Image Processing Using Multiple GPUs on Webcam Image Streams

Hannah Munoz[†], Sergiu M. Dascalu[†], Rui Wu[§], Lee Barford[†‡], Frederick C Harris, Jr.[†]

[†]*Dept. of Computer Science and Engineering*
University of Nevada, Reno
Reno, NV

[§]*Dept. of Computer Science*
East Carolina University
Greenville, NC

[‡]*Keysight Laboratories*
Keysight Technologies
Reno, NV

hannahmunoz@nevada.unr.edu, {dascalus,fred.harris}@cse.unr.edu, wur18@ecu.edu, lee.barford@ieee.org

***Abstract*—Image analysis is an important area in many fields of research, such as sensor networks, where webcams have become an increasingly popular addition. Sensor network webcams gather images frequently and, as such, a need for processing large image streams has occurred. In this paper, we test a variety of OpenCV functions on image streams from sensor networks on CPU, GPU and multiple GPU to find the most efficient way of processing image series. OpenCV is a popular computer vision and image processing library. Although OpenCV supports GPU functionality, the library's multiple GPUs functions are lacking. Two image processing algorithm for snow and cloud detection were developed using OpenCV and applied to an image series. The execution times, speedup, and throughput of the different implementations are compared and discussed. From this, The best execution method for various situations is determined.**

***Index Terms*—GPU programming, Remote sensing, Digital images, Image analysis**

I. Introduction

Analysis of image series is used in various research areas, from vehicle speed monitoring to land coverage recognition via satellites [1], [2]. Remote sensing is a vastly expanding field that has begun to welcome webcams as a cheap solution to collecting a different type of data from their research sites [3]. Webcams utilized in remote sensor networks can acquire large amounts of images by taking pictures minute or hourly, making it hard for a person to analyze image streams in a timely manner. This project aims to find the best method of performing image processing using OpenCV on high volumes of low resolution images. By varying the number of images in the stream, and the size of the images, we can determine the best method of processing image streams in minimal time.

The Nevada Research Data Center (NRDC) manages the data from several remote sensor network projects in Nevada [4]. Most of the supported projects deploy web cameras in their research sites. The Walker Basin Hydroclimate was chosen as our data set due to the southern camera's clear view of the sky and vast portion of land without sagebrush or forest [4]. The Walker Basin Hydroclimate project began in 2012 and collects images hourly. The project has collected over 17,000 landscape images in the Western Great Basin. We developed snow and cloud estimation algorithms. Each algorithm was applied to a small section within an image, called a regions of interest (ROI).

The Nevada Research Data Center (NRDC) manages the data from several remote sensor network projects in Nevada [4]. Six of the research sites deploy web cams as a type of sensor. The Walker Basin Hydroclimate Project's Rockland Summit research site was chosen as our data set due to the southern camera's clear view of the sky and vast portion of land without large sagebrush or forest obscuring the view [4]. Rockland Summit has 20 different camera angles, using pan-tilt-zoom presets in a Canon VB-H41 on site. The camera is set up to take HD pictures in 60-minute intervals from 10 AM PST to 5 PM PST. Images are taken in JPEG format with a resolution of 960 x 540. The southwest camera angle that was chosen has the best field of view for our use and does not have any sun interference. By using image analysis to track snow and cloud coverage in an area, time spent analyzing collected data can be reduced. The project has collected over 17,000 landscape images in the Western Great Basin.

Image processing is a method of analyzing and changing images [5]. Commonly, it is used for improving an image's quality. OpenCV is a computer vision/image processing library for various language and operating systems [6]. With over 2,500 available algorithms and 2.5 million downloads, OpenCV is widely used in many types of computer vision projects [7]. OpenCV uses matrices as a way of storing images, where each element of the matrix is a pixel in an image. Most computer vision algorithms work by analyzing pixel intensity values in a image. A pixel's intensity it's color value. Usually, a digital image has three color channels, red, blue, and green, each of which has its own intensity value. Since images can have thousands of pixels, computation on large images can become costly. However, OpenCV's algorithms have been highly optimized and as such have significantly outperformed other popular computer vision libraries [8].

