A Project Report on

"PARALLELIZATION OF ALGORITHMS USING DISTRIBUTED MEMORY ARCHITECTURE"

Submitted in partial fulfillment of the requirement for
Degree in Bachelor of Engineering (Information Technology)

By

Ananya Satoskar (501348)
Vishnu Nambiar (501337)
Aishwarya Mohan (501301)
Christina Rainy (501307)

Guided by:

Prof. Lakshmi Gadhikar

Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Sector 9A, Vashi, Navi Mumbai – 400703
University of Mumbai

2016-2017

CERTIFICATE

This is to certify that the project entitled

PARALLELIZATION OF ALGORITHMS USING DISTRIBUTED MEMORY ARCHITECTURE

Submitted By

Ananya Satoskar	501348
Vishnu Nambiar	501337
Aishwarya Mohan	501301
Christina Rainy	501307

In partial fulfillment of degree of **B.E. in Information Technology** for termwork of the project is approved.

External Examiner

Internal Examiner

Internal Guide

Head of Department

Principal

Date:

College Seal

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Ananya Satoskar (501348)

Vishnu Nambiar (501337)

Aishwarya Mohan (501301)

Christina Rainy (501307)

Date:

Abstract

The amount of data generated by the growing use of technology has been steadily increasing. Most current applications require the processing of large quantities of information quickly. The efficiency of this process can be improved by concurrently performing the computations on multiple processors instead of sequentially on one so that results can be obtained in a reasonable amount of time.

The proposed system aims to demonstrate this by implementing image processing algorithms to combine smaller images into composite images of a higher resolution. There will be four main stages in this process of image stitching, namely, detecting points of interest in multiple images and extracting unique features from them, matching the features and applying the desired transformations to the images to combine them. However, the algorithms required for this process tend to be compute intensive and need to process a very large amount of data quickly. Our system aims to improve the performance of the algorithms used in the stages of image stitching by using the power of parallel processing with distributed memory architecture. The proposed system will be implemented using the OpenCV libraries in the Python programming language to process the image data. MPI (Message Passing Interface) will be used to implement distributed memory architecture with multiple independent processors.

The proposed system will allow images having high pixel density to be produced at a lower cost. This can be used in multiple applications that require imaging, including the viewing of supercomputer simulations, virtual microscopy in pathology, and specifically in the composition of complex satellite images.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Nowadays, the amount of data generated by technology is increasing to a great extent. Most current applications require the processing of large quantities of information quickly. A better performance can be achieved by applications by concurrently performing the computations on multiple processors. Parallel algorithms on individual devices have become more common since the early 2000s because of substantial improvements in multiprocessing systems and the rise of multi-core processors. An effort to implement these is especially required in domains that require imaging, especially the composition of complex satellite images. Everyday millions of satellite images are captured need to be monitored to produce practicable results at a faster rate. The proposed system aims to demonstrate this by implementing image processing algorithms to combine smaller images into composite images of a higher resolution.

1.2 Motivation

India stands to lose up to 1.19 percent of its GDP as a side-effect of unchecked climate change by 2050 according to a study by the Council on Energy, Environment and Water [l]. It loses USD 9.8 billion every year due to how vulnerable the country is to earthquakes and cyclones [2]. Worldwide economic losses due to natural disasters exceeded USD 630 billion in the last decade [3]. Monitoring these risks efficiently so that action can be taken in a timely manner is of utmost importance. There is a need of a system that can regularly track climatic patterns so that natural disasters can be predicted and mitigated effectively. Detailed satellite images can be used for this purpose.

The efficient use of satellite images for practical purposes depends upon the ability to combine them for the purposes of analysis, but assembling large images usually requires costly hardware with a large shared memory. Image processing is a domain that requires a lot of computational power since there is a lot of data to be processed. Image processing algorithms involved in the field of computer vision require a high amount of accuracy which makes them more computationally intensive. Parallelizing these algorithms can give substantial improvements in their performance while providing the same level of accuracy.

More efficient creation of composite satellite images will allow for their use in applications at a lower cost. Images taken by a satellite camera can be combined to produce a complex larger image that will have a better resolution and will give an overview of the area to be monitored.

This will allow us to predict and mitigate natural crises more effectively and can be used natural disaster prediction systems. Satellite images can be used to monitor hazards such as volcanic eruptions, wildfires, floods and landslides even after they occur so that they can be managed effectively. The proposed system can also be used in fields such as biodiversity conservation. oceanography and forestry in order to regularly monitor the effects of climate change such as desertification, deforestation and urbanization.

1.3 Problem Definition

The proposed system aims to parallelize algorithms required for the process of stitching satellite images, viz. algorithms for feature detection, feature description, and feature matching in images and finally, for image transformation. The proposed system aims to use distributed memory architecture in order to allow for the execution of compute-intensive algorithms at an improved speed without compromising their accuracy.

An overview of the proposed system is shown in Figure l. l –

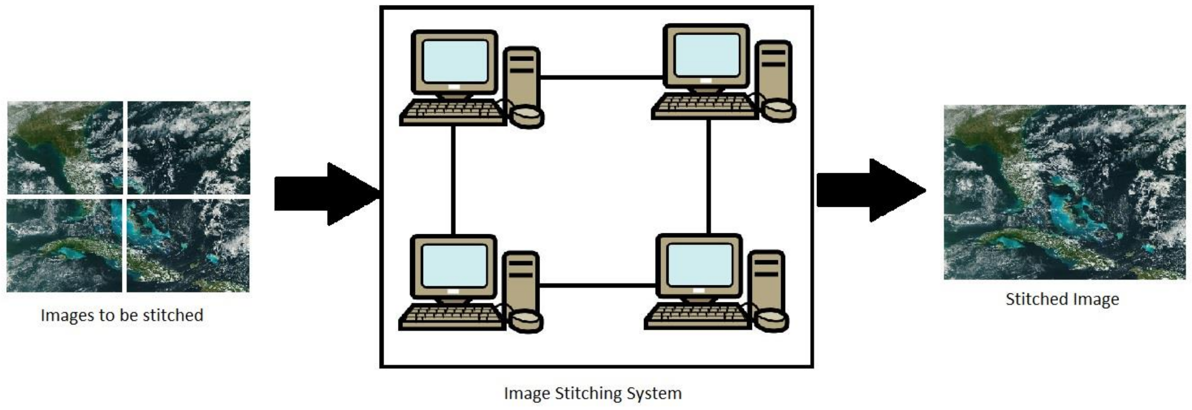


Figure l.1: Overview of the proposed system

The cameras present in a satellite capture parts of a particular geographical area. These images are gathered and sent to the proposed system, where they are combined into a single image of higher resolution to provide a wider view of the area.

1.4 Scope and Assumptions

The objective of the project is to develop a system to improve the performance of the algorithms used to implement image stitching so that images having high pixel density can be produced at a lower cost.

Although the proposed system can be used for improving the performance of existing algorithms, this performance improvement may vary with number of nodes connected. So for the purpose of this implementation, the scope has been limited to four nodes. With changes, the number of nodes can be extended.

The proposed system requires four independent nodes, each with their own primary memory. It uses the open source MPI library to handle the parallel processing while the actual operations in the algorithms are implemented in the Python programming language.

The input to the system will be a number of satellite images of a particular geographical area in the JPEG [4] format. The output of the system will be a composite image made by combining the input images as seamlessly as possible.

Features of the proposed system are -

1. Reduces the overall time required for stitching of images.

2. Cost effective in terms of the hardware used.

3. Can be used to monitor climatic changes in an area continuously so that disasters can be predicted.

4. Can be used to monitor the effects of natural disasters after they happen to provide efficient disaster management solutions.

5. Allows the overall monitoring of land use patterns and changes in natural habitats that can be used to study the effects of climate change.

1.5 Issues and Limitations

The limitations of the proposed system are as follows:-

1. Improvement in performance will vary with the number of nodes used. The scope of this project is limited to observing this improvement on four nodes.
2. Algorithms for this system have been selected to work well with the specific input images, i.e. aerial imagery taken from satellites. The same accuracy may not be maintained for other types of input images.
3. The proposed system will also require the images to be in the JPEG image format and cannot handle a variety of formats as input.

Chapter 2

Literature Survey and Analysis

2.1 Literature Survey

The literature survey addresses the two main aspects of the proposed system - namely the choice of architecture used for the multiprocessor system and the standard used for its implementation, and the algorithms parallelized. Section 2.1.1 elaborates upon the different types of parallel systems and their architectures and the Message Passing Interface specification to be used in the proposed system. Section 2.1.2 deals with the comparison of various algorithms for the implementation of the image stitching process in five phases.

2.1.1 Parallel System Architectures and the MPI Specification

Flynn's taxonomy [5] classifies computer systems according to the number of concurrent instruction streams and data streams as follows:-

1. Single instruction stream, single data streams (SISD) - Uniprocessing systems, in which a single instruction processes data stored in a single memory, fall into this category.

2. Single instruction stream, multiple data streams (SIMD) - Multiple processing elements, each with their own data memory, execute the same instruction.

3. Multiple instructions streams, single data stream (MISD) - Multiple processing elements execute different instructions on the same sequence of data.

4. Multiple instructions streams, multiple data stream (MIMD) - A set of processors simultaneously perform different instructions on different sets of data.

The proposed system is designed to be used with MIMD systems.

MIMD systems consist of multiple general purpose processors, each fully capable of performing all the necessary instructions required. MIMD organization can be further categorized upon the basis of how the processors communicate with each other in the following manner-

1. Shared Memory Organization (Tightly Coupled Organization) - Multiple processors work in coordination by accessing the same shared memory.

2. Distributed Memory Organization (Loosely Coupled Organization) - Multiple processors work in coordination by passing messages to each other while having access to their own independent primary memories.

1. Shared Memory Organization (Tightly Coupled Organization)

All the processors in this architecture share a common primary memory and have access to the program and data stored in it. Shared memory architectures may use different types of memory access such as uniform memory access (UMA), non-uniform memory access (NUMA) or cache-only memory access (COMA).

The most commonly used systems implementing Shared Memory Organization are known as Symmetric Multiprocessors (SMP).

Stallings [6] defines SMP systems as standalone computer systems having two or more processors of a similar capability which share the same memory and 1/0 facilities and are interconnected, such that the memory access time is the same for each processor.

There is a high level of coordination among the individual processors. The architecture of a symmetric multiprocessor system is shown in Figure 2.1.

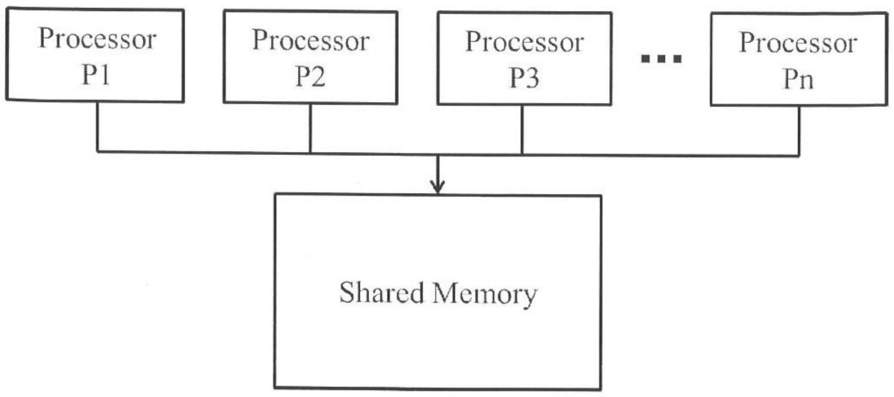


Figure 2.1: Architecture of Symmetric Multiprocessor System

The main issue with SMPs is limited amount of Random Access Memory. Most modern day Processors are much faster than their main memories and tend to find themselves stalled while waiting for memory access. The size of operating systems and other applications is usually far too big to accommodate in the cache memory alone and multiple processors may undergo starvation while waiting for access to data.

Non-Uniform Memory Access (NUMA) [6] tries to solve this issue by adding an intermediate local memory bank for each processor where data is stored so that it need not use the data bus for all data accesses. However, this requires NUMA to move data between memory banks if multiple processors require the same data. Thus, the speed increase is dependent on the nature of the processes.

Another approach to address this issue is to increase the number of memory channels by using specialized hardware; it is known as multi-channel memory architecture and has been implemented in a few high-end modern processor lines such as Intel i7 Extreme [7], Xeon [8] and AMD Socket [9].

Advantages of Symmetric Multiprocessors:

1. SMPs provide better performance than a single processor [6].
2. They provide transparency i.e. the existence of multiple processors is abstracted so that the user need not deal with scheduling and access related complications [6].
3. They are easier to configure as they only require a specialized scheduler function [6].
4. The technology used in SMPs is well-established and stable.

Disadvantages of Symmetric Multiprocessors:

1. Processors require fast access to the same memory, which may lead to access time degradation due to contention [10]. Solutions to this problem usually require modifications to the processor hardware.
2. The price/performance ratio of SMPs does not scale as well [11] as it does in Distributed Memory Architecture mentioned below.

2. Distributed Memory Organization (Loosely Coupled Organization)

Clusters are a type of system that implements Distributed Memory Organization in which multiple independent computers are interconnected by a network and work together as a single source of computational power [6]. Each computer in a cluster is referred to as a node and it consists of an independent processor that has its own primary memory. Each computer is connected to the rest of the cluster by a high-speed link or switch hardware. Processors must communicate over a network using message-passing to coordinate cluster activity.

Nodes in a cluster access not only their own disks, but commonly also a shared disk subsystem. Clusters use a middleware that can provide transparency to the user and present the distributed system as a single unified system to the user.

The architecture of a symmetric multiprocessor system is shown in Figure 2.2.

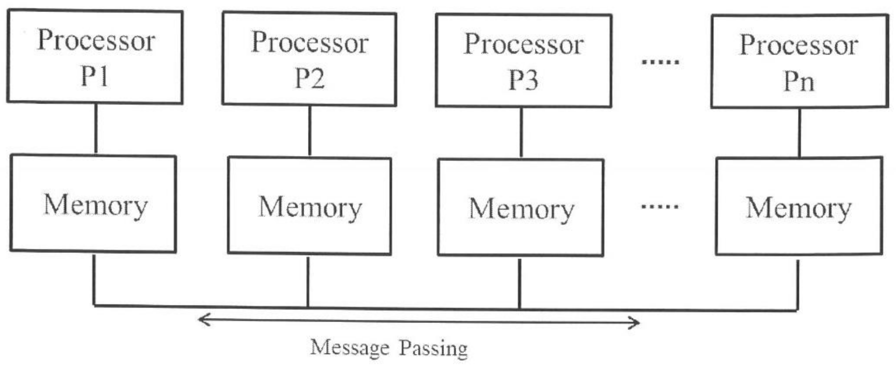


Figure 2.2: Architecture of Cluster

According to Brewer [12], clustering satisfies the following design requirements of high performance computing - absolute and incremental scalability that make it possible to create extremely large clusters that can be much more powerful than single machines, and higher fault tolerance in the case of failure so that better availability is provided.

Advantages of Clusters:

1. Highly scalable [11].
2. Provide better fault-tolerance [12].

3. The larger amount of primary memory available to the processors allows more redundancy that can reduce access time.

Disadvantages of Clusters:

1. Network latency may reduce performance overheads if a large amount of message passing is required. When applications are being parallelized, care must be taken by the programmer to decide upon the best approach so that the power of the cluster can be utilized well.
3. They are not as easily configured as SMPs as they require a middleware that can handle issues related to load-balancing and failure.

Choice of Distributed Memory Architecture

The proposed system aims to provide a robust and effective method for the combining of satellite images. Distributed Memory Architecture allows for greater redundancy of data which provides better fault tolerance in case of failure. Since each processor has its own primary memory, it allows for better access times as compared to SMPs. It is also easier to scale and therefore has better scope for future applications that may require a very large amount of computational power.

