SCALABLE ALGORITHM AND WORKLOAD EXECUTION FOR GEOLOCATING SATELLITE IMAGERY

by

AYMEN AL-SAADI

A thesis submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Shantenu Jha and Matteo Turilli

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

January, 2020

ABSTRACT OF THE THESIS

Scalable algorithm and workload execution for geolocating satellite imagery

By Aymen Al-Saadi

Thesis Director:

Shantenu Jha and Matteo Turilli

Climate change is having an impact on the polar regions, causing the retreat of sea ice and mountain glaciers as well as mass loss from the Greenland and Antarctic ice sheets. A better and deeper understanding of these changes requires processing large volumes of imagery data. Efficient analysis of these increasingly large volumes of data requires scalable computing, advanced and efficiently implemented algorithms and petabytes of dedicated storage on high-performance computing (HPC) platforms.

Image geolocation is the process of estimating the geographic location of a single image using of previously geolocated aerial and satellite imagery. The Scale-invariant feature transform (SIFT) algorithm is used to geolocate images by finding the similarities with other images in the database. The SIFT algorithm is invariant to image resizing and rotation, and partially invariant to changes in brightness and camera viewpoint.

In Chapter 3 we evaluate the accuracy in terms of the number of matched key points of two SIFT implementations: (a) CPU-SIFT, and (b) CUDA-SIFT. CPU-SIFT is an implementation of SIFT algorithm on CPU; CUDA-SIFT is an implementation of SIFT algorithm on GPU using CUDA parallel framework.

A performance characterization showed a number of limitations for CUDA-SIFT,

such as low matching accuracy and run one CUDA kernels on a single GPU device while it is possible to run two. We addressed these limitations by increasing the size of image tiles that can be processed, the number of kernels of the CUDA implementation that can be concurrently run on the same GPU device, and improved the number and type of image features that can be concurrently evaluated.

We also evaluated the throughput of CPU-SIFT and CUDA-SIFT in terms of image size in Megabytes per second (MB/s) and their memory consumption. Chapter 3 showed the performance evaluation of both implementations in matching multispectral and satellite imagery. In our evaluation, geolocating two 1 GB satellite images with no geolocation information took 15 minutes using the CPU-SIFT implementation with an accuracy level of 89% and 4 GB of memory on one CPU (Intel Xeon E5-8860 v3). CUDA-SIFT matched the same images, taking 3.5 minutes on one GPU (NVIDIA Tesla P100), with an accuracy of 56.81%, and 7.1 GB of GPU memory on the XSEDE Bridges supercomputer.

An analysis of the current CUDA-SIFT showed that both accuracy and throughput could be improved. Motivated by possible enhancements, we implemented GPU-SIFT, which is an implementation of the SIFT algorithm, for GPUs implemented and based on the previous implementation of CUDA-SIFT. GPU-SIFT offers three major improvements over the previous implementation: (i) we implemented the MPS CUDA parallel framework to enable concurrent matching of multiple image tiles via multiple GPUs; (ii) we doubled the amount of dynamic memory that can be allocated per GPU from 2 to 4 GB. In this way, we increased the size of the images that can be tiled and processed from a maximum of 3000 x 3000 to 5000 x 5000 pixels; and (iii) we implemented an adaptive contrast enhancement, increasing the number of extracted and matched keypoints for each image. GPU-SIFT offers a significant increase in the number of extracted features with an accuracy of 74.18% by applying histogram contrast enhancement.

Finally, we compared the tradeoffs between the accuracy of CPU-SIFT, CUDA-SIFT and GPU-SIFT in terms of the number of matches between a pair of images and the execution time for the entire matching process. The tradeoffs between SIFT matching accuracy and SIFT execution time can create performance variance and can lead to

different resource requirements. Characterizing the performance and understanding the performance tradeoffs between GPU-SIFT, CUDA-SIFT and CPU-SIFT can help to predict the performance of these algorithms and their applications with large-scale workloads.

Chapter 4 presents a workflow to run hundreds of image geolocating workflows concurrently on HPC using algorithms, and software tools that support the scalable and automated computation of image geolocation. We present a scalable image geolocating workflow with two different kernels (CPU or GPU) and evaluates the requirement of large-scale ensemble-based technique to run image geolocating pipelines.

The contributions of this thesis are: (i) Compare and characterize the performance between CPU-SIFT, CUDA-SIFT and implement GPU-SIFT as an improvement to serve the image geolocating use case; (ii) The design, implementation and concurrent execution of scalable image geolocating workflows as a set of pipelines to enable concurrent execution for 200 images with near linear scalability on 224/8 CPUs/GPUs.

This thesis is a part of the Imagery Cyberinfrastructure and Extensible Building-Blocks to Enhance Research in the Geo sciences (ICEBERG) project. Our work has an impact on the domain scientist that will help move geoscience studies of image analysis to a new infrastructure linking scientists, satellite imagery, and high performance computers. This new imagery-computing superhighway will make it easier for scientists to study processes at much larger spatial scales than has been previously possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we show the motivation and the objectives for image geolocating in specific and the thesis in general. We discuss briefly the existed computational challenges and the critical need for HPC in image processing and analysis development. We include a background section with a discussion of the building block approach in workflow systems, focusing on RADICAL-Cybertools (RCT) that provide a solution to the identified challenges. In addition, we discuss the computational challenges in image processing generally and image geolocating more specifically. Lastly, we discuss the related work of the image geolocating use case and why our work can serve the geoscience field in a better way.

1.1 Motivation

Global warming and dramatic climate change are having a massive impact on the polar regions, causing the retreat of sea ice and mountain glaciers, mass loss from the Greenland and Antarctic ice sheets and a generalized disruption of many related environmental and ecological processes (see Fig 1.1 and Fig 1.2).

A better and deeper understanding of these changes through time requires enormous quantities of diverse observational data from satellite and airborne imagery to automated field observations. These streams of data need integration into high-resolution computer models and sophisticated means of analyzing and visualizing emergent patterns. Efficient analysis of the increasingly large volumes of data requires petabytes of dedicated storage and millions of core hours on high-performance computing (HPC) infrastructures.

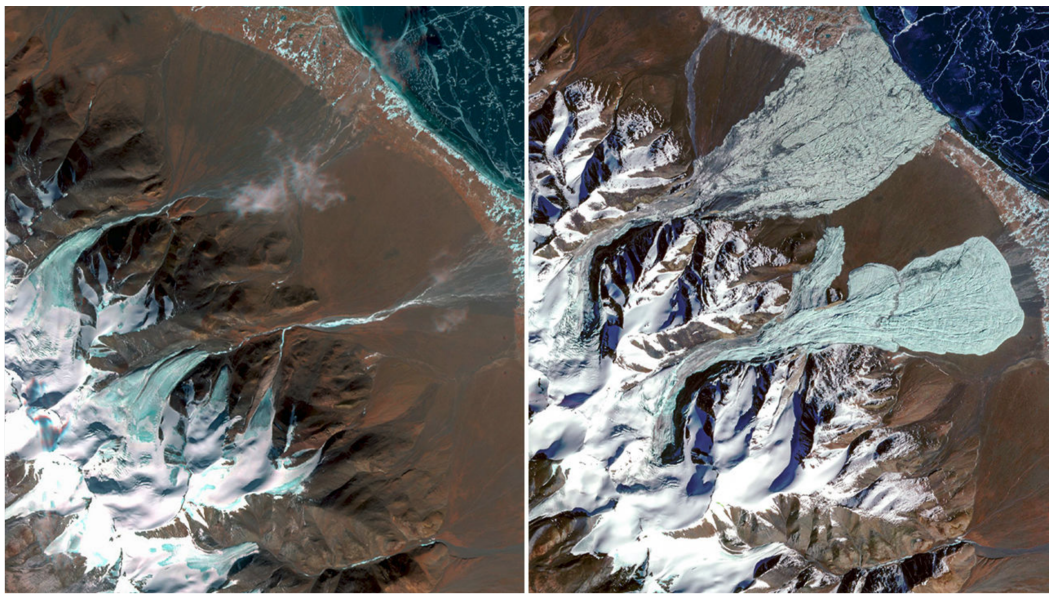


Figure 1.1: Two satellite images from March 2017(right) and September 2017(left) showing the fast and massive ice retreat of the ice mass in the Tibet illustrating the respective winter maximum and summer minimum extents

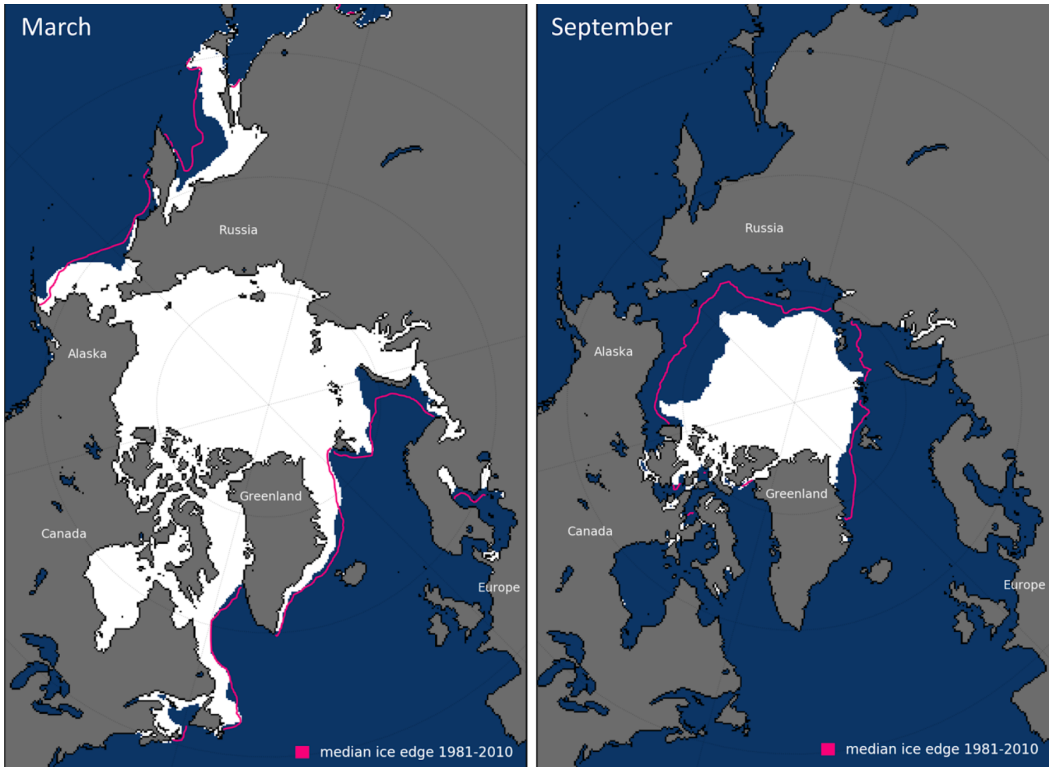


Figure 1.2: Average monthly sea ice extent illustrates the respective winter maximum and summer minimum extents. The magenta line indicates the median ice extent in March and September, respectively, during the period 1981-2010.

Satellite imagery geo-locating, the procedure of determining the geographical information such as longitude and latitude of an image using just the visual information from satellite imagery, is remarkably difficult. However, images often contain useful visual and important informative cues which help to predict an estimated location of an image with variable confidence. Yet, estimating the location of the satellite imagery in Antarctica and north pole in natural environments is substantially more challenging.

However, large satellite images and the lack of using GPU technologies in image geolocation techniques results in in excessive Computations and performance bottleneck. Thus, running image matching algorithm such as GPU-SIFT on GPU can solve the bottleneck of running large volumes pixel chunks at once. GPU-SIFT take advantage of the GPU high computational power by offering the ability of tiling and processing large image sizes (5000 x 5000 pixels) and matching them in parallel using CUDA parallel framework.

The GPU-SIFT described in Chapter 3 increases the number of extracted and matched features by enhancing the level of contrast using adaptive contrast enhancement techniques. Finally, overcoming the computational challenge of image geolocating demands rapid integration of high-performance and distributed computing (HPDC) infrastructure for the following:

- **Fast retrieval data storage:** Image geolocating demands a large space of fast retrieval storage type to store and retrieve the high resolution airborne and satellite imagery from the database. This requires about 1TB - 2 TB of storage space to save the satellite images in an organized databases.

- **Fast image processing:** The increased rate of image processing development and matching techniques allow for more data acquisition than ever before. Since satellite imagery has a higher resolution and the number of images that these satellite capture is exponentially growing, it requires to have a compute intensive platform such as HPCs that enables these requirements.

- **Concurrent task execution :** Using software like RADICAL-Pilot [**?**] and Ensemble Toolkit (EnTK) [**?**], it becomes easier to run large numbers of computational tasks concurrently on a remote HPC resources. Thus, running tasks concurrently can decrease the execution time of the entire task dramatically and reduce usage of compute resources.

1.2 Overview

Chapter 1 provides the motivation and the objectives of this thesis. We discuss briefly the current challenges and the critical need for HPC in imagery processing and analysis development.

We include a background section with a discussion of the building block approach to workflow systems, focusing on RADICAL-Cybertools (RCT) that provide a solution to run large scientific workflows concurrently on HPC infrastructures. In addition, we discuss the computational challenges in image processing generally and image geolocating more specifically.

Chapter 3 will present existing approaches, solutions and the algorithmic details of their geolocation workflow. This will include an overview of the current matching techniques, a study of each component, the effects of their possible interactions on the level of performance, and ways to improve their current implementation.

In Chapter 3, we discuss the first contribution of this thesis in implementing GPU-SIFT as an improvement on CUDA-SIFT for the image geolocating use case. We discuss the GPU-SIFT workflow and evaluates it performance, in addition, we include a performance evaluation of both CPU-SIFT and GPU-SIFT based on throughput, memory consumption, and number of features extracted and matched.

Chapter 4 describes and discusses the design of the image geolocating workflow. We discuss the implementation of the image geolocating pipeline using EnTK, in addition we includes a discussion of the computational challenges of scalability. We conclude the chapter with a set of experiments for the image geolocating pipeline to evaluate both strong and weak scaling for 200 pairs of high resolution airborne and satellite imagery.

In Chapter 5, we outline the key conclusion of this thesis with and the impact of deploying GPUs in image geolocating workflows. We discuss and present the importance of high performance computing in accelerating the image geolocating process.

