**Project ECTE 331 Report**

Karim Safarini

7391560

Github link:

https://github.com/karoomkk/Project\_331

PART A:

Introduction

To achieve this goal, the objective of this project is to build an image processing system to apply the template matching algorithm using single-thread and multi-threading. Template matching can be defined as a technique that is used in visual processing for the purpose of finding areas of the image that resemble a predefined template image. This project entails reading and processing the image and analyzing which part has the closest resemblance to the other and carrying out the single-thread method and the multi-thread technique to check the performance differences.

Project Overview

Image Processing: Loading images and converting the images to grayscale: The pictures of the liver images are initially pre-loaded before any image processing can be conducted based on its pixel intensity values.

Template Matching: Searching for corresponding areas in the source image with the help of the availability of the template image.

Single-Threaded Approach: It was established for the current to perform template matching within a single thread of execution.

Multi-Threaded Approach: To increase the speed of the template matching function the second parameter is to use multiple threads.

Performance Comparison: Quantitative comparison of the time performances of single-threaded and of multi-threaded programs.

Components

Image Processing

Class: ImageProcessor

Methods:

loadAndConvertImage(String fileName): This function involves decoding an image to be processed and converts it into a gray scale image.

getGrayscaleImage(): Returns the grayscale image of the object or structure.

createBufferedImage(short[][] grayImage): This operation converts the grayscale image to BufferedImage and compares the two in terms of pixel dimensions.

saveImage(BufferedImage image, String filePath): This creates a picture at the given path and saves it.

This component also plays a role in loading colour images for converting them into black and white images as well as providing for storage of the images.

Template Matching

Class: TemplateMatcher

Method:

matchTemplate(short[][] template, short[][] source): To find the normalized difference in pixel intensities, aligns a source image with a template image resulting in a matrix.

It is this component that’s charged with the responsibility of performing the basic template matching operation; that’s cycling through the source image in order to find sections that bear a close resemblance to the template.

Single-Threaded Approach

Class: SingleThreadProcessor

Methods:

main(String[] args): This process applies the activity of template matching on one thread of the machine.

drawMatchingRectangles(BufferedImage image, double[][] results, int templateHeight, int templateWidth): Place rectangles about relevant regions.

drawRectangle(BufferedImage image, int x, int y, int width, int height): Draw a rectangle on the image and make the location of the rectange single and the shape is rectangle.

This component also requires the TemplateMatcher class where the template matching is conducted simultaneously in a single-threaded manor and where the matching areas on the source image is marked.

Multi-Threaded Approach

Class: MultiThreadedTemplateMatcher

Inner Class: TemplateMatchTask

Methods:

main(String[] args): Multi-threaded process of matching the templates on two streams conducted by this class.

TemplateMatchTask. run(): Facilitates matching the image segment based on templates.

This component is responsible for the segmentation of the source image into blocks or stripes and for assigning each segment to a different thread for carrying out the template matching in parallel. It then fuses all detected threads present in the Video to find the maximum matched regions present.

Performance Comparison

Class: ExecutionComparison

Methods:

main(String[] args): Compares and quantifies the execution time of the test when performed under single-threaded and multi-threaded paradigms.

This component calculates the execution times for both the approaches and then based on physical times prints the results which gives the user an idea of difference between executing a program with multiple threads and executing it sequentially, thereby explaining the effectiveness of the multi-threading.

PART B:

**Introduction:**

This purpose of project is to ensure that two threads, namely ‘A’ and ‘B’, execute a sequence of related methods. The implementation guarantees that every call to one of the functions will only continue executing if its dependency has been computed by that point. In this context, the report describes the synchronization mechanism employed, provides code references to the lines of code where the synchronization is performed, and as well includes the synch testing.

**Shared Data Structure:**

Data class containing the global variables (dependent variables A1, A2, A3, B1, B2, B3) and Synchronized flags (goFunA2, goFunA3, goFunB2, goFunB3)

**Thread Creation:**

In the main() function, instances of ThreadA and ThreadB are created and started for each iteration.

**Synchronization Using wait() and notify():**

Each function call is synchronized using the wait() and notify() methods to ensure proper execution.

ThreadA: A1 notifies B2, A2 waits for B2 to complete, and A3 waits for B3 to complete.

ThreadB: B2 waits for A1 to complete, and B3 waits for A2 to complete.

**High Number of Iterations and Verification:**

In the main() function, the threads are run for 1000 iterations to make sure that the implementation of the threads is correct and properly synchronized. At the end of each iteration A3 is checked to ensure that it is correct.

**Testing Results:**

In the testing, the index was tested by running the threads for 1000 cycles. Every time round the calculation, the value for A3 was checked against the correct value to ensure correctness. The comparative was made through the formula that calculates the sum of the four hundred natural numbers plus the content in B3.

**Outcome:**

It was also successful for all the 1000 iterations and the value of A3 obtained was as per the expected value.

The verbose in the success message “successful operation!!!!!” assured that the synchronization and the order of execution was properly designed and followed.

**Conclusion:**

The project was able to achieve synchronization to be used for two coordinating threads with the use of wait(). The common variable count was correct in the needed order where numerous iterations provided the forked program’s correct and reliable performance. This helps to make sure that what thread is supposed to do kind of function sequence is done in the right order with correct function calls dependent on the other function.