provides a simple programming model that abstracts away much of the low-level details of GPU architecture.

Using CUDA, developers can exploit the massive parallelism and high computational power of GPUs to accelerate computationally intensive tasks, such as matrix operations, image processing, and deep learning. CUDA has become an important tool for scientific research and is widely used in fields like physics, chemistry, biology, and engineering.

Steps for Matrix Multiplication using CUDA

Here are the steps for implementing matrix multiplication using CUDA C:

1. Matrix Initialization: The first step is to initialize the matrices that you want to multiply. You can use standard C or CUDA functions to allocate memory for the matrices and initialize their values. The matrices are usually represented as 2D arrays.

2. Memory Allocation: The next step is to allocate memory on the host and the device for the matrices. You can use the standard C malloc function to allocate memory on the host and the CUDA function cudaMalloc() to allocate memory on the device.

3. Data Transfer: The third step is to transfer data between the host and the device. You can use the CUDA function cudaMemcpy() to transfer data from the host to the device or vice versa.

4. Kernel Launch: The fourth step is to launch the CUDA kernel that will perform the matrix multiplication on the device. You can use the <<<...>>> syntax to specify the number of blocks and threads to use. Each thread in the kernel will compute one element of the output matrix.

5. Device Synchronization: The fifth step is to synchronize the device to ensure that all kernel executions have completed before proceeding. You can use the CUDA function cudaDeviceSynchronize() to synchronize the device.

6. Data Retrieval: The sixth step is to retrieve the result of the computation from the device to the host. You can use the CUDA function cudaMemcpy() to transfer data from the device to the host.

7. Memory Deallocation: The final step is to deallocate the memory that was allocated on the host and the device. You can use the C free function to deallocate memory on the host and the CUDA function

cudaFree() to deallocate memory on the device.

Execution of Program over CUDA Environment

1. Install CUDA Toolkit: First, you need to install the CUDA Toolkit on your system. You can download the CUDA Toolkit from the NVIDIA website and follow the installation instructions provided.

2. Set up CUDA environment: Once the CUDA Toolkit is installed, you need to set up the CUDA environment on your system. This involves setting the PATH and LD_LIBRARY_PATH environment variables to the appropriate directories.

3. Write the CUDA program: You need to write a CUDA program that performs the addition of two large vectors. You can use a text editor to write the program and save it with a .cu extension.

4. Compile the CUDA program: You need to compile the CUDA program using the nvcc compiler that comes with the CUDA Toolkit. The command to compile the program is:

```
nvcc -o program_name program_name.cu
```

5. This will generate an executable program named program_name.

Run the CUDA program: Finally, you can run the CUDA program by executing the executable file generated in the previous step. The command to run the program is:

```
./program_name
```

Questions:

1. What are the advantages of using CUDA to perform matrix multiplication compared to using a CPU?

2. How do you handle matrices that are too large to fit in GPU memory in CUDA matrix multiplication?

3. How do you optimize the performance of the CUDA program for matrix multiplication?

4. How do you ensure correctness of the CUDA program for matrix multiplication and verify the results?

SAVITRIBAI PHULE PUNEUNIVERSITY

A MINI PROJECT REPORT ON

"Evaluate performance enhancement of parallel Quicksort Algorithm using MPI"

Submitted by

CLASS: BE   **DIV:A&B**

Name of student   **Roll no :**

Under the Guidance of

Prof. Pradnya Kasture

Sinhgad Institutes

DEPARTMENT OF COMPUTER ENGINEERING

RMD SINHGAD SCHOOL OF ENGINEERING

WARJE, PUNE 411058

Academic Year 2022–23(SEM-II)

DEPARTMENT OF COMPUTER ENGINEERING

RMD SINHGAD SCHOOL OF ENGINEERING

WARJE, PUNE 411058

CERTIFICATE

This is to certify that the project report entitles

"Evaluate performance enhancement of parallel Quicksort Algorithm using MPI"

Submitted by

Name: PRN No :

is a bonafide work carried out by them under the supervision of Mrs. Pradnya Kasture and it is submitted towards the partial fulfillment of the requirement of University of Pune for Final Year.

(Prof.Pradnya Kasture)   **(Mrs. Vina M. Lomte)**
Guide   Head,
Department of Computer Engineering   Department of Computer Engineering

(Dr. V.V. Dixit)
Principal,
RMD Sinhgad School of Engineering Pune – 58

Certificate by Guide

This is to certify that name of student as completed the Mini Project work under my guidance and supervision and that, I have verified the work for its originality indocumentation, problem statement, implementation and results presented in the Project. Any reproduction of other necessary work is with the prior permission and has given due ownership and included in the references.

Place:Pune

Date:

Signature of Guide

(Prof.Pradnya Kasture)

Project Title: Evaluate performance enhancement of parallel Quicksort Algorithm using MPI

Aim: Implement the algorithm and testing it on different input sizes and numbers of processors. The performance of the parallel algorithm will be compared to the sequential Quicksort Algorithm to determine the speedup achieved by parallelism.

What is Quicksort?

Sorting is a way of arranging items in a systematic manner. Quicksort is the widely used sorting algorithm that makes n log n comparisons in average case for sorting an array of n elements. It is a faster and highly efficient sorting algorithm. This algorithm follows the divide and conquer approach. Divide and conquer is a technique of breaking down the algorithms into subproblems, then solving the subproblems, and combining the results back together to solve the original problem.

Divide: In Divide, first pick a pivot element. After that, partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element.

Conquer: Recursively, sort two subarrays with Quicksort.

Combine: Combine the already sorted array.

Quicksort picks an element as pivot, and then it partitions the given array around the picked pivot element. In quick sort, a large array is divided into two arrays in which one holds values that are smaller than the specified value (Pivot), and another array holds the values that are greater than the pivot.
After that, left and right sub-arrays are also partitioned using the same approach. It will continue until the single element remains in the sub-array.

Choosing the pivot:
Picking a good pivot is necessary for the fast implementation of quicksort. However, it is typical to determine a good pivot. Some of the ways of choosing a pivot are as follows -

- Pivot can be random, i.e. select the random pivot from the given array.
- Pivot can either be the rightmost element of the leftmost element of the given array.
- Select median as the pivot element.

