Sandip Foundation's
SANDIP INSTITUTE OF TECHNOLOGY & RESEARCH CENTRE, NASHIK
DEPARTMENT OF COMPUTER ENGINEERING

Assignment No- 01

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

Problem Statement:-

Write a program to implement parallel BFS and DFS

Objective:-

1. To understand parallel BFS
2. To implement parallel DFS
3. To study about OpenMP

Outcome:-

After completion of this assignment students will be able to:

- Understand the concept of parallel BFS
- Understand about Open MP

Prerequisite-

- **Hardware Requirement-**

  M/C Lenovo Think center M700 Ci3,6100,6th Gen. H81, 4GB RAM ,500GB HDD.

- **Software Requirement-**Open MP

Introduction:-

Parallel BFS

To design and implement parallel breadth first search, you will need to divide the graph into smaller sub-graphs and assign each sub-graph to a different processor or thread Each processor or thread will then perform a breadth first search on its assigned sub-graph concurrently with the other processors or threads .Two methods : Vertex by Vertex OR Level By Level

THEORY:-

Parallel DFS: In this implementation, the **parallel_dfs** function takes in a graph represented as an adjacency list, where each element in the list is a vector of neighboring vertices, and a starting vertex. The **dfs** function uses a stack to keep track of the vertices to visit, and a boolean visited array to keep track of which vertices have been visited. The **#pragma omp parallel** directive creates a parallel region and the **#pragma omp single** directive creates a single execution context within that region.

- **ALGORITHM:**

```
Procedure    Parallel-Breadth-First-Search-Vertex(ALM, EM, U)
begin
    mark every vertex "unvisited"
    v ⇐ start vertex
    mark v "visited"
    instruct processor(i) where    1 ≤ i ≤ k
        for j = 1 to k do
            if (k * (j − 1) + i)≤EM(v)
            then delete v from U(ALM(v, k * (j − 1) + i))
            endif
        endfor
    end-instruction
    initialize queue with v
    while queue is not empty do
        begin
        v ⇐ first vertex from the queue
        for each w∈ U(v) do
            begin
            mark w "visited"
            instruct processor (i) where    1 ≤ i ≤ k
                for   j = 1 to k do
                    if (k * (j − 1) + i) ≤ EM(w)

                    then delete w from U(ALM(w, k * (j − 1) + i))
                    endif
                endfor
            end-instruction
            add w to queue
            end
        endfor
    endwhile
end
```

- **PROGRAM:**

Program Sample :

```
#include <iostream>
#include <vector>
#include <queue>
#include <omp.h>

using namespace std;

void bfs(vector<vector<int>>& graph, int start, vector<bool>&
visited) {
   queue<int> q;
   q.push(start);
   visited[start] = true;
   #pragma omp parallel
   {
      #pragma omp single
      {
         while (!q.empty()) {
            int vertex = q.front();
            q.pop();
```

–

```
#pragma omp task firstprivate(vertex)
            {
               for (int neighbor : graph[vertex]) {
                  if (!visited[neighbor]) {
                     q.push(neighbor);
                     visited[neighbor] = true;
                     #pragma omp  task
                     bfs(graph, neighbor, visited);
                  }
               }
            }
         }
      }
   }
}

void parallel_bfs(vector<vector<int>>& graph, int start) {
   vector<bool> visited(graph.size(), false);
   bfs(graph, start, visited);
}
```

```
#pragma omp task firstprivate(vertex)
            {
                for (int neighbor : graph[vertex]) {
                    if (!visited[neighbor]) {
                        s.push(neighbor);
                        visited[neighbor] = true;
                        #pragma omp  task
                        dfs(graph, neighbor, visited);
                    }
                }
            }
        }
    }
}
void parallel_dfs(vector<vector<int>>& graph, int start) {
    vector<bool> visited(graph.size(), false);
    dfs(graph, start, visited);
}
```

Program

```
#include <iostream>
#include <vector>
#include <stack>
#include <omp.h>

using namespace std;

void dfs(vector<vector<int>>& graph, int start,
vector<bool>& visited) {
    stack<int> s;
    s.push(start);
    visited[start] = true;
 #pragma omp parallel
    {
        #pragma omp single
        {
            while (!s.empty()) {
                int vertex = s.top();
                s.pop();
```

INPUT : To give input as Graph

OUTPUT: After applying algorithm we get output

CONCLUSION: Implemented Parallel Breadth First Search and Depth First Search

REFERENCES:

https://en.wikipedia.org/wiki/Dynamic-link_library
https://en.wikipedia.org/wiki/Visual_Basic https://www.google.co.in/search?q=dynamic+link+library+architecture&dcr=0&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjqubTAuJvZAhWHQ48KHRZbD7sQ_AUICigB&biw=136 6&bih=651#imgrc=LU8YqljE8-afxM
https://msdn.microsoft.com/en-us/library/9yd93633.aspx

Oral Questions: [Write short answer]

