Group A

Assignment

No: 1

Title: Design and implement Parallel Breadth First Search and Depth First Search based onexisting algorithms using OpenMP. Use a Tree or an undirected graph for BFS and DFS

Objective: Students should be able to perform Parallel Breadth First Searchbased on existing algorithms using OpenMP

Outcome: Students will be understand the implementation of BFS and DFS

Prerequisite:

1. Basic of programming language
2. Concept of BFS and DFS
3. Concept of Parallelism

Contents for Theory:

1. **What is BFS?**
2. **Example of BFS**
3. **What is DFS?**
4. **Example of DFS**
5. **Concept of OpenMP**
6. **How Parallel BFS Work**
7. **How Parallel DFS Work**
8. **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 inbreadth-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

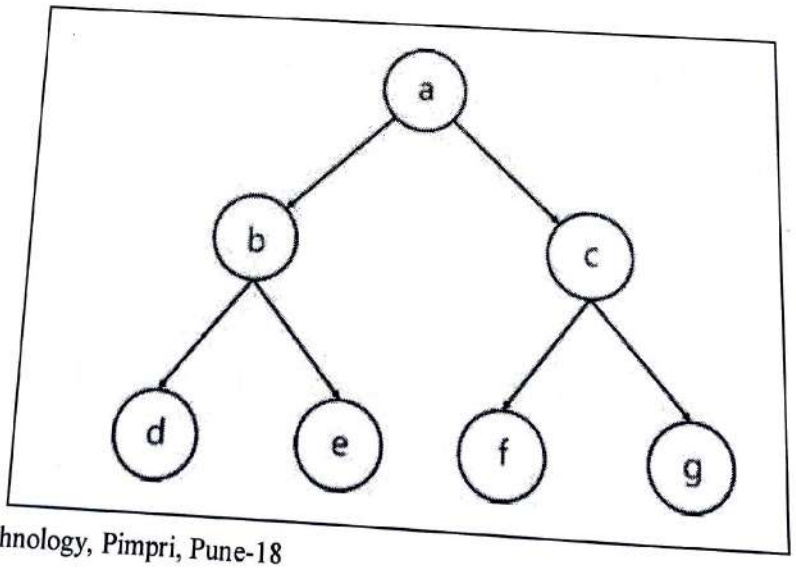
Now let's take a look at the steps involved in traversing a graph by using Breadth-First Search:

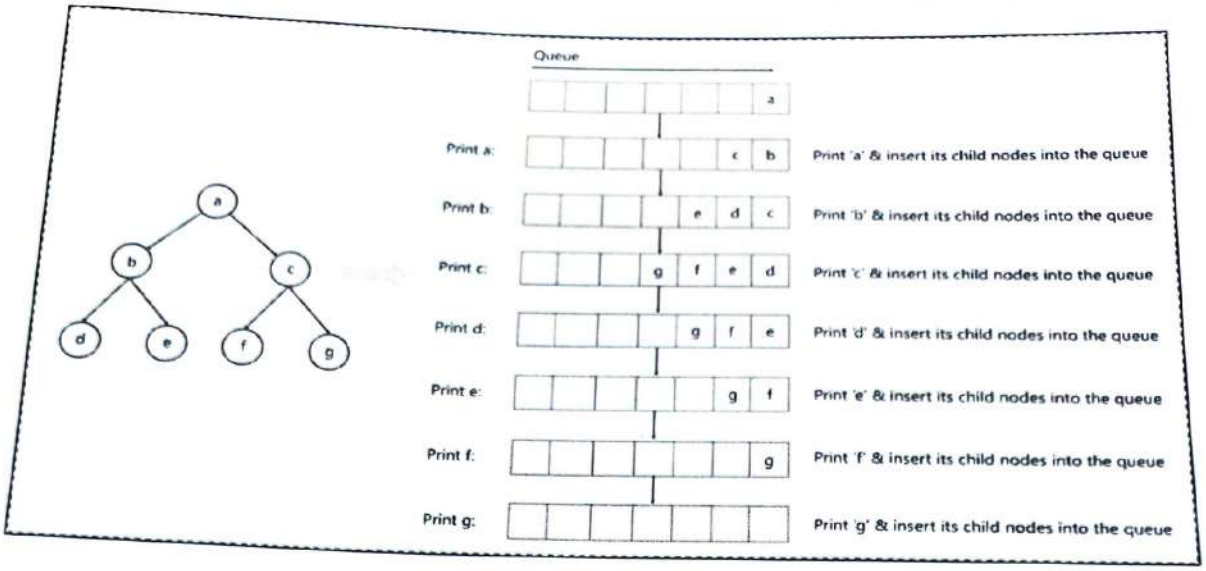
Step 1: Take an Empty Queue.