Message Passing Interface

MPI is a message-passing library interface specification developed by the Message Passing Interface Forum [13], which is used for exchanging information amongst coordinating processors in Distributed Memory Architecture using message passing. MPI is not a language or an implementation; it is an open standard that specifies all operations as functions, subroutines or methods according to the language it is used in.

MPI is suitable for general MIMD programs and works on top of protocols used for standard Unix interprocess communication, which makes it suitable for clusters and heterogeneous multiprocessing systems. MPI manages issues related to data consistency, load-balancing and failure for the distributed system.

According to a report by the Message Passing Interface Forum [13], the MPI standard addresses point-to-point communication, data types, collective operations, process groups, communication contexts, process topologies, environmental management and inquiry, Info objects, process creation and management, one-sided communication, external interfaces, parallel I/0, language bindings for multiples languages like FORTRAN and C, and tool support.

An MPI program consists of independent processes, each executing its own code, which communicate with each other by calling MPI communication primitives. MPI has support for both synchronous and asynchronous communication.

MPI manages system memory that is used for buffering messages and for storing internal representations of various MPI objects such as groups, communicators, data types, etc. This memory is not directly accessible to the user; thus, objects stored in it are opaque to the user. Opaque objects are accessed using handles which are stored in the user space and can be manipulated by passing handle arguments to functions. Opaque objects are only available to the process in which they are created and cannot be transferred to other processes. Arrays of opaque objects are accessed using arrays of handles.

MPI also allows the definition of argument data types that represent states, named constants, choice, absolute addresses, file offsets and counts.

There are multiple implementations of MPI such as Open MPI [14] and MPICH [15]. The proposed system will use an open source implementation of MPI known as MPICH [15] developed by the Argonne National Laboratory, as it has support for MPI 3.

2.1.2 Algorithms Used

The common definitions used in this section of the literature survey are:-

Interest points are defined as distinctive locations in images [16]. Moravec [17] defines them as small patches scattered more or less uniformly over images and having the property that the corresponding points are likely unambiguously findable in subsequent images.

Schmid, et al [18] define interest points as any point in the image for which the signal changes two-dimensionally. They compare multiple feature detection algorithms based on two main aspects - repeatability and information content.

Repeatability compares how stable the algorithm is while detecting the same features in different images under different viewing conditions. Interest points are 'repeated' if the same real-life 3D point is detected accurately in two different images. Information is defined as a measure of distinctiveness of an interest point. [18]

The neighbourhood of every interest point is represented by a feature vector, called a descriptor. Descriptors represent the local shape of the image at an interest point. Information content of a descriptor can be measured by its entropy.

The following literature survey details algorithms for each of the four main steps of the proposed system-

1. Feature Detection and Description
2. Feature Matching
3. Estimating Homography Matrix
4. Transforming and Compositing Images

i. FEATURE DETECTION AND DESCRIPTION

Feature detection is the process of finding interest points in an image [17]. In this step, we use algorithms that identify the interest points in each image, and then extract and store their descriptors for future processing. Feature detection algorithms must be robust to scale, rotation and affine transformations.

Feature description algorithms should be able to describe the extracted features in a way that reduces both the time and space complexity of the algorithms required in the next step while maintaining good accuracy. Since the proposed system aims to parallelize these algorithms, our choice of algorithm has been made based on the accuracy of the algorithm (as measured using the concepts of repeatability and information defined above) rather than their complexity.

(a) Algorithms that are used widely for the purposes of interest point detection (as seen in popular image processing libraries [19]) are:

i. Features from Accelerated Segment Test (FAST)
ii. Laplacian of Gaussians (LoG)
iii. Difference of Gaussians (DoG)

i. Features from Accelerated Segment Test (FAST)

FAST is a corner detection algorithm proposed by Edward Rosten and Tom Drummond that compares the intensity of pixels surrounding a particular pixel to detect interest points based on a threshold value T of variance from the point we are considering. Steps:

1. We arbitrarily select a pixel in the image and call it p. Let its intensity be Ip.
2. Select a threshold value T which will be used for the intensity comparisons.
3. After selecting a circle of 16 pixels around the pixel p, the algorithm requires 'N' contiguous pixels out of these 16 to be either above or below Ip by the threshold value Tor more. N has been taken to be 12 by Rosten, et al [20].
4. To make the algorithm faster, the intensity of pixels 1,5,9 and 13 is compared with Ip. For p to be an interest point, at least three of the four selected pixels should satisfy the criterion for the threshold; otherwise, p is not an interest point.
5. If the condition is satisfied, the remaining twelve pixels are checked.
6. Every pixel is checked in this way to detect interest points.

Advantages of FAST:

1. FAST is extremely efficient i.e. it can process a live video in 7 percent of the time as compared to SIFT [20].

Disadvantages of FAST:

1. FAST is not robust at high levels of noise and this property is dependent on the threshold selected.
2. For N<12, the algorithm does not work well as the number of interest points detected is very high.
3. The order in which the 16 pixels are queried determines the speed of the algorithm because it changes for different distributions of features in different images.
4. Machine learning methods are required to address the above disadvantages.

5. Non-maximal suppression is required to reduce the detection of adjacent features.

ii. Laplacian of Gaussians (LoG)

The Laplacian operator is a measure of the second-order spatial derivative of an image in two dimensions [21] it indicates regions of drastic change in the intensity of the image and thus can be used to detect interest points. It is applied to an image which has first been smoothed using the Gaussian filter that reduces its sensitivity to noise.

In order to convolute the image, we use a convolution kernel that is calculated in advance by convolving the Gaussian filter kernel with the Laplacian kernel, and then apply it to the image.

iii. Difference of Gaussians (DoG)

Blobs are smooth regions in the zero-order scale space representation of an image that are brighter or darker than the background and stand out from their surroundings. A blob should be associated with at least one local extremum [22].

Difference of Gaussians is a method of a feature detection algorithm which performs subtraction of one blurred image version of the original image with a less blurred version of the same image.

Blurring an image uses the Gaussian filter as a band pass filter to filter out the higher frequencies which represent noise in an image; it also filters out the low frequencies which represent the similar pixels or homogeneous pixels in an image. This leaves just the edges that are passed through.

DoG works mainly as a blob and corner detector by detecting image structure in the scale space representation [23] of an image.

The basic methodology of DoG to extract relevant blobs is as follows -

Blobs are extracted at all levels of scale, and their normalized scale-space volume is computed. They are then sorted in descending order of this computed volume.

For each blob, the scale at which it assumes maximum blob volume is determined and its support region at that scale is extracted.

Advantages of DoG:

1. The Difference of Gaussians approximates the Laplacian of Gaussians detector [24] and improves computational efficiency.
2. It is not affected by noise in the image (especially Gaussian noise).
3. It is free from tuning parameters. Disadvantages of DoG:

Disadvantages of DoG:

1. Not as efficient as the Determinant of Hessian (DoH) method used in the Speeded Up Robust Features [25] algorithm described below.

Table 2.1: Comparison between feature detection algorithms

Sr. No.	Algorithm	Features	Advantage	Disadvantage
1.	FAST(Features from Accelerated Segment Test)	Corner detection to detect interest points.	1.Extremely efficient.	1. Not robust at high levels of noise. 2. Machine learning methods required
2.	LoG(Laplacian Of Gaussian)	Indicates regions of drastic change in the intensity of the image.	1. Detects features accurately	1. Not as efficient as DoG.
3.	DoG(Difference Of Gaussian)	Performs subtraction of one blured image version of the original image with the less blured version of the same image	1. Improves computational efficiency. 2. Free from tuning parameters.	1. Not as efficient as Determinant of Hessians.

4.	DoH(Determinant Of Hessian)	Applies the Hessian function to perform a second partial derivative test.	1. Better efficiency.	1. It is compute intensive

(b) Algorithms that are used widely for the purposes of interest point description (as seen in popular image processing libraries):

i. Binary Robust Independent Elementary Features (BRIEF)
ii. Scale Invariant Feature Transform (SIFT)
iii. Speeded-Up Robust Features (SURF)

i. Binary Robust Independent Elementary Features (BRIEF)

The BRIEF descriptor [26] relies on a relatively small number of intensity difference tests to represent an image patch as a binary string. These binary strings are used to match features using Hamming distance. It provides a shortcut way to find the binary strings without finding descriptors.

BRIEF takes a smoothened image patch and selects a set of n_d(x,y)-location pairs defines a set of binary tests.

Pixel intensity comparisons are done on these location pairs as follows -

If the location pairs are (p,x) and (p,y), pixel intensity will be I(p,x) and I(p,y). If $I(p,x) < I(p,y)$ then the result will be 1, otherwise it is taken to be 0.

This is applied for all the n_d location pairs so that we get an n_d-dimensional bit string which can be matched using Hamming distance.

Advantages of BRIEF:

1. It provides better speed-up because finding the Hamming distance simply consists of applying XOR and bit count operations.

2. BRIEF is fast both to build and to match. [26]

Disadvantages of BRIEF:

1. BRIEF performs poorly with rotation.
2. BRIEF is a feature descriptor; it does not provide a method to detect features unlike SIFT or SURF, which integrate the feature detector and the descriptor.

ii. Scale Invariant Feature Transform (SIFT)

This approach by Lowe is an algorithm which transforms image data into scale invariant coordinates based on local features [16]. It detects and describes features in an image that are invariant to scaling and rotation in such a way that they are highly distinctive. Thus, the features can be matched against a database of features to provide some method of object recognition.

There are four major stages of SIFT, namely: -

1. Scale-space extrema detection

This stage implements a Difference of Gaussians function to identify potential interest points.

2. Keypoint localization

At each potential interest point, a model is fit to find out the location and scale and the keypoint is selected based on its stability.

3. Orientation assignment

Orientations are assigned to each selected keypoint based on the direction of the image gradients that are local to that keypoint.

4. Keypoint descriptor

The image gradients are measured at a particular scale in the area around each keypoint and transformed into the desired representation of a high-dimensional vector.

Advantages of SIFT:

1. Keypoint descriptors stored using this algorithm are highly distinctive, as the image gradients within the local range of an image are all considered. Therefore they provide more accuracy during the image matching process.
2. It generates a large number of features, thus making it a more robust algorithm as it increases the robustness of extracting objects from cluttered images.
3. Keypoints are detected over a complete range of scales.

Disadvantages of SIFT:

1. It requires a large number of dimensions for each feature vector i.e. 128, which affects the efficiency of the feature matching process. Thus, it can be improved using parallelization techniques.
2. However, it is less accurate than the SURF [25] algorithm described below.

iii. Speeded-Up Robust Features (SURF)

SURF [25] is a feature detector and descriptor based on the Hessian matrix, which relies on integral images to reduce the computation time. The second derivatives used in the Hessian matrix give strong responses to blobs and ridges and give results that are similar to results obtained using a Laplacian operator.

It uses only 64 dimensions to reduce the time taken for feature computation and matching but can also use 128 dimensions for increased accuracy.

It consists of the following stages -

We first fix a reproducible orientation based on a circular region around the interest point using Haar wavelet responses and represent these responses as vectors.

The dominant orientation is estimated by calculating the sum of all responses using a window function, both horizontally and vertically to yield a new vector.

To extract a descriptor, a square region is constructed that is centered on the interest point and oriented along the previously calculated vector.

The region is split into smaller 4 x 4 regions and simple features are calculated at regularly spaced sample points in each sub-region.

Wavelet responses are calculated and summed up for each smaller area in horizontal and vertical directions to form entries for the feature vector.

Advantages of SURF:

1. SURF has been shown [25] to outperform other descriptors with more than 10 percent improvement in recall for the same precision, i.e. it is very accurate.

Disadvantages of SURF:

1. Certain variations of the SURF algorithm can be computationally intensive, such as the SURF-128 that uses 128 dimensions to describe features.

Table 2.2: Comparison between feature description algorithms

Sr. No.	Algorithm	Features	Advantages	Disadvantages
1.	BRIEF(Binary Robust Independent Elementary Features)	1. Takes the smoothened image patches, selects a set of location pairs and defines a set of binary tests.	1. Provides better Speed up 2. It is Fast both to build and to match.	1. Performs poorly with rotation. 2. Does not provide a method to detect features.
2.	SIFT(Scale Invariant Feature Transform)	1. Detects and describes features in an image based on the scale space extrema of the image using	1. Generates large number of features thus making it a more robust algorithm. 2. Keypoints are	1. Requires large number of dimensions for each feature vector i.e 128 which effects the efficiency of the

		DoG.	detected over a complete range of scales. 3. Keypoints stored using this algorithm, are highly distinctive.	feature matching process. 2. Less accurate.
3.	SURF(Speed Up Robust Features)	Feature detector and descriptor based on Hessian matrix.	1. Better accuracy. 2. Scale and rotation invariant.	1. Certain variations of SURF are compute intensive.

Choice of SURF over alternative algorithms

The SURF algorithm has been shown [25] to exhibit better accuracy when compared to other descriptors. It is also scale and rotation invariant. However, the SURF does not have as good a performance as other algorithms such as BRIEF [26]. Thus, we have chosen to parallelize this algorithm.

2. FEATURE MATCHING

The second step of the proposed system, i.e. the feature matching stage, compares the features extracted and described in the first step to find features that are similar with a predefined level of error that is acceptable to the user. The matching algorithms use the nearest neighbour approximation approach that calculates the distance between data objects (in this case, features) to find the 'nearest' or most similar data objects in one image to an object in the other. Algorithms used for feature matching are -

(a) Hierarchical K-Means Tree Algorithm

(b) Randomized kd-Trees Algorithm

(a) Hierarchical K-Means Tree Algorithm

The hierarchical k-means tree is created by dividing the data into k non-overlapping clusters and assigning each cluster to a branch of the tree.

The same method is then applied to each individual cluster recursively to create a tree with k as the branching factor.

The recursion is stopped when the number of points in a cluster is smaller than k.

A best-bin-first algorithm explores the tree by initially traversing it once and adding all unexplored nodes to a priority queue.

It extracts the branch that has the closest centroid to the data point being matched and then restarts tree-traversal from that branch.

(b) Randomized kd-Tree Algorithm

The kd-tree algorithm proposed by Friedman, et al [27] constructs a decision tree by splitting the data in half at each level based on the dimension of the data that has the most variance. The data is split at the median. The kd-tree algorithm has been shown to be less efficient when the number of dimensions in the data increases because backtracking takes longer.

The randomized kd-tree algorithm [28) creates multiple kd-trees by randomly choosing the splitting dimension from the first n dimensions of the data that exhibit the most variance. They impose a limit of p nodes to be searched, and break the search into simultaneous searches along the multiple trees. Thus, if number of kd-trees is q, an average of p/q nodes will be searched in each tree.

While multiple trees are searched at the same time, a single priority queue is used to store the nodes in ascending order of their distance from the bin boundary. Thus, all nodes are not only ranked with respect to their own tree, but also with respect to other trees. Nodes from all the trees are searched simultaneously according to their distance from the node that must be matched.

Table 2.3: Comparison between feature matching algorithms

Sr. No.	Algorithm	Features	Advantage	Disadvantage
1.	Hierarchical k-means	Constructs a tree with branching factor k where each branch is assigned a cluster.	1. Does not consume as much memory as randomized kd-tree.	1. Not easily parallelized
2.	Randomized kd-tree	Constructs multiple decision trees by splitting the data in half at each level that has the most variance.	1. Multiple trees searched simultaneously so it is efficient.	1. Requires a lot of memory to store the trees.

Choice of Randomized kd-Tree Algorithm over Hierarchical K-Means Tree Algorithm

According to Lowe [29], the choice of nearest neighbour approximation algorithm can be made based on the cost. This is computed using weighted variables to give the desired importance to parameters such as build-time of tree, memory required by tree, etc. according to what the user desires.

In our case, we do not give importance to lowering memory consumption while storing the tree, since each processor has its own primary memory in distributed memory architecture. The algorithm that is shown to perform well when build-time is a major concern is the randomized kd-tree algorithm [29]. Thus, we choose this algorithm for the proposed system.