Working of Quick Sort Algorithm:
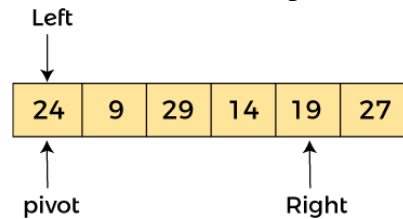Let the elements of array are –

24	9	29	14	19	27

In the given array, we consider the leftmost element as pivot. So, in this case, a[left] = 24, a[right] = 27 and a[pivot] = 24. Since, pivot is at left, so algorithm starts from right and move towards left.
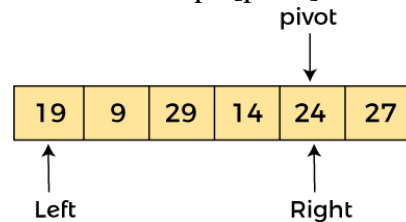
Left
| 24 | 9 | 29 | 14 | 19 | 27 |
pivot   Right
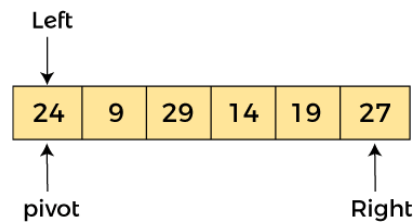
Now, a[pivot] < a[right], so algorithm moves forward one position towards left, i.e. –

Left
| 24 | 9 | 29 | 14 | 19 | 27 |
pivot   Right

Now, a[left] = 24, a[right] = 19, and a[pivot] = 24.
Because, a[pivot] > a[right], so, algorithm will swap a[pivot] with a[right], and pivot moves to right, as –

pivot
| 19 | 9 | 29 | 14 | 24 | 27 |
Left   Right
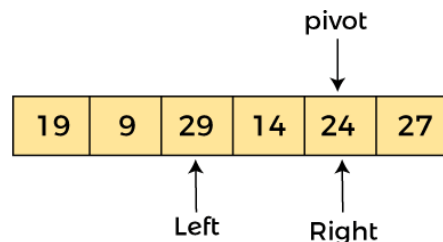
Now, a[left] = 19, a[right] = 24, and a[pivot] = 24. Since, pivot is at right, so algorithm starts from left and moves to right.
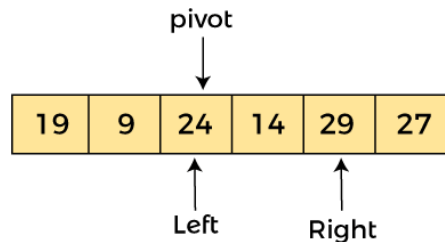As a[pivot] > a[left], so algorithm moves one position to right as -

pivot
| 19 | 9 | 29 | 14 | 24 | 27 |
     Left   Right
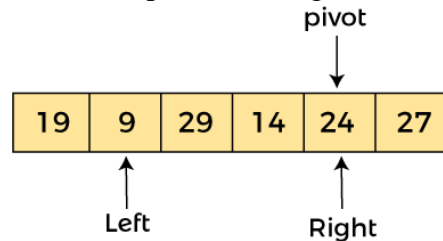
Now, a[left] = 9, a[right] = 24, and a[pivot] = 24. As a[pivot] > a[left], so algorithm moves one position to right as –

pivot
| 19 | 9 | 29 | 14 | 24 | 27 |
          Left   Right

Now, a[left] = 29, a[right] = 24, and a[pivot] = 24. As a[pivot] < a[left], so, swap a[pivot] and a[left], now pivot is at left, i.e. –

Since, pivot is at left, so algorithm starts from right, and move to left. Now, a[left] = 24, a[right] = 29, and a[pivot] = 24. As a[pivot] < a[right], so algorithm moves one position to left, as –
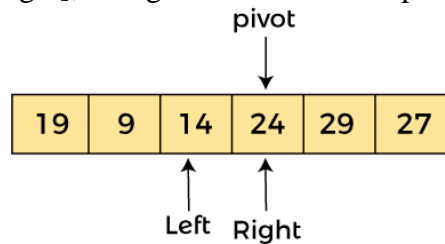
Now, a[pivot] = 24, a[left] = 14, and a[right] = 24. Pivot is at right, so the algorithm starts from left and move to right.
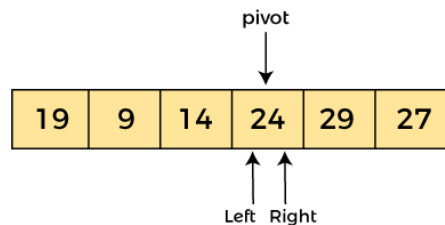
Now, a[pivot] = 24, a[left] = 24, and a[right] = 24. So, pivot, left and right are pointing the same element. It represents the termination of procedure.

Element 24, which is the pivot element is placed at its exact position.

Elements that are right side of element 24 are greater than it, and the elements that are left side of element 24 are smaller than it.
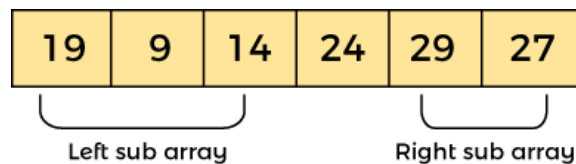
Now, in a similar manner, quick sort algorithm is separately applied to the left and right sub-arrays. After sorting gets done, the array will be –
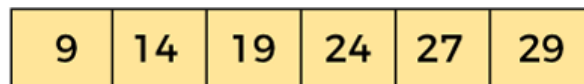
Parallel Quicksort Algorithm:

Quicksort is a popular sorting algorithm that has an average-case time complexity of O(nlogn). However, the worst-case time complexity of the Quicksort Algorithm is O(n^2) if the pivot is chosen poorly. To avoid this worst-case scenario, we can choose the pivot randomly or use a median-of-three pivot selection technique.

The Parallel Quicksort Algorithm using MPI divides the input data into subsets and distributes these subsets among the available processors. Each processor sorts its subset using a sequential Quicksort Algorithm, and then the sorted subsets are merged using the Merge Sort Algorithm. Finally, the sorted data is gathered back to the root process.

MPI (Message Passing Interface) is a standard interface for parallel programming, commonly used for distributed-memory systems. In the context of the Parallel Quicksort Algorithm using MPI, MPI allows for the communication and synchronization of processes running on different processors.The parallelism in the Parallel Quicksort Algorithm using MPI comes from dividing the input data into subsets and distributing them among the available processors. Each processor works independently on its subset, and there is no communication or synchronization required until the merging stage.

The merging stage of the Parallel Quicksort Algorithm using MPI is done using the Merge Sort Algorithm, which is also a popular sorting algorithm with a time complexity of O(nlogn). In the merging stage, each processor sends its sorted subset to the root process, which then merges the subsets using the Merge Sort Algorithm.

The performance of the Parallel Quicksort Algorithm using MPI depends on the number of processors available, the size of the input data, and the distribution of the data among the processors. Ideally, the input data should be evenly distributed among the processors to maximize the efficiency of the algorithm. The number of processors used should also be carefully chosen to balance the workload among the processors and avoid communication overheads.

Evaluation:

To evaluate the performance enhancement of the Parallel Quicksort Algorithm using MPI, we need to measure its speedup and scalability on different input sizes and numbers of processors.