In summary, we outlined both scientific and computational challenges to solve key problems in the imagery sciences, for which the efficient utilization of high-performance and distributed computing has become increasingly critical to continued scientific advancement.

It's worth mentioning that the design of the image geolocating workflow is mainly based on a set of requirements needed by the image geolocation use case to execute multiple sets of tasks in the geolocating workflow for every image in the desired dataset, extract and match set of features from every pair of images and extract geographical information. Designing and implementing the execution pipeline is done using a python library that supports the developing and executing large-scale ensemble-based workflows simply called "Ensemble Tool Kit" (EnTK).

Chapter 2

Background and Related Work

In this chapter we discuss the existing software solutions that enable execution of ensemble-based workflows on HPC infrastructures and their limitations, we include a brief description of the building blocks approach to workflow tools, focusing on RADICAL-CyberTools (RCT). Next, we provide a brief description of Imagery Cyber-infrastructure and Extensible Building-Blocks to Enhance Research in the Geosciences (ICEBERG) project and middleware. Next, we discuss the image geolocating process and we focus on our use case to geolocate set of historic aerial and satellite imagery. Next, we discuss the computational challenges associated with processing large size satellite images in the context of a image geolocating workflows. In addition, we include a brief overview describing the required memory to process large satellite imagery and the implications behind it. Finally we discuss the related work of the image geolocating and highlights the importance of our image geolocating approach.

2.1 RADICAL-CyberTools

RADICAL-Cybertools (RCT) is a set of python libraries that supports the development and execution of scalable workflows of different scientific applications on a range of high-performance and distributed computing (HPDC) infrastructures. RCT components are: (i) RADICAL-SAGA (RS); (ii) RADICAL-Pilot (RP) and (iii) Ensemble Toolkit (EnTK) see Figure 2.1.

The image geolocating workflow uses two RCT components, EnTK and RP. EnTK creates and executes ensemble-based workflows with different stages and tasks, and facilities complex communications between every stage without the need for explicit resource management. Radical-Pilot works as a run-time engine for EnTK that acquires

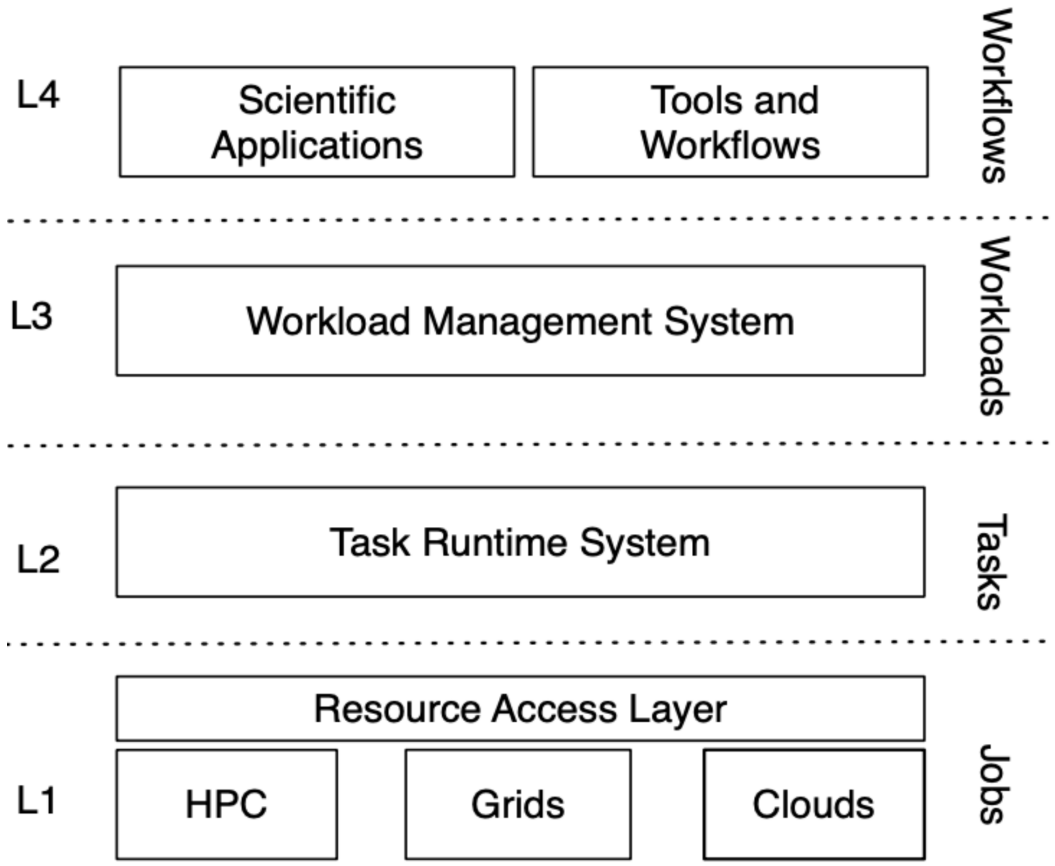


Figure 2.1: Functional level representation of RADICAL-Cybertools (RCT) showing the functional requirements for workflow systems, and the primary entities at each level.

and manage resources with task execution capabilities.

RCT [**?**] supports the concept of building blocks for scientific workflow systems. RCT provides scalable implementation of building blocks using Python that and allows different scientific applications to run on HPC and distributed systems. In this thesis, we present an understanding of how RCT has been deployed to support the concurrent and scalable execution of multiple image geolocating workflow pipelines to match a large database of airborne and satellite imagery.

2.2 Imagery Cyberinfrastructure and Extensible Building-Blocks to Enhance Research in the Geosciences (ICEBERG)

ICEBERG is an NSF EarthCube collaborative project [?] between Rutgers University, Stony Brook University , and three other Universities, focused on developing the cyberinfrastructure required to facilitate pan-Arctic and pan-Antarctic science projects requiring manipulation and interpretation of high-resolution satellite imagery.

The ICEBERG science drivers span the biological and geological sciences and will radically increase the spatial scale at which satellite imagery can be used for understanding the biological, hydrological, and geological functioning of the polar regions. Over the last decade, algorithms for using high resolution satellite imagery have been developed; however, the computational requirements needed to scale these pilot projects has become a major limitation. At this point, a transition to HPDC-based workflows is required, along with the development of tools that make such large-scale computation accessible for the larger polar community.

ICEBERG aims to provide a suite of diverse use cases that can demonstrate the use of imagery for understanding the biological and geosciences problems at very large scale. These use cases will work as a set of independent Python packages that can be installed under a global package named ICEBERG middleware. ICEBERG middleware [?] is a software that can be installed on Linux systems and allows users to execute these use cases on high performance computers out of the box. ICEBERG middleware will help the domain scientist to test, run, and analyze their use cases at a larger scale than previously possible and will deliver deeper faster and better understanding for most of the geological and geoscience problems.

2.3 Image geolocation

Image geolocating or geotagging is the process of appending geographical identification metadata to various aerial and satellite imagery. This process is mainly based on a image matching technique to extract the similarity level between two images and estimate the approximate location as a values of longitude a latitude.

In our image geolocating use case, we focus on geolocating and geo-rectifying set of historic aerial photo imagery using Worldview imagery as a basemap. Geo-rectified time series will allow for an analysis of major changes over the last century, specifically over the period prior to satellite imagery. Chapter 3 describes the image geolocation use case in detail and describes every phase in the image geolocating workflow in chapter 4 Figure 4.2.

2.4 Computational Challenges

HPC platforms were built to enable fast, scalable and concurrent execution of a large number of tasks. Image processing techniques contains several processing stages that can run independently or dependently from other stages [**?**].

In satellite imagery such as Worldview-1 [**?**], Worldview-2 [**?**] and SAR [**?**], data is received and stored in a raw state. Raw satellite imagery can not be interpreted directly as it requires several preprocessing tasks to extract a set of specific information from every image. The continuous and fast increase in the resolution of satellite images, scaling up current image processing techniques, such as image matching, is a current barrier.

In order to decode a single satellite image to prepare it for tiling, the entire image has to be loaded into memory. For any image, we can estimate the required memory to load the image based on the type, width and height of the image. For example, to load an RGBA image of size 60,000 x 60,000 pixels, we can use the following formula to get the actual memory requirement :

$$RequiredMemory_{RGBA} = Image_{width} \times Image_{height} \times 4$$

Where:

- $ImageWidth$: The number of pixels in every row in the image.
- $ImageHeight$: The number of pixels in every column in the image.

- $RGBA = 4$: Red Green Blue Alpha. Three-channel RGB color model supplemented with a 4th alpha channel.

The number of features extracted from any type of image depends on the width and height of the image and the descriptor size used by the algorithm. The SIFT descriptor is a 3-D spatial histogram of the image gradients in characterizing the appearance of a keypoint. SIFT keypoint (feature) is a circular image region with an orientation. SIFT key points can be represented using 4 parameters: the keypoint center coordinates x and y, the keypoint scale (the radius of the region), and the keypoint orientation (an angle expressed in radians)

For the SIFT descriptor of size 128D (128 dimensional feature vector) and image size of 640 x 640 pixels, the keypoints found are absolutely not the same as compared to those identified on an image of size 1024 x 1024 pixels. Based on our experiments in chapter 3, the number of extracted features from images of size 60,000 x 60,000 pixels using the current feature extraction techniques is about 100,000 (including correct and incorrect matches [false positive and false negative] from the entire image).

SIFT requires at least 2.5 GB of memory to perform the extraction process and about 5 GB to perform the matching process. In our use case, it is critical to use HPC due to the increased processing time and the required resources to match a dataset of 200 pairs of airborne and satellite images in the future.

2.5 Related Work

Image geolocation is an existing problem of interest for computer vision scientists. The most common approaches are using scene and object recognition technology to identify a set of images and classify them into categories based on the scene of interest such as “building,” “mountain,” or “lakes”. These approaches mainly do not estimate the approximate location of where these images were taken. PlanNet [**?**] is a glocalization system approach that can geolocate a set of images using a trained convolutional neural network. The system takes a single image as an input and generates a probability for around 26,000 cells in a grid covering the possible area of where this image was taken.

PlanNet takes several days to train their convolutional network and was not able to localize beyond the city or street level with an accuracy of 3.6%.

Another approach in computer vision is geolocating satellite and ground-level imagery. Ghouaiel and Lefèvre [?] proposed an automatic translate for ground photos into aerial viewpoint, the technique specifically supports only wide panoramic photos with an accuracy of 54%. In the large scale image geolocalization[cite], the approach is based on using the "IM2GPS" algorithm [?]. IM2GPS uses the convolutional neural network (CNN) to geolocalize images against a database of geotagged Internet photographs as training data. The approach can produce probable regions of the earth with a localization level of 25% of the 237 photos in their dataset.

By combining the previous approaches from above, we can conclude that these approaches can not serve the aerial and satellite image geolocating use case. It is worth to mention that the presented approaches helped to increase the location estimation accuracy to a certain limit with limited image size. But these approaches can not scale up to 5000 x 5000 pixels of image size because they were not designed and implemented to run on that scale.

Our image geolocating workflow that we show in Figure 4.3 can geolocate a dataset of previously validated and geotagged of ~200 satellite imagery against aerial and airborne imagery in 20 minutes. The image geolocating workflow can geolocate the entire dataset on both CPU and GPU devices with a level of geolocating accuracy of 78% using 224 cores of CPUs and 4 GPUs running in parallel. The image geolocating workflow can geolocate most of Antarctica ~200,000 images in ~5 hours.

Chapter 3

Scalable Implementations of SIFT Algorithm

HPC offers the potential of using multiple CPUs and Graphical Processing Units (GPUs) as well as parallel computing libraries such as the CUDA framework support for GPUs and the MPI framework for CPUs [**?**].

However, scalable algorithms and their implementations are needed to utilize the thousands of cores available on today's supercomputers. Scalability is often measured as the amount of time taken to complete the execution of the workload and is a function of workload. Strong scaling of an algorithm can be measured by fixing the amount of work while the execution time decreases with increasing resources. Weak scaling is measured by fixing the ratio between the amount of workload and the amount of available resources.

The most common CPU-GPU parallel processing frameworks are CUDA parallel framework and Message Passing Interface (MPI)[10]. MPI and CUDA frameworks use load balancing techniques to improve the distribution of the processes or tasks among available resources to make full use of available processors and reduces the interprocessor communication overheads. This is important to avoid starvation of resources and excessive queuing of parts of the workload on a specific resource. For example, in computer vision, imbalanced distribution of vision tasks among the available resources can create unpredictable execution time and inconsistent usage of memory.

Alternatively, there are hardware techniques that support faster communications between the hardware adapters. Here, we briefly describe the Single Instruction Multiple Data (SIMD) technique. SIMD performs the same operation on multiple data points simultaneously; this is very useful to common tasks such as adjusting the contrast in a digital image or performing a mathematical operation repetitively. These hardware

techniques can be triggered using software-level techniques to create a synchronization between the software and hardware components to ensure enhanced and non blocking data flows.

The goal of scalable image processing algorithms is to enable parallel image processing techniques using a set of adequate parallel programming tools that exploit specific levels of complex architectures and in particular clusters of GPUs, i.e. a cluster in which each node is equipped with a GPU. In this chapter, we use two implementations and develop a third one of the Scale Invariant Fast Transformation algorithm (SIFT) algorithm:

- **CPU-SIFT:** a pre-existing sequential implementation of SIFT on CPU [**?**].
- **CUDA-SIFT:** a pre-existing implementation of the SIFT algorithm for GPUs implemented using the CUDA parallel computing platform [**?**].
- **GPU-SIFT**: an implementation of the SIFT algorithm, for GPUs implemented and based on the previous implementation of CUDA-SIFT. GPU-SIFT use the adaptive contrast enhancement technique to enhance the levels of contrast in source and target images. GPU-SIFT offers a set of capabilities that were not implemented in CUDA-SIFT, such as reading GEO tiff satellite imagery, running two CUDA applications on the same GPU, and utilizing the resource of GPU more efficiently, loading and processing larger tile size up to 5000 x 5000 pixels compared to CUDA-SIFT 3000 x 3000 pixels.

3.1 CPU and GPU Image processing

Serial image processing requires a long time to load, process and store the image, as the load, process and store steps are performed sequentially. To solve this problem, parallel computing techniques, especially multicore and multiprocessing technologies, should be used, to run image processing algorithms on larger and bigger number of images. Adapting parallel computing techniques in the image geolocating use case helps to cover the whole surface of the Antarctic and Greenland in five hours instead of two days.

