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

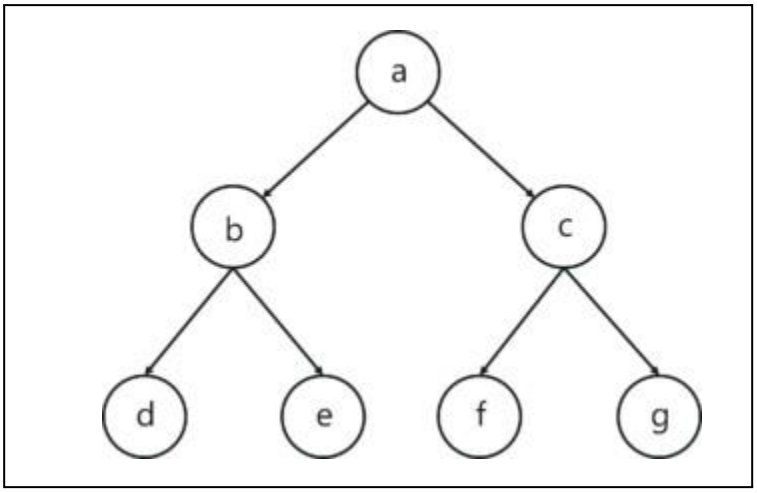
Now let's take a look at the steps involved in traversing a graph by using Breadth-First Search:

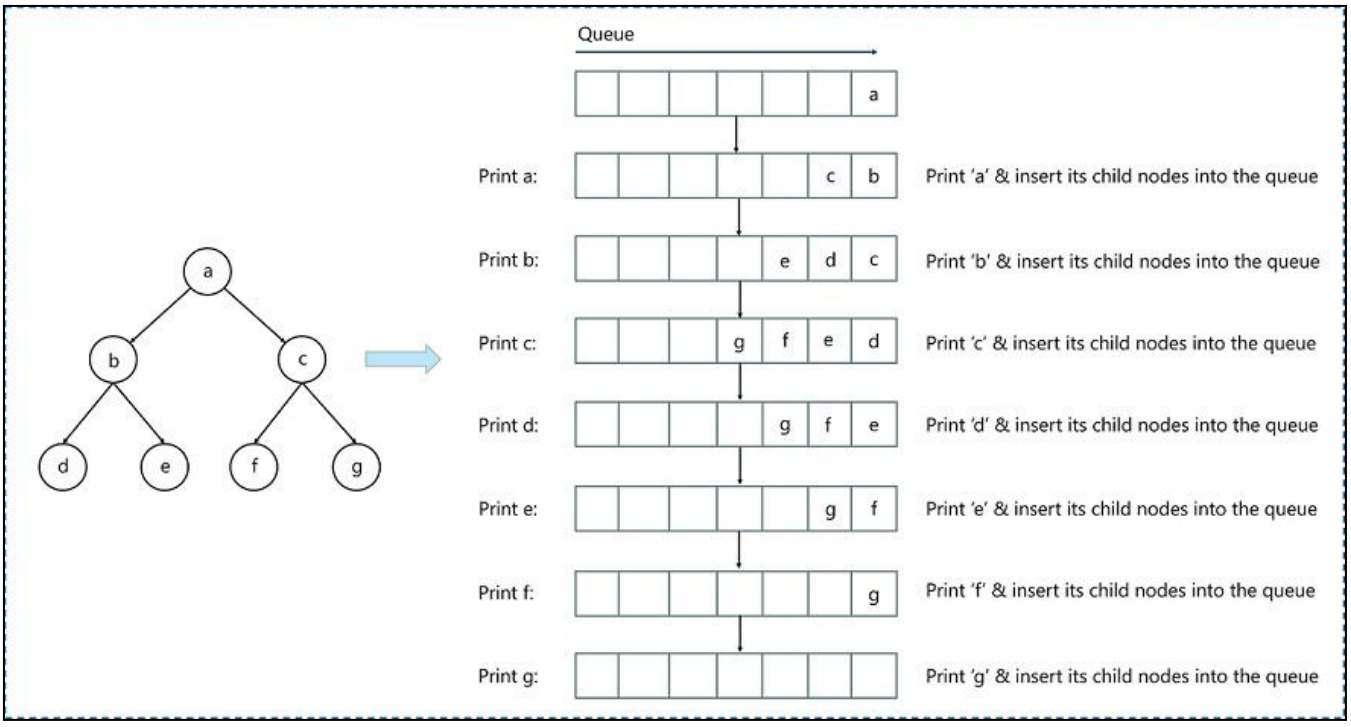
Step 1: Take an Empty Queue.

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 wide range 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 DF**

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.

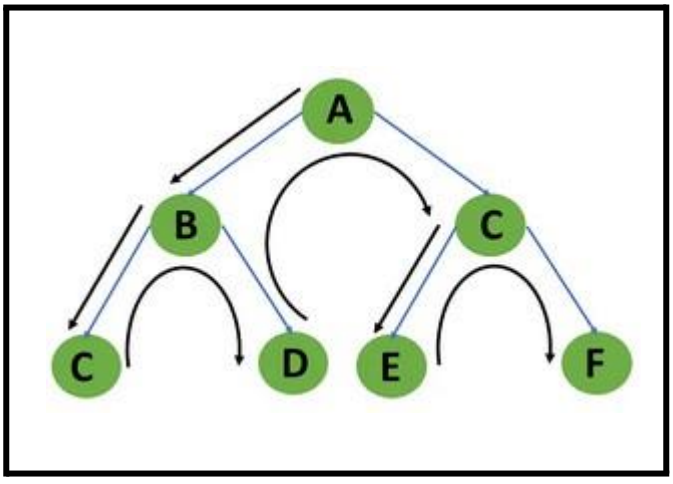
DFS can be implemented using either a recursive or an iterative approach. The recursive approach is simpler to implement but can lead to a stack overflow error for very large graphs. The iterative approach uses a stack to keep track of nodes to be explored and is preferred for larger graphs.

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.

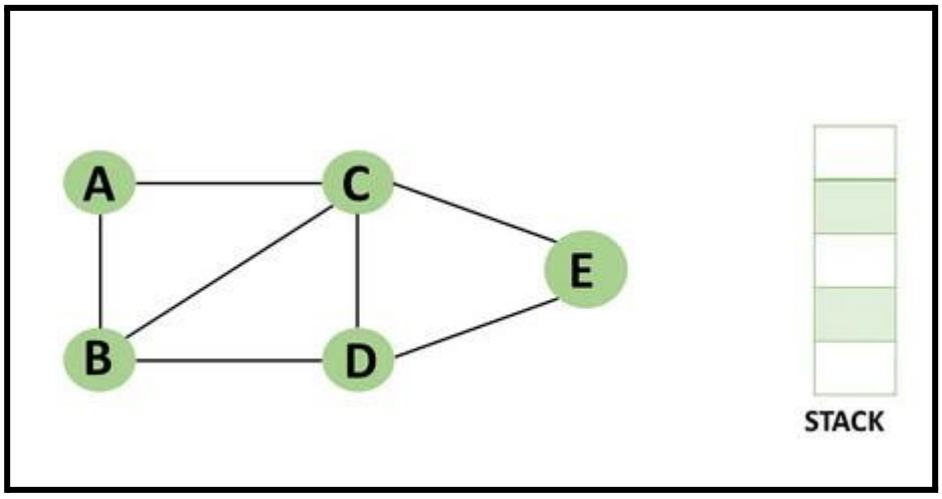
Step 4 - Repeat steps 3 and 4 until there are no more vertices to visit from the vertex at the top of the stack.

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.

