PARALLEL ALGORITHMS FOR IMAGE AND VIDEO MOSAIC BASED APPLICATIONS

by

HONGYU WANG

(Under the Direction of Suchendra M. Bhandarkar)

ABSTRACT

Image and video mosaic based applications have received growing interest in recent times. Image and video mosaicking techniques have been applied to various fields such as video compression, video enhancement, digital libraries, interactive video analysis, virtual environments, low-bitrate video transmission, and interactive video editing and manipulation systems. Despite all the improvements that have been made, image and video mosaicking is still a computationally intensive process. This characteristic is crucial when one considers deploying the mosaicking technique in a real time environment such as live video transmission.

Two parallel programming paradigms for a video mosaicking algorithm are discussed in this thesis, namely, shared memory programming algorithm (SMPA) using POSIX threads and distributed memory programming algorithm (DMPA) using MPI. Experimental results from a Sun Fire server and a cluster of Sun Blade workstations show that our SMPA achieves a linear speedup in most cases and a super linear speedup on some occasions.

INDEX WORDS: Parallel Programming, Pthreads, MPI, Image and Video Mosaic. Motion Panorama

PARALLEL ALGORITHMS FOR IMAGE AND VIDEO MOSAIC BASED APPLICATIONS

by

HONGYU WANG

B.E., Guangzhou Jinan University, China, 2000

A Thesis Submitted to the Graduate Faculty of

The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2005

PARALLEL ALGORITHMS FOR IMAGE AND VIDEO MOSAIC BASED APPLICATIONS

by

HONGYU WANG

Major Professor:	Suchendra M. Bhandarkar
Committee:	Eileen T. Kraemer
	Thiab R. Taha

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2005

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to my advisor Dr. Suchendra M. Bhandarkar, for his patient guidance and instructions at every step from preparation for the research to the draft of this thesis.

I would also like to thank Dr. Eileen T. Kraemer and Dr. Thiab R. Taha for their time and support as members of my committee.

My parents have supported me all the time. I would not have completed this research without their love and encouragement.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

The process of combining a set of small images into a larger composite image is known as mosaicking. With the help of a computer, mosaicking techniques have improved significantly so that various image and video mosaic-based applications have been proposed and some of them have already been commercialized.

1.1 Image and Video Mosaic Based Applications

The most obvious mosaicking applications are image and video compression (since mosaics are efficient scene representations) and as a means of visualization (since mosaics provide a wide and stable view). However, mosaics are also useful in many other applications, such as scene change detection, and video search and indexing, video editing and manipulation [1]. In addition, virtual environments and virtual travel are also two popular image and video mosaic-based applications [2].

1.1.1 Mosaic Based Compression

The motivation for using mosaics for video compression is the observation that the physical world captured by a camera typically consists of some moving objects superimposed over a dominant stationary background. The large scale temporal and spatial correlations within this background layer [12] make the video highly compressible.

In [3], Irani, Hsu, and Anandan describe a technique for mosaic based video compression. The technique combines information from a sequence of video frames into a coherent panoramic view of the scene captured by a moving camera. Two mosaic types are introduced, static mosaics and dynamic mosaics. In static mosaics, the input video sequence is segmented into contiguous scene sub-sequences, and the mosaic image is constructed for each scene subsequence to provide a snapshot view of the subsequence. This is done by aligning all frames of that sequence to a fixed coordinate system and then integrating the aligned images into a mosaic image. Figure 1.1 shows an example of a static mosaic image.

Figure 1.1: Static mosaic

The static mosaic image exploits long term temporal redundancies (over the entire scene subsequence) and large spatial correlations (over large portions of the image frames), and is therefore an efficient scene representation. It is ideal for video storage and retrieval, especially for rapid browsing in large digital libraries and to obtain efficient access to individual frames of interest. However, the changes in the scene (termed as “residuals”) with respect to the

background (e.g., a moving object) in the sequence are not captured by a static mosaic image and therefore need additional representation.

On the other hand, a dynamic mosaic is a sequence of evolving mosaic images. The content of each new mosaic image is updated with the most current information from the most recent frame. The sequence of dynamic mosaics can be visualized either with a stationary background (e.g., by completely removing any camera induced motion, see Figure 1.2), or in a manner such that each new mosaic image frame is aligned with the corresponding input video image frame (see Figure 1.3). In general, since changes between successive frames are relatively small, the amount of "residual" information in the dynamic mosaic will be smaller than that in the static case. A dynamic mosaic is an ideal tool for low bit-rate transmission. However, due to its incremental frame reconstruction, it lacks the important capability of random access to individual frames, which is essential for video manipulation and editing.

Figure 1.2: Dynamic mosaic with a stationary background

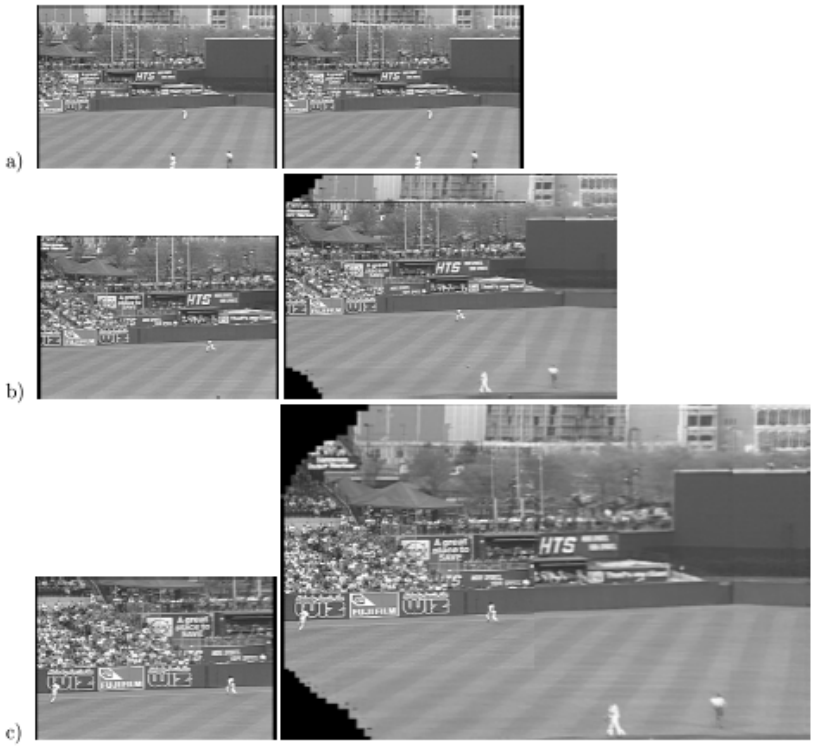


Figure 1.3: Dynamic mosaic with updating of the coordinate system

In each scheme, the mosaic construction process aligns the images using a global parametric motion transformation, usually canceling the effect of camera motion on the dominant portion of the scene. The residual motions that are not compensated by the parametric motion are then analyzed for their significance and encoded. In many applications where there is significant camera motion (e.g., remote surveillance), a mosaic-based scheme performs substantially better than traditional inter-frame compression methods, and offers the potential for very low bitrate transmission. In storage applications, such as digital libraries and video editing environments, it has the additional benefit of enabling direct access to and retrieval of individual frames of interest.

In addition to video compression, image and video mosaics are also used in image based rendering applications. Image based rendering is a powerful new approach for generating real time photorealistic computer graphics. In image based rendering applications, light rays from different directions are captured and then resampled in order to generate different views of a scene. Since the number of acquired images is often very large, it is therefore necessary to compress the images in order to store them efficiently. Leung and Chen [6] describe a mosaic based compression technique for an image based rendering application. In their scheme, the mosaic image is first constructed and then used to predictively encode the original images. Furthermore, motion compensation is applied to provide a closer match at the block level between the predicted image and the original image. Their experimental results show that mosaic based compression with motion compensation provides better performance in the rate-distortion sense compared to intra-frame coding while at the same time providing the advantage of random access over inter-frame coding.

1.1.2 Mosaic Based Visualization

One of the key benefits of a mosaic is that it offers a means of enhanced visualization. The panoramic view of the mosaic provides the scene context necessary for the viewer to better appreciate the events that take place in the video [13]. Several different types of panoramic visualizations are possible, each highlighting a different type of information.

Teodosio and Bender introduce the notion of Salient Stills [7], which are a class of images that reflect the aggregation of the temporal changes that occur in a moving-image sequence with the salient features of individual frames preserved. Multiple frames of an image sequence that may include variations in focal length or field of view are combined to create a single still image

(see Figure 1.4). The still image may contain multiresolution patches, a larger field of view, or higher overall resolution than any individual frame in the original image sequence. It may also contain selected salient objects from any one frame of the sequence of video frames. The still provides a visual summary of camera and object movements of an entire series of moving frames and represents the underlying scene better than any single frame. Salient Stills can therefore be used as "key frames" for rapid browsing through the entire video sequence which is digitally stored, and for other purposes as well (see Section 1.1.5).

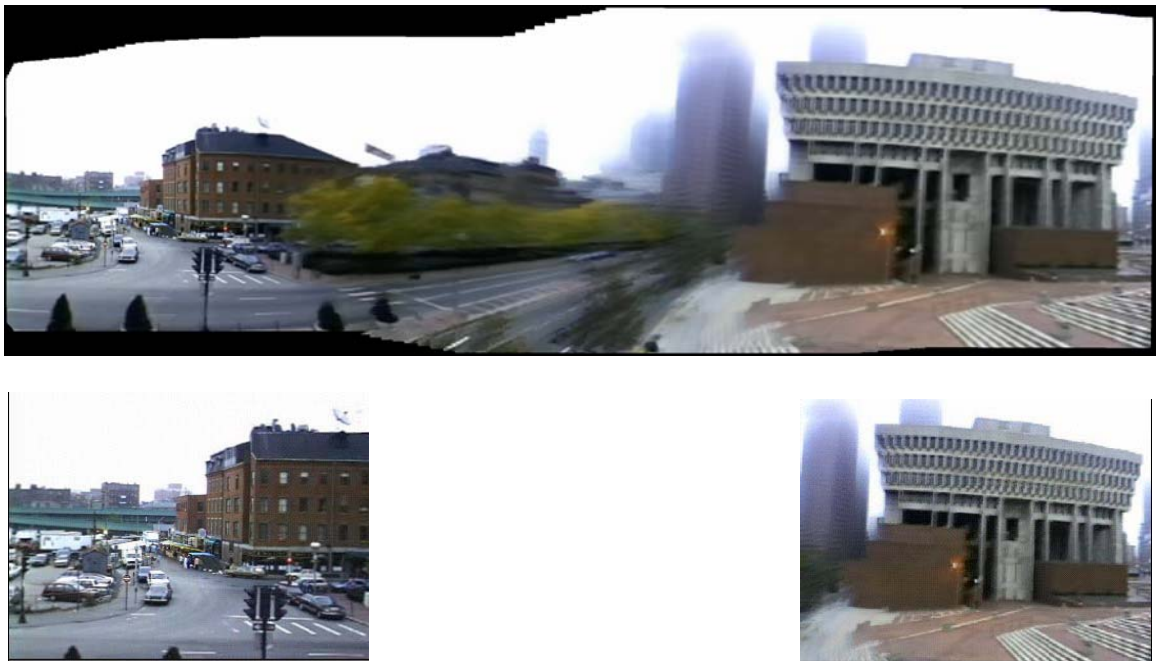


Figure 1.4: Salient still (top) with selected key frames (bottom)

While the salient stills mosaic is useful for capturing the background with the salient features preserved, in some cases, it may be desirable to get a synopsis of the events that take place within the video sequence (see Figure 1.5). This can be achieved by producing a background

panorama and the registration or overlay of the video sequence, i.e., the camera motion and dynamic layer. Two classes of techniques can be used to register the dynamic scenes: feature-based methods [9] and direct methods [8].

