Parallel Computing

A Key to Performance

Dheeraj Bhardwaj

Department of Computer Science & Engineering

Indian Institute of Technology, Delhi –110 016 India

http://www.cse.iitd.ac.in/~dheerajb

Introduction

- **Traditional Science**

  - **Observation**
  - **Theory**
  - **Experiment -- Most expensive**

- **Experiment can be replaced with Computers Simulation - Third Pillar of Science**

Introduction

- **If your Applications need more computing power than a sequential computer can provide ! ! !**

† **Desire and prospect for greater performance**

  - **You might suggest to improve the operating speed of processors and other components.**

- **We do not disagree with your suggestion BUT how long you can go ? Can you go beyond the speed of light, thermodynamic laws and high financial costs ?**

Three ways to improve the performance

- **Work harder - Using faster hardware**
- **Work smarter - - doing things more efficiently (algorithms and computational techniques)**
- **Get help - Using multiple computers to solve a particular task.**

<u>Definition</u> :

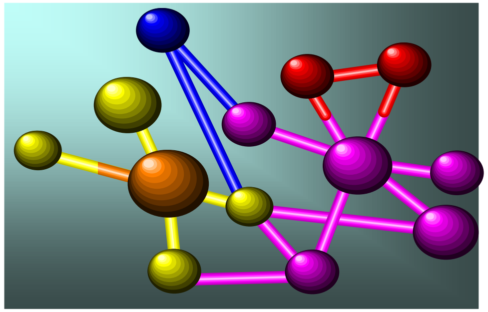
A parallel computer is a "Collection of processing elements that communicate and co-operate to solve large problems fast".

<u>Driving Forces and Enabling Factors</u>

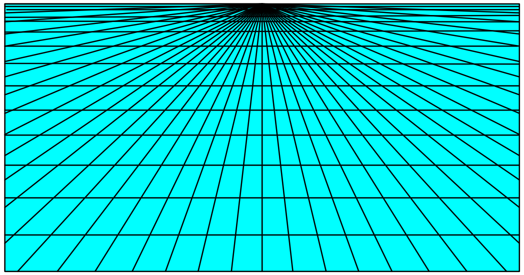
- **Desire and prospect for greater performance**

- **Users have even bigger problems and designers have even more gates**

Need of more Computing Power: Grand Challenge Applications

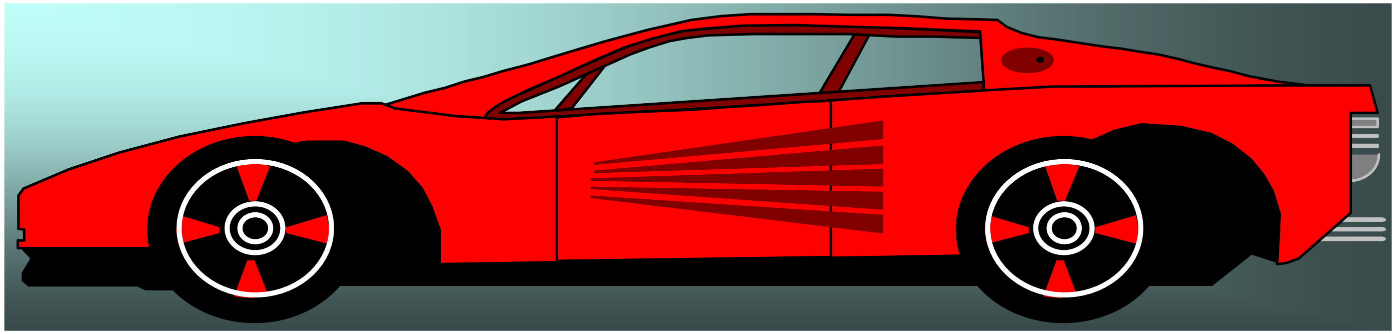


Life Sciences



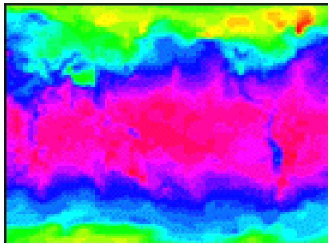
Aerospace

Geographic Information Systems

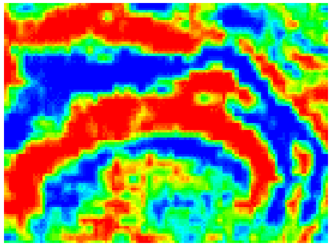


Mechanical Design & Analysis (CAD/CAM)