3. ESTIMATING HOMOGRAPHY USING RANDOM SAMPLING

Once matching features have been found in the previous step, we need to map the interest points in one image to another so that the appropriate transformation can be applied. This stage of the proposed system estimates the transformation that maps an image onto a matching image. Projective-mapping between any two projective planes with the same centre of projection is known as homography [30] and it can be represented as a 3x3 matrix in terms of homogenous coordinates.

In order to find the homography, sampling consensus methods are used to fit a mathematical model to the image data. We have chosen to use the Random Sampling Consensus (RANSAC) algorithm [31] by Fischler and Bolles.

RANSAC estimates parameters of a model that fits the data while classifying the data into inliers and outliers for that model. It uses the hypothesize-and-verify framework. An initial subset of data points is selected randomly and used to construct a model. The data is then classified into inliers and outliers according to this model and support for this model is calculated. This is repeated until the possibility of finding another model with greater support than the current best model is found to be extremely low.

RANSAC can tolerate a large fraction of outliers [32] and is thus appropriate for our task because of its robustness. RANSAC is popularly used because almost no prior assumptions are made about the data for it to successfully work [33].

4. APPLYING THE TRANSFORMATION AND COMPOSITING

The fourth stage of the proposed system deals with the creation of the final composite image. The homography matrix computed in the previous step is used to map one image onto another based on their matching features. Thus, for multiple images having matching features, we combine them into a single image using the following steps:-

(a) We select a compositing surface on which to combine the image: The proposed system will use a flat compositing surface to combine its satellite imagery, since we do not require a 360 degree panorama or a spherical 3-D result. The choice of compositing surface affects the manner in which a reference image is chosen in the next step.

(b) We select an initial reference image which is used to combine the remaining images: For a flat compositing surface, we can choose any one of the images to be combined as the reference image.

(c) We select which the other images will be added to the compositing surface for the final image.

(d) The homography estimated by the RANSAC algorithm is applied to the other image.

(e) The transformed image is blended with the reference image using a bitwise OR operation for each of the pixels.

2.2 Existing System

2.2.1 Stitching giga pixel images using parallel computing

This system, proposed by Kooper et al [34], addresses the problem of stitching aerial imagery acquired during flights over Costa Rica by the utilization of multiple hardware architectures. It performs intensity-based stitching of images in groups using a pyramid data structure for faster image access and retrieval.

It uses the SURF algorithm to detect and describe features, and matches them using Euclidean distance and the Delauney triangulation algorithm. Then, transformation parameters are estimated using a Least Square Fit method and the images are transformed and composited using bi-cubic interpolation.

When the creation of one quarter of a 4 GigaPixel image was benchmarked, it took 63 hours on a single processor, and the full-resolution image was estimated to require 42 days of processing on the same hardware. The parallelized system was implemented by constructing a pyramid of image tiles that are distributed between clients for processing by a server. The final stitched image was created on a cluster with 16 compute nodes and a single head node, each having 8 cores and 16GB of primary memory and using a Server Message Block file system. A 79 GigaPixel image consisting of 600,000 tiles was computed in 3 days.

The system uses coarse georeferencing information to group images initially before they are combined using intensity-based stitching. It deals with film-based rather than digital images.

2.2.2 Real-Time Spherical Panorama Image Stitching Using OpenCL

This system [35] used photographic equipment to capture images in real-time and convert them to 360 degree panoramas by stitching them together. It uses a shared memory multi-core architecture (NVIDIA 9600GT) to process eight images captured using webcam video using Open CL library. One of the main features of this project is the complete 360 degree panorama Which is created by transforming the images using conversion to spherical coordinate system. Liao, et al develop a real-time co-ordinate system converting module to present their panorama in a three-dimensional spherical image.

Eight images taken by a low resolution webcam could be processed simultaneously to create spherical panoramas. The performance enhancement was experimentally observed to be 76 times [36] when compared to the implementation of the system on a CPU.

2.2.3 AutoStitch - Automatic image stitching software

Brown and Lowe proposed AutoStitch [37] - an automatic image stitching software which is used to stitch images in the most effective way by finding matching points in all input images with the help of the Scale Invariant Feature Transform algorithm. AutoStitch then proceeds with the image stitching process by robustly aligning all images and using advanced blending algorithms to form a composite image with a much larger field of view. However, this software is proprietary and is not implemented in parallel.

Advantages of the proposed system over existing software

The systems proposed by Kooper et al [34] and Liao [35] described above use parallel processing to improve the efficiency of the image stitching process. However, they require certain constraints to be satisfied for input images. The system proposed by Kooper et al [34] has been designed for film-based images and requires georeferencing information to group them. The system proposed by Liao et al [35] uses low resolution webcam images. The proposed system does not require georeferencing information and is capable of processing images at the resolution provided by a satellite camera in the JPEG format.

AutoStitch by Brown and Lowe [37] does not have the limitations of the above systems, but it does not implement the algorithms in parallel and is marketed as a proprietary software. The proposed system aims to parallelize similar algorithms using open source libraries.

2.3 Requirements Analysis

2.3.1 Functional requirements

The basic input given to this system is a series of complex satellite images taken by a camera which are then combined into one large picture so that we will get a full view of the area. Computations are performed concurrently on multiple processors instead of sequentially on a single processor so that images can be generated in a reasonable amount of time.

Following are the functional requirements of each of the modules of the system:

1. Detecting points of interest

This module's main function is to detect points of interest from multiple images.

Input: Overlapping satellite images in the JPEG format.

Output: Locations where features are detected will be sent to next module.

2. Describing points of interest

This module's main function is to describe the detected features from multiple images so that they can be processed by the system.

Input: Overlapping satellite images and locations of features detected in them.

Output: Features represented in the form of a vector.

3. Matching the features

This module's main function is to match the features in one image with those in others with a certain level of approximation.

Input: List of all features extracted in the previous module for each image.

Output: List of matching features.

4. Computing the homography

This module is responsible for calculating the transformation matrix required to combine the images.

Input: List of matching features from each pair of images from the previous module.

Output: Homography matrices for pairs of images

5. Applying the desired transformations

This module is responsible for combining the multiple satellite images into one composite image.

Input: List of matching images and their homographies from the previous module.

Output: Multiple satellite images combined into one composite image.

2.3.2 Non-functional requirements

The proposed system should provide an improvement in performance when compared to the sequential implementation of the same algorithms. The system should minimize communication overheads between the processors in order to be as efficient as possible.

The proposed system should be built in a modular fashion so that individual modules can be further optimized in the future if the need arises. It should also facilitate reuse of modules so that commonly used operations need not be built from scratch every time they are required.

The system should handle minor variations in input images in terms of the algorithms that are chosen so that differences in the properties of the image cause minimal errors. The proposed system does not have prior knowledge of properties such as lens distortion, exposure and perspective, which depend on the equipment used to take the photograph. Thus, extremely robust algorithms must be used.

Chapter 3

System Design

3.1 Architectural Block Diagram

The architecture of the proposed system is shown in Figure 3.1.

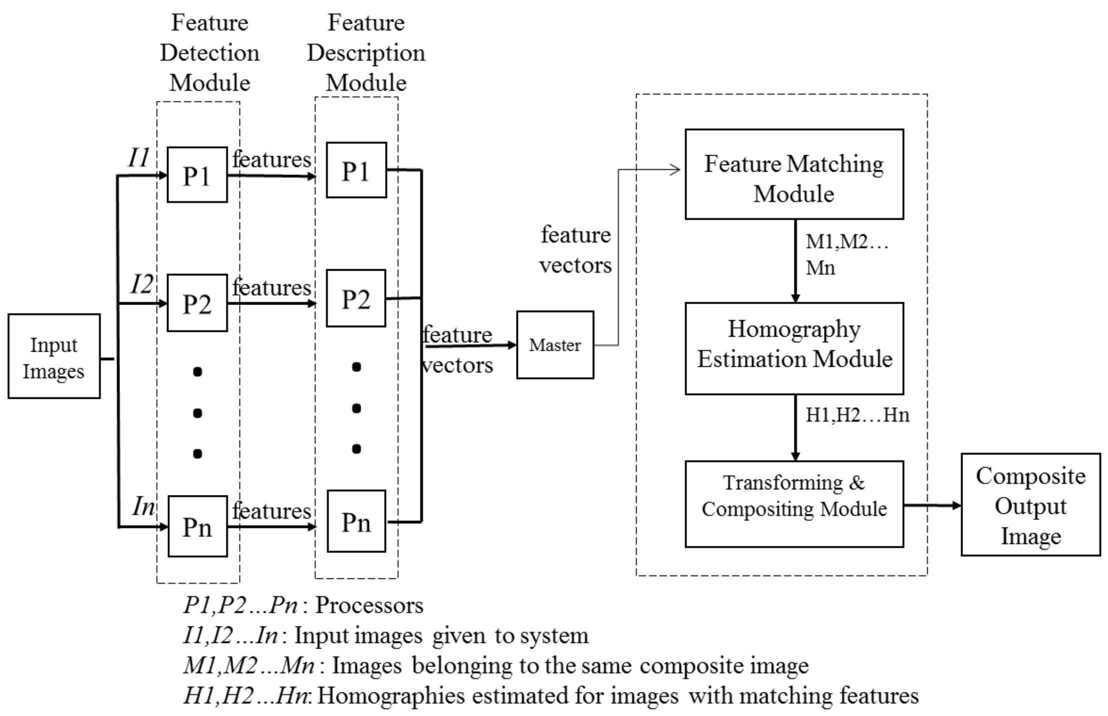

Feature Detection Module
Feature Description Module
Input Images
I1
I2
In
P1
P2
Pn
features
feature vectors
Master
Feature Matching Module
M1,M2…Mn
Homography Estimation Module
H1,H2…Hn
Transforming & Compositing Module
Composite Output Image
P1,P2...Pn : Processors
I1,I2...In : Input images given to system
M1,M2...Mn : Images belonging to the same composite image
H1,H2...Hn: Homographies estimated for images with matching features

Figure 3.1: Architecture of the proposed system

The proposed system has two major phases concerned with -

1. Parallel Feature Detection and Description
2. Sequential Stitching of Images

3.1.1 Parallel Feature Detection and Description

The following are the modules which will be implemented in parallel among the connected 4 nodes.

3.1.1.1 Feature Detection Module

This module will use the Determinant of Hessian blob-detector algorithm to find interest points in each image. The coordinates of the centre point of each feature will be sent to the next step of the system.

3.1.1.2 Feature Description Module

The feature description module will be responsible for representing the detected feature points in the form of vectors using the Speeded Up Robust Features algorithm [15].

It will extract information about the intensity gradient of the pixels around each feature point detected in the feature detection stage by calculating and summing up wavelet responses for each vector. Each descriptor vector will have 128 dimensions.

3.1.2 Sequential Stitching of Image

The following are the modules which will be implemented in sequential under the master node.

3.1.2.1 Feature Matching Module

This module will implement a nearest neighbour approximation for each feature in order to find other matching features using the randomized kd-trees algorithm [28]. It consists of two phases.

Multiple decision trees are constructed by using a randomized method to select the splitting. These trees are searched simultaneously using a common priority queue, to find the best matches for a feature. This module then decides which images are to be combined in the final output by checking the matches against a threshold value to reduce error. Details of matching images will be sent to the next module.

3.1.2.2 Homography Estimation Module

The homography matrix defines the transformation required to align one image to another. This module estimates a homography matrix using the Random Sampling Consensus algorithm that builds a model based on a random sample of data, computes inliers and then updates the model based on those inliers until the desired threshold is reached. The actual computation involves solving linear equations to obtain the elements of the homography matrix.

This will be done for every image in the set based on a single image chosen as a reference and the homography matrix is passed to the next module.

3.1.2.3 Transforming and Compositing Module

Once the homography has been estimated, this module will be used to blend the images on a flat compositing surface giving the final image as the output. The transforming and compositing module operates in two steps -

l. First, the image to be combined will be transformed by applying the estimated homography to it.

2. In the second step, these regions are blended together using a bitwise OR operation on their pixels to give the final image.

3.2 Data Flow Diagrams

3.2.1 Level O DFD of the Proposed System

Figure 3.2 shows the Level O DFD of the proposed system.

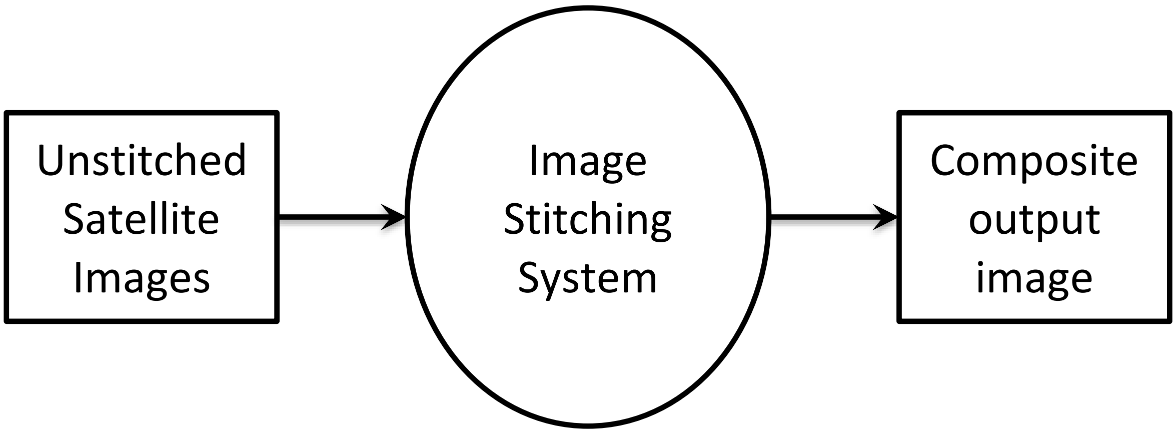


Figure 3.2: Level 0 DFD of the proposed system

This system takes multiple satellite images as input and processes them. At the end, the output is a single image made by combining multiple input images as seamlessly as possible.

3.2.2 Level 1 DFD of the Proposed System

Figure 3.3 shows the Level 1 DFD of the proposed system.

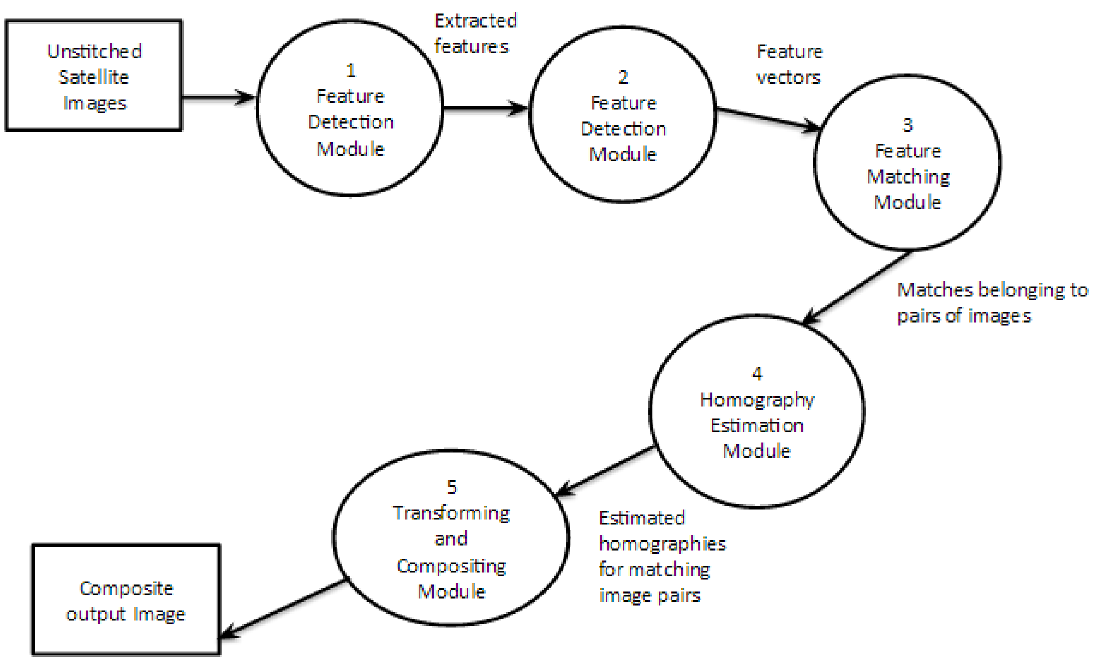

Unstitched Satellite Images
Extracted features
1 Feature Detection Module
2 Feature Detection Module
Feature vectors
3 Feature Matching Module
Matches belonging to pairs of images
4 Homography Estimation Module
Estimated homographies for matching image pairs
5 Transforming and Compositing Module
Composite output Image

Figure 3.3: Level 1 DFD of the proposed system

3.2.2.1 Feature Detection Module

This module takes multiple satellite images as input and detects features in them. The coordinates of the centre point of each feature wil1 be sent to the next step of the system.

