High Performance Digital Embedded Systems
Practical 2 - Static pThreads

Jonathan Oehley[†] and Alan Pohl[‡]
EEE4120F Class of 2019
University of Cape Town
South Africa
[†]OHLJON001 [‡]PHLALA005

***Abstract*—This report details an investigation into the use of multi-threaded applications in partitioning data for a median image filter. A golden measure, a serial implementation, was used to draw comparisons to an optimized and threaded implementation.**

I. Introduction

Things to still deal with:

- choose speedup or speed-up
- move image outputs

A Median Filter is a filtering method often used in image and signal processing. On a 2-dimensional image problem, each colour component of the output pixel is the median of the surrounding n x n pixel colour components. Such a filter is often implemented to reduce speckle or salt-and-pepper noise in images [1].

This investigation will implement a 9x9 median filter.

II. Methodology

The implementation of a median filter requires some definition around the pixels near the edge of an image. The outer edge-case pixels were calculated by the following means; the area over which the pixel component values were compared with to determine the median was simply truncated by the edge of the image. In other words, for the pixels in the furthest corners, the area over which the median was determined was 5 x 5 pixels.

A. Golden Measure

The golden measure was created as a relatively simple block of code that's main objective was to be a working model. From this, further implementation could be compared and a comparison drawn.

The golden measure sorting method was implemented as a simple bubble sort. While not the quickest sorting technique, the bubble sort is easy to code and understand. Therefore, it was possible to implement the sorting method relatively quickly in this manner. The filter was simply implemented by flattening the 2 dimensional comparison area, into a single array before passing it to the sorting algorithm to determine the median.

B. Multi-thread Implementation

This implementation looked to improve the execution time of the golden measure. This was done using two techniques.

Firstly, instead of implementing a bubble sort by default, a investigation was made between bubble sort, a select sort function (which only sorted just above halfway to get the median) and the hybrid sort function (std::sort()) built into c++. The fastest sorting method was used in this implementation which will be discussed in the Results section.

Secondly, the golden measure implementation was converted into a multi-threaded application. This was done through data-partitioning of the image into rows. This allows the data sorting tasks to be split up over multiple processors to decrease the overall execution time of the application. No mutex was required as each pixel of the output jpeg is only written to once.

C. Experiment Procedure

The program was coded using a shared git repository to allow collaboration between the two students. The code was tested periodically to ensure functionality of each function.

Testing was conducted by running the program a few times to ensure that it was stored in cache before taking readings. Then the program was run with different inputs, number of threads and sorting functions. The execution time for each case was measured for multiple runs of the program and averaged across them. This generates a single value for the execution time of the program with those parameters.

A number of jpg images were used for testing. These were of various sizes and are described in Table I. These are of note as the resulting execution times were directly related to the size of the image.

III. Results

All values of execution times are averages of at least 3 different measurements. The speed-up is calculated as the baseline (either the Golden Measure or Bubble Sort) divided by the new speed achieved. speed-up can be read as the compared value is speed-up times faster than the baseline.

A. Sorting Algorithms

The first comparison drawn was the difference in executing times of the median filter using different sorting functions. The results found are summarized in Table II.

TABLE I

IMAGE DIMENSIONS

	Height (Pixels)	Width (Pixels)	Pixels E+3
Greatwall.jpg	2560	1920	4915
Fly.jpg	1024	821	841
Alan.jpg	640	640	410
Small.jpg	304	300	91

TABLE II

SORTING ALGORITHM COMPARISONS

	Bubble	Select	std::sort
Fly.jpg (s)	25.23	7.75	4.66
Alan.jpg (s)	9.43	3.76	1.95
Average (s)	17.33	5.75	3.30
Speed-Up	1.00	3.01	5.25

The Select sort was significantly more efficient and the ability to stop halfway through sorting to get the median sped it up a lot but the select algorithm itself isn't particularly efficient so that slows it down. The std::sort method inbuilt to c++ is a hybrid function using much more efficient algorithms and therefore it is the fastest and was used in the final measurement for the multi-threaded implementation.