A multicore CPU is a single computing component with two or more independent actual central processing units called "cores". Pthreads, OpenMP (Open Multi-Processing), TBB (Threading Building Blocks), and Cilk are application programming interfaces (API) to efficiently use the capacity of a multicore CPU.

GPUs are highly parallel programmable microprocessors built to assist the development of image processing applications. GPU devices can run in combination with CPU devices to speed up the compute-intensive applications. Thus normally, the parts of the code that requires a lot of data parallelism are processed on GPUs (device), while at the same time, the CPU (host) can perform the less compute-intensive parts that require serial data processing.

Using “Compute Unified Device Architecture” (CUDA, Figure 3.1), it is possible to design, implement, and optimize kernels that can run on GPUs to accelerate the entire performance of non-CUDA application. CUDA kernels are executed as CUDA-threads combined into chunks that are called CUDA-blocks; each block can have its own local memory (shared) and synchronizes the accesses to that memory. The execution model of CUDA provides a high level of scalability by offering the technique of mapping the threads (as a block) to the streaming multiprocessors (SMs) and then map the total number of blocks to each streaming multiprocessor.

The number of CUDA blocks that can run in parallel on SM is defined by the number of CUDA warps supported by each SM and the number of warps that every SM handles. CUDA framework offers the combination of high-bandwidth for shared and private memories and the high level of hardware parallelism that can calculates the floating-point of arithmetic operations at a higher rates compared to the conventional CPUs [**?**].

3.1.1 CPU-SIFT

CPU-SIFT is based on distinctive image features extraction presented by David Lowe in 2004 [**?**]. The SIFT algorithm is widely used to perform reliable matching between objects or single tile of two images from a slanted view.

The SIFT descriptor that was explained in Section 2.4 is a 3D spatial histogram of

the image gradients in characterizing the appearance of a keypoint. It is important to understand that the size of the descriptor is a critical factor in the SIFT performance, as larger descriptors size can cover a wider range of pixels that requires more memory and processing time.

Algorithm 1 SIFT Algorithm Pseudo code for CPU

```
1: function FIND_MATCHING(R, A)                ▷ Where R is the Region and A = P_1...P_n
2:     Compute SIFT descriptors S(P_1).....S(P_n)   ▷ Where S(P_n) is the search space for region A
3:     Compute SIFT descriptors S(R)
4:     max_A = 100                             ▷ Where max_A is the maximum distance from that point
5:     min_A = 0                               ▷ Where max_A is the minimum distance from that point
6:     dist = 0
7:     for i = 1 to n do
8:         for ∀p ∈ S(R) do
9:             if Matching points exist then
10:                Append to FoundMatches
11:                Search space of region A
12:            end if
13:        end for
14:    end for
15:    for j = 1 to FoundMatches do
16:        if dist < min_A then
17:            min_A = dist
18:        end if
19:        if dist > min_A then
20:            max_A = dist
21:        end if
22:    end for
23: end function
```

Listing 3.1 SIFT Algorithm Pseudo code for CPU

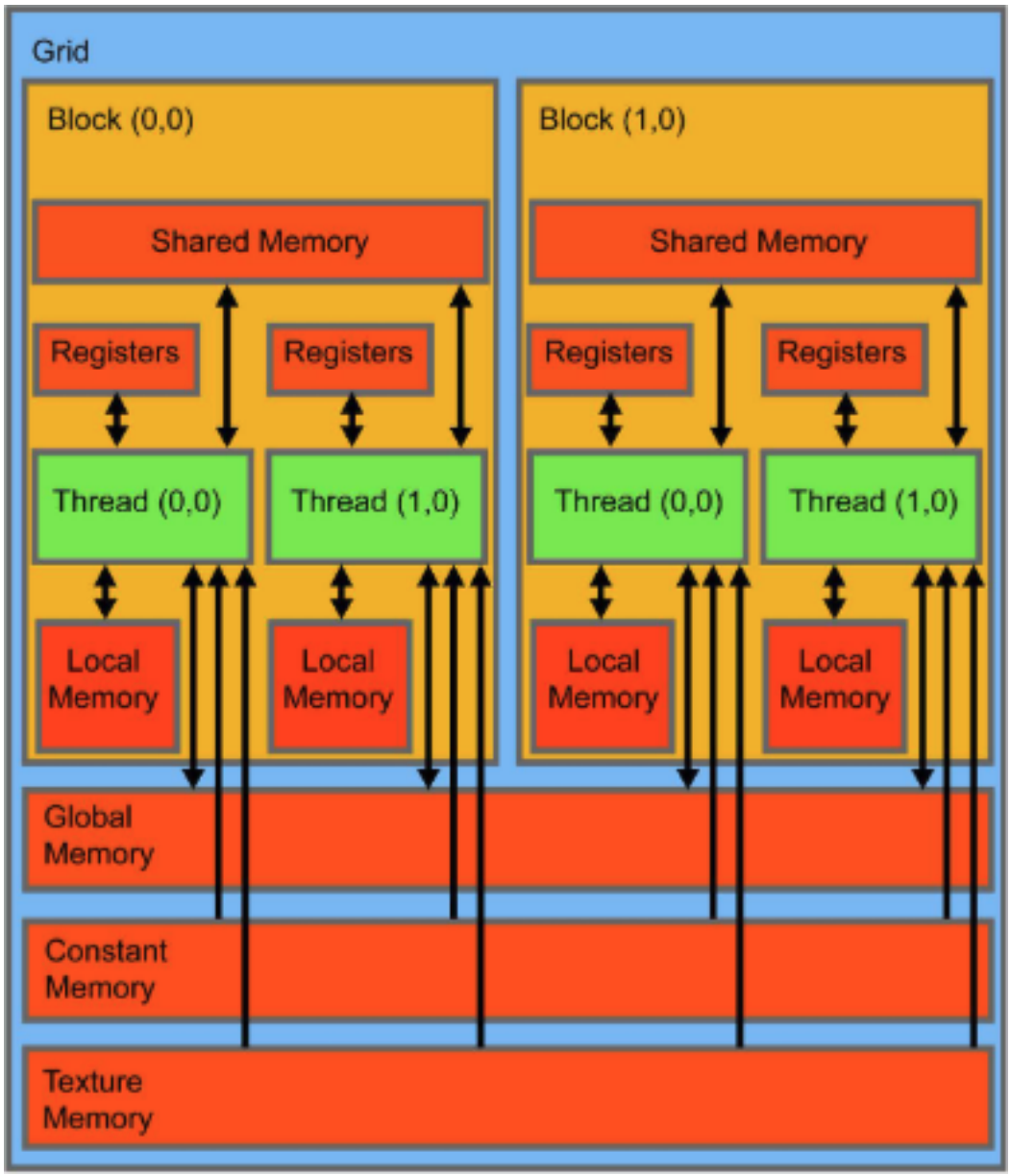


Figure 3.1: NVIDIA's representation of grids, blocks, and threads their memory accessibility.

Figure 3.1 shows the 4 stages of the SIFT algorithm. We highlight the primary steps of the SIFT algorithm workflow below:

- Detect and compute the Gaussian scale-space. Gaussian scale-space is a sequence of blurred 1D images, indexed by the amount of blur.
- Find candidate keypoints and filter them. A 3D difference of Gaussian (DoG) scale-space space is constructed and local extrema in this space are detected. DoG is obtained by subtracting one blurred version of an original image from another, less blurred version of the original.
- Assign a reference orientation to each keypoint.
- Calculate the keypoints descriptor.

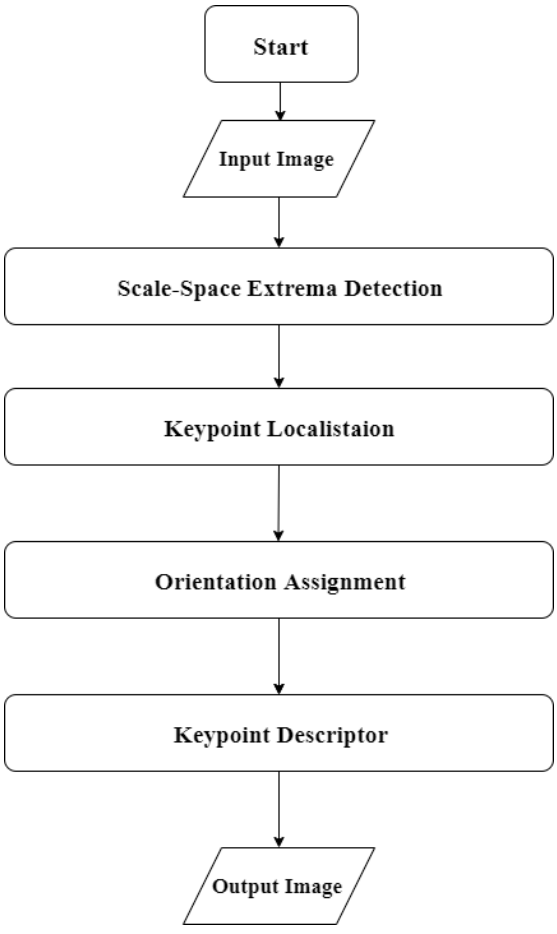


Figure 3.2: SIFT Algorithm Flowchart

3.1.2 CUDA-SIFT

CUDA-SIFT [?] is implemented using the CUDA parallel computing platform. CUDA-SIFT [4] uses techniques similar to CPU-SIFT by applying the same type of filters, number of octaves, contrast level, and features extraction threshold.

CUDA-SIFT is written as single kernel of thousands of CUDA threads, with multiple threads grouped into single or multiple blocks. Multiple CUDA-blocks can concurrently be assigned to the same Multiprocessor (MP). The current SIFT kernel is designed to run on a single GPU, so that it can use most of the GPU memory to achieve maximum performance.

Benchmarks of CUDA-SIFT in Table 1 shows that the kernel was tested on image of type ppm and JPEG of resolution 1920 x 1080 pixels and 1280 x 960 pixels.

GPU Architecture	1920 x 1080 pixel	1280 x 960 pixel
GTX 1080 Ti	0.8	0.6
GTX 1060	1.7	1.2
GTX 970	1.8	1.3
Tesla K40c	3.4	2.4

Table 3.1: CUDA-SIFT benchmark

3.1.3 GPU-SIFT

Running the SIFT algorithm on GPU can solve the bottleneck of running large volumes pixel chunks at once. Image geolocating use case that we mentioned in chapter 2 requires processing of image tiles up to 5000 x 5000 pixels. The GPU-SIFT was developed to handle PNG or JPEG images with no support for GeoTIFF imagery satellite imagery.

GPU-SIFT offers the following functionality: (i) reading large GeoTIFF satellite imagery to handle large size images (higher than 1920 x 1080); (ii) adding the capability of reading dual-band 8 and 16 bit GeoTIFF satellite imagery; (iii) increase the allocated CUDA memory per image; (iv) implementing CUDA Multi-Process Service (MPS) technology to run 2 CUDA kernels on single GPU device; (v) increase SIFT container (vector) to 100,000 to allow more keypoints for larger image size; and (vi) Implement adaptive contrast enhancer.

The proposed algorithmic improvement using adaptive contrast enhancement can be seen in Algorithm 2, and the implementation of the GPU-SIFT workflow can be seen in Figure 3.2. The implementation is composed of five steps. The first step reads the source and the target GeoTIFF images and converts them to grayscale mode. This temporarily saves the x and y coordinates for every source and the target image to be used in the next step.

The second step is called "simple region extraction" method. This step helps to check if the entire image matches or only a specific area, this can help to process smaller patches of the image to speed up the process of matching instead of processing the entire image.

The third step is converting both source and target images into CUDA images by allocating extra memory space for both images compared to the old implementation of CUDA-SIFT, this can help to process larger tile sizes in the GPU device.

Step four is the adaptive contrast enhancer, this step adjusts the contrast levels of both source and target image and can provide a higher level of accuracy. Finally, the source and target image are evaluated by the SIFT descriptors to extract the common features, match them and save them in CSV files.

Below we explain the implementation of GPU-SIFT.

- **Increase allocated CUDA dynamic memory:**

  The SIFT algorithm is considered to be memory intensive as it requires a certain number of multiplication operations to be done in the memory of the GPU device (memory accesses-read and write operations). The number of operations the SIFT algorithm performs is based on the number of octaves being generated by the algorithm as a larger number of octaves depends on the size of the original satellite image. An octave is the set of images generated by progressively blurring out the original image on different scales.

  The high image resolution is considered to be a bottleneck for the SIFT algorithm, SIFT generates 4 octaves per image (640 x 860 pixel) and 3 scales per octave. Also, the number of octaves generated by SIFT depends on the image size and as

Algorithm 3 Listing 3.2 GPU-SIFT Pesudo code

```
1: function FIND_MATCHING(R, A)
2:     Compute SIFT descriptors S(P_1).....S(P_n)
3:     Compute SIFT descriptors S(R)
4:     max_A = 100                          ▷ Where max_A is the maximum distance from that point
5:     min_A = 0                            ▷ Where max_A is the minimum distance from that point
6:     dist = 0
7:     contrast threshold = 40
8:     for i = 1 to n do
9:         for ∀p ∈ S(R) do
10:            if Matching points exist then
11:                Append to FoundMatches
12:                Search space of region A
13:            end if
14:        end for
15:    end for
16:    for j = 1 to FoundMatches do
17:        if dist < min_A then
18:            min_A = dist
19:        end if
20:        if dist > min_A then
21:            max_A = dist
22:        end if
23:        for i = 1 to FoundMatches do          ▷ Algorithmic difference from Alg. 1
24:            if (calc.contrast(P_i)) ≠ contrast threshold then
25:                adaptive.contrast(P_i,contrast threshold)   ▷ Adaptively adjust the contrast level of P_i
26:            end if
27:        end for
28:    end for

29: end function
```

a result, it requires much higher memory to store and process these octaves.

CUDA application controls the specified space memory for every CUDA device by deploying CUDA runtime. This includes the actual device memory allocation and deallocation as well as data transfer between the host and device memory.

