 CHAPTER 5

PARALLEL COMPUTING

Computer vision is a challenging application for high performance computing. Many vision applications are computationally intensive and involve complex processing. For a practical and real-time implementation of vision applications, high performance computing support is essential. Over the past several years, parallel computing has been perceived to be an attractive and economical way to achieve the required level of performance in vision applications. Computational demands and real-time constraints associated with vision applications have induced several research efforts to explore the use of parallel computing resources for parallelizing vision applications [Web 94]. Most vision applications consist of image pre-processing followed by object identification. Although, both these tasks involve heavy computation, they enclose different computational approaches. As a result, several special purpose parallel machines have been proposed, developed and used in implementing parallel solutions for many vision algorithms.

This chapter gives an overview of the parallel computing necessary for computer vision and presents parallel systems and methodologies with focus on the cluster of workstations. It is organized as follows. Section 5.1 introduces some concepts in parallel computing such as parallel computing systems and algorithmic classes. Focus is on performance criteria and application for vision. Section 5.2 ushers in the modern arena of workstation clusters. Cluster configuration, advantages and use of workstation clusters are all elaborated with the final note on parallel computing using clusters.
5.1 Parallel Computing

Parallel computing is concerned with applying multiple processors to solve a single computational problem for achieving better performance. This section begins with an introduction to parallel computing systems. It is followed by a description of abstract algorithmic classes, characterizing different parallel algorithms. These classes are useful when discussing algorithms at a higher level.

5.1.1 Parallel computing systems

A parallel computer is a collection of processors and memory connected by some type of communication network. Parallel computing systems include a full spectrum of sizes and prices, from a collection of workstations attached to a local-area network, to an expensive high-performance machine with thousands of processors connected by high-speed switches [Dun 92].

The architectures of the computing systems are commonly organized in terms of instruction streams and data streams [Fly 72]. The three cases that have become familiar terms to the parallel programmer are SISD (single instruction, single data), SIMD (single instruction, multiple data) and MIMD (multiple instruction, multiple data). SISD computers are the traditional Von Neumann computers that have a single instruction stream and a single data stream. All operations on these computers are logically sequential. In a SIMD parallel computer a single instruction stream is applied to multiple data streams. SIMD-based machines usually consist of a large number of simple processors connected by an interconnection network. The MIMD data model is

\[
\text{Bus} | \text{SIS} | 1362
\]
the most general model of a parallel computer. A MIMD computer has multiple processing elements, each of which is a complete computer in its own right.

Although SIMD systems are easy to program, optimizing SIMD programs to yield acceptable performance is very difficult. As a result, SIMD computers have not been very popular for scientific computing. This makes MIMD systems more acceptable, especially when a cluster of workstations is viewed as a single MIMD computer. A MIMD computer consists of processors and memory. The memory can be either shared or distributed among the processors. We can, therefore, consider two distinct programming models: shared memory MIMD and distributed memory MIMD. However, since the same issues of data locality and concurrency arise in both the cases, we can view MIMD computers in terms of a common programming model. One such model is the co-ordination model [Mat 96], [Fos 95], where a parallel computation is viewed as a collection of distinct processes which interact at discrete points through a co-ordination operation. The term co-ordination refers to the basic operations to control a parallel computer. It includes co-ordination operations for information exchange, process synchronization and process management. These co-ordination operations may vary in speed and structure, however, the overall model is essentially the same.

But, describing parallel and distributed computers in terms of a co-ordination model is not universally accepted like the Von Neumann model [Mat 96]. However, such a model can be stated and used for progressing parallel computers within a universal programming model. Although the computer systems differ, the difference is granularity (ratio of computation to communication) and not the fundamental programming model [Mat 96].

The programming model, in order to be useful, must be implemented as a programming environment. There are several programming environments supporting various forms of the co-ordination model, which run well on parallel computers as well
as a cluster of workstations [Tur 93], [Che 93]. One can develop a parallel code using some high level language designed specifically to support parallel and distributed computing. Alternatively, one can use a sequential language combined with a coordination library (often called as message-passing library) such as MPI or PVM [Snd 90].

Programs written for parallel MIMD systems fall into two categories: SPMD (single program multiple data) and MPMD (multiple program multiple data). For SPMD programs, each processor executes the same object code. SPMD style of programming is easy to code, since the programmer needs to maintain a single source code. In contrast, MPMD programs allow each processor to have a distinct executable code. A programmer can split the program into different modules, which can be developed and debugged independently or re-used as components of other programs. A MPMD program requires less memory as compared to its equivalent SPMD version [Mat 96].