B. Thread count investigation

The speed-up was measured using different numbers of threads (thread count) and the results were plotted in Figure 1. There was a peak in efficiency somewhere around 512 threads. The values near this point did not show much fluctuation and thus this was chosen as the thread count that will be used for the Implementation comparison

C. Implementation comparison

The execution time for the Golden Measure and the Multi-threaded implementations were compared in Table III. This speed-up fluctuated with different image sizes. This is likely due to many factors interacting simultaneously and would require further investigation to isolate the cause. But the speed-up is 29.74 times on average across image sizes.

D. Median Filter

The Median Filter was successful and resulted in blurred images as shown in Figure 2. This is the amount of blurring that was expected from a 9x9 median filter so it appears that the filter itself works

IV. CONCLUSION

It was found in this report that the use of data partitioning and multithreading when implementing a 9 x 9 median filter for an image significantly reduces the execution time of the application. On average a speedup of 29.74 was found when implementing a multi-threaded implementation compared with a serial one. The investigation into which sorting algorithm was faster for use in a median filter concluded that a Qsort was the most suitable. Finally, it was found that the speedup

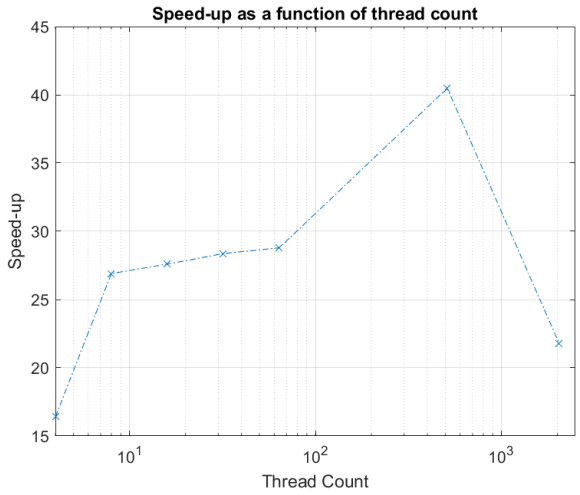

Fig. 1. Relationship between Thread Count and Speed-Up

TABLE III

GOLDEN MEASURE VS MULTI-THREADED COMPARISON

	Golden Measure (s)	Multi-Threaded(s)	Speed-Up
Greatwall.jpg	133,48	4.71	28.37
Fly.jpg	19.18	0.72	26.51
Small.jpg	1.33	0.04	34.34
Average			29.74

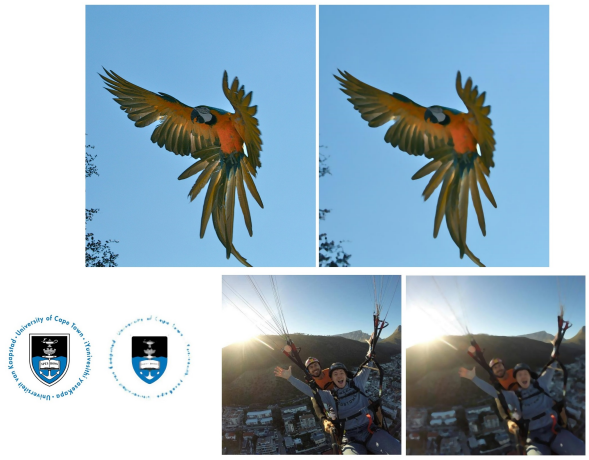

Fig. 2. Input Images and Output Filtered Images

of the implemented filter increased with thread-count up until a certain critical point where speedup starts to decrease with an increasing number of threads.

REFERENCES

[1] G. R. Arce, *Nonlinear Signal Processing: A Statistical Approach.* John Wiley and Sons, 2005.