Chapter 3

Literature Study
We studied the image processing applications, oriented towards remote sensing. We surveyed the usage of different computer languages and system for image processing. Most of the scientists are using either C or Fortran. The VAX/VMS and Unix were the most widely used systems when our study began. The transputer based PARAM was the most popular machine for parallel image processing. VAX/VMS, Unix and Transputer based machines were taken as test bench machines for our study.

We started our optimization study by exploring the parallel processing capabilities of VAX/VMS. The VAX/VMS operating system & its architecture was studied [10,11]. As the software was developed in Fortran, the efficiency aspect of Fortran was studied in detail [12,13]. Users considered Fortran as highly optimized compiler [10,13]. Only real time applications were using the assembly language [6]. Authors studied the assembly code generated by the Fortran compiler for the radiometric correction application. The authors felt that there was a considerable scope for improvement. The VAX/VMS had separate I/O controller offloading CPU for processing [15]. The operating system provided the system services (API in today’s terminology) to use them in parallel [16,17]. Parallelizing I/O and processing can improve the throughput. The hard disk controllers were intelligent [15]. Each controller had built-in capability to keep the multiple requests in queue. It first services the request whose data is near to the disk-head.

The data-products generation software were optimized and achieved 300 % efficiency [1]. Please see chapter 4 for details.

Then, we studied the parallel processing work carried out by various image-processing experts.

Ms T. Geeta Prasanna, from NRSA, had put efforts in studying the parameters affecting the parallelization of image processing algorithms [20]. They worked on transputer based parallel machine and used OCCAM. They concentrated on single application of median filter. The image size, of the input data and output data, was 512x512 for each of the four bands. They selected two important topologies; pipeline and star network. They compared various combinations of transputers (1T, 2T, 4T and 15T) for various sizes of the data packets. They concluded that overlap of reading and processing improves the response time. The pipeline topology was found more suitable. (Normally the star topology should be more efficient than pipeline.) The data communication timings achieved by them was approximately 32 to 38 seconds. These timings do not match with the transputer capability [21]. It was observed that they didn’t discriminate between the disk I/O overheads and transputer internal communication overheads.

V. K. Agarwal [22] implemented a prototype to visualize the usage of transputer for climate model using parallel Fortran. They compared the timing with the
corresponding sequential code. They found good scope for improving throughput, but recommended for fresh implementation instead of transformation from sequential to parallel. Such recommendation leads to the loss of efforts put for developing the sequential code. A tool that converts the sequential code into the parallel one would be very useful for such an application.

S.K. Basu [23] had implemented the data processing of the Synthetic Aperture Radar (SAR) on PARAM, the transputer based parallel machine. It was a big project and was implemented in collaboration with C-DAC, the designer of PARAM. They concluded that the processing efficiency was linearly increasing up to 32 nodes. It then saturates due to inter-processor communication overheads. The overhead shown in the graph was below 50 seconds for the typical case, irrespective of number of node. It was fractionally increasing from single node up to 64 nodes. It is very risky to conclude about the scalability when the processing requirement is low, i.e. below 50 secs. Furthermore, it was found out, during the discussion with them, that they didn’t parallelize computation and communication. Later, P. Robert [24] restructured the entire code, for airborne SAR, to parallelize the communication and computation. He achieved 45% efficiency in the same software. The same knowledge was used in efficient implementation of speckle-reduction algorithm [25].

H.K. Garg [26] had implemented a few image processing applications on transputer using the ring topology. They compared the 4 node transputer timings with 386/387 based Intel machine. They used OCCAM in some applications and parallel C in other applications. This effort only conclude that 4-node transputer machine is a better option when compared to 386/387. It does not give any idea about the scalability of application implementation. They paralleled the computation and communication. However, they did not give any numeric comparison. Furthermore, they didn’t isolate the host I/O overhead from the total time. Their throughput shows single figure including computation, communication and the host I/O overheads.