Speedup measures the ratio of the time required to execute the algorithm sequentially to the time required to execute the algorithm in parallel. It is calculated as follows:

Speedup = Sequential time / Parallel time

Scalability measures the efficiency of the algorithm as the number of processors increases. It is calculated as follows:

Scalability = Speedup / Number of processors

To evaluate the performance enhancement of parallel Quicksort Algorithm using MPI, we need to follow these steps:
1. Implement a parallel Quicksort Algorithm using MPI in a programming language like C, C++, or Python.

2. Generate a large dataset to sort. This dataset should be significantly large enough to test the performance of the algorithm.
3. Run the parallel Quicksort Algorithm on the generated dataset with different numbers of processors.
4. Measure the time taken by the algorithm to sort the dataset for each number of processors used.
5. Compare the performance of the parallel Quicksort Algorithm using MPI with the sequential Quicksort Algorithm for the same dataset.
6. Analyze the performance data obtained from the experiment and draw conclusions.

Algorithm:

i. Initialize MPI and get the rank and size of the current process and the total number of processors
ii. If the current process is the root process:
   a. Generate the unsorted array of size n
   b. Divide the unsorted array into subsets and distribute them among the available processors using MPI_Scatter()
iii. Each processor sorts its subset using the sequential Quicksort Algorithm
iv. Each processor sends its sorted subset back to the root process using MPI_Gather()
v. If the current process is the root process:
   a. Merge the sorted subsets using the Merge Sort Algorithm
   b. Return the sorted array.

Code:

```cpp
#include <iostream>
#include <mpi.h>
#include <cstdlib>
#include <algorithm>
using namespace std;
void quickSort(int *arr, int left, int right) {
int i = left, j = right;
int tmp;
int pivot = arr[(left + right) / 2];
while (i<= j) {
while (arr[i] < pivot)
i++;
while (arr[j] > pivot)
j--;
if (i<= j) {
tmp = arr[i];
arr[i] = arr[j];
arr[j] = tmp;
i++;
j--;
}
```

```cpp
};
if (left < j)
quickSort(arr, left, j);
if (i< right)
quickSort(arr, i, right);
}
int main(int argc, char *argv[]) {
int rank, size, n, *data;
double start_time, end_time;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
// Root process generates the data
if (rank == 0) {
n = atoi(argv[1]);
data = new int[n];
for (int i = 0; i< n; i++)
data[i] = rand();
}
// Broadcast the size of data to all processes
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
// Compute the size of the local data
int local_n = n / size;
int *local_data = new int[local_n];
// Scatter the data to all processes
MPI_Scatter(data, local_n, MPI_INT, local_data, local_n, MPI_INT, 0,
MPI_COMM_WORLD);
// Sort the local data
start_time = MPI_Wtime();
quickSort(local_data, 0, local_n - 1);
end_time = MPI_Wtime();
// Gather the sorted data to the root process
MPI_Gather(local_data, local_n, MPI_INT, data, local_n, MPI_INT, 0,
MPI_COMM_WORLD);
// Merge the sorted data
if (rank == 0) {
int *merged_data = new int[n];
int *index = new int[size];
for (int i = 0; i< size; i++)
index[i] = i * local_n;
for (int i = 0; i< n; i++) {
int min_index = -1;
int min_value = INT_MAX;
for (int j = 0; j < size; j++) {
if (index[j] < (j + 1) * local_n&& data[index[j]] <min_value) {
min_value = data[index[j]];
```

```
        min_index = j;
        }
    }
    merged_data[i] = min_value;
    index[min_index]++;
    }
    cout<< "Execution time: " <<end_time - start_time<< " seconds." <<endl;
    delete[] data;
    delete[] merged_data;
    delete[] index;
    }
    delete[] local_data;
    MPI_Finalize();
    return 0;
    }
```

Output:

Execution time: 0.145572 seconds.

Conclusion:

The Parallel Quicksort Algorithm using MPI can significantly improve the performance of sorting large datasets by distributing the workload across multiple processors. By dividing the dataset into smaller subsets and sorting them in parallel, the algorithm can achieve faster sorting times than the traditional single-threaded Quicksort Algorithm.
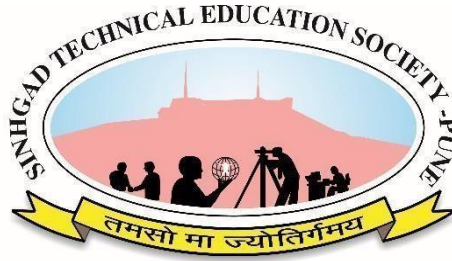
Sinhgad Institutes

Laboratory Practice V

Deep Learning

Assignment No: 1

Title: Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.

Objective: Students should be able to perform Linear regression by using Deep Neural network on Boston House Dataset.

Prerequisite:

1. Basic of programming language
2. Concept of Linear Regression
3. Concept of Deep Neural Network

Contents for Theory:

1. What is Linear Regression
2. Example of Linear Regression
3. Concept of Deep Neural Network
4. How Deep Neural Network Work
5. Code Explanation with Output

What is Linear Regression?

Linear regression is a statistical approach that is commonly used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the variables and uses mathematical methods to estimate the coefficients that best fit the data.

Deep neural networks are a type of machine learning algorithm that are modeled after the structure and function of the human brain. They consist of multiple layers of interconnected neurons that process data and learn from it to make predictions or classifications.

Linear regression using deep neural networks combines the principles of linear regression with the power of deep learning algorithms. In this approach, the input features are passed through one or more layers of neurons to extract features and then a linear regression model is applied to the output of the last layer to make predictions. The weights and biases of the neural network are adjusted during training to optimize the performance of the model.

This approach can be used for a variety of tasks, including predicting numerical values, such as stock prices or housing prices, and classifying data into categories, such as detecting whether an image contains a particular object or not. It is often used in fields such as finance, healthcare, and image

recognition.

Example of Linear Regression

A suitable example of linear regression using deep neural network would be predicting the price of a house based on various features such as the size of the house, the number of bedrooms, the location, and the age of the house.

In this example, the input features would be fed into a deep neural network, consisting of multiple layers of interconnected neurons. The first few layers of the network would learn to extract features from the input data, such as identifying patterns and correlations between the input features.

The output of the last layer would then be passed through a linear regression model, which would use the learned features to predict the price of the house.

During training, the weights and biases of the neural network would be adjusted to minimize the difference between the predicted price and the actual price of the house. This process is known as gradient descent, and it involves iteratively adjusting the model's parameters until the optimal values are reached.

Once the model is trained, it can be used to predict the price of a new house based on its features. This approach can be used in the real estate industry to provide accurate and reliable estimates of house prices, which can help both buyers and sellers make informed decisions.