Direct methods use measurable image information such as pixel brightness variations or image cross-correlation measures, while feature-based methods rely on correspondence of a sparse set of highly reliable image features. Feature-based methods minimize an error measure that is based on the distances between a few corresponding features. However, many alignment problems are caused in cases where the detected features are not uniformly distributed across the frames. On the other hand, direct methods minimize an error measure that is based on direct image information collected from all pixels in the image, but they only work well typically when the dynamic areas are small compared to the overall scene size.

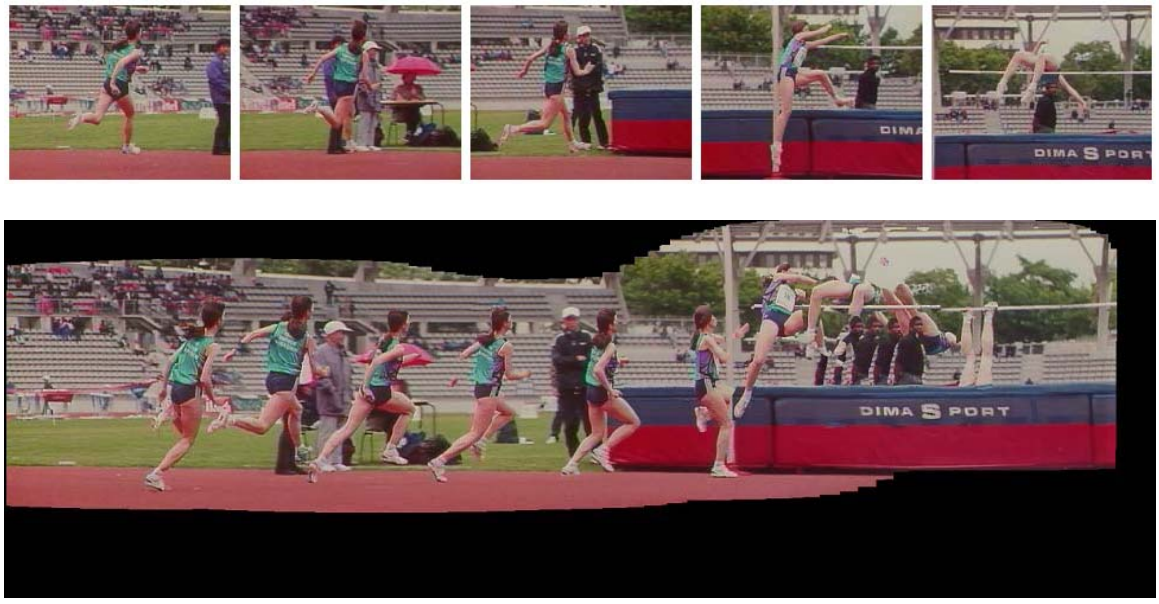


Figure 1.5: Motion panorama

Based on the complementary nature of these two methods, Bartoli et al. [14] proposed a two-step technique for the construction of Motion Panoramas. Their technique uses a feature-based method to initialize the registration, i.e. compute camera motion and perform layer segmentation, and a direct method to fine tune this registration. Several contributions to both feature-based and direct methods are given in their paper. In particular, they propose an efficient feature-based registration method, starting from robust frame-to-frame registration and ending with global bundle adjustment, followed by an integrated framework for motion panorama construction based on direct registration of frames. It is worthwhile to mention that a young company, Dartfish, commercializes software for producing static mosaics from videos [11]. Their mosaic construction procedure requires manual intervention both for building the background and for selecting dynamic objects to be eventually overlaid onto the background.

The above examples only show that the panoramic mosaic works well for static visualization. However, a mosaic is also well suited for dynamic visualization. In dynamic visualization, a new video sequence (also called the Mosaic Video) is generated which is a sequence of dynamic mosaic images. This type of visualization simulates the output of a virtual camera with desired features. For example, a virtual camera with an expanded field of view can be simulated in this fashion. Alternatively, a desired new camera trajectory can be simulated by applying the appropriate coordinate transformations to each of the mosaic video frames. The simplest example of this is a stabilized video mosaic display, in which case the camera motion is completely removed. Figures 1.2 and 1.3 show the examples of a mosaic video. Such a display has uses in various applications such as remote navigation, and remote surveillance.

1.1.3 Video Enhancement

Mosaic representations can serve as a useful and efficient tool for producing high quality stills from video as well as enhancing an entire video sequence. Image resolution depends on the physical characteristics of the camera: the optics and the density and spatial response of the detector elements. Increasing the resolution by sensor modification may not always be viable option. However, an increase in the sampling rate could be achieved by obtaining more samples of the scene from a sequence of displaced pictures. Therefore, aligning the sequence frames over a finer mosaic grid provides higher sampling rate of the background scene, and hence integrating over that grid provides higher spatial resolution. When the blur function of the camera is also known or can be computed and used for deblurring, the increase in resolution is even more pronounced. This method is known as super resolution mosaicking and is discussed in [15]. This idea was used by a company, Salient Stills Inc. Their product uses data from surrounding scan lines to reduce the residual noise at each pixel location [16].

1.1.4 Virtual Environment / Reality

Traditional virtual reality systems use 3D computer graphics to model and render virtual environments in real-time. This approach usually requires laborious modeling and expensive special purpose rendering hardware. The rendering quality and scene complexity are often limited because of real-time constraints. Chen [5] presents a new approach which uses 360-degree cylindrical panoramic images to compose a virtual environment. The panoramic image is digitally warped on-the-fly to simulate camera panning and zooming. The panoramic images can be created with computer rendering, specialized panoramic cameras or by "stitching" together overlapping photographs taken with a regular camera. Walking in a space is currently

accomplished by "hopping" to different panoramic points. This mosaic based approach has been used in the commercial product QuickTime VR, a virtual reality extension to Apple Computer's QuickTime digital multimedia framework. In addition to panoramic viewing, the system includes viewing of an object from different directions and hit-testing through orientation-independent hot spots.

1.1.5 Other Application

Image mosaics have also been used in many other applications. Some of these relate to managing large digital libraries, including video editing and manipulation and video browsing and search.

In video editing, it is sometimes required to modify the data in a video segment. For example, pulling an existing actor or object out of the sequence (while filling in the occluded regions convincingly, of course), or inserting a non existing object into the video sequence. It is a tedious work if the editing process is done manually, frame-by-frame. The process can be significantly speeded up and be done more accurately, using motion analysis and mosaic construction. Mosaic-assisted video editing is described in [10].

A collection of static mosaics for the different detected scene segments is useful for rapid browsing of a database of sequences. For such initial browsing, it can suffice to retrieve the key frame mosaic (salient stills) of the background scene alone (i.e., without the residuals), or alternatively the synopsis mosaic (motion panoramas). Once a scene of interest has been detected, the part of the video tape which corresponds to it can be retrieved on demand. The mosaic image can therefore be used to index into the video data.

1.2 The Need for Parallelization

Despite all the improvements that have been made, image and video mosaicking is still a computationally intensive process. For example, in [17], Pan describes an implementation of motion panorama construction from a streaming video. However, it takes more than 9 minutes for the algorithm to process a 12-frame video sequence on a Sun workstation with an UltraSPARC 1.05GHz CPU, where a 1-second video sequence usually consists of 24 frames. Therefore, even though the basic ideas underlying some image and video mosaic-based applications may be very good, they might not work very well in practice without faster processing capability even if mosaic generation is done off-line. For applications with real time constraints, the need for faster processing is even more pressing.

Two parallel programming paradigms for a video mosaicking algorithm are discussed in this thesis, namely, shared memory programming using Pthreads and distributed memory programming using MPI. Experimental results from a Sun Fire 880 server and a cluster of Sun Blade 1500 workstations show that on a shared memory environment, our parallel mosaic generation algorithm achieves a linear speedup in most cases and super linear speedup on some occasions. The speedup results in a distributed memory environment, though not as good as those in a shared memory environment, are quite satisfactory.

This paper is organized as follows: An overview of motion panorama construction and parallel computing is given in Chapter 2. In Chapter 3, we provide the implementation details for each parallel programming model. Performance analysis is described in Chapter 4. Finally, the conclusions and some directions for future work will be summarized in Chapter 5.

CHAPTER 2

OVERVIEW

2.1 Motion Panorama Construction

Motion panorama construction consists of three major phases: static background generation, background / foreground (moving objects) segmentation and final panorama composition. During the first phase, the homographies corresponding to the motion of the camera are computed for certain frames. The static background for the entire scene expressed in the video sequence is generated by stitching the individual frames into a large wide-angle panoramic image using these homographies. Then in the second phase, the dynamic foreground, which includes regions corresponding to both moving objects and false detections in the scene, is segmented by warping together three consecutive frames in the video sequence and consequently detecting the intensity discrepancy at each pixel. The noise in the dynamic foreground is smoothed using a Gaussian filter (see Section 2.1.3). Regions corresponding to false motion are deleted using a size filter to generate the connected components corresponding to real moving objects. Finally, the foreground objects are pasted back onto the static background using the location information such as the homographies and position coordinates computed during the second phase.

Since homography estimation is used in both background mosaic generation and dynamic foreground segmentation, also, it requires the most computation. Therefore, we discuss this process in a separate section.

