Chapter 5

Parallel Implementation Study

1. Introduction: ............................................................................. 2
2. State of art of ACCESS .............................................................. 3
3. Theoretical analysis and selection of topology ............................ 6
4. ACCESS implementation on Transputer .................................. 8
5. Algorithm .................................................................................. 10
5.1 Ring topology .......................................................................... 10
5.2 Tree topology .......................................................................... 13
6. Results ....................................................................................... 15
7. Conclusion ................................................................................ 16
Parallel implementation study

1. Introduction:

The optimization of the data product software had given a high degree of confidence in meeting the high performance requirements. The parallel machines were another resource that can fulfill the requirement. Many scientists have used the parallel machines in the field of image processing [5,8,9]. The literature study (see chapter-3) and the discussion with the individuals have helped in making following useful conclusions:

♦ Only the image processing scientists put effort to prove the usefulness of the parallel machine in the respective field. The computer professionals are not putting efforts in such direction. It may be due to the masses. They like to concentrate on the field where masses are affected by the research.

♦ The scientists were having lack of knowledge about importance and implication of the different topologies, paralleling computation communication, etc.

♦ It is mandatory for image processing scientist to get the parallel processing expertise to achieve better results. The secondary help by the computer professionals will not solve the problem. The SAR experience is the best example of the same [23,24,25].

♦ The image processing scientists put efforts for very limited applications. Most of them had implemented only one application, ie in the area of their interest.

♦ The usage of transputer system for real time image processing application has to be studied.

♦ It is required to compare the tree topology and ring topology.

♦ It is required to isolate the host overhead from the transputer internal communication overheads.

♦ It is required to quantify the efforts required to implement any algorithm on parallel machine in comparison to the sequential code.

♦ Most of the image-processing scientists have used data partitioning techniques for parallel machine. It is obvious due to the large volume of the image files. Many scientists have worked and are working towards finding out the automatic data partitioning techniques.
Parallel implementation study

My guide suggested implementing an application for parallel image processing and studying the above aspects. ACCESS (Automatic Cloud Cover Estimation Software System) was demanding the real time environment. It was simple in nature for implementation. Its sequential code was available and efforts to develop the same were known.

A transputer has four bi-directional dedicated channels per processor for fast data communication in parallel to the computation, which is appropriate for image processing of this nature. Hence, transputer network is considered to be one of the best operational environments for image processing.

In this chapter, theoretical comparative analysis of various topologies and topology selection criteria is discussed in terms of the data transfer overheads. This chapter deals with the algorithm adopted to use transputer capabilities under parallel processing. The comparative results and analysis of Ring and Tree topologies are discussed with respect to the real time processing requirements. The real time requirement is met using tree topology. An overview of auto cloud-cover estimation is also given.

2. State of art of ACCESS

During the last two decades, usage of satellite remote sensing data has increased and is providing encouraging results in many applications. These data is required to undergo certain standard processing like radiometric correction, geometric correction, etc. The precision of information has also increased multifold. These factors have increased the data volume and processing requirement. To meet the increasing demand, data-product generation systems have to eliminate cloudy data, which are not useful to the users, while preprocessing. The manual cloud cover estimation method is not only cumbersome & time consuming but also subjective to the persons perspectives. The automatic cloud-cover estimation can provide an elegant solution in comparison to the manual cloud-cover estimation.

ACCESS is operational program for the Indian Remote Sensing Satellite (IRS) 1A/1B in near real time environment. It takes approximately 3 hours per day to estimate cloud-cover for each satellite pass (approximately 200-MB data) on the VAX-11/750 computer.

Due to increasing data volume from all the passes of multi mission operations and finer resolution of future satellites, there was a need for higher processing power. If the cloud-cover estimation operationally has to be viable, it is necessary that it be done in real time/near real-time. This needs state of art techniques to be used for processing the data for cloud-cover.
A unique region code based thresholding technique is used for ACCESS [29]. The different region codes are identified for entire landmass of India and adjacent countries depending upon the spectral signature of the landmass. The IRS referencing scheme is taken as the base for generating the region codes [2]. The different threshold values are worked out for different types of region code, which is a function of season, operative gain of camera and temperature.

