5.1 INTRODUCTION

There has been a tremendous increase in the demand for digital video imagery due to the applications like consumer electronics, medical imaging, video-conferencing, telemedicine and scientific visualization. The main problem with the uncompressed (raw) video is that it contains immense amount of data and hence communication and storage capabilities are limited and are expensive. Large amount of bandwidth is required for transmission or storage of video signals. For example, High-definition motion picture (Frame size 1980x1080) requires a data rate greater than 1.49 Gigabits / second as in the following calculation (Wang et al 2002).

\[
\left( \frac{1920 \times 1080 \text{pixels}}{\text{frame}} \right) \times \left( \frac{3 \text{colours}}{\text{pixel}} \right) \times \left( \frac{8 \text{bits}}{\text{colour}} \right) \times \left( \frac{30 \text{frames}}{\text{second}} \right) = 1.49 \text{Gb / second}
\]  

(5.1)

But the available bandwidth in HDTV channel is very limited and can transmit only up to 40 Mb per second. Likewise a 4.7 Gigabyte DVD could store only 87 seconds of this uncompressed video information (Wang 2002). Hence, this problem of large data has driven the research area of video compression to develop algorithms that compress images to lower data rates with better fidelity. The major irony to come out of video compression research is that as the data rates come down, the computational complexity of the algorithms increases. This leads to the problem of long execution time to compress an image sequence. For example, Philips Corporation has reported
that it takes 80 minutes of computer time to compress 1 minute of a motion picture (using MPEG1) for its CD-I Video System (Shen et al 1994). This is unacceptable for real time coding or the task of the compression of large image libraries such as a motion picture library. In a typical video compression system, the compression tasks are computationally intensive, mainly due to the large amount of data to be processed and the time consuming repetitive operation of the processing algorithms.

Hence in order to transmit, store or display real time video, some scheme for fast video compression is necessary. The need for speed in video compression has driven to many hardware approaches that increase execution speed of compression algorithm just by increasing the processor speed (Yamauchi et al 1992). Unfortunately, many compression applications demand execution times that are not possible using a single serial processor and the computational demand for video compression is always far above the capacity of conventional sequential processing. It is apparent, then, that in order to transmit, store or display real time video or large volumes of high quality still images, some scheme for fast video and image data compression using parallel processing is necessary (Shen et al 1994, Nicolescu and Jonker 2001).

5.2 PARALLEL MOTION ESTIMATION APPROACHES

In this section, an overview of parallel motion estimation techniques and a study of parallel image processing methods for video compression are presented. Many parallel approaches have been identified in the literature and these can largely be divided into four areas.

a) Hardware approaches
b) Software approaches
c) Optical Image Processing Method
d) Distributed Computing approaches

The second approach is the algorithm driven, in which the structure of the compression algorithm describes the parallel execution (Nicolescu and Jonker 2001, Seinstra et al 2002, David et al 2003). Nicolescu and Jonker (2001) have dealt with the software approach of using algorithmic skeletons for adding data parallelism to an image processing library. Two more approaches suggested in this thesis are Optical image processing and Distributed computing. Optical image processing method uses lenses and other optical devices for performing image operations (Takahiro Manabe et al 2004, Bhattacharyya 2007). The advantage of this method is ultra speed as the entire image is processed at once by the light. Anne Cottour et al (2008) have described an optical video image compression using a multiplexing method based on the spectral fusion of information. The other approach is implementing the image processing operations using distributed cluster computing (Timo Kohlberger et al 2004, Clematis et al 2005, Yunsong Wu and Graham Megson 2006). A cluster has been proven an efficient parallel computing platform that uses a group of computer resources to improve the performance and availability of a single computer resource. In this work cluster computing has been used for parallel implementation of motion estimation.
5.3 CLUSTER COMPUTING ARCHITECTURE

Recent advances in the development of high performance microprocessors and high speed networks make clusters of PCs an attractive low cost platform for parallel computing. Cluster Computing, using commodity-off-the-shelf hardware components as well as free software is playing a major role in the field of supercomputing. The main advantages of Cluster Technology are low cost commodity supercomputing and scalability (Petersen and Arbenz 2004). A cluster is a collection of complete computers (nodes) interconnected by a high speed network typically, each node is a workstation, PC, or symmetric multiprocessor (SMP). Clusters are being used for climate modeling, synthetic aperture radar processing, environment modeling, astronomy database, space-height mission simulation and image processing.