In 2011, OpenCV introduced Graphical Processing Unit (GPU) accelerated algorithms [9]. Current goals of OpenCV GPU project is to provide a GPU computer vision framework consistent with the CPU functionality, achieve high performance with the GPU algorithms, and to complete as many algorithms as possible so image analysis can be done

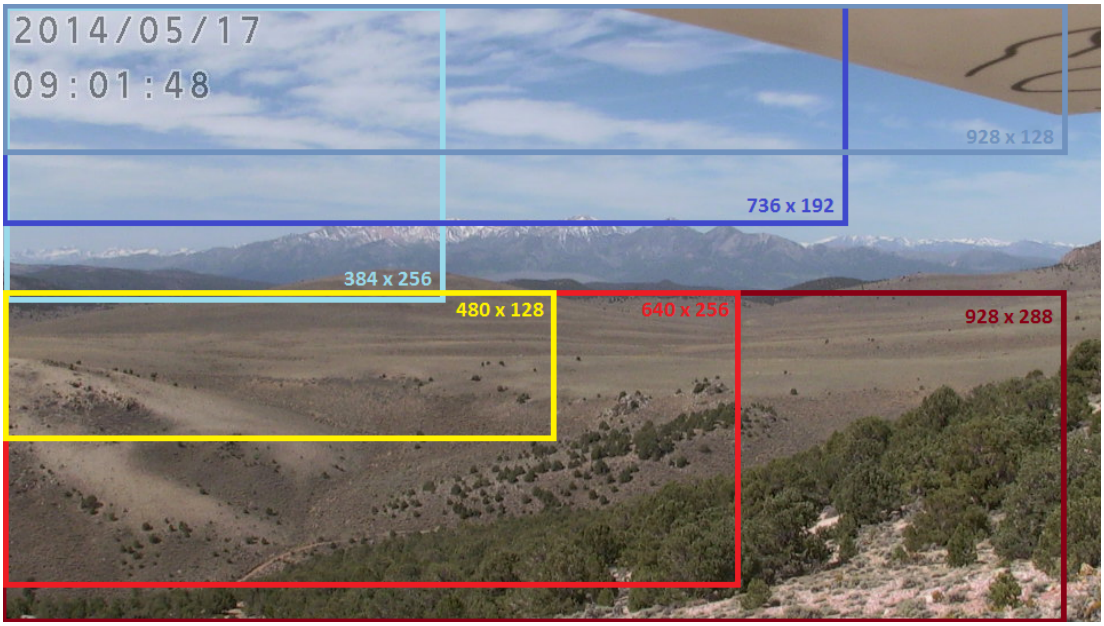


Fig. 1: The various region of interest on which the cloud and snow algorithms were applied.

completely on the GPU [9]. OpenCV's GPU implementation was written using CUDA, so developers could take advantage of previously developed CUDA libraries.

Algorithms developed for GPU often differ than algorithms implemented on CPU. GPUs can do hundreds of independent calculations simultaneously, which lends to immense speedup over CPU implementations. Because of the architecture differences between CPU and GPU, there is no guarantee that algorithms implemented on the GPU will necessarily be faster than those done on the CPU [10]. There can be many factors, such as image size or algorithm technique, that could increase execution time on GPU compared to CPU. If there are any dependencies between pixels in an algorithm, such as a blurring mask, the time spent trying to sync pixel values could attribute to low performance [11].

There is currently no multiple GPU (multi-GPU) support for the GPU algorithms beyond a method to manually switch GPU devices [12]. OpenCV does not recommend using multiple GPUs on smaller images, as added overhead of data transfers between GPUs could negate any speedup achieved from using GPU. However, there is no mention on whether multiple GPUs could attain speedup on streams of images

In this paper, we compare execution times, speedup factors, and throughput between various implementations of low-level image processing algorithms on image series. This paper is structured as follows: Section II review work previously done on the subject; Section III discusses the detection algorithms applied to the image series; Section IV discusses differences between CPU, GPU, and multiple GPU implementations; Section V discusses results; and Section VI contains our conclusions and goals for future development.

II. Related Works

Hwang et. al's work in real-time image processing on high-resolution images shows that by offloading image computation to the GPU, images can be analyzed for simple operations in real time [13]. By simply thresholding pixel differences against a given value, items in the foreground of an image can be detected. Image pre-processing is done on the CPU before the image is sent to the GPU for further analysis using OpenCV. No run time comparison on the proposed algorithm were done. In contrast, our project compares and discusses execution time between CPU and GPU implementations of image processing techniques on image streams.

