MODELING AND SIMULATION OF PARALLEL PROCESSING ARCHITECTURE FOR IMAGE PROCESSING

K.MANJUNATHACHARI[1], DR.K.SATYAPRASAD[2]
[1]PROF. AND PRINCIPAL, JNTUCE,KAKINADA
[2]ASSOCIATE PROFESSOR IN ECE, GPREC,KURNOOL

ABSTRACT

Typical real time computer vision tasks require huge amount of processing power and time for handling real time computer vision applications. The nature of processing in a typical computer vision algorithm usually ranges from many small arithmetic operations (Fine Grain Parallelism) to symbolic operations (Coarse grain parallelism). The task become more complicate while considering image processing application due to large data sets and there processing. The existing processing system responds efficiently under sequential working and result in efficient output, but results in a slow operating system which results in a inefficient processing system under high speed image processing systems. Parallel processing founds to be the only solution to obtain the require processing speed for handling high-speed image processing applications. The existing image processing systems support usually only one suit of operations at once and fail to respond under multiple tasks. System taking single instruction or multiple instruction process operates using low level and high-level operations. Generally SIMD architecture is suitable under low level processing while MIMD architecture is suitable for high-level processing. This paper explores on modeling and simulation of parallel Image Processing architecture for Image Processing applications using Parallel Virtual Machine(PVM) , MATLAB external interface API and C language on the Linux operating system platform.

KEYWORDS: Image processing, Parallel Processing, PVM, High Performance Computing(HPC)

1. INTRODUCTION

The overall application area of this paper is computer vision and parallel processing, the processing of vision information by means of a computer system. This chapter illuminates the area of computer vision by looking at applications and their associated computer system setup. It then looks at how computer vision is realized, what are the methods used, and what are the related problems.

Computer vision

Vision plays an important role in the life of living beings. The concept and feedback of vision is important for everyone in order to move around, communicate and interact. With *computer vision* we try to process (three-dimensional real world) vision information automatically by means of a computer system. The reasons to do this are numerous and computer vision covers a very broad research area ranging from the computational understanding of the vision concept to practical matters like automatic visual inspection of a production line, *machine vision*. Examples of several computer vision application areas are: Control of a robotic assembly cell, Image warping, compression, encoding, and transmission, Video conferencing, Autonomous vehicle control, Object detection/recognition/tracking, Machine vision The common factor in all these applications is that vision is used for an automated application, no human intervention or guidance is involved in the processing of vision. Yet, the processing and interpretation of the vision information are not trivial.

Although seemingly easy for living beings, coping with vision is less natural for computers. Images need to be captured, digitized and processed until some form of decision or conclusion can be made. Thus image processing, the analysis and manipulation of data, originally in the form of an image or image sequence, by a computer is acknowledged as one of the grand challenges of computing [1]. The reasons for this are: Data size, Computation complexity, Time constraints, Types of operations, Variety in image processing algorithms, Different data types.

Image processing applications are characterized by the requirement for transformations between disparate data types to be carried out efficiently, and for computation to be executed efficiently on all data types involved in the solution of a given problem.

System setup

Figure 1.1 shows a simple generic setup of a general computer vision system that could be used for the computer vision applications mentioned in Section 1.1. It consists of a sensor part to capture vision information (from the real world) and convert this to electrical signals. These electrical signals carry the (sequence of) image(s) generated by the sensor. The specific sensor used determines the resolution (number of pixels) of the images, type of the image information (color, range, etc.), as well as the number of generated images per time unit (frame rate). For processing, the images need to be captured in order to be able to be processed. This makes the images available in the computer memory for processing. The representation of the image can be different and depends on the specific application and memory resources. The processing of the captured image(s) is done by the computer system followed by analysis and interpretation resulting in control feedback to the application.

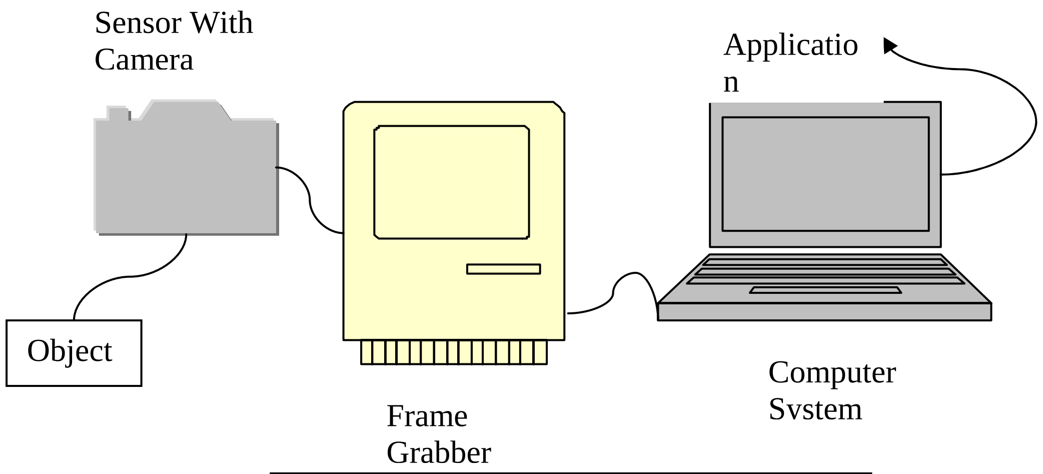


Fig 1 1 General Setun of a computer

The different steps that can be distinguished in a computer vision application on such a computer vision system are:
Image formation, Image acquisition and (pre-)processing, Image analysis, Image interpretation Application control.