2.1.1 Homography Estimation

In order to estimate the homographies for the geometric transformations between frames, the corresponding points between frames must be known. If all possible corresponding points are considered, the computation complexity is usually very high. Pan describes an algorithm that simplifies this process by examining only the “interest points”, for example, the corners of objects [17]. These interest points are detected by the Moravec corner detector [18]. The detector first calculates the interest value of each pixel in the frame using the following equation:

$$MO(i,j) = \frac{1}{8} \sum_{k=i-3}^{i+3} \sum_{l=j-3}^{j+3} \left| f(k,l) - f(i,j) \right| \tag{2.1}$$

Where $f(i,j)$ and $f(k,l)$ are the color values of the pixels at (i,j) and (k,l) in the frame.

Each frame to be processed is then divided into a number of neighboring and non-overlapping windows of size 30x30. For each window, the pixel with maximum interest value is marked as the interest point within this region. This is done to solve the problem of detected interest points not being homogeneously distributed across the frames. Figure 2.1 shows an example of the detected interest points.

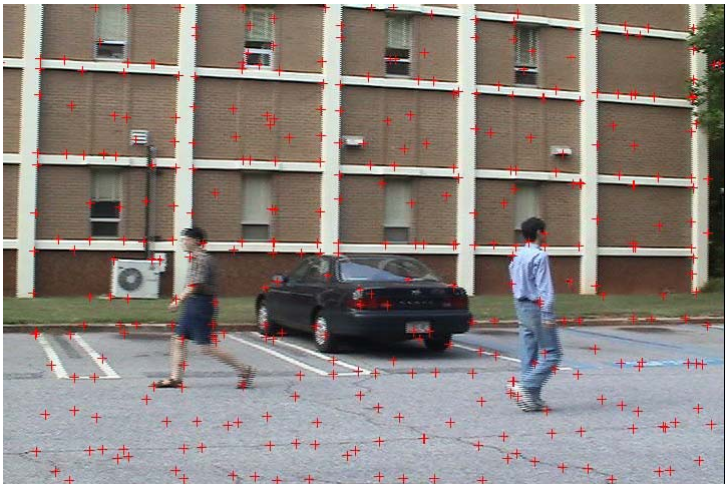


Figure 2.1: Interest points detected by the Moravec corner detector

Once the interest points are extracted from one frame, the Template Matching [19] technique is used to search the next frame in the video sequence for all corresponding points. The current implementation uses a 15×15 template in the current frame with the interest point at the top left corner and searches a 105×105 region in the next frame. The putative points detected by the above technique are not necessarily the real correspondences. It has been reported that more than 40% of correspondences obtained by the above technique are incorrect [9]. Therefore, a robust estimation method such as RANSAC is used for a more accurate estimation. The implementation of the RANSAC technique is done using the following steps:

- Randomly select 4 correspondences which may include both the correct ones and the mismatched ones to compute a homography H.
- Compute the Euclidean distance for every correspondence $\{ x_i \leftrightarrow x_i^{'} \}$ using the following function:

$$\sqrt{d^2(x_i^{'}, Hx_i)} \tag{2.2}$$

- Compute the number of inliers whose Euclidean distance is less than a threshold D. These inliers constitute a consensus set S.
- Repeat the above steps for M samples.
- After M samples, re-compute H from the consensus set with the largest number of inliers.

For the number of samples *M*, if one chooses to try all possible samples, then $M = C_4^n$, where *n* is number of correspondences. Even for a modest value of *n*, the total number of possibilities will be huge, which implies very expensive computation. Therefore, a value of *M*=5000 is empirically chosen for the algorithm. Experimental results show that this number is sufficient for finding the largest possible number of inliers in most cases (i.e. by increasing the sample number *M*, the largest number of inliers remain the same).

The homography obtained by robust estimation can be used as a guideline for further optimal estimation. All correspondences $\{ x_i \leftrightarrow x_i^{'} \}$ between any two frames are calculated by the function given in equation (2.2). The outliers of these correspondences are filtered out using the same threshold value used in RANSAC procedure. The correspondences classified as inliers are then used to determine a maximum likelihood estimate of H by minimizing the following object function:

$$f_1 = \sum_i [d(x_i, H^{-1}x_i^{'})^2 + (x_i^{'}, Hx_i)^2] \qquad (2.3)$$

In our experiments, a linear least squares method is used to obtain the optimal H which best satisfies all the inliers.

2.1.2 Background Mosaic Generation

The homography H for any two non-consecutive frames is obtained exactly by the composition of homographies of all the frames between them. For example, the homography for the first and the third frame H_{13} can be computed by the composition of the homography between the first frame and the second frame and the second frame and the third frame as $H_{13} = H_{23}H_{12}$. By using a special frame as the reference frame such as the first frame or the last frame, the homographies between all frames in the video sequence and this reference frame can be computed. Consequently, all frames can be mapped onto the reference frame to generate the background mosaic.

2.1.3 Dynamic Foreground Segmentation

The frames extracted from the original motion video can be segmented into two layers: the static background layer and the dynamic foreground layer. The static background layer generated by the procedure described in the previous section includes all relatively static objects in the scene such as buildings or mountains. The dynamic foreground layer that needs to be segmented, on the other hand, is associated with the moving objects such as walking people or moving cars.

The foreground segmentation for dynamic scenes captured by moving camera is computationally much more complex than the case when the dynamic scenes are capture by static cameras. The camera motions such as pan and tilt usually compensate for the motion of the moving objects in the scene such that these objects remain in the center of the frame. For example, actors or athletes always stay in the center of the images or frames of the movie sequence because the camera is panned or tilted in order to follow them.

A possible approach is to first map the previous and next frames onto the current frame using the estimated homographies. The color values of every pixel in the frame are then compared at each pixel location. The pixels belonging to the static background follow the estimated camera motion and hence the changes of intensity value between the corresponding pixels are relatively small. On the other hand, large discrepancy in intensity values occurs at pixels which do not conform to the estimated homography. The comparison of color values at each pixel location is achieved by the following distance function:

$$f_2 = d(\Gamma_{i-1}(q_{i-1}), \Gamma_i(q_i)) + d(\Gamma_{i+1}(q_{i+1}), \Gamma_i(q_i)) \tag{2.4}$$

where $\Gamma_i(q_i)$ is the intensity value of the pixel q in the ith frame of the video sequence, and d is the Mahalanobis distance [20] which represents the discrepancy in color values between the two pixels when they appear in two consecutive frames. The Mahalanobis distance is given by:

$$f_3 = d(\Gamma_{i-1}(q_{i-1}), \Gamma_i(q_i)) = (\Gamma_{i-1}(q_{i-1}), \Gamma_i(q_i))^T C^{-1} (\Gamma_{i-1}(q_{i-1}), \Gamma_i(q_i)) \tag{2.5}$$

where C is the covariance matrix for the RGB color space, and is estimated using red, green and blue color values for all the pixels and for each frame in the video sequence.

Based on the values obtained for each pixel location in the frame computed using the function in equation (2.4), a probability image is generated. It is made up of the likelihood of every pixel in the frame belonging to the dynamic foreground. For example, a large discrepancy in color value at the same pixel position *q* in three consecutive frames has a large probability of being the dynamic foreground. The pixels belonging to moving objects which do not conform to

the homography between consecutive frames have a greater probability of possessing larger discrepancy in color values.

False detection which is caused by noise or the presence of large homogeneous regions and complex motions can be solved by performing a Gaussian filtering on the probability image. The Gaussian smoothing filter is very well suited for removing noise that is drawn from a normal distribution. In the context of image processing, the two-dimensional zero-mean discrete Gaussian filter is given by

$$g[i, j] = e^{\frac{(i^2 + j^2)}{2\sigma^2}}$$

and is used as a smoothing filter. The smoothing procedure is performed on the probability image instead of the original frame. The detected regions corresponding to the moving objects prior to smoothing and the regions corresponding to the moving objects after smoothing by a Gaussian filter are illustrated in figure 2.2.

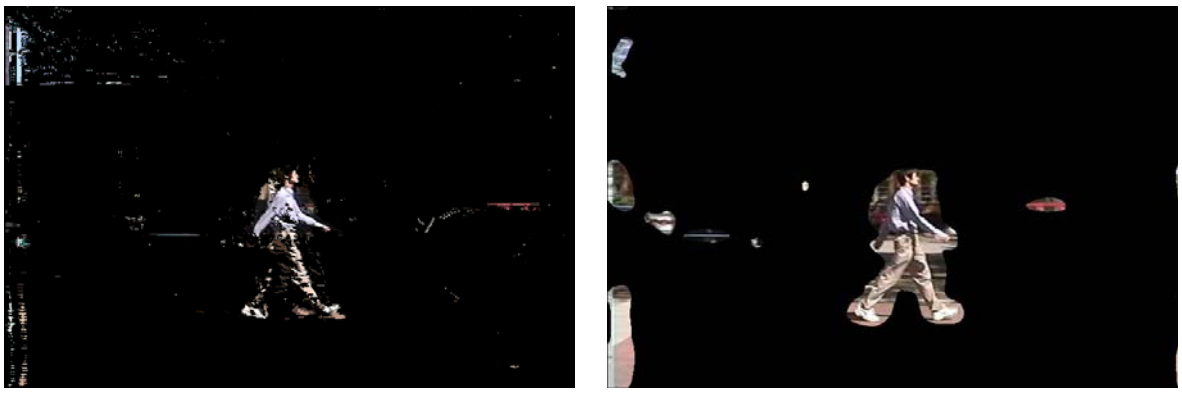


Figure 2.2: The frame on the left shows the detected regions corresponding to the moving objects prior to smoothing. The one on the right shows the detected regions corresponding to the moving objects after smoothing by a two-dimensional Gaussian filter.

All the frames in the video sequence are now divided into two layers: the static background layer and the dynamic foreground layer. To find all the connected components in the dynamic foreground which includes both real moving objects and the noisy or spurious regions, each frame is converted to a binary image where the value “1” is assigned to the pixels of the dynamic foreground and the value “0” is assigned to the pixels of the static background. Here a connected component is a set of pixels in which each pixel is connected to all other pixels in that set.

Even after the segmented dynamic foreground has been smoothed by the Gaussian filtering, a certain number of noisy or spurious regions still persist. The motion components are found by the Mahalanobis distance method which detects the motion based on the color discrepancies of the corresponding pixels in the subsequent frames. Sometimes the small changes in reflectance and illumination characteristics of other objects in the scene can lead to incorrect detections of the motion. One important property of these spurious regions is that their sizes are small compared to those of the real moving objects in the scene and hence can be removed by a size filter.

