# 1

# Parallel Computing at a Glance

It is now clear that silicon based processor chips are reaching their physical limits in processing speed, as they are constrained by the speed of electricity, light, and certain thermodynamic laws. A viable solution to overcome this limitation is to connect multiple processors working in coordination with each other to solve grand challenge problems. Hence, high performance computing requires the use of Massively Parallel Processing (MPP) systems containing thousands of powerful CPUs. A dominant representative computing system (hardware) built using MPP approach is C-DAC's PARAM supercomputer.

By the end of this century, all high performance systems will be parallel computer systems. High-end super computers will be the Massively Parallel Processing (MPP) systems having thousands of processors interconnected. To perform well, these parallel systems require an operating system radically different from current ones. Most researchers in the field of operating systems (including PARAS microkernel designers!) have found that these new operating systems will have to be much smaller than traditional ones to achieve the efficiency and flexibility needed. The solution appears to be to have a new kind of OS that is effectively a compromise between having no OS at all and having a large monolithic OS that does many things that are not needed. At the heart of this approach is a tiny operating system core called a microkernel. Dominant representative operating systems built using microkernel approach are Mach and C-DAC's PARAS microkernel.

This chapter presents an overview of parallel computing in general and correlates all those concepts to the PARAM and PARAS advented by the Centre for Development of Advanced Computing (C-DAC). It starts with the discussion on need of parallel systems for High Performance Computing and Communication (HPCC). It also presents an overview of PARAM family of supercomputers with the PARAS operating environment for respective representative systems. Thus, it brings out the four important elements of computing: hardware architectures, system software, applications, and problem solving environments.

## 1.1 History Parallel Computing

The history of parallel processing can be traced back to a tablet dated around 100 BC. Tablet had three calculating positions capable of operating simultaneously. From this, we can infer that, these multiple positions were aimed either at providing reliability or high speed computation through parallelism. Just as we learned to fly, not by constructing machine that flap their wings like birds, but by applying aerodynamic principles demonstrated by nature; we modeled parallel processing after these biological species. The feasibility of parallel processing can be demonstrated by neurons in brain. Aggregate speed with which complex calculations carried out by neurons is tremendously high, even though individual response of neurons is too slow (in terms of milli seconds).

### Eras of Computing

The most prominent two eras of computing are: sequential and parallel era. In the past decade, parallel machines have become significant competitors to vector machines in the quest for high performance computing. A century wide view of development of computing eras is shown in Figure 1.1. The computing era starts with a development in hardware architectures, followed by system software (particularly in the area of compilers and operating systems), applications, and reaching its saturation point with its growth in problem solving environments. Every element of computing undergoes three phases: R & D, commercialization, and commodity.