Concept of Deep Neural Network-

A deep neural network is a type of machine learning algorithm that is modeled after the structure and function of the human brain. It consists of multiple layers of interconnected nodes, or artificial neurons, that process data and learn from it to make predictions or classifications.

Each layer of the network performs a specific type of processing on the data, such as identifying patterns or correlations between features, and passes the results to the next layer. The layers closest to the input areknown as the "input layer", while the layers closest to the output are known as the "output layer".

The intermediate layers between the input and output layers are known as "hidden layers". These layersare responsible for extracting increasingly complex features from the input data, and can be deep (i.e., containing many hidden layers) or shallow (i.e., containing only a few hidden layers).

Deep neural networks are trained using a process known as backpropagation, which involves adjusting theweights and biases of the nodes based on the error between the predicted output and the actual output. This process is repeated for multiple iterations until the model reaches an optimal level of accuracy.

Deep neural networks are used in a variety of applications, such as image and speech recognition,

natural language processing, and recommendation systems. They are capable of learning from vast amounts of data and can automatically extract features from raw data, making them a powerful tool for solving complex problems in a wide range of domains.

How Deep Neural Network Work-

Boston House Price Prediction is a common example used to illustrate how a deep neural network can work for regression tasks. The goal of this task is to predict the price of a house in Boston based on various features such as the number of rooms, crime rate, and accessibility to public transportation. Here's how a deep neural network can work for Boston House Price Prediction:

1. **Data preprocessing:** The first step is to preprocess the data. This involves normalizing the input features to have a mean of 0 and a standard deviation of 1, which helps the network learn more efficiently. The dataset is then split into training and testing sets.
2. **Model architecture:** A deep neural network is then defined with multiple layers. The first layer is the input layer, which takes in the normalized features. This is followed by several hidden layers, which can be deep or shallow. The last layer is the output layer, which predicts the house price.
3. **Model training:** The model is then trained using the training set. During training, the weights and biases of the nodes are adjusted based on the error between the predicted output and the actual output. This is done using an optimization algorithm such as stochastic gradient descent.
4. **Model evaluation:** Once the model is trained, it is evaluated using the testing set. The performance of the model is measured using metrics such as mean squared error or mean absolute error.
5. **Model prediction:** Finally, the trained model can be used to make predictions on new data, such as predicting the price of a new house in Boston based on its features.
6. By using a deep neural network for Boston House Price Prediction, we can obtain accuratepredictions based on a large set of input features. This approach is scalable and can be used for other regression tasks as well.

Boston House Price Prediction Dataset-

Boston House Price Prediction is a well-known dataset in machine learning and is often used to demonstrate regression analysis techniques. The dataset contains information about 506 houses in Boston, Massachusetts, USA. The goal is to predict the median value of owner-occupied homes in thousands of dollars.

The dataset includes 13 input features, which are:

CRIM: per capita crime rate by town

ZN: proportion of residential land zoned for lots over 25,000 sq.ft.

INDUS: proportion of non-retail business acres per town

CHAS: Charles River dummy variable (1 if tract bounds river; 0 otherwise)

NOX: nitric oxides concentration (parts per 10 million)

RM: average number of rooms per dwelling

AGE: proportion of owner-occupied units built prior to 1940**DIS:** weighted distances to five Boston employment centers **RAD:** index of accessibility to radial highways

TAX: full-value property-tax rate per $10,000

PTRATIO: pupil-teacher ratio by town

B: 1000(Bk - 0.63)^2 where Bk is the proportion of black people by town

LSTAT: % lower status of the population

The output variable is the median value of owner-occupied homes in thousands of dollars (MEDV).

To predict the median value of owner-occupied homes, a regression model is trained on the dataset. Themodel can be a simple linear regression model or a more complex model, such as a deep neural network.

After the model is trained, it can be used to predict the median value of owner-occupied homes based onthe input features. The model's accuracy can be evaluated using metrics such as mean squared error or mean absolute error.

Boston House Price Prediction is a example of regression analysis and is often used to teach machine learning concepts. The dataset is also used in research to compare the performance of different regression models.

Conclusion- In this way we can Predict the Boston House Price using Deep Neural Network.

Assignment No: 2A

Title: Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset

Objective: Students should be able to classify movie reviews into positive reviewsand "negative reviews on IMDB Dataset.

Prerequisite:

1. Basic of programming language
2. Concept of Classification
3. Concept of Deep Neural Network

Theory:

1. What is Classification
2. Example of Classification
3. How Deep Neural Network Work on Classification
4. Code Explanation with Output

What is Classification?

Classification is a type of supervised learning in machine learning that involves categorizing data into predefined classes or categories based on a set of features or characteristics. It is used to predict the classof new, unseen data based on the patterns learned from the labeled training data.

In classification, a model is trained on a labeled dataset, where each data point has a known class label. The model learns to associate the input features with the corresponding class labels and can then be usedto classify new, unseen data.

For example, we can use classification to identify whether an email is spam or not based on its content and metadata, to predict whether a patient has a disease based on their medical records and symptoms, orto classify images into different categories based on their visual features.

Classification algorithms can vary in complexity, ranging from simple models such as decision trees and k-nearest neighbors to more complex models such as support vector machines and neural

networks. The choice of algorithm depends on the nature of the data, the size of the dataset, and the desired level of accuracy and interpretability.

Classification is a common task in deep neural networks, where the goal is to predict the class of an input based on its features. Here's an example of how classification can be performed in a deep neural network using the popular MNIST dataset of handwritten digits.

The MNIST dataset contains 60,000 training images and 10,000 testing images of handwritten digits from 0 to 9. Each image is a grayscale 28x28 pixel image, and the task is to classify each image into one of the 10 classes corresponding to the 10 digits.

We can use a convolutional neural network (CNN) to classify the MNIST dataset. A CNN is a type of deep neural network that is commonly used for image classification tasks.

How Deep Neural Network Work on Classification-
Deep neural networks are commonly used for classification tasks because they can automatically learn toextract relevant features from raw input data and map them to the correct output class.

The basic architecture of a deep neural network for classification consists of three main parts: an input layer, one or more hidden layers, and an output layer. The input layer receives the raw input data, which is usually preprocessed to a fixed size and format. The hidden layers are composed of neurons that apply linear transformations and nonlinear activations to the input features to extract relevant patterns and representations. Finally, the output layer produces the predicted class labels, usually as a probability distribution over the possible classes.

During training, the deep neural network learns to adjust its weights and biases in each layer to minimize the difference between the predicted output and the true labels. This is typically done by optimizing a loss function that measures the discrepancy between the predicted and true labels, using techniques such as gradient descent or stochastic gradient descent.