3.2.2.2 Feature Description Module

It takes coordinates of centre points of features as input and constructs vectors to describe each feature which are given as output.

3.2.2.3 Feature Matching Module

It takes all the feature vectors as input, searches them to find matches, and returns connected Components of a graph as output based on the matches.

3.2.2.4 Homography Estimation Module

It takes connected component as input, computes homography with reference to one image and outputs this as a matrix.

3.2.2.5 Transforming and Compositing Module

Homography matrix is taken as input and the transformed and blended composite image is given as output.

3.2.3 Level 2 DFD of the Feature Matching Module

Figure 3.4 shows the Level 2 DFD of the Feature Matching Module.

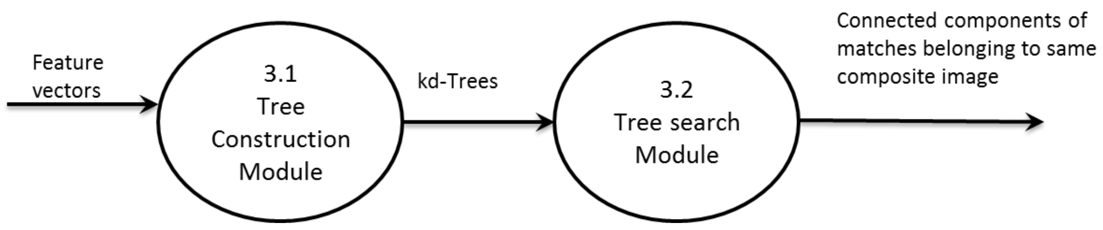

Feature vectors
3.1 Tree Construction Module
kd-Trees
3.2 Tree search Module
Connected components of matches belonging to same composite image

Figure 3.4: Level 2 DFD of the feature matching module

3.2.2.1 Tree Construction Module

This module take feature vectors as the input and constructs multiple kd-trees which it passes as output to the next module.

3.2.2.2 Tree Search Module

This module traverses the kd-trees constructed in the previous step, constructs connected components out of the matches and returns them in graph form.

3.2.4 Level 2 DFD of the Transforming and Compositing Module

Figure 3.5 shows the Level 2 DFD of the Transforming and Compositing Module.

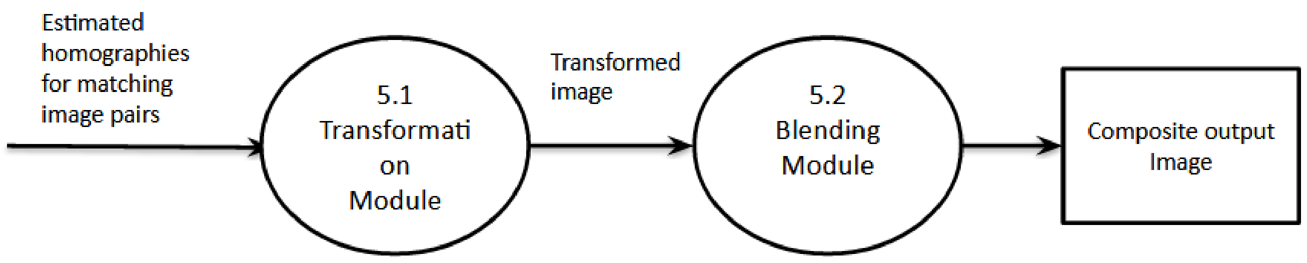

Estimated homographies for matching image pairs
5.1 Transformati on Module
Transformed image
5.2 Blending Module
Composite output Image

Figure 3.5: Level 2 DFD of the transforming and compositing module

3.2.3.1 Transformation Module

It takes the pairs of images and the homography as input, and transforms one image with respect to the other.

3.2.3.2 Blending Module

This module takes the transformed images as input, performs blending of the images and returns the composite image.

Chapter 4

Implementation Details

4.1 System Requirements (Hardware /Software)

The proposed system requires the following hardware and software to establish the connection between different nodes and implement algorithms.

Hardware Requirements

The proposed system has the following minimum hardware requirements for each node:

Processor: Intel Core i3 or equivalent with x64 architecture

Memory: 4GB RAM

Disk Space: 500GB

At least one network adapter

4 independent nodes consisting of the above configuration are required, connected by a conventional Ethernet network consisting of cables and RJ45 connectors. Each node must have adequate PCI slots, network ports, etc. for the connections.

A network switch is required to connect all the nodes having at least 100Mb/s Ethernet speed.

Software Requirements

The proposed system has the following minimum software requirements

1. Any 64-bit Linux Operating System with kernel v4.8.4. The proposed system uses Fedora 25 for the purpose of demonstration
2. Python 2.7.3 with the Numpy, Scikit-image, OpenCV and MPI4Py libraries.
3. MPICH v3.2 libraries for the MPI implementation.

The proposed system is implemented in the Fedora operating system (based on Linux kernel) as Linux operating systems are stable, powerful operating systems that runs on cheap commodity hardware and can be freely distributed [38].

Python language is used for programming of the image stitching algorithms such as feature detection, feature description, feature matching, homography estimation, Transformation and Compositing. Python supports powerful image processing capabilities.

OpenCV is a library of Python bindings designed to solve computer vision problems. OpenCV makes use of Numpy, which is a highly optimized library for numerical operations [19]. The Scikit-Image library is also an image processing library designed to work on Numpy arrays in Python.

The Message Passing Interface[13] is a message passing system designed to work on parallel computing systems. MPI manages issues related to data consistency, load-balancing and failure for the distributed system.

The MPI4Py Python library provides a Python interface to an MPI implementation. The proposed system uses the MPICH, a high performance and widely portable implementation of MPI

4.2 Implementation Details

The implementation of the proposed system consisted of the following main steps –

1. Setting up of a multimode cluster
2. Installation of packages
3. Implementation of modules

4.2.1 Setting Up a Cluster Using MPI4Py on Fedora 25

Step 1: Install the packages required for the setup on all machines.

```
sudo dnf install python-devel mpich-devel mpich-autoload python-mpi4py
```

Note: Assuming openssh and nfs-utils are installed. Else, install those packages as well.

Step 2: Configure the /etc/hosts file on every machine

1. Enter the following command on each machine to find the IP-address for your desired interface

```
ip-address
```

Note down these IP addresses as they need to be added in the /etc/hosts file along with the hostname.

2. Change your default hostname.

To change the defalt hostname, open /etc/hostname in the text editor of your choice. Here, nano is used.

```
sudo nano /etc/hostname
```

Add the following lines to the file

```
yourhostname.localdomain
```

Where yourhostname will be the name you want to give the machine. Log out and log in again for the change to take effect.

3. Configure /etc/hosts file

Open /etc/hosts in the text editor of your choice. Here, nano is used.

```
sudo nano /etc/hosts
```

Enter all the IP addresses that you noted down in Step 1, along with the hostname assigned to each machine in Step 2 as follows.

```
IP-address(tab)hostname(tab)hostname.localdomain.
```

Step 3: Set up the passwordless SSH Login

The machines should be able to log in and out of each other without requiring a password. In order for this to work, we perform the following steps -

1. Create a new user dedicated for running MPI programs. This requires you to have root privileges.

```
sudo useradd mpiuser
```

```
passwd mpiuser
```

Enter password as 'password'. These are the standard inputs we will be using on all machines for simplicity's sake.

Give this user sudo privileges

```
sudo usermod -a -G wheel mpiuser
```

2. Change to that user

```
su - mpiuser
```

3. Generate the SSH keys

```
ssh-keygeyn -t rsa
```

For easier setup, just press enter when prompted for a passphrase and file in which to save keys.

4. Now start the required daemons on all the machines

```
systemctl start sshd
```

5. Sign into the client machine using the following command

```
ssh clientmachinehostname
```

It should ask for the password of mpiuser@clientmachinehostname, enter 'password'. You should then be signed into mpiuser@clientmachinehostname. Type exit to exit the session.

6. Copy the keys onto other machine

```
ssh-copy-id clientmachinehostname
```

7. Sign into the client machine using the following command

```
ssh clientmachinehostname
```

It should not ask for the password.

Step 4: Set up NFS with master as the server and remaining slave computers as clients

1. Create a directory in the home folder of user mpiuser

```
sudo mkdir /home/mpiuser/cloud
```

2. Edit the /etc/exports file to 'export' the folder i.e. make it remotely mountable by other machines

```
sudo nano /etc/exports
```

Add the following lines to the file

```
/home/mpiuser/cloud (tab) *(rw,sync,no_root_squash,no_subtree_check)
```

3. After editing the file, exit nano and run the following command. (It must be run every time changes are made to the /etc/exports file)

```
exportfs -a
```

4. Create the same directory in all the clients/slaves

```
sudo mkdir /home/mpiuser/cloud
```

5. Run nfs daemons on all machines

```
sudo systemctl start rpcbind
```

For clients

```
sudo systemctl start nfs
```

For server

```
sudo systemctl start nfs-server
```

6. Mount the directory remotely on all clients

```
sudo mount -t nfs masterhostname:/home/mpiuser/cloud /home/mpiuser/cloud
```

The first address is the source address on the master computer. The second path is the destination to mount on the slave computer. Instead of hostname, IP address of master may also be used.

Step 5: Run MPI programs

1. Copy all executable files to the shared /home/mpiuser/cloud directory on the master.

2. Create a 'machinefile' and add the IP Addresses of all the slaves to it, each on a new line. This machinefile tells the master exactly which computers to run the processes on.

```
sudo nano /home/mpiuser/cloud/machinefile
```

Enter the Ip-addresses of all the slaves, one on each line.

1.To run a Python program using the MPI4Py library, copy the demo program to /home/mpiusers/cloud and then use the following command.

```
mpiexec -n number of processes -machinefile /path/to/machinefile python /path/to/demo/program.py
```

For our system,

```
mpiexec -n 4 -machinefile /home/mpiuser/cloud/machinefile python
/home/mpiuser/cloud/path/to/demo/helloworld.py
```

This should run the file as needed.

4.2.2 Installation of packages

The following packages are required to be installed -

1 Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays [39].

2 Matplotlib is a 2D plotting library and it provides both a very quick way to visualize data from Python and publication-quality figures in many formats [40].

3 MPI4PY: This package provides Python bindings for the Message Passing Interface (MPI) standard [41].

4 Pylab is a module in matplotlib that gets installed alongside matplotlib. The Pylab interface is convenient for interactive calculations and plotting, as it minimizes typing [42].

5 OpenCV with the opencv_contrib modules

Installation of OpenCV from Source on Fedora 25 with opencv_contrib modules requires the following steps to be performed -

For installation process

```
sudo dnf install cmake python-devel numpy gcc gcc-c++ git
```

For GUI features

```
sudo dnf install gtk2-devel libdc1394-devel libv4l-devel gstreamer-plugins-base-devel
```

For image format support

```
sudo dnf install libpng-devel libjpeg-turbo-devel jasper-devel openexr-devel libtiff-devel libwebp-devel
```

For Thread Building Blocks support

```
sudo dnf install tbb-devel
```

For Eigen support

```
sudo dnf install eigen3-devel
```

Download OpenCV from GitHub

```
cd ~

git clone https://github.com/opencv/opencv.git
git clone https://github.com/opencv/opencv_contrib.git
```

Set up the build

```
cd ~/opencv

mkdir build

cd bulid
```

Now type in the following cmake command :-

CMake Command To Build OpenCV with OpenCV_Contrib

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \

-D CMAKE_INSTALL_PREFIX=/usr/local \

-D INSTALL_C_EXAMPLES=ON \

-D INSTALL_PYTHON_EXAMPLES=ON \

-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \

-D WITH_TBB=ON \

-D WITH_EIGEN=ON \

-D BUILD_DOCS=ON \

-D BUILD_EXAMPLES=ON ..
```

In the same folder, run the make command

```
make
```

This will take a good amount of time. Once the 'make' command finishes:

```
sudo make install
```

OpenCv should now be installed in : /usr/local/lib/python2.7/site-packages

Add this location to the PYTHONPATH in ~/.bashrc file in order to use the cv2 module.

```
sudo nano ~/.bashrc
```

Add the following to the end of the file

```
export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python2.7/site-packages
```

Save and exit.

Test the install