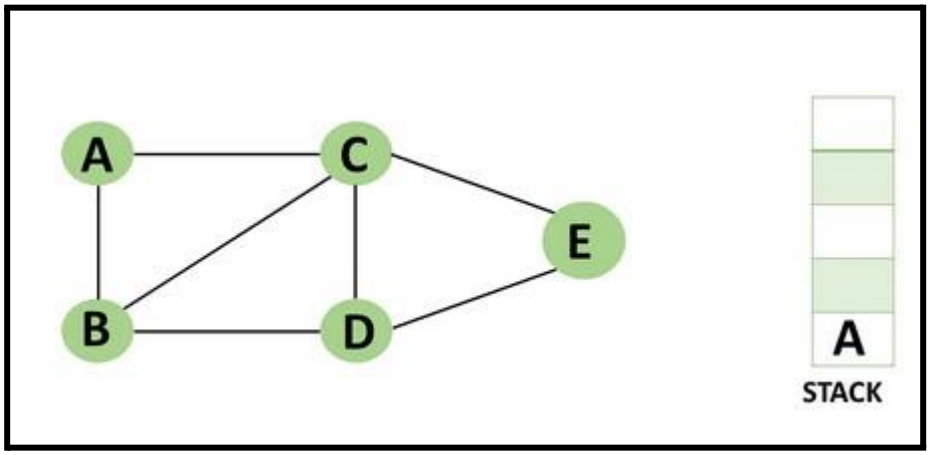
Step 7 - When the stack is entirely unoccupied, create the final spanning tree by deleting the graph's unused edges.

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



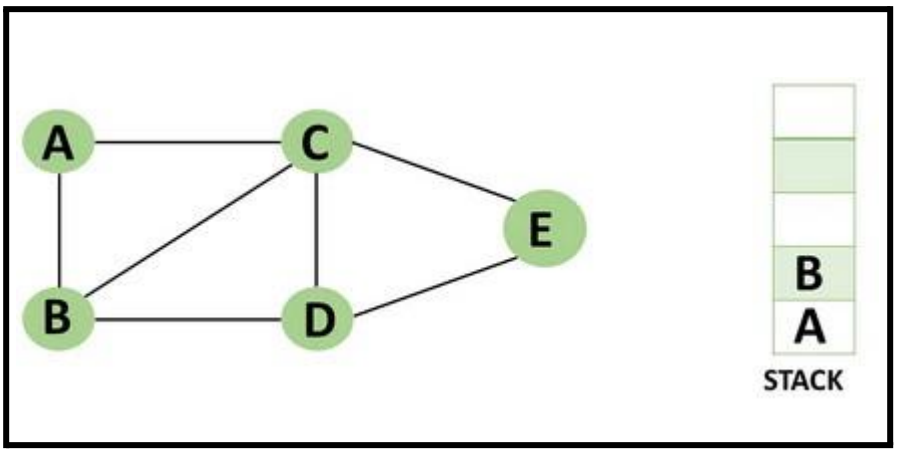
Step 1: Mark vertex A as a visited source node by selecting it as a source node.

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



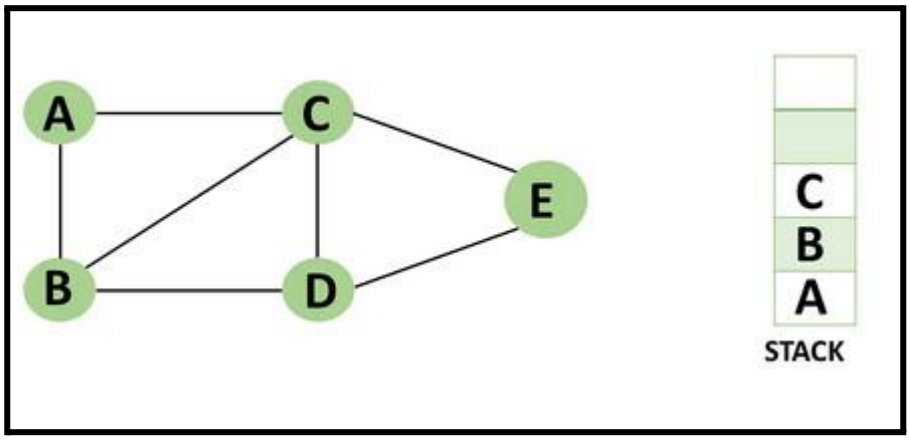
Step 2: Any nearby unvisited vertex of vertex A, say B, should be visited.

You should push vertex B to the top of the stack



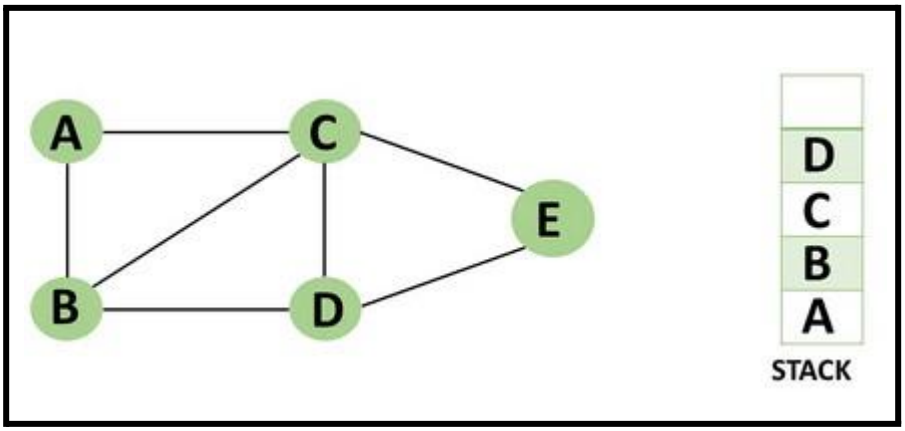
Step 3: From vertex C and D, visit any adjacent unvisited vertices of vertex B. Imagine you have chosen vertex C, and you want to make C a visited vertex.

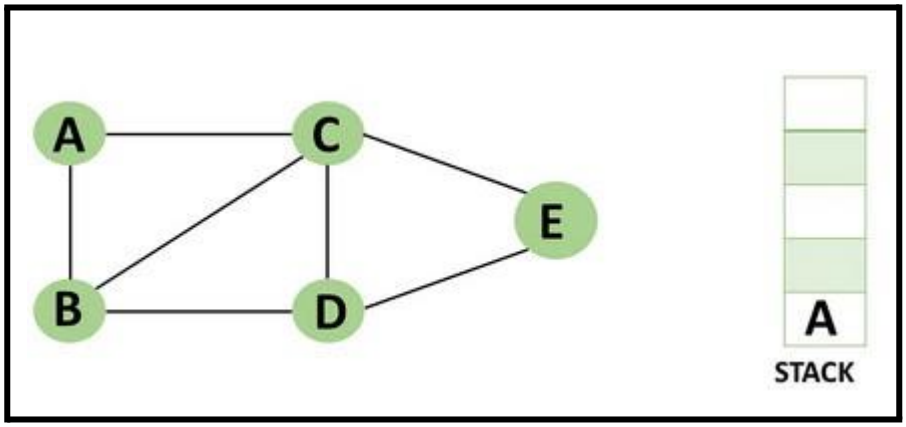
- Vertex C is pushed to the top of the stack.



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.

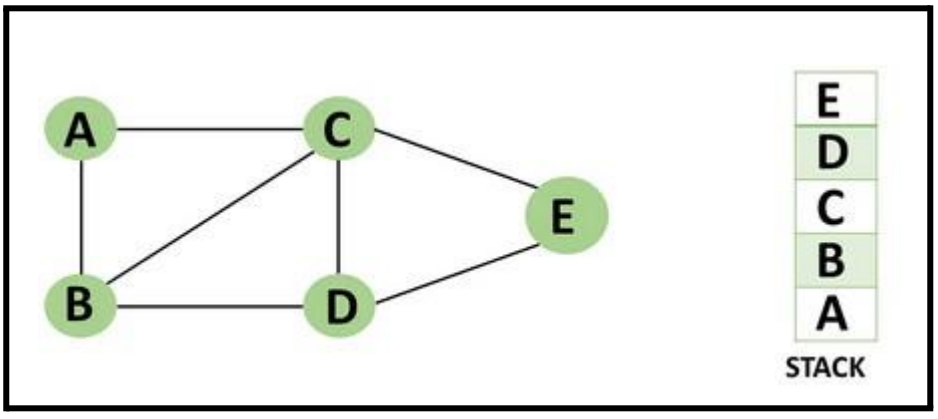
- Vertex D is pushed to the top of the stack.