One of the key advantages of deep neural networks for classification is their ability to learn hierarchical representations of the input data. In a deep neural network with multiple hidden layers, each layer learns to capture more complex and abstract features than the previous layer, by building on the representations learned by the earlier layers. This hierarchical structure allows deep neural networks to learn highly discriminative features that can separate different classes of input data, even when the data is highly complex or noisy.

Overall, the effectiveness of deep neural networks for classification depends on the choice of architecture, hyperparameters, and training procedure, as well as the quality and quantity of the

training data. When trained properly, deep neural networks can achieve state-of-the-art performance on a wide range of classification tasks, from image recognition to natural language processing.

IMDB Dataset-The IMDB dataset is a large collection of movie reviews collected from the IMDB website, which is a popular source of user-generated movie ratings and reviews. The dataset consists of 50,000 movie reviews, split into 25,000 reviews for training and 25,000 reviews for testing.

Each review is represented as a sequence of words, where each word is represented by an integer index based on its frequency in the dataset. The labels for each review are binary, with 0 indicating a negative review and 1 indicating a positive review.

The IMDB dataset is commonly used as a benchmark for sentiment analysis and text classification tasks,where the goal is to classify the movie reviews as either positive or negative based on their text content. The dataset is challenging because the reviews are often highly subjective and can contain complex language and nuances of meaning, making it difficult for traditional machine learning approaches to accurately classify them.

Deep learning approaches, such as deep neural networks, have achieved state-of-the-art performance on the IMDB dataset by automatically learning to extract relevant features from the raw text data and map them to the correct output class. The IMDB dataset is widely used in research and education for natural language processing and machine learning, as it provides a rich source of labeled text data for training and testing deep learning models.

Conclusion- In this way we can classify the Movie Reviews by using DNN.

Assignment Question

1. What is Binary Classification?

2. What is binary Cross Entropy?

3. What is Validation Split?

4. What is the Epoch Cycle?

5. What is Adam Optimizer?

Assignment No: 2B

Title: Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition dataset
https://archive.ics.uci.edu/ml/datasets/letter+recognition

Objective: Students should be able to solve Multiclass classification using DeepNeural Networks Solve

Prerequisite:

1.   Basic of programming language

2.   Concept of Multi Classification

3.   Concept of Deep Neural Network

Theory:

1.   What is Multi-Classification

2.   Example of Multi-Classification

3.   How Deep Neural Network Work on Multi-Classification

4.   Code Explanation with Output

What is multiclass classification?

Multi Classification, also known as multiclass classification or multiclass classification problem, is a type of classification problem where the goal is to assign input data to one of three or more classes or categories. In other words, instead of binary classification, where the goal is to assign input data to one of two classes (e.g., positive or negative), multiclass classification involves assigning input data to one of several possible classes or categories (e.g., animal species, types of products, etc.).

In multiclass classification, each input sample is associated with a single class label, and the goal of the model is to learn a function that can accurately predict the correct class label for new, unseen input data. Multiclass classification can be approached using a variety of machine learning algorithms, including decision trees, support vector machines, and deep neural networks.

Some examples of multiclass classification problems include image classification, where the goal is to classify images into one of several categories (e.g., animals, vehicles, buildings), and text classification, where the goal is to classify text documents into one of several categories (e.g., news topics, sentiment analysis).

`Example of multiclass classification-`

Here are a few examples of multiclass classification problems:

Image classification: The goal is to classify images into one of several categories. For example, an image classification model might be trained to classify images of animals into categories such as cats, dogs, and birds.

Text classification: The goal is to classify text documents into one of several categories. For example, a text classification model might be trained to classify news articles into categories such as politics, sports, and entertainment.

Disease diagnosis: The goal is to diagnose patients with one of several diseases based on their symptoms and medical history. For example, a disease diagnosis model might be trained to classify patients into categories such as diabetes, cancer, and heart disease.

Speech recognition: The goal is to transcribe spoken words into text. A speech recognition model might be trained to recognize spoken words in several languages or dialects.

Credit risk analysis: The goal is to classify loan applicants into categories such as low risk, medium risk, and high risk. A credit risk analysis model might be trained to classify loan applicants based on their creditscore, income, and other factors.

In all of these examples, the goal is to assign input data to one of several possible classes or categories.

Multiclass classification is a common task in machine learning and can be approached using a variety ofalgorithms, including decision trees, support vector machines, and deep neural networks.

Conclusion- In this way we can do Multi classification using DNN.

Assignment No: 3B

Title : Use MNIST Fashion Dataset and create a classifier to classify fashion clothinginto categories.

Objective : Students should be able to Classify movie reviews into positive reviewsand "negative reviews on IMDB Dataset.

Prerequisite:

1.   Basic of programming language

2.   Concept of Classification

3.   Concept of Deep Neural Network

Theory:

1.   What is Classification

2.   Example of Classification

3.   What is CNN?

4.   How Deep Neural Network Work on Classification

5.   Code Explanation with Output

What is Classification?

Classification is a type of supervised learning in machine learning that involves categorizing data into predefined classes or categories based on a set of features or characteristics. It is used to predict the classof new, unseen data based on the patterns learned from the labeled training data.

In classification, a model is trained on a labeled dataset, where each data point has a known class label. The model learns to associate the input features with the corresponding class labels and can then be usedto classify new, unseen data.

For example, we can use classification to identify whether an email is spam or not based on its content and metadata, to predict whether a patient has a disease based on their medical records and symptoms, orto classify images into different categories based on their visual features.

Classification algorithms can vary in complexity, ranging from simple models such as decision trees andk-nearest neighbors to more complex models such as support vector machines and neural networks. The choice of algorithm depends on the nature of the data, the size of the dataset, and the desired level

of accuracy and interpretability.

Example- Classification is a common task in deep neural networks, where the goal is to predict the classof an input based on its features. Here's an example of how classification can be performed in a deep neural network using the popular MNIST dataset of handwritten digits.

The MNIST dataset contains 60,000 training images and 10,000 testing images of handwritten digits from 0 to 9. Each image is a grayscale 28x28 pixel image, and the task is to classify each image into one of the 10 classes corresponding to the 10 digits.

We can use a convolutional neural network (CNN) to classify the MNIST dataset. A CNN is a type of deep neural network that is commonly used for image classification tasks.

What us CNN-
Convolutional Neural Networks (CNNs) are commonly used for image classification tasks, and they are designed to automatically learn and extract features from input images. Let's consider an example of using a CNN to classify images of handwritten digits.

In a typical CNN architecture for image classification, there are several layers, including convolutional layers, pooling layers, and fully connected layers. Here's a diagram of a simple CNN architecture for the digit classification task:

The input to the network is an image of size 28x28 pixels, and the output is a probability distribution over the 10 possible digits (0 to 9).