A typical computer vision task contains various different types of processing operations. Normally a vision task starts with a plain image and while processing the type of operations moves from arithmetic to symbolic and the amount of data to process reduces until in the end some decision is reached, the *image understanding*. Generally [2,3], three levels of image processing are distinguished to analyze and tackle the vision application: low-level operations, intermediate-level operations, and high-level operations.
1. Low-level operations. Images are transformed into modified images. These operations work on whole image structures and yield an image, a vector, or a single value. The computations have a local nature, they work on single pixels in an image. Examples of low-level operations are: smoothing, convolution, histogram generation.
2. Intermediate-level operations. Images are transformed into other data structures. These operations work on images and produce more compact data structures (e.g. a list). The computations usually do not work on a whole image but only on objects/segments (so called areas of interest) in the image. Examples of intermediate-level operations are: region labeling, motion analysis.

3. High-level operations. Information derived from images is transformed into results or actions. These operations work on data structures (e.g. a list) and lead to decisions in the application. So the high-level operations can be characterized as symbolic processing. An example of a high-level operation is object recognition.

Parallelism

Given the huge amount of data and processing involved with computer vision, solutions of image processing problems have almost always been tackled by the exploitation of parallelism in one form or another. The types of parallelism present in computer vision applications vary. Low-level operations have a fine-grain type of parallelism where lots of simple operations can be done in parallel. The parallelism seen with high-level operations is more coarse-grain; a limited number of more complex operations or tasks can be executed in parallel.
Although parallel computers offer sufficient raw processing power in order to solve the time constraint problems, they have more degrees in freedom with respect to architecture and are more difficult to program than sequential computers. But when real-time requirements are to be met, normal (sequential) workstations simply are not fast enough. With the current trend that applications get more and more

demanding with respect to processing power, it does not seem likely that plain workstations will become fast enough for real-time imaging applications in the near future. So more processing power is needed than can be achieved by a single sequential workstation and using parallel processing systems seems to be the most likely way to satisfy these real-time requirements, despite their complexity of architecture and programming.

Architectures
Simply using a parallel system is not sufficient to successfully run computer vision applications. The parallel system should perform well, i.e. have a high performance, in all three levels of computation in image processing applications: low, intermediate, and high-level operations. The two main types of parallel systems, homogeneous SIMD (Single Instruction Multiple Data) or MIMD (Multiple Instruction Multiple Data) systems, fail this requirement, even if they are designed with image processing in mind [4,5, 6, 7,8, 9, 10,11, 12, 13]. Complete image processing applications could be executed on either SIMD or MIMD systems but this alwassys involves inefficient computation of one part of the problem as neither SIMD nor MIMD perform well in all levels of image processing.
Generally, SIMD architectures are suitable for exploiting the fine-grain parallelism of low-level image processing operations while MIMD architectures are able to use the coarse-grain parallelism of high-level image processing operations. But even heterogeneous combined SIMDMIMD systems may not perform well enough for real-time imaging applications when intermediate-level image processing is not handled sufficiently. Reconfigurable systems that can switch between the type of parallelism that is supported are also not a real option: the real-time requirements demand all levels of image processing to work in parallel, not sequentially. Even when reconfiguration of the system could be done without overhead, the reconfigurable system would still support only one type of parallelism at a time and not simultaneously.

Bottlenecks
Given that low-level operations can be efficiently carried out by highly-parallel systems operating in SIMD mode and high-level processing can be effectively executed by moderately parallel systems acting in MIMD mode, the problem thus boils down to the interfacing of the low-level and the high-level operations. The (parallel) architecture component that is suitable for the intermediate-level processing is yet unclear. Interfacing these levels is however essential considering the growing number of applications having real-time processing constrains; applications like video conferencing will play an important role in the information technology era of the future.
Besides interfacing the different SIMD and MIMD parallel architectures, programming the system is another problem. The attractiveness of using homogeneous SIMD or MIMD systems is that they have a coherent programming model and data storage structure. Heterogeneous systems incorporating more than one parallel computing paradigm not only add complexity to the architecture but to the programming (model) as well. Yet, to meet the real-time constraints of a vision application, the construction, programming, and use of heterogeneous systems for computer vision is desirable.
The structure of this paper is as follows: section 2 reviews high performance computing (HPC), HPC software technology and parallel image processing. Section 3 and 4 describes our programming environment using parallel virtual machine (PVM), analysis, design and implementation. Section 5 presents our results for the above simulation model and draws conclusion and points to future work.

2. High Performance Computing and Parallel Image Processing

2.1 Classification
Modern day problems taken from Information Technology (IT) application areas such as engineering and scientific numerical simulations, information processing and wide-area data exchange in commerce are too demanding to run on a single microprocessor machines and yet at the same time do not need the performance level provided by supercomputers. Such applications often require multiprocessor systems, which are capable of parallel execution, ensuring more accurate, reliable results, increases in throughput and reduced turnaround time. Solving these kind of problems which often require significant computational power, processing of very large amounts of data quickly or need to operate interactively across a geographically distributed network, falls under the domain of High Performance Computing (HPC).
HPC covers a range of hardware platforms and software techniques, which are explained below.

