Stitching giga pixel images using parallel computing

Rob Kooper*[a], Peter Bajcsy[a], Néstor Morales Hernández[b]
[a]National Center for Super Computing Applications
University of Illinois, 1205 W. Clark St., Urbana, IL 61801
[b]Departamento de Ingeniería de Sistemas y Automática y Arquitectura y Tecnología de Computadores (ISAATC), 38206
University of La Laguna, Spain

ABSTRACT

This paper addresses the problem of stitching Giga Pixel images from airborne images acquired over multiple flight paths of Costa Rica in 2005. The set of input images contains about 10,158 images, each of size around 4072x4072 pixels, with very coarse georeferencing information (latitude and longitude of each image). Given the spatial coverage and resolution of the input images, the final stitched color image is 294,847 by 269,195 pixels (79.3 Giga Pixels) and corresponds to 238.2 GigaBytes. An assembly of such large images requires either hardware with large shared memory or algorithms using disk access in tandem with available RAM providing data for local image operation. In addition to I/O operations, the computations needed to stitch together image tiles involve at least one image transformation and multiple comparisons to place the pixels into a pyramid representation for fast dissemination. The motivation of our work is to explore the utilization of multiple hardware architectures (e.g., multicore servers, computer clusters) and parallel computing to minimize the time needed to stitch Giga pixel images.

Our approach is to utilize the coarse georeferencing information for initial image grouping followed by an intensity-based stitching of groups of images. This group-based stitching is highly parallelizable. The stitching process results in image patches that can be cropped to fit a tile of an image pyramid frequently used as a data structure for fast image access and retrieval. We report our experimental results obtained when stitching a four Giga Pixel image from the input images at one fourth of their original spatial resolution using a single core on our eight core server and our preliminary results for the entire 79.3 Gigapixel image obtained using a 120 core computer cluster.

Keywords: Image stitching, parallel computing, gigapixel images

1. INTRODUCTION

There have been significant advancements in imaging instruments, and their dissemination and adoption in many scientific areas. The increase in pixel counts generating by various imaging instruments pose challenges for storage, integration, search, and image understanding. In addition, the cost per pixel has been steadily decreasing which makes the challenges widely spread across many consumers of images. For example, as of October 2010, the following cost per pixel can be derived from the commonly accessible imaging devices: (a) Scanners (~$165) such as Canon's CanoScan 8800F Flatbed Scanner, generate 4800 x 9600 dpi Resolution (46.1 Megapixel) for a letter size scan area; (b) Cameras (~$100) such as Kodak's EasyShare C195 generate 14 Megapixel or Samsung's camera 12.2 Megapixel; (c) Microscopes (~$100) such as Avangard's Optics AN-E500 eScope generates 2.0 Mega Pixel; (d) Telescopes (~$170) such as COSTAR's 8x32 Digital Binoculars Telescope generates 2 M Pixel Pixel. The availability of advanced imaging instruments and the low cost per pixel leads to questions about efficient storage, processing and dissemination of Giga and Tera pixel images. Our investigation aims at understanding of computational requirements and algorithmic approaches for multi-core and computer cluster hardware architectures.The focus of our work is on the problem of stitching thousands of 16Mega pixel image tiles to a 79.3 Giga pixel image and forming a pyramid representation for fast access and retrieval. Image pyramid is a standard representation of very large size images that allows fetching only image sub-areas of interest to an end user at the requested resolution [1]. The pyramid representation avoids downloading the entire image as it might not be necessary to transfer all data and/or the end user might not have enough memory and bandwidth to receive the data. The pyramid is created by cropping tiles out of the highest resolution image, down-sapling the original image and cropping tiles out of the down-sampled image. The down-sampling continues until the tip of the pyramid is reached of pixel size.

Imaging using digital cameras or scanners can be performed over spatial segments of a single object whose entire digital image at the high spatial resolution has to be eventually stitched together. The motivation of our work is to understand the challenges of stitching Giga and Tera Pixel images using parallel computing resources. The applications of this stitching problem can be found, for example, in microscopy imaging with motorized stages, airborne imaging in multiple flight paths, or scanning of large size historical maps presented in pieces or rolls of film negatives.