The connected components detected by the iterative connected-component labeling algorithm consist of components belonging to both the real moving objects in the scene and the unexpected noisy artifacts. In order to remove these noisy artifacts, a size filter is used based on the size property of these noisy artifacts. When all connected components have been found in the dynamic foreground, the size filter is used to suppress the noisy artifacts with relatively small size in terms of number of pixels.

The threshold value of the size filter can not be set to be large, since it will delete the components corresponding to the real moving objects. At the same time, the threshold can not be

set too small, which will retain too many noisy artifacts. Considering the different applications, the algorithm should be robust to different cases. In the experiments presented in this thesis, a threshold of 1/5 of the maximum component size in the dynamic foreground is used.

After the size filtering operation, only large components corresponding to the moving objects are retained. A bounding box which is composed of the minimum and maximum coordinates of a certain component in the frame is recorded in an information file for later transmission. The thresholded components are extracted from the original frames and used to generate a set of small image files which store only the pixels corresponding to regions in the bounding boxes.

2.1.4 Panorama Composition

When all the information has been transmitted from the server to the end user, it is used to reconstruct a motion panorama. The static background image and the dynamic foreground for each frame in the video sequence are now available. The dynamic foreground objects are then pasted onto the static background based on the parameters in the information file to reconstruct the motion panorama.

2.2 Parallel Computing

Two major reasons for using parallel computing are: to save time and to solve larger problems. Parallel computing also benefits from cost savings and overcoming memory constraints.

There exist several parallel programming models. The two most commonly used are the shared memory programming model and distributed memory programming model. Each model has its advantages and disadvantages. For example, the shared memory model is much easier to use because it does not require the user to specify explicitly the communication of data between

tasks. While at the same time, it ignores data locality/placement, which might sometimes have a negative impact on performance and scalability. On the other hand, the distributed memory model offers performance and scalability but it compromises the ease-of-use because the user is typically responsible for sending and receiving data. In addition, the distributed memory model might not scale well on problems that require a lot of data communication among tasks due to the fact that the complexity of communication overhead scales as $O(n^2)$.

Two standard implementation environments for the shared memory model and the distributed memory model are POSIX threads (Pthreads) and Message Passing Interface (MPI), respectively. In the next chapter, we provide the implementation details for the above two models.

CHAPTER 3

IMPLEMENTATION DETAILS

3.1 Homography Estimation

Figure 3.1 shows the CPU time distribution of the various tasks in the homography estimation process. As we can see, tasks such as Interest Point Detection, Correspondences Matching and RANSAC Estimation require the most computation. These three steps together account for 98% of the total execution time. Hence, an efficient parallel algorithm for each of these three steps is crucial to the final outcome. Meanwhile, Optimal Estimation requires little computation and the parallelization potential is limited. The remaining time is spent mostly on file I/O which, in the current implementation, is not parallelized.

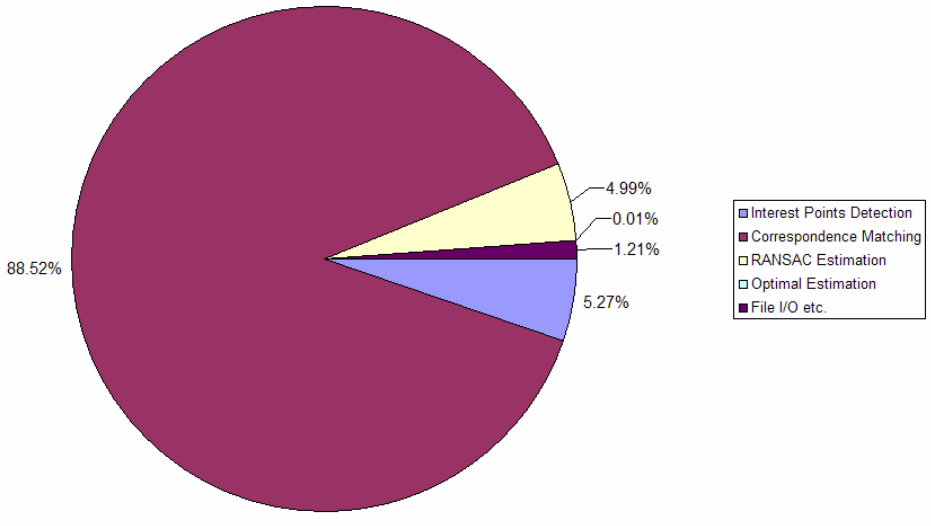


Figure 3.1: CPU time distribution for various tasks in homography estimation

3.1.1 Interest Point Detection

Since the interest value of a pixel only depends on the color values of neighboring pixels in the original frame, no data dependency exists between any two computations. The parallelization can be achieved by partitioning the original frame into several blocks, each with an almost equal number of pixels, and assigning one block to a processor for computation. Once the above process is completed, the resulting matrix of interest points is divided into a number of blocks. Each processor searches for the pixel that has the largest interest value within each neighboring and non-overlapping 30x30 window in its assigned block. These pixels are then marked as interest points.

In the shared memory parallel algorithm (SMPA), no synchronization is required during the whole process because the partition remains the same. The overhead for the SMPA, if any, is due to memory contention. The distributed memory parallel algorithm (DMPA) does not suffer from memory contention, however, it requires explicit communication among the nodes at the end of the computation so that all nodes can obtain a local copy of the matrix of interest points. The simplest approach is to communicate the entire matrix. However, because there is only one interest point within a 30x30 region, communicating the entire matrix is extremely inefficient. So in order to reduce the redundancy during the communication, an array of size $\frac{480*720}{30*30}*2$ is used in the communication instead of the entire matrix. Each node copies the coordinates of all interest points within its assigned block into the array before performing an all-to-all communication over the entire array. At the end of the communication, the array in each node, which contains the coordinates of all the interest points, is used to mark the local interest point matrix. As a result, the size of data used in the communication is reduced by $\frac{2}{30*30}=\frac{1}{450}$.

3.1.2 Searching for Correspondence

The next step is to search the next frame in the video sequence for the corresponding points. Again, there is no data dependence between any two distinct searches for the corresponding points. So the key issue for parallelization is how to evenly assign points to each processor so that the work loads are balanced among all processors. Our implementation is similar to a multiple server queue, where several servers simultaneously serve a group of clients. A client will wait in the queue until a server has finished its work. So all servers will be busy until each server finishes its job and there is no other client in the queue.

In the SMPA, we use a global cursor to identify the first interest point in the queue. When a processor is available, it first locks the cursor from being accessed by another processor. After it gets the information about the first interest point in the queue, it updates the cursor's position, releases the lock attached to the cursor and begins to search for the corresponding point in the next frame. If there is no interest point in the queue, the processor releases the lock attached to the cursor and waits for other processors to finish their jobs. The overhead incurred during this step is the time spent in synchronizing the value of the global cursor.

Another approach is to statically assign the interest points based on a certain criterion. For example, we can divide equally the interest points set and assign each subset to a processor. Such an approach can eliminate the synchronization overhead, but may also suffer from unbalanced load. This is because the computation required for each single search may vary and with static assignment, the worst case happens when all points that require less computation are assigned to one processor and all points that require extra computation are assigned to the other.

We have tested the performance for both dynamic and static assignments. In the case of static assignment, we employ the same criterion used in the DMPA, which will be discussed in the following section. The experimental results show that there is almost no difference when using 2 or 4 processors. When using 8 processors, our dynamic assignment mechanism performs on average 2-3% better than the static version in term of total execution time. As the number of processors increases, the chance for a more unbalanced situation also increases.

In the DMPA, we did not employ the above dynamic assignment mechanism. The reason is that if a "global" cursor is used, the overhead for the explicit communication over the cursor will be too costly. Any performance gain will most probably be offset by such a communication overhead. Hence, in the DMPA, the interest points are assigned based on certain criterion. In our implementation, we first number the interest points from 0 to *P*. The interest points with the numbers 0, N_n, $2N_n$, $3N_n$…, where N_n is the total number of nodes, are assigned to the node with the *id* 0, interest points with the numbers 1, N_n+1, $2N_n$+1, $3N_n$+1… are assigned to the node with the *id* 1, so on and so forth. As has been discussed, this static assignment may suffer from a load balancing problem. However, the experiment results provided in Chapter 4 show that the resulting performance is still quite satisfactory.

3.1.3 RANSAC Estimation

The RANSAC algorithm basically just repeats a sampling process for a number of iterations. Each sampling process is independent. Also, the computation required by each process is almost the same. Therefore, the step can be easily parallelized by sharing the total sampling iterations among all the processors.

Regardless of which parallel algorithm is used, the total number of sampling iterations is divided by the total number of processors. The result is the number sampling iterations a processor should repeat. If there is a residual R (where $R > 0$), processors with an id from 0 to R-1 will perform an additional iteration.

Each processor will compute the homography from the local consensus set with the largest number of inliers (see Section 2.1.1). The local largest number of inliers will then be compared at the end and the homography computed from the largest number of inliers among all the processors will be selected. In the SMPA, this is done by using a mutex_lock() & mutex_unlock() so that the homography and the largest number of inliers are updated by each processor one by one. In the DMPA, all child nodes send their local copies of homographies along with the largest number of inliers to the master node. The master node computes the result based on the data it receives.