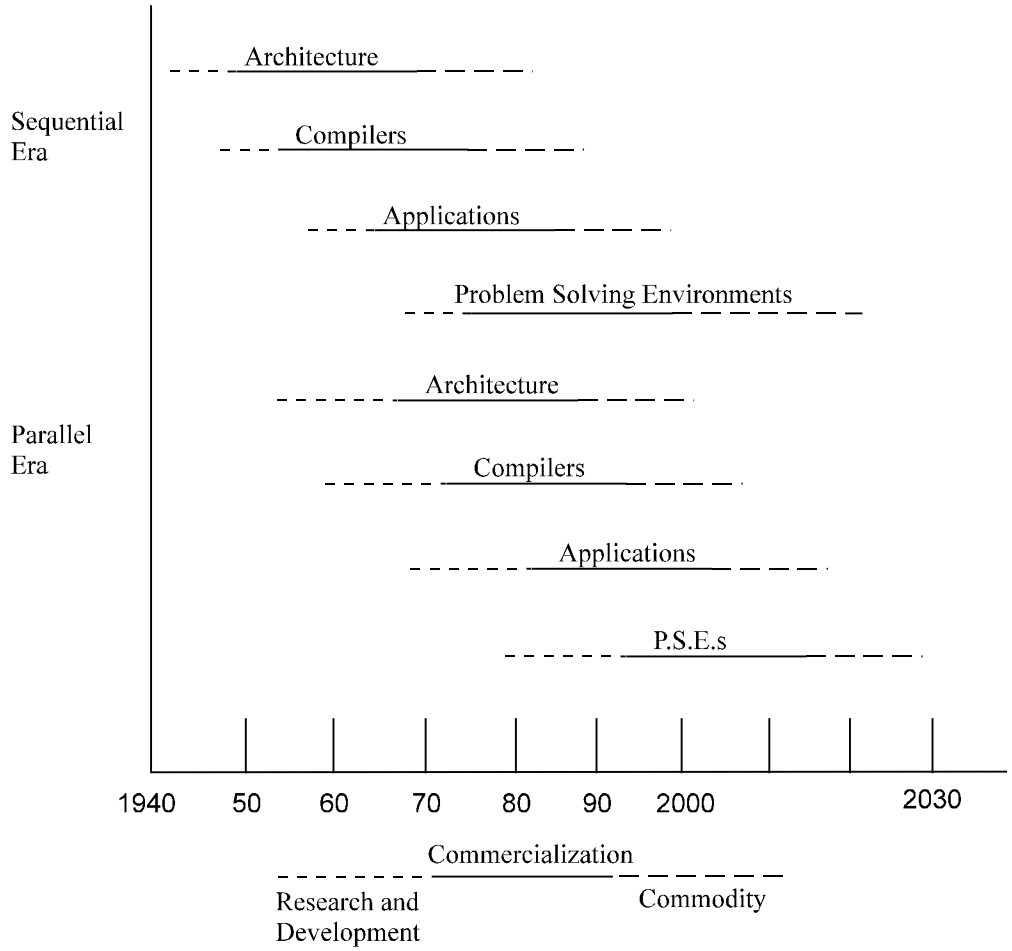


**Figure 1.1: Two Eras of Computing**

## 1.2 What is Parallel Processing ?

Processing of multiple tasks simultaneously on multiple processors is called *parallel processing*. The parallel program consists of multiple active *processes* simultaneously solving a given problem. A given task is divided into multiple subtasks using *divide-and-conquer* technique and each one of them are

processed on different CPUs. Programming on multiprocessor system using divide-and-conquer technique is called *parallel programming*.

## 1.3 Why Parallel Processing ?

Many applications today require more computing power than a traditional sequential computer can offer. Parallel processing provides a cost-effective solution to this problem by increasing the number of CPUs in a computer and by adding an efficient communication system between them. The work-load can now be shared between different processors. This results in much higher computing power, performance than could be achieved with traditional single processor system.

The development of parallel processing is being influenced by many factors. The prominent among them include the following:

- Computational requirements are ever increasing, both in the area of scientific and business computing. The technical computing problems, which require high speed computational power are related to life sciences, aerospace, geographical information systems, mechanical design and analysis, etc.
- Sequential architectures reaching physical limitation as they are constrained by the speed of light and thermodynamics laws. Speed with which sequential CPU's can operate is reaching saturation point (no more vertical growth), and hence an alternative way to get high computational speed is to connect multiple CPU's (opportunity for horizontal growth).
- Hardware improvements in pipelining, superscalar, etc., are non-scalable and requires sophisticated compiler technology. Developing such compiler technology is difficult task.
- Vector processing works well for certain kind of problems. It is suitable for only scientific problems (involving lots of matrix operations). It is not useful to other areas such as database.
- The technology of parallel processing is mature and can be exploited commercially; there is already significant research and development (R & D) work on development tools and environment is achieved.
- Significant development in networking technology is paving a way for heterogeneous computing.

## 1.4 Hardware Architectures for Parallel Processing

The core elements of parallel processing are CPUs. Based on a number of instruction and data streams that can be processed simultaneously, computer systems are classified into the following four categories:

1. Single Instruction Single Data (SISD)
2. Single Instruction Multiple Data (SIMD)
3. Multiple Instruction Single Data (MISD)
4. Multiple Instruction Multiple Data (MIMD)

### Single Instruction Single Data (SISD)

A SISD computing system is a uniprocessor machine capable of executing a single instruction which operates on a single data stream (see Figure 1.2). In SISD machine instructions are processed sequentially and hence computers adopting this model are popularly called sequential computers. Most of conventional computers are built using SISD model. All the instructions and data to be processed have to be stored in the primary memory. The speed of processing element in SISD model is limited by the rate at which computer can transfer information internally. Dominant representative SISD systems are IBM-PC, Macintosh, Workstations, etc.