Our work is related to past efforts that process many snapshot images to a large size single image such Google Map, various Giga Pixel images generated by photographers and representations to provide fast access to Giga pixel images (see [1]). Our work is different from the past efforts by working with airborne imagery (10,158 images acquired over multiple flight paths of Costa Rica in 2005), by dealing with a film-based rather than direct digital image acquisition, and by having a highly uncertain georeferencing information available for stitching. The georeferencing information was obtained during film scanning by manually creating an entry into an excel spread sheet. Our approach is to utilize the coarse georeferencing information for initial image grouping followed by an intensity-based stitching of groups of images.

Given the spatial coverage and resolution of the input images, the final stitched image is 294,847 by 269,195 pixels (79.3 Giga Pixels). We opted to create an image pyramid directly as we are stitching the input images together to overcome size limitations. The resulting image pyramid consists of 19 levels ($\log_2(\max(width, height)) = 18.17$) with the bottom level being formed from 1,211,904 tiles out of the total 1,616,015 tiles of the full pyramid corresponding to 317.6 GB of total size. Every level of the pyramid above the bottom level is created by sub-sampling multiple tiles to a form new tile. Thus, we just have to build the bottom level tiles from a set of input images.

An image group is formed from all images by finding those images that overlap with a chosen pyramid tile based on the course georeferencing information. First, images in each image group are stitched together (referred to as intra-group stitching). Next, the stitched image is cropped to fit a pyramid tile with a small extending overlap. Last, the tiles are adjusted for tile-to-tile misalignments (referred to as inter-tile stitching) using an image matching method, and the final tiles are cropped out.

We benchmarked a preliminary result of a quarter size image, resulting in a 4 Giga Pixel image (73,712 x 67,298 pixels). This process took 63 hours on a single processor. A full resolution image would take an estimated time of 42 days on the same hardware. If we want to move to larger countries or larger states (Texas for example will result in a Tera Pixel image), we have to use parallelization. Fortunately, the methodology described is trivial to parallelize since the computation for each tile is independent of all the other tiles. We have explored creation of the full size Costa Rica image using a 120 core computer cluster and report our preliminary results.

2. INPUT DATA

In 2005 the Costa Rica government hired NASA to capture aerial data of Costa Rica. The data was collected between March 2005 and May 2005. Figure 1 shows all the flight paths taken during the acquisition. The plane was equipped with a hyperspectral camera as well as an infrared camera. The images we used for the stitching of the final image are from the infrared camera. The camera inside the airplane recorded the images to a filmstrip, at the same time the airplane recorded the heading of the aircraft as well as altitude. The filmstrips where then developed and scanned in Costa Rica. For each image the operator manually entered, a sequence number, filename, central latitude and longitude, latitude and longitude for each of the four corners as well as the date of the image was recorded in an excel spreadsheet as each of the images was scanned. We received this excel spreadsheet together with 10,158 images as our input data. We downloaded the flight plans from the NASA aircraft as an additional input for our algorithm.