GPU-SIFT allocates 4 GB of memory space per CUDA image instead of 2 GB, this helps to increase the copied data amount as input to the kernel from the host to the device. Increasing the allocated device memory from the host to the device should be corresponded by increasing the allocated space to copy result back from the device to the host later.

```
size_t rsize = 1024ULL*1024ULL*1024ULL*4ULL;// allocate 4GB
//Dynamic Memory Allocation
cudaDeviceSetLimit(cudaLimitMallocHeapSize, rsize);
```

- **Run multiple Nvidia kernels using Multi-process Service (MPS):** CUDA

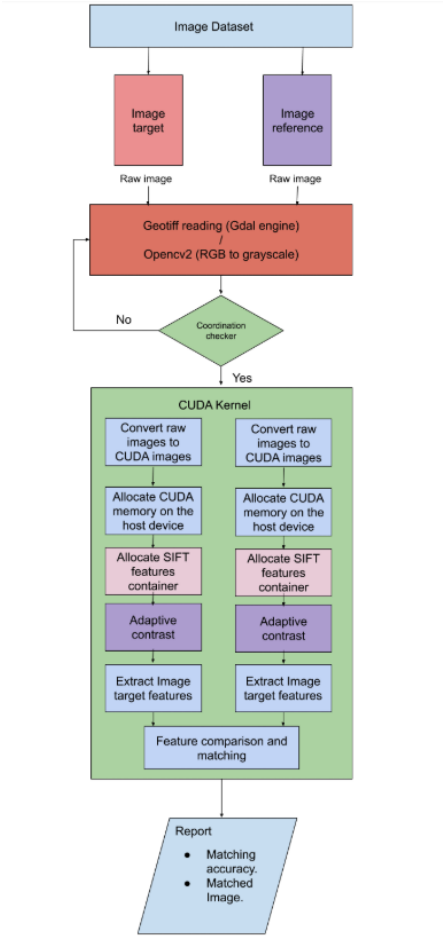


Figure 3.3: CUDA Improved code implementation (adaptive contrast and SIFT feature containers)

Multi-Process Service (MPS) is a feature that allows multiple CUDA processes to share a single GPU context. Each process receives some subset of the available connections to that GPU. This allows the overlapping of kernel and **memcopy** operations from different processes on the GPU to achieve maximum utilization. CUDA program runs in MPS mode if the MPS control daemon is running on the system. Therefore we can improve the GPU utilization with MULTI-PROCESS SERVICE (MPS).

GPU-SIFT takes advantage of MPS to run 2 kernels of original CUDA-SIFT application on a single GPU device instead of one kernel. Based on the performance characterization in section 3.3.1. CUDA-SIFT uses up to 6.5/16 GB of Tesla P100-GPU memory to match 2 images of tile size 5000 x 5000 pixels. Technically

we can run two kernels of 5000 x 5000 pixels images on the same GPU device.

The MPS server creates the shared GPU context, manages its clients, and issues work to the GPU on behalf of its clients. An MPS server can support up to 16 client CUDA contexts at a time with respect to the memory size per GPU. MPS is transparent to CUDA programs, with all the complexity of communication between the client process, the server and the control daemon hidden within the driver binaries.

Below we explain how to implement and setup the CUDA-MPS server (client-host):

- Set the GPU in exclusive mode:

```
sudo nvidia-smi -c 3 -i 0,1
```

- Start the mps deamon (in first window) and adjust pipe/log directory:

```
export CUDA_VISIBLE_DEVICES=${DEVICE}
export CUDA_MPS_PIPE_DIRECTORY=${HOME}/mps${DEVICE}/pipe
export CUDA_MPS_LOG_DIRECTORY=${HOME}/mps${DEVICE}/log
export nvidia-cuda-mps-control -d
```

- Run the application (In second window):

```
mpirun -np 4 ./cuda_sift.sh NGPU=2
lrank=$MV2_COMM_WORLD_LOCAL_RANK GPUID=$(($lrank\%$NGPU))
export CUDA_MPS_PIPE_DIRECTORY=${HOME}/mps${DEVICE}/pipe
```

- Profile the application:

```
 nvprof -o profiler_mps_mgpu$lrank.pdm
./cuda_sift -x1 -x2 -y1 -y2 -x3 -x4 -y3 -y4
```

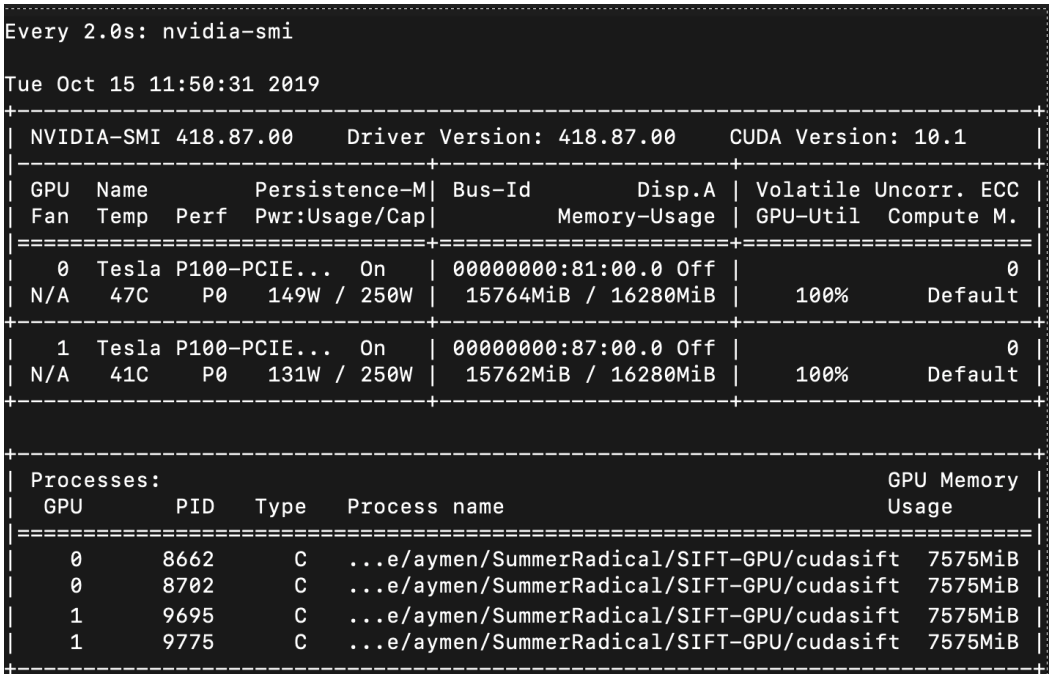


Figure 3.4: nvprof running 4 kernels per 2 GPU using P100 GPU on XSEDE Bridges compute node

- **Reading Geospatial images:** In addition, we implemented the functionality of reading geospatial data such as GeoTIFF imagery using the GDAL package. GDAL can help reading and writing of raster and vector geospatial formats, data format translation, Geospatial processing: sub-setting, image warping, re-projection, mosaicing, tiling, DEM processing [?].

```
#ifdef _GDAL
GDALDatasetH  hDataset;
GDALAllRegister();

hDataset = GDALOpen( argv[count++], GA_ReadOnly );

int width = GDALGetRasterXSize(hDataset);
int height = GDALGetRasterYSize(hDataset);
int bands = GDALGetRasterCount(hDataset);
```

```
int xoff = atoi(argv[count++]);//23000;
int yoff = atoi(argv[count++]);//5000;
w2 = atoi(argv[count++]);
h2 = atoi(argv[count]);

float * iarr2 = (float *) CPLMalloc(sizeof(float)*w2*h2*bands);
GDALDatasetRasterIO( hDataset, GF_Read,xoff,yoff, w2, h2,
                     iarr2, w2, h2, GDT_Float32,
                     bands, NULL, 0,0,0 );
GDALClose( hDataset );



float max = iarr2[0];
for (int i =1;i<w2*h2*bands;i++)
    if (max<iarr2[i])
        max = iarr2[i];
for (int i =0;i<w2*h2*bands;i++)
{
    iarr2[i] = 255.0*iarr2[i]/max;
}

im2 = *new vector<float>(iarr2, iarr2 + w2 * h2);
free(iarr2); /*memcheck*/
```

- **Increase SIFT features containers:** CUDA-SIFT container (vector) size is 35,000 for image size of 1920 x 1980 pixels. Matching GeoTIFF image of size 5000 x 5000 pixels requires larger SIFT containers to store more features. Thus, keeping the SIFT container size to 35,000 limits the number of extracted features from GeoTIFF imagery that the SIFT can store. GPU-SIFT increases the vector size of SIFT containers up to 100,000 allowing for more SIFT features to store

and process.

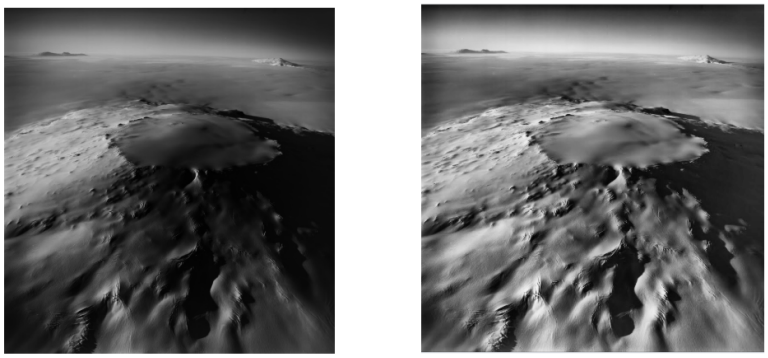
```
std::cout<<"Threshold value :"<<thresh<<std::endl;
InitSiftData(siftData1, 100000, true, true);
InitSiftData(siftData2, 100000, true, true);
```

- **Contrast Enhancement Using Adaptive Contrast:** Satellite, airborne and aerial imagery can be taken under different lighting environments, that can dramatically affect the levels of brightness, and create distribution in the overall contrast quality. This can be a direct reason of why many of the GeoTIFF image matching techniques fail to deliver enough or accurate matching keypoint under these circumstances.

  Our dataset contains 200 airborne, aerial and satellite images that are taken under different lighting environments; thus, we have implemented an adaptive contrast enhancer using Open Source Computer Vision (OpenCV). OpenCV is a library of programming functions mainly aimed at real-time computer vision and image enhancement techniques. Listing 3.2 above shows the pseudo-code of GPU-SIFT using the adaptive contrast enhancer as described later on. The adaptive contrast enhancer works as the following:

  - Breaks the image into tiles and restricts these tiles.
  - Check If any noise exists in these areas then it will be amplified.
  - Decrease the level of contrast for every pixel to a specific limit (by default 40 in OpenCV) .
  - Remove any artifacts in tile borders by applying nearest neighbour technique for every artifact. See Figure 3.5 and Figure 3.6.

  In the image processing context, the histogram of an image normally refers to a histogram of the pixel intensity values. This histogram is a graph showing the number of pixels in an image at each different intensity value found in that image. For an 8-bit grayscale image, there are 256 different possible intensities; thus,



(a) Original contrast (original) (b) Adaptive contrast

Figure 3.5: Effect of adaptive contrast enhancement. The image on the left (a) has unequalized level of contrast. The image on the right (b) is after applying the adaptive contrast feature.

the histogram graphically displays 256 numbers showing the distribution of pixels amongst those grayscale values.

Pixel intensity is the contrast level (grey value) in the image with a range from 0-255, as larger pixel intensity value leads to a darker image. Pixel intensity is an important factor used by the adaptive contrast enhancement technique. The adaptive contrast enhancer usually equalizes the global contrast of the image, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities of every pixel in the image can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast.

The method is useful in images with backgrounds and foregrounds that are both bright or dark. In particular, this method can lead to better views and sharp details in photographs that are over or under-exposed. A key advantage of the method is that it is a fairly straightforward technique and an invertible operator.

Figure 3.9 represents the histogram of the original image (a) plotted in blue and the enhanced image (b) plotted in red. The histogram shows how many pixels are at a given intensity level in the original image compared to the enhanced image.

Since the image is grayscale it has 256 levels of gray and that can be seen from the x-axis of Figure 3.6 that represents the range of pixel intensities (gray values).

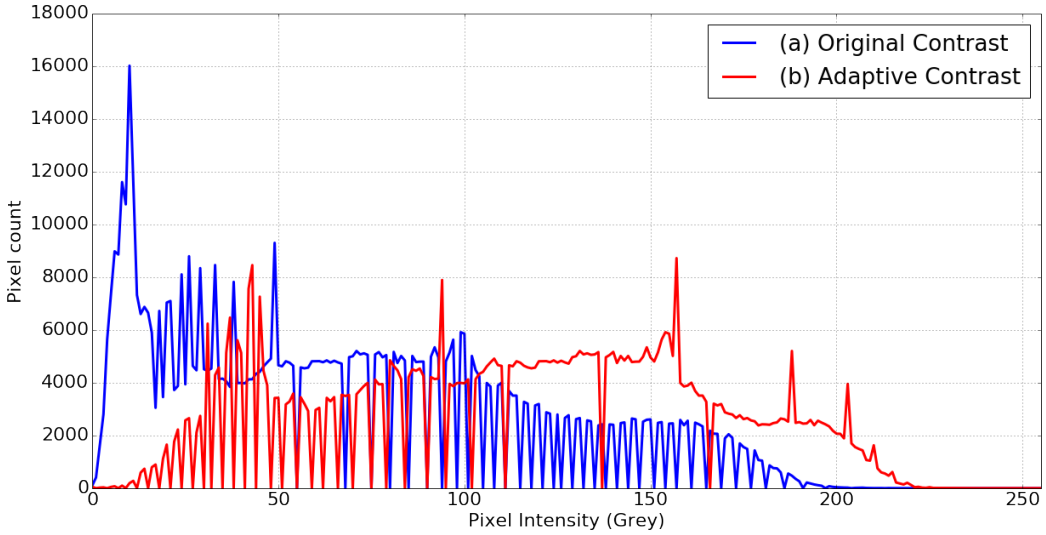
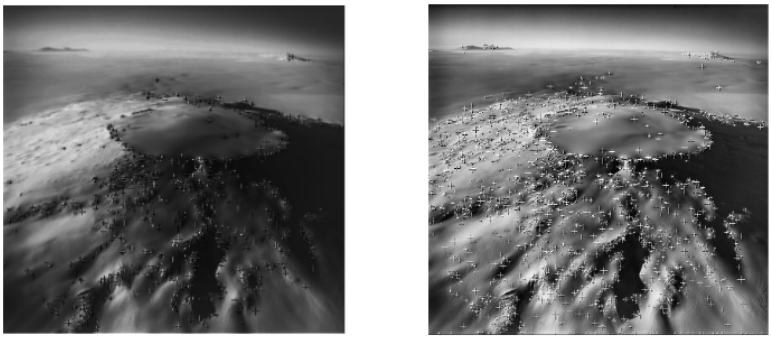


Figure 3.6: Contrast distribution in satellite imagery with and without adaptive contrast enhancement.

