pThreads: Partitioning Data In Median File Using Multi-Threaded Applications

Tawanda Muzanenhamo[†]
EEE4120F Class of 2019
University of Cape Town
South Africa
[†]MZNTAW004

***Abstract*—This report illustrates the use of C++ static pThreads to implement multi-threading on CPUs.A median filter on images of different sizes is used to test the implementation and report on the relationship between performance, number of threads and size of image.**

I. Introduction

Multi-Threaded applications are used to make programs run faster and more efficiently. This report focuses on the use of multi-threads to run the same algorithm on different sections of the same data.The results obtained through multi-threading which is implemented by partitioning data and executing tasks on the sections of the data independently. The results obtained are then compared to a less fast but more accurate golden measure and a relationship between image size, algorithm performance and number of threads is explored.

II. Methodology

A golden measure which doesn't include optimization was implemented using the sequential implementation of a median filter using the bubble sort algorithm. Bubble sort was chosen over a much faster quick sort algorithm because the goal of the golden measure is to obtain a more accurate result and it is easy to implement. The parallel implementation used static pThreads with different number of threads being used to evaluate performance.

A. Golden Measure

The golden measure goal was to find the pixel value for the output image through iterating each pixel in the output image of the same size as the input image. This was achieved by populating a 9x9 grid of pixels into an array. The array was then put through the bubble sort which was implemented using the code in Fig 1. In order to handle cases where the output pixel was near the edge thus making part of the 9x9 grid fell outside the image, the image was zero padded thus allowing the returned value of the pixel to be 0.

B. Median Filter pThreads Implementation

The pThread implementation of the median filter finds the tasks which can be executed independently whilst creating the 9x9 grid using the code in Fig. 3.The values in the 9x9 grid are then sorted using a quick sort in Fig. 2 and the median value is returned.The median value of the neighbouring elements in the

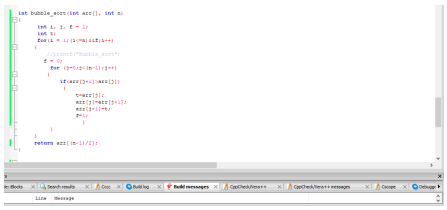


Fig. 1. Bubble Sort Implementation [**?**]

grid is then used to replace it's neighbouring values thereby blurring the image.Threads are used to run through sections of the partitioned sections image independently. A mutex was used to avoid race conditions whereby multiple threads try to access shared data and change it at the same time.

Fig. 2. Quick Sort Implementation [**?**]

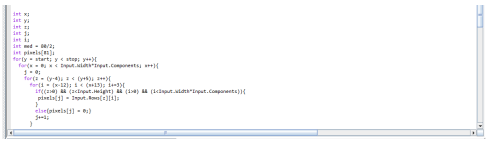


Fig. 3. Median Filter Implementation [**?**]

C. Correlation Of Output Images

Using the correlation function in Octave the output images from the golden measure implementation were correlated with the images from the parallel implementation and it was observed that the images had a correlation coefficient of almost 1 which validates the correctness of the parallel implementation relative to the golden measure which produced a result which was known to be correct.The correlation function is in Fig 4.

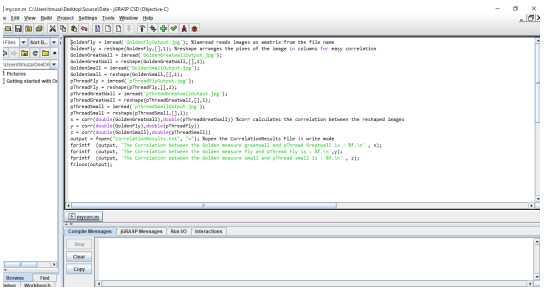


Fig. 4. Correlation Fucntion [**?**]

III. Results

A. Golden measure

The golden measure was implemented without optimization and the results obtained which were obtained are tabulated in Table I. The results obtained were accurate because a bubble sort was implemented and even though it is less fast than other sorting algorithms like quick sort it gives the most accurate results.