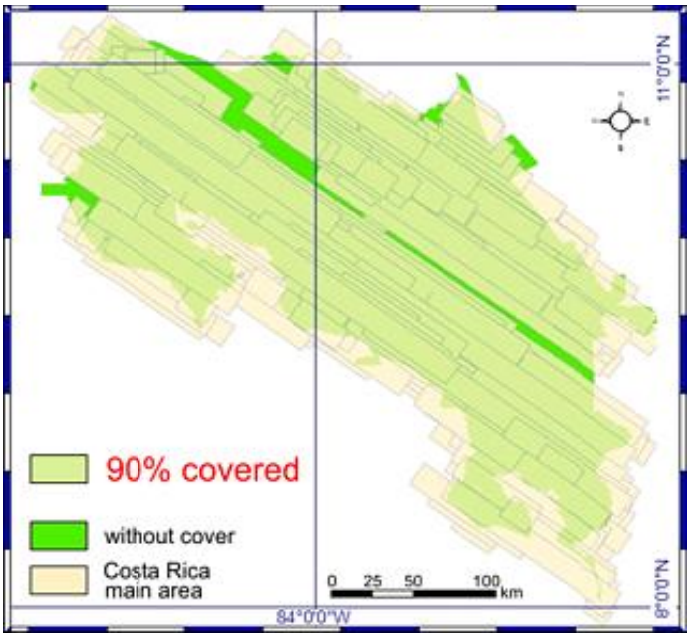

Figure 1 area covered of Costa Rica during the data acquisition in 2005.

We were given the task to create a single image of all this data, and come up with an easy way to disseminate this image. Using the information of the spreadsheet we estimated the final image to be around 300,000 x 300,000 pixels. We were not aware of any image formats that could handle an image of this size. Instead of creating a single image we started to look at different methods to disseminate the image. Our first approach was to use Google Maps and/or Google Earth to load the data and display the tiles as overlays on the map. We created a subsampled and correctly rotated image of each of the images we were provided as well as a single KML file with all of the image locations. Neither Google Maps or Google Earth was capable of handling the large number of images that we tried to overlay (even at 10% of the original size).

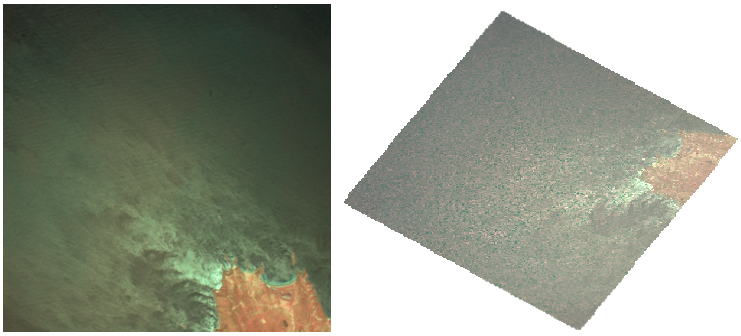

Figure 2: On the left is an example of an image tile for the region close to the coast of Costa Rica. On the right is the same tile but rotated based on the flight path.

Our next approach, and the one we will discuss in more detail in this paper, is to leverage of image pyramids [1]. Instead of creating a single image this approach will split the image in many smaller images (usually 256 x 256 pixels). Separate software will load each of these tiles and display such that to the user it looks like there is only a single image. Since the final image is significantly larger than the screen these image pyramids often create subsampled versions of the image until the smallest size image that is only 1x1 pixels. We opted to use Microsoft Seadragon since it does not require the user to install any software and can be viewed from inside the web browser as well as being cross platform (thanks to the use of JavaScript).

3. STITCHING FINAL IMAGE

To obtain a better understanding of the scale of the image stitching problem we worked with a subset of the input images. We selected 23 tiles from the complete dataset. The image content is a river and will help us with determining the accuracy of stitching. Using the georeferencing information provided to us we stitched all 23 tiles together and created the image in Figure 3. The image is 21,466 x 12,585 pixels and covers the area (in degrees, lat/lon) from (10.7149, -84.2812) to (10.8726, -84.0200). The image pyramid that was created is 15 levels deep and contains a total of 5641 tiles. In this image we can see the inaccuracy of the georeferencing information, as well as the potential pincushion effect from the camera as well as the scanner. Before stitching the final image we looked at improving the accuracy of the georeferencing information by applying a feature matching algorithm to the images.

