High Performance Digital Embedded Systems

EEE4120F

Practical 2 – Static pThreads

[30 Marks]

Introduction

The focus of this practical is partitioning data for multi-threaded applications. The test case is a median filter.

Median filters utilise statistics to remove extraneous values (noise) from data sets. These data sets can be of an arbitrary number of dimensions, but for this practical you will only be working with two-dimensional data sets, or images.

The Algorithm

A 9x9 median filter should be used. Each pixel in the output image is the median of the corresponding pixel in the input image and the surrounding 80 pixels.

You can decide how to handle the border cases. Typically the pixel indices are clamped to the border, but you can also use zeros for pixels outside the image, for instance.

Each colour component is handled on its own, so for a full colour image you need to run the algorithm three times on the same pixel.

Skeleton Code

This practical comes with a file called “Source.zip”. It contains C++ source code from which you can start this practical. The only file you need to edit is “Prac2.cpp” (and maybe “Prac2.h”).

On Linux, open the source files in whatever editor you like (Vim, GEdit, etc.) and run by using a terminal (change the directory to the project folder and type “make” at the command-line). This does not work from a FAT32-based flash-drive: you need to copy the source into the home folder. In Windows, install MinGW and Code::Blocks (these are links) and open ”Prac2.cbp”.

The “Tools” folder contains wrapper modules for LibJPEG and the system timer. Go through the code in “Prac2.cpp” and make sure you understand how to make use of these wrappers. While you're at it, make sure you understand all the pThread commands.

The “Libraries” folder is used in Windows only. It contains the static library and headers for LibJPEG. On Linux, LibJPEG is installed in the standard system folders and you should link to those instead (the makefile does this for you).

Part 1: Golden Measure

The first task is to set up a golden measure. This version should not involve much optimisation. For instance, rather use an easy-to-implement bubble-sort, instead of a faster quick-sort. See http://www.sorting-algorithms.com/ for other sorting algorithms. The aim is to obtain a solution that is known to be correct, rather that fast.

Measure how long it takes, and keep the output image as further reference.

Part 2: Multi-threaded Version

The next task is to implement an optimised version. Here you can make use of quick-sort, multiple threads, etc.

Try different number of threads and different ways to partition the data. Say, for instance, you have an image with 16 rows and spawn 4 threads. You can let thread 1 handle rows 1 to 4, thread 2 rows 5 to 8, etc. Or you could let thread 1 handle rows 1, 5, 9 and 13; thread 2 rows 2, 6, 10 and 14; etc. Compare the time taken for each partitioning method / thread count and comment on the results.

Does the image access (RAM access) stage of each thread need to go into a critical section? Make an argument for / against.

Bonus marks are available if you correlate the results of the golden measure with that of the parallel version(s) to show that the result is correct.

Part 3: Report

Compile your experiments and findings into an IEEE-style conference paper. The page limit is 3 pages.

Submit your paper to the Vula Assignment for this practical.

END OF ASSIGNMENT