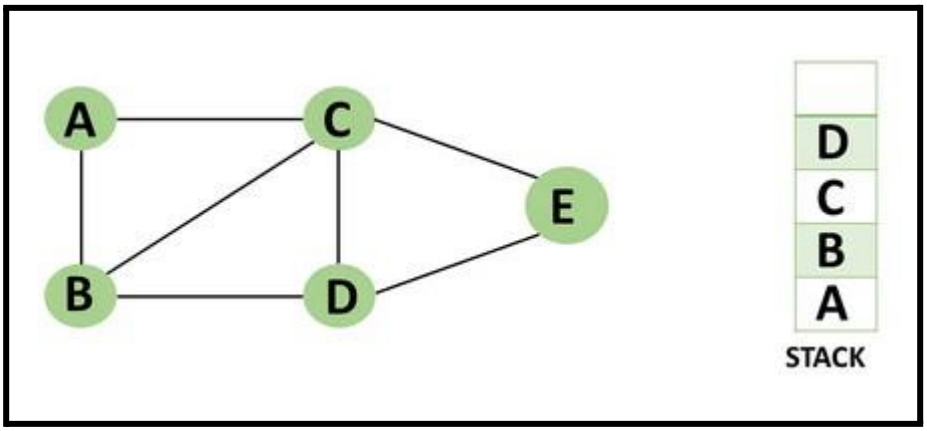


Step 5: Vertex E is the alone unvisited adjacent vertex of vertex D, thus marking it as visited.

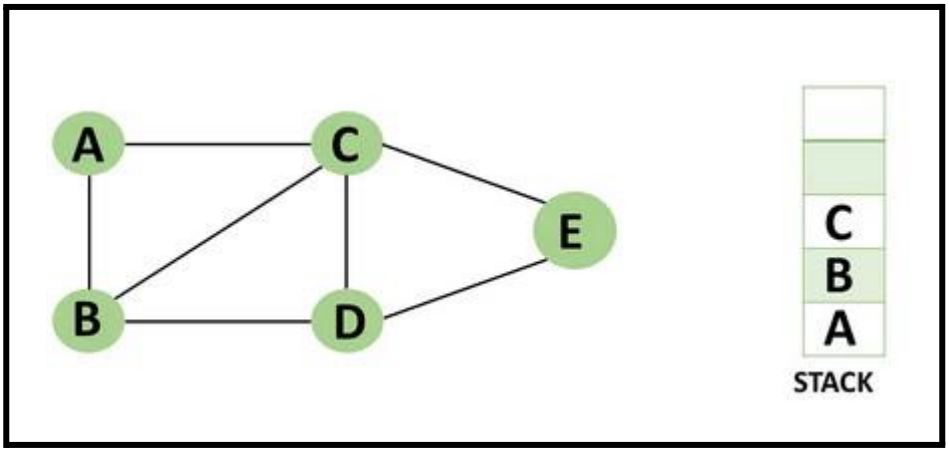
- Vertex E should be pushed to the top of the stack.



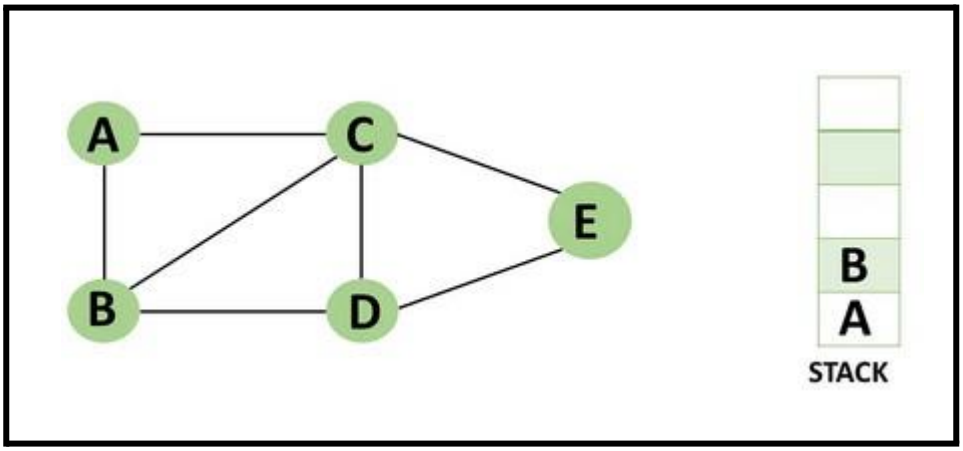
Step 6: Vertex E's nearby vertices, namely vertex C and D have been visited, pop vertex E from the stack.



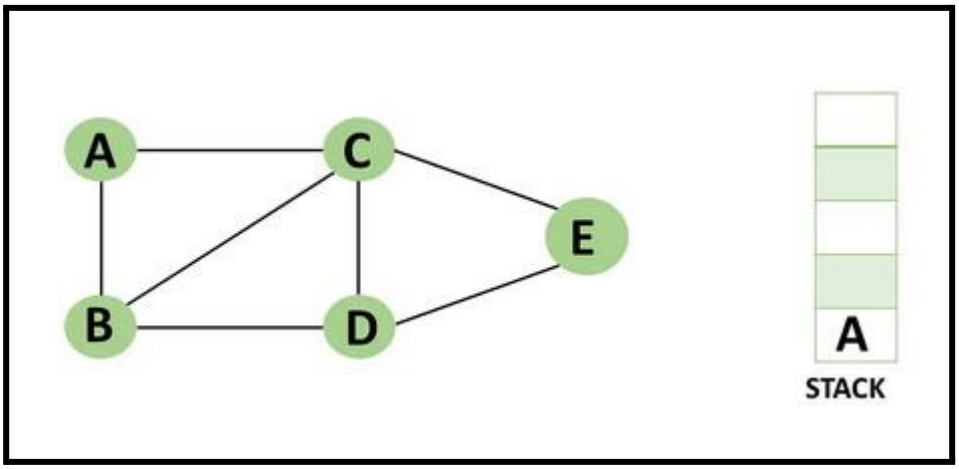
Step 7: Now that all of vertex D's nearby vertices, namely vertex B and C, have been visited, pop vertex D from the stack.



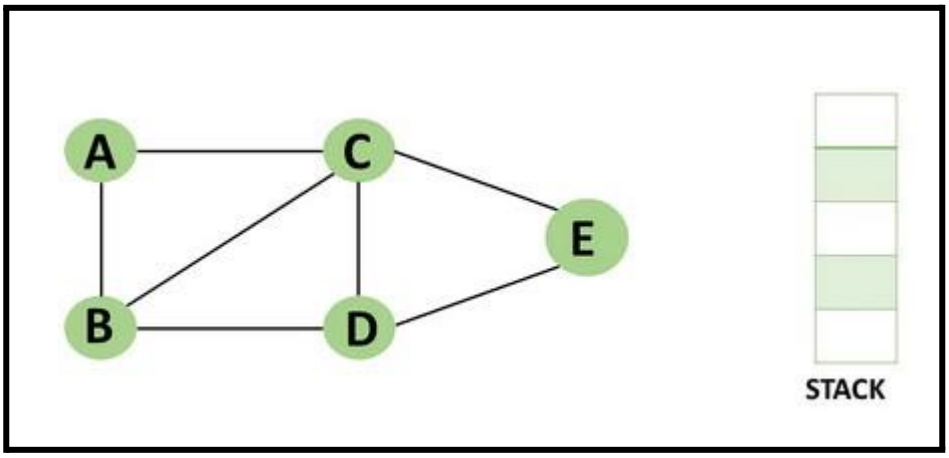
Step 8: Similarly, vertex C's adjacent vertices have already been visited; therefore, pop it from the stack.