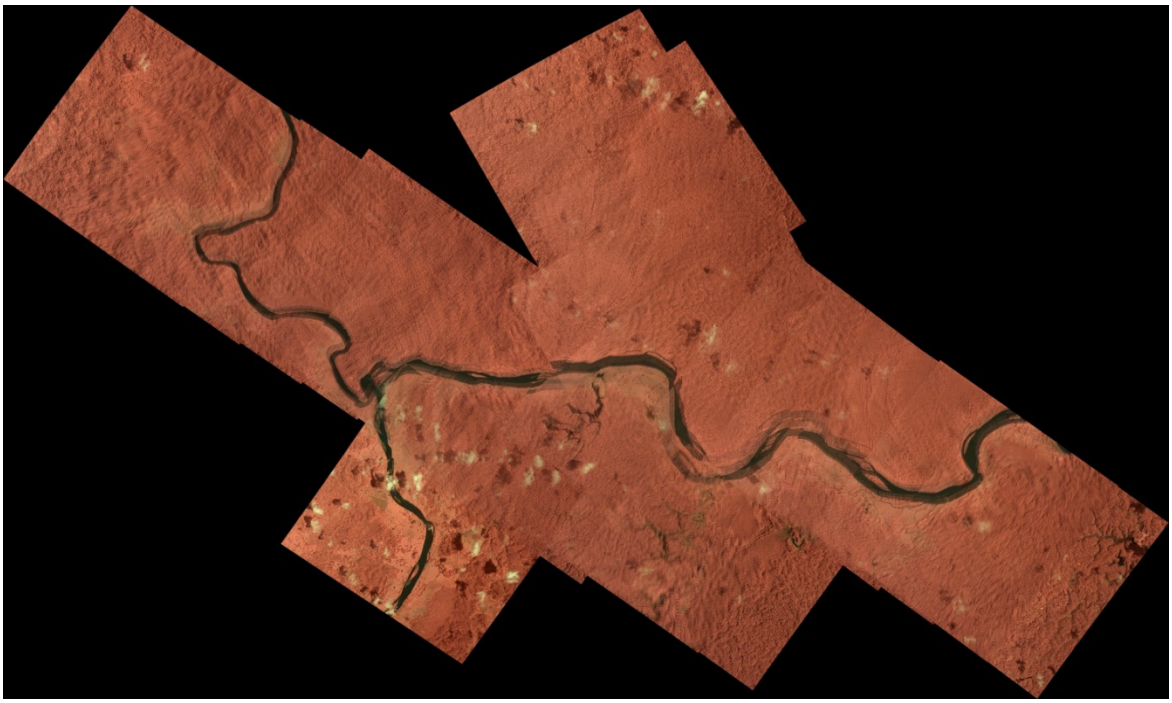

Figure 3 Initial stitching of subset of images.

The problem of alignment and stitching of images is a problem for which a lot of literature has been written. The algorithms for this task are among the oldest and more widely used in computer vision. These algorithms fall inside the image registration problems, which try to align some images by overlaying them, in the way that the pixels that represent the same parts of the scene are in the same position. In the case of remotely sensed images, there a lot of implemented and tested techniques, as documented in [2]. All of them can be used although the results would depend on the quality of input images (warping, accuracy of georeferencing information, presence of reliable features, intensity variation over the same area acquired at different times, etc.). As noted in [3], there is not a method that will work for absolutely all the images. There are four steps in the process of registration: feature detection, feature matching, transform model estimation and image resampling and transformation [4].

First, contrast salient and temporally invariant features are extracted from all images as the reference points. These features should be invariant across all images as they will be matched in later steps and used to calculate a spatial transformation. Good salient features can be regions, edges or points. In general, all types of features and all methods found in the literature are good candidates for stitching remotely sensed imagery. In our initial analysis, we selected regions as features and extracted them using the SURF feature detector [5].

Second, to match region features, we used the Euclidean distance metric applied to the descriptor vectors of each pair of SURF features. As in the matching method presented in the work reported by Herbert Bay et al.[5], a match is found if its distance is closer than 0.5 times the distance of the second nearest neighbor [6] [7] [8] (experimentally it gave better

results than the 0.7 value in the original implementation). Additional geometric restrictions reduce the possibility of a bad match, although bad matches cannot be completely avoided due to temporally dynamic objects, such as clouds or ocean waves along the coast. To minimize bad matches, a double relaxation process is implemented. The first step of the relaxation process is based on the Delaunay triangulation algorithm [9]. It is based in the fact that most points are well matched and hence the triangulation in both images should be very similar. The second step of the relaxation process is based on the outlier detection, assuming that the approximate geographical distance is the same between points in both images. Thus, corresponding pairs of points with a different geographical distance could be detected by comparing the distance to the mean distance and then declaring them as fake matches.