```
python2
>>>import cv2
```

4.2.3 Implementation of modules

The proposed system has two major phases concerned with -

1. Parallel Feature Detection and Description
2. Sequential Stitching of Images

1. Parallel Feature Detection and Description

The communicator variable comm provided by MPI4Py is used to store details about the multiprocessing environment. The comm.rank attribute gives the process ID of each process in the environment, while the comm.size attribute gives the total number of processes running in the system. This allows us to parallelize the required program in a single file.

This module scans the /data directory and stores all the names of the files in a list variable called folderlist. Images are read one by one using the cv2.imread() function by iterating over the list. This ensures that all the images are not read into the memory simultaneously and avoids a memory overflow problem.

A loop reads all the filenames in the list, loads each image and saves it in the img variable and processes it using the surf() function provided by the xfeatures2d class of the OpenCV library.

According to the value of the comm.rank variable i.e., depending on the process ID of the process, the image is processed by a different process.

From the set of processors the rank of each processor is obtained in comm.rank, and the ith image is assigned to the $i\,mod\,size^{th}$ process. The detectAndCompute() function performs the feature description of the detected feature point in the form of a 64 dimensional vector using the Speeded Up Robust Features algorithm[25].

The Keypoint objects returned by this function are serialized into a list containing image number, keypoint details and descriptor for each keypoint.

The serialized features are stored locally in the allfeat variable on each process. The MPI.gather() call is used to gather them all onto the root process and save them in the allfeat variable on the root process in the form of a list of four elements – each element contributed by a process.

This is converted into a single-level deep list of elements by using the iter.chain.from_iterable() function provided by the iter module and stored in the rawfeat variable. This list is then sorted according to image number to give the final list in the features variable.

The Keypoint objects from the sorted list in the features variable are deserialized to construct a final list of keypoints called allkps along with image number and another list called alldescrs that contains image-wise descriptors in the same order as that of the keypoints.

2. Sequential Stitching of Images

The actual process of stitching is performed sequentially by the system and consists of the following modules –

(a) Feature Matching Module

This module is implemented by the matchKeypoints() function that takes a pair of keypoints and their features as input from the allkps and alldescrs list. Matching of keypoints is performed using the FLANN matcher interface provided by the OpenCV library.

The parameters of the FLANN matcher object are stored in the dictionaries index_params and search_params. The matcher object is created and stored in the matcher variable. The knnMatch() function of this object performs matching by taking features from allkps and descriptors from alldescrs.
This produces a collection of Match objects that are used to calculate the homography for each pair of images. Each Match object consists of a queryId which stores the index of the feature being queried, a trainId consisting of the index of the matching feature, and distance between the matches.

(b) Homography Estimation Module

The Match objects created in the previous module are passed to the homography estimation module along with keypoint data where the trainId and queryId are used to extract (x,y) co-ordinates of keypoint matches. A transform from one image to another is estimated by the RANSAC (Random Sampling Consensus) algorithm using the EstimateHomography() function.
This transform is in the form of a 3x3 matrix stored in a Numpy array in the variable homography. This matrix stores the 2D perspective transform in homogeneous coordinates that should be applied to the image.

(c) Transforming and Compositing Module

First image is taken and compared pair-wise with all the remaining images till a matching image is found and then a homography is calculated using the Homography Estimation module. The level of match of the images is calcualated by a simple threshold value that has been experimentally calculated for maximum accuracy for the desired images.

If the homography exists, the transformation is applied to one of the images. If image is outside the canvas i.e. it is translated to negative coordinates by the homography, it is translated in the reverse direction to bring it into the visible are of the canvas. This same translation is applied to the other image in order to maintain consistency between their positions. Images are then blended using a simple bitwise_or() operation and stored in the variable finalimg. This combined image is then considered as a reference image. Its features are detected and stored in variable finalfeatures. It is then compared pairwise with the remaining images once more. This process continues till all the images have been stitched into the composite image. The composite image is stored as the final value of the finalimg variable and is saved in the results.png file in the present working directory using the cv2.imsave() function.

Chapter 5

Software Testing/Test Cases

5.1 Software Testing

The following testing has been performed on the system so as to ensure its proper functioning:

1) Unit testing of all modules
2) System integration testing

5.1.1 Unit Testing

1) **Feature Detection and Description Module**
The feature detection and description module was unit tested to ensure that images were being read from the /data directory correctly and that features were being detected and described. This was done by testing it with sample images and graphically plotting the features on the image where they were discovered. The keypoint descriptors were observed in the standard output of the terminal.

This module was tested in parallel as well by checking that the keypoints were converted to simple list objects on the all processes and combined into a single list on the root process along with identifying numbers for the image.

2) **Feature Matching Module**
The feature matching module was unit tested to ensure that it took two lists of features and computed matches by successfully returning Match objects.

3) **Homography Estimation Module**
The homography estimation module was unit tested by passing it a collection of Match objects along with the features to produce a homography. Accuracy of the homography was tested by applying the transform to a sample image and checking it visually.

4) **Transforming and Compositing Module**

The transforming and compositing module was tested with multiple pairs of images and their corresponding homographies in different positions and alignments. The stitched image was observed as the output.

5.1.2 System Integration Testing

The incremental approach to system integration testing has been used.

1) The feature detection and description module is first integrated with the feature matching module by passing the list of features to it. The feature matching module reads this list of features, constructs a new list of features by arranging them according to which image they belong to and performs matching of the features to return Match objects.
2) These two modules are then collectively integrated with the Homography Estimation module by passing the Match objects to the EstimateHomography() function which returns the homography between the two images. If the matches are insufficient i.e. the threshold value is not crossed, then this function returns None instead of the homography.
3) The above modules are then collectively integrated with the transformation and compositing module. The homography between two matching images is passed to this module along with the two images involved. The transforming and compositing module gives the stitched image as the output.

5.2 Test Cases

Tests cases for each module are as follows –

Table 5.1 Test cases for feature detection and description module

ID	Test Case Description	Expected Result	Actual Result	Test Case Result
001	Images should be read from the /data directory.	List in which images are stored should not be empty	List was found to contain all images present in the /data directory	Pass
002	Features should be detected in the image	Image should be shown with features marked by red points.	Image was shown with features marked in red and Keypoint objects were obtained for each feature.	Pass
003	Feature should be described using SURF algorithm	Feature descriptors should be written to standard output of terminal.	64 dimensional feature vectors were observed in the standard output of the terminal.	Pass

Table 5.2 Test cases for feature matching module

ID	Test Case Description	Expected Result	Actual Result	Test Case Result
004	Features should be read as input from the previous module on the root process.	*allfeat* variable on root process should have a list of features.	*allfeat* variable on the root process was found to contain features along with image number for each.	Pass
005	Checking for arrangements for features	*features* variable should contain a flatten list of only features grouped according to image number.	*features* variable was found to contain list of features grouped by image number.	Pass
006	Checking for arrangements for images	Variable *ic* should have images arranged in same order as their numbers in the feature list	When printed, *ic* showed images in the order in which their features were stored.	Pass
007	Checking for image matching according to features	List of Match objects should be returned	List of Match objects was printed to the standard output	Pass

Table 5.3 Test cases for homography estimation module

ID	Test Case Description	Expected Result	Actual Result	Test Case Result
008	Checking for correct homography estimation	Homography matrix is printed to standard output and image is transformed correctly	Homography matrix was printed correctly and image transformation was verified visually	Pass

Table 5.4 Test cases for transforming and compositing module

ID	Test Case Description	Expected Result	Actual Result	Test Case Result
009	Checking that two images are stitched horizontally	Horizontally stitched composite image	Two images were stitched in a row and saved as result.png	Pass
010	Checking that two images are stitched vertically	Vertically stitched composite image	Two images were stitched in a column and saved as result.png	
011	Checking that more than two images are stitched in both directions	Composite image with more than two images	Composite image was created of more than two images and saved as result.png	

Chapter 6

Experimental Results

6.1 Result Analysis

The performance of the parallel stage consisting of feature detection and feature description is shown in Table 6.1. Each image had an average size of 500px by 500px.

Table 6.1 Comparison of processing time of serial and parallel stages

Number of images	Time required on single node (in sec)	Time required on 4 nodes (in sec)
10	2.452	1.866
20	5.036	4.164
30	7.629	6.182
40	9.895	8.506
50	12.538	9.486
60	14.995	10.744
70	17.742	13.170
80	20.144	15.192
90	22.671	16.921
100	25.214	18.443
110	27.636	19.968
120	30.294	22.211
130	32.801	23.357
140	36.078	24.875
150	38.565	26.971
160	41.817	29.565
170	44.301	30.645
180	47.024	33.359
190	50.141	35.657
200	54.350	35.972

From Table 6.1 we observe that the performance of the parallel stage on the four node cluster is faster than on the single node cluster.

The performance improvement is summarized as follows in Figure 6.1

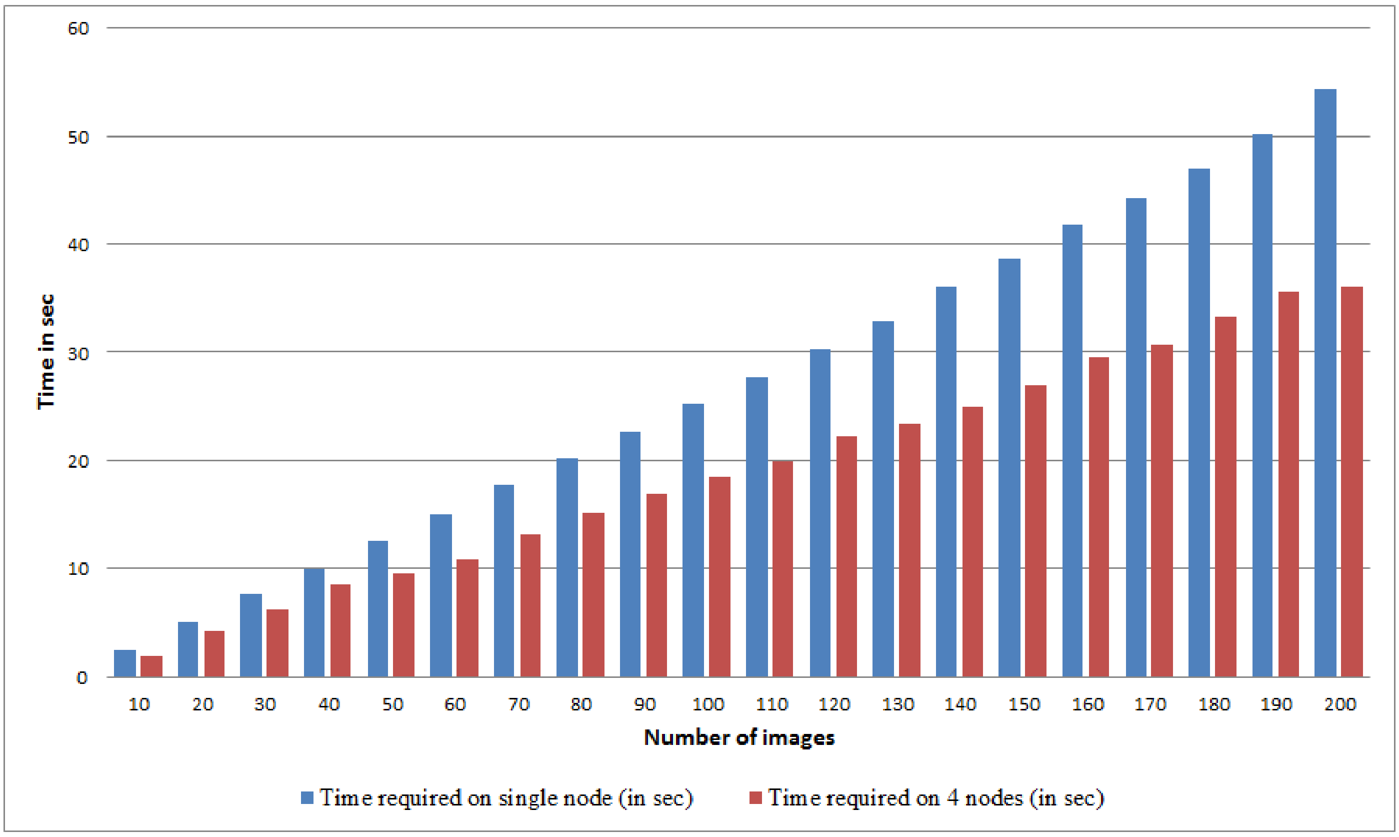


Figure 6.1 Bar graph comparison of processing time required in single node and 4 node

The performance of the entire process consisting of all the phases of parallel as well as sequential phases on a single node and a four node cluster is depicted in Table 6.2 below –

Table 6.2 Time taken for entire process in single node and on cluster

Number of images	Time taken for stitching on single node (in sec)	Time taken for stitching on cluster (in sec)
2	1.543	1.967
4	5.935	5.442
6	10.981	10.543
8	26.790	24.203
10	40.899	35.800
12	81.422	68.521
14	109.912	92.079
16	136.644	113.716
18	157.608	134.142
20	231.316	191.528
30	358.021	308.308
40	585.045	502.089
50	770.406	675.751
100	3197.495	2741.466

The comparison of time required for entire process including parallel and sequential phases in single node and in 4 nodes is shown in Figure 6.2

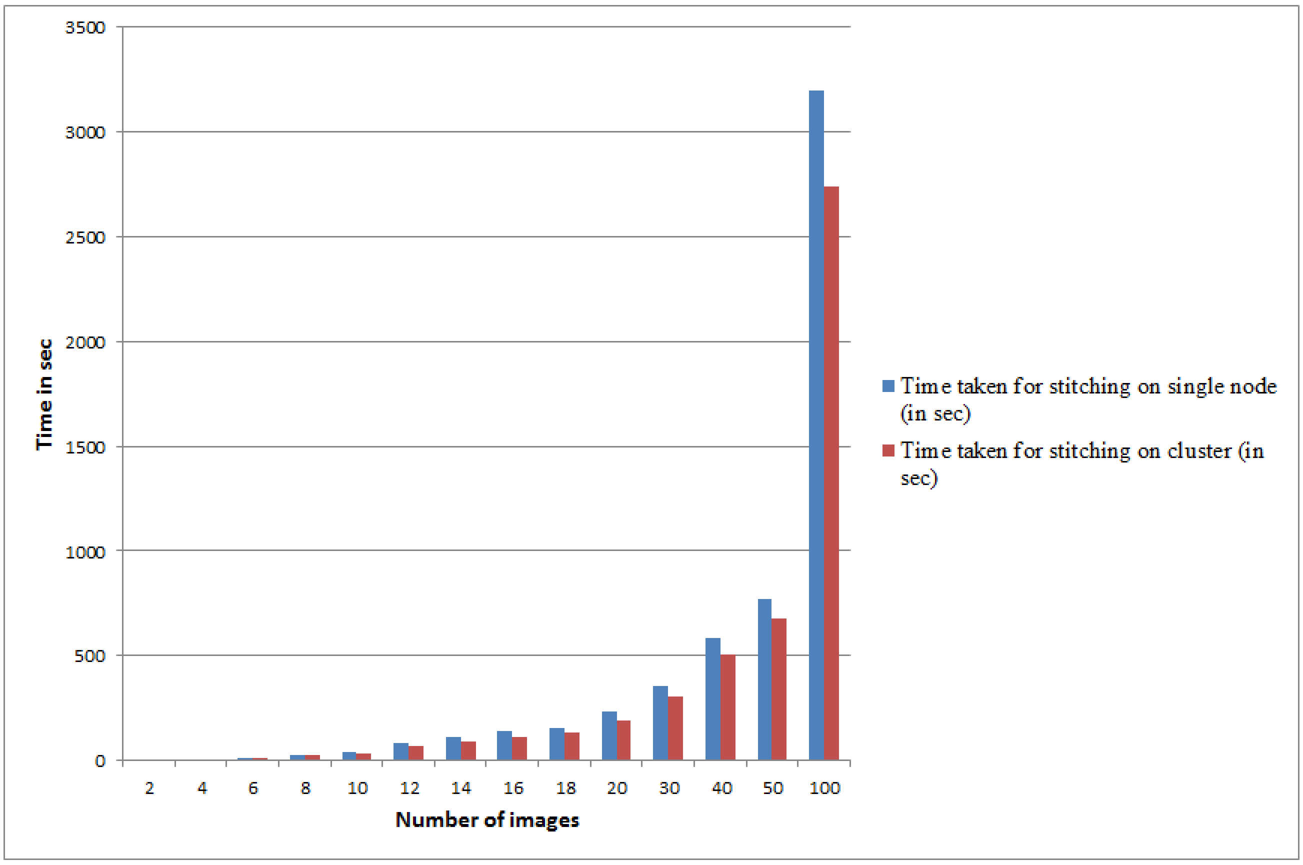
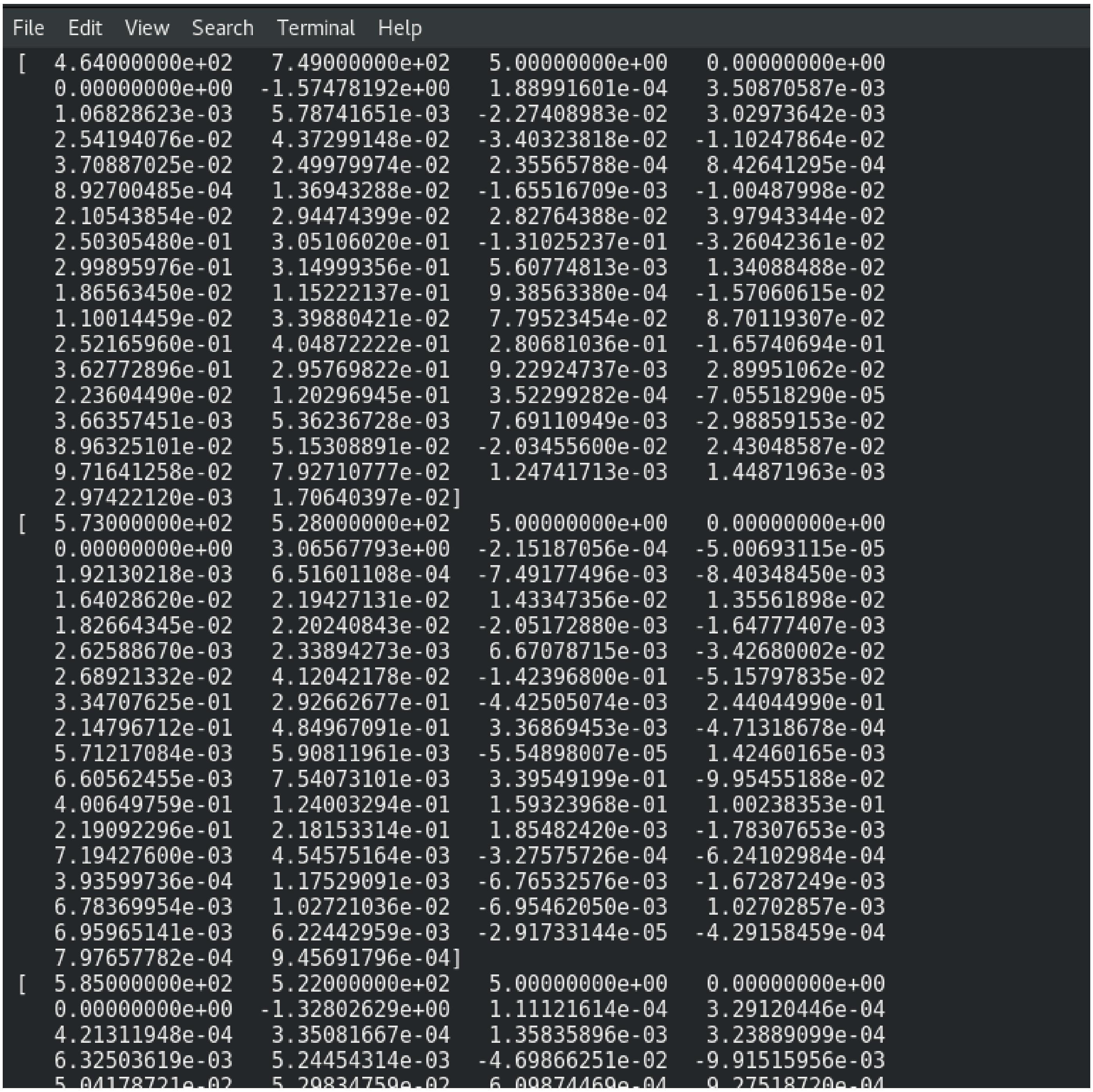


Figure 6.2 Bar graph comparison of entire process time required in single node and 4 nodes

6.2 Output

The output of each module is shown as follows. Figure 6.2 shows the features calculated at the end of the parallel feature detection and description stage.