A few topologies were tried out for a few image-processing applications on transputer network which includes ring, farm feed, bilinear pipe, hyper-cube [27]. They did not give the reasons for selecting a specific topology for a specific application. It would have given better conclusions if they had compared different topologies for a single application.

Critical observations were made from all of the above studies for deciding the focus of our work. Please see section 4 of chapter 2.

We implemented ACCESS on transputer using parallel C and OCCAM [II]. Chapter 5 discusses the same.
Thereafter a study was conducted to evaluate the available automatic parallelizing tools.

John Webb has shown the capability of ADAPT language for architecture independent image processing using Split and Merge model [30]. Han Wang and J Webb have also shown the importance of APPLY, the low-level language for image processing on a transputer array [31]. The issue addressed by them is quite interesting. Each parallel machine is having its own parallel language. The program written for one machine is normally not portable on any other machine. The higher level tools, which provides the portability, are not efficient enough to explore the full power of the machine. They introduced two more languages (APPLY and ADAPT) to solve the portability issue. They implemented various image-processing applications using APPLY and ADAPT. They tested the implementation of the languages on two machines, Warp and Sun. They proved ADAPT efficiency on Warp. However, the scientists now have to learn two more languages and understand the new model.

Manish Gupta and P. Banerajee have demonstrated the automatic data partitioning on multi-computers by introducing the notion of constraints on data distribution of various data structures [32]. They dealt with the large array operations inside a loop. They had considered multi dimensional array and multiple loops also. They tried to distribute this loop operation to the various processors. They calculated the parallelization constraints in terms of communication penalty and corresponding time overhead. As an example,

\[
\text{Do } I=1,n \\
\quad \text{Do } J=1,n \\
\quad \quad A(I,J)=f(A(i,j),B(Ij)) \\
\quad \text{Endo} \\
\text{Endo} \\
\text{If it is distributed on two processors to calculate cyclic value of A then} \\
\text{Penalty} = \frac{C_p}{N}
\]

Where \(C_p\) is estimated time for sequential execution and \(N\) is the achieved speed up.

Both, Gupta and Banerajee, are working in developing a parallel compiler where they would like to take full responsibility about the parallelization. They have admitted the limitations in their approach. There is no guarantee about the optimality of results.
Literature Survey

obtained. They have dealt with only Fortran. This particular work is implemented as feasibility study for restructuring of Parafrase-2 [33]. It seems that they are working in the field of mid grain data parallelization. In case of remote sensing, the input image size is having large volume of data and hence it falls in the category of coarse-grained data parallelism. Hence, a tool, that can carry out the coarse-grained image partitioning and give better as well as consistent optimization, is required.

Zhiywan Li et. al. have discussed lambda test algorithm for an efficient and accurate data dependency analysis of multidimensional array reference [34]. Manish Gupta has used this approach for calculating data dependency for development of parallel Fortran compiler [32].

The usage of recursive data structures is quite common among programmers, for efficient implementation. The recursion is a very efficient and useful facility available in some of the conventional language but it makes programs more complex to parallelize. L Hendren and A Nicolau have presented a method to parallelize the programs having recursive data structures [35]. The approach followed by them consists of the following steps:

- Design a simple imperative language with recursively defined dynamic data structures.
- Develop interference analysis tools for programs that use dynamic data structures in a regular manner.
- Implement the tools with a prototype system.
- Develop parallelization techniques based on interference analysis.
- Test the method on a set of representative programs.

The scope of this particular work is very limited to recursive data structure but it is very useful for development of parallel compiler. This approach also falls in the category of mid grain parallelism.

L. M. Patanayak et.al. had demonstrated the parallelising parallel translator (PPT) to convert a Fortran program into OCCAM [36]. PPT parallelises the outermost parallelizable loop in nested loops. It converts Fortran input and output statements into its equivalent OCCAM statements. Its parallelism analyzer finds dependency relationship in terms of direction vector. Banerjee-Wolf test [37] have been adopted for the same. The example taken for proving the concept falls in the category of mid-grain. It seems that major concentration is given to the language conversion rather than parallelization. However, the introduction of parallelism analyzer shows the future direction of the work. They have claimed that multiple iterations of nested
loop were considered perfectly for concurrent executions by PPT but no example was given. This work also falls in the category of mid grain parallelism.