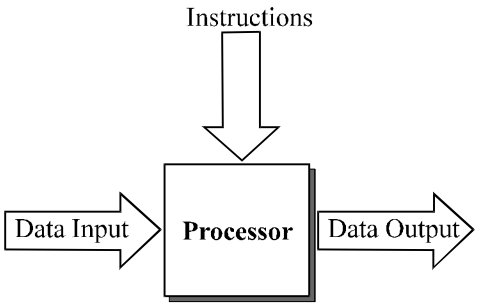


**Figure 1.2: SISD Architecture**

### Single Instruction Multiple Data (SIMD)

A SIMD computing system is a multiprocessor machine capable of executing the same instruction on all the CPUs, but operating on different data streams (see Figure 1.3). Machines based on SIMD model are well suited for scientific computing since they involve lots on vector and matrix operations. For instance, statements such as

□␣□□□□␣□□□□␣□

can be passed to all the PEs (processing elements); and organized data elements of vector A and B into multiple sets (N-sets for N PE systems); and trigger each PE to process one data set. Dominant representative SIMD systems are CRAY's vector processing machine, Thinking Machines's cm*, etc.

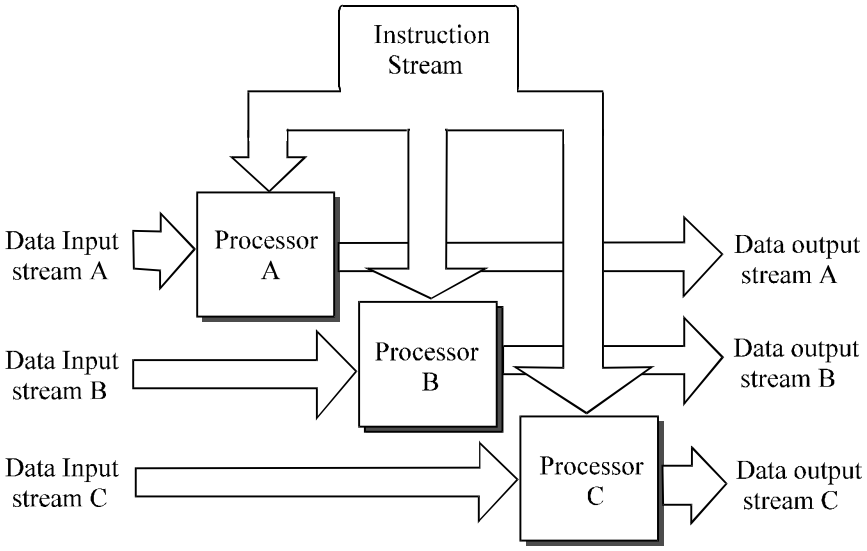


**Figure 1.3: SIMD Architecture**

### Multiple Instruction Single Data (MISD)

A MISD computing system is a multiprocessor machine capable of executing different instructions on

different PEs, but all of them operating on the same data-set (see Figure 1.4). For instance, statements such as

□□□□□□□□□□□□□□

perform different operations on the same data set. Machines built using MISD model, are not useful in most of the applications; a few machines are built, but none of them are available commercially. They become more of an intellectual exercise than a practical configuration.