The y-axis represents the count of the pixels that have different pixel intensity values.

The contrast enhancement technique fairly utilizes the range of intensity values between the pixels in the enhanced image and the original image, shown in Figure 3.6.



(a) Original contrast (b) Adaptive contrast

Figure 3.7: Number of Matches found in the original image (left) compared to the number of matched found in adaptive contrast image (right).

Figure 3.7. shows that the number of matches found by our implementation was 1457 matches using the adaptive contrast, with a matching rate of 74.81% while in the original contrast the number of matches found was 912 matches with a matching rate of 52%.

The adaptive contrast enhancement helps when the pictures are taken under different lighting environments in different areas not only contrast levels. The mechanism of the adaptive contract enhancement starts by calculating the threshold for each area separately instead of applying one global threshold for the entire image. The threshold value is weighted as the sum of neighborhood values where weights are the level of intensity for every pixel.

3.1.4 Performance characterization

In this subsection, the performance of CPU-SIFT, CUDA-SIFT, and GPU-SIFT is characterized. The performance characterization is done by evaluating the process of matching single pair of an increasing size of GeoTIFF satellite images (source and target) and evaluate their performance in terms of execution time, throughput, memory consumption and level of matching accuracy.

To characterize the performance of the implementations, we focus on three important metrics. The first metric is the throughput which can be defined by how many Megabytes are processed by the CPU or GPU per second or minute.

The second metric is the matching accuracy. To measure the accuracy for CPU-SIFT and GPU-SIFT descriptors we collect the true positive (TP), true negative (TN), false positive (FP) and false-negative (FN) matches to calculate the accuracy. The third metric is the memory consumption in Gigabytes, and it can be calculated by measuring the amount of physical memory that every computational task of each SIFT descriptor consumes for the matching process.

We characterize the physical memory consumption for each descriptor with an increased tile size of 2000 pixels - 5000 pixels for source and target images. No previous research has investigated the performance of these descriptors with a tile size of more than 1920 x 1080 pixels. It is worth noting that the memory consumption is a function of the tile size of both source and the target image.

3.1.5 Throughput

In this subsection we calculate the throughput (images/second) of CPU-SIFT, CUDA-SIFT, and GPU-SIFT using different image size in Megabytes. CPU-SIFT and CUDA-SIFT can run one application per CPU/GPU, while GPU-SIFT can run two applications per GPU using CUDA MPS technology as we described in subsection 3.1.3.

In order to show a valid comparison between the three implementations of how much data every implementation can process per second, we execute two CPU-SIFT applications on two CPUs, two CUDA-SIFT on two GPUs concurrently and one GPU-SIFT (two applications) on one GPU. Figure 3.8 shows that CPU-SIFT (red) has the lowest throughput with a value of 54.38 MB/sec. CPU-SIFT low throughput is due to code sequentially, CPU-SIFT uses one core per CPU to process and match two images. As a result, the amount of data being processed per second is relatively small.

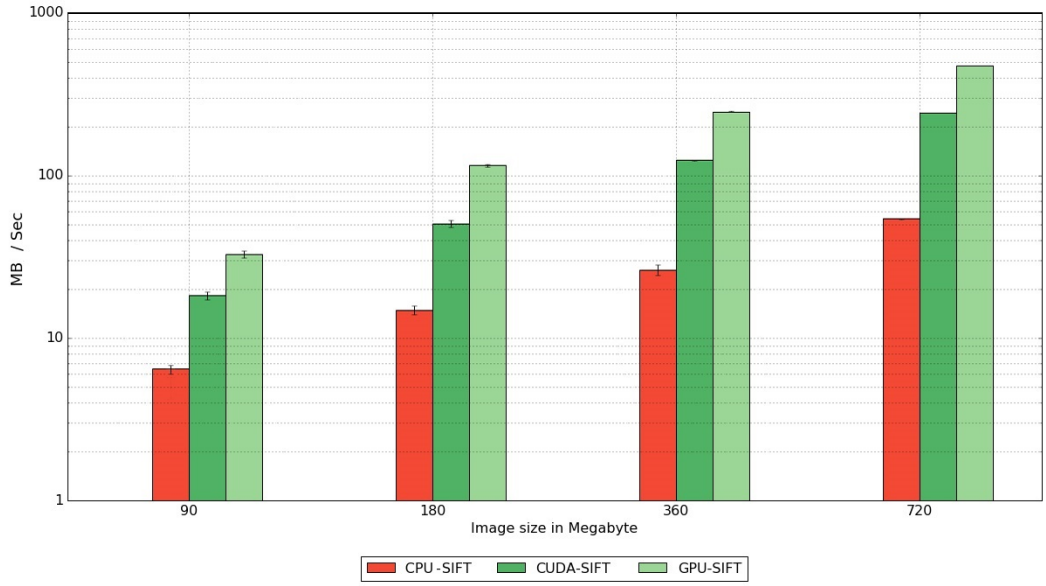


Figure 3.8: The throughput of CPU-SIFT, CUDA-SIFT and GPU-SIFT as a function of image size in Megabytes.

CUDA-SIFT (dark green) shows higher throughput with a value of 244.44 MB/sec. CUDA-SIFT higher throughput is due to the GPU device number of cores (3584 CUDA cores) and the load balancing that the CUDA framework performs among their CUDA cores. CUDA-SIFT utilizes around 1658/3584 cores to read, load and match 2 satellite images.

GPU-SIFT throughput (light green) is shown in Figure 3.8. GPU-SIFT has the highest throughput value of 478.25 MB/sec. GPU-SIFT uses 3400/3584 CUDA cores to run 2 CUDA applications concurrently. The performance analysis is done using the Intel Xeon processor of 3.4 GHz CPU and Nvidia P100 GPU on XSEDE Bridges supercomputers.

Finally, our results show that the GPU throughput is invariant to the image size, as we show in Figure 3.4, as increasing the image size increases the amount of data processed by the GPU or CPU per second with respect to the CPU/GPU bandwidth size. It is worth mentioning that the error bars for the GPU implementations are relatively small and that indicates the stability in execution time for these kernels compared to the CPU kernels which are relatively large.

3.1.6 Memory consumption

In this subsection, we characterize the physical memory consumption of CPU-SIFT, CUDA-SIFT and GPU-SIFT. Memory consumption stands for the amount of physical memory a particular program utilizes during the run time. The memory consumption in Figure 3.9 represents the physical memory being consumed by both kernels. To measure the memory consumption for both kernels we use pre-implemented c++ default libraries as follows:

```
long long physMemUsed = memInfo.totalram - memInfo.freeram;
physMemUsed *= memInfo.mem_unit;
```

Figure 3.9 shows the total memory consumption for CPU-SIFT plotted in red, GPU-SIFT plotted in light green and CUDA-SIFT plotted in dark green. The memory usage includes image reading, detecting, extracting, and matching SIFT features.

In order to compare GPU-SIFT with CUDA-SIFT and CPU-SIFT, we run 2 kernels concurrently on two different CPUs for CPU-SIFT and on two different GPU devices for CUDA-SIFT to make the comparison valid. Figure 3.9 shows CPU-SIFT memory consumption is relatively high with a value of 24.67 GB compared to CUDA-SIFT with

a value of 4.054 GB and GPU-SIFT with a value 8.08 GB per 2 kernels for the 5000 pixels.

Comparing GPU-SIFT to CUDA-SIFT, the latter shows an increase in the total memory usage for both kernels due to the relatively consistent MPI server communications overheads for each kernel running separately [30].

CPU-SIFT overheads fluctuate between 2000 and 3000 pixels and are relatively small with a value of 0.084±0.0023 GB, and relatively higher between 4000 and 5000 pixels with a value of 0.157±0.0113 GB. CUDA-SIFT and GPU-SIFT overheads are relatively small and consistent with values of 0.00131±0.0039 GB and 0.00182±0.0021 GB between 2000 and 3000 pixels and start remain consistent between 4000 and 5000 pixels with a value of 0.017±0.0119 GB. CPU-SIFT overhead shows inconsistency compared to CUDA-SIFT and GPU-SIFT due to implementation differences, type, and model of the used devices CPU or GPU.

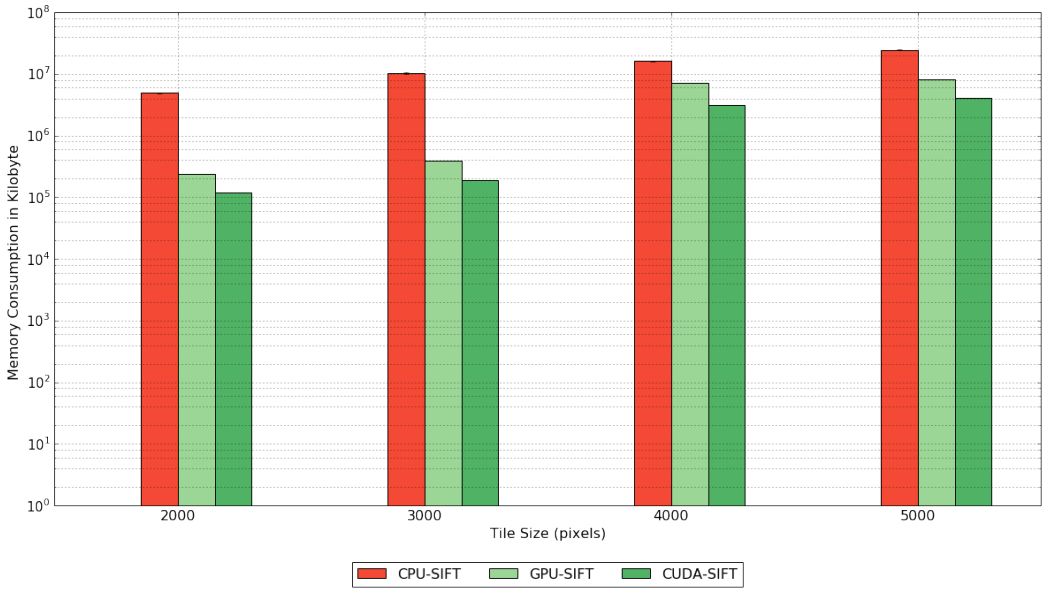


Figure 3.9: Memory consumption of CPU-SIFT, CUDA-SIFT and GPU-SIFT as a function of tile size

3.1.7 Numbers of Matched points

In this subsection, we characterize the performance of three implementations in terms of the number of matching points. We picked a well-known matched and satellite image

of a size 60,000 pixels by 60,000 pixels.

The image contains 16850 true positive matches that have been identified previously. We apply CPU-SIFT, GPU-SIFT and CUDA-SIFT on the pair of images with different tile sizes to measure the number of matches that the three implementations can detect between both images and validate the assumption that the tile size can affect the number of matches.

Correct / Incorrect Matches	Match Found	Match Not Found
Correct Matches	True Positive (TP)	False Negative (FN)
Incorrect Matches	False positive (FP)	True Negative (TN)

Table 3.2: shows the correct and incorrect matches corresponding to false and true positive, negative.

The equation of accuracy can be given as follows:

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP} \tag{3.1}$$

Figure 3.10 shows the number of matches that CPU-SIFT, GPU-SIFT, and CUDA-SIFT detected (right figure) and the accuracy level in percentage (Left Plot). To measure the accuracy of both implementations we repeat the experiments up to 75 times using the same satellite image with a range of tile size of 2000, 3000, 4000 and 5000 pixels.

Figure 3.10 (left) shows that the number of matched points detected by CPU-SIFT is more than that by GPU-SIFT or CPU-SIFT, reaching 13500/16850 correct matches with an accuracy of 80.11% using 5000 pixels tile size. It is about 5.92% more than GPU-SIFT, and 8.97% more than the CUDA-SIFT. GPU-SIFT plotted in green reaches about 12500/16850 correct matches with an accuracy of 74.18%, while CUDA-SIFT plotted in orange shows that the number of matches is 9500/16850 with an accuracy of up to 56.81%.

It's worth mentioning that CPU-SIFT and GPU-SIFT apply the contrast enhancement to both source and target image before processing them, while CUDA-SIFT does not apply the contrast enhancement. In conclusion, GPU-SIFT reaches a better number

of matches and in return, it reaches an accuracy level of 74% compared to the CUDA-SIFT with 56.81%. The accuracy characterization of CPU-SIFT and GPU-SIFT validates our experiments that enhancing the level of contrast of an image can participate in increasing the numbers of detected keypoints and as a result, it directly increases the number of matches between both images.

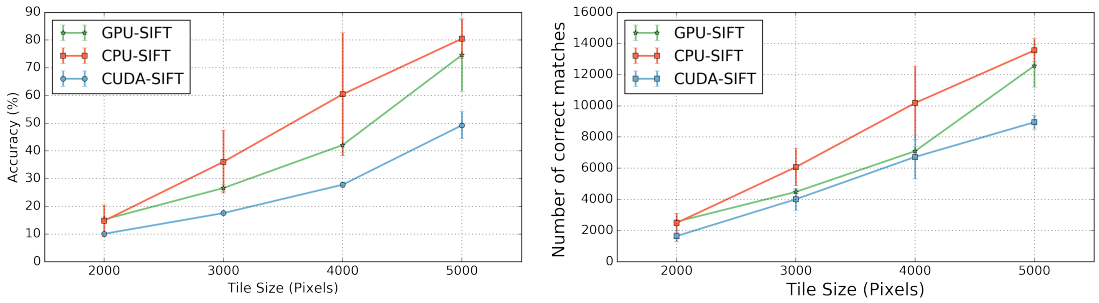


Figure 3.10: shows the matching accuracy (Left) and the total number of correct correspondences (Right) found with GPU-SIFT, CUDA-SIFT and CPU-SIFT for well-known satellite image as a function of tile size.

Finally, CPU-SIFT overheads shown in Figure 3.10 are high and inconsistent compared to CUDA-SIFT and GPU-SIFT. It is unclear why CPU-SIFT is inconsistent in terms of the number of matches and accuracy, this inconsistency in terms of accuracy and number of found matches make the CPU-SIFT accuracy level unpredictable and affects the certainty of how accurate the global level of accuracy for the entire image geolocating process.