The convolutional layers in the CNN apply filters to the input image, looking for specific patterns and features. Each filter produces a feature map that highlights areas of the image that match the filter. The filters are learned during training, so the network can automatically learn which features are most relevant for the classification task.

The pooling layers in the CNN downsample the feature maps, reducing the spatial dimensions of the data. This helps to reduce the number of parameters in the network, while also making the features more robust to small variations in the input image.

The fully connected layers in the CNN take the flattened output from the last pooling layer and performa classification task by outputting a probability distribution over the 10 possible digits.

During training, the network learns the optimal values of the filters and parameters by minimizing a loss function. This is typically done using stochastic gradient descent or a similar optimization algorithm.

Once trained, the network can be used to classify new images by passing them through the network and

computing the output probability distribution.

Overall, CNNs are powerful tools for image recognition tasks and have been used successfully in many applications, including object detection, face recognition, and medical image analysis.

CNNs have a wide range of applications in various fields, some of which are:

Image classification: CNNs are commonly used for image classification tasks, such as identifyingobjects in images and recognizing faces.

Object detection: CNNs can be used for object detection in images and videos, which involves identifying the location of objects in an image and drawing bounding boxes around them.

Semantic segmentation: CNNs can be used for semantic segmentation, which involves partitioning animage into segments and assigning each segment a semantic label (e.g., "road", "sky", "building").

Natural language processing: CNNs can be used for natural language processing tasks, such as sentiment analysis and text classification.

Medical imaging: CNNs are used in medical imaging for tasks such as diagnosing diseases from X-raysand identifying tumors from MRI scans.

Autonomous vehicles: CNNs are used in autonomous vehicles for tasks such as object detection andlane detection.

Video analysis: CNNs can be used for tasks such as video classification, action recognition, and videocaptioning.

Overall, CNNs are a powerful tool for a wide range of applications, and they have been used successfully in many areas of research and industry.

How Deep Neural Network Work on Classification using CNN-

Deep neural networks using CNNs work on classification tasks by learning to automatically extractfeatures from input images and using those features to make predictions. Here's how it works:

Input layer: The input layer of the network takes in the image data as input.

Convolutional layers: The convolutional layers apply filters to the input images to extract relevant features. Each filter produces a feature map that highlights areas of the image that match the filter.

Activation functions: An activation function is applied to the output of each convolutional layer tointroduce non-linearity into the network.

Pooling layers: The pooling layers downsample the feature maps to reduce the spatial dimensions of thedata.

Dropout layer: Dropout is used to prevent overfitting by randomly dropping out a percentage of theneurons in the network during training.

Fully connected layers: The fully connected layers take the flattened output from the last pooling

layerand perform a classification task by outputting a probability distribution over the possible classes.

Softmax activation function: The softmax activation function is applied to the output of the last fullyconnected layer to produce a probability distribution over the possible classes.

Loss function: A loss function is used to compute the difference between the predicted probabilities andthe actual labels.

Optimization: An optimization algorithm, such as stochastic gradient descent, is used to minimize theloss function by adjusting the values of the network parameters.

Training: The network is trained on a large dataset of labeled images, adjusting the values of theparameters to minimize the loss function.

Prediction: Once trained, the network can be used to classify new images by passing them through thenetwork and computing the output probability distribution.

MNIST Dataset-

The MNIST Fashion dataset is a collection of 70,000 grayscale images of 28x28 pixels, representing 10 different categories of clothing and accessories. The categories include T-shirts/tops, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots.

The dataset is often used as a benchmark for testing image classification algorithms, and it is considered a more challenging version of the original MNIST dataset which contains handwritten digits. The MNIST Fashion dataset was released by Zalando Research in 2017 and has since become a popular dataset in the machine learning community.The MNIST Fashion dataset is a collection of 70,000 grayscale images of 28x28 pixels each. These images represent 10 different categories of clothing and accessories, with each category containing 7,000 images. The categories are as follows:

T-shirt/tops,Trousers ,Pullovers ,Dresses ,Coats ,Sandals, Shirts ,Sneakers ,Bags,Ankle boots

The images were obtained from Zalando's online store and are preprocessed to be normalized and centered. The training set contains 60,000 images, while the test set contains 10,000 images. The goal of the dataset is to accurately classify the images into their respective categories.

The MNIST Fashion dataset is often used as a benchmark for testing image classification algorithms, and it is considered a more challenging version of the original MNIST dataset which contains handwritten digits. The dataset is widely used in the machine learning community for research and educational purposes.

Here are the general steps to perform Convolutional Neural Network (CNN) on the MNIST Fashiondataset:

- Import the necessary libraries, including TensorFlow, Keras, NumPy, and Matplotlib.
- Load the dataset using Keras' built-in function, keras.datasets.fashion_mnist.load_data().

Thiswill provide the training and testing sets, which will be used to train and evaluate the CNN.

- Preprocess the data by normalizing the pixel values between 0 and 1, and reshaping the images tobe of size (28, 28, 1) for compatibility with the CNN.

- Define the CNN architecture, including the number and size of filters, activation functions, andpooling layers. This can vary based on the specific problem being addressed.

- Compile the model by specifying the loss function, optimizer, and evaluation metrics. Commonchoices include categorical cross-entropy, Adam optimizer, and accuracy metric.

- Train the CNN on the training set using the fit() function, specifying the number of epochs andbatch size.

- Evaluate the performance of the model on the testing set using the evaluate () function. This willprovide metrics such as accuracy and loss on the test set.

- Use the trained model to make predictions on new images, if desired, using the predict() function.

Conclusion- In this way we can classify fashion clothing into categories using CNN.

Deep Learning Mini Project -1

Title - Implement Human Face RecognitionProject title: "Celebrity Face Recognition"

Objective: To build a deep neural network model that can recognize the identities of celebrities in the "CelebA" dataset.

Theory - Human face recognition using deep neural networks involves building a neural network model that can take an image of a human face as input and accurately recognize the person in the image. Here is a sample code using TensorFlow to implement a basic face recognition system using a deep neural network:

Human face recognition using deep neural networks (DNNs) involves training a neural network to identify and distinguish between different faces. The process typically involves the following steps:

Data collection: A large dataset of face images is collected, including images of different individuals and under different lighting and pose conditions.

Data preprocessing: The face images are preprocessed to remove noise, align the faces, and normalize the illumination.

Feature extraction: The preprocessed face images are then fed into a deep neural network to extract high-level features that capture the important characteristics of a face. The neural network typically consists of several layers of convolutional and pooling operations, followed by fully connected layers that produce a feature vector.

Training: The extracted features are then used to train the neural network to distinguish between different faces. This is typically done using a supervised learning approach, where the network is trained on a labeled dataset of face images and their corresponding identities.