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.

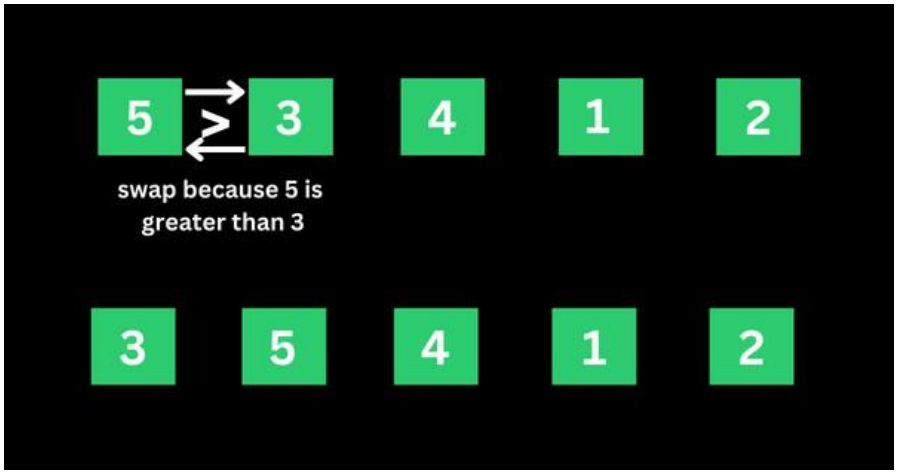
Example of Bubble sort

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…

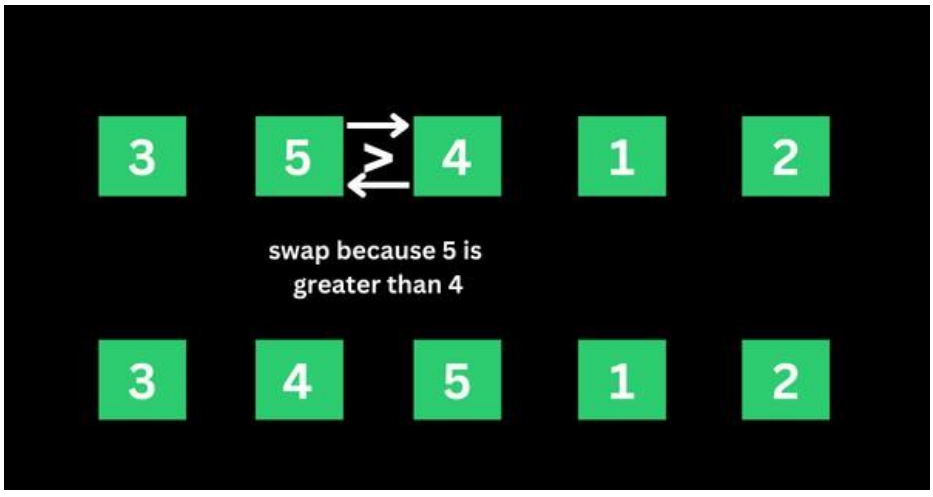
The sorting begins the first iteration by comparing the first two values. If the first value is greater than the second, the algorithm pushes the first value to the index of the second value.

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.

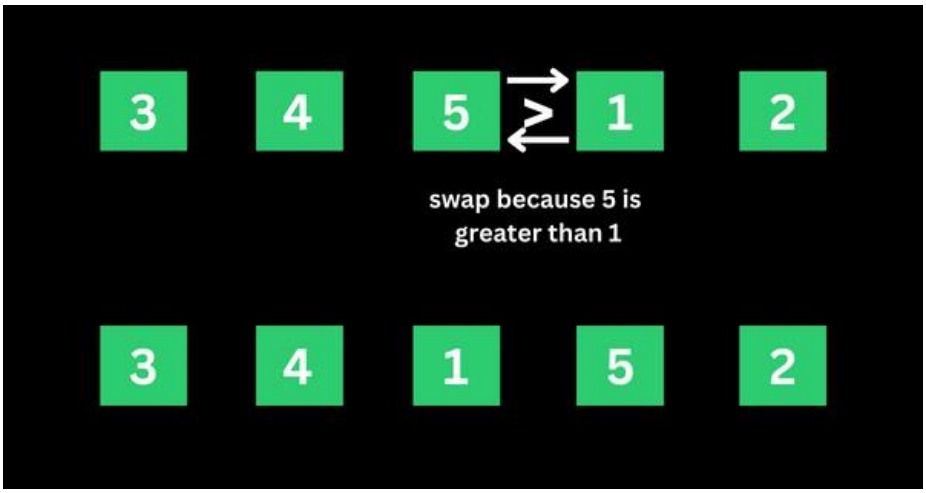


Step 2: The algorithm now has 3, 5, 4, 1, and 2 to compare, this time around, it compares the next two values, which are 5 and 4. 5 is greater than 4, so 5 takes the index of 4 and the values now become 3, 4, 5, 1, and 2.

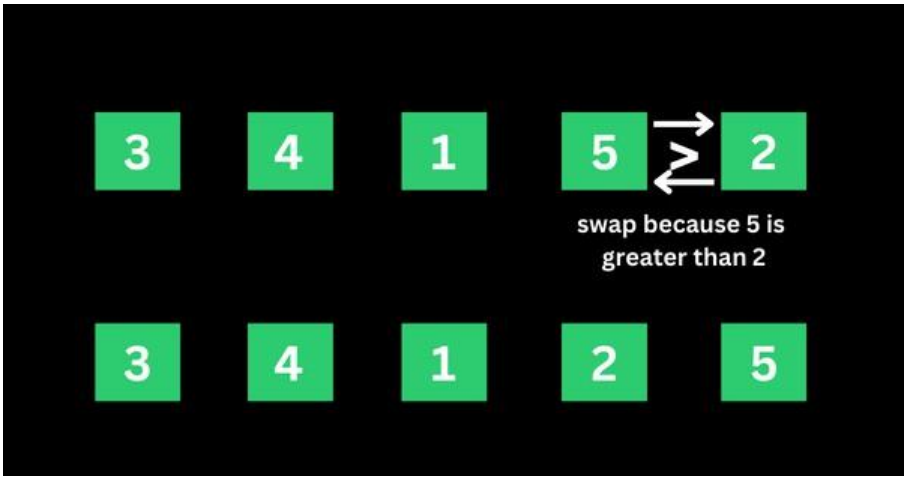


Step 3: The algorithm now has 3, 4, 5, 1, and 2 to compare. It compares the next two values, which are 5

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



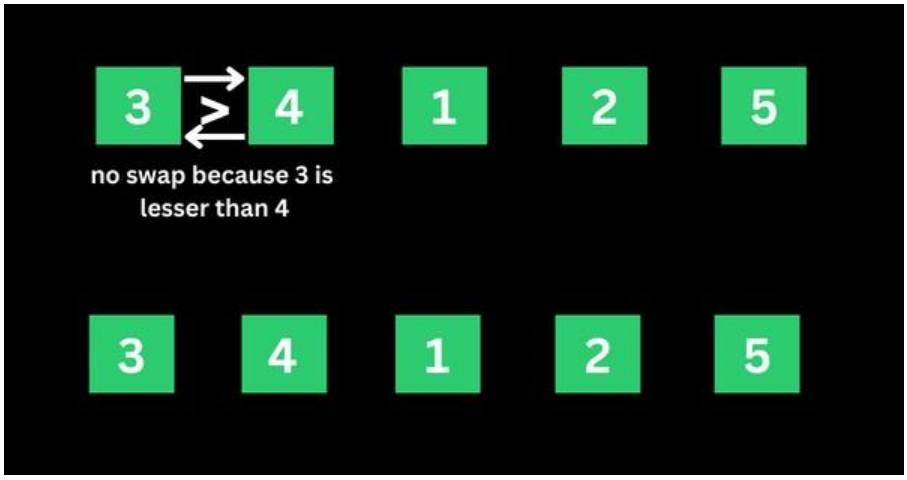
Step 4: The algorithm now has 3, 4, 1, 5, and 2 to compare. It compares the next two values, which are 5 and 2. 5 is greater than 2, so 5 takes the index of 2 and the numbers become 3, 4, 1, 2, and 5.