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.





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.

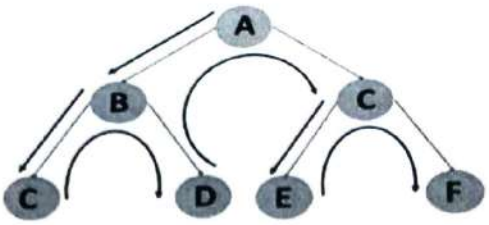
DFS can be implemented using either a recursive or an iterative approach. The recursive approach is simpler to implement but can lead to a stack overflow error for very large graphs. The iterative approach uses a stack to keep track of nodes to be explored and is preferred for larger graphs.

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 tothe stack.

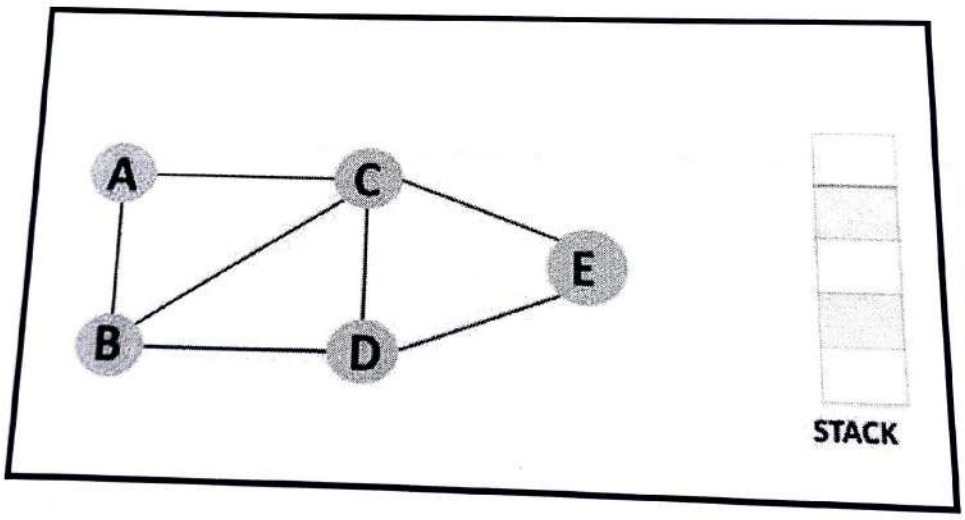
Step 3 - Push any non-visited adjacent vertices of a vertex at the top of the stack to the top of thestack.

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

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.

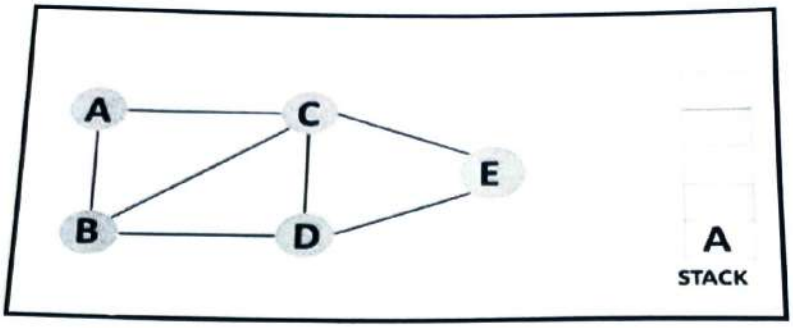
Step 7 - When the stack is entirely unoccupied, create the final spanning tree by deleting the graph'sunused edges.

Consider the following graph as an example of how to use the dfs algorithm.