Testing: After the neural network has been trained, it can be tested on a separate dataset to evaluate its performance. This typically involves measuring the accuracy of the network in correctly identifying the individuals in the test dataset.

Deployment: Once the neural network has been trained and tested, it can be deployed in a real-worldapplication for face recognition. This typically involves capturing a face image, preprocessing it, and then feeding it into the neural network to obtain a feature vector. The feature vector is then compared to a database of known faces

Overall, human face recognition using DNNs is a complex process that requires a large amount of data, sophisticated neural network architectures, and careful preprocessing and training. However, with the increasing availability of large datasets and powerful computing resources, DNN-based face recognition systems have become increasingly accurate and effective in real-world applications.

Example- One example of human face recognition using DNNs is the FaceNet algorithm, which was developed by researchers at Google in 2015. FaceNet is a deep neural network that is trained to directly optimize the embedding of face images into a high-dimensional feature space, where distances between faces correspond to their similarity.

The FaceNet architecture consists of a deep convolutional neural network that takes a raw face image as input and produces a 128-dimensional feature vector as output. The network is trained on a large dataset of face images with labels that correspond to different individuals.

During training, the FaceNet network is optimized to minimize the distance between the feature vectors of images that depict the same person and maximize the distance between feature vectors of images that depict different people. This is done using a loss function called the triplet loss, which compares the distance between the feature vectors of an anchor image, a positive image (of the same person as the anchor), and a negative image (of a different person). The goal is to minimize the distance between the anchor and positive images while maximizing the distance between the anchor and negative images.

After training, the FaceNet network can be used for face recognition by capturing a face image, preprocessing it, and then feeding it into the network to obtain a 128-dimensional feature vector. This feature vector is then compared to a database of known faces by computing the Euclidean distance between the feature vector andthe feature vectors of the faces in the database. The closest matching face in the database is then returned as the recognized identity.

Overall, FaceNet is an example of a DNN-based face recognition system that has achieved high accuracy and robustness in real-world applications. It has been used in a variety of applications, including security systems, access control, and social media tagging.

import tensorflow as tf

```python
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import cv2

# Define constants

img_height = 128

img_width = 128

batch_size = 32

epochs = 10

num_classes = 10

# Load the "CelebA" dataset

df = pd.read_csv('list_attr_celeba.csv')


df = df.sample(frac=1).reset_index(drop=True)  # shuffle the dataset

df['image_id'] = df['image_id'].apply(lambda x: x[:-4]) # remove file extension from image IDsdf_train

= df[:int(len(df)*0.8)] # 80% for training

df_val = df[int(len(df)*0.8):int(len(df)*0.9)] # 10% for validationdf_test

= df[int(len(df)*0.9):] # 10% for testing



# Define data generators for training, validation, and testing sets

train_datagen = ImageDataGenerator(rescale=1./255) val_datagen

=     ImageDataGenerator(rescale=1./255)     test_datagen     =

ImageDataGenerator(rescale=1./255)
```

```
train_generator = train_datagen.flow_from_dataframe(

    dataframe=df_train,

    directory='img_align_celeba',

    x_col='image_id',

    y_col='Smiling',

    target_size=(img_height,        img_width),

    batch_size=batch_size,

    class_mode='binary')

val_generator = val_datagen.flow_from_dataframe(

    dataframe=df_val,

    directory='img_align_celeba',

    x_col='image_id',

    y_col='Smiling',

    target_size=(img_height, img_width),

    batch_size=batch_size,

    class_mode='binary')

test_generator = test_datagen.flow_from_dataframe(

    dataframe=df_test,

    directory='img_align_celeba',

    x_col='image_id',

    y_col='Smiling',

    target_size=(img_height,        img_width),

    batch_size=batch_size,

    class_mode='binary')
```

```python
# Define the neural network model

model = Sequential([

    Conv2D(32, (3,3), activation='relu', input_shape=(img_height, img_width, 3)),

    MaxPooling2D((2,2)),

    Conv2D(64, (3,3), activation='relu'),

    MaxPooling2D((2,2)),

    Conv2D(128, (3,3), activation='relu'),

    MaxPooling2D((2,2)),

    Conv2D(128, (3,3), activation='relu'),

    MaxPooling2D((2,2)),

    Flatten(),

    Dropout(0.5),

    Dense(512,            activation='relu'),

    Dense(num_classes, activation='sigmoid')

])
# Compile    the    model
model.compile(optimizer='adam',

        loss='binary_crossentropy',

        metrics=['accuracy'])
# Train the model

history = model.fit(train_generator,

        epochs=epochs,

        validation_data=val_generator)
# Evaluate the model on the test set

loss, accuracy  = model.evaluate(test_generator)

print("Test accuracy:", accuracy)
```

Predict the smiling attribute of a sample image

img = cv2.imread('sample_image.jpg')

img = cv2.resize(img, (img_height, img_width))

img = np.expand_dims(img, axis=0)

Conclusion- In this way Human Face Recognition Implemented.

Deep Learning Mini Project -2

Title - Implement Gender and Age Detection: predict if a person is a male or female and also theirage

Project title: "Gender and Age Detection on UTKFace dataset

Objective: To build a deep neural network model that can recognize Gender and Age Detection on UTKFace dataset

Theory -

Collect and prepare the dataset: In this case, we can use the "UTKFace" dataset which contains images of faces with their corresponding gender and age labels. We need to preprocess the data, like resizing the images to a uniform size, shuffling the dataset, and limit the age to a certain value (like 100 years).

Split the dataset: Split the dataset into training, validation, and testing sets. The usual split ratio is 80%, 10%, and 10%, respectively.

Define data generators: Define data generators for training, validation, and testing sets using the "ImageDataGenerator" class in Keras. This class provides data augmentation techniques that can improve the model's performance, such as rotation, zoom, and horizontal flip.

Define the neural network model: Define a convolutional neural network (CNN) model that takes the face images as input and outputs two values - the probability of being male and the predicted age. The model can have multiple convolutional and pooling layers followed by some dense layers

Compile the model: Compile the model with appropriate loss and metrics for each output (gender and age). In this case, we can use binary cross-entropy loss for gender and mean squared error (MSE) for age.

Train the model: Train the model using the fit method of the model object. We need to pass the data generators for the training and validation sets, as well as the number of epochs and batch size.

Evaluate the model: Evaluate the model's performance on the testing set using the evaluate method of the model object. This will give us the accuracy and mean absolute error (MAE) of the model.

Predict the gender and age of a sample image: Load a sample image and preprocess it. We can use the "cv2" library to read the image, resize it to the same size as the training images, and normalize it. Then, we can use the

"predict" method of the model object to get the predicted gender and age.