In a typical time-sharing cluster system, resources must be shared among users, and the contention that results from this sharing causes the deliverable performance to vary over time. To make the best use of the resources that are at hand, the resource monitoring program is required (Teddy Surya Gunawan and Cai Wen Tong 2002). Furthermore, the resource information can be used by any parallel applications for load distribution purpose.

Pfister (1998) has defined that a cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers working together as a single, integrated computing resource. A computer node can be a single or multiprocessor system (PCs, workstations, or SMPs) with memory, I/O facilities, and an operating system. A cluster generally refers to two or more computers (nodes) connected together in a single cabinet or be physically separated and connected via a LAN. An interconnected (LAN-based) cluster of computers can appear as a single system to users and applications. Such a system can provide a cost-effective way to gain features and benefits (fast and reliable services) that
have historically been found only on more expensive proprietary shared memory systems. The typical architecture of a cluster is shown in Figure 5.1.

![Figure 5.1 Architecture of Cluster Computing](image)

**Figure 5.1 Architecture of Cluster Computing**

Following are the major components of cluster computers:

- Multiple Computers (PCs, Workstations, or SMPs)
- Operating Systems
- High Performance Networks / Switches
- Network Interface Cards
- Cluster Middleware (Single System Image)
- Applications and Subsystems
- Resource Management and Scheduling software
- Parallel Programming Environments and Tools
- MPI (Message Passing Interface)
- Parallel / Distributed Applications

The cluster nodes can work collectively, as an integrated computing resource, or they can operate as individual computers. The cluster middleware
is responsible for offering an illusion of a unified system image (single system image) and availability out of a collection on independent but interconnected computers. Programming environments can offer portable, efficient, and easy-to-use tools for development of applications. They include message passing libraries, debuggers, and profilers. It should be noted that clusters could be used for the execution of sequential or parallel applications.

5.4 IMPLEMENTATION OF PARALLEL MODEL

In this research work, novel parallel image processing approach that applies distributed client-server computing concept has been introduced. This technique uses the power of local computer network with master-slave concept (Kambiz et al 2002, Yunsong Wu and Graham Megson 2006). The environment consists of a server with a number of workstations. A simple master-slave computing model is shown in the Figure 5.2.

![Master-Slave model](image)

**Figure 5.2 Master-Slave model**

This is a simple approach which distributes independent, non-overlapping image blocks on a multi, single processor cluster of workstations, using Message Passing Interface mechanism. The master-slave concept is the standard approach, in which the master sends the data to a slave and the slave sends back the output after computation. The system consists of a master server and many slave processors. The master server controls synchronization of all processes, assigns slave processes which blocks to process and notifies
the output server (same machine or another client) which blocks are done. The output server combines output blocks to an output file. The slave processors perform motion estimation process, output results and notify master process which and when the assigned blocks are done. Static data scheduling and First Serve First Finish (FSFF) assignment approach are used for load balancing in the cluster. The block diagram of the proposed parallel implementation of motion vector estimation and motion vector approximation is shown in Figure 5.3.

Figure 5.3 Parallel motion estimation model
The algorithm and steps for the parallel work are as below.

a. Maintain a collaborative memory in the master server to keep the video sequences as a queue.

b. By default, the first and last frames are considered as key frames and compressed using spatial predictor method before the parallel run.

c. Server then distributes the frames among all the processors in the cluster of workstations sequentially.

d. Each slave then proceeds through its assigned frame, performs the temporal prediction.

e. After finishing up, the slave sends message to master server and it writes 1 to the completion register of the respective frames.

f. When the completion register of specific workstation becomes 1, the processed frame is sent to the output server from the slave processor.

g. The master server computes the next chunk of frames for this slave to compress, encode a message and send it to the slave.

h. The process is repeated and the server will remain in the waiting state till the completion of all slave processes.