5.1.2 Algorithmic classes

Most of the parallel algorithms can be classified in terms of the regularity of the underlying data structures and the synchronization required as these data elements are updated [Ang 89], [Mat 96]. Based on this classification scheme there are four general classes of parallel algorithms:

1. Synchronous. Synchronous algorithms are those in which regular data elements are updated at regular intervals of time. They are regular in space and regular in time. They involve tightly coupled manipulation of identical data elements. Synchronous algorithms can be expressed in terms of a single instruction stream.
They are therefore, easily mapped onto SIMD computers. The parallelism is usually expressed in terms of the breakdown of the data. In fact, the data drives the parallelism. Hence, the name data parallelism. However, data parallelism is more general than SIMD parallelism, since data parallelism does not insist on a single instruction stream.

2. *Loosely synchronous.* A loosely synchronous algorithm synchronously updates data elements, which differ from one processor to another. Loosely synchronous algorithms are regular in time but, irregular in space. They have a tight coupling between tasks, as in the synchronous case. However, due to variations in the data elements across the processors, the workload can vary from processor to processor. Hence, loosely synchronous algorithms need some mechanism to balance the computational load among the processors of the parallel computer.

3. *Asynchronous.* Algorithms, which are asynchronous, do not have regular data updates. Hence, the system proceeds with non-uniform and sometimes random synchronization. These algorithms are irregular in time and usually, irregular in space, with unpredictable or non-existent coupling between the tasks. This class of problems, other than the parallel subset, described below, is most rare. This is because programs for implementing asynchronous algorithms are difficult to construct. While synchronous and loosely synchronous algorithm are usually parallelized by focussing on data decomposition, asynchronous algorithms are usually parallelized by decomposition of the control, which is referred to as functional or control parallelism.

4. *Embarrassingly parallel.* This class contains those asynchronous algorithms for which the tasks are completely independent and uncoupled. The parallelism in this case is trivial. The programs are among the simplest parallel programs to construct. Problems in this class are very common in parallel computing since
their computations easily map into this model. In fact, any program, consisting of a loop with compute-intensive and independent iterations can be parallelized using this model. Embarrassingly parallel programs usually utilize an SPMD style of programming combined with some mechanism for load balancing. Load balancing schemes can either be static or dynamic.

5.1.3 Performance of parallel programs

The main goal of parallel computing is to reduce the execution time of the whole program in order to produce faster results. The performance estimates of a parallel program are based on the timings of its complete sequential code. The sequential program typically comprises of two distinct sections of code, inherently sequential code and potentially parallel code. The parallel content $p$ of the program is defined as the ratio of the time taken to execute the potentially parallel code to the time taken to execute the entire code. The maximum theoretical speedup that can be achieved for a given program is a function of the parallel content $p$ and the number of utilized processors ($N$). It is given by Amdahl’s law [Amd 88], which is stated as follows:

$$\text{Theoretical speedup} = \frac{1}{(1 + p) + (p/N)} \quad (6.1)$$

The theoretical speedup is lower than the ideal speedup. This reflects the ideal that applying $N$ processors to a program should cause it to complete $N$ times faster. The size of the gap between the ideal and theoretical speedup is a function of the serial content of the program. Hence the suggestion, that the amount of speedup that can be achieved for every program is limited beyond a certain number of processors. The gap between the theoretical and ideal speedup may change due to the increase in
problem size (e.g. when number of iterations are increased in a simulation). The gap narrows down when the parallel content of the program increases due to increase in problem size, while, the gap may actually widen, if, the length of the serial bottlenecks also increase upon increase in problem size. However, the theoretical speedup is rarely achievable by a parallel application. There will actually be an observed speedup, which is much closer than the theoretical speedup, reflecting the effect of external overhead on the total execution. This overhead comes from two sources a) the additional processor cycles expended in simply managing the parallelism b) wasted time spent waiting for I/O communications among processors and competition from the operating system and other users [Pan 96]. The theoretical speedup does not take these factors into consideration.

5.1.4 Parallel systems for vision

Many applications in computer vision have enormous data throughput and processing requirements which have far exceeded the capabilities of uniprocessor architecture. Parallel processing has been perceived as a necessary solution that has led to the conception, design and subsequent analysis of a number of parallel systems for computer vision, listed in the next paragraph. Literature on parallel systems for computer vision is vast [Duf 82], [Ken 82], [Uhr 87], [Pag 88], [Pra 91], [Nar 92] and [Sie 92].