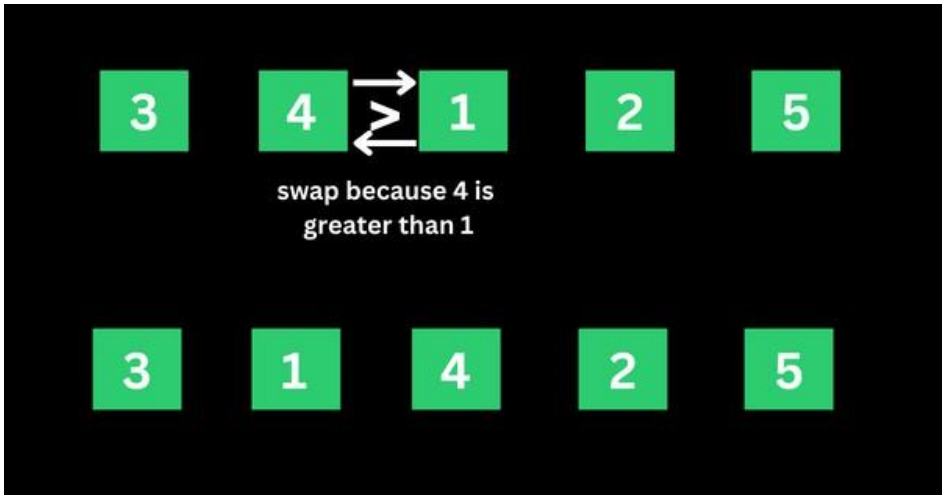
That's the first iteration. And the numbers are now arranged as 3, 4, 1, 2, and 5 – from the initial 5, 3, 4, 1, and 2. As you might realize, 5 should be the last number if the numbers are sorted in ascending order. This means the first iteration is really completed.

Second Iteration of the Sorting and the Rest

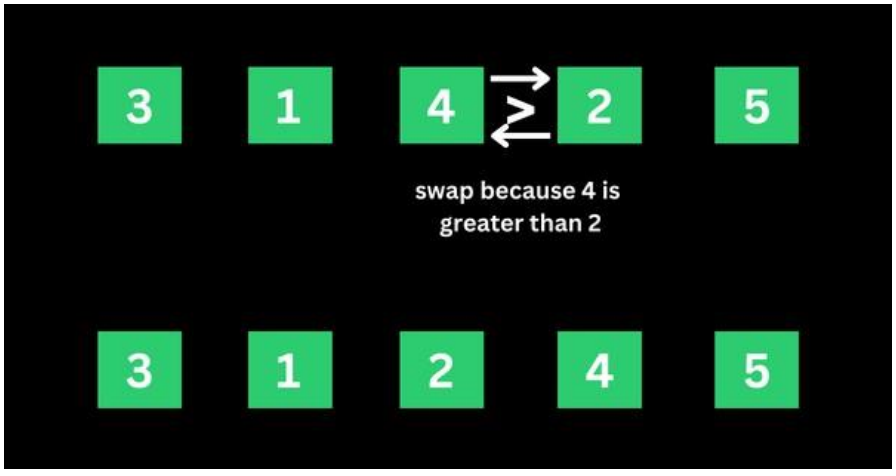
The algorithm starts the second iteration with the last result of 3, 4, 1, 2, and 5. This time around, 3 is smaller than 4, so no swapping happens. This means the numbers will remain the same.



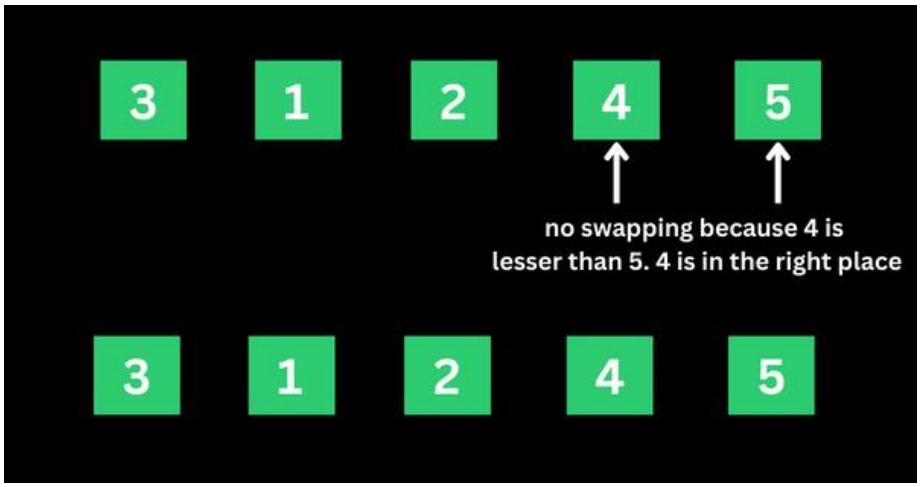
The algorithm proceeds to compare 4 and 1. 4 is greater than 1, so 4 is swapped for 1 and the numbers become 3, 1, 4, 2, and 5.



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 continuing with 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 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:

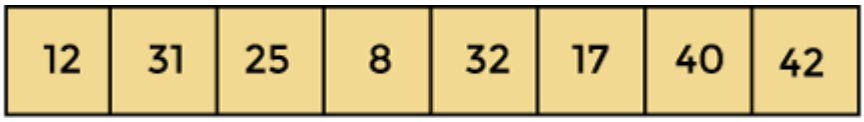
1. Divide the input array into two halves.
2. Recursively sort the left half of the array.
3. Recursively sort the right half of the array.
4. Merge the two sorted halves into a single sorted output array.

- 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.

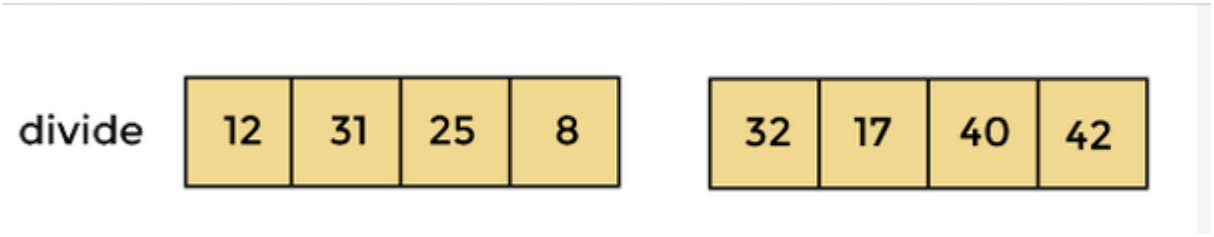
Example of Merge sort

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.