Chapter 4

Implement and execute large scale ensemble-based image geolocating workflows using RADICAL-Cybertools

In this chapter we discuss RADICAL-Cybertools (RCT), ensemble toolkit (EnTK), RADICAL-Pilot (RP), and their importance in enabling scalable and concurrent image geolocating execution on remote HPC machines. Additionally, we present and discuss the concurrent execution of multiple image geolocating workflows using the ensemble toolkit workflow system. We conclude with 4 sets of experiments to evaluate the strong and weak scaling properties of workloads comprised of 196 high resolution airborne and satellite images provided by our driving use case [**?**].

4.1 RADICAL-Cybertools

RADICAL-Cybertools (RCT) are a set of software components that support the description and effective execution of scientific jobs, workloads and workflows. Each component of RCT can be used as a self-contained software system or as a building block of an end-to-end middleware solution. In this way, RCT support a vast array of use cases from diverse scientific disciplines, enabling the execution of scientific payloads on multiple HPC resources. Currently, RCT consists of three software systems: RADICAL-SAGA (RS), RADICAL-Pilot (RP) and RADICAL-EnsembleToolkit (EnTK).

RS is a Python implementation of the Open Grid Forum (OGF) standard for a unified interface to multiple batch job systems used by HPC facilities. RP is a Python implementation of the pilot paradigm [**?**], a system that separates the acquisition of HPC resources from scheduling and executing computing tasks on those resources. EnTK is a python implementation of a workflow engine, specifically designed to support the description of ensemble applications, and to manage their execution via RP on HPC

resources. The implementation of our image geolocating workflow uses two components of RCT: RP and EnTK.

RP performs as the runtime system of EnTK by: (1) acquiring the requested HPC resources (2) setting up the necessary environment variables; (3) installing all the required software dependencies; and (4) scheduling, placing and launching computing tasks on the available HPC resources. EnTK offers the capability to create and execute ensemble-based workflows, without the need for explicit resource management from the user. In this subsection, we briefly discuss and explain the details of RP and EnTK, describing their deployment and how to enable the scalable execution of our multiple image geolocation workflow pipeline.

4.1.1 RADICAL-Pilot

Executing a large number of scalable tasks with data dependencies present several challenges. In general, two methods are used to execute multiple tasks in parallel on any resource: (1) concurrently executing tasks (i.e., executables), gathering their outputs once all have executed; or (2) executing and single task (i.e., executable) in which multiple processes execute in parallel, exchanging messages via the Message Passing Interface (MPI).

Pilot systems [**?**] support the concurrent execution of multiple tasks by using multi-level scheduling of jobs on the HPC infrastructures and tasks on those resources. The pilot abstraction separates resource acquisition and resource usage for task execution: a job placeholder is used to acquire HPC resources. Once those resources becomes available, a software agent is executed to enable task pulling, scheduling, placement and execution. In this way, RP directly executes tasks into the acquired resources, rather than through the system's job scheduler. This increases task throughput by avoiding each task to wait in the queue of the HPC system's scheduler.

The pilot abstraction supports the requirements of task-level parallelism and high-throughput as needed by scientific applications, without affecting the queue time of HPC resources. By implementing this abstraction, RP supports scalable and effective launching of heterogeneous tasks across different HPC machines.

4.1.2 Ensemble Toolkit

EnTK offers the ability to create and execute ensemble-based workflows, without the need for the user to explicitly manage resources. EnTK exposes an API with pipelines, stages, and tasks. EnTK implements Task as a self-contained executable, stage as that includes a set of tasks, and Pipeline as a sequence of stages. The formal properties of sets and sequences are used to specify the relation of priority among tasks: tasks in a stage can be executed concurrently while stages in a pipeline have to be executed sequentially.

In this way, tasks of different stages are guaranteed to be executed sequentially while tasks of the same stage can be executed concurrently if enough resources are available. EnTK API avoids the need to express the explicit relationship among tasks in the form of, for example, a Directed Acyclic Graph (DAG). Further, EnTK enables the concept of pre and postcondition of task execution, enabling workflow adaptivity both at the local and the global level [**?**]. Adaptivity allows expressing elaborated coordination patterns among tasks and pipelines as done, for example, with the implementation of the replica-exchange coordination pattern [**?**].

EnTK architecture has three main components: Application Manager, Workflow Processor, and Execution Manager. Application Manager allows specifying the target resource for the execution of the ensemble-based application. In addition to that, there are other properties that can be specified such as the number of remote nodes, number of CPUs, number of cores, GPUs and resource credentials. These properties are variables that can allow tuning the performance of EnTK depending on the number of tasks, stages, pipelines and the number of resources available to the toolkit.

By traversing sets and sequences, Workflow Processor transforms the application workflow into workloads, i.e., set of tasks that can be concurrently submitted to the runtime system and, as with RP, concurrently executed when enough resources are available. **Execution Manager** is responsible for resource acquisition and execution management of the workload see Figure 4.1.

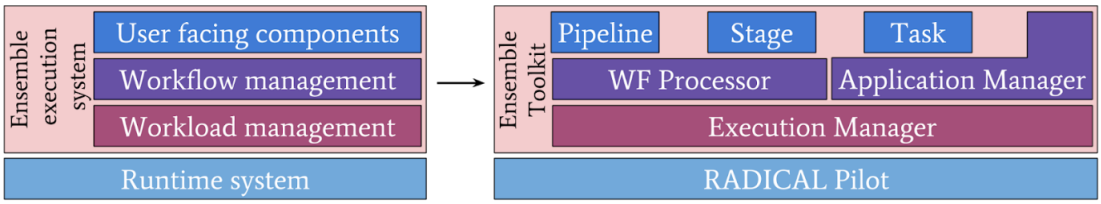


Figure 4.1: A High level representation of the Ensemble Toolkit design showing the main components and their structure

4.1.3 Image Geolocating Use Case

A workflow is described as a set of tasks with data and/or control dependencies that determine the order of their execution. Our image geolocating pipeline consists of a workflow with four different tasks, each with data dependencies and with different concurrency requirements.

Figure 4.2 depicts our workflow, consisting of 4 stages, each containing one task except for stage 1 which contains two dependent tasks. Below, we describe in detail every task in the image geolocating pipeline.

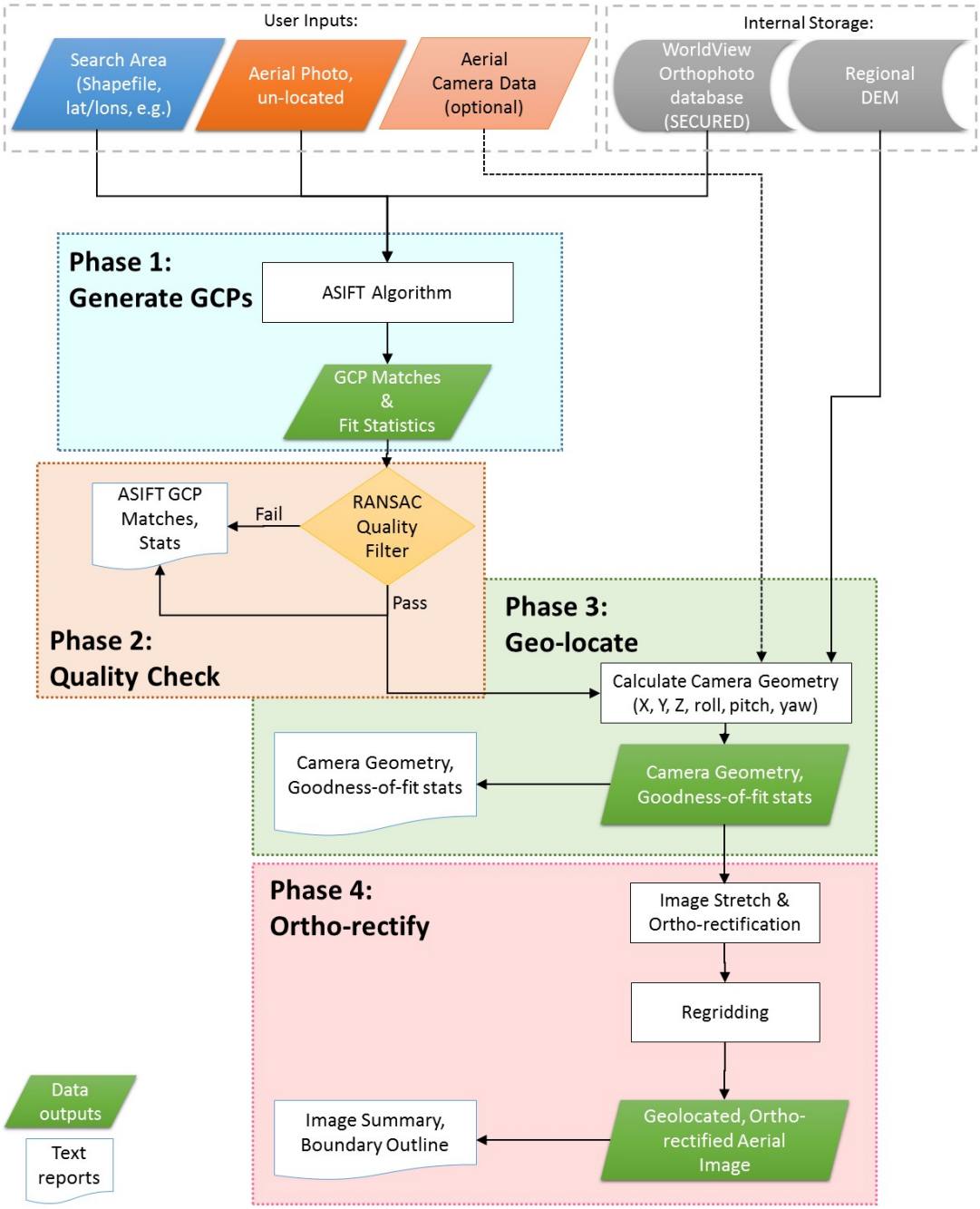


Figure 4.2: shows the the phases of image geolocating workflow

4.1.4 Image Prepossessing

The first phase in the workflow is called "Image preprocessing", with two data-dependent tasks: tile planner and run case planner. The first task, tile planner reads all the images

of the dataset-extracting the number of bands in every image. In addition, the tile planner task tiles the entire dataset by dividing every aerial and satellite image into smaller rectangular tiles, all of the same size. Code listing **L.4.1** shows the dataset setup and the tiling process.

```
for path, dirs,src_images in os.walk(args.path):
    for img in src_images :
            if img.startswith("CA") and  img.endswith(".tif"):
                print ('found Source GEOTIFF Images '+path+'/'+img)
                img1=cv2.imread(path+'/'+img)
                crop_img = img1[x1:x2,y1:y2]
                cv2.imwrite(aerial_path+img,crop_img)
            elif img.startswith("WV") and  img.endswith(".tif"):
                 print ('found Source GEOTIFF Images '+path+'/'+img)
                 img1=cv2.imread(path+'/'+img)
                 crop_img = img1[x1:x2,y1:y2]
                 cv2.imwrite(sat_path+img,crop_img)
```

Listing 4.1: Image cropping and tiling

The tiling process is implemented to create a set of tiles optimized to stay within the range, provided by the domain scientists, of 2000–5000 pixels and to avoid the tiles that are at the edge of images. When needed, this task has an additional tiling technique called "Image pyramids" to reduce the processing time of high-resolution images. The pyramid tiling generates a set of concentric tiles in which each tile is half the size of the previous tile.

Run case planner, the second task of the first phase of the workflow, iterates through all the image tiles generated by the first task, creating a list of tiles names saved to a JSON or CSV file (see code listing **L. 4.2**). This file contains the name of both source and target tiles: source tiles are combinations of aerial and airborne images, while target tiles are always satellite images. Information about these tiles includes the x and y coordinates of every tile as well as the tile path in the database. In this way, every

single aerial or airborne image is cross-matched with all the satellite images that are in the dataset.

```
    tmp_dict = {}
                tmp_dict["img1"] = args.source_img
                tmp_dict["img2"] = path + filename
                tmp_dict["x1"] = img1.shape[0]
                tmp_dict["y1"] = img1.shape[1]
                tmp_dict["x2"] = img2.shape[0]
                tmp_dict["y2"] = img2.shape[0]
                data.append(tmp_dict)

json_dict["Dataset"] = data
```

Listing 4.2: Building the dataset of image geolocating, by reading the images and distinguish the source image from target image based on the name of the image (Satellite, Airborne image etc.)

4.1.5 Generating and Matching Keypoints

In this phase, every single aerial or airborne tile is matched with all satellite tiles. The main task of this phase is to extract all known features from every source and target tile by creating an overlap between all tiles.

Feature matching has two main steps: keypoint detection and keypoint matching. Keypoint detection identifies and extracts a set of local feature keypoints from the source and the target tile, showing high invariance in rotation, translation, etc. This step can be tuned based on a set of parameters such as the number of octaves, blur level, contrast level or the threshold value of the total number of features.

Keypoint matching matches the local features extracted from both tiles. This is a simple step where the similarity between the source and the corresponding tile is

measured as the total number of matched keypoints. A larger number of matched keypoints indicates a bigger probability that both images are similar. Feature extraction and matching use the SIFT technique [?], currently, we support both CPU-SIFT [?] and an accelerated and modified version of GPU-SIFT for Geotiff satellite imagery based on GPU-SIFT work by Mårten Björkman [?].

The GPU-SIFT represents part of our research contribution. We implemented GPU-SIFT to run larger tiles of Geotiffs images. GPU-SIFT provides certain advantages compared to CUDA-SIFT, such as accurate matching, robust feature detection, and fast feature extraction and matching. The use of a GPU or CPU kernel depends on the amount of the available computational resources on every HPC node: concurrently running multiple image geolocation pipelines requires large amounts of computational resources such as memory, CPU cores, and GPU devices. The keypoints generation phase is the core process of the proposed geolocating pipeline, as a higher and more accurate number of matched keypoints increase the accuracy level of the geolocating process.

4.1.6 RANSAC Keypoints filtration

The SIFT algorithm can generate a good set of matches between two images but mismatched keypoints remain a problem. Removing both false and positive mismatches requires finding the transformation matrix of the keypoints. Using the Random Sampling Consistency (RANSAC) algorithm helps to eliminate the possible mismatches and increasing the accuracy of the total matches between both tiles.