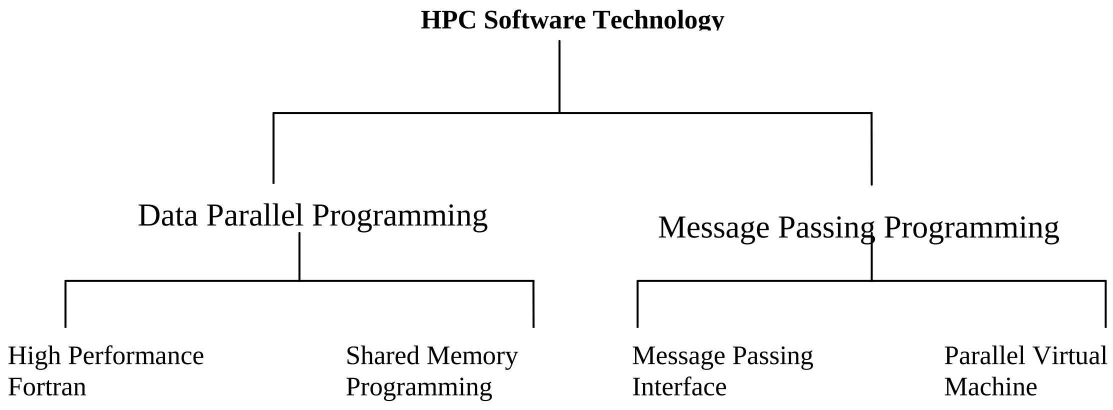
2.2 HPC software Technology
HPC systems make use of leading-edge processor technology and involve parallelism or multi-processing. These systems can be classified into four categories as shown in Figure2.1. In addition, these four categories can be grouped under the following two headings

Shared Memory Systems and Distributed Memory Systems . The Figure 2.1 shows the classification of HPC systems graphically.

Parallel Virtual Machine

PVM predates MPI by a few years, the original project being started in 1989 at Oak Ridge National Lab in the US. PVM was originally designed to operate on heterogeneous networks of workstations, and has important features for supporting applications in such environments. From its cluster-oriented beginnings, PVM has been ported onto SMP and MPP systems and is a popular choice for an MPS. PVM consists of two parts, a daemon process that any user can install on a machine and a user library containing routines for initiating processes on other machines, for communicating between processes and changing the configuration of machines. PVM uses the following routines to identify other processes in the system: - pvm_mytid(), pvm_bufinfo, pvm_gettid() and pvm_tasks(). Processes can also be identified by a name and an instance number by joining a group.

Figure 2.1: Classification of HPC Software Technology


HPC Software Technology
Data Parallel Programming
Message Passing Programming
High Performance Fortran
Shared Memory Programming
Message Passing Interface
Parallel Virtual Machine

There are also routines to add (pvm_addhost()) and delete(pvm_delhost()) hosts from the virtual machine, routines to start up (pvm_spawn()) and terminate (pvm_kill()) PVM tasks, routines to send signals to other tasks (pvm_sendsig()) and to find out information about the virtual machine configuration(pvm_conf()) [15].

The two features of PVM that distinguish it clearly from MPI are

- Dynamic process management which is the ability to create and destroy processes during the lifetime of an application run.
- Standard machine configuration as PVM defines a standard method of configuring the parallel machine.

These features are in some sense higher level than the types of function defined in MPI. As MPI defines a standard inter-processor communications API that can be implemented efficiently on native hardware, it is thus possible to implement PVM on top of MPI. Attempts have been made to merge the features of PVM and MPI and the project PVMPI[15] involves creating an environment that allows access to message passing features of MPI and virtual machine features of PVM.

Benefits

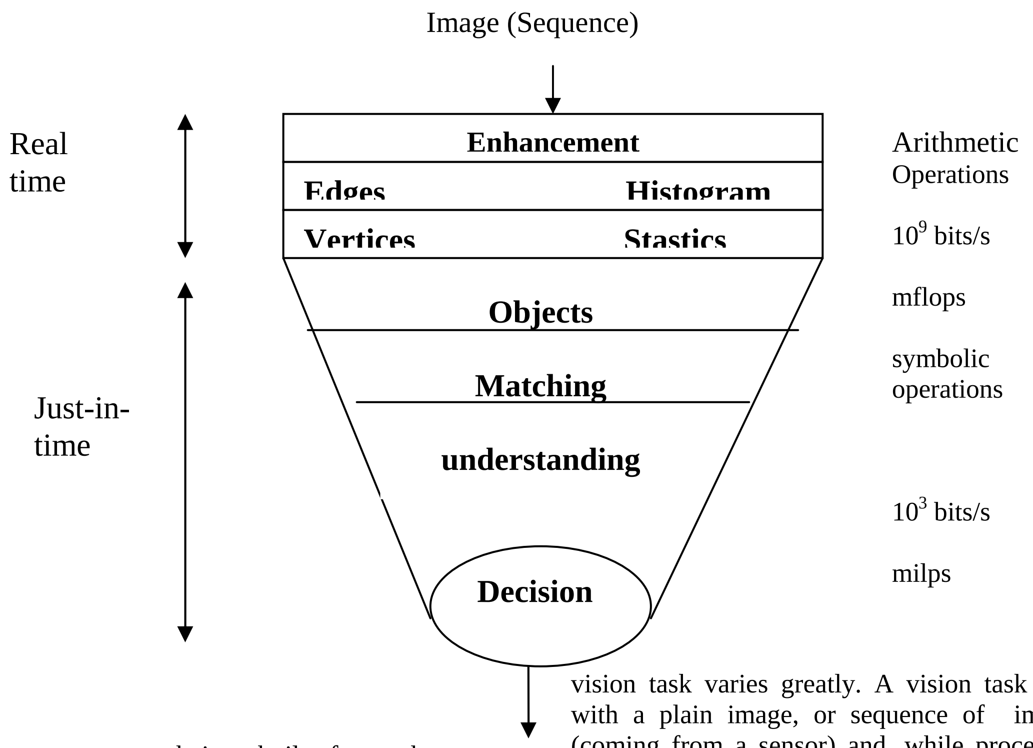
The increasing affordability of HPC is helping in improving price-performance ratio of systems and the emergence and stabilization of cross-platform HPC software standards now means that truly portable applications can be developed to garner the benefits of HPC hardware [16].