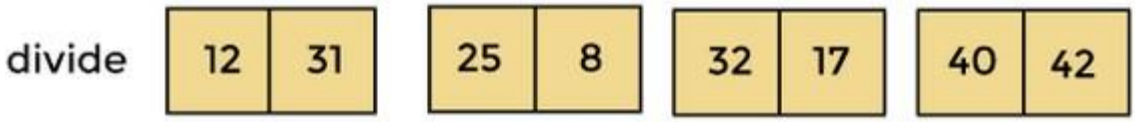
Let the elements of array are -



- 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.

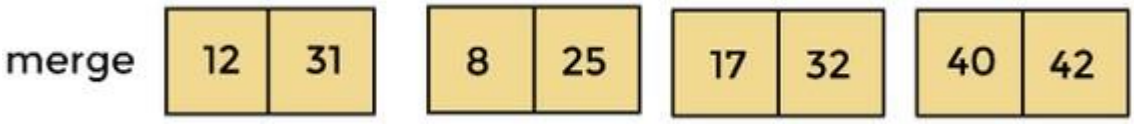


- Now, again divide these two arrays into halves. As they are of size 4, divide them into new arrays of size 2.

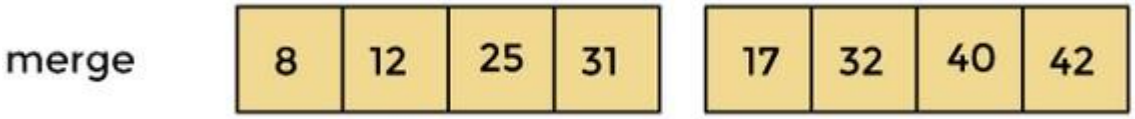


- Now, again divide these arrays to get the atomic value that cannot be further divided.

- 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.



- 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.



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

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 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 we have measured 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.

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 performProgram 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?

Department Of Computer Engineering

BE COMPUTER (2019 COURSE)

LAB MANUAL LP-V

Semester - VIII

410251: Deep Learning

Sr. no	Assignment
1	**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.
2	**Classification using Deep neural network** 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)** • 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 series analysis and prediction system using RNN.

Assignment No – 01

Title : Linear regression by using Deep Neural Network.

Objective:

1. To implement different deep learning models.
2. To illustrate the concepts of Artificial Intelligence | Machine Learning (AI/ML).

Problem Statement :

Implement Boston Housing Price prediction problem by Linear Regression using Deep Neural Network.

Software and Hardware Requirements :-

1. 64-bit open source operating system or its derivative
2. Programming languages - Python
3. Jupyter Notebook

Theory:

-**Deep Neural Network** → A Deep Neural Network (DNN) is an ANN with example hidden layers between the input and output layers. Similar to show ANNS, DNNS can model complex non-linear relationships.

-The main purpose of a neural network is to receive set of input, perform progre-asively complex calculations on them & give output to solve the real work problems like classification. We restrict ourselves to feed forward neural networks.

-We have an input and an output and a flow of Sequential data in deep network.

-Neural networks are widely used in supervised learning and reinforcement learning problems.

-These problems or networks are based on set of layers connected to each other.

-In Deep Learning, the number of hidden layers, mostly non-linear, can be large: say about 1000 layers.

-DL networks or models produce much better results than normal ML networks.

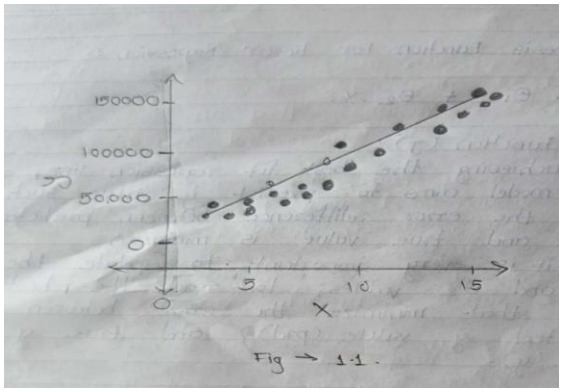
- We mostly use the gradient descent method for optimizing the network and minimising the loss function.

Linear Regression :

-Linear Regression is a machine learning algorithm based on supervised learning.

-It performs regression task.

-Regression models a target prediction value based on independent variables.
-It is mostly used for finding out the relationship between variables and forecasting.
-Different regression models differ based the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.
-There are many names for a regression's dependent variable, It may be called an outcome variable, criterion variable, endogeous variable or regressand.
-The independent veriable can be called exogenous variables, predictor variables or regressors.
-Linear regression is used in many Fields including finance, economics and psychology to understand and predict the behaviour of particular variable.
-linear regression performs the task to predict a dependent variable value(y) based on a given independent variable(x).
-Hence the name is Linear Regression.
- In fig 1.1. X (input) is the work experience and y(output) is the salary of a Person.



-The regression line is the best fit line for our model.
Hypothesis Function For Linear Regression :-
Y = ∅1+ ∅2 .X

Cost Function (J):
-By achieving the best fit regression line, the model aims to predict y value such that error difference between predicted value and true value is minimum.
-So, it is very important to update the ∅1 and ∅2 values, to reach the best value that minimize the error between predicted y value (prd) and true y value (y)

minimise $\frac{1}{n}\sum_{i=0}^{n}(predi - yi)2$
j = $\frac{1}{n}\sum_{i=0}^{n}(predi - yi)2$

-Cost Function (T) of linear Regression is the Root Mean Squared Error (RMSE) between predicted y value (pred) and true value (y).

Gradient Descent :-

-To update Ø1 and Ø2 values in order to reduce cost Function (minimizing RMSE value) and acheiving the best-fit line the model use Gradient Descent.
-The Ideal is to start with random Ø1 and Ø2 values and then iteratively, updating the values, reaching minimum cost.

-Now let's understand some libraries of Python :

1) Pandas :-

-Pandas is an open-source library that is made mainly for working with rational labeled data both easily and intuitively.
-it provides various data structures and operations for manipulating numerical data and time series.
-This library is built on top of the Numpy library.
-Pandas is fist and it has high performance and productivity for users.
-This module is generally imported as:
import pandas as pd
Here, pd is referred to as an alias to the Pandas.

2) NumPy :-

•numpy stands for Numerical Python is a library consisting of multidimensional array objects and a collection of runtime for processing the array.
-using Numpy, mathematical and logical operation on array can be performed.
-This module is generally imported as: Import NumPy as np

3) Matplotlib :-

-Matplotlib is an amazing visualization library In python for 2D plots of arrays.
-Matplotlib is a multi-platform data visualization library built on Numpy, arrays and designed to with the broader scipy stack.
-it was introduced by John Hunter in the year 2002 .
-Matplotlib consists of several plots like line, bar, scatter, histogram, etc.
-This module is generally imported as:
from matplotlib import pyplot as plt or import matplotlib pyplot as plt.

Conclusion:

Hence, we implement the Boston housing price prediction problem successfully by using linear regression using Deep Neural Network.

Assignment No – 02

Title : Classification using Deep neural network.

Objective:

1. To implement different deep learning models.
2. To illustrate the concepts of Artificial Intelligence | Machine Learning (AI/ML).

Problem Statement:

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

Software and Hardware Requirements:-

1. 64-bit open source operating system or its derivative
2. Programming languages - Python
3. Jupyter Notebook

Theory:

What is Classification?

Classification problem involves predicting if something belongs to one class or not. In other words, while doing it we try to see something is one thing or another.

Types of Classification

- Suppose that you want to predict if a person has diabetes or not. İf you are facing this kind of situation, there are two possibilities, right? That is called **Binary Classification.**
- Suppose that you want to identify if a photo is of a toy, a person, or a cat, right? this is called **Multi-class Classification** because there are more than two options.
- Suppose you want to decide that which categories should be assigned to an article. If so, it is called **Multi-label Classification**, because one article could have more than one category assigned.

What is Binary Classification?