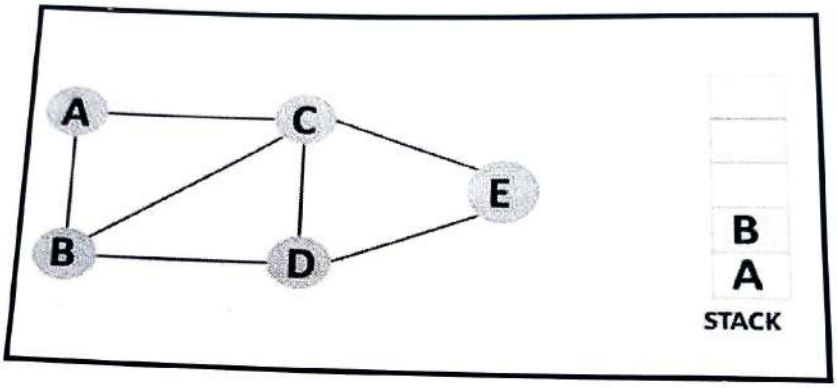


Step 1: Mark vertex A as a visited source node by selecting it as a source node.

- You should push vertex A to the top of the stack.

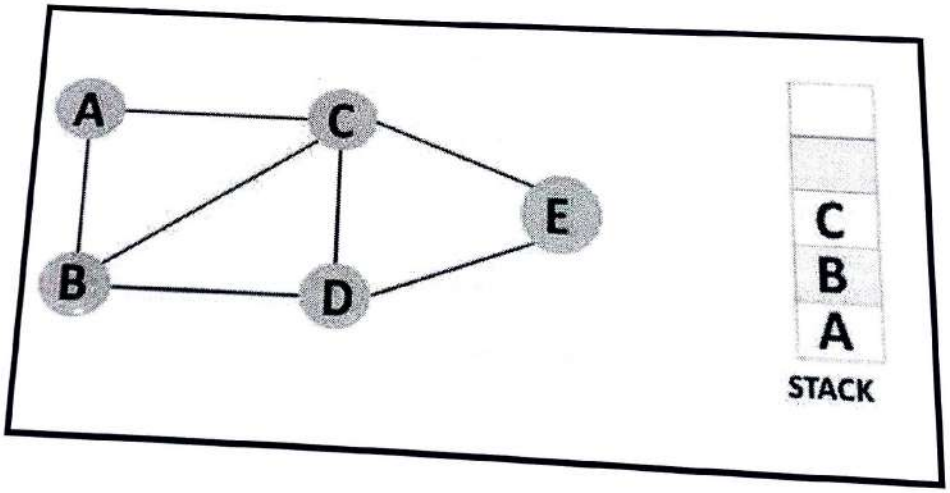


Step 2: Any nearby unvisited vertex of vertex A, say B, should be visited. You should push vertex B to the top of the stack



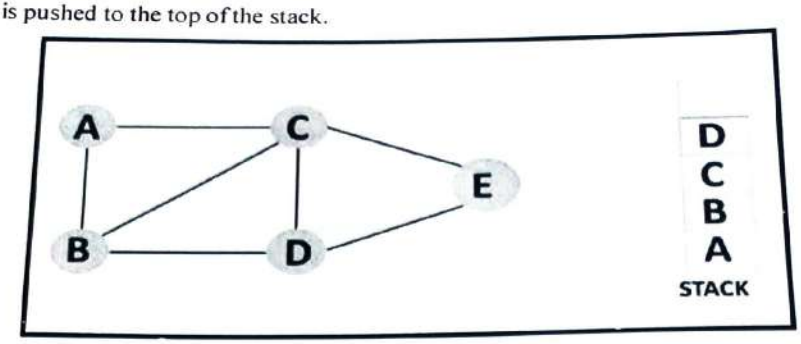
Step 3: From vertex C and D, visit any adjacent unvisited vertices of vertex B. Imagine you havechosen vertex C, and you want to make C a visited vertex.

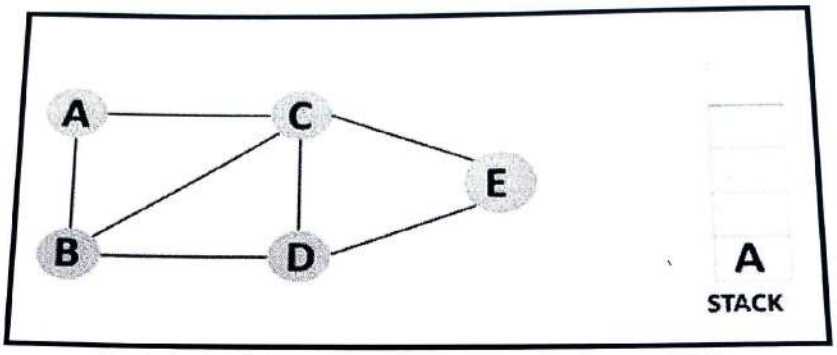
- Vertex C is pushed to the top of the stack.



