Assignment 3:
Image Enhancement via Client-Server Communication

Paul Prince

Re-Submitted Sun. 2010-04-25

CS350 - Dr. Don Adjeroh, Spring 2010

Enhanced Image

Another Enhanced Image

Original Image

Mean Image

Median Image

Variance Image

(Standard Deviation)

Introduction

Target Platform

I try to write portable code, however there may be dependencies specific to a Linux operating environment. I also test all submissions on the WVU CSEE shell server. Consult the included files “Makefile” and “run_tests.sh” for assistance building and running the programs.

Image Formats

Currently, the code still supports only PGM greyscale images, i.e. PBM Type 5 images.

If time permits, I may extend it to additional formats before final submission.

Image Processing Algorithms

Window Selection

Same as in Assignment 2:

We attempt to select a square window centered around a given center pixel. If any of the edges of this window fall outside the boundaries of the input image, we crop off the overhanging portions of the window.

Standard Deviation Calculations

Again, same as in Assignment 2, Knuth & Welford's “online” algorithm for calculating population variance in a single pass over the items.

Median Calculations

As yet, unchanged from Assignment 2: use the standard library's *qsort()* routine to find medians naïvely.

This is an aspect of the program I hope to improve before final submission. However, the current method should be numerically accurate, even if inefficient.

Enhanced Value Calculation

Counter to the apparent “conventional wisdom” regarding this assignment, I am currently calculating the enhanced pixel value in the child threads, and not in the parent/main thread.

The constant values have been updated according to Assignment 3.

I will modify this behavior if required.

Global Statistics

I notice on my grade sheet from Assignment 2 that I lost a few of points in the global statistics portion. I am reviewing my code to ensure that any errors are corrected before final submission of Assignment 3.

Changes from Assignment 2

im_shared.c

I intend to produce a multi-process version of the assignment as well, for the bonus credit. As a result, I have moved everything that I think can be shared between the multi-process and multi-threaded versions in *im_shared.c*

Communication Between Threads

Due to the nature of the assignment (and that awesome hint from Dr. Adjeroh after class on 4/15), I have avoided many things that might have been difficult:

- Most thread-shared variables are read-only during the execution of multiple threads.
- Each thread writes only to distinct elements in the output arrays.

As a result, I have not had to implement any locking, etc.. And I don't think I have any deadlocks or race conditions, but I could always be wrong!

Child Thread Entry Point / Main Loop

Execution for the child threads begins in *iterate_input_image()*:

```
/* Iterate over the input image, populating the others along the way. */
void *iterate_input_image(void *arg){
        int i, j, c; /* loop counters */
        double window_mean, window_variance, window_median, window_stddev;

        int child_id = get_child_id();

        c = 0;
        for (i=0; i<rows; i++){
                for (j=0; j<cols; j++) {
                        c++; /* fixes bug #002 */
                        if (c % num_threads == child_id) {
                                /* Choose the subregion of interest (window) around this pixel. */
                                window_t window = select_window(window_size, i, j, rows, cols);

                                /* Calculate the median of just the window. */
                                windowcalc_median(Image, window, &window_median);

                                /* Calculate the mean and variance of just the window. */
                                windowcalc_mean_and_variance(Image, window, &window_mean, &window_variance);

                                /* Calculate standard deviation of just the window. */
                                window_stddev = sqrt(window_variance);

                                /* Calculate the enhanced value of this pixel. */
                                int window_enhanced = calc_enhanced( Image[i*cols+j],
                                                        mean, stddev, window_mean, window_stddev );

                                /* Assignments to the output images. */
                                Mean_Image[i*cols+j]     = window_mean;
                                Median_Image[i*cols+j]   = window_median;
                                Enhanced_Image[i*cols+j] = window_enhanced;
                                Variance_Image[i*cols+j] = window_stddev;
                        }
                }
        }

        return NULL;
}
```

Bonus Problem *(added 2010-04-25)*

Overview

I have created a multi-process version of the assignment, using pipes for communication, as *pimEnhance.c*.

Both programs (pimEnhance and previously-submitted timEnhance) can be exercised side-by-side by:

1) placing some PGM images (named **.pgm*) into the *test_input/* directory, and then
2) running *make test*

My Expectations

I anticipated that the threaded version would be faster than the multi-process version, and that both of these would be faster than the single-threaded version, at least when:

1) The programs are being executed on a multi-processor machine, and
2) The number of parallel jobs is equal to or slightly greater than the number of processor cores available.