Third, transformation model parameters are estimated to align one image with another. In our implementation, we have approximated parameters of an affine transformation based on the control points determined in the previous step. The estimation is completed using a least-square fit method to find out the affine transformation model parameters.

Finally, the image is transformed and resampled using the affine transform computed in the previous step and the actual intensity values at each pixel location are calculated using a bi-cubic interpolation.

We have used the above method to warp 23 images from our subset and stitch them together. Figure 4 shows the resulting image after warping the tiles. This image uses the center image as the image to which all other images are warped (i.e. the center image is not warped but other images are). Figure 5 shows the advantage of warping the images in the resulting alignment quality. However, we can also see in Figure 4 that the images that are farthest away from the original image are extremely warped. The 23 tiles used are only a small subset of the final image. The larger the image we create and the farther away from the center image we go, the worse the warping and errors become. The best results would be obtained if the center image is also warped based on the surrounding image using a feedback loop. Nevertheless, we would still have to deal with the error propagation the farther away we go from the central image.

Based on the aforementioned considerations about (a) additional computation time needed to perform the warping of the images, and (b) the error propagation for image tiles far from the center tile, we have decided to pursue the non-perfect stitching approach.

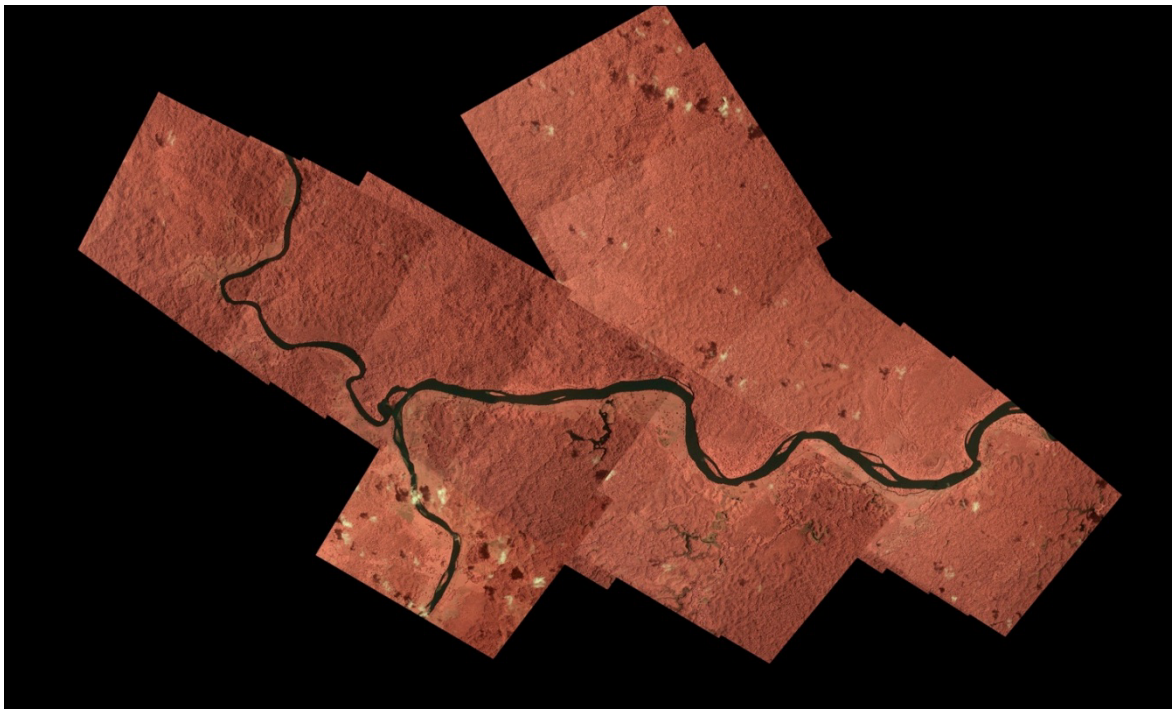

Figure 4 Stitching of subset images after applying corrections.