3.1.4 Pseudo Code for Homography Estimation.

SMPA:

1: **for all** rows from $i = \frac{pid * ROW}{N_p}$ to $\frac{(pid+1) * ROW}{N_p}$ **do**

2: **for all** columns from $j = 0$ to COL **do**

3: calculate the interest value of pixel (i, j) in a frame

4: **end for**

5: **end for**

6: **for all** rows from $i = \frac{pid * ROW}{N_p}$ to $\frac{(pid+1) * ROW}{N_p}$ with $step = 30$ **do**

7: **for all** columns from $j = 0$ to COL with $step = 30$ **do**

8: find the pixel with maximum interest value within a 30x30 window and mark it as interest point

9: **end for**

10: **end for**

11: barrier()

12: global cursor $C = 0$, total interest point $P = \frac{ROW * COL}{30 * 30}$

13: **for all** interest points from $i = 1$ to P **do**

14: mutex_lock()

15: **if** $i > C$ **then**

16: $C = i$

17: mutex_unlock()

18: search for the corresponding point of interest point *i*

19: **else**

20: mutex_unlock()

21: **continue**

22: **end if**

23: **end for**

24: barrier()

25: $samples = \frac{5000}{N_p}$

26: $residual = 5000\%N_p$

27: **if** $pid < residual$ **then**

28: $samples = samples + 1$

29: **end if**

30: **for all** i from $i = 1$ to samples **do**

31: randomly select 4 pairs of corresponding points and estimate the homography H based on these points, use the homography to calculate the number of inliers in

32: **if** $in > in_{pid}$ **then**

33: $in_{pid} = in$

34: $h_{pid} = h$

35: **end if**

36: **end for**

37: $in_{\max} = 0$

38: mutex_lock()

39: **if** $in_{pid} > in_{\max}$ **then**

40: $in_{\max} = in_{pid}$

41: $H = h_{pid}$

42: **end if**

43: mutex_unlock()

44: barrier()

45: processor 0 runs Optimal Estimation

DMPA:

1:

… Same as line 1-10 in SMPA

10:

11: copy the coordinates of all interest points into an array

12: MPI_Allgather()

13: convert the array back to interest point matrix

14: **for all** interest points from $i = 1$ to P **do**

15: **if** $i\%N_n = nid$ **then**

16: search for the corresponding point of interest point *i*

17: **else**

18: **continue**

19: **end if**

20: **end for**

21: MPI_Allgather()

22:

… Same as line 25-37 in SMPA

35

36: **if** $nid == 0$ **then**

37: **for all** $i = 1$ to $N_n - 1$ **do**

receive in_{nid} and h_{nid} from node *i*

38: **end for**

39: **end if**

40: master node runs Optimal Estimation

3.2 Background Mosaic Generation

Background mosaic generation consists of two computations: estimating the homographies for a series of frames in the video sequence and mapping those frames onto the final static background mosaic using the estimated homographies. This task is parallelized by first running the parallel algorithm for homography estimation on one frame and then using only one processor to execute the mapping process. This procedure is repeated on a series of frames in the video sequence.

3.3 Dynamic Foreground Segmentation

Figure 3.2 shows the CPU time distribution for the various steps in dynamic foreground segmentation. The processes of generating the probability image and then performing Gaussian filtering on the probability image together take up over 99% of the total execution time. Our works focuses mainly on the parallelization of these two tasks.

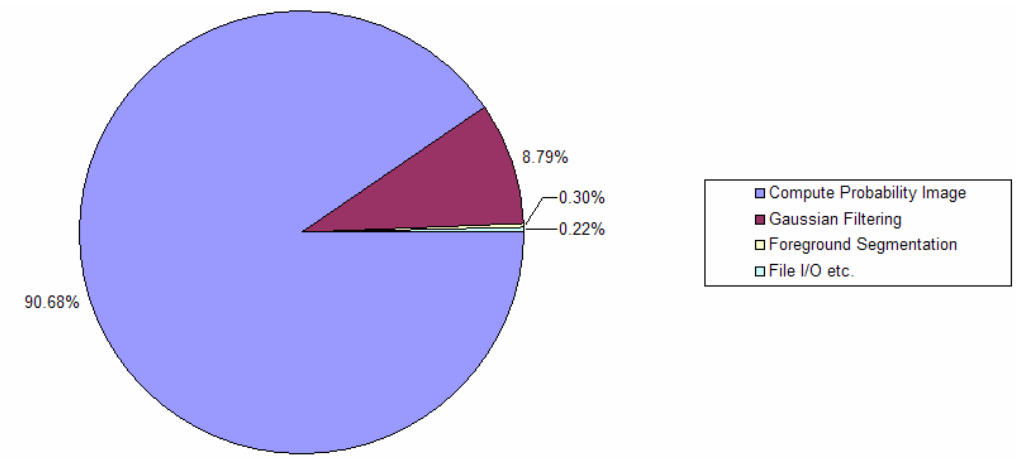


Figure 3.2: CPU time distribution for various tasks in dynamic foreground segmentation

3.3.1 Computation of the Probability Image

Two major steps in this task are: estimating the homography between two consecutive frames and computing the Mahalanobis distance matrix for all pixels using equation (2.5). We use the same parallel algorithm for homography estimation as the one used in static background mosaic generation. To parallelize the computation of the Mahalanobis distance matrix, we use the technique that partitions the matrix into a several blocks and shares the computation among all processors. This matrix computation depends only on the estimated homography and the color values of all pixels in a single frame. So once the homography is obtained, no more data synchronization or communication is required.

3.3.2 Gaussian Filtering of the Probability Image

A two-dimensional Gaussian filter is applied on the probability image. When applying the filter along the x-axis, the value of each pixel is updated based on the values of its neighbors on the same row. This is illustrated in Figure 3.3(a) and 3.3(b). For example, the value of the blue pixel in (b) is updated based on the values of all blue pixels in (a). On the other hand, when applying the filter along the y-axis, the value of each pixel is updated based on the values of its neighbors in the same column, as shown in Figure 3.3(c) and 3.3(d).

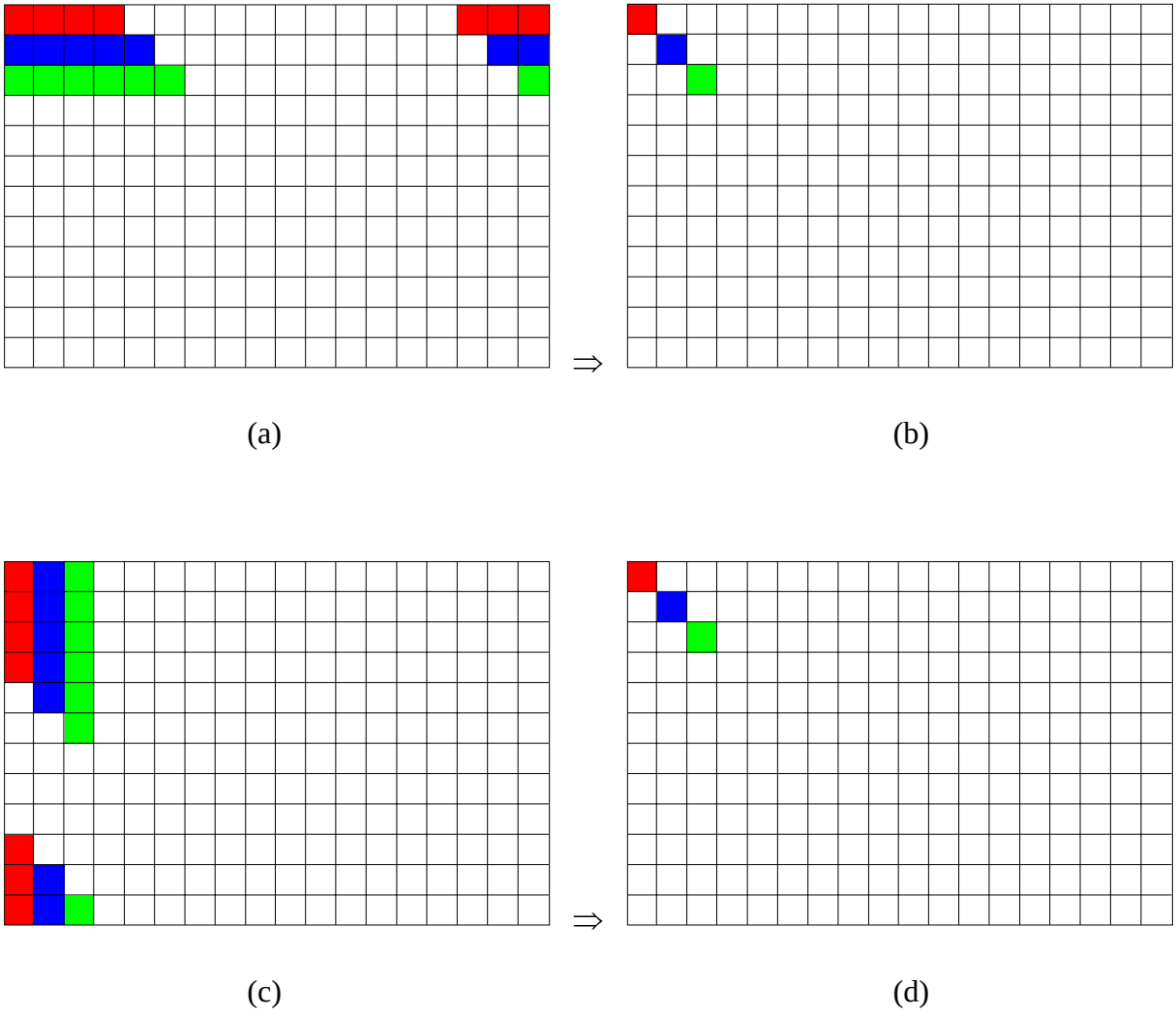


Figure 3.3: Data dependence pattern

In the SMPA, the changes in data dependence pattern require the data to be synchronized before a different filter is applied. This can be easily done by placing a barrier() before applying the Gaussian filter along the second dimension. In the DMPA, the need for explicit communications among all processors hurt the performance badly. Our original implementation uses MPI_Allgather() to communicate the entire probability image matrix. This approach is

proven to be very inefficient. In Section 4.3, we provide an optimized version of the communication process.

After the probability image has been filtered by the Gaussian filter masks along each dimension, it is partitioned into a several blocks such that each processor works on only one block for the remainder of the computation.