The tools like KAP parallelizer for DEC Fortran and DEC C programs also had given importance to the mid grained parallelism [38]. The KAP parallelizer provides tool to the user for selecting loops to parallelize. This can definitely increase the efficiency because the user is the best judge to select the parallelizable program code.

Conclusions made from the above study are dealt in section 4 of chapter 2.

We decided to carry out a feasibility study for developing a tool that converts sequential ‘C’ programs to parallel ‘C’. Its focus should be on coarse grain application. We extended the idea of parallelizing technique based on interference analysis[35] and developed a directive-based translator (DBT) for image processing having single partitionable input image [III]. Please see chapter-6 for more details.

The distributed processing technology was emerging when the study began. It was becoming realizable due to the new inventions in the field of networking. Distributed processing was becoming a necessary tool for optimal use of resources and computational power. High Performance Computing (HPC) initiative in North America, India, Australia and Europe aims at developing a generalized wide-area computing infra-structure, that can be easily used by research worker in various fields. The deliverables generated by the work is discussed in perspective of similar efforts in the world, during the last couple of years.

The research in the field of distributed processing was focused on the following three areas (pertaining to the image processing)

- Communication facilities for distributed computing
- Efficient usage of distributed resources
- Parallel distributed processing
- Load balancing

Many scientists are working in the field of finding out optimum schedulers for parallel processing, multiprocessors, array processors, etc. They have concentrated on efficient usage of the technology, not on the ease of use.

C Kim and H Kameda proposed a load balancing algorithm that determines the optimal load for each host so as to minimize the overall mean job response time in a distributed computer system that consists of heterogeneous hosts [40]. We concluded from our experience with the data-product scheduler system[18] that we need not put
such special efforts for load-balancing algorithm. The kernel of such futuristic multi processor system will be inherently efficient.

MOSIX is an enhancement of Unix that allows distributed memory multi computers, including LAN connected network of workstations, to share their resources by supporting primitive process migration and dynamic load balancing among homogeneous subsets of nodes [41]. These mechanisms respond to variations in the load of workstations by migrating processes from one workstation to another, primitively, at any stage of the life cycle of a process. The granularity of the work distribution in MOSIX is the UNIX process. Users can benefit from the MOSIX execution environment by initializing multiple processes, e.g. for parallel execution. Alternatively, MOSIX supports an efficient multi-user execution environment. The NOW MOSIX is designed to run on configurations that include personal workstations, file servers and CPU servers that are connected by LAN's, a shared bus, or fast interconnection networks. In these configurations each workstation (node) is an independent computer, with one or many (shared memory) processors, local memory, communication and I/O devices. MOSIX can be implemented for any version of Unix and for any hardware, but the processors must be homogeneous to allow process migration. Users interact with multi computers via their own workstations. The results shown by NOW MOSIX were very encouraging giving more than 98% efficiency compared to theoretical one. NOW MOSIX concentrated on efficient load balancing on homogeneous system. NOW MOSIX does not give importance to heterogeneous configuration. Further more it does not give any tool regarding automatic parallelization. That means the users will have to hand code for parallel processing; MOSIX will only provide an efficient environment.

Data processing using middle ware (software) between operating system and end user's application) is described by W. Farrel on networks of Unix based workstations [42]. This application is very similar to that of IRS data processing systems. They too preferred to develop more than 600 thousand lines of code using standard TCP/IP technology over networks of workstations instead of using any distributed technologies like MPI [43,44], or PVM [39,45], or NOW MOSIX [41]. They have developed the application-specific client-server model and carried out workflow management within the application. According to their conclusion, PVM was unable to prove reliability.