Picture name	Width	Height	Runtime
fly.jpg	821	1024	7119.35
greatwall.jpg	1920	2560	44796.2
small.jpg	300	304	511.45

TABLE I
GOLDEN MEASURE RESULTS

B. pThreads Results

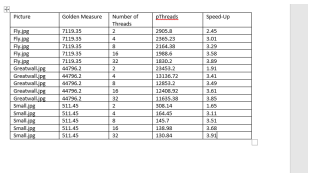
The pThreads implementation was executed with different number of threads and the results obtained were tabulated in table II.

Picture	Number of threads	Runtime
fly.jpg	2	2905.8
	4	2365.23
	8	2164.38
	16	1988.6
	32	1830.2
greatwall.jpg	2	23453.2
	4	13136.72
	8	12853.6
	16	12408.92
	32	11635.38
small.jpg	2	308,14
	4	164,45
	8	145,7
	16	138,98
	32	132,84

TABLE II
PTHREADS RESULTS.

C. Results Discussion and Validation

As illustrated by the results in Table II, there was an increase in the performance of the pThreads running on the images with bigger height and width. This conformed to the theory that the number of threads is directly proportional to the performance of the threads and this leads to a significant decrease in the execution time of the pThreads. The speedup (Ts/Tp) is also directly proportional to the number of threads as illustrated in Fig. 5. This is further proof that the performance of pThreads depends on the number of threads executed.



Picture	Golden Measure	Number of Threads	pThreads	Speed-Up
Fly.jpg	7119.35	2	2905.8	2.45
Fly.jpg	7119.35	4	2365.23	3.01
Fly.jpg	7119.35	8	2164.38	3.29
Fly.jpg	7119.35	16	1988.6	3.58
Fly.jpg	7119.35	32	1830.2	3.89
Greatwall.jpg	44796.2	2	23453.2	1.91
Greatwall.jpg	44796.2	4	13136.72	3.41
Greatwall.jpg	44796.2	8	12853.2	3.49
Greatwall.jpg	44796.2	16	12408.92	3.61
Greatwall.jpg	44796.2	32	11635.38	3.85
Small.jpg	511.45	2	308.14	1.66
Small.jpg	511.45	4	164.45	3.11
Small.jpg	511.45	8	145.7	3.51
Small.jpg	511.45	16	138.98	3.68
Small.jpg	511.45	32	130.84	3.91

Fig. 5. Speed-Up Results [**?**]

The Golden measure image output and the pThreads image output were correlated with each other using the correlation function in Fig. 4 and it was observed that the correlation coefficent was close to 1 as shown in Fig. 6. This conformed to our expected results that the image outputs are supposed to look alike if the algorithms are executed correctly only the execution time will differ.

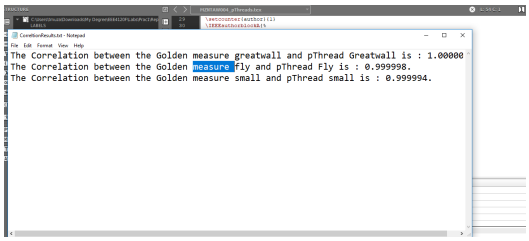


Fig. 6. Correlation Results [**?**]

IV. Conclusions

The findings of this report show that the pThreading implementation executed much faster than the golden measure which was executed sequentially. The speed of the pThreads is also affected by the number of threads running as since the speed was slower for less threads and faster for more threads. The speed of execution of the pThreads was also influenced by the fact that there are no critical sections thus the threads don't have to wait to access sections of the image.

References

[1] Article name:Image Correlation, Convolution and Filtering Author: Carlo Tomasi Site: https://www2.cs.duke.edu
Article name: Image Filtering Site: https://www.cs.auckland.ac.nz