Agrawal et. al's paper implements a GPU version of a saliency model performing in real time [14]. Saliency is a pre-processing technique which must be done quickly. On CPU, the processes is slow, however, by exploiting explicitly paralleled parts of the method onto the GPU, a speedup by a factor of almost 600 was achieved. Agrawal's et. al's implementation outperformed OpenCV by a speedup factor of almost 300. This was only done for a single image, however, not a stream of images as our results show.

Wang et. al's work proposes a multi-GPU accelerated version of two popular algorithms for satellite image processing [15]. They discuss the difference between two popular methods of doing multi-GPU image processing. Namely, multiple image at once or division of a single image among GPU. They compared CPU implementation to GPU and multi-GPU. Speedups of over 100 were achievable on both proposed algorithms. Our project tries to develop similar, but with much lower resolution images.

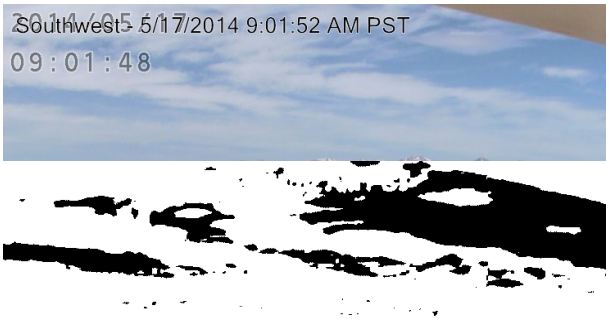


Fig. 2: A comparison of the cloud coverage algorithm's output.

Fig. 3: A comparison of the snow coverage algorithm's output.

III. Algorithms

Two low-level detection algorithms for snow and cloud coverage were developed for testing. Figure 1 shows an example image from our data set that the cloud and snow algorithms were developed on, as well as the regions of interest chosen for testing.

A. Cloud Coverage

Algorithm 1 Cloud Coverage Algorithm

1: Split Image into BGR Color Streams
2: **for** Pixel i in Image **do**
3: **if** $Blue_i$ - $Red_i > Threshold_c$ **then**
4: i is Cloud
5: **else**
6: i is Sky
7: **end if**
8: **end for**

The cloud coverage algorithm was used as our low computation test using only one color channel: blue, green, red (BGR). As such, it does very little calculations to determine cloud coverage. A comparison between a ROI from the data set and the results of the algorithm are in Figure 2. Algorithm 1 was based off a hybrid threshold algorithm previously developed by Li [16]. Little computational work is done in determining cloud coverage, however it was determined to be is very accurate with a threshold value of 32. Lines 3 and 4 of Algorithm 1 are completely independent and can be run simultaneously on GPU causing speedup.

B. Snow Coverage

The snow coverage algorithm requires converting color streams, blurring, and several elemental operations making its execution a bit more intensive than the cloud algorithm. Because some of these operations rely on pixels related to them, the actions cannot always be done in parallel. The snow detection algorithm was based off work previously done by Salvatori [17]. In the version implemented $Thresold_l$ is set to 20 and $Thresold_b$ is set to 127. Figure 3 shows how the output of Algorithm 2 compares to the original image.

Salvatori's algorithm did not have high accuracy results when tested on our dataset. This may be because the image set tested in Salvatori's project was from mountainous regions, where as our image set is from a desert. Salvatori's algorithm had problems distinguishing snow from sand.

Algorithm 2 Snow Coverage Algorithm

1: Split Image into RGB Color Channels
2: Convert Image to HLS Color Space
3: Split Image into HLS Color Channels
4: Gaussian Blur (5x5) Hue Channel
5: **for** Pixel i in Image **do**
6: **if** $Hue_i > Threshold_h$ **then**
7: **if** $Blue_i > Threshold_b$ **then**
8: i is Snow
9: **end if**
10: **else**
11: i is Ground
12: **end if**
13: **end for**

We were able to fix this problem by adding a test of another color channel. The difference in pixel intensity in the hue, saturation, and luminance (HSL) color space for snow and sand pixel was significant enough to distinguish between the two. This helped improved the accuracy of our snow coverage algorithm.

Although some functions must have algorithm changes to increase performance of the GPU, our implemented CPU algorithms did not need to be changed. The only difference between implementations is that the GPU algorithms have an added overhead of transferring the image to and from the graphics card. All developed algorithms were made to be as similar to the CPU implementation as possible. This allowed for more accuracy between comparisons of the algorithms. OpenCV's design allows for an almost direct correlation between its CPU and GPU code. Figure 1 shows regions of interest chosen for testing.