MPI, PVM, and SNIPE, are the earliest systems to provide the distributed processing environment. The MPI supports the library for data communication in distributed processing environment. PVM has extended the facility and provided the process scheduling and monitoring facilities. SNIPE [46] is a meta-computing environment.
Message Passing Interface (MPI) provides the useful library routines for data communication over the heterogeneous systems configuration.

Parallel virtual Machine attracted the computer professional working in the filed of parallel processing [39]. The PVM supported C and Fortran. The PVM support was restricted to the homogeneous Unix configuration initially and had future scope to support heterogeneous environment. PVM forum conducted regular survey for its usage [39]. The following observations were made from the survey conducted during 1995:

- 31 image-processing professional were using PVM.
- Most of them were from medical imaging field.
- Approximately, 10 remote sensing scientists used PVM in their field.
- Only 3 persons showed interest in PVM from INDIA. They used PVM for simulation, parallel-processing tutorials and fluid mechanics.

PVM forum site does not have any information about the survey conducted after 1995.

We studied the efforts required to develop a distributed application using PVM. It was found out during the exercise that the efforts required for such a development are very similar to those required for a parallel machine. The advantage is that we can configure our low-end machines as a powerful parallel virtual machine. The PVM site also claims that PVM is nothing but a parallel machine for a poor man.

SNIPE is to metacomputing systems as Unix is to operating systems. It is distributed systems test bed that provides much of the functionality of the systems like PVM without the rigid definition of a virtual machine. It provides process control like PVM based on daemons and resource mangers.

Legion [47] is a metacomputing system that can accommodate a heterogeneous mix of geographically distributed high-performance machines and workstations. Legion is an object-oriented system where the focus is on providing transparent access to an enterprise-wide distributed computing framework. As such, it does not attempt to cater to changing needs and it is relatively static in the types of computing models it supports as well as implementation.

HARNESS [48] presents a next generation distributed virtual machine (DVM) model. It presents primitive mechanism that address key requirements of this new model. Its modular kernel architecture supports a level of flexibility in the set of system components that is not available under monolithic operating environment such as PVM and MPI. Its system registry allows distributed control of the system.
configuration in order to enable the dynamic addition of new components and libraries, as well as the merging and splitting of distinct virtual machines. These flexible mechanisms do not fully define how dynamic reconfiguration will proceed, but merely make such a reconfigurable DVM possible.

Above work concentrated to provide computation-power to a large and complex application efficiently. All of the models need a specialized skill of parallel and distributed processing using C & Fortran. They are the parallel machines over distributed systems. They concentrated on efficiency, scalability, heterogeneity, etc. We concentrated on vast community of scientists rather than efficient programmers. Our tool, DEDIP [VI, V], made the distributed application development very easy. It extends and generalizes the work by [18] and [66]. Furthermore it adopted a GUI usage for parallelizing compiler from [38] and extended for distributed application configuration.

CVS prakash suggested scheduler support for smooth operation [18]. The efforts were well appreciated by the operation staff. These data products were being generated in pipeline mode. He implemented the Bruno's [69] model to reduce the mean finish time. The data processing units were reconfigured in terms of small independent processes. This scheduler environment schedules five predefined processes of data product generation in sequential manner on VAX/VMS system configuration. Pipeline processing was supported for better FPU utilization.

CVS Prakash have described a network server daemon to access remote image displaying system in a networked distributed systems configuration [66].

It saves manpower efforts and compromises with little efficiency loss. We adopted well-known intermediate-file based interface among the processes of the application. It is widely used by the vast community. We automated the transferring of such files from one node to another. However, we do not restrict people using MPI in our environment. The complete model is discussed in chapter 7.

XPVM [49] is a graphical control interface and a performance monitoring for PVM. It does not support functionalities for parallel compilation. At XPVM start up, if PVM is not already running, the interface starts it on each node of the metacomputer. XPVM starts the application execution using the pvm_spawn() function and it uses PVM's mapping approach. XPVM allows parameters to be passed to the application at startup, and the output of the execution is redirected to the interface. XPVM does not support the execution of interactive applications. XPVM supports on-line & post-mortem monitoring, and provides several animated views to monitor the execution of PVM application. To analyze an application using XPVM, a user does not need to instrument the source code but only to compile the code using the PVM library,
which has been instrumented to capture tracing information at run time. XPVM is well documented and it provides on-line help.