3.3.3 Pseudo Code for Foreground Segmentation

SMPA:

1: estimate the homography H_p between the current and the previous frame

2: **for all** rows from $i = \frac{pid * ROW}{N_p}$ to $\frac{(pid+1) * ROW}{N_p}$ **do**

3: **for all** columns from $j = 0$ to COL **do**

4: compute the Mahalanobis Distance matrix MD_p for pixel(i, j) based on H_p

5: **end for**

6: **end for**

7: estimate the homography H_n between the current and the next frame

8: **for all** rows from $i = \frac{pid * ROW}{N_p}$ to $\frac{(pid+1) * ROW}{N_p}$ **do**

9: **for all** columns from $j = 0$ to COL **do**

10: compute the Mahalanobis Distance matrix MD_n for pixel(i, j) based on H_n

11: **end for**

12: **end for**

13: the probability image is calculated as the sum of MD_p and MD_n

14: **for all** rows from $i = \frac{pid * ROW}{N_p}$ to $\frac{(pid+1) * ROW}{N_p}$ **do**

15: **for all** columns from $j = 0$ to COL **do**

16: apply x-mask of the Gaussian filter

17: **end for**

18: **end for**

19: barrier()

20: **for all** rows from $i = \frac{pid * ROW}{N_p}$ to $\frac{(pid+1) * ROW}{N_p}$ **do**

21: **for all** columns from $j = 0$ to COL **do**

22: apply y-mask of the Gaussian filter

23: **end for**

24: **end for**

25: **for all** rows from $i = \frac{pid * ROW}{N_p}$ to $\frac{(pid+1) * ROW}{N_p}$ **do**

26: **for all** columns from $j = 0$ to COL **do**

27: find the maximum value md_{pid} of pixel (*i*, *j*) in the filtered probability image

28: **end for**

29: **end for**

30: $md_{\max} = 0$

31: mutex_lock()

32: if $md_{pid} > md_{\max}$ then

33: $md_{\max} = md_{pid}$

34: end if

35: mutex_unlock

36: barrier()

37: $md_{\max} = md_{\max} / 5$

38: **for all** rows from $i = \frac{pid * ROW}{N_p}$ to $\frac{(pid+1) * ROW}{N_p}$ **do**

39: **for all** columns from $j = 0$ to COL **do**

40: mark the pixel whose value is greater than the threshold as foreground

41: **end for**

42: **end for**

43: processor 0 runs the dynamic component labeling and segmentation

DMPA:

1:

… Same as line 1-19 in SMPA

19:

14: **for all** rows from $i = \frac{nid * ROW}{N_n}$ to $\frac{(nid+1) * ROW}{N_n}$ **do**

15: **for all** columns from $j = 0$ to COL **do**

16: apply x-mask of the Gaussian filter

17: **end for**

18: **end for**

19: MPI_Allgather()

20: **for all** rows from $i = \frac{nid * ROW}{N_n}$ to $\frac{(nid+1) * ROW}{N_n}$ **do**

21: **for all** columns from $j = 0$ to COL **do**

22: apply y-mask of the Gaussian filter

23: **end for**

24: **end for**

25: **for all** rows from $i = \frac{nid * ROW}{N_n}$ to $\frac{(nid+1) * ROW}{N_n}$ **do**

26: **for all** columns from $j = 0$ to COL **do**

27: find the maximum value of pixel(i, j) in the final probability image

28: **end for**

29: **end for**

30: MPI_Reduce()

31: **for all** rows from $i = \frac{nid * ROW}{N_n}$ to $\frac{(nid+1) * ROW}{N_n}$ **do**

32: **for all** columns from $j = 0$ to COL **do**

33: mark the pixel whose value is greater than the threshold as foreground

34: **end for**

35: **end for**

36: node 0 runs the foreground segmentation

CHAPTER 4

PERFORMANCE ANALYSIS

In this section, we discuss our experimental results for the shared memory and distributed memory parallel algorithms. The results for the SMPA were obtained using a Sun Fire 880 server with 8 UltraSPARC III 750MHz processors, where each processor has 8MB internal cache and shares a total of 16GB of global memory. On the other hand, results for the DMPA were gathered from a cluster of 16 Sun Blade 1500 workstations, each equipped with an UltraSPARC IIIi 1.05Ghz processor, 1MB Cache and 1GB local memory.

We measure the speedup for each parallelizable task during static background generation and dynamic foreground segmentation. As mentioned before, the final panorama composition does not involve intensive computation and is intended to be run on the user end. Therefore, it is not discussed in this chapter. In addition, the overall performance which includes both static background generation and dynamic foreground segmentation is included for comparison. In the SMPA, the total number of processors used in the test ranges from 2 to 8, covering each power of 2. In the DMPA, we used the maximum available 16 processors. All tests were run three times and the best results were selected. The input video sequence consists of 14 480x720 image frames. Table 4.1 shows the problem size of each parallelizable task. Note that the problem size does relate to the number of input frames, it only relates to the dimensions of a single frame.

Interest Points Detection	480 x 720 x 49
Correspondence Matching	16 x 24 x O(finding one correspondence)
RANSAC Estimation	5000 x O(estimating one homography)
Compute Probability Image	O(homography estimation) + 480 x 720
Gaussian Filter	480 x 720 x 48

Table 4.1: Problem size of each parallelizable task

4.1 Performance of the SMPA

Figure 4.1 shows the SMPA speedup results. It is quite obvious that SMPA performs very well. The overall CPU efficiency is shown in Table 4.2. Such a high efficiency is mostly due to the fact that motion panorama construction algorithm does not require too many data communication operations. The shared memory architecture with a high bandwidth memory bus helps to reduce the time for data synchronization.

In both Correspondence Matching and RANSAC Estimation, the dynamic work load distribution technique has proven to be extremely effective in terms of load balancing. Both tasks achieve nearly 100% CPU efficiency when using 4 or fewer processors. While using 8 processors, we were unable to achieve a consistent result over a number of the tests which were taken during different time period. The reason is because we had used up all processors that are available and there were always some jobs running on at least one processor. The speedup result for 8 processors shown in Figure 4.1 is the best result we obtained.

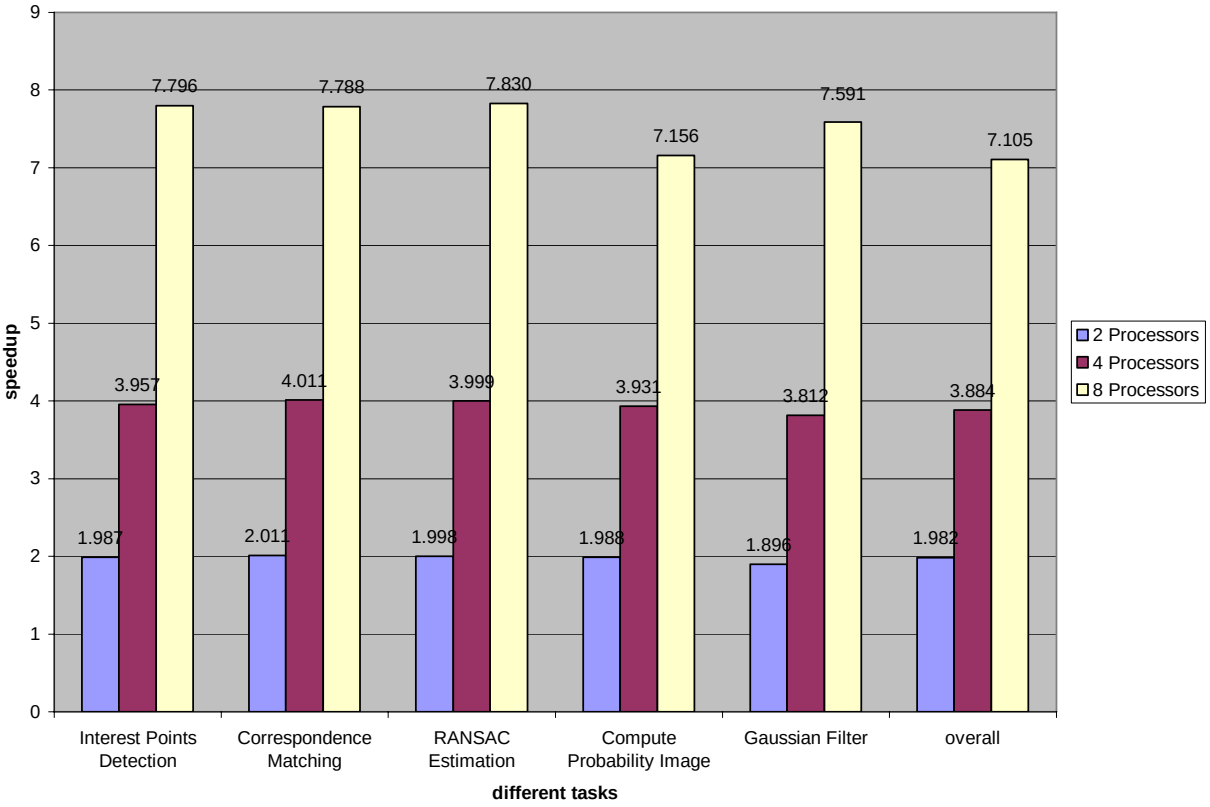
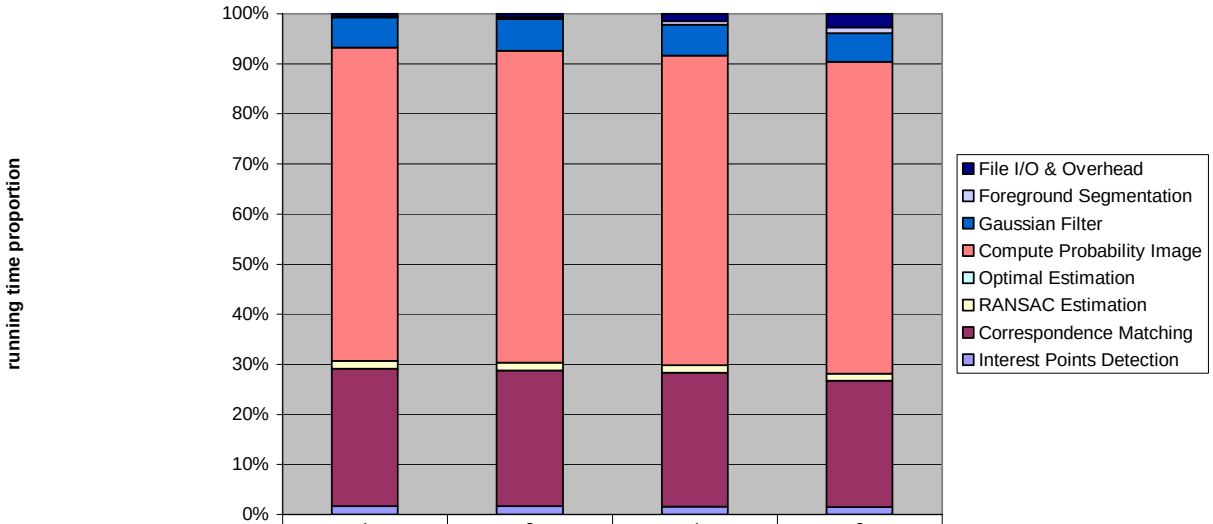


Figure 4.1: Speedup results for the SMPA

Number of processors	2	4	8
CPU efficiency	99.1%	97.1%	88.8%

Table 4.2: Overall CPU efficiency for the SMPA with different number of processors

Figure 4.2 shows the proportion of the total execution time taken by each task. As we can see, the time spent in both file I/O and the communication overhead does not increase much as the number of processors increases, thus proving that the parallel algorithm does very well in balancing the load. However, the proportion of total execution time taken by non-parallelizable tasks such as file I/O and dynamic component segmentation increases with an increasing number of processors. With 8 processors, these two tasks take up around 5% of the total execution time. From the data tabulated in Figure 4.2, we predict that the SMPA will work very well on at least 16 processors.