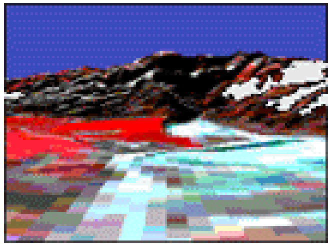
Need of more Computing Power: Grand Challenge Applications



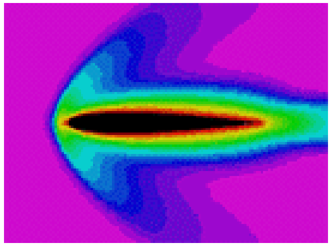
- Weather Forecasting



- Seismic Data Processing
- Remote Sensing, Image Processing & Geomatics



- Computational Fluid Dynamics

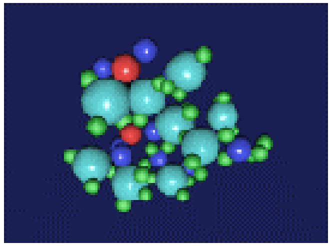


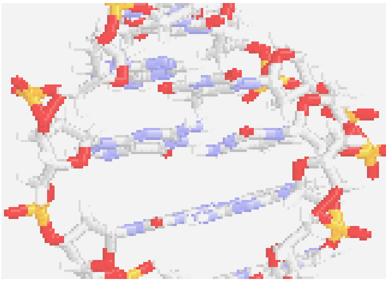
- Astrophysical Calculations

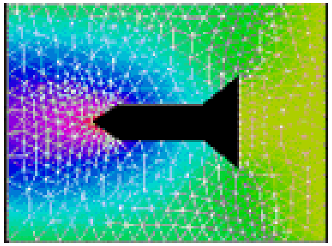
Grand Challenge Applications

Scientific & Engineering Applications

- Computational Chemistry
- Molecular Modelling
- Molecular Dynamics
- Bio-Molecular Structure Modelling
- Structural Mechanics