HPC can offer solutions to problems in a wide range of business areas, from traditional large-scale engineering to the emerging entertainment markets of the Internet. Some of the applications of HPC are optimisation of industrial processes, computational modelling, online transaction processing, data mining, decision support, complex visualisation and virtual reality[17].

Trends and Future

High performance computing has come of age and is now a stable mature technology. It can be no longer considered as the preserve of computer scientists in research labs, plugging together printed circuit boards and writing new flavours of parallel operating systems[18]. Today, HPC

Figure 2.2: Levels of processing in computer vision.



computers are being built from the same commodity chips found in desktop personal workstations and it is the performance of these commodity chips that will determine the power of tomorrow's HPC systems. “The Grid has emerged as a unifying concept for the future of HPC and no one remotely involved in HPC could have failed to have noticed the recent emergence of the Grid as the dominant buzzword in the field”[19]. It is expected that the Grid will do for HPC what the Web did for computer documents and desktop environment. Metacomputing, which refers to connecting HPC machines rather than individual workstations, is expected to be the next step from high performance parallel computing. In addition to the use of PC and workstation clusters as explicit parallel HPC systems, software tools are emerging to facilitate the automatic management of regular serial applications running over the cluster. These tools schedule applications to run on individual machines in the cluster, monitoring their relative loads and redistributing where necessary to maximise the efficiency of the cluster resources. Such tools are increasingly attractive to firms with more modest computing needs that do not require the raw power or memory of a fully parallel system. Thus we have seen that HPC is a broad combination of both hardware platforms and software techniques and this form of computing is grsadually being adopted widely.

2.2 Image processing levels

The type of processing operations in a typical computer vision task varies greatly. A vision task starts with a plain image, or sequence of images, (coming from a sensor) and, while processing, the type of operations moves from arithmetic (Floating Point Operations Per Second, FLOPS) to symbolic (Million Logic Inferences Per Second, MLIPS) and the amount of data to process is reduced until in the end some decision is made (image understanding).

The initial processing of real-time just-in-time in a computer vision task is real-time as it needs to keep up with the data rate of the incoming data, for example from a Charged-Coupled Device (CCD) camera. The end type of processing leading to the decision and possibly feedback (like steering a robot arm) may be characterized as “just-in-time”. The time, the processing takes is tuned to give a result in time for the specific application. For instance with industrial inspection on product defects, the decision should be ready before the product leaves the conveyor belt.

Generally, three levels of processing can be distinguished [2,3], pictured in Figure 2.2, although the boundaries between these levels are not well defined and sometimes a level is subdivided in sublevels:

1. Low-level operations: Image oriented, these operations work on whole pixel image structures and yield an image, a vector, or a single value.

2. Intermediate-level operations: Symbolic processing, in this the operations work on pixel images and produce symbolic descriptions of the image or features in compact data structures (e.g. a list).

3. High-level operations: Knowledge-based processing, these operations work on symbolic descriptions of the image and lead to decisions in the application, e.g. the understanding of a scene, the understanding of the contents of the image.

3. Programming model for Parallel Image Processing using PVM

This chapter describes the layered hierarchical programming model we defined in the SM-IMP project [19, 20] for computer vision applications. The challenge in defining the layered programming model is to have one programming model that is common to all (or a maximum number of) architectural paradigms. Thus coding a task is independent of the underlying architecture, and a task which is already coded can be placed on a specific unit (automatically or by hand) at compile-time.
The idea behind the hierarchy of programming models is to allow the user to select the selected, using gradually more detailed knowledge of the hardware levels. We propose to organise the programming model as a hierarchy of three (abstraction) layers or levels, illustrated in Figure 3.1: a system expert layer, an image processing layer, and an application programming layer.
The idea is that the user composes his application of code (blocks) implemented at the image

processing level and at the application level decides on which part of the system (e.g. a SIMD component) each code block of the image processing level is going to be run. The application layer is concerned with the specification of composite image processing applications by means of combinations of tasks, which may run in parallel on a heterogeneous system. The image processing layer is concerned with the actual specification of each task of a composite image processing application. A task is specified by means of a program and is