To collect the timing data, I ran the programs on my Fedora Linux laptop. This machine has a dual-core Intel Core 2 Duo P8600 CPU. Therefore, I expected the programs to be fasted when executed using 2 or 3 parallel jobs.

Actual Results

See graphs on next page.

The threaded version of the program fit my expectations well; the fastest run times were found using 2 threads, and the times were much faster than the single-threaded Assignment 2.

My threaded and multi-process versions support num_threads = 1, and this example is represented in the graphs; if you look closely you can see that there is indeed some thread overhead in the one-thread case compared to the native single-threaded version.

I was suprised to find the multi-process version slower than my Assignment 2 program in all cases. Run times are similar (but consistently slower) up to about 10 processes; after this, the per-process overhead begins to appear as a significant factor.

I suspect that I am doing something inefficiently in my multi-process version, but I have not yet determined what it is.

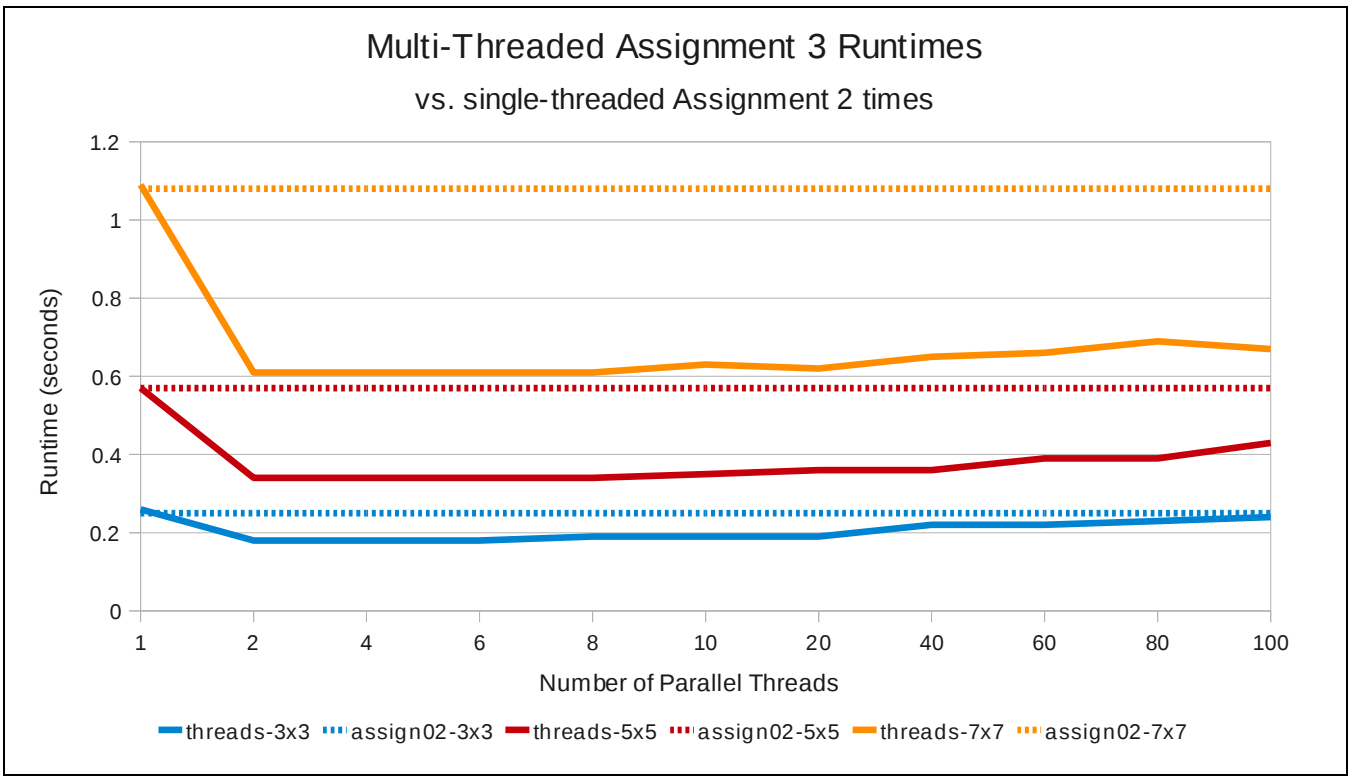
Effect of num_threads and window_size on Results