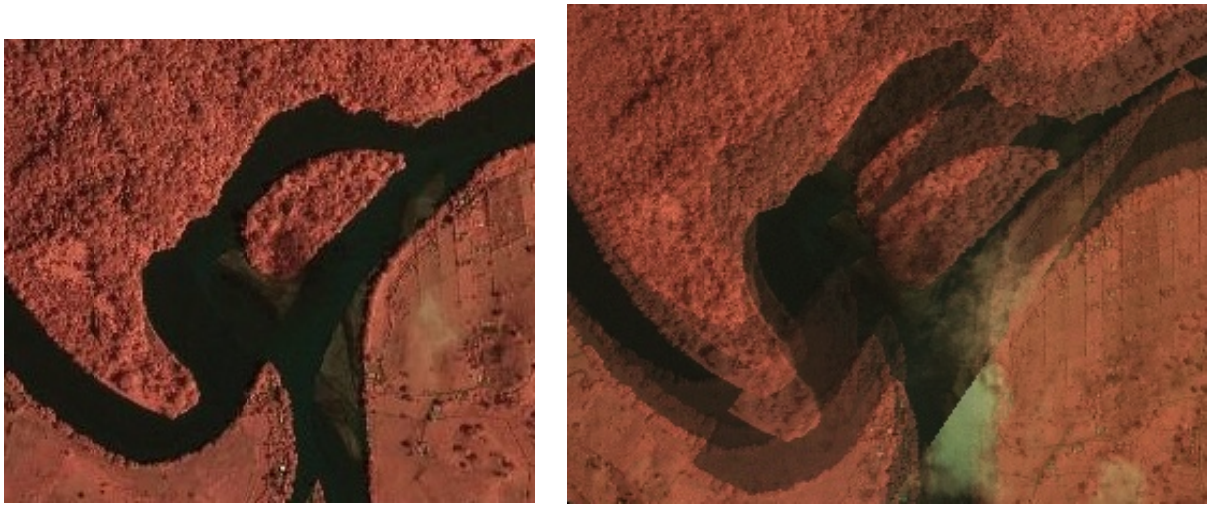

Figure 5 Close-up of warped image (left) and non-warped image (right).

4. PARALLEL ALGORITHM

The final image that we need to create is 19 levels deep and will have 1,616,015 tiles. Each of these tiles, however, can be created independently of any of the other tiles, resulting in a system that is easy to parallelize. We created a client/server model to compute each of the pyramid tiles. The server is responsible for handing out to each of the clients which pyramid tile needs to be computed next. Each of the clients will connect to the server and ask for pyramid tiles to be computed. The client will then convert the image location to a geographical location and find all images that are inside the bounding box of the pyramid tile. All images will be retrieved from disk, rotated and scaled to match the flight path as well as the pyramid tile and finally all tiles are put together to create the pyramid tile. The tile is written to disk at which point the client will notify the server of completion and ask for the next tile. One major advantage of this setup is that we can restart the system and continue computation in case the server crashes or any of the client crashes. If a client crashed then the server would simply reissue the tile that was lost. In addition, if the server crashed then it could start to issue all tiles again and perform a simple check of a client whether the tile already exists or not. If the tile already exists then it can be retrieved and does not have to be re-computed. This client/server design is important for any hardware with high failure mode over a long period of time (i.e., almost every hardware).