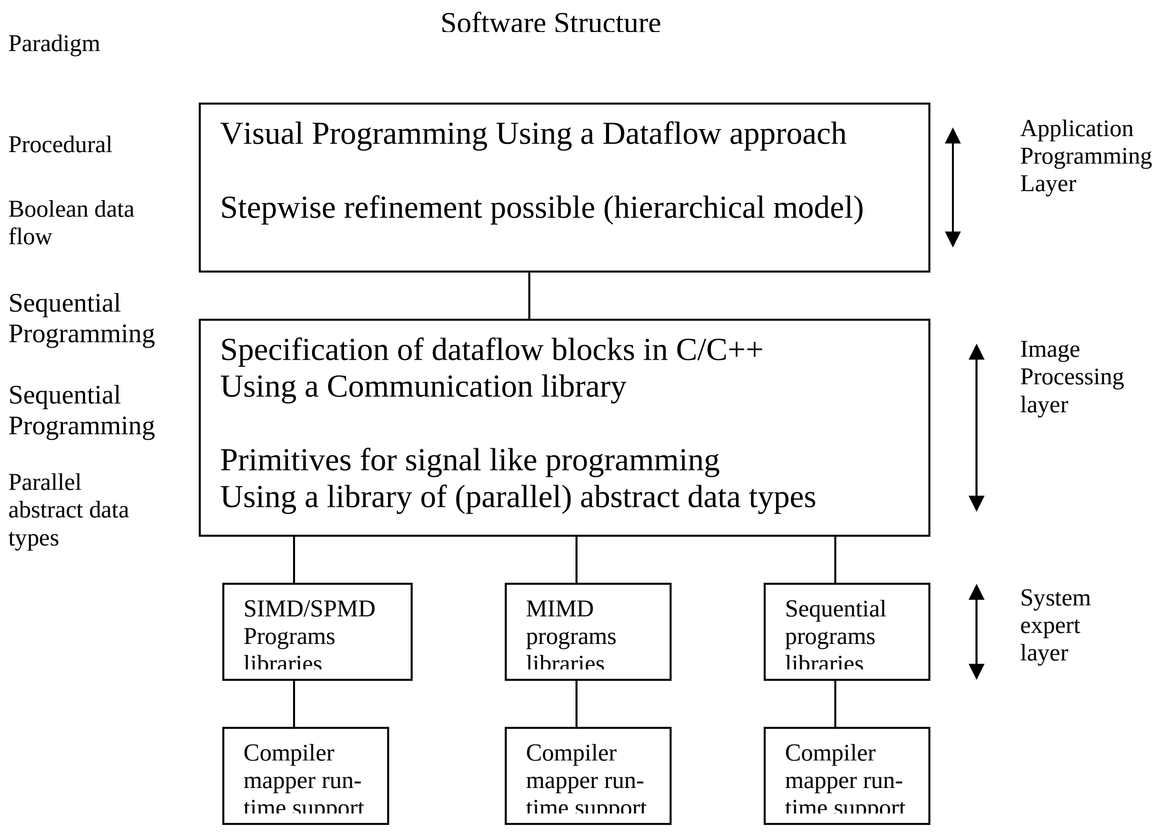
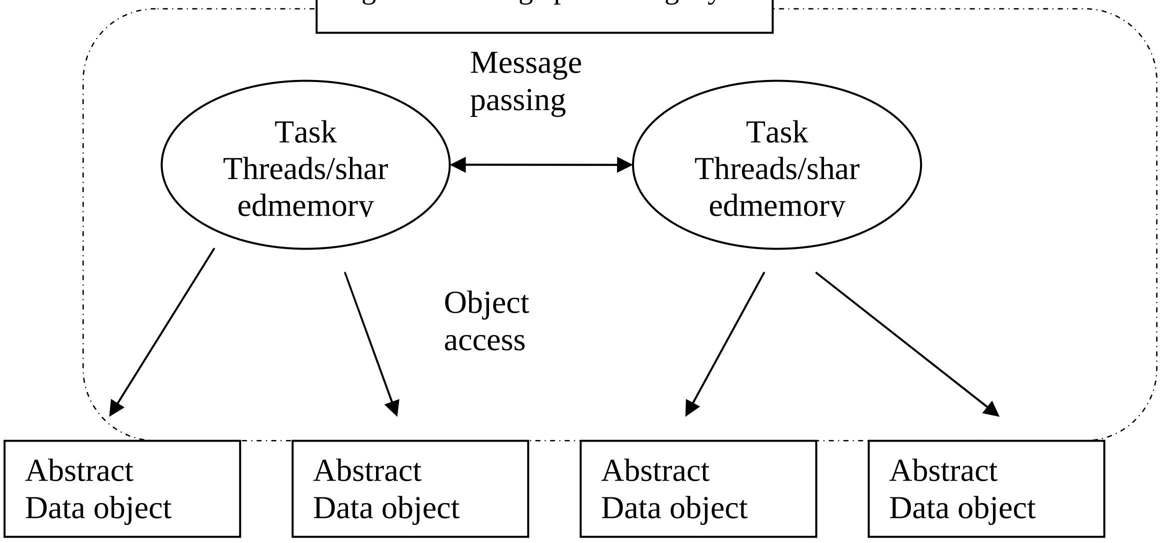

Software Structure
Paradigm
Procedural
Boolean data flow
Sequential Programming
Sequential Programming
Parallel abstract data types
Visual Programming Using a Dataflow approach
Stepwise refinement possible (hierarchical model)
Specification of dataflow blocks in C/C++
Using a Communication library
Primitives for signal like programming
Using a library of (parallel) abstract data types
SIMD/SPMD Programs libraries
MIMD programs libraries
Sequential programs libraries
Compiler mapper run-time support
Compiler mapper run-time support
Compiler mapper run-time support
Application Programming Layer
Image Processing layer
System expert layer

Figure 3.1: SM-IMP layered hierarchical programming model.