1. What Is parallel BFS
2. What Is parallel DFS

Sandip Foundation's
SANDIP INSTITUTE OF TECHNOLOGY & RESEARCH CENTRE, NASHIK
DEPARTMENT OF COMPUTER ENGINEERING

Assignment No- 02

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

OBJECTIVES:

1. To implement Parallel Bubble Sort
2. To implement Merge sort

OUTCOMES:

After completion of this assignment students will be able to:

1. Understand the concept of parallel bubble sort
2. Understand the concept of parallel merge sort

Prerequisite-

- **Hardware Requirement-**

  M/C Lenovo Think center M700 Ci3,6100,6th Gen. H81, 4GB RAM ,500GB HDD

- **Software Requirement-**

  open MP

THEORY:

- The **#pragma omp parallel for** directive tells the compiler to create a team of threads to execute the for loop within the block in parallel
- Each thread will work on a different iteration of the loop, in this case on comparing and swapping the elements of the array.
- The **bubbleSort** function takes in an array, and it sorts it using the bubble sort algorithm. The outer loop iterates from 0 to n-2 and the inner loop iterates from 0 to n-i-1, where i is the index of the outer loop. The inner loop compares the current element with the next element, and if the current element is greater than the next element, they are swapped
- The **main** function creates a sample array and calls the **bubbleSort** function to sort it. The sorted array is then printed.
- The two **#pragma omp parallel for** inside while loop, one for even indexes and one for odd indexes, allows each thread to sort the even and odd indexed elements simultaneously and prevent the dependency

Algorithm:

Parallel Odd-Even Transposition

```
1.  procedure ODD-EVEN_PAR(n)
2.  begin
3.      id := process's label
4.      for i := 1 to n do
5.      begin
6.          if i is odd then
7.              if id is odd then
8.                  compare-exchange_min(id + 1);
9.              else
10.                 compare-exchange_max(id − 1);
11.         if i is even then
12.             if id is even then
13.                 compare-exchange_min(id + 1);
14.             else
15.                 compare-exchange_max(id − 1);
16.     end for
17. end ODD-EVEN_PAR
```

In this implementation, the bubble_sort_odd_even function takes in an array and sorts it using the odd-even transposition algorithm. The outer while loop continues until the array is sorted. Inside the loop, the #pragma omp parallel for directive creates a parallel region and divides the loop iterations among the available threads. Each thread performs the swap operation in parallel, improving the performance of the algorithm.

Algorithm: Odd-Even(A,B,S)

```
begin
if A and B are of length 1
    then
Merge A and B using one Compare-and-Exchange operation
    else
    begin
    compute S_odd and S_even In Parallel do
    S_odd = Merge(A_odd,B_odd)
    S_even = Merge(A_even,B_even)
    S_odd- even = Join(S_odd,S_even)
    end
endif
end
```

Program:

```
#include <iostream>
#include <vector>
#include <omp.h>

using namespace std;

void merge(vector<int>& arr, int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    vector<int> L(n1), R(n2);

    for (i = 0; i < n1; i++) {
        L[i] = arr[l + i];
    }
    for (j = 0; j < n2; j++) {
        R[j] = arr[m + 1 + j];
    }
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }
```

```
void merge_sort(vector<int>& arr, int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        #pragma omp task
        merge_sort(arr, l, m);
        #pragma omp task
        merge_sort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void parallel_merge_sort(vector<int>& arr) {
    #pragma omp parallel
    {
        #pragma omp single
        merge_sort(arr, 0, arr.size() - 1);
    }
}
```

FAQ's

1. What is Parallel Bubble Sort
2. What is Parallel Merge Sort

REFERENCES:

https://en.wikipedia.org/wiki/Lex_(software)
http://epaperpress.com/lexandyacc/prl.html
https://www.ibm.com/developerworks/library/l-lexyac/index.html

Conclusion: Implemented Parallel Bubble Sort and Merge sort

Assignment No- 03

TITLE: Implement Min, Max, Sum and Average operations using Parallel Reduction

PROBLEM STATEMENT:
i) Min, Max, Sum and Average operations using Parallel Reduction

OBJECTIVES:

1. To understand Min, Max, Sum and Average operations using Parallel Reduction

OUTCOMES:

After completion of this assignment students will be able to:
1. Understand the concept of Min, Max, Sum and Average operations using Parallel Reduction

- **Hardware Requirement-**

  M/C Lenovo Think center M700 Ci3,6100,6th Gen. H81, 4GB RAM ,500GB HDD

- **Software Requirement-**

  Open MP

THEORY:

- The **min_reduction** function finds the minimum value in the input array using the **#pragma omp parallel for reduction(min: min_value)** directive, which creates a parallel region and divides the loop iterations among the available threads. Each thread performs the comparison operation in parallel and updates the **min_value** variable if a smaller value is found
- Similarly, the **max_reduction** function finds the maximum value in the array, **sum_reduction** function finds the sum of the elements of array and **average_reduction** function finds the average of the elements of array by dividing the sum by the size of the array
- The **reduction** clause is used to combine the results of multiple threads into a single value, which is then returned by the function. The **min** and **max** operators are used for the **min_reduction** and **max_reduction** functions, respectively, and the + operator is used for the **sum_reduction**
- **And average_reduction functions. In the main function, it creates a vector and calls the functions min_reduction, max_reduction, sum_reduction, and average_reduction to compute the values of min, max, sum and average respectively**

```
#include <iostream>
#include <vector>
#include <omp.h>

using namespace std;
void min_reduction(vector<int>& arr) {
    int min_value = INT_MAX;
    #pragma omp parallel for reduction(min: min_value)
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] < min_value) {
            min_value = arr[i];
        }
    }
    cout << "Minimum value: " << min_value << endl;
}
void max_reduction(vector<int>& arr) {
    int max_value = INT_MIN;
    #pragma omp parallel for reduction(max: max_value)
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] > max_value) {
            max_value = arr[i];
        }
    }
    cout << "Maximum value: " << max_value << endl;
}

void sum_reduction(vector<int>& arr) {
```

```
    int sum = 0;
    #pragma omp parallel for reduction(+: sum)
    for (int i = 0; i < arr.size(); i++) {
        sum += arr[i];
    }
    cout << "Sum: " << sum << endl;
}

void average_reduction(vector<int>& arr) {
    int sum = 0;
    #pragma omp parallel for reduction(+: sum)
    for (int i = 0; i < arr.size(); i++) {
        sum += arr[i];
    }
    cout << "Average: " << (double)sum / arr.size() << endl;
}

int main() {
    vector<int> arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};

    min_reduction(arr);
    max_reduction(arr);
    sum_reduction(arr);
    average_reduction(arr);
}
```

INPUT : Enter number of characters, words and lines.
OUTPUT: It count the total number of characters, words and lines.

FAQ's

1. What are the steps to calculate min,max using parallel reduction

REFERENCES:

https://en.wikipedia.org/wiki/Lex_(software)
http://epaperpress.com/lexandyacc/prl.html
https://www.ibm.com/developerworks/library/l-lexyac/index.html

Conclusion: Implemented Min, Max, Sum and Average operations using Parallel Reduction

Assignment No- 04

Title:Write a CUDA Program for :

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

Problem Statement:-

To implement parallel Addition of two large vectors and matrix multiplication using CUDA C

Objective:

Write a program to implement parallel Addition of two large vectors and matrix multiplication using CUDA C

Outcome:-

After completion of this assignment students will be able to:

- Understand the concept of parallel addition and matrix multiplication
- Understand about CUDA C

Theory:

In this program, the `addVectors` kernel takes in the two input vectors `A` and `B`, the output vector `C`, and the size of the vectors `n`. The kernel uses the `blockIdx.x` and `threadIdx.x` variables to calculate the index `i` of the current thread. If the index is less than `n`, the kernel performs the addition operation `C[i] = A[i] + B[i]`.

.In the `main` function, the program first allocates memory for the input and output vectors on the host and initializes them. Then it allocates memory for the vectors on the device and copies the data from the host to the device using `cudaMemcpy

• In this program, the `matmul` kernel takes in the two input matrices `A` and `B`, the output matrix `C`, and the size of the matrices `N`. The kernel uses the `blockIdx.x`, `blockIdx.y`, `threadIdx.x`, and `threadIdx.y` variables to calculate the indices of the current thread. If the indices are less than `N`, the kernel performs the matrix multiplication operation `Pvalue += A[Row*N+k] * B[k*N+Col]` and store the Pvalue in

`C[Row*N+Col]`.

```
#include <cuda_runtime.h>
#include <iostream>

__global__ void matmul(int* A, int* B, int* C, int N) {
    int Row = blockIdx.y*blockDim.y+threadIdx.y;
    int Col = blockIdx.x*blockDim.x+threadIdx.x;

    if (Row < N && Col < N) {
        int Pvalue = 0;
        for (int k = 0; k < N; k++) {
            Pvalue += A[Row*N+k] * B[k*N+Col];
        }
        C[Row*N+Col] = Pvalue;
    }
}

int main() {
    int N = 512;
    int size = N * N * sizeof(int);

    int* A, * B, * C;
    int* dev_A, * dev_B, * dev_C;

    cudaMallocHost(&A, size);
    cudaMallocHost(&B, size);
    cudaMallocHost(&C, size);
```

```
cudaMalloc(&dev_A, size);
    cudaMalloc(&dev_B, size);
    cudaMalloc(&dev_C, size);

// Initialize matrices A and B
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            A[i*N+j] = i*N+j;
            B[i*N+j] = j*N+i;
        }
    }

    cudaMemcpy(dev_A, A, size,
cudaMemcpyHostToDevice);
    cudaMemcpy(dev_B, B, size,
cudaMemcpyHostToDevice);

    dim3 dimBlock(16, 16);
    dim3 dimGrid(N/dimBlock.x, N/dimBlock.y);
    matmul<<<dimGrid, dimBlock>>>(dev_A, dev_B,
dev_C, N);
```

```
// Print the result
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        std::cout << C[i*N+j] << " ";
    }
    std::cout << std::endl;
}

// Free memory
cudaFree(dev_A);
cudaFree(dev_B);
cudaFree(dev_C);
cudaFreeHost(A);
cudaFreeHost(B);
cudaFreeHost(C);

return 0;
```

Conclusion: Implemented Addition of two large vectors and matrix multiplication using CUDA C

Deep Learning

Assignment:01

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

Objective:-

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

Program Outcome:-

- Understand Linear regression using Deep Neural network.