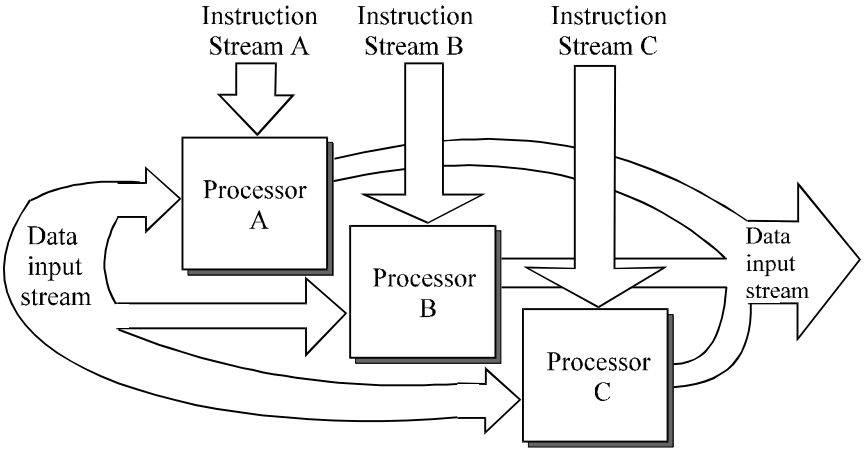


**Figure 1.4: MISD architecture**

### Multiple Instruction Multiple Data (MIMD)

A MIMD computing system is a multiprocessor machine capable of executing multiple instructions on multiple data sets (see Figure 1.5). Each PEs in MIMD model have separate instruction and data stream and hence machines built using this model are well suited for any kind of applications. Unlike SIMD and MISD machines, PEs in MIMD machines work asynchronously.

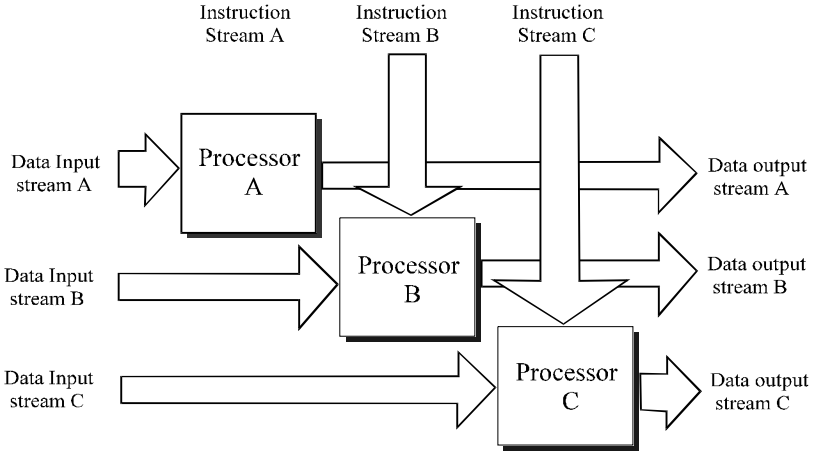


**Figure 1.5: MIMD architecture**

MIMD machines are broadly categorized into shared-memory MIMD and distributed-memory MIMD machines based on how all the PEs are coupled to the main memory.

### Shared Memory MIMD Machine

In shared memory MIMD model, all the PEs are connected to a single global memory; all the PEs have access to this global memory (see Figure 1.6); also called as the tightly-coupled multiprocessor system. The communication between PEs in this model, takes place through the shared memory; modification of the data stored in global memory by one PE is visible to all other PEs. Dominant representative shared-memory MIMD systems are Silicon Graphics machines, Sun's SMP's (Symmetric Multi-Processing), BARC's Anupam, etc.

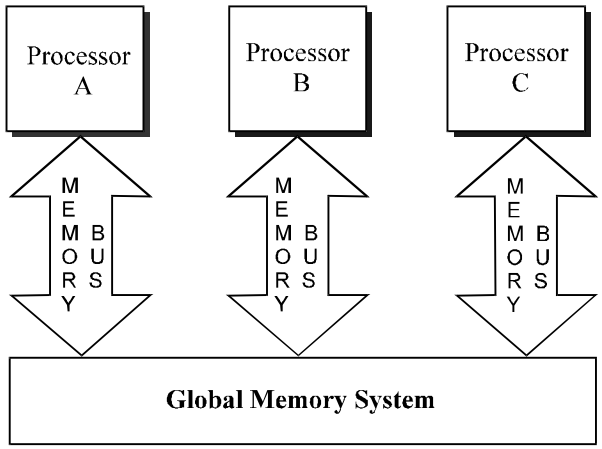


**Figure 1.6: Shared-Memory MIMD Architecture**

The following are the characteristics of shared-memory MIMD machines:

- **Manufacturability:** Easy to build; a well established conventional operating systems can be easily adapted.
- **Programmability:** Easy to program and it does not involve much communication overhead during the communication between the programs executing simultaneously.
- **Reliability:** Failure of a memory component or any processing element affects the whole system.
- **Extensibility and Scalability:** Adding more PEs to the existing design (system) is very difficult since it leads to memory contention.