This stage includes two main steps. The first step is applying the RANSAC on the extracted set of matches between source and target tiles. The input file *datamatches.csv* of this step is from stage 2 with all coordinates for both matches from source and target tiles. The output is another file *ransac.csv* that contains the final set of filtered matches. The second step recombines the accurate keypoints from the source and the target tiles, performing another RANSAC filter to maintain internal geometric consistency between the image matches.

4.1.7 Geolocating Keypoints

After stage 2 extracts and matches image features and and stage 3 removes the mismatches from each pair of tiles, stage 4 refines camera parameters using the set of the final matched keypoints to perform camera calculations. Achieving an accurate geolocation between the matched tiles in the database requires camera parameterization followed by forward geo-referencing [**?**].

The proposed geolocation method includes executing two steps in parallel: target geolocating and altitude estimation. The accuracy of the geolocation process depends on the accuracy of the matching between aerial and satellite imagery, and on the amount and accuracy of the information contained in the geo-referenced terrain database. The output of this stage is a set of geographic coordinates with an estimated location for the matched tiles.

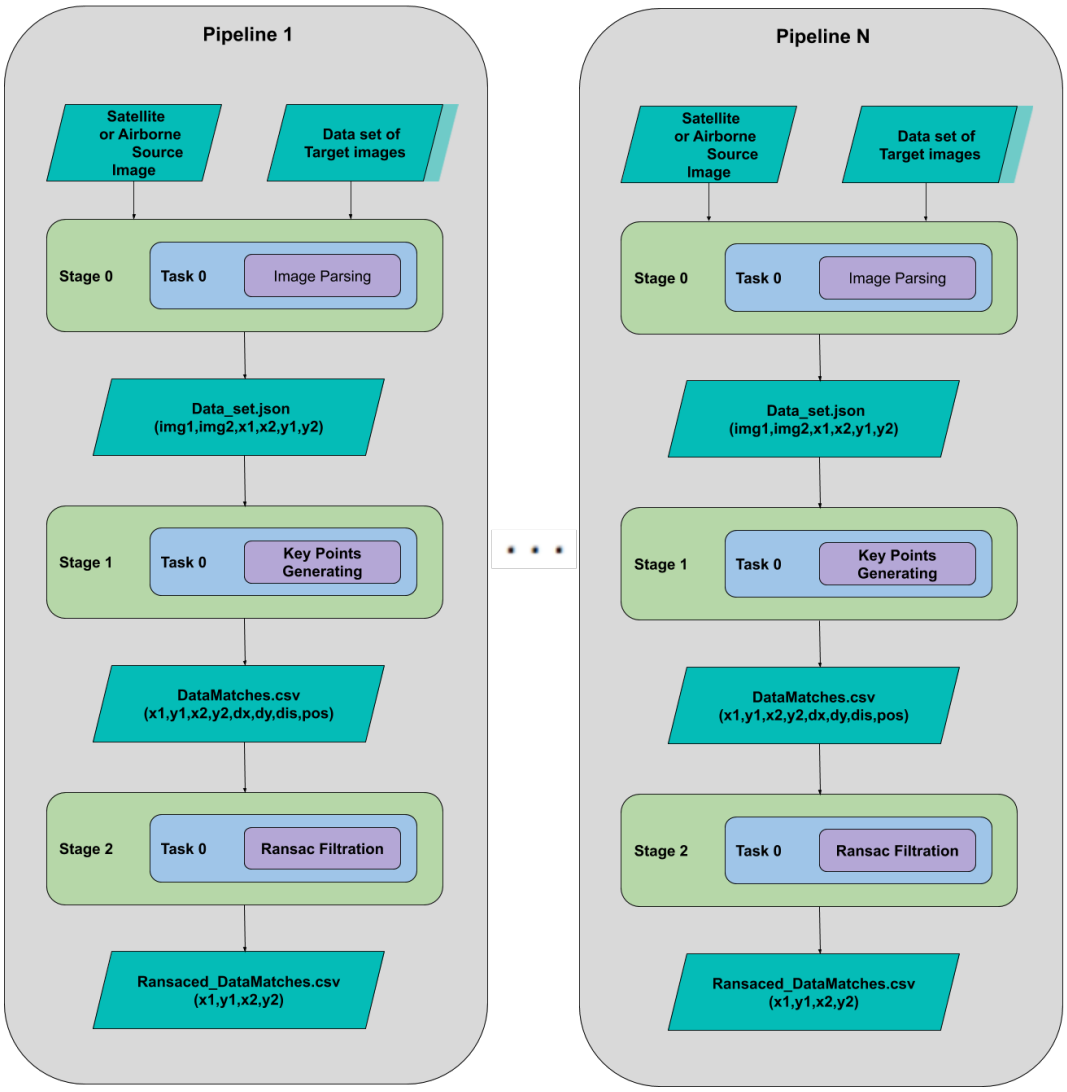


Figure 4.3: shows the workflow design of EnTK to run multiple image geolocating pipelines on HPC resources.

Figure 4.3 shows the components of image geolocating pipeline using EnTK. EnTK model an application by combining the Pipeline, Stage and Task (PST). The PST model, have the following objects:

- **Task:** an abstraction of a computational task that contains information regarding an executable, its software environment and its data dependencies.
- **Stage:** a set of tasks without mutual dependencies and that can be executed concurrently.

- **Pipeline:** a list of stages where any stage 'i' can be executed only after stage '$i-1$' has been executed.

Every image geolocating pipeline can match and orthorectify 2 images. The number of pipelines being executed concurrently depends on the size of the dataset that we are trying to geolocate at once. Our PST model has 3 stages, each stage contains a single task and every task can have certain pre-execution libraries and packages that can be set up during the PST model implementation to load before executing the task. In the next subsection, we characterize the performance of our image geolocating pipeline on XSEDE Bridges supercomputers.

4.1.8 Performance Characterization Experiments

ID	Experiments Type	Tile Size	Number of Images	Total Number of Cores
1	CPU-SIFT Strong Scaling	2000, 3000, 4000, 5000	224	28, 56, 112, 224 cores
2	GPU-SIFT Strong Scaling	2000, 3000, 4000, 5000	224	2 GPUs, 4 GPUs, 8 GPUs
3	CPU-SIFT Weak Scaling	2000, 3000, 4000, 5000	28, 56, 112, 224	28 core, 56 core, 112 core ,224 core
4	GPU-SIFT Weak Scaling	2000, 3000, 4000, 5000	28, 56, 112, 224	2 GPUs, 4 GPUs, 8 GPUs

Table 4.1: Experiment type on both CPUs and GPUs.

Table 4.1 shows four experiments we design to characterize the behavior of the image geolocating pipeline on the XSEDE PSC bridges supercomputers. Each experiment executes the image geolocating pipeline on a different number of CPU cores and GPUs. Experiments 1 and 2 measure the weak scaling of the image geolocating pipeline on a supercomputer, using both GPU-SIFT and CPU-SIFT. Experiments 4 and 5 measure the strong scaling of the same image geolocating pipeline with a fixed number of image pairs, also using both GPU-SIFT and CPU-SIFT.

In weak scaling Experiments 1 and 2, we keep the ratio fixed between the number of tasks and the numbers of available resources. We start from 28 cores on XSEDE bridges supercomputer (one compute node) and scale up to 256 cores (eight compute

nodes, i.e., the maximum number of compute nodes available with Tesla P100 GPU devices). For both GPU-SIFT and CPU-SIFT, we progressively increase the number of resources and the number of tasks by a factor of 2. In Experiments 3 and 4 we perform strong scaling, keeping in each experiment the number of tasks fixed and increasing the resources (number of cores) by a factor of 2.

Weak scaling experiments measure the speedup for a fixed problem size with respect to the number of cores, providing a better understanding of how the size of the workload correlates with its execution time. Strong scaling experiments show how the execution time of the same workload scales with the number of available resources. For both strong and weak scaling experiments, we characterize the overheads of EnTK and RP, measuring the time taken by the entire image geolocating pipeline. Here the definition of the metrics of our experiments:

- **Total Task Execution Time (TTX):** Time taken by all the task executables to run on the computing infrastructure.
- **Task 1 TX:** The Time taken by image parser 1 to parse and generate all image tiles.
- **Task 2 TX:** The Time taken by CPU-SIFT or GPU-SIFT to match all pairs of images.
- **Task 3 TX:**The Time taken by Ransac filter to eliminate undesired points.
- **EnTK and RP Overhead:** Time taken by EnTK and RP to manage the execution of tasks.
- **Time To Completion:** The total time taken by RP and EnTK to manage the execution of the tasks added to the time taken by the task executable to run and finish execution.
- **Accuracy:** Matching accuracy between source and target image.

4.1.9 Strong Scaling characterization

In Experiment 1, we fix the number of tasks in the image geolocating pipelines to 224 tasks and increase the number of resources (cores) from 28 cores, 56 cores, 112 cores up to 224 cores respectively. Based on the CPU-SIFT implementation we present in Chapter 3 in Figure 3.6 every CPU-SIFT task requires one CPU-core to execute. Given the number of available resources, we can concurrently execute 224 tasks on a total of 224 cores.

Figure 4.4 shows the strong scaling for CPU-SIFT image geolocating pipelines while Figure 4.5 shows the strong scaling of GPU-SIFT image geolocating pipelines for the workload described in Table 4.1. The CPU and GPU image geolocating workflow components are shown in Figure 4.3. The workflow is composed of 3 tasks: Task 1: image parsing, Task 2: image matching and Task 3: Rasnac filtration.

Based on the performance characterization shown in Chapter 3 Figure 3.6, Task 2 can take 15 minutes to match the pair of images of size 5000 pixels using CPU-SIFT and 2 minutes to match the same pair of images using GPU-SIFT. Task 1 in the workflow is responsible for parsing and tiling the images into specific tile size while Task 3 is responsible for eliminating the undesired matched points. Tasks 1 and 3 execution time in the workflow is not related to the tile size, experiments 1 and 2 show that Task 1 takes 15s$\pm$0.3 to execute while Task 3 5s$\pm$0.03.

Figure 4.4 shows that in every datapoint on the x-axis there are 6 bars between 28 and 224 cores, TTX (orange) represents the total time to execute Task 1, Task 2 and Task3. RP and EnTK overhead (Blue and Purple) represent the amount of time added to TTX to schedule, submit and execute Task 1, Task 2, and Task 3 on XSEDE Bridges supercomputer. The last 3 bars (green) from left to right represent the execution time of Task 1, Task 2 and Task 3 respectively without any overheads. Comparing Task 1 and Task 3 to Task 2 in terms of execution time shows that Task 2 is dominating most of the TTX in Experiments 1 and 2.

Figure 4.4 shows that TTX scales sublinearly for images with 2000 pixels, 3000 pixels and 4000 pixels and almost linearly for images with 5000 pixels. Figure 4.3 also shows

the overheads of both RP are slightly increasing with the number of cores with a value of 44s±1, confirming that RP performance decreases with the pilot size as measured in [18]. EnTK overhead is relatively constant across the number of cores with a value of 136s±3, confirming that its performance is independent of the size of the pilot used to execute a workflow [**?**].

Figures 4.4 also shows that CPU-SIFT does not scale well while increasing the number of cores. The bottleneck of CPU-SIFT implementation execution is always found to be the heavy read/write that the code is performing from memory as shown in Chapter 3 Figure 3.7. CPU-SIFT performs 20 floating operations per keypoint operation on both images in the memory and that can increase the number of floating-point computations performed for each memory operation.

In addition, CPU-SIFT poor scaling can be due to the lack of code parallelization, as the entire code is not multithreaded, i.e., the entire code is using one thread per task per core. This creates a performance bottleneck in image geolocating workflow and leads to three times longer execution time compared to the parallelized version (GPU-SIFT), as shown in Chapter 3 Figure 3.6.

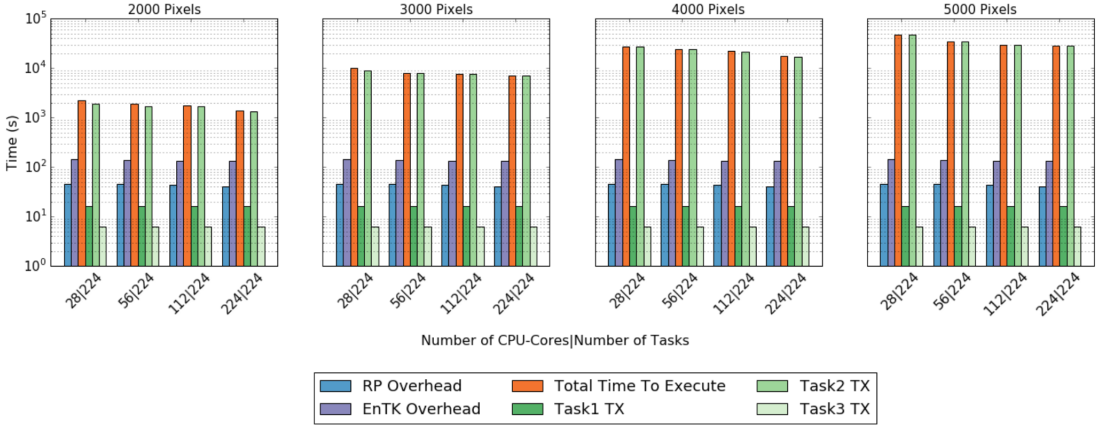


Figure 4.4: Strong scaling of Image geolocating pipelines using CPU-SIFT for experiments (1).