Prerequisite-

- **Hardware Requirement-**
  - M/C Lenovo Think center M700 Ci3,6100,6th Gen. H81, 4GB RAM ,500GB HDD
- **Software Requirement-**
  - Data SET

Theory:-

Housing prices are an important reflection of the economy, and housing price ranges are of great interest for both buyers and sellers. Ask a home buyer to describe their dream house, and they probably won't begin with the height of the basement ceiling or the proximity to an east-west railroad. But this playground competition's data-set proves that much more influences price negotiations than the number of bedrooms or a white-picket fence

Data Set:

About the Dataset

Housing prices are an important reflection of the economy, and housing price ranges are of great interest for both buyers and sellers. In this project, house prices will be predicted given explanatory variables that cover many aspects of residential houses. The goal of this project is to create a regression model that is able to accurately estimate the price of the house given the features.

In this dataset made for predicting the Boston House Price Prediction. Here I just show the all of the feature for each house separately. Such as Number of Rooms, Crime rate of the House's Area and so on. We'll show in the upcoming part.

Data Overview

1. **CRIM** per capital crime rate by town
2. **ZN** proportion of residential land zoned for lots over 25,000 sq.ft.

3. **INDUS** proportion of non-retail business acres per town

4. **CHAS** Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)

5. **NOX** nitric oxides concentration (parts per 10 million)

6. **RM** average number of rooms per dwelling

7. **AGE** proportion of owner-occupied units built prior to 1940

8. **DIS** weighted distances to five Boston employment centers

9. **RAD** index of accessibility to radial highways

10. **TAX** full-value property-tax rate per 10,000 USD

11. **PTRATIO** pupil-teacher ratio by town

12. **Black** 1000(Bk — 0.63)² where Bk is the proportion of blacks by town

13. **LSTAT** % lower status of the population

In this dataset made for predicting the Boston House Price Prediction. Here I just show the all of the feature for each house separately. Such as Number of Rooms, Crime rate of the House's Area and so on. We'll show in the upcoming part.

ALGORITHM:

The major aim of in this project is to predict the house prices based on the features using some of the regression techniques and algorithms.

1. Linear Regression

2. Random Forest Regressor

Data Collection

I got the Dataset from Kaggle. This Dataset consist several features such as Number of Rooms, Crime Rate, and Tax and so on. Let's know about how to

INPUT :

To give the data set

OUTPUT:

Implement the algorithm

CONCLUSION:Implemented Boston housing price prediction problem by Linear regression using Deep Neural network.

REFERENCES:

https://en.wikipedia.org/wiki/Lex_(software)
http://epaperpress.com/lexandyacc/prl.html
https://www.ibm.com/developerworks/library/l-lexyac/index.html **http://epaperpress.com/lexandyacc/pry2.html**

Oral Questions: [Write short answer]

1. What is deep neural network

Assignment:02

1. 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

Problem Statement:

- **Binary classification using Deep Neural Networks**

2. Outcomes:

After completion of this assignment students will be able to:

- Understand Binary classification using Deep Neural Networks

3. Software Requirements:

Data Set

4. Hardware Requirement:

- M/C Lenovo Think center M700 Ci3,6100,6th Gen. H81, 4GB RAM ,500GB HDD

5. Theory Concepts:

The IMDB dataset is a set of 50,000 highly polarized reviews from the Internet Movie Database. They are split into 25000 reviews each for training and testing. Each set contains an equal number (50%) of positive and negative reviews.

The IMDB dataset comes packaged with Keras. It consists of reviews and their corresponding labels (0 for *negative* and 1 for *positive* review). The reviews are a sequence of words. They come preprocessed as a sequence of integers, where each integer stands for a specific word in the dictionary.