IV. Implementations

A. Hardware

The algorithms were applied on a Ubuntu 16.04 machine with eight GeForce GTX 1080 (Pascal Architecture v6.1) graphics cards, two Intel Xeon Processors E5-2650 CPU and 64GB of RAM. Each card has 8114 MiB of memory,

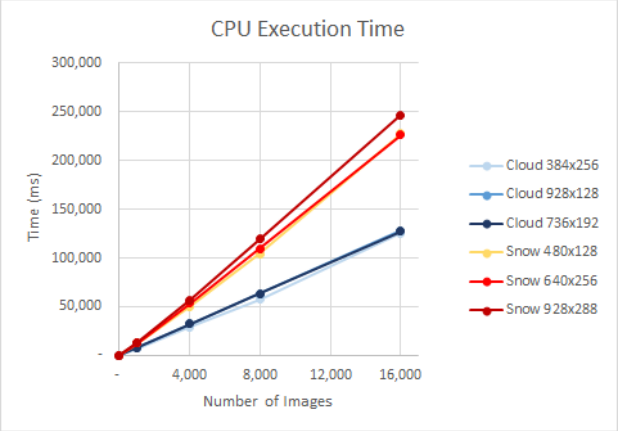


Fig. 4: The execution time in milliseconds of both the cloud and snow algorithm run on the CPU.

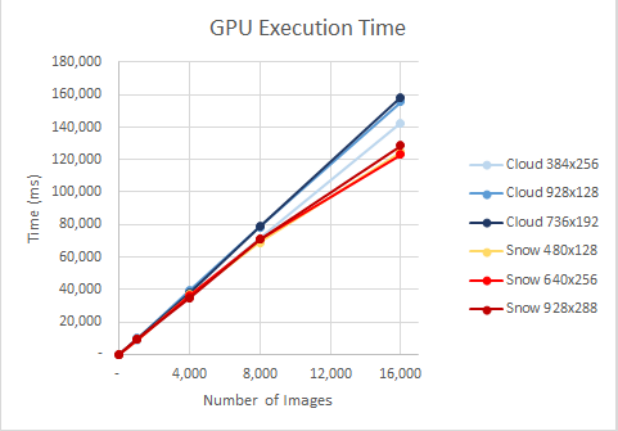


Fig. 5: The execution time in milliseconds of both the cloud and snow algorithm run on the GPU.

65536 bytes of constant memory, and 49152 bytes of shared. Concurrent copying and execution is enabled with 2 copy engines running.

B. CPU

The CPU implementation for Algorithm 1 and Algorithm 2 are applied during separate runs to each image in the series. Image sizes and number of images in a series were varied over five runs and an average execution time was calculated in Figure 4.

Both algorithms run in linear time; as the number of images in a series increases so does the execution time. The image size does affect execution time, as seen from the increase in execution times in Cloud 736x128 and Snow 928x288, but not until a significant amount of images are added to the series. Algorithm 2 has higher execution times than Algorithm 1 because there are more steps that need to be done during it.

C. GPU

Similarly, the GPU algorithm was implemented using OpenCV's GPU algorithm. In this implementation, after the image is read in from the file, it must be "uploaded" to the GPU. This take a significant amount of time initially to set up. To try to alleviate the initial overhead, a 1x1, single channel GPU matrix is created before processing the image series. This reduced the execution time of a single image using Algorithm 2 from an average of 755.822 ms to 23.356 ms; 32 times speedup. Once either algorithm has been applied to the image series, the resulting image is "downloaded" from the GPU to the CPU and the coverage percentage is pushed to a vector. The average execution time over 5 runs can be seen in Figure 5.

When moved to the GPU, OpenCV's matrices are divided up into $(c+15)/15$ by $(r+15)/15$ grids where c and r are the columns and rows in the image. Each block in the grid has 16 threads in both the x and y direction, resulting in 256 threads per block. Since 256 is divisible by 32, it takes advantage of the GPU's warp scheduling to increase speedup.

One the GPU, all the execution times begin to converge together. The snow coverage algorithm has a significant amount of speedup. However, the cloud coverage algorithm actually takes longer on the GPU than it did on the CPU. This is because the added overhead of sending the images to the GPU and back takes more time than execution on the GPU could decrease. In other words, there is not enough computations done to justify sending it to the GPU.