The task looks simpler but it has an inherent problem. The actual percentage of cloud may be small but the effective area masked by the cloud on the ground below may turn out to be high due to the cloud-spread. This needs better techniques to convert actual percentage cloud to effective cloud-coverage. It increases the complexity of the whole processing. The entire image is divided into 16 equal segments. The cloud spread is quantified by adopting a new technique in which the entire scene is divided into 16 segments. The percentage cloud-cover (CC) is estimated individually for each segment. The cloud spread within a segment is quantified by statistical parameters $\mu_P$, $\mu_S$, $\sigma_P$ and $\sigma_S$ which are called as CSP (Cloud Spread Parameters) [29]. $P$ denotes pixel whereas $S$ denotes scan line. $\mu_P$ and $\mu_S$ denotes the mean position of the cloud within the segment. $\sigma_P$ and $\sigma_S$ is the variation in the cloud position which is nothing but the cloud spread.

$$\mu_P = \frac{\sum X_i}{n}$$

$$\mu_S = \frac{\sum Y_i}{n}$$

$$\sigma_P = \frac{\sum (X_i - \mu_P)^2}{n}$$

$$\sigma_S = \frac{\sum (Y_i - \mu_S)^2}{n}$$

$X_i, Y_i$ = Position of the cloudy pixel.

$n$ = No of cloudy pixels.

Such cloud cover estimation is required for all the cameras onboard. The actual estimation is carried out using the data from one camera, the reference camera. The estimation for other cameras is calculated using mapping technique. The area coverage of all cameras is mapped on reference camera as seen in figure-5.1.
The automatic cloud cover estimation technique had given more than 90% accuracy. This is the real achievement as other models in the world are giving only 65% accuracy. The task involved in ACCESS deals with examining a large volume of data and evaluating percentage cloud-coverage scene by scene. The problem essentially is of SAMD (Single Algorithm Multiple Data) type where data partitioning technique is adopted for parallel implementation.
3. Theoretical analysis and selection of topology

The efficiency of the algorithm for parallel processing highly depends on the topology of the processor network. The ACCESS problem essentially is of SAMD type in which thresholding and CSP algorithm is to be applied on each element of the entire data volume. The data partitioning technique is commonly adopted for parallel implementation of image processing problems [5,8,9]. The entire scene can be partitioned into multiple segments in proportion to the number of processor and can be processed in parallel. The overhead for the same is due to the input and output data transfer within the network of processors which depends highly on the network topology and the data transfer rate. Different types of the topologies are considered for the theoretical analysis.

In pipe topology, input data volume is read by root processor and passed to the different worker processors [fig 5.2.a]. Each worker in this topology will have data transferring overhead depending upon the subsequent number of the workers. The typical data transfer overhead for 'm' input data is 'n*m/(2*dt)' for 'n-l' workers.

The input data transfer mechanism and overhead for the ring topology is exactly same as that of pipe topology [fig 5.2.b]. The output data transfer mechanism differs but the overhead is exactly same.

The formation of tree topology depends on the number of communication links available with each processor for networking. Typical example of the tree topology for 4 processors, each having 4 links, is shown in figure (5.2.c). The data transfer overhead for this topology is minimum and equal to 'm/dt' for 'm' data sets and 3 workers. The data transfer overhead increases with an increase in number of sub trees.

The cube connection topology (in general hyper cube; fig 5.2.d) and two-dimensional mesh [figure 5.2.e] can be considered as a sub set of tree topology for finding out the minimum data transfer overhead. For a typical 'm' input data transfer overhead for a 8 processors connected in cube topology is '11*m/(7*dt)' assuming one master and 7 workers. For typical 9 processor mesh network, the input data transfer overhead is '18*m/(8*dt)' assuming one master and 8 workers.

A network of 4 transputers was considered for ACCESS implementation due to its cost effectiveness and processing power capability. The ring topology was selected for the ACCESS implementation considering its expandability and lesser software complexity. Tree topology for the same system was selected considering its minimum data transfer overhead and maximum efficiency.
Parallel implementation study

Figure 5.2-a: Pipe topology

Figure 5.2-b: Ring topology

Figure 5.2-c: Tree topology

Input data flow

Output data flow

Input & Output data flow

Figure 5.2-d: Cube topology

Figure 5.2-e: Mesh
Parallel implementation study

4. ACCESS implementation on Transputer

Master-Slave model is adopted for implementing ACCESS. Master concentrates on I/O and data communication whereas slave does the actual processing. The Master program reads the data from input image files, distributes the same among the workers. Each worker is given a set of scan lines at a time. All the workers are performing the actual computation on the given set. At the same time, the master keeps another set ready. At the end master collects the results from all the workers. It writes to the output file after integrating all of them. The multi threading technique is adopted to improve the efficiency.