PADE [50] helps the application developers to write parallel applications for heterogeneous networked computers. It includes: a GUI to configure a virtual machine, a program organizer and a parallel make utility. Moreover, it allows other development tools to be accessed such as editors, debuggers and performance monitors. To configure a virtual machine PADE uses a file that stores the list of Internet addresses of the nodes containing in it. To authorize them to execute remote commands PADE uses the .rhost mechanism. To manage the virtual machine PADE exploits the PVM console or the XPVM capabilities. Compilation in PADE is supported by two processes: PVMmake, which is executed on the user local host and Target, which is executed on the remote nodes. PVMmake broadcasts source files, make files and the compilation commands on the Target processes. The Target processes start the compilation in parallel. A useful compilation functionality in PADE is the possibility to transfer only the files modified after the last compilation. The compilation results can be redirected to a user xterm window or to a PADE window. In the latter case, the overall output is displayed at the end of the last compilation. This means that a long time may have to pass before the user can verify the correctness of the compilation. In PADE, an application can be executed with two different methods; in terminal window or by PVMmake. The former is useful for interactive applications and when the user is not interested in saving the standard output. On other hand, the PVMmake method is useful if the user wishes to receive the output on the screen and if the input to the program can be redirected. To start the execution of an application, a binary file specified by the user must be executed on the local node. It spawns the other application components on the remote nodes. The master component can thus exploit user-defined mapping mechanism. PADE is also well documented and provides online help.

WAMM [51] is a graphical user interface based on OSF/Motif and PVM. The authors claim that it integrated XPVM and PADE functionalities in WAMM. To use WAMM, users have to write a configuration file containing the description of the nodes that can be inserted into virtual machine. WAMM initiates the execution of PVM based on the configuration file. The WAMM GUI supports various operations: insert hosts into PVM; remove hosts from PVM; check the status of the host; compile on selected hosts. WAMM does not destroy the source code copied on the host for compilation. This has an advantage enabling the users to modify the source code later if needed. On other hand, use has responsibility to manage the code consistency among the nodes. WAMM has integration with PVM to monitor the applications. WAMM has scope to support MPI.
Literature Survey

The above graphical user interfaces are mainly concentrating on ease of use for one application. We have not only concentrated on remote compilation but also provided efficient and smooth operational environment for simultaneous execution of multiple parallel and distributed applications. We also can provide the parallel compilation facility in future.

Khoros [52], AVS [53], SACImage [28] are the interactive tools for image processing and analysis. SACImage provides this tool on low-end machine like 386, 486. AVS provides very good facilities for 2D and 3D visualization. Khoros is an excellent tool providing many image processing functions. It allows the users to insert their own modules in the Khoros environment. Khoros has a visual programming tool, canata that makes the image processing application development very easy. Khoros and AVS support distributed processing environment also.

Above tools are having many built-in functions for image processing. These tools do not support the smooth and integrated operational environment in contrast to our model. Furthermore, one has to logon to a system having the tool for accessing it. Our model supports the browser based GUI allowing the access from anywhere on Internet [VI, VII].

JPVM [54], and Java MPI [55] are the Java extension of PVM and MPI respectively. They have provided the facility to users for parallel processing using Java.

JavaParty [56], ParaWeb [57], Charlotte [58], Popcorn [59], and Javelin [60] are Java based future generation systems for distributed computing using Java. JavaParty provides mechanisms (built on top of Java RMI) for transparently distributing remote objects. ParaWeb is an implementation of the JVM that allows Java threads to be transparently executed remotely.