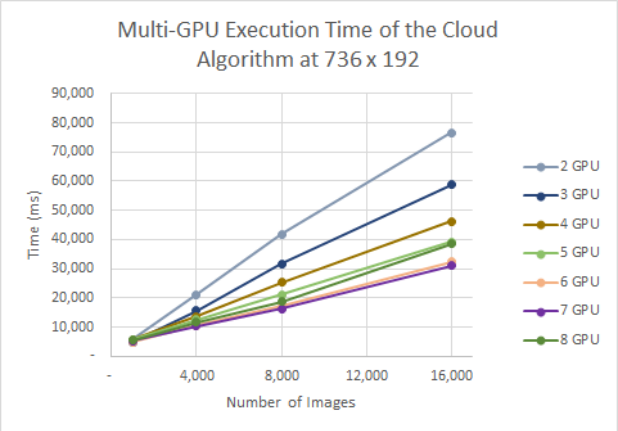
D. Multiple GPU

The multiple GPU implementation utilizes threads to control multiple GPUs at once. Each thread is given all the file names of the images in the series and the ROI's height and width. The number of images in the series is divided among available GPUs. Each thread then runs the previously implemented single GPU algorithms on their own GPU stream. This allows for all image analysis to happen concurrently and images are uploaded and downloaded from the GPU asynchronously. The average execution time over 5 runs can be seen in Figure 6. For simplicity's sake, Figure 6 shows the largest of the ROI run for both algorithms. The full data set, however, is available [18].

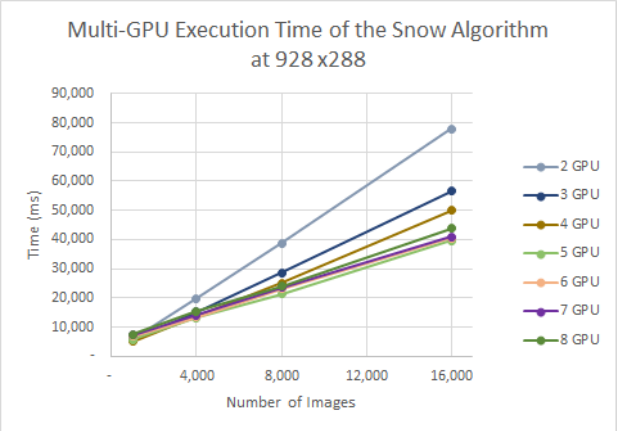
As the number of GPUs increases, the execution time decreases. Even in the cloud detection algorithm, which did not achieve speedup when run on a single GPU, decreases its run time when more GPUs are added. Once 8 GPUs are added, however, execution time begins to increase again. Once again, the issue is the overhead resulting from sending the images to the GPUs. The amount of work being done on each GPU is not efficient and the images are going to so many different GPUs that they are bogging down the PCIe bus that connects the GPUs together.

V. Results

While Algorithm 1 does not increase speedup on a single GPU, speedup is gained during multi-GPU computing. Figure 7a shows the speedup factor of the 736x192 ROI over 1 to 8 GPUs. For GPUs 2 through 7 a steady increase of speedup occurs. At 8 GPUs, speedup begins to decrease. A discussed previously, this is because there is too much data transfer happening between GPUs.

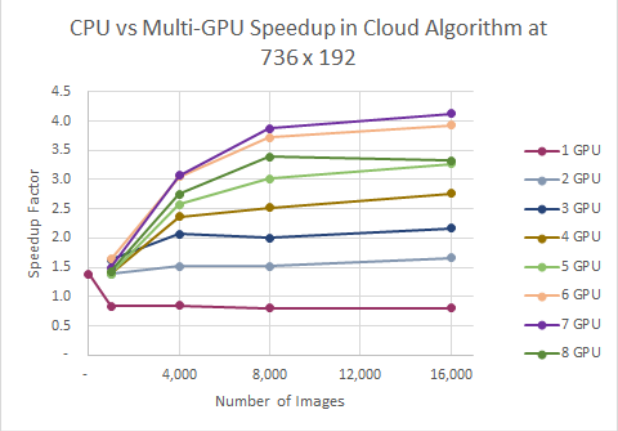


(a) The cloud algorithm with 736x192 ROI.