The IMDB dataset can be loaded directly from Keras and will usually download about 80 MB on your machine

Loading the Data

Let's load the prepackaged data from Keras. We will only include 10,000 of the most frequently occurring words.

```
from keras.datasets import imdb
 # Load the data, keeping only 10,000 of the most frequently occuring words (train_data, train_labels),
(test_data, test_labels) = imdb.load_data(num_words = 10000)
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/text-datasets/imdb.npz
17465344/17464789 [==============================] - 1s 0us/step


# Since we restricted ourselves to the top 10000 frequent words, no word index should exceed 10000
# we'll verify this below
# Here is a list of maximum indexes in every review --- we search the maximum index in this list of max
indexes print(type([max(sequence) for sequence in train_data]))
# Find the maximum of all max indexes max([max(sequence) for sequence in train_data])
<class 'list'>
9999
```

Packages USED:

```
from keras import models
from keras import layers model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
 model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

References:

https://en.wikipedia.org/wiki/Lex_(software)
http://epaperpress.com/lexandyacc/prl.html
https://www.ibm.com/developerworks/library/l - lexyac/index.html
http://epaperpress.com/lexandyacc/pry2.html

CONCLUSION:Implemented 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

Sandip Foundation's
SANDIP INSTITUTE OF TECHNOLOGY & RESEARCH CENTRE, NASHIK
DEPARTMENT OF COMPUTER ENGINEERING

Assignment:03

Title: Use any dataset of plant disease and design a plant disease detection system using CNN.

Objectives:

Design a plant disease detection system using CNN.

Software Requirements:

- Data set

Hardware Requirement:

M/C Lenovo Think center M700 Ci3,6100,6th Gen. H81, 4GB RAM ,500GB HDD

Theory:

.

Machine learning, Deep learning, and Artificial intelligence are the Future. We use these technologies in almost every field. In the Farming sector, we can also use this technology for the Preparation of soil, adding fertilizers, sowing of seed, Irrigation, weed protection, harvesting, disease prediction, etc

We are using Deep Learning for Plant disease detection based on images of a leaf of a plant. We are using deep learning for this task because here we are working with image data. Deep learning has a Convolution neural network that is used to find features from the leaf of the plant.

Data Set

In this data-set, 39 different classes of plant leaf and background images are available. The data-set containing 61,486 images. We used six different augmentation techniques for increasing the data-set size. The techniques are image flipping, Gamma correction, noise injection, PCA color augmentation, rotation, and Scaling.
There is a total of 39 Classes that we have to predict using the CNN Model.

Program:

```
 General
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt# Torch
import torch
from torchvision import datasets, transforms # datsets  , transforms
from torch.utils.data.sampler import SubsetRandomSampler
import torch.nn as nn
import torch.nn.functional as F
from datetime import datetime


split = int(np.floor(0.85 * len(dataset)))  # train_sizevalidation = int(np.floor(0.70 *
split))  # validationnp.random.shuffle(indices)train_indices, validation_indices,
test_indices = (
```

```
    indices[:validation],
    indices[validation:split],
    indices[split:],
)
```

- **train_sampler = SubsetRandomSampler(train_indices)**

```
validation_sampler = SubsetRandomSampler(validation_indices)
test_sampler = SubsetRandomSampler(test_indices)
```

SubsetRandomSampler is used to sample our data. Here we are creating an object of SubsetRandomSampler Object and later we will use this sampler in train data loader and test data loader.

```
batch_size = 64train_loader = torch.utils.data.DataLoader(
    dataset, batch_size=batch_size, sampler=train_sampler
)test_loader = torch.utils.data.DataLoader(
    dataset, batch_size=batch_size, sampler=test_sampler
)validation_loader = torch.utils.data.DataLoader(
    dataset, batch_size=batch_size, sampler=validation_sampler
)
```

Working:

If you know the working of CNN then You get my point about what I do in this project. This section is for those who don't understand. Basically, First we Resize every image into 224 x 224. After that this image feed into the Convolutional Neural Network. We feed color image so it has 3 channels RGB. First conv layer we apply 32 filter size or output channels. That means 32 different filters apply to the images and try to find features and after that using 32 features, we create a features map that has channels 32. So from 3 x 224 x 224 it will become 32 x 222 x 222. After that we are applying ReLU activation function to remove non linearity and after that we are applying Batch Normalization to normalize the weights of the neuron. After that this image we feed to the max pool layer which takes only the most relevant features only so that why we get the output image in shape 32 x 112 x 112. After that, we feed this image to the next convolutional layer and its process is the same as mentioned above. At last, we flatten the final max pool layer output and feed to the next linear layer which is also called a fully connected layer, and finally, as a final layer, we predict 39 categories. So as a model output we get tensor 1x39 size. And from that tensor, we take an index of the maximum value in the tensor. That particular index is our main prediction. That's how everything works

Conclusion: Use any dataset of plant disease and design a plant disease detection system using CNN.

Assignment:04

Title:Write a CUDA Program for :

3. Addition of two large vectors
4. Matrix Multiplication using CUDA C

Problem Statement:-

To implement parallel Addition of two large vectors and matrix multiplication using CUDA C

Objective:

Write a program to implement parallel Addition of two large vectors and matrix multiplication using CUDA C

Outcome:-

After completion of this assignment students will be able to:

- Understand the concept of parallel addition and matrix multiplication
- Understand about CUDA C

Theory:

In this program, the `addVectors` kernel takes in the two input vectors `A` and `B`, the output vector `C`, and the size of the vectors `n`. The kernel uses the `blockIdx.x` and `threadIdx.x` variables to calculate the index `i` of the current thread. If the index is less than `n`, the kernel performs the addition operation `C[i] = A[i] + B[i]`.

.In the `main` function, the program first allocates memory for the input and output vectors on the host and initializes them. Then it allocates memory for the vectors on the device and copies the data from the host to the device using `cudaMemcpy

• In this program, the `matmul` kernel takes in the two input matrices `A` and `B`, the output matrix `C`, and the size of the matrices `N`. The kernel uses the `blockIdx.x`, `blockIdx.y`, `threadIdx.x`, and `threadIdx.y` variables to calculate the indices of the current thread. If the indices are less than `N`, the kernel performs the matrix multiplication operation `Pvalue += A[Row*N+k] * B[k*N+Col]` and store the Pvalue in

`C[Row*N+Col]`.

Program:

```
#include <cuda_runtime.h>
#include <iostream>

__global__ void matmul(int* A, int* B, int* C, int N) {
    int Row = blockIdx.y*blockDim.y+threadIdx.y;
    int Col = blockIdx.x*blockDim.x+threadIdx.x;

    if (Row < N && Col < N) {
        int Pvalue = 0;
        for (int k = 0; k < N; k++) {
            Pvalue += A[Row*N+k] * B[k*N+Col];
        }
        C[Row*N+Col] = Pvalue;
    }
}

int main() {
    int N = 512;
    int size = N * N * sizeof(int);

    int* A, * B, * C;
    int* dev_A, * dev_B, * dev_C;

    cudaMallocHost(&A, size);
    cudaMallocHost(&B, size);
    cudaMallocHost(&C, size);
```

```
cudaMalloc(&dev_A, size);
   cudaMalloc(&dev_B, size);
   cudaMalloc(&dev_C, size);

// Initialize matrices A and B
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         A[i*N+j] = i*N+j;
         B[i*N+j] = j*N+i;
      }
   }

   cudaMemcpy(dev_A, A, size,
cudaMemcpyHostToDevice);
   cudaMemcpy(dev_B, B, size,
cudaMemcpyHostToDevice);

   dim3 dimBlock(16, 16);
   dim3 dimGrid(N/dimBlock.x, N/dimBlock.y);
   matmul<<<dimGrid, dimBlock>>>(dev_A, dev_B,
dev_C, N);
```

```
// Print the result
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        std::cout << C[i*N+j] << " ";
    }
    std::cout << std::endl;
}

// Free memory
cudaFree(dev_A);
cudaFree(dev_B);
cudaFree(dev_C);
cudaFreeHost(A);
cudaFreeHost(B);
cudaFreeHost(C);

return 0;
```

Conclusion: Implemented Addition of two large vectors and matrix multiplication using CUDA C

Title: To implement matrix vector multiplication

Problem Statement:-

To implement matrix vector multiplication

Objective:

Write a program to implement matrix vector multiplication

Outcome:

After completion of this assignment students will be able to:

- Understand the concept of matrix vector multiplication

Theory:

Matrix-vector multiplication is an operation between a matrix and a vector that produces a new vector. Notably, matrix-vector multiplication is only defined between a matrix and a vector where the length of the vector equals the number of columns of the matrix

As a "row-wise", vector-generating process: Matrix-vector multiplication defines a process for creating a new vector using an existing vector where each element of the new vector is "generated" by taking a weighted sum of each row of the matrix using the elements of a vector as coefficients

1. **As taking a linear combination of the columns of a matrix:** Matrix-vector multiplication is the process of taking a linear combination of the column-space of a matrix using the elements of a vector as the coefficients

2. **As evaluating a function between vector spaces:** Matrix-vector multiplication allows a matrix to define a mapping between two vector spaces.

Program:

```
#include<stdio.h>
#include<cuda.h>

#define BLOCKSIZE 16
#define SIZE 1024
#define EPS 1.0e-15

cudaDeviceProp deviceProp;


double *host_Mat,*host_Vect,*host_ResVect,*cpu_ResVect;
double *device_Mat,*device_Vect,*device_ResVect;
int     vlength ,matRowSize , matColSize;
int     device_Count;
int     size = SIZE;

/*mem error*/
void mem_error(char *arrayname, char *benchmark, int len, char *type)
{
      printf("\nMemory not sufficient to allocate for array %s\n\tBenchmark : %s  \n\tMemory requested = %d
number of %s elements\n",arrayname, benchmark, len, type);
      exit(-1);
}

/*calculate Gflops*/
double calculate_gflops(float &Tsec)
{
      float gflops=(1.0e-9 * (( 2.0 * size*size )/Tsec));
      return gflops;
}

/*sequential function for mat vect multiplication*/
void CPU_MatVect()
{
      cpu_ResVect = (double *)malloc(matRowSize*sizeof(double));
      if(cpu_ResVect==NULL)
            mem_error("cpu_ResVect","vectmatmul",size,"double");

      int i,j;
      for(i=0;i<matRowSize;i++)
      {cpu_ResVect[i]=0;
      for(j=0;j<matColSize;j++)
      cpu_ResVect[i]+=host_Mat[i*vlength+j]*host_Vect[j];
      }
}