Charlotte provides a comprehensive environment specifically for volunteer based computing which transparently handles faults and load balancing for parallel applications. Charlotte provides high-level solution that decouples programming environment from the execution. The advantage of this approach is that problems such as load balancing and fault tolerance are handled by the runtime systems; the disadvantage is that the programmer does not have explicit control over resource utilization. However, its eager scheduling enables the runtime systems to efficiently provide load balancing. Charlotte provides a predictable virtual machine model and a runtime system that implements this machine on the unpredictable machines of the web. It has a single point failure, the manager. Also, having a single manager limits scalability.
Popcorn provides a Java API for writing parallel programs for Internet distribution. Applications are decomposed, by the programmer, into small, self-contained subcomputations, called computelets. The application does not specify a destination on which the computelet is to execute. Rather, a centralized entity called “market” brings together buyers and sellers of CPU and determines which seller will run the computelet. Market incentives are supported, e.g. two different types of auction for CPU cycles. User participation as a seller is made extremely easy—just point a Java enabled browser to the market website and fill in a user profile to open an account for so-called ‘Popcoins’, the micro currency used by Popcorn.

Javelin is an infrastructure for volunteer based computing. In Javelin, a stand-alone application called the broker, acts as the central task repository and scheduler. Javelin provides multiple communication models, such a PVM and Linda, and the broker is used to forward messages from one volunteer to another. No OS or compiler modifications are needed and system administrative costs are zero. Javelin allows machines connected to the Internet to make a portion of their idle resources available to remote clients, and at other times use resources from other machines when more computational power is needed.

Bayanihan [61] and NinFlit [62] are also very similar to the Javelin. Bayanihan provides communication in the form of migratable objects which are implemented on top of the HORB [63] system. It uses eager scheduling, and thus, is able to provide transparent fault tolerance and load balancing like Charlotte. NinFlit is an infrastructure for migratable objects which is being targeted for ideal cycle based parallel computing.

All of the works are very similar to that of JPVM expecting efficient parallel processing skills. Moreover, they have given different treatment to the subject. Most of them have used the Java for their development to support the metacomputing (Distributed parallel computing) support. Some have supported browser based GUI using Java applets. All of them expect the users to use Java for developing their metacomputing application. Java byte code can be executed on any system supporting JVM. Hence, remote compilation is not required. Furthermore, the code can be transferred during run time. Hence, any machine can be plugged dynamically in the runtime environment. Javelin proposed volunteer using this philosophy. All of them are working in the filed of parallel processing since years and have sound collaboration.

Our final model, WebDedip [VI, VII], is developed in Java making it truly system independent like others. Our GUI is also browser based providing the roaming profile to the users. We also use the volunteer concept in our agents. However, our agents are not browser-based. We focused on requirement of scientist rather than efficient
programmer. Most of the above works have concentrated on scalability and adaptability. We concentrated on ease of use. The models expect users to use Java as development language. We provide the freedom to the user to use any language. The final model is discussed in chapter 8.

The WebFlow [64] is the closet to our work. They also concentrated on "ease of use". They Java Swing enabled graphical user interface for connecting the computing modules. The WebFlow expects the programmer to use Java for his development. Other languages have to use the Java native interface to use WebFlow. The users have to implement three methods; initialize – to define their input and output ports for communication with other module; run – to be executed on remote machine; destroy – for smooth cleanup. WebFlow uses Globus [65] tool kit at back-end. It has futuristic plan for interfacing with UML-based GUI developed by Rational.

It is very sound work. Its online document does not give any clarity about how to implement the interface. The pages are under construction.

The WebFlow, aiming to be future generation computer-system, expects object oriented programming using Java. We do not impose such restriction. We allow the users to use any language. We have conducted a survey in three major areas working in the field of image processing at Space Applications Centre. We contacted their system administrator to find out how many are using Java. Out of 150 people, only 10 people are using Java and 10 are learning Java. Most of them are either BE or MCA. Rest of the people are using Fortran and C. Furthermore, We support strong operational environment for simultaneous execution of multiple applications. We have used Java 1.1 instead of Java Swing. This enables us in using very old systems for user interaction. Incase user wants to directly interface with one environment to execute his function on remote nodes, he needs to implement only one interface called "Execution On Remote Node (object)".