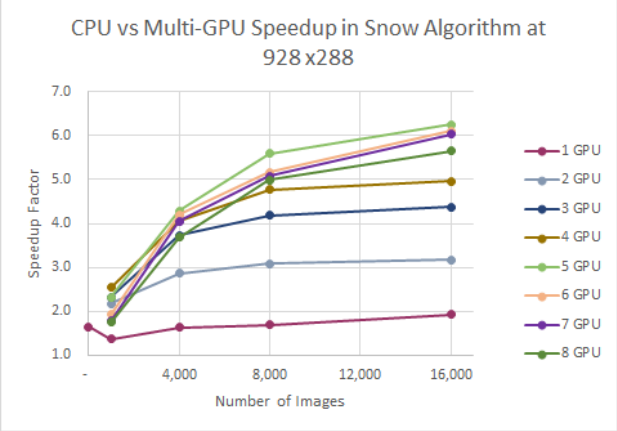


(b) The snow algorithm with 928x288 ROI.

Fig. 6: The execution time in milliseconds of both the cloud and snow algorithm run on the GPU.

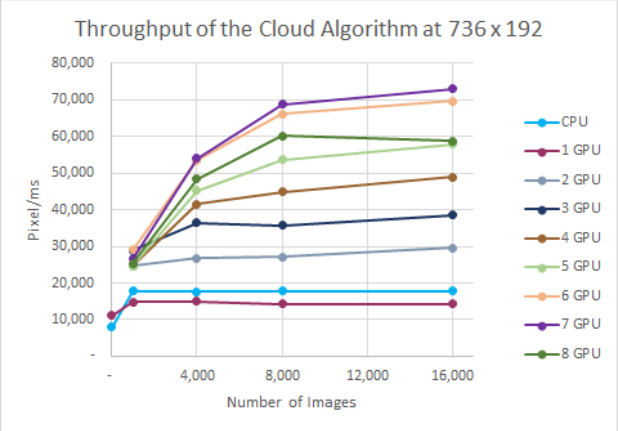


(a) The cloud algorithm with 736x192 ROI.

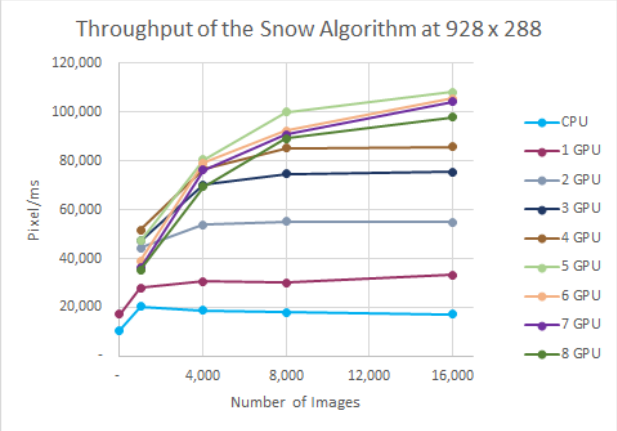


(b) The snow algorithm with 928x288 ROI.

Fig. 7: The speedup factor of the largest ROI run on CPU, GPU, and multiple GPUs.



(a) The cloud algorithm with 736x192 ROI.



(b) The snow algorithm with 928x288 ROI.

Fig. 8: The throughput of the largest ROI run on CPU, GPU, and multiple GPUs.

Algorithm 1's throughput in Figure 8a is quite clear. As the number of GPUs increases, up to 7 GPUs, the throughput increases as well.

Algorithm 2's speedup graph in Figure 7b is a little different. A steady increase in speedup factor occurs from the use of 1 to 5 GPUs, not 7 GPUs. This is because of the extra work load that needs to be done to calculate snow coverage in the region. At a smaller number of images, anything less than 4,000, it becomes confusing to determine the most efficient number of GPU to use. The speedup for such a small amount of images does not increase with more GPUs. This could be because the machines the algorithms were run on were either closer or farther from each other, thus affecting execution time.

Algorithm 2's throughput in Figure 8b is similar, in that it reflects trends previously discussed in its speedup graph. The throughput also increases as more images are added to the series, because large amounts of images can be more efficiently handled on more GPUs.

Figure 7b displays a superlinear speedup characteristic of Algorithm 2. This is an expected characteristic, as OpenCV algorithms can have up to 100x speedup over their original implementation [9].

VI. Conclusion and Future Work

OpenCV is a popular image processing and computer vision library. Although they have a expansive GPU-based algorithms library, they have yet to implement algorithms that utilize multiple GPUs. In this paper, we have implemented low and high work intensity image processing algorithms on a large image series to determine when it's appropriate to do image processing on CPU, single GPU, or multiple GPUs. It was determined that high work intensity algorithms on image series could achieve speedup on single GPU, however low intensity algorithms could not. Both algorithm types could achieve speedup using multiple GPUs with an increase in throughput with larger image series. There is, however, a maximum amount of GPUs that can be used before speedup begins to decrease.