Mesh connected systems [Cho 90], Pyramids [Cho 90], Hypercubes [Cho 90], Shared memory machines, Pipelined systems and Systolic arrays [Yala 94], Partitionable systems [Yala 94] and General purpose parallel systems [Ste 96] are some of the
parallel systems for computer vision. A brief description of the general purpose parallel system is given below.

General purpose parallel systems are the current high-performance parallel machines such as IBM SP-2, Meiko CS-2, Intel Paragon, Cray T3D and Param 10000\(^1\). Since they are based on workstation microprocessor technology, these systems are versatile and cost-effective compared to the other specialized vision systems described earlier. These systems mainly consist of processing units, each with a local memory and a high speed inter-connection network. They are mostly tightly-coupled, i.e. the inter-connects are system-specific with point-to-point links between the processors. Their major disadvantage is that it is difficult for a parallel application to use the resources efficiently. Also, the system specific inter-connects do not provide a flexibility of adding existing machines as hosts. They cannot incorporate heterogenous architectures. Hence, applications cannot select most suitable computing resources for each computation. Therefore, although tightly-coupled systems always support faster communications, their advantage is likely to shrink over time [Ste 96].

5.2 Computing on workstation clusters

During the past several years, network-based computing environments, such as a cluster of workstations have proved to be an attractive alternative for high-performance computing over the conventional parallel machines. This is due to rapid advances in microprocessor technology and emergence of high-speed networks having a network bandwidth of the order of gigabit per second [Bod 95], [Ste 96]. A cluster of

\(^1\)Developed by C-DAC (Centre for Development of Advanced Computing), Pune, India
workstations offers several advantages for implementing high-performance computing solutions. It provides multiple CPUs, large memory, stable software and heterogenous computing environments for developing high-performance computing solutions for many computation-intensive problems. It is believed that the future computing environments will slowly migrate towards the concept that ‘the network is the computer’ [Tur 96].

5.2.1 Cluster configuration

A workstation cluster is basically a collection of workstations connected by a commodity high speed network, such as Fast Ethernet or ATM. The most common network topology employed with workstation clusters is shown in Fig.5.1. Switch based interconnects are typically configured in a star arrangement used exclusively with dedicated clusters. There are also hierarchical designs where multiple types of interconnects are utilized.

The workstations in a cluster communicate with each other by exchanging messages or data packets transmitted using either transmission control protocol (TCP) or user datagram protocol (UDP). The former is a connection-oriented protocol and processes streams of data such that the reliability of message delivery is assured. The latter, being a connection protocol, sends data packets that are attempted to be delivered (i.e. reliability of message delivery is not assured [Tur 96]. Two software methods are used for communicating the messages: message passing and distributed shared memory. Message passing involves explicit transmitting of messages between the systems. Distributed shared memory (DSM), which is usually implemented using message passing involves accessing of data without concern for the physical location.
Workstation clusters have one obvious limitation due to the use of relatively slow network interconnections. The interconnects have a low bandwidth and a high latency, where, bandwidth refers to the speed at which message data is transmitted and latency is the time spent in initiating the transmission of a message. Fast Ethernet, the most commonly implemented network for today’s clusters, transmits information at 100 Mb/s and has a message latency of 3 μs. There are also other high speed interconnecting technologies, which can help in building clusters such as ATM OC-12 (622 Mb/s) and Gigabit Fast Ethernet (1000 Mb/sec) [Tur 96].

The need to maximize the network performance (high bandwidth and low latency), particularly for parallel applications, has yielded unique solutions. A recent example of one such network is Myrinet [Bod 95]. It consists of a collection of workstations, the network comprising links and switches to route the data and the network interface between the workstations and the network links. The network interface consisting of a special processor can transfer blocks of data to handle overlap of computation and communication.

Workstation clusters are simple to configure. However, it is important to identify categories of applications which can be implemented on these systems more
effectively. The applications which require computational capabilities of high-performance computing systems can be categorized as follows [Tur 96]:

1) Capability demand which includes megaproblems that require all the computational capabilities of any available system including memory and CPU. Grand Challenge applications which require massive parallel processing fall into this category.

2) Capacity demand which includes applications requiring substantial, but far from ultimate performance, whilst making moderate demands on memory. These jobs are ideal candidates for workstation clusters. Workstation clusters provide practical, cost-effective computing solutions for the capacity demand problems. They provide complementary rather than practical replacements for the general purpose parallel computing machines.

5.2.2 Advantages of workstation clusters

Workstation clusters offer several advantages over the traditional parallel computing environments [Tur 96] as described below:

1) Workstation clusters provide simple, inexpensive and readily accessible computing platform to design, develop and implement parallel solutions for a wide range of applications. They offer excellent price/performance benefits in comparison with the traditional parallel computing solutions.