### Distributed Memory MIMD machine

In distributed memory MIMD model, all the PEs have their own local memory (see Figure 1.7); also called as the loosely-coupled multiprocessor system. The communication between PEs in this model, takes place through the interconnection network (IPC-inter-process communication channel). The network connecting processing elements can be configured to tree, mesh, cube, etc.

Each and every PEs operate asynchronously and if communication/synchronization among tasks is necessary, they can do so by communicating messages between them. Dominant representative distributed-memory MIMD systems are C-DAC's PARAM, IBM's SP/2, Intel's Paragaon, etc.

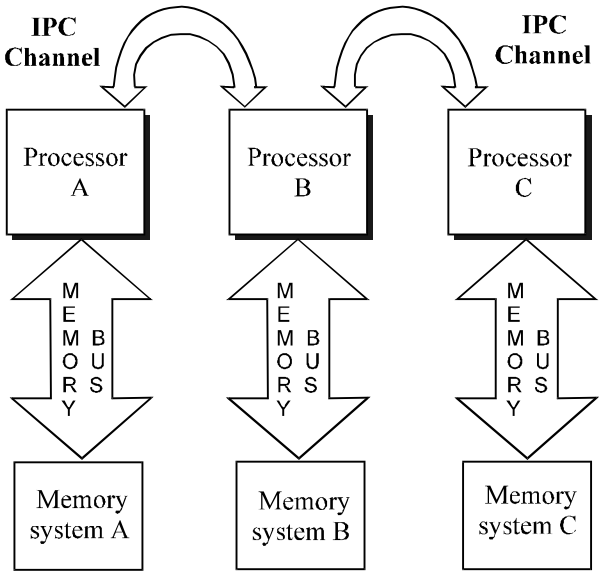


**Figure 1.7: Distributed MIMD Architecture**

The following are the characteristics of distributed memory MIMD machines:

- **Manufacturability:** Easy to build, but it requires light-weight operating system which consumes little system resources.
- **Programmability:** Slightly difficult when compared to shared-memory, but it is well suited for real-time applications.
- **Reliability:** Failure of any component will not affect the entire system; since any PE can be easily isolated.
- **Extensibility and Scalability:** Adding more PEs to the existing design (system) is much easier.

The shared-memory MIMD model, is easy to program but suffers from extensibility and scalability and the distributed memory MIMD model difficult to program but it can easily be scaled and hence, such systems are popularly called massively parallel processing (MPP) systems.

The PARAM series of parallel supercomputers adopts distributed memory multiprocessor technology. All the processing elements are connected by a high speed network and communication among them takes places through message passing interfaces supported on it.

## 1.5 Approaches to Parallel Programming

A sequential program is one which runs on a single processor and has a single line of control. To make many processors collectively work on a single program, the program must be divided into smaller independent chunks so that each processor can work on separate chunks of the problem. The program decomposed in this way is a parallel program.

A wide variety of parallel programming approaches are available. The most prominent among them

supported on PARAM are the following:

- Data Parallelism
- Process Parallelism
- Farmer and Worker Model

All these three models are suitable for task level parallelism. In case of data parallelism, divide-and-conquer technique is used to split data into multiple sets and each data set are processed on different PEs by using the same instruction. This approach is highly suitable for processing on machines based on SIMD model. In case of process parallelism, a given operation has multiple (but distinct) activities, which can be processed on multiple processors. In case of farmer and worker model, job distribution approach is used; one processor is configured as master and all other remaining PEs are designated as slaves; master assigns job to slave PEs and they on completion informs the master which in turn collects results. The above approaches can be utilized in different levels of parallelism (discussed later).

