HPC 1 Design and implement Parallel

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

using namespace std;

// Graph class representing an undirected graph
using adjacency list representation
class Graph {
private:
   int numVertices;          // Number of vertices
   vector<vector<int>> adj;  // Adjacency list

public:
   Graph(int vertices) : numVertices(vertices),
adj(vertices) {}

   // Add an edge between two vertices
   void addEdge(int src, int dest) {
      adj[src].push_back(dest);
      adj[dest].push_back(src);
   }

   // View the graph
   void viewGraph() {
      cout << "Graph:\n";
      for (int i = 0; i < numVertices; i++) {
         cout << "Vertex " << i << " -> ";
         for (int neighbor : adj[i]) {
            cout << neighbor << " ";
         }
         cout << endl;
      }
   }

   // Perform Breadth First Search (BFS) in
parallel
   void bfs(int startVertex) {
      vector<bool> visited(numVertices, false);
      queue<int> q;

      // Mark the start vertex as visited and
enqueue it
      visited[startVertex] = true;
      q.push(startVertex);

      while (!q.empty()) {
         int currentVertex = q.front();
         q.pop();
         cout << currentVertex << " ";

         // Enqueue all adjacent unvisited vertices
         #pragma omp parallel for
         for (int neighbor : adj[currentVertex]) {
            if (!visited[neighbor]) {
               visited[neighbor] = true;
               q.push(neighbor);
            }
         }
      }
   }

   // Perform Depth First Search (DFS) in
parallel
   void dfs(int startVertex) {
      vector<bool> visited(numVertices, false);
      stack<int> s;

      // Mark the start vertex as visited and push
it onto the stack
      visited[startVertex] = true;
      s.push(startVertex);

      while (!s.empty()) {
         int currentVertex = s.top();
         s.pop();
         cout << currentVertex << " ";

         // Push all adjacent unvisited vertices onto
the stack
         #pragma omp parallel for
         for (int neighbor : adj[currentVertex]) {
            if (!visited[neighbor]) {
               visited[neighbor] = true;
               s.push(neighbor);
            }
         }
      }
   }
};

int main() {
   int numVertices;
   cout << "Enter the number of vertices in the
graph: ";
   cin >> numVertices;

   // Create a graph with the specified number of
vertices
   Graph graph(numVertices);

   int numEdges;
   cout << "Enter the number of edges in the
graph: ";
   cin >> numEdges;

   cout << "Enter the edges (source
destination):\n";
   for (int i = 0; i < numEdges; i++) {
      int src, dest;
```

```
        cin >> src >> dest;
        graph.addEdge(src, dest);
    }

    // View the graph
    graph.viewGraph();

    int startVertex;
    cout << "Enter the starting vertex for BFS and DFS: ";
    cin >> startVertex;

    cout << "Breadth First Search (BFS): ";
    graph.bfs(startVertex);
    cout << endl;

    cout << "Depth First Search (DFS): ";
    graph.dfs(startVertex);
    cout << endl;

    return 0;
}

// Output from the program will be:

// Enter the number of vertices in the graph: 5
// Enter the number of edges in the graph: 6
// Enter the edges (source destination):
// 0 1
// 0 2
// 1 3
// 1 4
// 2 4
// 3 4
```

HPC 2 Write a program to implement Parallel Bubble Sort and Merge sort

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <omp.h>

using namespace std;

void bubbleSort(int arr[], int n)
{
    for (int i = 0; i < n - 1; ++i)
    {
        for (int j = 0; j < n - i - 1; ++j)
        {
            if (arr[j] > arr[j + 1])
            {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    int *L = new int[n1];
    int *R = new int[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];
            ++i;
        }
        else
        {
            arr[k] = R[j];
            ++j;
        }
        ++k;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        ++i;
        ++k;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        ++j;
        ++k;
    }

    delete[] L;
    delete[] R;
}

void mergeSort(int arr[], int l, int r)
```

```
{
    if (l < r)
    {
        int m = l + (r - l) / 2;
        #pragma omp parallel sections
        {
            #pragma omp section
            {
                mergeSort(arr, l, m);
            }
            #pragma omp section
            {
                mergeSort(arr, m + 1, r);
            }
        }

        merge(arr, l, m, r);
    }
}

void printArray(int arr[], int size)
{
    for (int i = 0; i < size; ++i)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main()
{
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;

    int *arr = new int[n];
    srand(time(0));
    for (int i = 0; i < n; ++i)
    {
        arr[i] = rand() % 100;
    }

    // cout << "Original array: ";
    // printArray(arr, n);

    // Sequential Bubble Sort
    clock_t start = clock();
    bubbleSort(arr, n);
    clock_t end = clock();

    // cout << "Sequential Bubble Sorted array: ";
    // printArray(arr, n);

    double sequentialBubbleTime = double(end -
start) / CLOCKS_PER_SEC;

    // Parallel Bubble Sort
    start = clock();
    #pragma omp parallel
    {
        bubbleSort(arr, n);
    }
    end = clock();

    // cout << "Parallel Bubble Sorted array: ";
    // printArray(arr, n);

    double parallelBubbleTime = double(end -
start) / CLOCKS_PER_SEC;

    // Merge Sort
    start = clock();
    mergeSort(arr, 0, n - 1);
    end = clock();

    // cout << "Sequential Merge Sorted array: ";
    // printArray(arr, n);

    double sequentialMergeTime = double(end -
start) / CLOCKS_PER_SEC;

    // Parallel Merge Sort
    start = clock();
    #pragma omp parallel
    {
        #pragma omp single
        {
            mergeSort(arr, 0, n - 1);
        }
    }
    end = clock();

    // cout << "Parallel Merge Sorted array: ";
    // printArray(arr, n);

    double parallelMergeTime = double(end -
start) / CLOCKS_PER_SEC;

    // Performance measurement
    cout << "Sequential Bubble Sort Time: " <<
sequentialBubbleTime << " seconds" << endl;
    cout << "Parallel Bubble Sort Time: " <<
parallelBubbleTime << " seconds" << endl;
    cout << "Sequential Merge Sort Time: " <<
sequentialMergeTime << " seconds" << endl;
    cout << "Parallel Merge Sort Time: " <<
parallelMergeTime << " seconds" << endl;

    delete[] arr;

    return 0;
}
```

HPC 3 Implement Min, Max, Sum and Average

```
#include <iostream>
#include <vector>
#include <omp.h>
#define SIZE 1000000  // Size of the array
int main() {
    std::vector<double> arr(SIZE);

    // Initialize array
    for (int i = 0; i < SIZE; i++) {
        arr[i] = i + 1;  // Sample data
    }
    double min_val = arr[0];
    double max_val = arr[0];
    double sum_val = 0.0;
    double avg_val;
    // Parallel Reduction for Min
    #pragma omp parallel for
reduction(min:min_val)
    for (int i = 0; i < SIZE; i++) {
        if (arr[i] < min_val) {
            min_val = arr[i];
        }
    }
    // Parallel Reduction for Max
    #pragma omp parallel for
reduction(max:max_val)
    for (int i = 0; i < SIZE; i++) {
        if (arr[i] > max_val) {
            max_val = arr[i];
        }
    }
    // Parallel Reduction for Sum
    #pragma omp parallel for
reduction(+:sum_val)
    for (int i = 0; i < SIZE; i++) {
        sum_val += arr[i];
    }
    // Calculate Average
    avg_val = sum_val / SIZE;

    std::cout << "Min Value: " << min_val <<
std::endl;
    std::cout << "Max Value: " << max_val <<
std::endl;
    std::cout << "Sum Value: " << sum_val <<
std::endl;
    std::cout << "Average Value: " << avg_val <<
std::endl;

    return 0;
}
```

HPC 4

1.Addition of two large vectors

```
#include <iostream>
#include <vector>
#include <omp.h>
using namespace std;
int main() {
    const int size = 1000000;
    vector<int> vec1(size, 1);
    vector<int> vec2(size, 2);
    vector<int> result(size);
    #pragma omp parallel for
    for(int i = 0; i < size; ++i) {
        result[i] = vec1[i] + vec2[i];
    }
    cout << "RESULT VECTOR:" << endl;
    for(int i = 0; i < 10; ++i) {
        cout << result[i] << " ";
    }
    cout << endl;

    return 0;
}
```