/*Check for safe return of all calls to the device */
```

```
void CUDA_SAFE_CALL(cudaError_t call)
{
	cudaError_t ret = call;
	//printf("RETURN FROM THE CUDA CALL:%d\t:",ret);
	switch(ret)
	{
		case cudaSuccess:
		//		printf("Success\n");
				break;
	/*	 case cudaErrorInvalidValue:
				{
				printf("ERROR: InvalidValue:%i.\n",__LINE__);
				exit(-1);
				break;
				}
		case cudaErrorInvalidDevicePointer:
				{
				printf("ERROR:Invalid Device pointeri:%i.\n",__LINE__);
				exit(-1);
				break;
				}
		case cudaErrorInvalidMemcpyDirection:
				{
				printf("ERROR:Invalid memcpy direction:%i.\n",__LINE__);
				exit(-1);
				break;
				}			*/
		default:
			{
				printf(" ERROR at line :%i.%d' ' %s\n",__LINE__,ret,cudaGetErrorString(ret));
				exit(-1);
				break;
			}
	}
}


/*free memory*/
void dfree(double * arr[],int len)
{
	for(int i=0;i<len;i++)
		CUDA_SAFE_CALL(cudaFree(arr[i]));
	printf("mem freed\n");
}

/* function to calculate relative error*/
void relError(double* dRes,double* hRes,int size)
{
	double relativeError=0.0,errorNorm=0.0;
	int flag=0;
	int i;
```

```
	for( i = 0; i < size; ++i) {
		if (fabs(hRes[i]) > fabs(dRes[i]))
			relativeError = fabs((hRes[i] - dRes[i]) / hRes[i]);
		else
			relativeError = fabs((dRes[i] - hRes[i]) / dRes[i]);

		if (relativeError > EPS && relativeError != 0.0e+00 )
		{
			if(errorNorm < relativeError)
			{
				errorNorm = relativeError;
				flag=1;
			}
		}

	}
	if( flag == 1)
	{
		printf(" \n Results verfication : Failed");
		printf(" \n Considered machine precision : %e", EPS);
		printf(" \n Relative Error		: %e\n", errorNorm);

	}
	else
		printf("\n Results verfication : Success\n");

}


/*prints the result in screen*/
void print_on_screen(char * program_name,float tsec,double gflops,int size,int flag)//flag=1 if gflops has been
calculated else flag =0
{
	printf("\n---------------%s----------------\n",program_name);
	printf("\tSIZE\t TIME_SEC\t Gflops\n");
	if(flag==1)
	printf("\t%d\t%f\t%lf\t",size,tsec,gflops);
	else
	printf("\t%d\t%lf\t%lf\t",size,"---","---");

}

/*funtion to check blocks per grid and threads per block*/
void check_block_grid_dim(cudaDeviceProp devProp,dim3 blockDim,dim3 gridDim)
{

	if( blockDim.x >= devProp.maxThreadsDim[0] || blockDim.y >= devProp.maxThreadsDim[1] || blockDim.z
>= devProp.maxThreadsDim[2] )
	{
```

```
            printf("\nBlock Dimensions exceed the maximum limits:%d * %d * %d \
n",devProp.maxThreadsDim[0],devProp.maxThreadsDim[1],devProp.maxThreadsDim[2]);
            exit(-1);
      }

      if( gridDim.x >= devProp.maxGridSize[0] || gridDim.y >= devProp.maxGridSize[1] || gridDim.z >=
devProp.maxGridSize[2] )
      {
            printf("\nGrid Dimensions exceed the maximum limits:%d * %d * %d \
n",devProp.maxGridSize[0],devProp.maxGridSize[1],devProp.maxGridSize[2]);
            exit(-1);
      }
}


/*Get the number of GPU devices present on the host */
int get_DeviceCount()
{
      int count;
      cudaGetDeviceCount(&count);
      return count;
}


/*Fill in the vector with double precision values */
void fill_dp_vector(double* vec,int size)
{
      int ind;
      for(ind=0;ind<size;ind++)
            vec[ind]=drand48();
}


////////////////////////////////////////////////////////////////////////////////
//
// MatVect : this kernel will perform actual MatrixVector Multiplication
//
////////////////////////////////////////////////////////////////////////////////
__global__ void MatVectMultiplication(double *device_Mat, double *device_Vect,int matRowSize, int
vlength,double *device_ResVect)
  {
      int tidx = blockIdx.x*blockDim.x + threadIdx.x;
      int tidy = blockIdx.y*blockDim.y + threadIdx.y;
      int tindex=tidx+gridDim.x*BLOCKSIZE*tidy;


      if(tindex<matRowSize)
      {
      int i;int m=tindex*vlength;
      device_ResVect[tindex]=0.00;
      for(i=0;i<vlength;i++)
```