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.

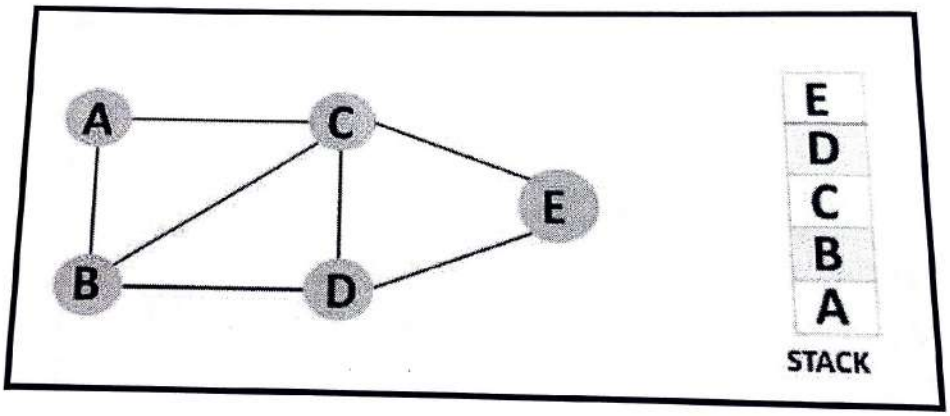
- Vertex D is pushed to the top of the stack.



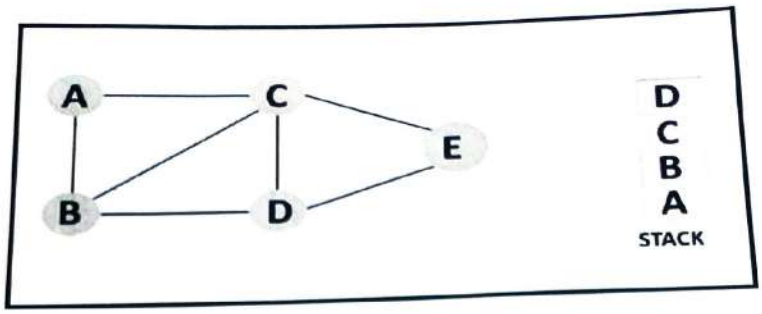


Step 5: Vertex E is the lone unvisited adjacent vertex of vertex D, thus marking it as visited.

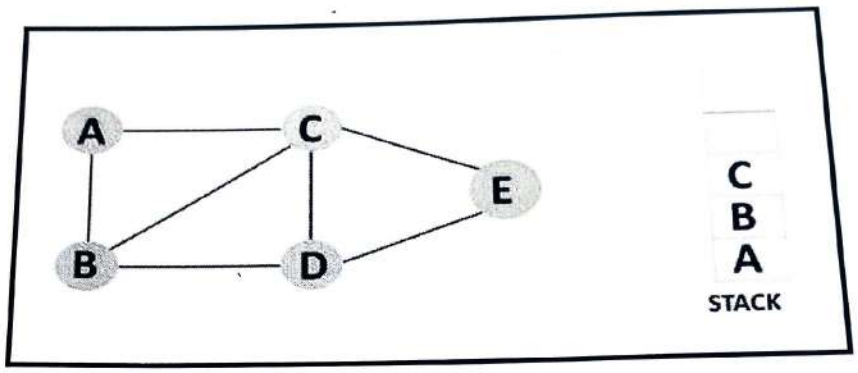
- Vertex E should be pushed to the top of the stack.