```
[  4.64000000e+02   7.49000000e+02   5.00000000e+00   0.00000000e+00
   0.00000000e+00  -1.57478192e+00   1.88991601e-04   3.50870587e-03
   1.06828623e-03   5.78741651e-03  -2.27408983e-02   3.02973642e-03
   2.54194076e-02   4.37299148e-02  -3.40323818e-02  -1.10247864e-02
   3.70887025e-02   2.49979974e-02   2.35565788e-04   8.42641295e-04
   8.92700485e-04   1.36943288e-02  -1.65516709e-03  -1.00487998e-02
   2.10543854e-02   2.94474399e-02   2.82764388e-02   3.97943344e-02
   2.50305480e-01   3.05106020e-01  -1.31025237e-01  -3.26042361e-02
   2.99895976e-01   3.14999356e-01   5.60774813e-03   1.34088488e-02
   1.86563450e-02   1.15222137e-01   9.38563380e-04  -1.57060615e-02
   1.10014459e-02   3.39880421e-02   7.79523454e-02   8.70119307e-02
   2.52165960e-01   4.04872222e-01   2.80681036e-01  -1.65740694e-01
   3.62772896e-01   2.95769822e-01   9.22924737e-03   2.89951062e-02
   2.23604490e-02   1.20296945e-01   3.52299282e-04  -7.05518290e-05
   3.66357451e-03   5.36236728e-03   7.69110949e-03  -2.98859153e-02
   8.96325101e-02   5.15308891e-02  -2.03455600e-02   2.43048587e-02
   9.71641258e-02   7.92710777e-02   1.24741713e-03   1.44871963e-03
   2.97422120e-03   1.70640397e-02]
[  5.73000000e+02   5.28000000e+02   5.00000000e+00   0.00000000e+00
   0.00000000e+00   3.06567793e+00  -2.15187056e-04  -5.00693115e-05
   1.92130218e-03   6.51601108e-04  -7.49177496e-03  -8.40348450e-03
   1.64028620e-02   2.19427131e-02   1.43347356e-02   1.35561898e-02
   1.82664345e-02   2.20240843e-02  -2.05172880e-03  -1.64777407e-03
   2.62588670e-03   2.33894273e-03   6.67078715e-03  -3.42680002e-02
   2.68921332e-02   4.12042178e-02  -1.42396800e-01  -5.15797835e-02
   3.34707625e-01   2.92662677e-01  -4.42505074e-03   2.44044990e-01
   2.14796712e-01   4.84967091e-01   3.36869453e-03  -4.71318678e-04
   5.71217084e-03   5.90811961e-03  -5.54898007e-05   1.42460165e-03
   6.60562455e-03   7.54073101e-03   3.39549199e-01  -9.95455188e-02
   4.00649759e-01   1.24003294e-01   1.59323968e-01   1.00238353e-01
   2.19092296e-01   2.18153314e-01   1.85482420e-03  -1.78307653e-03
   7.19427600e-03   4.54575164e-03  -3.27575726e-04  -6.24102984e-04
   3.93599736e-04   1.17529091e-03  -6.76532576e-03  -1.67287249e-03
   6.78369954e-03   1.02721036e-02  -6.95462050e-03   1.02702857e-03
   6.95965141e-03   6.22442959e-03  -2.91733144e-05  -4.29158459e-04
   7.97657782e-04   9.45691796e-04]
[  5.85000000e+02   5.22000000e+02   5.00000000e+00   0.00000000e+00
   0.00000000e+00  -1.32802629e+00   1.11121614e-04   3.29120446e-04
   4.21311948e-04   3.35081667e-04   1.35835896e-03   3.23889099e-04
   6.32503619e-03   5.24454314e-03  -4.69866251e-02  -9.91515956e-03
   5.04178721e-02   5.29834759e-02   6.09874469e-04   9.27518720e-04
```

Figure 6.3 SURF features calculated by parallel stage

Figure 6.3 shows the features detected in the first stage depicted by red dots.

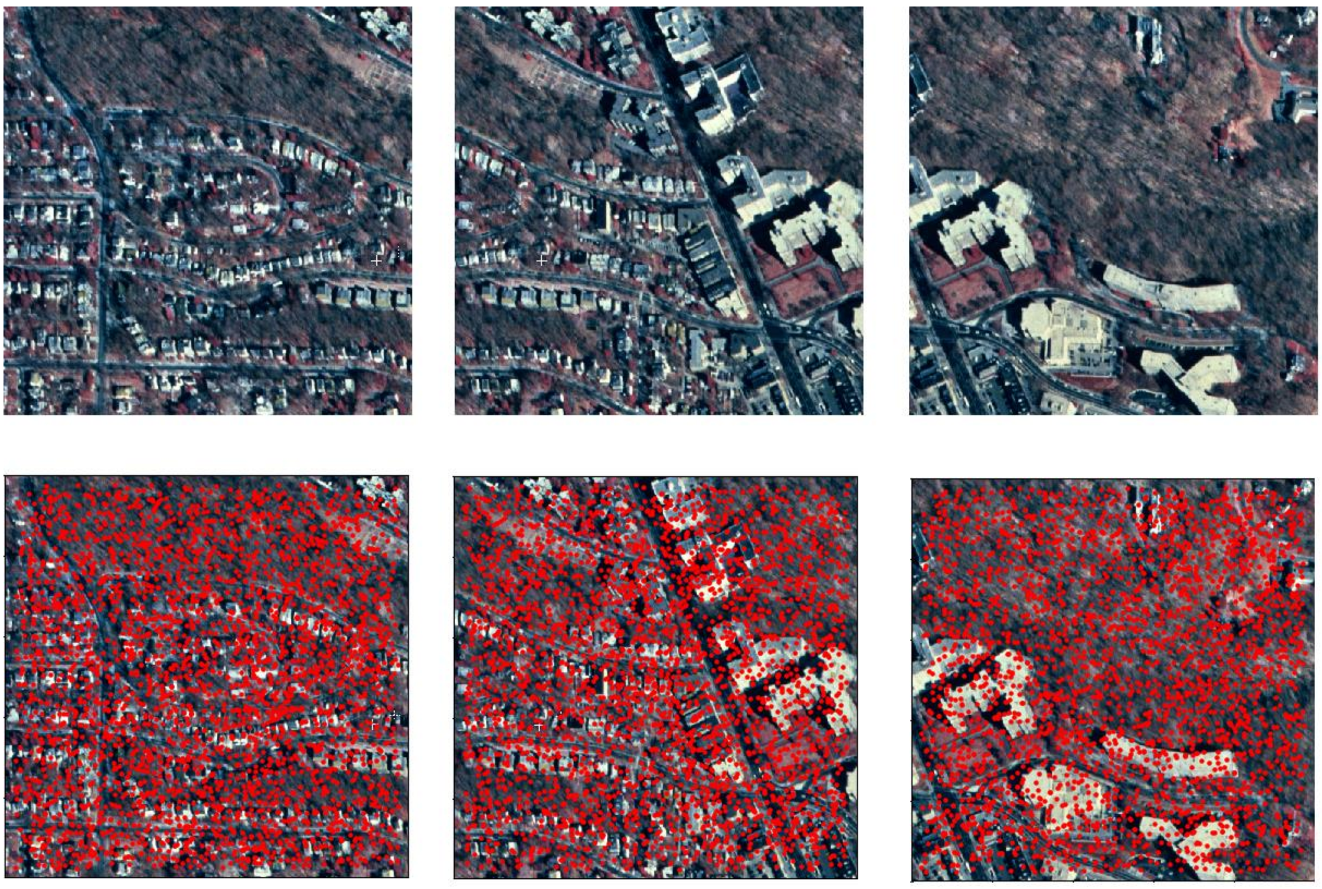


Figure 6.4 Features detected in three satellite images

Figure 6.3 shows the matches between two sample images plotted as red lines between the two images.

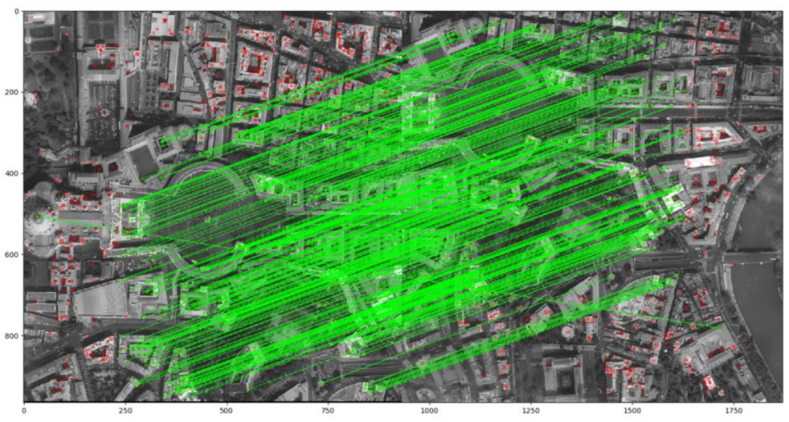
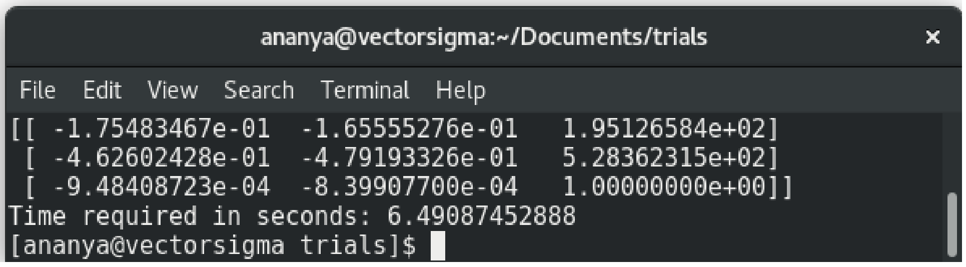


Figure 6.5 Output of feature matching module

Figure 6.5 shows the output of the homography estimation module in the form of a 3x3 perspective transform matrix.


ananya@vectorsigma:~/Documents/trials
File Edit View Search Terminal Help

```
[[ -1.75483467e-01  -1.65555276e-01   1.95126584e+02]
 [ -4.62602428e-01  -4.79193326e-01   5.28362315e+02]
 [ -9.48408723e-04  -8.39907700e-04   1.00000000e+00]]
Time required in seconds: 6.49087452888
[ananya@vectorsigma trials]$
```

Figure 6.6 Output of homography estimation module

The sequential stitching of two images and its result with the third image is show in Figure 6.6

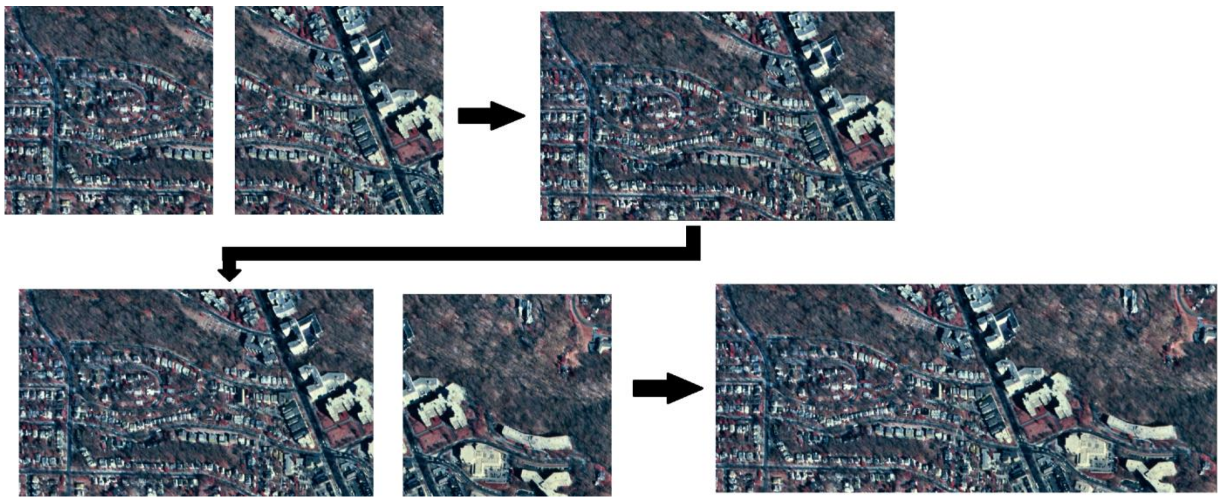


Figure6.7 Sequential stitching of three images

Figure 6.7 shows the final stitched image obtained from 50 images. The size of the final image was 6215px by 1228px and it took up 14.6 Mb of storage space.

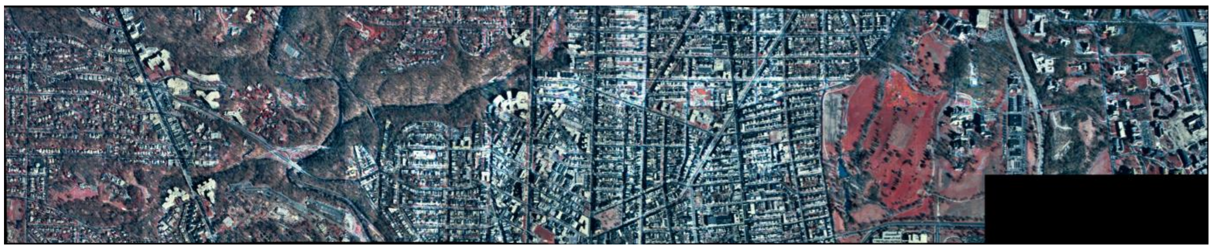


Figure 6.8 Final composite image

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

Worldwide economic losses due to natural disasters exceeded USD 630 billion in the last decade [3]. Monitoring these risks efficiently so that action can be taken in a timely manner is of utmost importance. The proposed system aims to parallelize algorithms required in the stitching of satellite images by using distributed memory architecture. The use of invariant features and random sampling will make this system more robust, while maintaining its accuracy.

The proposed system shows a better performance as compared to the serial implementation of algorithms for smaller set of images tested. The performance of the proposed system will increase with increase in number of images. The proposed system shows a performance improvement of about 1 minute 40 seconds when executed compared to execution of algorithms on a single node for the processes to execute for 50 images. We can see here that there is a clear improvement in the processing of the algorithms when it is parallelized using the proposed system.

7.2 Future Scope

The proposed system can be used for improving the performance of existing algorithms, by increasing the number of nodes connected. With changes, number of nodes can be extended.

Parallelizing the algorithm for feature matching, homography estimation and transforming and compositing can improve the performance of the system even better.

Appendix I

Code Sample