	1	2	4	8
File I/O & Overhead	4139.05	2788.94	3054.81	3014.33
Foreground Segmentation	1616.71	1269.55	1325.41	1258.04
Gaussian Filter	47388.16	24992.93	12432.14	6242.37
Compute Probability Image	488743.79	245887.43	124336.93	68301.79
Optimal Estimation	12.89	12.97	9.75	11.27
RANSAC Estimation	12142.26	6077.42	3036.5	1550.82
Correspondence Matching	215191.71	107018.74	53650.67	27630.84
Interest Points Detection	12814.97	6448.66	3238.35	1643.72

number of processors

Figure 4.2: Proportion of the total execution time taken by various tasks in the SMPA

4.2 Performance of the DMPA

The DMPA for motion panorama construction does not work as well as the SMPA. This is mostly due to the overhead arising from explicit data communication among processors. The parallel algorithm does work well on the first three tasks, where little communication is required. The overall CPU efficiency is shown in Table 4.3. It is quite obvious that the $O(n^2)$ communication overhead has a significant negative impact on the performance of the DMPA. The extreme case occurs during the process of Gaussian filtering, where the speedup for 8 processors exceeds the speedup for 16 processors. With 16 processors, the DMPA achieves a CPU efficiency of only 62%.

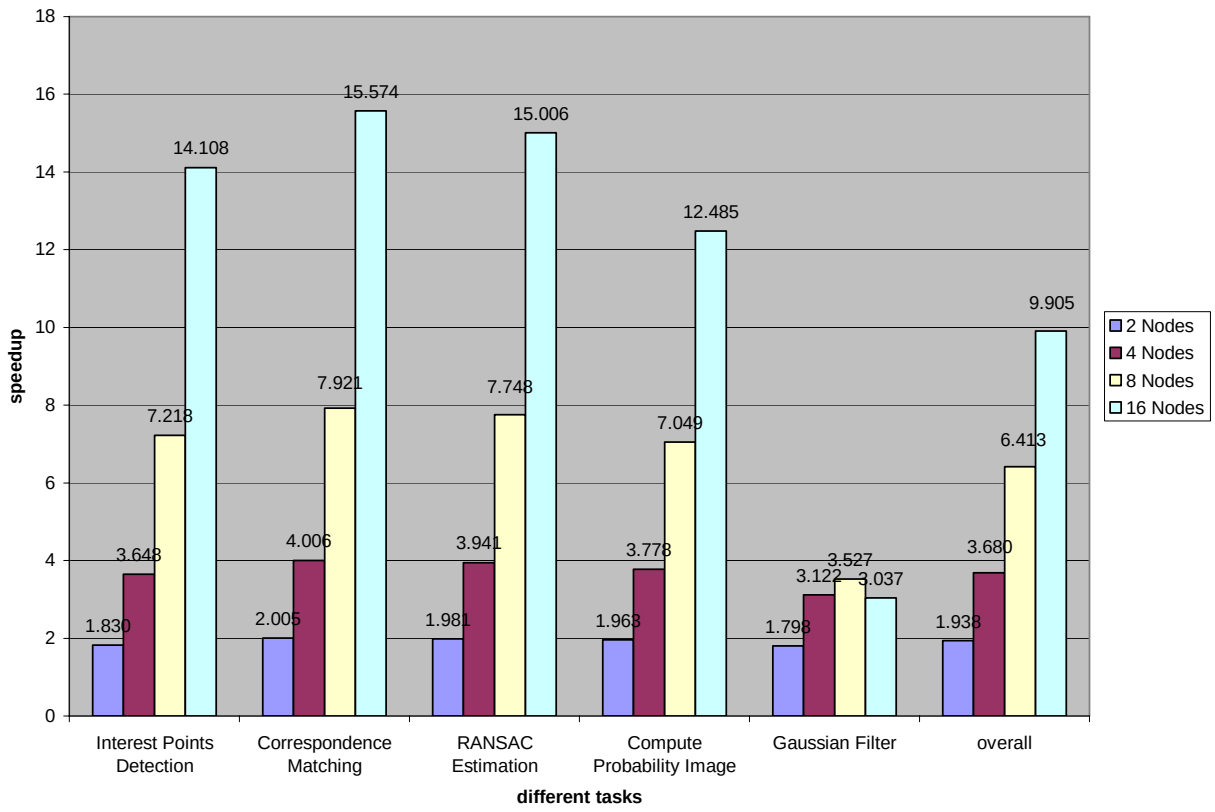
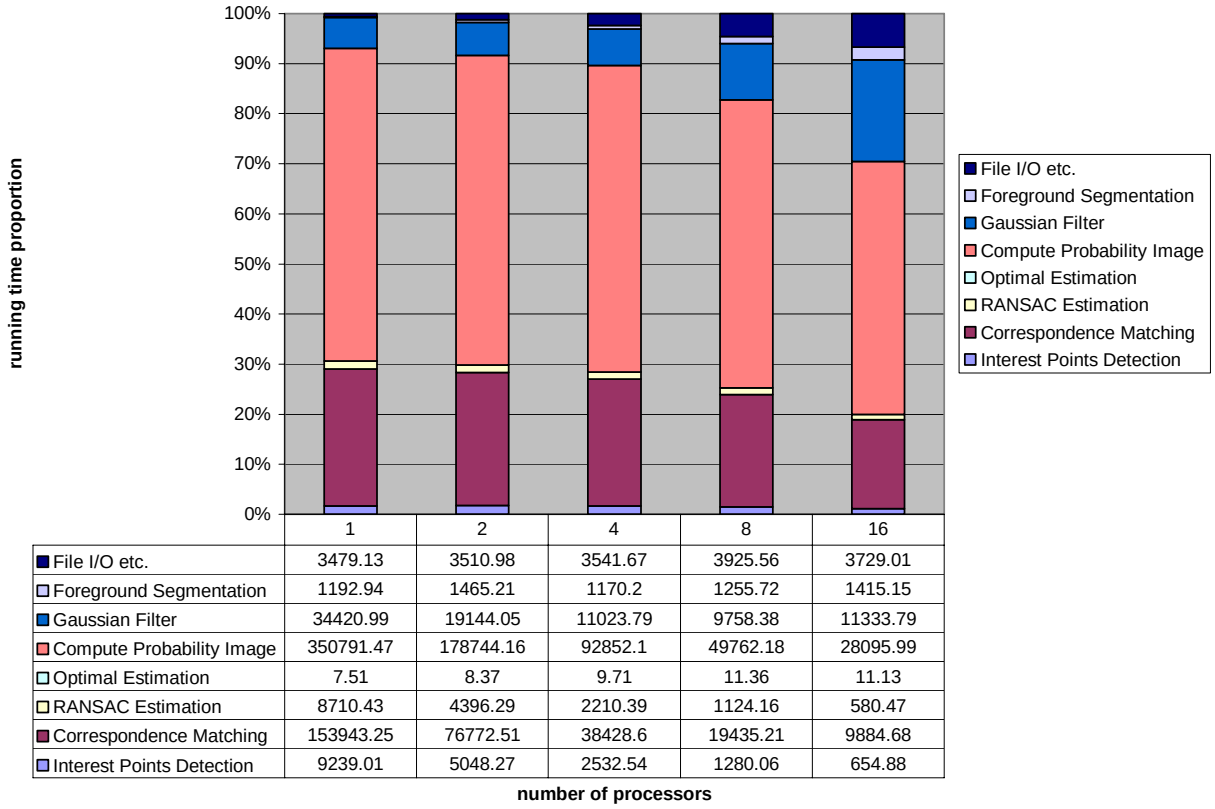


Figure 4.3: Speedup results for the DMPA

Number of processors	2	4	8	16
CPU efficiency	96.9%	92%	80.2%	61.9%

Table 4.3: Overall CPU efficiency for the DMPA with different number of processors



	1	2	4	8	16
File I/O etc.	3479.13	3510.98	3541.67	3925.56	3729.01
Foreground Segmentation	1192.94	1465.21	1170.2	1255.72	1415.15
Gaussian Filter	34420.99	19144.05	11023.79	9758.38	11333.79
Compute Probability Image	350791.47	178744.16	92852.1	49762.18	28095.99
Optimal Estimation	7.51	8.37	9.71	11.36	11.13
RANSAC Estimation	8710.43	4396.29	2210.39	1124.16	580.47
Correspondence Matching	153943.25	76772.51	38428.6	19435.21	9884.68
Interest Points Detection	9239.01	5048.27	2532.54	1280.06	654.88

Figure 4.4: Proportion of the total execution time taken by various tasks in the DMPA

From Figure 4.4, we can see that file I/O and Gaussian filtering have a significant negative impact on the performance of the DMPA. Optimization of the file I/O is beyond the scope of this thesis. In the next section, we describe the optimization for Gaussian filtering in the DMPA.

4.3 Optimization

For reason of simplicity, we used MPI_Allgather() in the DMPA implementation of the Gaussian filtering when the direction of data dependences changes from along the rows to along the columns. However, only 2 x 48 rows of data for each node are actually useful, whereas a total $480 - \frac{480}{NODES} - 2 \times 48$ rows of data per node during the communication are redundant. To eliminate this redundancy, we use the following technique:

- Each node sends 48 rows of data to its upper and lower neighbors. The ‘upper’ neighbor for the node 0 is the last node, and the ‘lower’ neighbor for the last node is node 0.
- If a node has less than 48 rows of data, for example, when using 16 nodes, each will have only 45. In this case, 2 or more nodes will be involved in sending these 48 rows data to the nodes upper and below.
- All sending and receiving will be non-blocking. This process is blocked until all receiving are completed.

In the original implementation, MPI_Allgather() is used at the end of the Gaussian filtering operation so that each processor will have a local copy of the filtered probability image. However, this is unnecessary because only the master node performs the dynamic foreground segmentation. Therefore, we further optimize the algorithm by only gathering the data in the

master node. Figure 4.5 shows a significant improvement in reduction of the communication overhead, and Figure 4.6 shows a noticeable improvement in overall performance.