2) Workstations provide large, cost-effective memory, which is not available in most traditional parallel computers. As problems continue to grow in complexity and detail, availability of a large memory is, as important as, the processor speed.
3) Workstation clusters offer a stable software environment as compared to dedicated parallel machines. Software environments such as operating systems, compilers, libraries and software tools are yet to develop to a point of general acceptability for dedicated parallel algorithms.

4) Clusters provide a cost-effective environment to study topics related to heterogenous computing. It is generally believed that, in future, high performance computing systems will achieve maximum performance capabilities only by exploiting the benefits of a heterogenous computing environment.

5) Clusters have a graceful degradability. The entire cluster is not lost due to failure of a single system in the cluster. In addition, since the clusters are created using commodity components, maintenance costs are usually much less than for an equivalent investment in a dedicated parallel computer.

6) The workstations used in clusters are commodity systems (commonly available) and can be implemented in heterogeneous workstation platforms.

### 5.2.3 Use of clusters

Clusters can be used as *enterprise* clusters or *dedicated* clusters [Tur 96]. Enterprise clusters are configured with workstations that are owned by different individuals or groups. The machines in this cluster are normally heterogenous (multivendor) and are almost exclusively connected via Ethernet (or Fast Ethernet in newer environments). This type of clustering relies on individual owners contributing their unused computing cycles to a shared pool. The individual owners expect to receive more resources than they contribute. Enterprise clusters are controlled and governed by a management software. This software enables effective use of collective idle time available on most
workstations. This idle time can be used to process jobs of several different users in the group. The management software ensures that the systems of individual owners are not saturated when they try to sue their own systems. The individual owners can specify how their systems will participate in the resource pool.

Several papers have proposed different schemes for sharing resources in enterprise clusters, where the main idea is to identify idle machines in the network and schedule background jobs on them with minimum disruption to the individual owners of the machines. When the owner resumes activity at a workstation, the job is either suspended, terminated or moved to another machine in the cluster. These efforts have resulted in either speeding individual jobs or programs by locating idle resources [Alo 88], [Mut 87] or in simply achieving higher levels of machine utilization through load balancing or load sharing [The 88], [Lit 88], [Tnd 96], [Cla 92].

Dedicated clusters are installed as substitutes or replacements to the traditional parallel computing systems. They consist of individual workstations managed by a single group, which administers the clusters like a central mainframe. They are usually interconnected by high-speed networks such as FDDI (Fibre Distributed Data Interface), and HiPPI (High Performance Parallel Interface) [Tur 96]. Dedicated clusters usually have a control workstation, which manages the job queue and acts as an interface to the remaining clusters. The control system can be used to dynamically partition the clusters for execution of interactive jobs (e.g. code development, graphics, etc.), serial batch jobs and jobs that have been parallelized.

5.2.4 Parallel computing using Clusters

Workstation clusters, both enterprise and dedicated, can be used as parallel computing environments for implementing parallel solutions for a wide range of
applications. There have been several papers, which have addressed the issues involved in solving a single problem on a collection of workstations. Silverman and Stuart [Sil 89] have used the cluster as a loosely coupled message passing parallel computer to solve some asynchronous algorithms in numerical analysis. Magee and Cheung [Mag 91] have proposed a supervisor-worker-programming model to distribute computations over a set of workstations.

Atallah, et al [Ata 92] have proposed a resource management technique called co-scheduling or gang scheduling. It involves dividing a large task into subtasks. These are scheduled to execute concurrently on a set of workstations. The subtasks need to coordinate their execution by starting at the same time and computing at the same pace. Wang and Blum [Wan 96] have developed a small message passing library to implement iterative numerical algorithms, which require synchronization at the end of each iteration (synchronous algorithms). Finally, there are attempts to demonstrate the power of workstations clusters to solve grand challenge problems [Beg 91], [Nak 92].

Two commonly used approaches to parallelize applications using clusters are:

1) Extension of existing sequential languages (e.g. C++, FORTRAN) to handle necessary communications and synchronization [Wls 96].

2) Defining new programming languages or an environment based on object-oriented, functional or logical paradigms.

Express, Linda, p4, PVM (Parallel Virtual Machine) and MPI (Message Passing Interface) are software, which create parallel applications on workstation clusters. A comprehensive review of these systems is contained in [Tur 93]. The next chapter discusses characteristics of the Message Passing Interface (MPI), the programming environment and the Open Frame architecture, utiized in the implementation of this thesis.