Source Code-

```python
import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import cv2

# Define constants

img_height = 128

img_width = 128

batch_size = 32

epochs = 10

# Load the "UTKFace" dataset df

= pd.read_csv('UTKFace.csv')

df['age'] = df['age'].apply(lambda x: min(x, 100)) # limit age to 100df

= df.sample(frac=1).reset_index(drop=True) # shuffle the dataset

df['image_path'] = 'UTKFace/' + df['image_path']

df_train = df[:int(len(df)*0.8)] # 80% for training

df_val = df[int(len(df)*0.8):int(len(df)*0.9)] # 10% for validationdf_test

= df[int(len(df)*0.9):] # 10% for testing

# Define data generators for training, validation, and testing sets

train_datagen = ImageDataGenerator(rescale=1./255) val_datagen
```

```python
=    ImageDataGenerator(rescale=1./255)    test_datagen    =

ImageDataGenerator(rescale=1./255)        train_generator        =

train_datagen.flow_from_dataframe(

    dataframe=df_train,

    x_col='image_path',

    y_col=['male', 'age'],

    target_size=(img_height, img_width),

    batch_size=batch_size,

    class_mode='raw')

val_generator = val_datagen.flow_from_dataframe(

    dataframe=df_val,

    x_col='image_path',

    y_col=['male', 'age'],

    target_size=(img_height, img_width),

    batch_size=batch_size,

    class_mode='raw')

test_generator = test_datagen.flow_from_dataframe(

    dataframe=df_test,
    x_col='image_path',

    y_col=['male', 'age'],

    target_size=(img_height, img_width),

    batch_size=batch_size,

    class_mode='raw')

# Define the neural network model

model = Sequential([

    Conv2D(32, (3,3), activation='relu', input_shape=(img_height, img_width, 3)),
```

```python
    MaxPooling2D((2,2)),

    Conv2D(64, (3,3), activation='relu'),

    MaxPooling2D((2,2)),

    Conv2D(128, (3,3), activation='relu'),

    MaxPooling2D((2,2)),

    Conv2D(128, (3,3), activation='relu'),

    MaxPooling2D((2,2)),

    Flatten(),

    Dropout(0.5),

    Dense(512, activation='relu'),

    Dense(2)

])
# Compile the model
model.compile(optimizer='adam',

        loss={'dense_1': 'binary_crossentropy', 'dense_2': 'mse'},

        metrics={'dense_1': 'accuracy', 'dense_2': 'mae'})
# Train the model
history = model.fit(train_generator,

epochs=epochs, validation_data=val_generator)
# Evaluate the model on the test set
loss, accuracy, mae = model.evaluate(test_generator)

print("Test accuracy:", accuracy)

print("Test MAE:", mae)
# Predict the gender and age of a sample image
img = cv2.imread('sample_image.jpg')

img
```

Conclusion- In this way Gender Age Detection Implemented.

Deep Learning Mini Project -3

Title - Implement Colorizing Old B&W Images: color old black and white images to colorful images

Project title: Colorizing Old B&W Images using CNN

Objective: To build a deep neural network model that can Colorizing Old B&W Images: color old black and white images to colorful images using CNN

Theory - Colorizing black and white images to colorful images involves a complex process that requires expertise and specialized software. However, here are some general steps involved in the process:

Scan the black and white image: The first step is to scan the black and white image and convert it into a digital format.

Preprocess the image: The image needs to be preprocessed to remove any scratches, dust, or other defects. This can be done using image editing software like Photoshop or GIMP.

Convert the image to grayscale: The black and white image needs to be converted to grayscale. This can be done using image editing software or programming languages like Python.

Collect training data: The next step is to collect training data for the colorization model. This can include a dataset of colorful images with their corresponding grayscale versions.

Train the colorization model: A deep learning model can be trained to colorize grayscale images using a dataset of colorful images. This model can be trained using software like TensorFlow, PyTorch, or Keras.

Apply the colorization model to the black and white image: Once the model is trained, it can be applied to the black and white image to generate a colorized version. This can be done using programming languages like Python.

Refine the colorized image: The colorized image may need some manual refinement to ensure that the colors are accurate and the image looks natural. This can be done using image editing software like Photoshop or GIMP.

Save the final image: Once the image has been colorized and refined, it can be saved in a digital format for printing or online use.

Note that the quality of the colorized image will depend on the quality of the original black and white image,

the accuracy of the colorization model, and the manual refinement process.

Source Code-

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, UpSampling2D, InputLayer, BatchNormalization

from   tensorflow.keras.preprocessing.image   import   ImageDataGenerator,   array_to_img,   img_to_array,

load_img

import numpy as np

Load the grayscale image

img_gray = load_img('bw_image.jpg', grayscale=True)#

Resize the image to the desired size

img_gray = img_gray.resize((256,256)) #

Convert the image to a numpy array

img_gray = img_to_array(img_gray)

Normalize the image

img_gray = img_gray / 255.0
# Add a new dimension to the array to make it compatible with the input shape of the model

img_gray = np.expand_dims(img_gray, axis=0)

Load the pre-trained model

model = Sequential()

model.add(InputLayer(input_shape=(None,   None,   1)))

model.add(Conv2D(64, (3,3), activation='relu', padding='same', strides=2))

model.add(Conv2D(128,   (3,3),   activation='relu',   padding='same'))

model.add(Conv2D(128, (3,3), activation='relu', padding='same', strides=2))

model.add(Conv2D(256, (3,3), activation='relu', padding='same'))

```
model.add(Conv2D(256, (3,3), activation='relu', padding='same', strides=2))

model.add(Conv2D(512,      (3,3),      activation='relu',      padding='same'))

model.add(Conv2D(512,      (3,3),      activation='relu',      padding='same'))

model.add(Conv2D(256,      (3,3),      activation='relu',      padding='same'))

model.add(UpSampling2D((2,2)))

model.add(Conv2D(128, (3,3), activation='relu', padding='same'))

model.add(UpSampling2D((2,2)))

model.add(Conv2D(64, (3,3), activation='relu', padding='same'))

model.add(Conv2D(32, (3,3), activation='relu', padding='same'))

model.add(Conv2D(2, (3, 3), activation='tanh', padding='same'))

model.add(UpSampling2D((2,                    2)))

model.compile(optimizer='adam',  loss='mse')  #

Load        the        pre-trained        weights

model.load_weights('colorization_weights.h5') #

Colorize the grayscale image

img_colorized = model.predict(img_gray)
# Save the colorized image

img_colorized = img_colorized * 128 + 128

img_colorized = np.clip(img_colorized, 0, 255).astype('uint8')

img_colorized        =        array_to_img(img_colorized[0])

img_colorized.save('colorized_image.jpg')
```

Conclusion- In this way coloring B & W images Implemented.