abstraction level at which to work. One user may decide, for example, not to take into account the performance of an application while focusing on its functional correctness, and in this case the highest abstraction level is sufficient. On the contrary, when the performance has to be optimized, a lower level of abstraction must be supposed to run on specific nodes of the heterogeneous parallel system, following a single processing paradigm, such as SIMD, or MIMD. The system expert layer is concerned with the improvement of the efficiency of programs by means of machine specific program

manipulations, aimed at taking advantage of the specific features of the host hardware.
Figure 3.2 pictures the image processing layer. The image processing layer program is composed of tasks that communicate with each other using message passing. Each task can consist of multiple threads. Exploitation of these threads can be done by specification of parallel control constructs in the application’s code.
In order to allow analysis and use of different parallel architectures without rewriting an application for each separate architecture, the idea is to program in a data parallel way in combination with a task parallel approach. For implementing a certain application the user can use a set of parallel abstract data types. These types define data structures and the operations that can be performed on them. The types are abstract in the sense that actual implementation mapping of these operations and the internal structure, and possible decomposition/mapping of data on different processors, of the object are shielded from the user.

Figure 3.2: Image processing layer.



The data parallel data structures with appropriate operations (library functions) that can be performed on them, offer a simple way to implement parallel programs. The user just needs to consider the structures as single entities where the operations on such an entity are performed in an efficient, and possibly parallel, manner. So the user need not be concerned with the actual implementation and execution of an operation and the distribution of the data of the structure over the available processors.
Application programming layer:
This layer expresses the behavior of a whole application running on the entire (heterogeneous) machine. At this level the application is constructed in a (hierarchical) data flow pipe-lined manner using the code blocks of the image processing layer. So this layer is associated with composition of code blocks that perform specific functions and with the synchronization of the data streams that flow through these blocks. Also the user can indicate the mapping of the blocks on specific execution paradigms of the system. Obviously a block can only be mapped on a certain paradigm when an implementation for that paradigm exists at the image processing layer.

4. Analysis, Design and Implementation

The parallel computing functionality was developed keeping in view the processor intensive tasks such as performing s-fold cross-validation on a large amount of data in MATLAB. The basic operations that needed to be performed were setting up the server farm which would be an array of active processors running the remote MATLAB engine. Other operations included distribution of data to the remote hosts in the farm, execution of commands on the remote data, fetching of the result after the data had been processed and finally after the tasks were done, shutting down the farm of active processor nodes. The aim was to use the parallel functionality transparently from within the MATLAB environment. This meant that programming would be done using MATLAB M-files which would utilize the parallel functionality in the form of special functions to achieve parallelism.

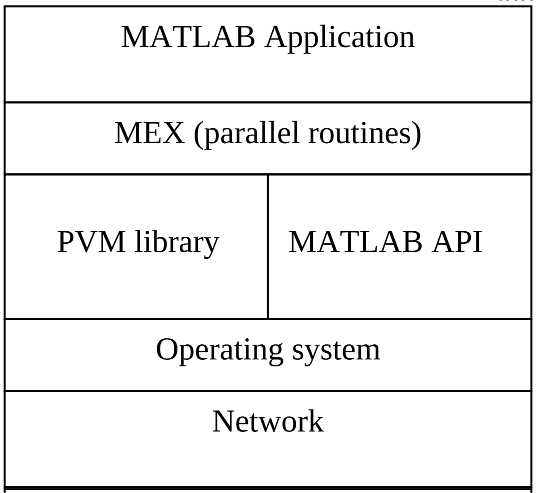


Figure 4.1 Architecture Overview

The Figure 4.1 shows the overview of the architecture stack. The lowest layer is the network layer, which includes the interconnecting physical layer of wires, hubs etc. The layer above the network layer is the operating system layer, which in our case is the Linux operating system. MATLAB and PVM library share the same layer and the functionality exposed by them is used by MEX parallel routines layer, which sits in the next higher layer. The parallel routines developed for the project reside in the MEX layer. On the top most layer is the MATLAB Application layer which in turn uses functionality exposed by the MEX parallel routines layer. All the parallel applications developed will reside in the top most layer and use the functionality of the lower layers.

After having understood the task, the development environment needed to be set up. This involved the choice of the operating system, the message-passing library, the development and debugging tools and languages. A collection of routines was developed based on the concept of Master/Slave paradigm to do parallel programming in MATLAB. Keeping in mind the task of s-fold cross validation, this approach was suitable, as there needed to be minimum inter-slave communication. Data could be distributed to and processed separately by each slave process and results retrieved at the end the master process would start up a desired number of computational slaves either on the same machine (multiprocessor machine) or on other machines (cluster). The slave process would in turn invoke an instance of the MATLAB engine on its processor, thus acting as an interface between the master and the remotely invoked MATLAB engine. Of course, the number of MATLAB licenses available would cap the limit on the number of slaves that could be spawned by the user. Once the remote slaves have been started, data would be distributed to each slave which would put this data into its corresponding MATLAB engine. Now the master would invoke a set of commands via the remote slave processes to process the remotely distributed data. Ideally the set of commands will be written in a MATLAB M-file. So executing the M-file at the remote slave would amount to execution of the desired set of commands. Thus the number of iterations to be performed on the data during the s-fold cross validation can be assigned to a number of slaves spawned on different nodes in the parallel virtual machine. The processing of data, which in this case refers to s-fold cross validation, done in this manner would reduce the computational load at the master node and also arrive at the overall result faster.