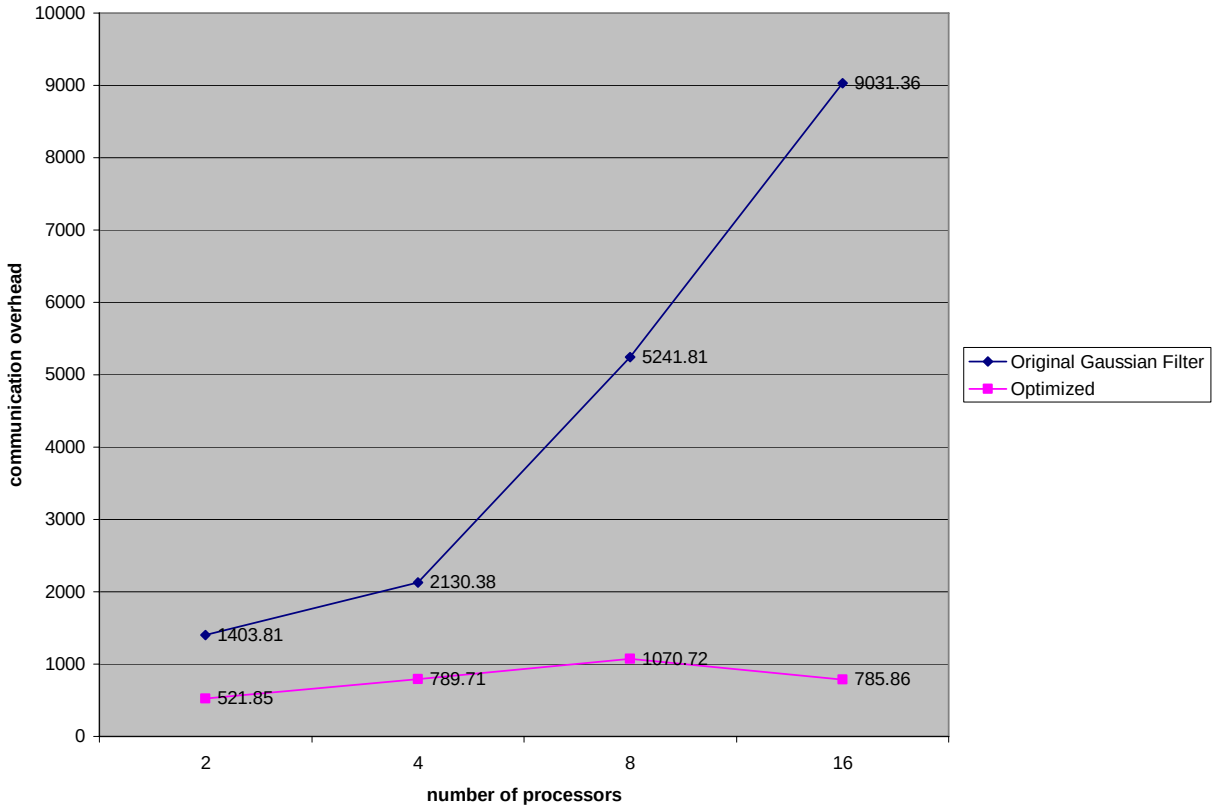


Figure 4.5: Communication overhead for original and optimized Gaussian filtering

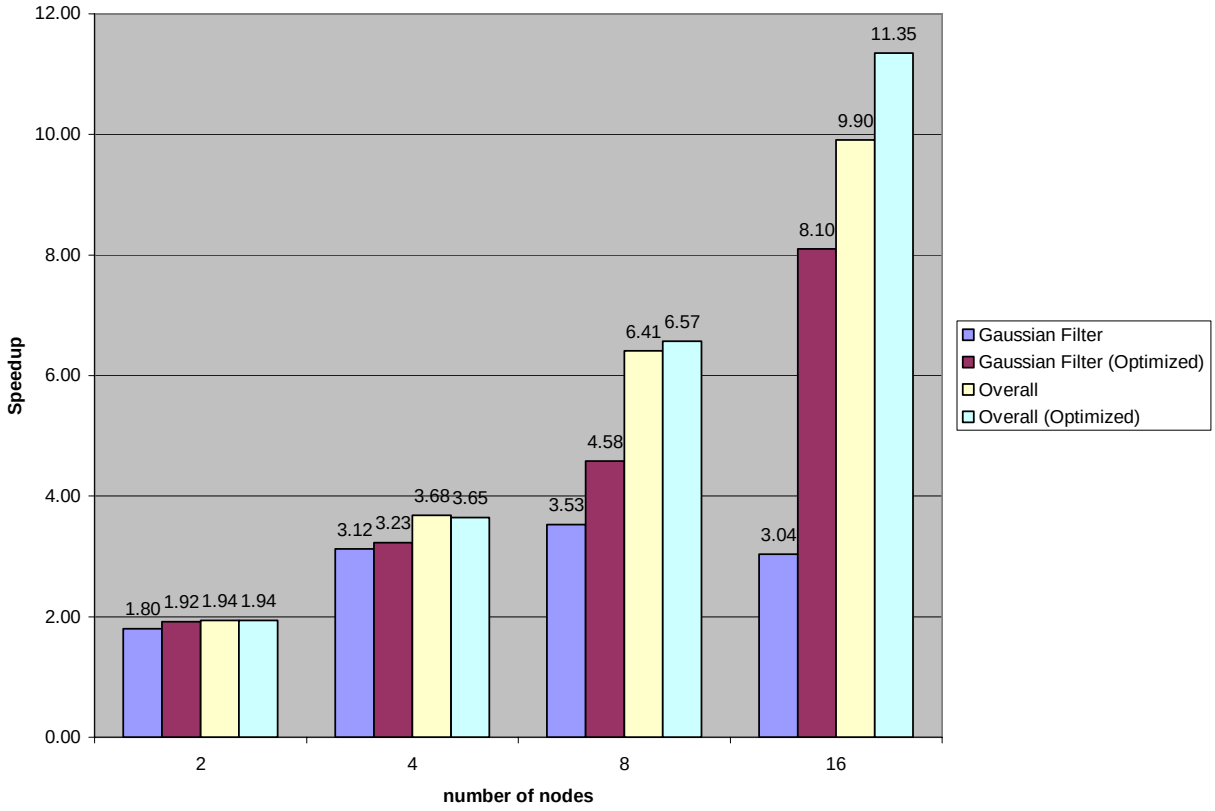


Figure 4.6: Performance comparison for original and optimized Gaussian filtering

CHAPTER 5

SUMMARY

Motion panorama construction is commonly viewed as an efficient representation for a video sequence. Pan has described an algorithm for motion panorama construction. Parallelization is proposed in this thesis to speed up the motion panorama construction process in order to make it suitable for real time applications.

In the previous chapters, we have compared two widely used parallel algorithms, i.e., the SMPA and the DMPA, for motion panorama construction. In the SMPA, we proposed a dynamic work load distribution mechanism which leads to a very well balanced algorithm. A good speedup result is achieved on 8 processors. Based on the results from 2 and 4 processors, we predict that the SMPA will continue to work well on 16 processors. In the case of the DMPA, we provide an optimized scheme for data communication when the Gaussian filter is applied on the probability image, which leads to a noticeable improvement in terms of overall performance of the DMPA.

We have also tried to implement a hybrid parallel algorithm, which works on a cluster that has a combination of both shared memory and distributed memory architectures. Our idea is to first parallelize the computation as if it were being implemented on a distributed memory architecture consisting of the nodes in the cluster, and then within each node, have all the processors share equally the work that has been assigned to the node. However, this approach is somewhat immature and the results so far are discouraging. Future work can be considered in

this direction. An idea worth pursuing is to employ a more sophisticated computation distribution mechanism so that each node can work on an individual frame independently and each processor within a node can share the computation being performed on this frame. In this way, the communication overhead amongst the nodes can be minimized and the proportion of the total execution time taken by file I/O can be minimized as well.

REFERENCES

[1] M. Irani, P. Anandan, J. Bergen, R. Kumar, and S. Hsu, Efficient representations of video sequences and their applications, Signal Processing: Image Communication, special issue on Image and Video Semantics: Processing, Analysis, and Application, 8(4), pages 327-351, May 1996.

[2] H. Y. Shum and R. Szeliski, Panoramic image mosaics, Microsoft Research Technical Report, vol. MSR-TR-97-23, 1997.

[3] M. Irani, S. Hsu, and P. Anandan, Mosaic based video compression, In Proceedings of SPIE Conference on Electronic Imaging, Digital Video Compression: Algorithms and Technologies, volume 2419, pages 242-253, February 1995.

[4] S. Hsu and P. Anandan, Hierarchical Representations for Mosaic Based Video Compression, Proc. Picture Coding Symp., pp. 395-400, Mar. 1996.

[5] S. E. Chen, QuickTime VR - An Image-based Approach to Virtual Environment Navigation, In Proc. SIGGRAPH '95, pp. 29-38.

[6] W. H. Leung and T. Chen. Compression with Mosaic Prediction for Image-Based Rendering Applications, IEEE International Conference on Multimedia and Expo (ICME 2000), Vol. 3, pp.1649-1652, New York, July 2000.

[7] L. Teodosio and W. Bender, Salient video stills: Content and context preserved, In Proceedings of the First ACM International Conference on Multimedia, pages 39-46, 1993.

[8] M. Irani and P. Anandan, About direct methods, In Proc. of the International Workshop on Vision Algorithms: Theory and practice, pages 267-277, July 1999.

[9] P.H.S. Torr and A. Zisserman, Feature Based Methods for Structure and Motion Estimation, In Proc. of the International Workshop on Vision Algorithms: Theory and practice, pages 278-294, 1999.

[10] M. Irani and S. Peleg, Motion analysis for image enhancement: Resolution, occlusion, and transparency, Journal of Visual Communication and Image Representation, vol. 4, pp. 324–335, Dec. 1993.

[11] DARTFISH Ltd, DartTrainer Software, 2001/2002. http://www.dartfish.com/.

[12] J.Y.A. Wang and E.H. Adelson. Representing moving images with layers, IEEE Transactions on Image Processing Special Issue: Image Sequence Compression, vol. 3, no. 5, pp 625-638, September 1994.

[13] P.C. McLean. Structured video coding, MSc thesis, MIT, June 1991.

[14] A. Bartoli, N. Dalal, B. Bose, and R. Horaud. From video sequences to motion panoramas. In IEEE Workshop on Motion and Video Computing, pages 201- 207, December 2002.

[15] M. Irani and S. Peleg. Improving resolution by image registration, CVGIP: Graphical Models and Image Processing, 53:231-239, May 1991.

[16] Salient Stills Inc, VideoFOCUS Software, http://www.salientstills.com/.

[17] X. Y. Pan, Motion panorama construction from streaming video for power-constrained mobile multimedia environments, MSAI Thesis, the University of Georgia. 2004.

[18] H.P. Moravec, Towards Automatic Visual Obstacle Avoidance, In Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI '77), Cambridge, MA, USA, Page 584, 1977.

[19] Ramesh Jain, Rangacher Kasturi and Brian Schunck, Machine Vision, MIT Press and McGraw-Hill, March 1995.

[20] D.C. Alexander and B.F. Buxton, Statistical Modeling of Color Data, International Journal of Computer Vision 44 (2):87-109, September 2001.