In machine learning, binary classification is a supervised learning algorithm that categorizes new observations into one of **two** classes.
The following are a few binary classification applications, where the 0 and 1 columns are two possible classes for each observation:

Application	Observation	0	1
Medical Diagnosis	Patient	Healthy	Diseased
Email Analysis	Email	Not Spam	Spam
Financial Data Analysis	Transaction	Not Fraud	Fraud
Marketing	Website visitor	Won't Buy	Will Buy
Image Classification	Image	Hotdog	Not Hotdog

Example:

In a medical diagnosis, a binary classifier for a specific disease could take a patient's symptoms as input features and predict whether the patient is healthy or has the disease. The possible outcomes of the diagnosis are **positive** and **negative**.

Evaluation of binary classifiers

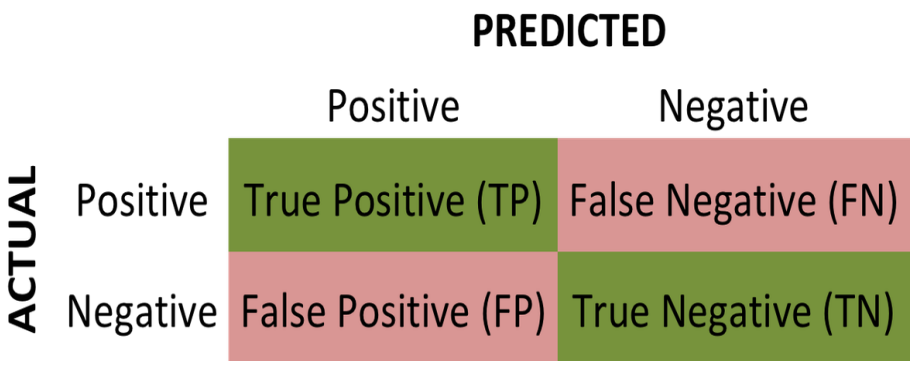
If the model successfully predicts the patients as positive, this case is called *True Positive (TP)*. If the model successfully predicts patients as negative, this is called *True Negative (TN)*. The binary classifier may misdiagnose some patients as well. If a diseased patient is classified as healthy by a negative test result, this error is called *False Negative (FN)*. Similarly, If a healthy patient is classified as diseased by a positive test result, this error is called *False Positive(FP)*.
We can evaluate a binary classifier based on the following parameters:

- True Positive (TP): The patient is diseased and the model predicts "diseased"

- False Positive (FP): The patient is healthy but the model predicts "diseased"
- True Negative (TN): The patient is healthy and the model predicts "healthy"
- False Negative (FN): The patient is diseased and the model predicts "healthy"
    - After obtaining these values, we can compute the **accuracy score** of the binary classifier as follows:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

    - The following is a *confusion matrix*, which represents the above parameters:



In machine learning, many methods utilize binary classification. The most common are:

- Support Vector Machines
- Naive Bayes
- Nearest Neighbour
- Decision Trees
- Logistic Regression
- Neural Networks

Assignment No – 03

Title : Convolutional neural network (CNN)

Objective:

3. To implement different deep learning models.
4. To illustrate the concepts of Artificial Intelligence | Machine Learning (AI/ML).

Problem Statement :

Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

Software and Hardware Requirements :-

4. 64-bit open source operating system or its derivative
5. Programming languages - Python
6. Jupyter Notebook

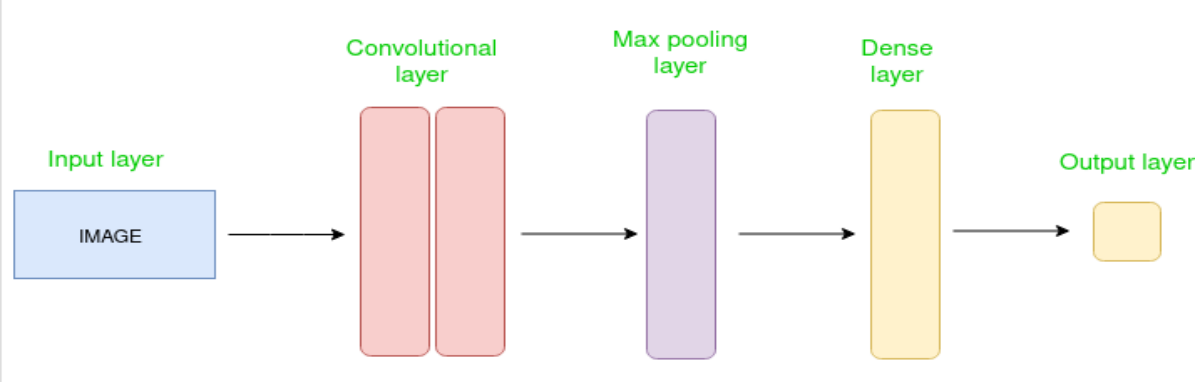
Theory:

A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.

Convolutional Neural Network (CNN) is the extended version of artificial neural networks (ANN) which is predominantly used to extract the feature from the grid-like matrix dataset. For example visual datasets like images or videos where data patterns play an extensive role.

CNN architecture

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers.

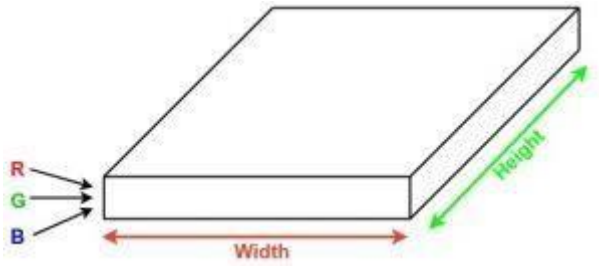


Simple CNN architecture

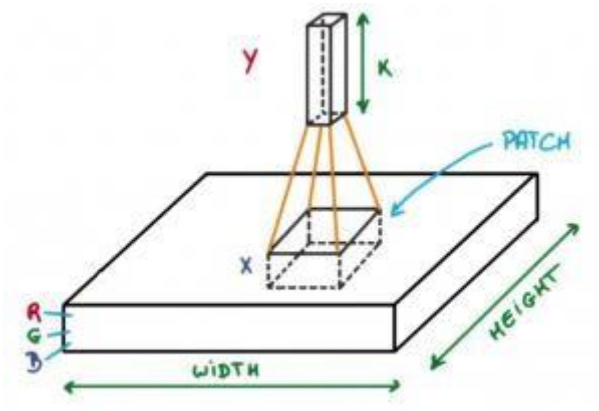
The Convolutional layer applies filters to the input image to extract features, the Pooling layer down samples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

How Convolutional Layers works

Convolution Neural Networks or covnets are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (i.e the channel as images generally have red, green, and blue channels).



Now imagine taking a small patch of this image and running a small neural network, called a filter or kernel on it, with say, K outputs and representing them vertically. Now slide that neural network across the whole image, as a result, we will get another image with different widths, heights, and depths. Instead of just R, G, and B channels now we have more channels but lesser width and height. This operation is called Convolution. If the patch size is the same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.