The experiments were done on a cluster architecture consisting of 1 server 4 nodes:

- Server - HP Proliant Server MI 350
- Slave #1 - Pentium IV 2.66 GHz 512 MB RAM
- Slave #2 - Pentium Quad Core 2.4 GHz 1 GB RAM
- Slave #3 - Pentium D CPU 2.8 GHz 512 MB RAM.
- Slave #4 - Pentium IV 2.4 GHz of 256 MB RAM

The systems are interconnected to one 24-port 10/100 Mbps switch using CAT 5E UTP cable.

5.5 RESULTS AND DISCUSSION

The proposed parallel motion estimation algorithm has been implemented in IDL with embedded Java platform in Windows environment for standard test video sequences. In order to check the efficiency of this parallel implementation, 25 video frames (Jane sequences) are processed with 4 slaves and one master server. By default, first and last frames are considered as I-frames and remaining 23 frames are compressed using block based temporal prediction. The parallel output is verified with sequential processing and in all cases the output matches with the output produced by sequential version. The proposed parallel implementation of video compression using distributed master-slave model has the following unique features compared to other parallel implementations of motion estimation developed by Teddy et al (2002) and Yunsong Wu and Megson (2006).

- This model has been implemented in Windows environment which is user friendly.
- It uses JAVA RMI for remote procedure call and MPI for effective communication and is also platform independence.
- It uses IDL program for image processing operations. Hence video compression algorithm is made free from parallel overhead.
Collaborative memory and frame buffer is used for frame distribution and reordering.

Table 5.1 shows the execution time of a cluster of 4 workstations and one master server and the Table 5.1 is the speedup of parallel implementation of the temporal predictor.

Table 5.1 CPU Execution time (in Second) and Frame rate (per minute) for a cluster of 4 slaves

<table>
<thead>
<tr>
<th>Slave No.</th>
<th>Frames Processed</th>
<th>Total Time (in Second)</th>
<th>Frame Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slave #1</td>
<td>5</td>
<td>292</td>
<td>0.85</td>
</tr>
<tr>
<td>Slave #2</td>
<td>7</td>
<td>340</td>
<td>1.22</td>
</tr>
<tr>
<td>Slave #3</td>
<td>6</td>
<td>332</td>
<td>1.00</td>
</tr>
<tr>
<td>Slave #4</td>
<td>5</td>
<td>295</td>
<td>0.84</td>
</tr>
</tbody>
</table>

The speedup (Petersen and Arbenz 2004) for the proposed parallel method is calculated by,

\[
\text{Speedup} = \frac{\text{Execution - time}_{\text{Sequential}}}{\text{Execution - time}_{\text{Parallel}}} \quad (5.2)
\]

The total time taken by a single PC (Slave #2) to complete the motion vector estimation and approximation for 23 frames is 1212.19 seconds and the time consumed by parallelization of the same operation with 2 to 4 slaves are listed in Table 5.2. The time accounted for the performance measurement is the CPU processing time from the assignment of a frame to the slave till the completion of coding the predicted residuals and the communication cost is not included.
Table 5.2 Speedup of parallel implementation

<table>
<thead>
<tr>
<th>No. of Slaves</th>
<th>Total time (in Second)</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1212.19</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>673.44</td>
<td>1.79</td>
</tr>
<tr>
<td>3</td>
<td>448.53</td>
<td>2.70</td>
</tr>
<tr>
<td>4</td>
<td>338.72</td>
<td>3.59</td>
</tr>
</tbody>
</table>

Figure 5.4 shows the speedup curve. It is observed from the figure that the speedup for parallel video compression is linear with the number of processors used. Theoretically, the speedup should be in linear with increasing number of slaves. But this was not attained by the result of the previous work of parallel video processing model found in the literature (Yunsong Wu and Graham Megson 2006). Due to the enhanced software features and the algorithm introduced in this proposed model, the linearity in parallelism has been achieved.

Figure 5.4 Speedup Curve
This parallel implementation has shown promising results for parallel video compression in a multiprocessor cluster environment. The speedup comparison shows that, the execution time of video compression can be much reduced by performing the compression of different frames simultaneously, on multiple PCs of a cluster of nodes. The parallel compression using this distributed cluster computing model is easy to simulate, implement and test the performance. The data parallelism and task parallelism can also be very well applied to this approach.