Further work could be done to determine if multiple GPUs should be used to analysis images in real time from a video stream. The algorithms developed in this paper could be tested on live cameras to see if they can calculate coverage in real time. This could be helpful for use in citizen science projects, where citizens can use their own webcams to help scientists in their research.

Acknowledgments

This material is based in part upon work supported by the National Science Foundation under grant number IIA-1301726. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] J. Gerát, D. Sopiak, M. Oravec, and J. Pavlovicová, "Vehicle speed detection from camera stream using image processing methods," in *Proceedings of ELMAR-2017 - 59th International Symposium ELMAR-2017*, Sept 2017, pp. 201–204.

[2] Z. N. Absardi and R. Javidan, "Classification of big satellite images using hadoop clusters for land cover recognition," in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, Dec 2017, pp. 0600–0603.

[3] N. Jacobs, W. Burgin, N. Fridrich, A. Abrams, K. Miskell, B. H. Braswell, A. D. Richardson, and R. Pless, "The global network of outdoor webcams: Properties and applications," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS '09. New York, NY, USA: ACM, 2009, pp. 111–120. [Online]. Available: http://doi.acm.org/10.1145/1653771.1653789

[4] "Nevada research data center." [Online]. Available: https://sensor.nevada.edu/NRDC/

[5] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*. Cengage Learning, 2014. [Online]. Available: https://books.google.com/books?id=QePKAgAAQBAJ

[6] "Opencv library." [Online]. Available: https://opencv.org/

[7] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, "A brief introduction to opencv," in *Proceedings of the 35th International Convention MIPRO*, May 2012, pp. 1725–1730.

[8] S. Matuska, R. Hudec, and M. Benco, "The comparison of cpu time consumption for image processing algorithm in matlab and opencv," in *Proceedings on 9th International Conference - 2012 ELEKTRO*, May 2012, pp. 75–78.

[9] "Cuda." [Online]. Available: https://opencv.org/platforms/cuda.html

[10] V. Saahithyan and S. Suthakar, "Performance analysis of basic image processing algorithms on gpu," in *2017 International Conference on Inventive Systems and Control (ICISC)*, Jan 2017, pp. 1–6.

[11] J. Ke, T. Bednarz, and A. Sowmya, "Optimized gpu implementation for dynamic programming in image data processing," in *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, Dec 2016, pp. 1–7.

[12] "Gpu module introduction." [Online]. Available: https://docs.opencv.org/2.4/modules/gpu/doc/introduction.html

[13] S. Hwang, Y. Uh, M. Ki, K. Lim, D. Park, and H. Byun, "Real-time background subtraction based on gpgpu for high-resolution video surveillance," in *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication*, ser. IMCOM '17. New York, NY, USA: ACM, 2017, pp. 109:1–109:6. [Online]. Available: http://doi.acm.org/10.1145/3022227.3022335

[14] R. Agrawal, S. Gupta, J. Mukherjee, and R. K. Layek, "A gpu based real-time cuda implementation for obtaining visual saliency," in *Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing*, ser. ICVGIP '14. New York, NY, USA: ACM, 2014, pp. 1:1–1:8. [Online]. Available: http://doi.acm.org/10.1145/2683483.2683484

[15] M. Wang, L. Fang, D. Li, and J. Pan, "Using multiple gpus to accelerate mtf compensation and georectification of high-resolution optical satellite images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 10, pp. 4952–4972, Oct 2015.

[16] Q. Li, W. Lu, and J. Yang, "A hybrid thresholding algorithm for cloud detection on ground-based color images," *Journal of atmospheric and oceanic technology*, vol. 28, no. 10, pp. 1286–1296, 2011.

[17] R. Salvatori, P. Plini, M. Giusto, M. Valt, R. Salzano, M. Montagnoli, A. Cagnati, G. Crepaz, and D. Sigismondi, "Snow cover monitoring with images from digital camera systems," *Italian Journal of Remote Sensing*, vol. 43, 06 2011.

[18] "Project timings." [Online]. Available: https://github.com/hannahmunoz/\\MultiGPUOpenCV/blob/master/FinalRuntimes.xlsx