```
#! /usr/bin/python2

from __future__ import print_function
from operator import itemgetter
import numpy as np
import matplotlib
from pylab import *
import cv2
import sys
import os
from mpi4py import MPI
import itertools


def matchKeypoints(kpsA, kpsB, featuresA, featuresB, ratio = 0.75, reprojThresh = 4.0):
    # compute the raw matches and initialize the list of actual
    # matches

    FLANN_INDEX_KDTREE = 0
    index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
    search_params = dict(checks=50)
    matcher = cv2.FlannBasedMatcher(index_params,search_params)
    rawMatches = matcher.knnMatch(np.asarray(featuresA,np.float32), np.asarray(featuresB,np.float32),
k=2)
    matches = []

    # loop over the raw matches
    for m in rawMatches:
        # ensure the distance is within a certain ratio of each
        # other (i.e. Lowe's ratio test)
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            matches.append((m[0].trainIdx, m[0].queryIdx))
    #print(matches)
    # computing a homography requires at least 4 matches
    if len(matches) > 50:
        # construct the two sets of points
        ptsA = np.float32([kpsA[i].pt for (_, i) in matches])
        ptsB = np.float32([kpsB[i].pt for (i, _) in matches])


        # compute the homography between the two sets of points
        (H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC,reprojThresh)

        # return the matches along with the homograpy matrix
        # and status of each matched point
        return H

    # otherwise, no homograpy could be computed
    return None

def crop(img):
    tmp = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _,thresh = cv2.threshold(tmp,0,255,cv2.THRESH_BINARY)
    contours = cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    cnt = contours[0]
    x,y,w,h = cv2.boundingRect(cnt)
    crop = img[y:y+h,x:x+w]

    return crop

def stitchtwo(imageA,imageB,H):
    tx=H[0][2]
    ty=H[1][2]
    M1=np.array([[1,0,0],[0,1,0]], float32)
    M2=np.array([[1,0,0],[0,1,0]], float32)

    if tx<0:
        M1[0][2]=-tx
#        M2[0][2]=imageA.shape[1]
    if ty<0:
        M1[1][2]=-ty
#        M2[0][2]=imageA.shape[0]

    resA = cv2.warpAffine(imageA, M1,
(2*imageA.shape[1]+imageB.shape[1],2*imageA.shape[0]+imageB.shape[0]))
    #imshow(resA)
    #show()
    resA = cv2.warpPerspective(resA, H,
(2*imageA.shape[1]+imageB.shape[1],2*imageA.shape[0]+imageB.shape[0]))
    #imshow(resA)
    #show()
    resB = cv2.warpAffine(imageB, M1,
(2*imageA.shape[1]+imageB.shape[1],2*imageA.shape[0]+imageB.shape[0]))
    #imshow(resB)
    #show()
    img = cv2.bitwise_or(resA,resB)
    img = crop(img)
    #imshow(img)
    #show()

    return(img)
    #show()


start = MPI.Wtime()
features = []
```

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
descriptor = cv2.xfeatures2d.SURF_create()
allfeat=[]
#ic = io.ImageCollection('data/*')
folder="data/"
folderlist = os.listdir(folder)
for i, filename in enumerate(folderlist):
    if rank==i%size:
        img = cv2.imread(os.path.join(folder,filename))
        if img is not None:
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            (kps, desc) = descriptor.detectAndCompute(img, None)
            for kp,des in zip(kps,desc):
                temp = (i,kp.pt, kp.size, kp.angle, kp.response, kp.octave, kp.class_id, des)
                allfeat.append(temp)


allfeat = comm.gather(allfeat, root=0)

if rank==0:
    ic = None
    rawfeat=list(itertools.chain.from_iterable(allfeat))
    features=sorted(rawfeat, key=itemgetter(0))
    #print(features)
    rawfeat=None


    imgkps = []
    imgdescrs = []
    allkps = []
    alldescrs = []
    homographies = []
    previmgno, currimgno = 0,0
    previmgno = 0
    for feature in features:
        currimgno = feature[0]
        if previmgno != currimgno:
            allkps.append(imgkps)
            alldescrs.append(imgdescrs)
            imgkps = []
            imgdescrs = []
        tempkp = cv2.KeyPoint(x=feature[1][0],y=feature[1][1],_size=feature[2], _angle=feature[3],
_response=feature[4], _octave=feature[5], _class_id=feature[6])
        tempdesc = feature[7]
        imgkps.append(tempkp)
        imgdescrs.append(tempdesc)
        previmgno = currimgno
    allkps.append(imgkps)
    alldescrs.append(imgdescrs)
    #print(len(allkps))
    features = None
    folder="data/"
    folderlist = os.listdir(folder)



    finalimg = cv2.imread(os.path.join(folder,folderlist[0]))
    finalimg = cv2.cvtColor(finalimg, cv2.COLOR_BGR2RGB)
    finalfeatures = allkps[0]
    finaldesc = alldescrs[0]
    donelist = [0]

    #print ("Before loop")

    for j in range(len(folderlist)-1):
        for i, filename in enumerate(folderlist,0):
            if i not in donelist:
                img = cv2.imread(os.path.join(folder,filename))
                if img is not None:
                    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                    homography=matchKeypoints(allkps[i],finalfeatures,alldescrs[i],finaldesc)
                    if homography is not None:
                        donelist.append(i)
                        finalimg = stitchtwo(img,finalimg,homography)
                        finalfeatures,finaldesc = descriptor.detectAndCompute(finalimg, None)

                        #imshow(finalimg)
                        #show()

    cv2.imwrite(“result.png”,finalimg)



else:
    assert allfeat is None

if rank == 0:
    end = MPI.Wtime()
    print(end-start)
```

REFERENCES

[1] V. Chaturvedi, "The costs of climate change impacts for india," tech. rep., CEEW, New Delhi,India, 2016.

[2] "India loses usd 9.8 billion every year due to disasters," 2015.

[3] "Extreme losses from natural disasters - earthquakes, tropical cyclones and extratropical cyclones," tech. rep., Applied Insurance Research, Inc.

[4] "Jpeg - aboutjpeg," https://jpeg.org/about.html, 2016.

[5] M. J. Flynn, "Some computer organizations and their effectiveness," IEEE Transactions on Computers, vol. C-21, no. 9, pp. 948-960, 1972.

[6] W. Stallings, Computer Organization and Architecture: Designing for Performance. Prentice Hall, 2000.

[7]"Intel® Core™ i7 Processor Extreme Edition and X-Series Lineup", *Intel*, 2017. [Online]. Available: http://www.intel.ca/content/www/ca/en/processors/core/core-i7ee-processor.html.

[8] "Intel® Xeon® Processor E7 Family", *Intel*, 2017. [Online]. Available: http://www.intel.in/content/www/in/en/products/processors/xeon/e7-processors.html.

[9] "Opteron™ Series CPU Processors | AMD", *Amd.com*, 2017. [Online]. Available: http://www.amd.com/en-us/products/server/opteron.

[10] H. A. El-Rewini and M. El-Barr, Advanced computer architecture and parallel processing, John Wiley, 2005.

[11] P. Keleher, A. L. Cox, S. Dwarkadas, and W. Zwaenepoel, "Treadmarks: Distributed shared memory on standard workstations and operating systems," in Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference, WTEC'94, (Berkeley, CA, USA), pp. 10-10, USENIX Association, 1994.

[12] E. Brewer, "Clusters: Multiply and conquer," Data Communications, 1997. [13] "Mpi forum," 2016.

[14] "Open mpi: Open source high performance computing," 2016.

[15] "Mpich : High performance portable mpi," 2016.

[16] D. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, vol. 60, no. 2, pp. 91-100, 2004.

[17] H. Moravec, "Proceedings of the 5th international joint conference on artificial intelligence-volume 2," in A Treatise on Electricity and Magnetism (J. C. Maxwell, ed.), vol. 2, pp. 68-73, Oxford Clarendon, 1977.

[18] C. Schmid, R. Mohr, and C. Bauckhage, "Evaluation of interest point detectors," International Journal of Computer Vision, vol. 37, no. 2, pp. 151-172, 2000.

[19] "Opencv tutorials - opencv," http://docs.opencv.org/2.4/docltutorials!tutorials. html, 2016. [20] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection" IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 1, pp. 105-119, 2010.

[21]R. Horan and M.Lavelle, "The laplacian," rhoran@plymouth.ac.uk, mlavelle@plymouth.ac.uk, 2005.

[22] T. Lindeberg, "Detecting salient blob-like image structures and their scales with a scalespace primal sketch: A method for focus-of-attention," International Journal of Computer Vision, vol. 11, no. 3, pp. 283-318, 1993.

[23] J. Babaud, A. Witkin, M. Baudin, and R. Duda, "Uniqueness of the gaussian kernel for scale-space filtering," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 1, pp. 26-33, 1986.

[24] T. Lindeberg, "Edge detection and ridge detection with automatic scale selection," International Journal of Computer Vision, vol. 30, no. 2, pp. 117-156, 1998.

[25] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-up robust features (surf)," Computer Vision and Image Understanding, vol. 110, no. 3, pp. 346-359, 2008.

[26] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: binary robust independent elementary features," in ECCV' 10 Proceedings of the I 1th European conference on Computer vision: Part IV, pp. 778-792, 2010.

[27] J. Freidman, J. Bentley, and R. Finkel, "An algorithm for finding best matches in logarithmic expected time," ACM Transactions on Mathematical Software, vol. 3, no. 3, pp. 209- 226, 1997.

[28] C. Silpa-Anan and R. Hartley, "Optimised kd-tree for fast image descriptor matching," in Proceedings of IEEE conference on computer vision and pattern recognition, 2008.

[29] M. Muja and D. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," VISAPP International Conference on Computer Vision Theory and Applications, 2009.

[30] A. Olivia and Y. Aytar, "C6.869 advances in computer vision." MIT, 2015.

[31] M. Fischler and R. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," Communications of the ACM, vol. 24,no.6,pp.381-395.

[32] R. Raguram, J. Frahm, and M. Pollefeys, "A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus." Lecture Notes in Computer Science.

[33] 0. Chum, J. Matas, and J. Kittler, "Locally optimized ransac." Lecture Notes in Computer Science, 2003. pp. 236-243.

[34] R. Kooper, P. Bajcsy, and N. Hem Aandez, "Stitching giga pixel images using parallel computing," in Parallel Processing for Imaging Applications, 2011.

[35] W. Liao, T. Hsieh, W. Liang, Y. Chang, C. Chang, and W. Chen, "Real-time spherical panorama image stitching using opencl," hgpu.org, 2012.

[36] W. Liao, T. Hsieh, and Y. Chang, "Gpu parallel computing of spherical panorama video stitching," in IEEE 18th International Conference on Parallel and Distributed Systems, 2012.

[37] M. Brown and D. Lowe, "Automatic panoramic image stitching using invariant features," International Journal of Computer Vision, vol. 74, no. 1, pp. 59-73, 2006.

[38] The Linux Operating System: An Introduction – Shahid H. Bokhari

[39]J. Johnson, "Python Numpy Tutorial", *Cs231n.github.io,* 2017. [Online]. Available: http://cs231n.github.io/python-numpy-tutorial/#numpy.

[40]N. Rougier, "Matplotlib tutorial", *Labri.fr*, 2017. [Online]. Available: https://www.labri.fr/perso/nrougier/teaching/matplotlib/.

[41]"mpi4py 2.0.0 : Python Package Index", *Pypi.python.org*, 2017. [Online]. Available: https://pypi.python.org/pypi/mpi4py.

[42]"Usage — Matplotlib 2.0.0 documentation", *Matplotlib.org*, 2017. [Online]. Available: http://matplotlib.org/faq/usage_faq.html.

Paper Published

And

Certificates

Parallelization of Algorithms for Image Stitching Using Distributed Memory Architecture

Ananya Satoskar
Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Vashi, Navi Mumbai, India

Vishnu Nambiar
Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Vashi, Navi Mumbai, India

Aishwarya Mohan
Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Vashi, Navi Mumbai, India

Christina Rainy
Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Vashi, Navi Mumbai, India

Lakshmi Gadhikar
Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Vashi, Navi Mumbai, India

***Abstract*—The efficient use of satellite images for practical purposes depends upon the ability to combine them for the purposes of analysis. However, this process involves many computationally intensive computer vision algorithms. Assembling large images usually requires costly hardware with a large shared memory. This paper addresses this problem by proposing a system that implements algorithms required in the process of image stitching, viz. algorithms for feature detection, feature description, feature matching, image transformation and image compositing, in parallel on a distributed memory architecture. The proposed system aims to improve the performance of the image stitching process while providing a more easily scalable system.**

Keywords—image stitching; distributed memory architecture; parallel computing

I. INTRODUCTION

Vast amounts of data are generated by the daily capture of satellite images. These images play an important role in monitoring the climate and predicting weather conditions and hazards. However, satellite images require a system that combines them into a composite image if they are to be used effectively. This process requires a lot of computational power due to the intensive nature of the algorithms involved and the large quantities of image data they process. The proposed system implements these algorithms in parallel on multiple processors in distributed memory architecture.

Distributed memory architecture refers to a computing system with multiple independent processors each having their own primary memory. The processors are interconnected so that they can communicate with each other. Each processor is capable of running its own process, thus allowing the parallel execution of many processes at once.

The process of image stitching involves following stages. The first step is to detect features or interest points [1] in the input images that can be used to find out if they belong to the same output image by matching them in different images. Once features have been detected, they must be represented in a manner that is distinct and repeatable [2]; they must contain adequate information and be described in a manner that is invariant to scale, rotation and affine transformations. This representation is known as the feature descriptor.

In the next stage, these features must be matched in order to find out which images represent the same real-world objects. The matching process is performed using approximation algorithms.

Once the features have been matched, images are transformed to minimize minor differences in perspective, scale, etc. before they are combined using a blending algorithm. This transformation is usually estimated by obtaining a homography matrix [3].

An overview of the proposed system is shown in Fig. 1

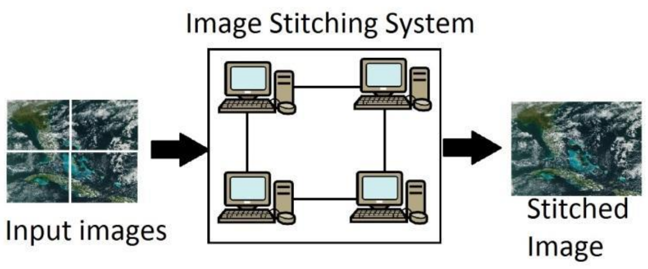


Fig. 1. Overview of the proposed system

The cameras present in a satellite capture parts of a particular geographical area. These images are gathered and sent to the proposed system, where they are combined into a single image of higher resolution to provide a wider view of the area.

According to Szeliski [4], the task of image stitching concerns itself with the following problems – finding distinctive features or interest points in images and efficiently matching these features, determining the mathematical model that relates pixels in one image to pixels in another, and seamless blending of overlapping images onto a composite surface.