## 1.6 PARAM Supercomputers

The PARAM is an acronym for PARAllel Machine. The PARAM super computer is a distributed memory, message passing parallel computer. The following are the PARAM family of supercomputers designed by C-DAC:

| Supercomputer | Processor in Compute Engine | Philosophy of Design |
|---|---|---|
| PARAM 8000 | INMOS Transputer | MPP |
| PARAM 8600 | i860 | MPP |
| PARAM 9000 | Sun's SPARC, Alpha, PowePC (MPP) | MPP and Cluster |
| PARAM OpenFrame | Sun's UltraSPARC | Cluster |

**Table 1.1: PARAM Family**

The PARAM 8000 is based on the transputers both as compute and service node. The PARAM 8600 is based on the i860 RISC processor as compute node and transputer as a service node. The PARAM 9000 is based on the SPARC series or POWER PC or Digital Alpha processors working both as a compute and service nodes. Unlike PARAM 8000 and 8600, PARAM 9000 supports both MPP personality and Cluster personality. All PARAM supercomputers work as a back-end compute engine to hosts such as PC/AT, SUN workstations, Micro VAX machines, and U6000 machines etc. They provides multi-user facility by partitioning back-end compute nodes into one or more logically disjoint fragments and assigning them to users.

The PARAM 9000 is a multifaceted product. It supports both the cluster computing and MPP computing. In both the personality, the basic hardware of PARAM 9000 remains the same. However, they differ only in terms of operating environment and their configuration in terms of software. Architecture of PARAM 9000 machine will be discussed in later chapters.

## 1.7 PARAS Operating Environment

The PARAS is not just a microkernel, it is a complete parallel programming environment for C-DAC PARAM called *PARAS operating environment*. At the lowest level, the parallel programming model

offered by the PARAS microkernel is one of kernel-level threads. PARAS offers constructs for tasks, threads, memory management services, and synchronous/asynchronous communication between the threads of different tasks or the same task.

### PARAS - Parallel Programming Environment

PARAS is a comprehensive parallel programming environment that has been developed for PARAM and similar class of message passing parallel computers. It comprises of the following:

- OS kernel
- host servers
- compilers
- run-time environment
- parallel file system
- on-line debugger and profiling tool
- graphics and visualization support
- networking interface
- off-line parallel processing tools
- program restructurers
- libraries

PARAS is an advanced parallel programming environment that provides comprehensive support for the complex task of developing and executing parallel programs for MIMD message passing machines. PARAS is aimed at a host/backend hardware model and provides an environment that efficiently harnesses the power of parallel processing offered by distributed memory, message passing machines such as PARAM. Written in C, PARAS is designed for easy portability across hardware platforms.

The various components of PARAS can be broadly classified into:

- Program development environment
- Program run-time environment
- Utilities

### PARAS Programming Model

The PARAS operating system environment provides a new programming model that is well suited for developing parallel programs for high-performance computing on massively parallel systems. It provides the user with the model of a coherent computer system supporting virtually indeterminate number of physical nodes in a reliable, network transparent fashion. The principle abstractions are tasks, threads, messages, ports, and regions. These are enhanced by other abstractions such as port groups, multicast, multimode services. The PARAS program development environment includes the following.

1. PARAS Microkernel
2. COncurrent Runtime Environment (CORE)
3. POSIX threads Interface
4. Popular Message Passing interfaces such as
    - MPI (Message Passing Interface)
    - PVM (Parallel Virtual Machine)
5. Parallelizing Compilers
6. Tools and Debuggers for Parallel Programming
7. Load balancing and distribution tools

The detailed discussion on various tools available on PARAS is beyond the scope of this and hence, it is omitted.

The PARAS microkernel is one of a prime component in the PARAS operating environment. It is highly optimized for high performance computing. It is available on all PARAM family of supercomputers. It allows seamless migration of application from early systems to latest systems. More detailed discussion on the PARAS microkernel can be found in later chapters.