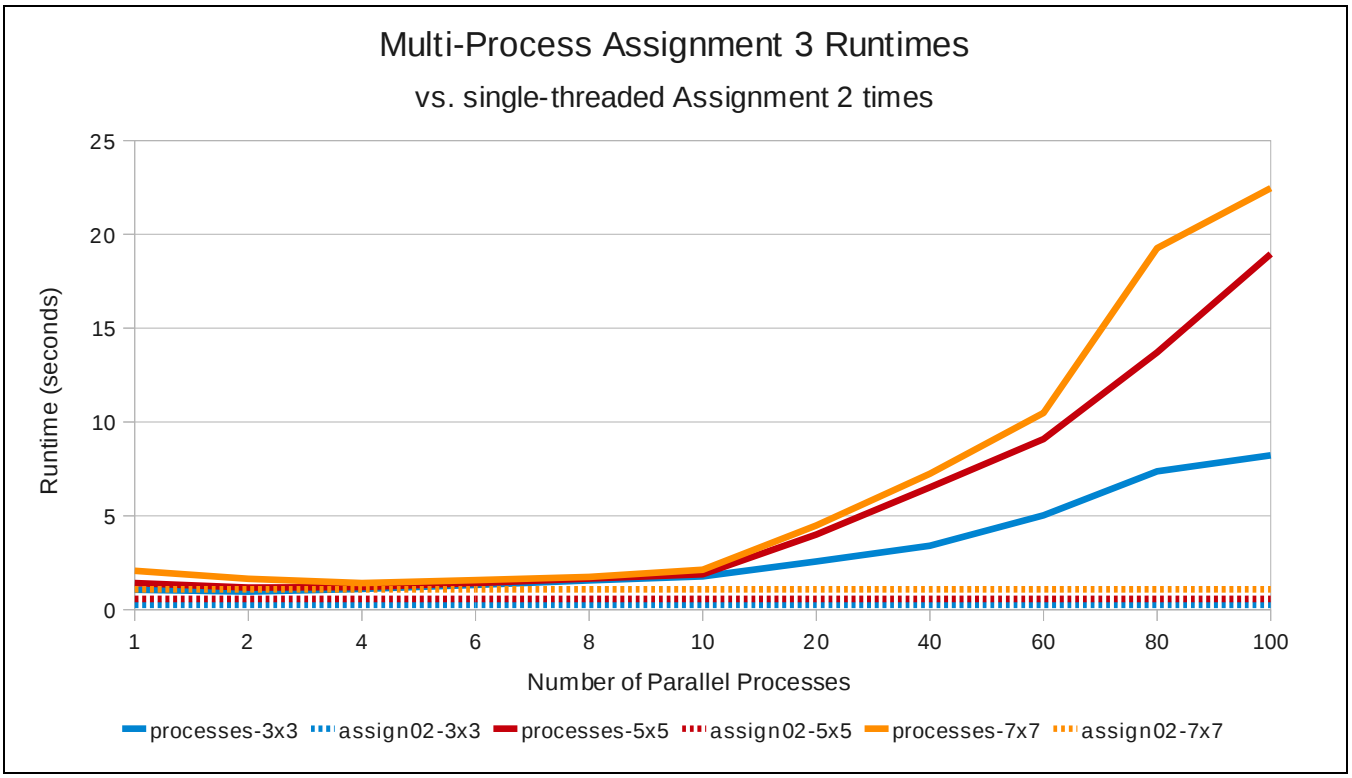
Modifying the number of parallel jobs should never result in a change in the output.

However, the window size has a noticeable effect on all of the transformations (mean, variance, etc.)

The median transformation is particularly fun to play with at large window sizes.

Runtime Graphs





Runtime Table

bg448x448.pgm

num_threads	threads-3x3	threads-5x5	threads-7x7	processes-3x3	processes-5x5	processes-7x7
1	0.26	0.57	1.09	1.06	1.41	2.07
2	0.18	0.34	0.61	0.96	1.17	1.65
4	0.18	0.34	0.61	1.11	1.2	1.42
6	0.18	0.34	0.61	1.32	1.41	1.57
8	0.19	0.34	0.61	1.56	1.66	1.75
10	0.19	0.35	0.63	1.78	1.89	2.13
20	0.19	0.36	0.62	2.57	4	4.49
40	0.22	0.36	0.65	3.41	6.52	7.24
60	0.22	0.39	0.66	5.02	9.09	10.48
80	0.23	0.39	0.69	7.37	13.71	19.27
100	0.24	0.43	0.67	8.22	18.96	22.46

assign02-3x3	assign02-5x5	assign02-7x7
0.25	0.57	1.08