To test our hypothesis about computational speed up using the above parallel computation design, we benchmarked the image stitching and pyramid creation from the subset of images using a different number of parallel worker threads. For this purpose, we leveraged a dual quad core machine with 16GB of memory and 8TB of disk storage in a RAID-5 configuration. Each of the threads would contact a central server to fetch information about the next tile that needs to be created. Each thread will then load all original images needed to compute the pyramid tile, stitch the images together and write the final pyramid tile to disk. To optimize the system we first load all 23 images into memory before any tiles are computed (this prevents two threads from trying to load the same image at the same time). In Figure 6 we show the improvement of using many threads to do the actual computations. We are using an 8 core machine and thus we expect the graph to have a dip around 10 parallel computations since at that point the different threads start to compete for resources.

On the same dual quad core machine we benchmarked building an image pyramid of the full set of image tiles (10,158 images) but sub-sampled first to one quarter of the original resolution. It took 13 hours to create the image pyramid using 20 threads. During the image pyramid creation, the system was almost always 100% busy waiting for I/O, leading us to believe that adding more CPU resources would not improve the computational speed in our I/O limited hardware.

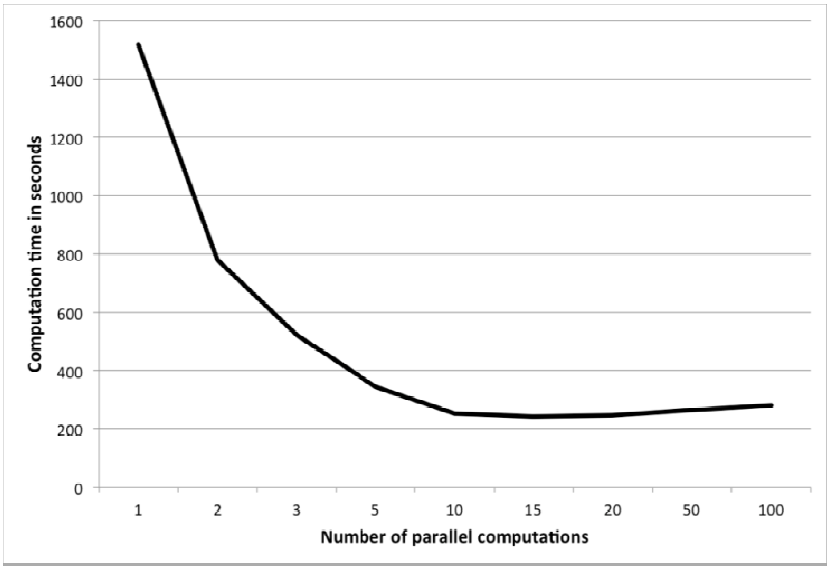

Figure 6: Graph showing time versus number of threads to create image pyramid.

5. FINAL STITCHED IMAGE

The final stitched image was created on one of the NCSA high performance clusters. The NCSA cluster called MS ABE is a subset of nodes of the ABE cluster running Windows operating system. The MS ABE cluster consists of 16 compute nodes and a single head node. Each of these nodes has 8 cores and 16GB of memory. The file system is provided as a Server Message Block (SMB) file system that is mounted on each of the compute nodes. For the creation of the final stitched image we were given full access to all 16 compute nodes. Our initial implementation had been running for around 30 days and had completed the computation of approximately 1,000,000 of the pyramid tiles. At this point we discovered the option of optimizing the code by loading the images from disk using our own file loader. Our file loader loads the images as RGB images, which resulted in a faster combination of images into each final image pyramid tile. This optimization led to a 20 fold speed-up in computation. Leveraging the client/server design of parallel algorithm (i.e., we could start and stop our computations without losing any previous results), we used the improved code for computing the final 600,000 tiles in 3 days.

Screenshots of the final stitched image and its pyramid representation are shown in Figure 7. Figure 7 (left) illustrates the result of stitching all 10,158 images together and viewing the image zoomed out far enough to show the outline of Costa Rica. As we zoom in on the bay of Costa Rica coastline in Figure 7 (right), we can start to distinguish the individual images that are stitched together to make the final image.