## 1.8 Levels of Parallelism

Levels of parallelism decided based on the lumps of code (grain size) that can be a potential candidate for parallelism. Table 1.2 lists categories of code granularity for parallelism.

| Grain Size | Code Item | Comments/parallelized by |
|---|---|---|
| Large | Program-Separate heavyweight process | Programmer |
| Medium | Standard One Page Function | Programmer |
| Fine | Loop/Instruction block | Parallelizing compiler |
| Very fine | Instruction | Processor |

**Table 1.2: Levels of Parallelism**

All of forgoing approaches have a common goal to boost processor efficiency by hiding latency. To conceal latency through, there must be another thread ready to run whenever a lengthy operation occurs. The idea is to execute concurrently two or more single-threaded application, such as compiling, text formatting, database searching, and device simulation.

Parallelism in an application can be detected at several level. They are

- Large-grain (or task-level)
- Medium-grain (or control-level)
- Fine-grain (data-level)
- Very-fine grain (multiple instruction issue)

The different levels of parallelism is depicted in Figure 1.8.

Among the four levels of parallelism, the PARAM supports medium and large grain parallelism explicitly. However instruction level of parallelism is supported by the processor used in building compute engine of the PARAM. For instance, the compute engine in PARAM 8600 is based on i860 processor having capability to execute multiple instructions concurrently.

A programmer can use PARAS programming environment for the parallelization of an application. A basic thread level and task level programming on PARAM is supported by the PARAS microkernel in the form of primitive services. Much sophisticated programming environment is built using the microkernel services in the form of subsystem. Some of the prominent and powerful subsystems built are CORE, MPI, POSIX threads, and port group communication systems.