The routines for the parallel execution of MATLAB programs have been written using MEX -files, PVM (Parallel Virtual Machine), MATLAB external interface API, and C language [22] on the Linux operating system platform[21].

5. Results and Conclusion

Figure 5.1 shown illustrates the processes analysis obtained for the two implemented systems namely sequentially process system (SPS), parallely process system (PPS). The processes analysis for the two systems is obtained by applying the single input image to the varying number of processes. From the graph obtained it is n that the computational time for the sequentially processing system is more when compared with the parallel processing system

Figure 5.2 shown above illustrates the dimension analysis obtained for the two implemented systems namely sequentially process system (SPS), parallely process system (PPS). The dimension analysis for the two systems is obtained by applying the varying the dimensions of the input images to the single process. From the graph obtained it is seen that the computational time for the sequentially processing system is more when compared with the parallel processing system.

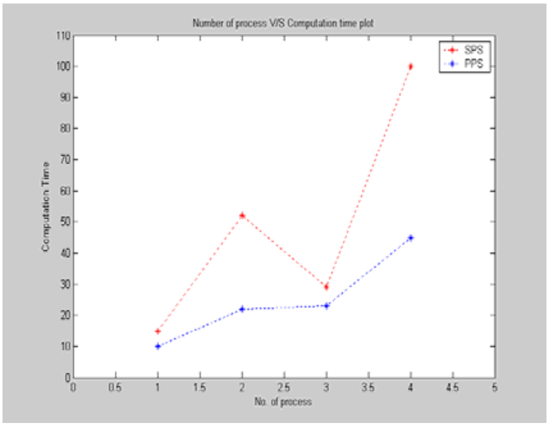


Figure 5.1

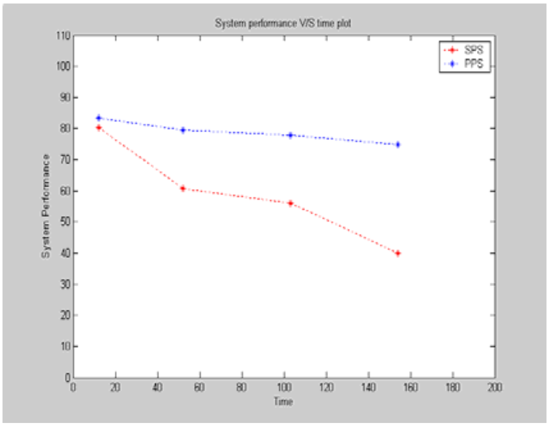


Figure 5.3

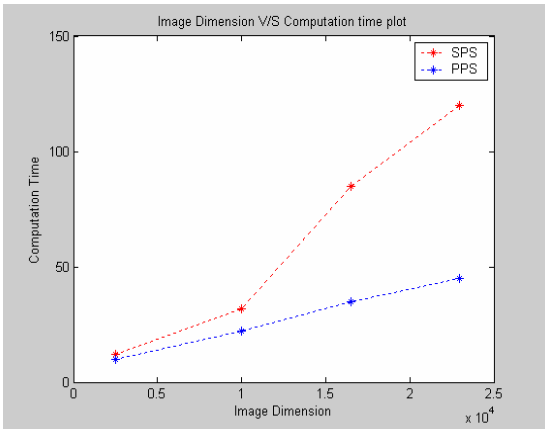


Figure 5.2

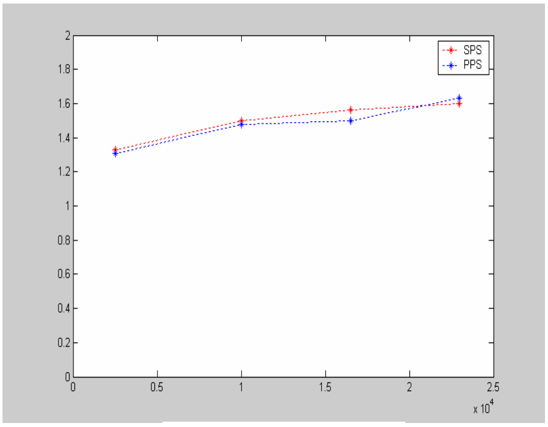


Figure 5.4

Figure5.3 shown above illustrates the system performance level obtained for the two implemented systems namely sequentially process system (SPS), parallely process system (PPS).

The system performance level for the two systems is obtained by varying the image dimensions and the number of the processes.

The performance is obtained

$$\text{Performance} = \frac{(\text{Number of processes})*(\text{total image dimension})}{(\text{Total computation time})}$$

From the graph obtained it is seen that the sequential performance is degraded when compared with the parallel performance

Figure 5.4shown above illustrates the error level obtained for the two implemented systems namely sequentially process system (SPS) ,parallely process system(PPS).

From the g een that the accuracy level for the two system remain almost similar with the variation in the image dimension.

The aim of this paper was to design a programming model for the development of time-constrained image processing applications on currently available parallel architectures, like a cluster of workstations. The goal was to bring the benefits of parallel computing to the image processing community at large, without requiring comprehensive skills needed to write a parallel program. Parallel programming to utilize the latent processing power of idle processors in a network is an interesting area of computing and today this form of computing is being been seen as a means to obtain competitive advantage by cost effective means [Booth et.al.1997]. Clusters of geographically dispersed computers are being connected for enhanced collaboration and resource sharing. There is a strong trend towards the emergence of ‘Grid Computing’..