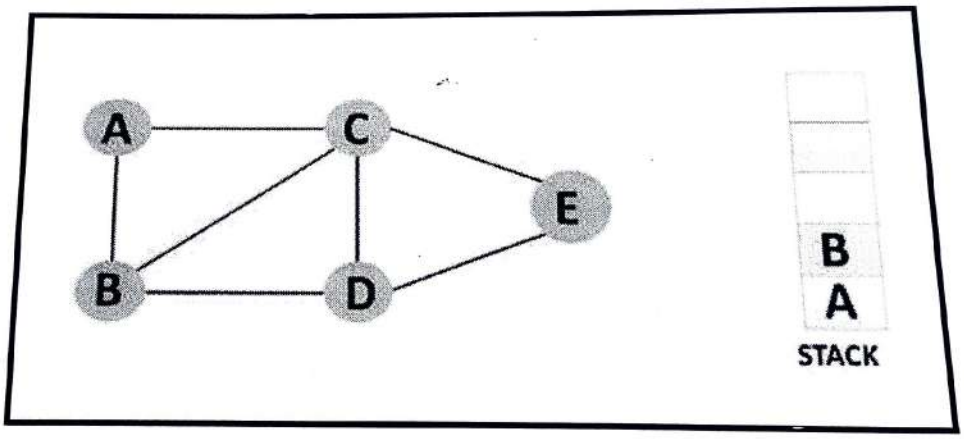
Step 6: Vertex E's nearby vertices, namely vertex C and D have been visited, pop vertex E fromthe stack.



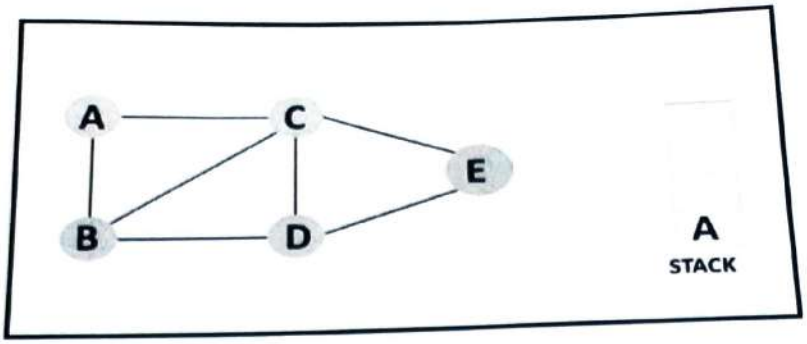
Step 7: Now that all of vertex D's nearby vertices, namely vertex B and C, have been visited, popvertex D from the stack.



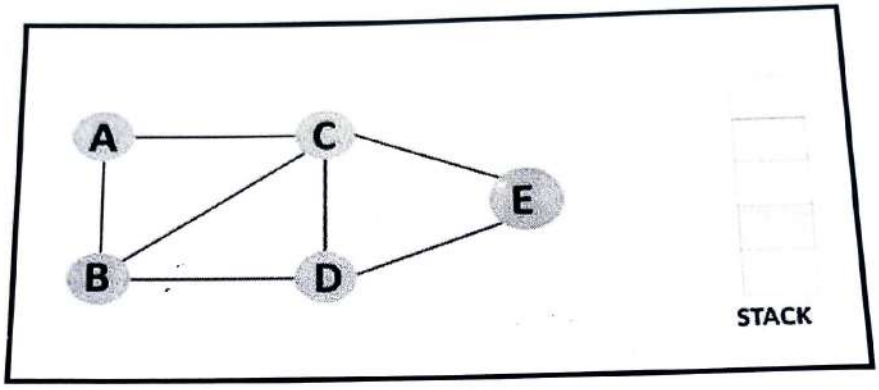
Step 8: Similarly, vertex C's adjacent vertices have already been visited; therefore, pop it fromthe stack.



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-joinmodel 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 widerangeof 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 bevisited and marks each visited node to avoid processing it again.
- The algorithm then proceeds in levels, where each level represents a set of nodes that are at a certain distance from the root node. Each thread processes the nodes in its local queue at the current level, 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. Inthe 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 otherthreads or processors.
- The parallel BFS algorithm terminates when all nodes have been visited or when a specified node hasbeen 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.

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 exploredsimultaneously. 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 BFS and DFS

FAQ

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 algorithmusing OpenMP.
5. Write Down Commands used in OpenMP?
6. What if DFS?
7. Write a parallel Depth First Search (DFS) algorithm using OpenMP
8. What is the advantage of using parallel programming in DFS?
9. How can you parallelize a DFS algorithm using OpenMP?
10. What is a race condition in parallel programming, and how can it be avoided inOpenMP?

Group A

Assignment

No: 2

Title: 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.

