HOMEWORK – 2

Name: Shushhma Dewie Koppireddy

ID : 02190512

**Problem - 1**

**Serial for- loop ( in Python )**

import os

import time

physical\_cores = os.cpu\_count() // 2

logical\_cores = os.cpu\_count()

print(f"Python detected: {physical\_cores} physical cores.")

print(f"Python detected: {logical\_cores} logical cores.")

def serial\_task(n):

result = 0

for i in range(n):

result += i \* i

return result

start\_time = time.time()

serial\_result = serial\_task(100000000)

end\_time = time.time()

print(f"Serial result: {serial\_result}")

print(f"Time taken in serial loop: {end\_time - start\_time} seconds")

**Parallel for loop ( in Python )**

import multiprocessing

import os

import time

def compute\_sum\_of\_squares(start, end):

return sum(i \* i for i in range(start, end))

if \_name\_ == '\_main\_':

physical\_cores = 4

logical\_cores = os.cpu\_count()

print(f"Detected physical cores: {physical\_cores}")

print(f"Detected logical cores: {logical\_cores}")

n = 100\_000\_000

chunk\_size = n

ranges = [(i \* chunk\_size, (i + 1) \* chunk\_size) for i in range(physical\_cores)]

ranges[-1] = (ranges[-1][0], n)

start\_time = time.time()

with multiprocessing.Pool(processes=physical\_cores) as pool:

results = pool.starmap(compute\_sum\_of\_squares, ranges)

total\_sum = sum(results)

end\_time = time.time()

print(f"Total sum of squares: {total\_sum}")

print(f"Time taken for parallel computation: {end\_time - start\_time:.2f} seconds")

**Serial for – loop ( In Java )**

import java.time.Duration;

import java.time.Instant;

public class SerialLoop {

public static void main(String[] args) {

int physicalCores = Runtime.getRuntime().availableProcessors() / 2;

int logicalCores = Runtime.getRuntime().availableProcessors();

System.out.println("Java detected: " + physicalCores + " physical cores.");

System.out.println("Java detected: " + logicalCores + " logical cores.");

Instant start = Instant.now();

long result = serialTask(100\_000\_000);

Instant end = Instant.now();

System.out.println("Serial result: " + result);

System.out.println("Time taken in serial loop: " + Duration.between(start, end).toMillis() + " milliseconds");

}

public static long serialTask(int n) {

long result = 0;

for (int i = 0; i < n; i++) {

result += i \* i;

}

return result;

}

}

**Parallel for – loop ( In Java )**

import java.util.concurrent.\*;

import java.time.Duration;

import java.time.Instant;

import java.util.ArrayList;

import java.util.List;

public class ParallelLoop {

public static void main(String[] args) throws InterruptedException, ExecutionException {

int physicalCores = Runtime.getRuntime().availableProcessors() / 2;

int logicalCores = Runtime.getRuntime().availableProcessors();

System.out.println("Java detected: " + physicalCores + " physical cores.");

System.out.println("Java detected: " + logicalCores + " logical cores.");

Instant start = Instant.now();

long parallelResult = parallelTask(100\_000\_000, logicalCores);

Instant end = Instant.now();

System.out.println("Parallel result: " + parallelResult);

System.out.println("Time taken in parallel loop: " + Duration.between(start, end).toMillis() + " milliseconds");

}

public static long parallelTask(int n, int numberOfTasks) throws InterruptedException, ExecutionException {

ExecutorService executor = Executors.newFixedThreadPool(numberOfTasks);

List<Future<Long>> futures = new ArrayList<>();

int chunkSize = n / numberOfTasks;

for (int i = 0; i < numberOfTasks; i++) {

int start = i \* chunkSize;

int end = (i == numberOfTasks - 1) ? n : (i + 1) \* chunkSize;

futures.add(executor.submit(() -> parallelSubTask(start, end)));

}

long result = 0;

for (Future<Long> future : futures) {

result += future.get();

}

executor.shutdown();

return result;

}

public static long parallelSubTask(int start, int end) {

long result = 0;

for (int i = start; i < end; i++) {

result += i \* i;

}

return result;

}

}

**PROBLEM – 2**

**Small Number of iterations:**

% Small number of iterations

clear all;

if isempty(gcp())

parpool(); % Create a parallel pool if none exists

end

n = 100; % Small number of iterations

tic;

parfor i = 1:n

timeconsumingfun(i); % Placeholder for a function

end

tp\_small = toc;

fprintf('Time for small number of iterations: %f seconds\n', tp\_small);

**Medium Number of Iterations:**

% Medium number of iterations

clear all;

if isempty(gcp())

parpool(); % Create a parallel pool if none exists

end

n = 10000; % Medium number of iterations

tic;

parfor i = 1:n

timeconsumingfun(i); % Placeholder for a function

end

tp\_medium = toc;

fprintf('Time for medium number of iterations: %f seconds\n', tp\_medium);

**Large Number of Iteration:**

% Large number of iterations

clear all;

if isempty(gcp())

parpool(); % Create a parallel pool if none exists

end

n = 1000000; % Large number of iterations

tic;

parfor i = 1:n

timeconsumingfun(i); % Placeholder for a function

end

tp\_large = toc;

fprintf('Time for large number of iterations: %f seconds\n', tp\_large);

**PROBLEM – 3**

import time

import random

from concurrent.futures import ThreadPoolExecutor

def timeconsumingfun():

pause\_time = random.uniform(1, 5) # Random pause between 1 and 5 seconds

time.sleep(pause\_time)

def run\_serial(n):

start\_time = time.time() # Start timer

for \_ in range(n):

timeconsumingfun() # Run time-consuming function serially

end\_time = time.time() # Stop timer

return end\_time - start\_time

def run\_parallel(n, num\_workers):

start\_time = time.time() # Start timer

with ThreadPoolExecutor(max\_workers=num\_workers) as executor:

futures = [executor.submit(timeconsumingfun) for \_ in range(n)]

for future in futures:

future.result()

end\_time = time.time()

return end\_time - start\_time

def calculate\_metrics(serial\_time, parallel\_time, num\_workers):

speedup = serial\_time / parallel\_time

efficiency = speedup / num\_workers

return speedup, efficiency

def run\_experiment(n, num\_workers):

serial\_time = run\_serial(n)

print(f"Serial Execution Time for n = {n}: {serial\_time:.2f} seconds")

parallel\_time = run\_parallel(n, num\_workers)

print(f"Parallel Execution Time for n = {n} with {num\_workers} workers: {parallel\_time:.2f} seconds")

speedup, efficiency = calculate\_metrics(serial\_time, parallel\_time, num\_workers)

print(f"Speedup: {speedup:.2f}")

print(f"Efficiency: {efficiency:.2f}")

print('-----------------------------------')

small\_n = 10

medium\_n = 100

large\_n = 1000

num\_workers = 4

if \_name\_ == "\_main\_":

print('Running small n experiment...')

run\_experiment(small\_n, num\_workers)

print('Running medium n experiment...')

run\_experiment(medium\_n, num\_workers)

print('Running large n experiment...')

run\_experiment(large\_n, num\_workers)