```
    device_ResVect[tindex]+=device_Mat[m+i]*device_Vect[i];
    }

  __syncthreads();

 }//end of MatVect device function



/*function to launch kernel*/
void launch_Kernel_MatVectMul()
{
/*        threads_per_block, blocks_per_grid  */


int max=BLOCKSIZE*BLOCKSIZE;
int BlocksPerGrid=matRowSize/max+1;
 dim3 dimBlock(BLOCKSIZE,BLOCKSIZE);
if(matRowSize%max==0)BlocksPerGrid--;
dim3 dimGrid(1,BlocksPerGrid);
 check_block_grid_dim(deviceProp,dimBlock,dimGrid);

MatVectMultiplication<<<dimGrid,dimBlock>>>(device_Mat,device_Vect,matRowSize,vlength,device_ResVe
ct);

}


/*main function*/
int main()
{
      // Vector length , Matrix Row and Col sizes.....
      vlength = matColSize = SIZE;
      matRowSize = SIZE;

      //  printf("this programs does computation of square matrix only\n");
      float elapsedTime,Tsec;
      cudaEvent_t start,stop;

      device_Count=get_DeviceCount();
      printf("\n\nNUmber of Devices : %d\n\n", device_Count);

      // Device Selection, Device 1: Tesla C1060
      cudaSetDevice(0);

      int device;
      // Current Device Detection
      cudaGetDevice(&device);
      cudaGetDeviceProperties(&deviceProp,device);
      printf("Using device %d: %s \n", device, deviceProp.name);
```

```
/*allocating the memory for each matrix */
host_Mat =new double[matRowSize*matColSize];
host_Vect = new double[vlength];
host_ResVect = new double[matRowSize];


// ---------------checking host memory  for error.....................
 if(host_Mat==NULL)
      mem_error("host_Mat","vectmatmul",matRowSize*matColSize,"double");

 if(host_Vect==NULL)
      mem_error("host_Vect","vectmatmul",vlength,"double");

 if(host_ResVect==NULL)
      mem_error("host_ResVect","vectmatmul",matRowSize,"double");

//--------------Initializing the input arrays.............
fill_dp_vector(host_Mat,matRowSize*matColSize);
fill_dp_vector(host_Vect,vlength);

/* allocate memory for GPU events
start = (cudaEvent_t) malloc (sizeof(cudaEvent_t));
stop = (cudaEvent_t) malloc (sizeof(cudaEvent_t));
if(start==NULL)
      mem_error("start","vectvectmul",1,"cudaEvent_t");
if(stop==NULL)
      mem_error("stop","vectvectmul",1,"cudaEvent_t");*/

//event creation...
CUDA_SAFE_CALL(cudaEventCreate (&start));
CUDA_SAFE_CALL(cudaEventCreate (&stop));

//allocating memory on GPU
CUDA_SAFE_CALL(cudaMalloc( (void**)&device_Mat, matRowSize*matColSize* sizeof(double)));
CUDA_SAFE_CALL(cudaMalloc( (void**)&device_Vect, vlength* sizeof(double)));
CUDA_SAFE_CALL(cudaMalloc( (void**)&device_ResVect, matRowSize* sizeof(double)));

//moving data from CPU to GPU
CUDA_SAFE_CALL(cudaMemcpy((void*)device_Mat, (void*)host_Mat,
matRowSize*matColSize*sizeof(double) ,cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaMemcpy((void*)device_Vect,
(void*)host_Vect,vlength*sizeof(double),cudaMemcpyHostToDevice));

// Launching kernell.....
CUDA_SAFE_CALL(cudaEventRecord (start, 0));

launch_Kernel_MatVectMul();

CUDA_SAFE_CALL(cudaEventRecord (stop, 0));
```

```
    CUDA_SAFE_CALL(cudaEventSynchronize (stop));
    CUDA_SAFE_CALL(cudaEventElapsedTime ( &elapsedTime, start, stop));

    Tsec= 1.0e-3*elapsedTime;

    // calling funtion for measuring Gflops
    calculate_gflops(Tsec);

    //printing the result on screen
    print_on_screen("MAT VECT MULTIPLICATION",Tsec,calculate_gflops(Tsec),size,1);


    //retriving result from device
    CUDA_SAFE_CALL(cudaMemcpy((void*)host_ResVect,
(void*)device_ResVect,matRowSize*sizeof(double),cudaMemcpyDeviceToHost));

    // CPU calculation..and checking error deviation....
    CPU_MatVect();
    relError(cpu_ResVect,host_ResVect,size);
    printf("\n ------------------------------------------------------------------\n");

    /*free the memory from GPU */
    double *array[3];
    array[0]=device_Mat;
    array[1]=device_Vect;
    array[2]=device_ResVect;
    dfree(array,3);

    //free host memory----------
    free(host_Mat);
    free(host_Vect);
    free(host_ResVect);
    free(cpu_ResVect);

    return 0;
}// end of main
```

Conclusion: Implemented matrix vector multiplication