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

Outcome: Students will be understand the implementation of Parallel Bubble Sort and Merge Sort

Prerequisite:

1. Basic of programming language
2. Concept of Bubble Sort and Merge Sort
3. Concept of Parallelism

Contents for Theory:

1. **What is Bubble Sort? Use of Bubbie Sort**
2. **Example of Bubble sort?**
3. **What is Merge? Use of Merge Sort**
4. **Example of Merge sort?**
5. **Concept of OpenMP**
6. **How Parallel Bubble Sort Work**
7. **How Parallel Merge Sort Work**
8. **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 theend 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 morecomplex 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.

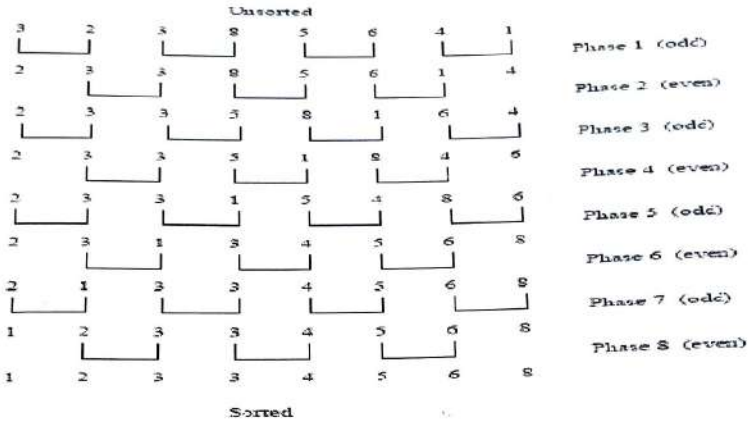
Example of Bubble sort

Here we want to sort an array containing [8, 5, 1]. The following figure shows how we can sortthis array using bubble sort. The elements in consideration are shown in **bold.**

8, 5, 1	Switch 8 and 5
5, **8, 1**	Switch 8 and 1
5, 1, 8	Reached end start again.
5, 1, 8	Switch 5 and 1
1, **5, 8**	No Switch for 5 and 8
1, 5, 8	Reached end start again.
1, 5, 8	No switch for 1, 5
1, **5, 8**	No switch for 5, 8
1, 5, 8	Reached end.

But do not start again since this is the nth iteration of same process

Odd Even Transposition:



Sorting n= 8 elements, using the odd-even transposition sort algorithm. During each phase, n= 8 elements are compared.

What is Merge Sort?

Merge Sort is a classical sorting algorithm using a divide-and-conquer approach. The initial unsorted list is first divided in half, each half sub list is then applied the same division method until individual elements are obtained. Pairs of adjacent elements / sub lists are then merged into sorted sub lists until the one fully merged and sorted list is obtained.

To sort $A[p \,..\, r]$:

a. Divide Step

If a given array A has zero or one element, simply return; it is already sorted. Otherwise,

split $A[p .. r]$ into two subarrays $A[p .. q]$ and $A[q + 1 .. r]$, each containing about half of the elements of $A[p .. r]$. That is, q is the halfway point of $A[p .. r]$.

b. Conquer Step

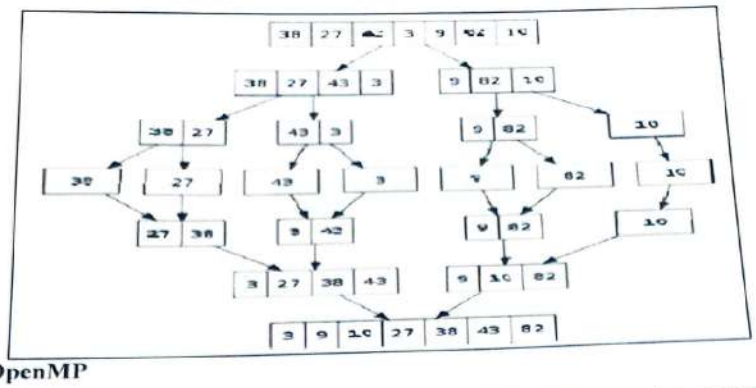
Conquer by recursively sorting the two subarrays $A[p .. q]$ and $A[q + 1 .. r]$.

c. Combine Step

Combine the elements back in $A[p .. r]$ by merging the two sorted subarrays $A[p .. q]$ and $A[q + 1 .. r]$ into a sorted sequence. To accomplish this step, we will define a procedure MERGE (A, p, q, r).