ACCESS was implemented on transputer using '3LC' language considering its similarity with standard 'c' [II]. The master task is executing on root transputer, which does the optimum data distribution to the different workers. The data distribution is carried out using worker identification tags. The compute bound module running on different workers are identical. The master has two threads executing in parallel, viz. Input and Send, one reading the data from input file and second sending data to workers. The worker tag mechanism is adopted for the ring topology. Master sends data to the different workers directly in tree topology. The load balancing becomes inherent as the data are being distributed dynamically to the free worker. The synchronization is carried out using semaphore. The multi buffer technique is used for exploiting DMA (Direct Memory Access) power in parallel to computation. The workers have three parts, viz. receive, compute and send, running in parallel. The computation power of root transputer is also used by scheduling the fourth worker on itself in the case of Tree topology. The implementation model is shown in figure-5.3. The ACCESS was also implemented in OCCAM on a single transputer to estimate the gain in processing time [II].
Parallel implementation study

Figure 5.3.A: Parallel ACCESS architecture for ring topology.
5. Algorithm

5.1 Ring topology

Master

1.0 Open input file and output file
2.0 worker_tag = 1
3.0 Send worker tag to First Worker (FW)
4.0 Read first set of lines from input file in buffer A
5.0 Repeat until (EOF)

Figure 5.3 B: Parallel ACCESS architecture for tree topology
Parallel implementation study

{
  5.1 Do in Parallel
      { Send buffer A to FW }
       { Read next set of lines from input file in buffer B }
        Synchronize
  5.2 Increment the worker tag
  5.3 If ( worker_tag > no of worker ) worker_tag = 1
  5.4 toggle buffer A and B
  5.5 Send worker tag to FW
}

6.0 Send Termination code to FW
7.0 Send zero initialized cloud-cover parameters to FW
8.0 Receive cloud-cover values from Last Worker (LW)
9.0 write results in output file and EXIT

Worker

1.0 Receive Tag_Code from previous worker (PW) or master
2.0 Allocate required buffer of memory to Read_ptr, Send_ptr and Proc_ptr
3.0 Receive input data from PW using Read_ptr
4.0 while ( Tag_Code != Terminate )

{
  5.0 Switch ( Tag_code )

  Case Self :
5.1 Toggle Read_ptr and Proc_ptr

5.2 Do in parallel

{  5.2.1 Compute cloud-cover parameters using Proc_ptr }  
{  5.2.1 Receive Tag_code from PW

5.2.2 if ( Tag_code != Terminate ) Receive input data from PW using Read_ptr

}

Case Default:

{  5.1 Toggle Read_ptr and Send_ptr

5.2 Do in parallel

{  5.2.1 Send input data to Next Worker using Next_ptr }  
{  5.2.1 Receive Tag_code from PW

5.2.2 if ( Tag_code != Terminate ) Receive input data from PW using Read_ptr

}

}

6.0 Wait for current processing to Complete

7.0 Receive cloud-cover parameters from PW

8.0 Compute collective cloud-cover parameters

9.0 Send Collective CCP NW
10.0 Send Terminate code to NW
11.0 EXIT

5.2 Tree topology

**Master**

1.0 initialize CODE = Process
2.0 Set all workers to FREE mode
3.0 Open input file and output file
4.0 Read first set of lines from input file in buffer A
5.0 Repeat until (EOF)

{ 
6.0 DO in parallel

{ 
6.1.1 Wait until ( all workers are BUSY )
6.1.2 Send CODE to FREE worker
6.1.3 Send buffer A to FREE worker

}
{ 
6.1 Read next set of lines from input file in buffer B

}

7.0 wait for input completion

}

8.0 CODE = Terminate
Parallel implementation study

9.0  Send CODE to all workers
10.0 Collect cloud-cover parameters from all workers
11.0 Compute collective cloud-cover parameters
12.0 Write cloud-cover parameters in output file and EXIT

Worker

1.0  Receive CODE from master
2.0  Allocate required memory buffer to Read_ptr & Proc_ptr
3.0  Receive input data from master using Read_ptr
4.0  while ( CODE != Terminate )

{
5.0  Toggle Read_ptr and Proc_ptr
6.0  Do in parallel

{ Compute cloud-cover parameters using Proc_ptr }

{ 6.1 Receive CODE from master
6.2 if ( CODE != Terminate ) Receive input data from master using Read_ptr
}

Synchronize

}
7.0  Wait for processing to complete
8.0  Send cloud-cover parameters to master
9.0  EXIT
Parallel implementation study

