Parallel Architectures and Programming Models

Assignment 1 (Documentation)

Student name: Md Rafiqul Islam

Student Id: 12123971

Framework used:

JAVA Fork/Join framework

How to run program:

1. Compile java class (with package):

~ “javac -d . \*.java”

1. Execute java code:

~ “java <package name>.<class name> <image name>.<image format>”

Example: ~ “java testimagefilter.TestImageFilter IMAGE2.JPG”

Code Structure:

There is total three java file in this project. They are:

* **ImageFilter.java**

-This class contains the sequential image filtering process.

* **ParallelFJImageFilter.java**

-This is the class where parallel image filter has been done. This class contains some methods and one inner class.

* **TestImageFilter.java**

-This is the main class of the project. From this class the sequential and parallel image filtering has been called. The performance of sequential and parallel image filter also had been measured here. And finally, the comparison also had been done here between sequential and parallel image filtered file.

Threshold:

For parallel execution we need a proper threshold to limit the number of tasks generated. Here we compute the threshold by using formula below:

Threshold = height/totalThreads.

Here, Threshold is an integer type number. Basically, For parallel image filter we spilt height loop of the filter image for fork join task that’s why We took height of the image and we also took total Threads used for parallel execution. Then the rounded fraction of image height and total threads is used as threshold.

Number of tasks generated:

As it is told that here the Fork/Join framework is used for parallel image filter. The **ParallelFJImageFilter** class has an inner class called **InnerFJImageFilter.** Basically, the Fork/Join has applied in **InnerFJImageFilter** class. This class override a function called **compute ()**. Inside compute () function the tasks has been generated for parallel execution. There are two tasks (**task1** and **task2**) has been generated for parallel execution. These tasks working recursively and here the synchronization has been handled by **ForkJoinTask.invokeAll().**

Algorithm:

The class **ParallelFJImageFilter** implements a parallel, iterative nine-point image convolution filter working on a linearized (2D) image. In each of the NRSTEPS (=100) iteration steps, the average RGB-value of each pixel p in the source array is computed, considering p and its 8 neighbor pixels (in 2D) and written to the destination array. To compare the sequential image filter and parallel image filter here two different source and destination were used.

Algorithm:

1. Create a class with constructor which takes src, dst, width and height of image
2. Define total number of threads which will be use in parallel execution.
3. Calculate threshold (threshold = imageHeight/nThreads).
4. Create ForkJoinPool with nThreads.
5. Create tasks for parallel execution.
6. Execute the tasks recursively.
7. Properly synchronize the tasks
8. Write the output and check if it is correct or not.

Description:

Here **ParallelFJImageFilter** class has been cleated with a constructor look like this **ParallelFJImageFilter(int[] src, int[] dst, int w, int h).** This constructor takes image source, destination, height, width as a parameter. **ParallelFJImageFilter** has an inner class called **InnerFJImageFilter** with constructor **InnerFJImageFilter(int[] src, int[] dst, int start, int end)**. Here start and end is basically the splited image height.

From main class when a instance of **ParallelFJImageFilter** class has been created first of all the constructor of this class called **apply(int nthreads)** function and here is the basically, ForkJoinPool initiate with nThreads (nThreads is the number of threads we want to execute parallel program). After that the pool invoke the task which generated inside **compute ()** function in the **InnerFJImageFilter** class. Mainly here is the parallel execution happened. Inside **compute ()** function the program check if the height of image is greater than threshold or not. If height is greater than threshold then it called function **filterProcess()** which start the filter processing. Otherwise, the program split the image height into 2 division and recursively called again and again up to finish the work. When execution has done, **invokeAll(task1, task2)** synchronize the tasks and attach them. This way the parallel image filter has been done.

Graph:

Description:

Conclusion:

After executing the sequential image filter and parallel image filter with 1, 2, 4, 8, 16 threads, it has seen parallel execution is faster than the sequential process. In the future, if we face any hardware restrictions then parallel programming can be one solution.