- 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 outputarray.
- 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 valuesis 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



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 executea parallel region of the code, and then waits for all threads to complete before continuing withthe 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 suppo

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 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 parallelprocessing capabilities.

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 reducethe impact of variations in system load and other factors.
- Monitor system resource usage during execution, such as CPU utilization and memoryconsumption, to detect any performance bottlenecks.
- Visualize the results using charts or graphs to make it easier to compare the performance of thetwo algorithms.

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

1. **Execution time:** Execution time is the amount of time it takes for the algorithm to completeits sorting operation. This metric can be used to compare the speed of sequential and parallelmergesort 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 theparallel algorithm is faster than the sequential algorithm.
3. **Efficiency:** Efficiency is the ratio of the speedup to the number of processors or cores used intheparallel algorithm. This metric can be used to determine how well the parallel algorithm isutilizing the available resources.

4. **Scalability**: Scalability is the ability of the algorithm to maintain its performance as the input sizeand number of processors or cores increase. A scalable algorithm will maintain a consistentspeedup 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 executiontime, 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, youcan use profiling tools to analyze the performance of the algorithms and identifyareas for optimization

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 processesand 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 and Merge Sort in parallel way using OpenMP alsocome to know how to measure performance of serial and parallel algorithm

FAQ:

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
6. What is parallel Merge Sort?
7. How does Parallel Merge Sort work?
8. How do you implement Parallel MergeSort using OpenMP?
9. What are the advantages of Parallel MergeSort?
10. Difference between serial Mergesort and parallel Mergesort

Group A

Assignment

No: 3

Title: Implement Min, Max, Sum and Average operations using ParallelReduction.

Objective: To understand the concept of parallel reduction and how it can be usedto perform basicmathematical operations on given data sets.

Outcome: Students will be understand the implementation of basic operation using parallel reduction

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 aretheir advantages and limitations?

Parallel Reduction.

Here's a **function-wise manual** on how to understand and run the sample C++ program that demonstrateshow 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 thevector using parallel reduction.
   - The OpenMP reduction clause is used with the "min" operator to find the minimum valueacross all threads.
   - The minimum value found by each thread is reduced to the overall minimum value of theentire 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 thevector using parallel reduction.
   - The OpenMP reduction clause is used with the "max" operator to find the maximum valueacross all threads.
   - The maximum value found by each thread is reduced to the overall maximum value of theentire 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 thevector using parallel reduction.
   - The OpenMP reduction clause is used with the "+" operator to find the sum across allthreads.
   - 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 inthe vector using parallel reduction.
   - The OpenMP reduction clause is used with the "+" operator to find the sum across allthreads.

- 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, andaverage_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, compilethe 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 terminaland 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 theseoperations using parallel reduction in C++ with OpenMP. We also provided a manual for running OpenMP programs on the Ubuntu platform.

FAQ

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 basedon input size?
5. How can you modify the provided code example for more complex operations using parallelreduction?

Group A

Assignmen

t 4

Title: Write a CUDA Program for:

1. Addition of two large vectors
2. Matrix Multiplication using CUDA C

Objective: Students should be able to perform Addition of two large vectors and MatrixMultiplication using CUDA C

Outcome: Students can design and implement solutions for parallel environment.

Prerequisite:

1. CUDA Concept
2. Vector Addition
3. Matrix Multiplication
4. How to execute Program in CUDA Environment

Contents for Theory:

1. **What is CUDA**
2. **Addition of two large Vector**
3. **Matrix Multiplication**
4. **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 toallocate 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 thedevice. 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.

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 usestandard 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 functioncudaMalloc() 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 theCUDA 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 andthreads 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 kernelexecutions 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 functioncudaFree() to deallocate memory on the device.

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 twolarge 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 thatcomes 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 filegenerated 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.

Conclusion: We have implemented the Addition of two large vectors and Matrix MultiplicationusingCUDA C

FAQ

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 thedevice 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
5. What are the advantages of using CUDA to perform matrix multiplication compared to using aCPU?
6. How do you handle matrices that are too large to fit in GPU memory in CUDA matrix multiplication?
7. How do you optimize the performance of the CUDA program for matrix multiplication?
8. How do you ensure correctness of the CUDA program for matrix multiplication and verify theresults?