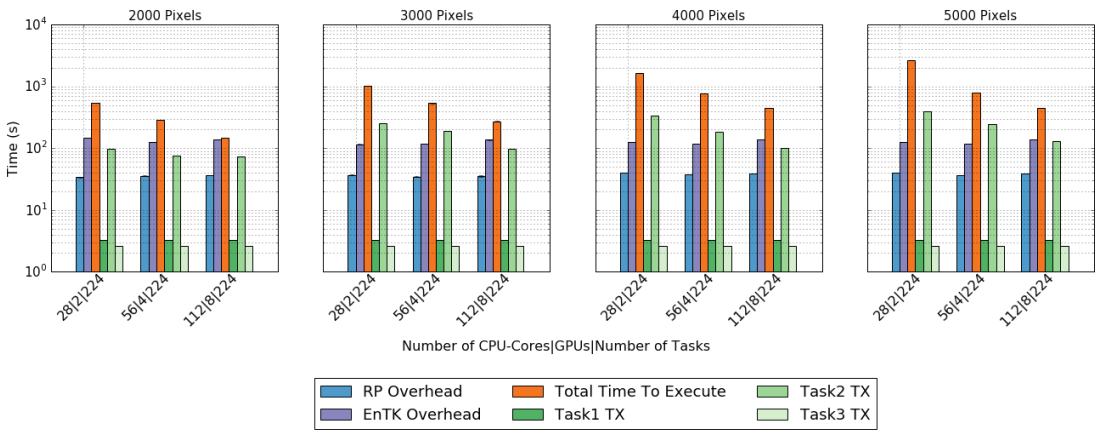


Figure 4.5: Strong scaling of Image geolocating pipelines using GPU-SIFT for experiments (2).

In Experiment 2 we changed the CPU-SIFT implementation in the second task with our GPU-SIFT. We set the total number of tasks to 224 as in Experiment 1 and, differently from Experiment 1, Task 1 and Task 3 require CPU while Task 2 requires a GPU to run and execute. GPU-SIFT is able to run 2 GPU tasks on the same GPU device concurrently compared to CUDA-SIFT which allows only one executable to run per GPU device. As a result, the ideal concurrency that can be achieved with the optimized GPU-SIFT is 16 GPU tasks and 4 CPU tasks on 8 GPUs / 4 CPUs. Due to the limited amount of GPU-nodes available on XSEDE Bridges, we can only request up to 4 GPU-nodes (for a total of 8 GPUs).

Figure 4.5 shows excellent linear scaling in TTX of the GPU-SIFT image geolocating pipeline, proportional to the increase in the number of GPUs and CPUs. We expect the GPU-SIFT image geolocating pipeline to scale better with the availability of more GPUs for a fixed number of tasks. Overheads remain essentially constant for both RP and EnTK with a value of 47s$\pm$2s and 136s$\pm$4s respectively when increasing the number of GPUs and CPU cores. RP overhead is scaling linearly with the number of cores for 2000, 3000 and 4000 pixels and sublinearly between 28 and 56 cores and linearly between 112 and 224 cores for 5000 pixels.

4.1.10 Weak Scaling characterization

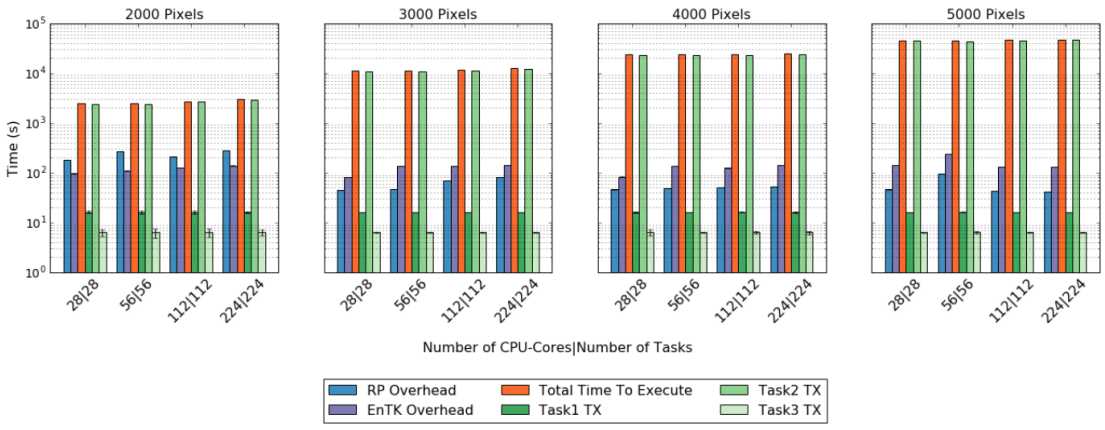


Figure 4.6: Weak scaling of Image geolocating pipelines using CPU-SIFT for experiments (3).

Figure 4.6 shows the weak scaling for CPU-SIFT image geolocating pipelines for the workload described in Table 4.1. The ratio between the total number of tasks executed and cores is constant Ideally, this will enable full concurrent task execution.

Figure 4.6 shows the TTX plotted in orange and the dominating task (CPU-SIFT) in green scales linearly in 2000 pixels and 3000 pixels, linearly between 28 and 56 cores, and sublinearly between 112 and 224 cores for 4000 and 5000 pixels. The average value of TTX between 28 cores and 224 cores is 26571±11s, 11710±23s, 23870±51s, 45553±21s for 2000, 3000, 4000 and 5000 respectively. RP and EnTK overheads are scaling linearly with the number of cores for 2000 and 3000 pixels and mostly constant for 4000 and 5000 pixels.

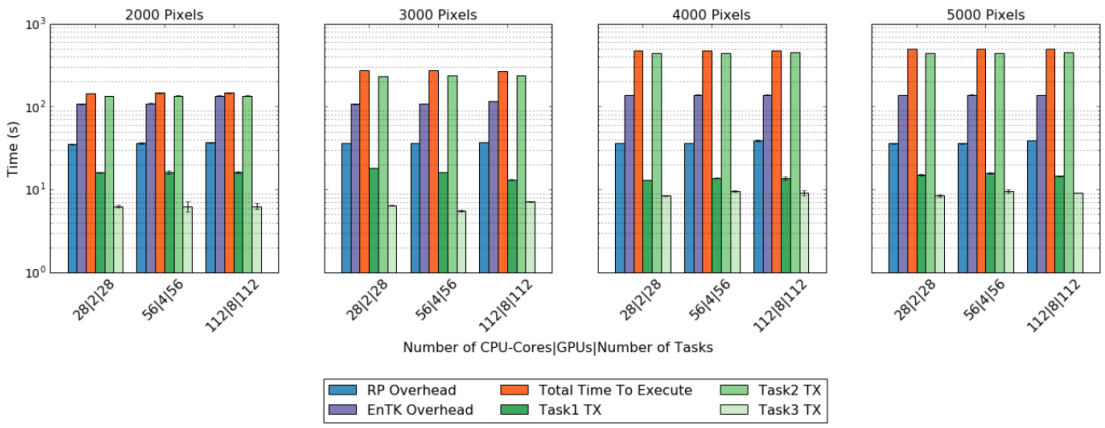


Figure 4.7: Weak scaling of Image geolocating pipelines using GPU-SIFT for experiments (4).

Figure 4.7 shows the weak scaling for GPU-SIFT image geolocating pipelines for the workload described in Table 4.1. The figure shows the TTX plotted in orange and the dominating task (GPU-SIFT) in green scales linearly with the number of GPUs and CPUs in 2000 pixels, 3000 pixels, 4000 and 5000 pixels. The average value of TTX between 2 GPU/28 cores and 8 GPUs/224 cores are 144.930708 ±2s, 272±2s, 475±2s,497±5s for 2000, 3000, 4000 and 5000 respectively. RP and EnTK overheads scale linearly with the number of cores for 2000 and 3000 pixels and is mostly constant for 4000 and 5000 pixels.

Finally, we conclude that, for polar and geosciences, there is an existing and demonstrated need to support large scale MPI and non-MPI tasks as part of a single workload. The scalable execution of workloads comprised of many heterogeneous tasks are an increasingly critical requirement RP's and EnTK modular design and implementation enable us to balance generality and performance while promoting integration with application tools and system software. As such, we consider RP and its software ecosystem to be well-equipped to support these and future workloads on a wide variety of HPC platforms.

Chapter 5

Conclusion

Climate change has dramatic effects on the polar regions, causing the retreat of sea ice and mountain glaciers, mass loss from the Greenland and Antarctic ice sheets. A better and deeper understanding of these changes over time requires the processing of large volumes of satellite and aerial data. Efficient analysis of increasingly large volumes of data requires Petabytes of dedicated storage and millions of core hours on high-performance computing (HPC) infrastructures. Large data volumes help provide greater scientific insight, they necessitate scalable image processing algorithms. Specifically, this thesis makes the following contributions: (i) Compare and characterize the performance between CPU-SIFT and CUDA-SIFT, and implement GPU-SIFT as an improvement to serve the image geolocating use case; (ii) The design, implementation and concurrent execution of scalable image geolocating workflows as a set of pipelines to enable concurrent execution for 200 images with near linear scalability on 224/8 CPUs/GPUs.

In Chapter 3, we described three implementations of the SIFT algorithm — CPU-SIFT, CUDA-SIFT, and GPU-SIFT, and characterized their performance, showing the trade-offs among the implementations in terms of throughput, memory consumption and accuracy. We implemented the GPU-SIFT using CUDA MPS technology to run 2 kernels on the same GPU, validating our work against data published by Nvidia [**?**]. Furthermore, we presented an adaptive contrast enhancement implementation that increased the level of accuracy of GPU-SIFT from 56.81% of the original CUDA-SIFT to 74.81% of the improved implementation.

In Chapter 4, we described how we designed and implemented an image geolocating workflow using RADICAL Ensemble-toolkit (EnTK), supporting two image matching

kernels based on the GPU and CPU implementation of SIFT. Furthermore, we characterized the scalability performance of image geolocating pipelines in terms of weak and strong scaling. We showed near-ideal weak and strong scaling behavior using a GPU image geolocating pipeline and near-linear behavior for both strong and weak slicing for CPU image geolocating pipelines.

We attribute the majority of performance overheads for both CPU and GPU geolocating pipelines to the task launching delay arising from the use of RADICAL-Pilot. We measured how this overhead increases as a function of the number of tasks simultaneously submitted for execution. Further, we noticed that RADICAL-Pilot has certain performance limitations when scheduling multiple GPU-tasks on the same GPU device. RADICAL-Pilot deals with every GPU set (two GPUs per node) as a single unit (one GPU), no matter how many physical GPU devices exist in the compute node. This limited the number of GPU tasks that can be submitted on GPU devices.

The development of image geolocation use case has produced a computational platform to study climate changes and their impacts on the Antarctic and Greenland on production-grade HPC resources and at scale. Further, our pipeline is independent from the number of images of the analyzed dataset and can scale the number of resources utilized without further modifications to the code used to analyze each image. Thus, our image geolocating pipeline approach contributes to solving the continuous challenge of analyzing and understanding terabytes of satellite imagery, within 1 hour and using a dataset with 200, 1GB satellite images.

The image geolocating workflow can geolocate a dataset that covers most of Antarctica ~200,000 images in ~5 hours instead of 2 days on a CPU and GPU devices using 224 cores of CPUs and 4 GPUs running in parallel.

Our developments can be used to solve geoscience problems such as counting the number of seals or penguins in Antarctica using a similar approach. In conclusion, our development and approach are general and can be used to run bigger and different workflows on HPC platforms.

Future work will include concurrently using a greater number of GPU, possibly across multiple clusters. Further, we could support different GPU architectures by using CUDA

MPS technology, analyzing a bigger and larger dataset on different geographical areas. Finally, we could explore different image matching algorithms such as ASIFT and SURF to achieve better accuracy than with SIFT.

References

[1] Matteo Turilli, Mark Santcroos, and Shantenu Jha. A comprehensive perspective on pilot-job systems. *ACM Comput. Surv.*, 51(2):43:1–43:32, April 2018.

[2] V. Balasubramanian, A. Treikalis, O. Weidner, and S. Jha. Ensemble toolkit: Scalable and flexible execution of ensembles of tasks. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 458–463, Aug 2016.

[3] Imagery Cyberinfrastructure and Extensible Building-Blocks to Enhance Research in the Geosciences. https://github.com/iceberg-project/ICEBERG-middleware.

[4] Kayvon Fatahalian, Jeremy Sugerman, and Pat Hanrahan. Understanding the efficiency of gpu algorithms for matrix-matrix multiplication. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 133–137. ACM, 2004.

[5] World View1. The Start Of a New Generation. https://directory.eoportal.org/web/eoportal/satellite-missions/v-w-x-y-z/worldview-1, January 2007.

[6] World View2. The Start Of a New Generation. https://directory.eoportal.org/web/eoportal/satellite-missions/v-w-x-y-z/worldview-2, January 2015.

[7] Envisat Sentinel-1 ERS-1, ERS-2. Synthetic Aperture Radar (SAR) Satellites, November 2015.

[8] Tobias Weyand, Ilya Kostrikov, and James Philbin. Planet - photo geolocation with convolutional neural networks. *Lecture Notes in Computer Science*, page 37–55, 2016.

[9] Nehla Ghouaiel and Sébastien Lefèvre. Coupling ground-level panoramas and aerial imagery for change detection. *Geo-spatial Information Science*, 19:222–232, 2016.

[10] Nam Vo, Nathan Jacobs, and James Hays. Revisiting im2gps in the deep learning era, 2017.

[11] Mariano Rodríguez, Julie Delon, and Jean-Michel Morel. Fast Affine Invariant Image Matching. *Image Processing On Line*, 8:251–281, 2018.

[12] Mårten Björkman, Niklas Bergström, and Danica Kragic. Detecting, segmenting and tracking unknown objects using multi-label mrf inference. *Comput. Vis. Image Underst.*, 118:111–127, January 2014.

[13] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004.

[14] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten Von Eicken. Logp: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 1993*, Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP, pages 1–12. Association for Computing Machinery,, 8 1993.

[15] Four dimensional geolocation using historic, aerial and satellite imagery. https://github.com/iceberg-project/ASIFT/tree/devel.

[16] Wikimedia Wikipedia. Image analysis Wikimedia Foundation. https://en.wikipedia.org/wiki/Image$_a nalysis, March$2019.

[17] A. Treikalis, A. Merzky, H. Chen, T. S. Lee, D. M. York, and S. Jha. Repex: A flexible framework for scalable replica exchange molecular dynamics simulations. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 628–637, Aug 2016.

[18] Guy E Blelloch and Bruce M Maggs. A brief overview of parallel algorithms, 1994.

[19] Mustafa Teke and Alptekin Temizel. Multi-spectral satellite image registration using scale-restricted surf. In *2010 20th International Conference on Pattern Recognition*, pages 2310–2313. IEEE, 2010.

[20] Sah Priyanka. Improving GPU utilization with Multi-Process Services (MPS). http://on-demand.gputechconf.com/gtc/2015/presentation/S5584-Priyanka-Sah.pdf.