This paper has attempted a task-oriented approach to parallel computing and functionality was developed with a view of doing s-fold cross validations in parallel. However it is apparent

that the parallel routines developed can be used in other tasks which involve remote distribution, remote execution and remote collection of data. Because of time constraint, complex MATLAB data types such as struct matrix, cell matrix, sparse matrix, objects etc are not handled. The software has been tested to work with double matrices. Thus these are also the areas where functionality can be built to extend the software.
There are some future improvements to be made to our work. One of the issues would be to improve the data redistribution scheme. Data redistribution is critical for implementing a task and data parallel execution scheme. Our implementation was very simple; the master processor gathers the data processed by the allocated processors in a task and sends it to the master processor of the successor task in the graph. It would be more efficient if the processors allocated to a task can send the computed data directly to the processors allocated to the successor task in the graph, as it was proposed in [84]. Another extension would be performing the data dependency analysis of a given image processing application. In our research, we started from the assumption that we already have the information related to data dependencies in the form of the Image Application Task Graph. It only inserts Inter-processor communication when data is missing or outdated on a certain processor. This method would be an excellent tool to replace our simple data redistribution scheme, yielding a system that has to be best of both worlds.

Bibliography

[1]. Benjamin W. Wah, Thomas Huang, Aravind K. Joshi, and Dan Moldovan. Preliminary Report on the Workshop on High Performance Computing and Communication for Grand Challenge Applications: Computer Vision, Natural Language and SpeechProcessing, and Artificial Intelligence, March 1992. Workshop held in Arlington, Virginia,U.S.A., February 21-22, 1992.

[2]. Vipin Chaudhary and J.K. Aggarwal. Parallelism in Computer Vision: a review. In Vipin Kumar, P.S. Gopalakrishnan, and Laveen N. Kanal, editors, *Parallel Algorithms for Machine Intelligence and Vision*, pages 271–309. Springer-Verlag, 1990.

[3]. D. Ballard and C. Brown. *Computer Vision*. Prentice Hall, 1982.[4] W.D.

[4]. Hillis. *The Connection Machine*. MIT Press, 1985.

[5]. K.E. Batcher. Design of a Massively Parallel Processor. *IEEE Transactions on Computers*,C-29(9):836–840, September 1980.

[6]. L.A. Schmitt and S.S. Wilson. The AIS-5000 parallel processor. *IEEE Transactionson Pattern Analysis and Machine Intelligence*, 10(3):320–330, May 1988.

[7]. T.J. Fountain, K.N. Matthews, and M.J. Duff. The CLIP7A Image Processor. *IEEETransactions on Pattern Analysis and Machine Intelligence*, 10(3):310–319, May1988.

[8]. V. Cantoni and S. Levialdi. PAPIA: A Case History. In L. Uhr, editor, *Parallel ComputerVision*, pages 3–13. Academic Press, 1987.

[9]. W.D. Hillis and L.W. Tucker. The CM-5 Connection Machine: A Scalable Supercomputer.*Communications of the ACM*, 36(11):31–40, November 1993.

[10]. V. Michael Bove and John A. Watlington. Experiments in Hardware and Softwarefor Real-Time Image Sequence Processing. In *Proc. IEEE Workshop on Visual SignalProcessing and Communications*, September 1992. Raleigh, NC.

[11]. A. A° stro¨m, P.E. Danielsson, K. Chen, P. Ingelhag, and S. Svensson. Videorate signalprocessing with PASIC and VIP. In *Proc. of Barnaimage '91*, Barcelona, Spain,September 1991.

[12]. H. Miyaguchi, H. Krasawa, and S.Watanabe. Digital TV with Serial Video Processor. In *Proc. of the 9th IEEE International Conference on Consumer Electronics*, Illinois,USA, 1990.

[13]. Y. Fujita, N. Yamashita, and S. Okazaki. A 64 Parallel Integrated Memory Array Processor and a 30 GIPS Real-Time Vision System. In *Proc. of Computer Architectur efor Machine Perception '95*, pages 242–249, Como, Italy, September 1995. IEEE Computer Society Press.

[14]. Ian Foster, Designing and Building Parallel Programs: Concepts and Tools for arallel Software Engineering, Addison-Wesley Publishing Co.,1995.

[15]. A. Geist, A. Beguelin, J. Dongarra, W.Jiang, R. Mancheck, V. Sunderam: PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing,

[16]. Yung-Lin Liu, Hau-Yang Cheng, Chung-Ta King, High performance computing on networks of workstations

[17]. HPC Info, EPCC, 2002.

[18]. Dr Rob Baxter, Whither HPC in Europe? A Strategic Review of High-Performance Computing from a European Perspective, The DIRECT Initiative, EPCC, August 1999

[19]. J.M.Brooke, R.J.Allan, F.Costen, M. Westhead, Grid-based High Performance Computing, 2000.

[20]. J.G.E. Olk. SIMD-MIMD Processor Architectures applied to Image Processing: A Project Overview. In J. van Katwijk, J.J. Gerbrands, M.R. van Steen, and J.F.M. Tonino, editors, *ASCI'95: Proceedings of the first annual conference of the Advanced School for Computing and Imaging, Heijen, The Netherlands*

[21]. M.Mitchell, J.Oldham, A.Samuel, Advanced Linux Programming, New Riders, Inc. 2001

[22]. B.W.Kernighan and D.M.Ritchie, The C Programming