Group B Deep

LearningAssignment

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 usingDeep Neural networkon Boston House Dataset.

Outcome: Student can apply the technique of Deep Neural network for implementing Linear regression

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 gradientdescent, 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 layers are 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 the weights 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.

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 aspredicting 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 accurate predictions 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.

FAQ:

1. What is Linear Regression?
2. What is a Deep Neural Network?
3. What is the concept of standardization?
4. Why split data into train and test?
5. Write Down Application of Deep Neural Network?

Group B Deep

LearningAssignment

No: 2B

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 "negativereviews on IMDB Dataset.

Outcome: Student can apply the technique of Deep Neural network for implementing classification

Prerequisite:

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

Contents for 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.

FAQ:

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?

Group B Deep

LearningAssignment

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 "negativereviews on IMDB Dataset.

Outcome: Student can apply the technique of Convolution (CNN) for implementing Deep Learningmodels.

Prerequisite:

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

Contents for 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 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.

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 perform a 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 identifying objects 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 an image 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-rays and identifying tumors from MRI scans.

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

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

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 to introduce 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 fully connected 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 the parameters to minimize the loss function.

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

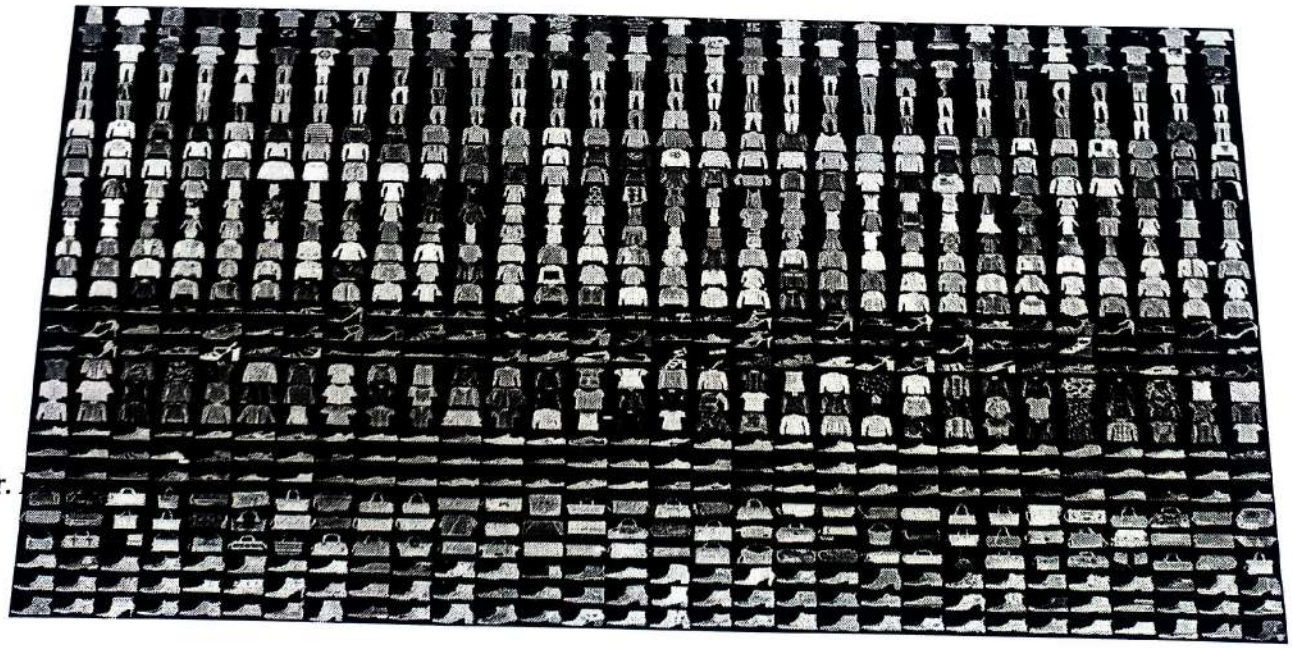
MNIST Fashion dataset was released by Zalando Research in 2017 and has since become a popular dataset in the machine learning community.

he 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 7,000images. The categories are as follows:

T-shirt/tops
Trousers
Pullovers
Dresses
Coats
Sandals
Shirts
Sneakers
s 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.

FAQ:

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?