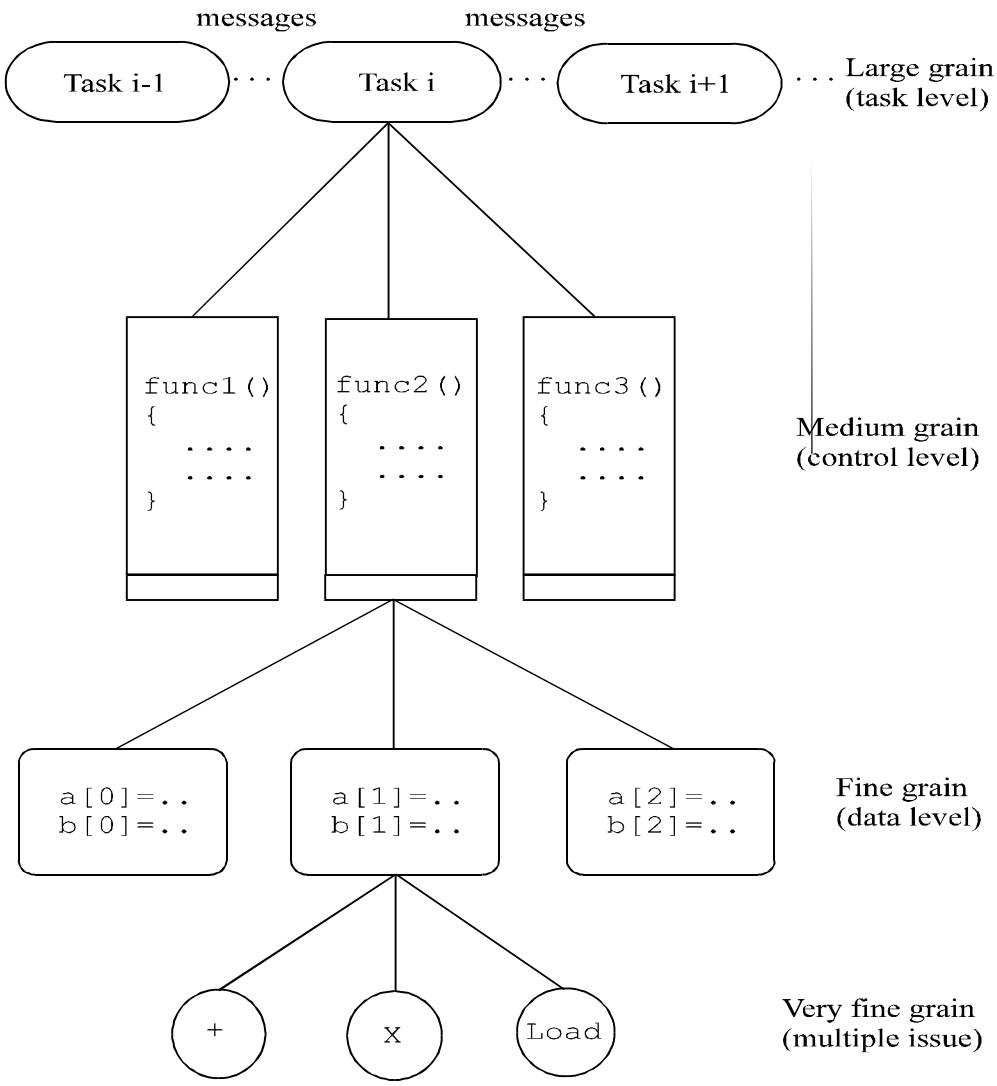


**Figure 1.8: Detecting Parallelism**

### Thread Level Programming

Threads are an emerging model for expressing concurrency on multiprocessor and multicomputer systems. In multiprocessors, threads are primarily used to simultaneously utilize all the available processors, whereas in uniprocessor or multicomputer system, threads are used to utilize system resources effectively by exploiting the asynchronous behavior (opportunity for computation and communication overlap) of threads. More details on threads can be found in the Process Management chapter.

### Task Level Programming

PARAM as a MIMD distributed memory machine, offers a wide variety of interfaces for task level parallelism. They include CORE (COncurrent Runtime Environment), MPI (Message Passing Interface), PVM (Parallel Virtual Machine). CORE is a custom built interface whereas MPI is the standard interface, which is available on most of the modern parallel supercomputers. The various primitives offered by them include task creation, deletion, control, and communication. They offer both the synchronous and asynchronous mode of communication.

## 1.9 Laws of Caution

For a given ***n*** processors, the user expects speed to be increased by ***n*** times. It is an ideal situation, which never happens because of communication overhead. Here are a few laws of caution.

1. Speed of computation is proportional to the square root of system cost; they never increase linearly as shown in Figure 1.9.

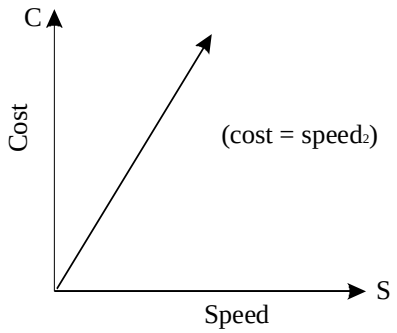


**Figure 1.9: Cost vs. Speed tradeoff in Parallel Systems**

2. Speedup by a parallel computer increases as the logarithm of the number of processors; i.e., □□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□. It is shown in Figure 1.10.

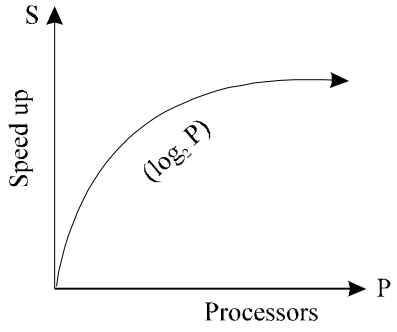


**Figure 1.10: Processors vs. Speedup tradeoff**

3. Very fast development in parallel processing and related area have blurred concept boundaries, causing lot of terminological confusion.

4. Even well-defined distinctions like shared memory and distributed memory are merging due to new advances in technology.
5. Good environments for developments and debugging are yet to emerge.
6. There is no strict delimiters for contributors to the area of parallel processing. Hence, computer architects, OS designers, language designers, computer network designers, all have a role to play.

It is hard to imagine a field that changes as rapidly as computing. Latest developments in the area of software have blurred the concept of programming on shared memory and distributed memory multiprocessor systems. Because of lack of taxonomy and terminologies, computer science can be called as immature science.