Transformations that map images onto each other are found by using statistical sampling methods as this process needs to be robust to minor variations in images.

Blending of images in the final step must ensure that images are combined as seamlessly as possible by handling moving objects and preserving the structure of the image at multiple levels of detail i.e. they must simultaneously preserve the clarity of sharper features while being robust to noise.

Our approach selects a blob-detector [5] algorithm for the detection and description of features, followed by a nearest neighbour approximation algorithm for matching the features, and blends images using a pyramid-based approach.

II. Related Work

The system 'Stitching giga pixel images using parallel computing', proposed by Kooper, et al[6], addresses the problem of stitching aerial imagery acquired during flights over Costa Rica by the utilization of multiple hardware architectures. This system uses the Speeded Up Robust Features (SURF) algorithm [7] to detect and describe features, and matches them using Euclidean distance and the Delauney triangulation algorithm. Then, transformation parameters are estimated using a Least Square Fit method and the images are transformed and composited using bi-cubic interpolation.

The parallelized system was implemented by constructing a pyramid of image tiles for faster indexing and retrieval that are distributed between clients by a server. The final stitched image was created on a cluster with 16 compute nodes and a single head node, each having 16GB of primary memory, using a Server Message Block file system. A 79 GigaPixel image consisting of 600,000 tiles was computed in 3 days.

This system deals with film-based rather than digital images and requires coarse georeferencing information to group images before they are combined using intensity-based stitching.

Liao, et al[8] developed a system 'Real-Time Spherical Panorama Image Stitching Using OpenCL' that captures images in real-time and converts them to 360 degree panoramas by stitching them together. It uses a multi-core architecture (NVIDIA 9600GT) to process images captured during webcam video using the OpenCL library. One of the main features of this project is the complete 360 degree panorama which is created by transforming the images using a real-time co-ordinate system converting module to present their panorama in a three-dimensional spherical image.

Eight images taken by a low resolution webcam could be processed simultaneously to create spherical panoramas. The performance enhancement was experimentally observed to be 76 times [9] when compared to the implementation of the system on a CPU.

This system uses webcam-quality images as input and requires expensive hardware. The system cannot be easily scaled as it uses shared memory architecture.

Brown and Lowe [10] proposed AutoStitch - an automatic image stitching software which is used to stitch images in the most effective way by finding matching points in all input images with the help of the Scale Invariant Feature Transform (SIFT) [11] algorithm. AutoStitch then proceeds with the image stitching process by robustly aligning all images and using advanced blending algorithms to form a composite image with a much larger field of view. However, this software is proprietary and is not implemented in parallel.

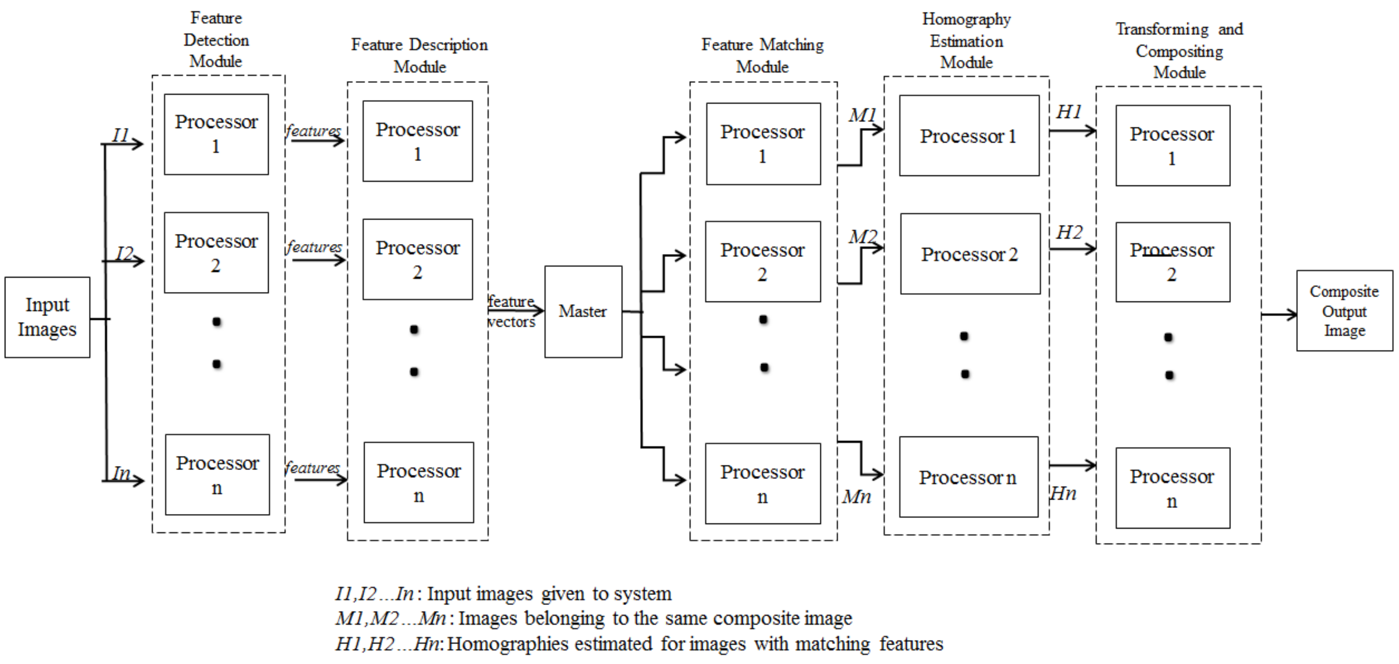
III. Proposed System

The proposed system will parallelize algorithms for feature detection, feature description, and feature matching in images and finally, for image transformation. It will be implemented on a distributed memory architecture consisting of multiple independent nodes.

The main steps involved in the image stitching process are -

1. Feature Detection and Description
2. Feature Matching
3. Estimation of Homography Matrix
4. Transformation and Compositing of Images

They are implemented using the following five modules as shown in Fig. 2:-



Input Images
Feature Detection Module
Feature Description Module
Feature Matching Module
Homography Estimation Module
Transforming and Compositing Module
I1
I2
In
features
feature vectors
Master
M1
M2
Mn
H1
H2
Hn
Processor 1
Processor 2
Processor n
Composite Output Image
I1,I2…In: Input images given to system
M1,M2…Mn: Images belonging to the same composite image
H1,H2…Hn: Homographies estimated for images with matching features

Fig. 2. Architecture of proposed system to parallelize algorithms involved in image stitching

A. *Feature Detection Module*

This module will use the Determinant of Hessians [7] algorithm to detect blobs in the images given to it as input, and detect interest points in each image. This algorithm is chosen as it is invariant to scale, rotation and affine transformation. The coordinates of the centre point of each feature will be sent to the next step of the system.

B. *Feature Description Module*

The feature description module will be responsible for representing the detected feature points in the form of 128 dimensional vectors using the Speeded-Up Robust Features (SURF) descriptor devised by Bay, et al [7]. It will extract information about the intensity gradient of the pixels around each feature point detected in the feature detection stage by calculating and summing up wavelet responses for each feature. This descriptor has been shown to outperform most other descriptors in terms of accuracy.

C. *Feature Matching Module*

This module will implement a randomized kd-tree algorithm [12] to perform a nearest neighbour approximation for each feature in order to find other matching features. This algorithm constructs multiple kd-trees randomly and searches them simultaneously using a single queue to keep track of nodes. The feature matching module will also check for a particular threshold of matches for each pair of images so as to decide which images are to be combined in the final output by creating connected sets of images for each panorama.

D. *Homography Estimation Module*

A homography matrix will be estimated using the Random Sampling Consensus algorithm [13] that will define the transformation required for one image to be combined with another. The homography matrix defines the transformation required to align one image to another. This will be done for every image in the connected component based on a single image chosen as a reference.

E. *Transforming and Compositing Module*

Once the homography has been estimated, the Optimal Seam Selection Algorithm [4] will be used to detect where images can be combined most seamlessly by placing the join where the images agree. This overcomes the problem of handling moving objects in the picture. The images will then be combined using Laplacian Pyramid Blending [4]. This algorithm constructs a pyramid of the images at different levels of detail and combines each level separately before collapsing it into the final image.

IV. Conclusion

This paper proposes a system to parallelize algorithms required in the stitching of satellite images by using distributed memory architecture. The use of invariant features and random sampling will make this system more robust, while maintaining its accuracy. The multi-level blending approach will preserve detail in the images while providing a smoother transition between them. The use of independent nodes will improve performance of these algorithms while making the system easier to scale, as hardware can be added as required.

References

[1] H. Moravec, "Proceedings of the 5th international joint conference on Artificial intelligence -

Volume 2", 1977, p. 584. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.

[2] C. Schmid, R. Mohr and C. Bauckhage, *International Journal of Computer Vision*, vol. 37, no. 2, pp. 151-172, 2000. K. Elissa, “Title of paper if known,” unpublished.

[3] C. Gava and G. Bleser, "2D Projective Transformations (homographies)", Technische Universität Kaiserslautern, Germany.

[4] R. Szeliski, "Image Alignment and Stitching: A Tutorial", *Foundations and Trends® in Computer Graphics and Vision*, vol. 2, no. 1, pp. 1-104, 2006. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

[5] T. Lindeberg, "Detecting salient blob-like image structures and their scales with a scale-space primal sketch: A method for focus-of-attention", *International Journal of Computer Vision*, vol. 11, no. 3, pp. 283-318, 1993.

[6] R. Kooper, P. Bajcsy and N. Hernández, "Stitching giga pixel images using parallel computing", *Parallel Processing for Imaging Applications*, 2011.

[7] H. Bay, A. Ess, T. Tuytelaars and L. Van Gool, "Speeded-Up Robust Features (SURF)", *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346-359, 2008.

[8] W. Liao, T. Hsieh, W. Liang, Y. Chang, C. Chang and W. Chen, "Real-Time Spherical Panorama Image Stitching Using OpenCL", 2012.

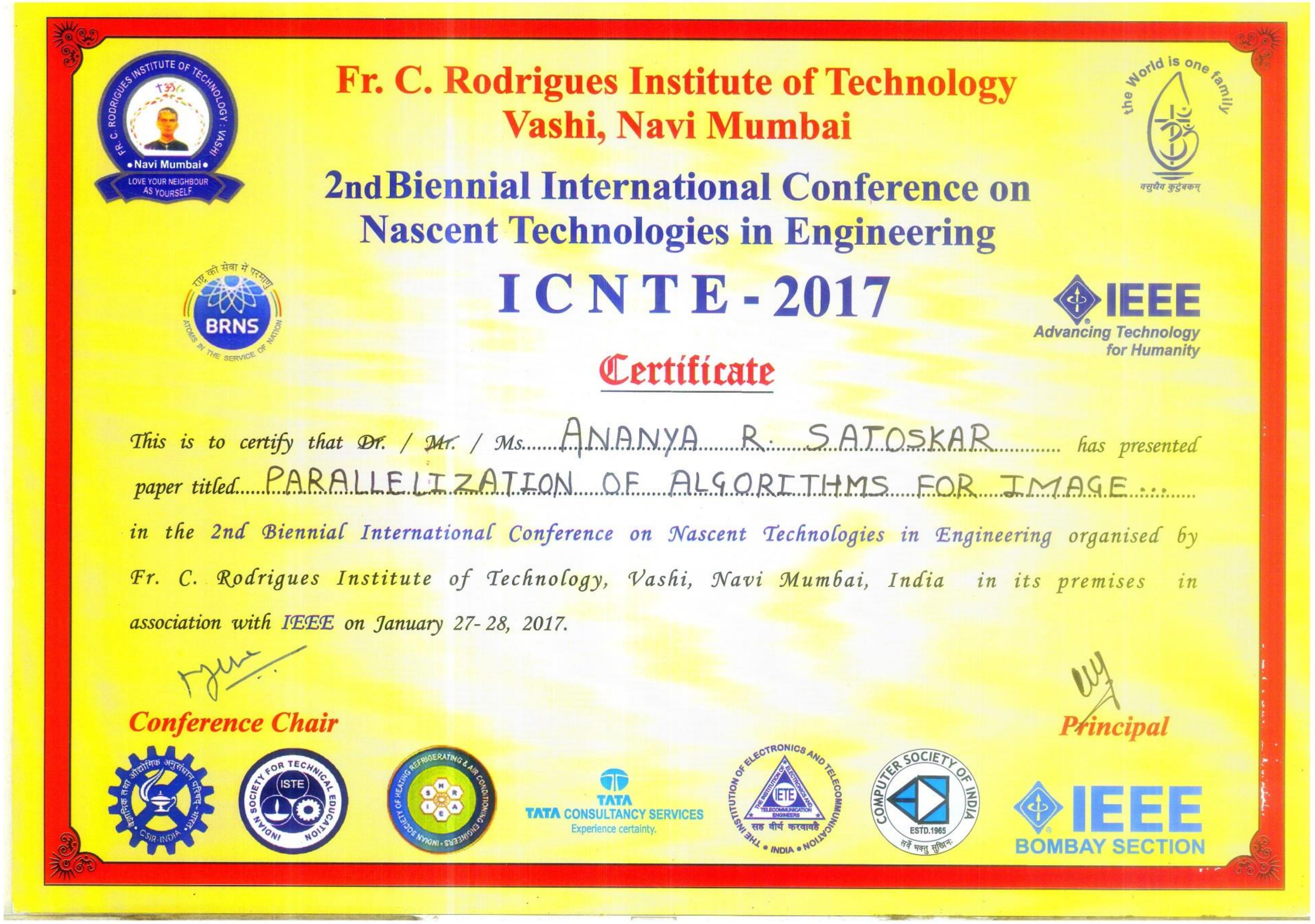
[9] W. Liao, T. Hsieh and Y. Chang, "GPU Parallel Computing of Spherical Panorama Video Stitching", *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, 2012.

[10] D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, 2004.

[11] M. Brown and D. Lowe, "Automatic Panoramic Image Stitching using Invariant Features", *International Journal of Computer Vision*, vol. 74, no. 1, pp. 59-73, 2006.

[12] R. Hartley and C. Silpa-Anan, "Optimised KD-trees for fast image descriptor matching", *CVPR, IEEE Computer Society*, 2008.

[13] M. Fischler and R. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography", *Communications of the ACM*, vol. 24, no. 6, pp. 381-395, 198



Fr. C. Rodrigues Institute of Technology
Vashi, Navi Mumbai

2nd Biennial International Conference on
Nascent Technologies in Engineering

ICNTE - 2017

Certificate

This is to certify that ~~Dr.~~ / ~~Mr.~~ / Ms. ANANYA R. SATOSKAR has presented paper titled PARALLELIZATION OF ALGORITHMS FOR IMAGE ... in the 2nd Biennial International Conference on Nascent Technologies in Engineering organised by Fr. C. Rodrigues Institute of Technology, Vashi, Navi Mumbai, India in its premises in association with IEEE on January 27-28, 2017.

[illegible]
Conference Chair

[illegible]
Principal

ACKNOWLEDGEMENT

The making of the project Parallelization of Algorithms Using Distributed Memory Architecture involves contribution of many people.

We express our gratitude to Dr. Khot, Principal of Fr. C. Rodrigues Institute of Technology, Prof. Archana Shirke, H.O.D. of IT department, and Prof. Dhanashree Hadsul and Prof. Kalpana Wani, Project Coordinators of IT department, for extending their inevitable and valuable support to us.

We would like to take this opportunity to thank our guide Prof. Lakshmi Gadhikar for guiding and correcting various documents related to this report with great attention and care. This synopsis would not have been completed without her knowledge and expertise.

We would like to thank Prof. H. K. Kaura for providing us with the inspiration for this project. We also thank the institution and our faculty members for their help and support during this project proposal. We also extend our heartfelt thanks to our families and our classmates.

Ananya Satoskar

Vishnu Nambiar

Aishwarya Mohan

Christina Rainy