Types of layers: datasets

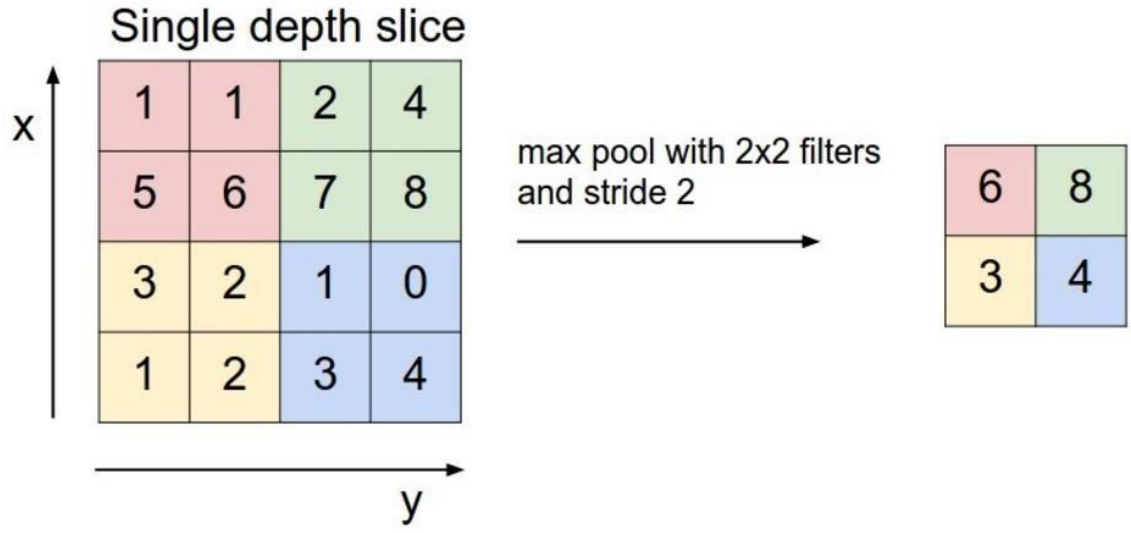
Let's take an example by running a covnets on of image of dimension 32 x 32 x 3.

Input Layers: It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.

Convolutional Layers: This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred ad feature maps. Suppose we use a total of 12 filters for this layer we'll get an output volume of dimension 32 x 32 x 12.

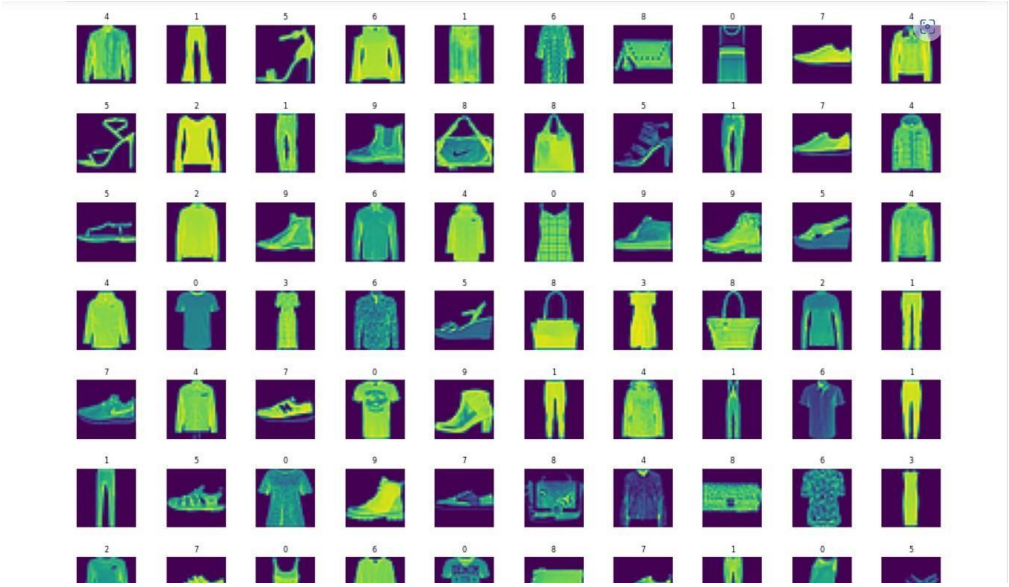
Activation Layer: By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are RELU: max(0, x), Tanh, Leaky RELU, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12.

Pooling layer: This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are max pooling and average pooling. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.



Flattening: The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.

Fully Connected Layers: It takes the input from the previous layer and computes the final classification or regression task.



Output Layer: The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

Conclusion :

Convolutional neural networks are multi-layer neural networks that are really good at getting the features out of data. They work well with images and they don't need a lot of pre-processing. Using convolutions and pooling to reduce an image to its basic features, you can identify images correctly.

Assignment No – 03

Title : Recurrent neural network (RNN)

Objective:

1. To implement different deep learning models.
2. To illustrate the concepts of Artificial Intelligence | Machine Learning (AI/ML).

Problem Statement :

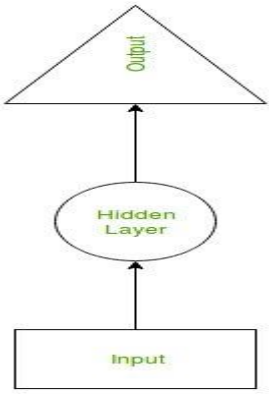
Use the Google stock prices dataset and design a time Series analysis and prediction system using RNN.

Software and Hardware Requirements :-

1. 64-bit open source operating system or its derivative
2. Programming languages - Python.
3. Jupyter Notebook

Theory:

Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step are fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is Hidden state, which remembers some information about a sequence.



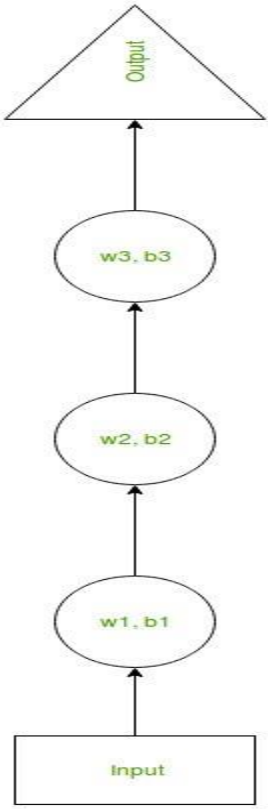
RNN have a “memory” which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task

on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

How RNN works

The working of an RNN can be understood with the help of the below example:

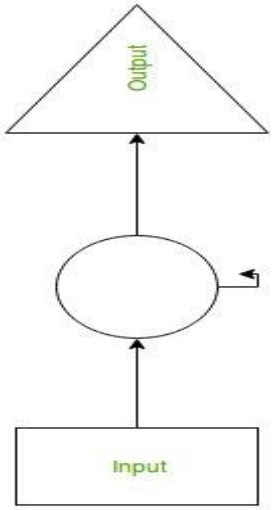
Example: Suppose there is a deeper network with one input layer, three hidden layers, and one output layer. Then like other neural networks, each hidden layer will have its own set of weights and biases, let's say, for hidden layer 1 the weights and biases are (w1, b1), (w2, b2) for the second hidden layer, and (w3, b3) for the third hidden layer. This means that each of these layers is independent of the other, i.e. they do not memorize the previous outputs.



Now the RNN will do the following:

RNN converts the independent activations into dependent activations by providing the same weights and biases to all the layers, thus reducing the complexity of increasing parameters and memorizing each previous output by giving each output as input to the next hidden layer.

Hence these three layers can be joined together such that the weights and bias of all the hidden layers are the same, in a single recurrent layer.



The formula for calculating current state:

$$h_t = f(h_{t-1}, x_t)$$

where:
ht -> current state
ht-1 -> previous state
xt -> input state

Formula for applying Activation function(tanh):

$$h_t = \tanh\left(W_{hh}h_{t-1} + W_{xh}x_t\right)$$

where:
whh -> weight at recurrent neuron
wxh -> weight at input neuron

The formula for calculating output:

$$y_t = W_{hy}h_t$$

Yt -> output
Why -> weight at output layer

Training through RNN

1. A single-time step of the input is provided to the network.
2. Then calculate its current state using a set of current input and the previous state.
3. The current ht becomes ht-1 for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.

5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained.

Conclusion :

Recurrent Neural Networks are a powerful tool for handling sequential data. They are widely used in a variety of applications and have proven to be very effective in tasks such as natural language processing, speech recognition, and time-series forecasting. While RNNs can be challenging to train, they offer a lot of flexibility in modeling sequential data and are an important part of the modern AI toolkit.