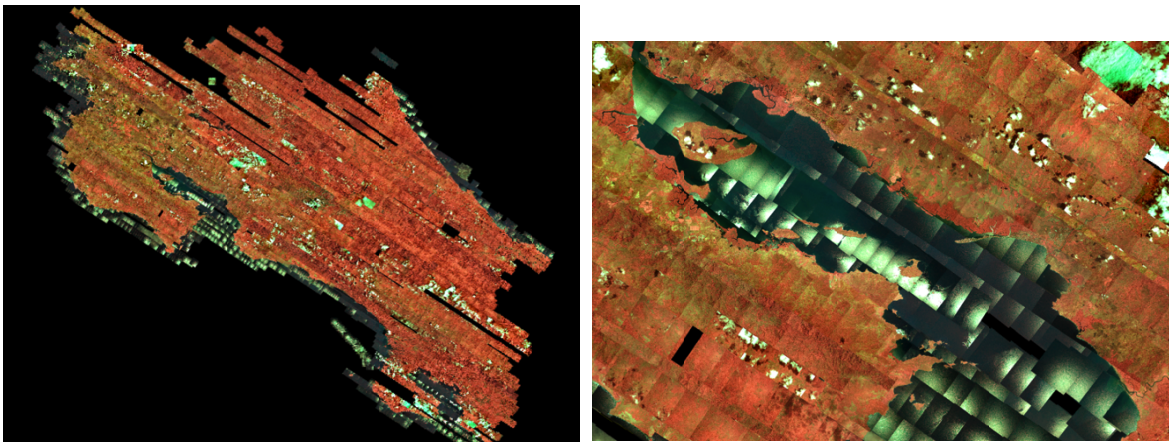

Figure 7: Left - the fully stitched image. Right - a zoomed viewed of the fully stitched image.

6. CONCLUSIONS

Using image pyramids we were capable of disseminating a 79 Giga Pixel image which would not be otherwise possible because the final image would be too large for any file format, or because the browsers or other applications cannot handle the large number of images or the full size of the stitched image.

Stitching and creation of these image pyramids is trivial to parallelize since each of the pyramid tiles can be created independently of each other. Having the ability restart the process is important especially considering the fact that the system initially was running for 30 days and survived two reboots (scheduled every first Tuesday of the month).

Leveraging of the massive parallelization we expect to be able to repeat this process to create the final stitched image in under a week of computation time.

ACKNOWLEDGEMENTS

This research has been funded through the National Science Foundation Cooperative Agreement NSF OCI 05-25308 and Cooperative Support Agreement NSF OCI 05-04064 by the National Archives and Records Administration (NARA). Additional funding was provided as part of the Google's Summer of Code 2009.

REFERENCES

[1] J. Kopf, M. Uyttendaele, O. Deussen *et al.*, “Capturing and viewing gigapixel images,” ACM Transactions on Graphics, 26(3), 93 (2007).

[2] L. M. G. Fonseca, and B. S. Manjunath, “Registration techniques for multisensor remotely sensed imagery,” Photogrammetric Engineering and Remote Sensing, 62(9), 1049-1056 (1996).

[3] D. Fedorov, L. M. G. Fonseca, C. Kenney *et al.*, “Automatic registration and mosaicking system for remotely sensed imagery,” Anais do XI Simpósio Brasileiro de Sensoriamento Remoto. Belo Horizonte: Instituto Nacional de Pesquisas Espaciais, 317-24 (2003).

[4] B. Zitova, and J. Flusser, “Image registration methods: a survey,” Image and vision computing, 21(11), 977-1000 (2003).

[5] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” Computer Vision–ECCV 2006, 404-417 (2006).

[6] A. Baumberg, "Reliable feature matching across widely separated views." 1774.

[7] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” International journal of computer vision, 60(2), 91-110 (2004).

[8] K. Mikolajczyk, and C. Schmid, “A performance evaluation of local descriptors,” IEEE transactions on pattern analysis and machine intelligence, 1615-1630 (2005).

[9] B. Delaunay, “Sur la sphere vide,” Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk, 7, 793-800 (1934).