2.Matrix Multiplication using CUDA C

```
#include <iostream>
#include <vector>
#include <omp.h>
using namespace std;
int main() {
    const int rows = 1000;
    const int cols = 1000;

    vector<vector<int>> matrix1(rows,
vector<int>(cols, 1));
    vector<vector<int>> matrix2(rows,
vector<int>(cols, 2));
    vector<vector<int>> result(rows,
vector<int>(cols, 0));

    #pragma omp parallel for collapse(2)
    for(int i = 0; i < rows; ++i) {
        for(int j = 0; j < cols; ++j) {
            for(int k = 0; k < cols; ++k) {
                result[i][j] += matrix1[i][k] *
matrix2[k][j];
            }
        }
    }

    cout << "Result matrix:" << endl;
    for(int i = 0; i < 3; ++i) {
        for(int j = 0; j < 3; ++j) {
            cout << result[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

DL 1 Linear regression by using Deep Neural network

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense

df = pd.read_csv('Boston.csv')
df.head(10)

df.drop(columns=['Unnamed: 15','Unnamed: 16'],inplace=True)

df.drop(columns=['CAT. MEDV'],inplace=True)

df.isnull().sum()

df.info()

df.describe()

df.corr()['MEDV'].sort_values()

X = df.loc[:,['LSTAT','PTRATIO','RM']]
Y = df.loc[:,"MEDV"]
X.shape,Y.shape

x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.25,random_state=10)

scaler = StandardScaler()

x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

model = Sequential()

model.add(Dense(128,input_shape=(3,),activation='relu',name='input'))
model.add(Dense(64,activation='relu',name='layer_1'))
model.add(Dense(1,activation='linear',name='output'))
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.summary()

model.fit(x_train,y_train,epochs=100,validation_split=0.05)

output = model.evaluate(x_test,y_test)

print(f"Mean Squared Error: {output[0]}"
      ,f"Mean Absolute Error: {output[1]}",sep="\n")

y_pred = model.predict(x=x_test)

print(*zip(y_pred,y_test))
```

DL 2 Multiclass classification

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical

# Load the dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/letter-recognition/letter-recognition.data"
names = ['letter', 'x-box', 'y-box', 'width', 'high', 'onpix', 'x-bar', 'y-bar', 'x2bar', 'y2bar', 'xybar', 'x2ybr', 'xy2br', 'x-ege', 'xegvy', 'y-ege', 'yegvx']
data = pd.read_csv(url, names=names)

# Split the data into features and target
X = data.drop('letter', axis=1)
y = data['letter']

# Encode the target variable
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
y = to_categorical(y)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the neural network model
model = Sequential([
   Dense(128, input_dim=X_train.shape[1], activation='relu'),
   Dropout(0.5),
   Dense(64, activation='relu'),
   Dropout(0.5),
   Dense(len(label_encoder.classes_), activation='softmax')
])
```

```
# Compile the model
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=30,
batch_size=32, validation_split=0.1)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

DL 3 CNN MNIST Fashion Dataset and

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

train_df = pd.read_csv('fashion-mnist_train.csv')
test_df = pd.read_csv('fashion-mnist_test.csv')

train_df.shape

test_df.shape

train_df.describe()

train_df.label.unique()

class_names = ['T-shirt/top', 'Trouser',
'Pullover', 'Dress', 'Coat',
'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

x_train = train_df.iloc[:,1:].to_numpy()
x_train = x_train.reshape([-1,28,28,1])
x_train = x_train / 255

y_train = train_df.iloc[:,0].to_numpy()

x_test = test_df.iloc[:,1:].to_numpy()
x_test = x_test.reshape([-1,28,28,1])
x_test = x_test / 255

y_test = test_df.iloc[:,0].to_numpy()

plt.figure(figsize=(10,10))
for i in range(25):
   plt.subplot(5,5,i+1)
   plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   plt.imshow(x_train[i], cmap=plt.cm.binary)
   plt.xlabel(class_names[y_train[i]])
plt.show()

from keras.models import Sequential
from keras.layers import
Dense,Conv2D,Flatten,MaxPooling2D,Dropout

model = Sequential()

model.add(Conv2D(filters=64,kernel_size=(3,3),input_shape=(28,28,1),activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(rate=0.3))
model.add(Flatten())
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=10,
activation='sigmoid'))
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()

model.fit(x_train,y_train,epochs=50,batch_size=1200,validation_split=0.05)

evaluation = model.evaluate(x_test,y_test)

print(f"Accuracy: {evaluation[1]}")

y_probas = model.predict(x_test)

y_pred = y_probas.argmax(axis=-1)

y_pred

plt.figure(figsize=(10,10),)

for i in range(25):
   plt.subplot(5,5,i+1)
   plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   plt.imshow(x_test[i], cmap=plt.cm.binary)
#    plt.xlabel(f"True Class:{y_test[i]}")
   plt.title(f"Pred:{class_names[y_pred[i]]}")
plt.show()

from sklearn.metrics import
classification_report

num_classes = 10
class_names = ["class {}".format(i) for i in
range(num_classes)]
cr = classification_report(y_test, y_pred,
target_names=class_names)
print(cr)
```