6. Results

The results obtained for the above mentioned system configurations are given in table 5.1.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>System configuration</th>
<th>Time (Min:Sec)</th>
<th>Total</th>
<th>Disk I/O by root</th>
<th>Processing+ data transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>3LC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Single Transputer</td>
<td>2:34</td>
<td>1:27</td>
<td>1:05</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Ring topology (3 workers)</td>
<td>1:54</td>
<td>1:32</td>
<td>0:26</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Tree (3 workers)</td>
<td>1:29</td>
<td>1:29</td>
<td>0:22</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Tree (4 workers)</td>
<td>1:31</td>
<td>1:29</td>
<td>0:16</td>
<td></td>
</tr>
<tr>
<td>OCCAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Single Transputer</td>
<td>-</td>
<td>-</td>
<td>0:32</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Tree (3 workers)</td>
<td>-</td>
<td>-</td>
<td>0:11</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Tree (4 Workers)</td>
<td>-</td>
<td>-</td>
<td>0:08</td>
<td></td>
</tr>
</tbody>
</table>

It is clearly seen that the disk I/O time has become a bottleneck in all the above configurations. In fact during real time ACCESS processing the data will be directly fed to the Master using one of the transputer links and hence the CPU plus data transfer timings would be a realistic one. To visualize the processing speedup, 3 buffers were simulated in master itself for 100% cloudy scene (worst case). The simulation time was ignored from the throughput calculation. The very same buffers were repeatedly analyzed to simulate the entire scene. The volume of an IRS-1A/1B scene is approximately 5 MB for single band. The third column in the table shows the timings achieved during this exercise. Linear speed up is achieved in tree topology in comparison with single transputer and indicating the efficient usage of the transputer capabilities. The ring topology does not show the linear speedup.

Throughput obtained for ACCESS implementation in 3LC for ring and tree topologies are approximately 1.5 Mbps and 2.5 Mbps respectively. The OCCAM implementation has shown the two-fold speedup giving approximately 5 Mbps throughput.
Parallel implementation study

The processing requirement for IRS-1C is much higher than the IRS-1A/1B. The data transmission rates for the IRS-1A/1B and IRS-1C are shown in table 5.2. The rates correspond to multiple bands. ACCESS deals with single band only hence real time data processing requirement reduces to 1.3 Mbps and 14.1 Mbps for IRS-1A/1B and IRS-1C respectively.

<table>
<thead>
<tr>
<th>Satellite</th>
<th>Camera</th>
<th>Rate (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRS-1A/1B</td>
<td>LISS-2A/2B</td>
<td>20.8</td>
</tr>
<tr>
<td></td>
<td>LISS-1 (4 bands)</td>
<td>5.2</td>
</tr>
<tr>
<td>IRS-1C</td>
<td>LISS-3 (3 bands)</td>
<td>42.3</td>
</tr>
</tbody>
</table>

The real time requirement for IRS-1A/1B can be met even by 3LC ring topology. The IRS-1C requirement can not be met by as the best results from OCCAM implementation in tree topology gives 5 Mbps speed. The only solution is now to feed the sub sampled data fed at interval of every 3rd line. We conducted a small experiment on sub sampling impact on the estimation accuracy. We found that the accuracy was affected only by 1%. We requested the ACCESS designers to conduct the detailed study for the same. Amol Mahamuni et.al. have shown that the cloud-cover estimation accuracy reduces by negligible amount even with data sub sampling of the order of 3 pixels by 3 scan lines [6]. Thus data processing speed can be met to the IRS-1C data transmission rate by sub sampling the input data at every 3rd line while feeding to transputer link.

An experiment was conducted for finding out the optimum buffer length data communication. A small program was written that transfers 50 MB data buffer by buffer from root node to another node. Various buffer lengths starting from 4K to 64K at step of 2K were considered for the experiment. The buffer length of 32K was found to be optimum. The variation between worst case and best case was found to be 10%.

7. Conclusion

The application was implemented on transputer system [II]. Tree, mesh and cube topologies were considered for comparison. Mesh and cube topologies are extension of the tree. The theoretical analysis also shows that the communication overhead was 160% more for cube and 225% more for mesh when compared to tree. This was due to larger number of nodes involved in the respective topologies. Hence, it was decided to use ring and tree topologies. The experiment concluded the following:
Parallel implementation study

- The development time was almost 3 fold even after having knowledge of parallel processing and expertise in C. The amount of effort required to learn C was excluded.

- The disk I/O time (time taken to load the data from disk to master node of the transputer network) of the application was 1 minute and 27 seconds for approximately 5 MB data. This observation may be very useful for T. Geeta Prasanna to refine their conclusion [20].

- Almost linear scalability was observed in case of tree topology. The same was not observed in case of ring topology due to inter process communication overheads.

- The transputer can be used for the real time implementation of the application for IRS-1A and IRS-1B satellites. The computation and communication can be made fully parallel.

- The tree topology requires more concentrated efforts compared to ring topology due to synchronization issues but provides very good efficiency.

- The application was then implemented using OCCAM to compare its efficiency with that of parallel C. It was found that the OCCAM was 200% efficient in comparison to parallel C.