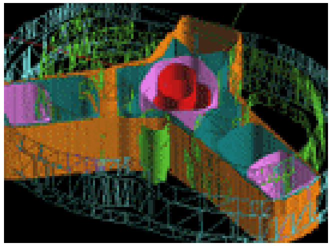




Grand Challenge Applications

Business / Industry Applications

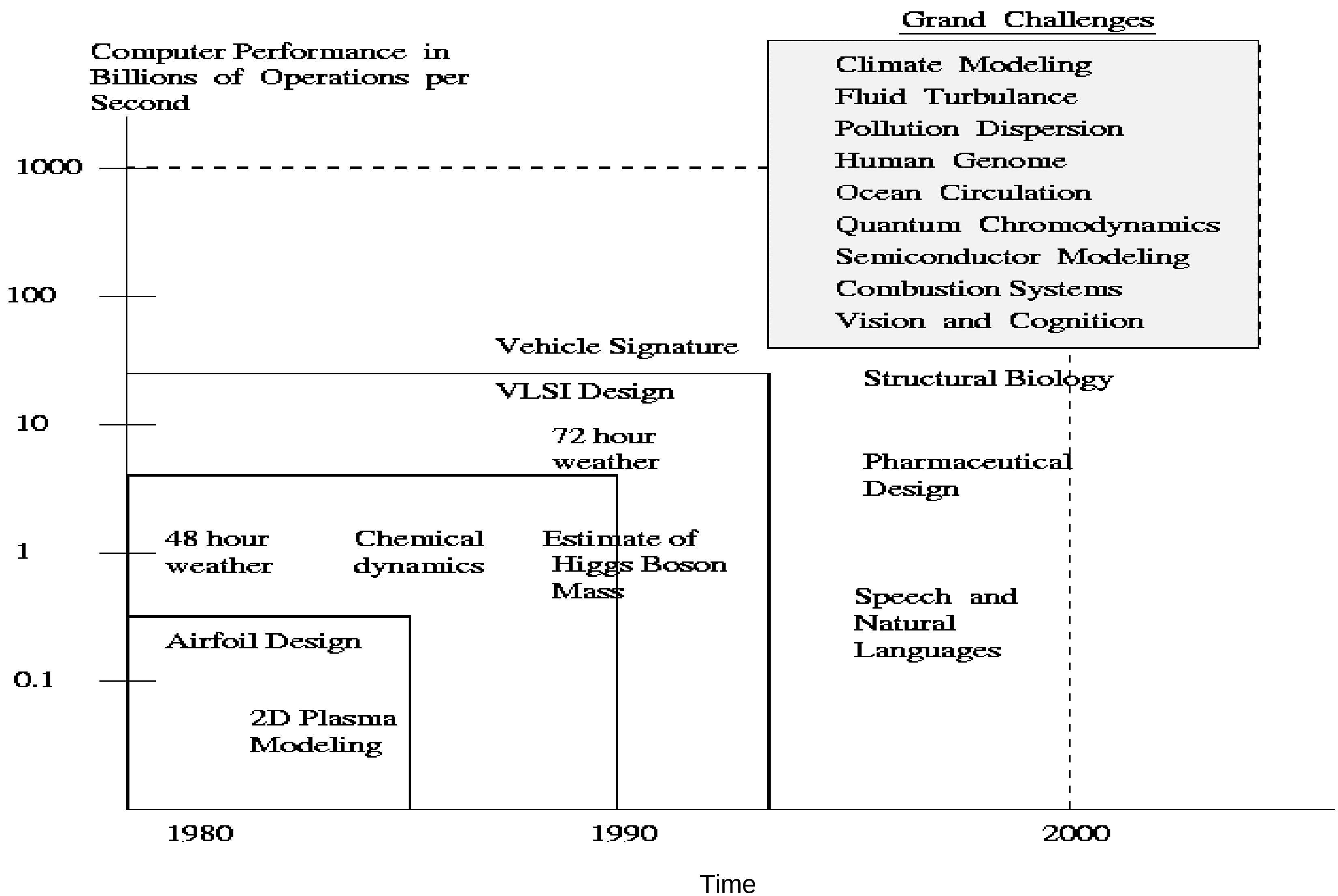
- Data Warehousing for Financial Sectors
- Electronic Governance
- Medical Imaging



Internet Applications

- Web Servers
- Digital libraries

Requirements for Applications



Need of numerical and non-numerical algorithms

- **Numerical Algorithms**
  - **Dense Matrix Algorithms**
  - **Solving linear system of equations**
  - **Solving Sparse system of equations**
  - **Fast Fourier Transformations**
- **Non-Numerical Algorithms**
  - **Graph Algorithms**
  - **Sorting algorithms**
  - **Search algorithms for discrete Optimization**
  - **Dynamic Programming**

Applications – Commercial computing

Commercial Computing

- ✍ **The database is much too large to fit into the computer's memory**
- ✍ **Opportunities for fairly high degrees of parallelism exist at several stages of the operation of a data base management system.**
- ✍ **Millions of databases have been used in business management, government administration, Scientific and Engineering data management, and many other applications.**
- ✍ **This explosive growth in data and databases has generated an urgent need for new techniques and tools**.

Applications – Commercial computing

Sources of Parallelism in Query Processing

- **Parallelism within Transactions (on line transaction processing)**
- **Parallelism within a single complex transactions.**
- **Transactions of a commercial database require processing large complex queries.**

Parallelizing Relational Databases Operations

- **Parallelism comes from breaking a relational operations (Ex : JOIN)**
- **Parallelism comes from the way these operations are implemented.**

Applications – Commercial computing

Parallelism in Data Mining Algorithms

- **Process of automatically finding pattern and relations in large databases**
- **Data sets involved are large and rapidly growing larger**
- **Complexity of algorithms for clustering of large data set**
- **Algorithms are based on decision trees. Parallelism is there on the growth phase due to its data intensive nature**

Requirements for Commercial Applications

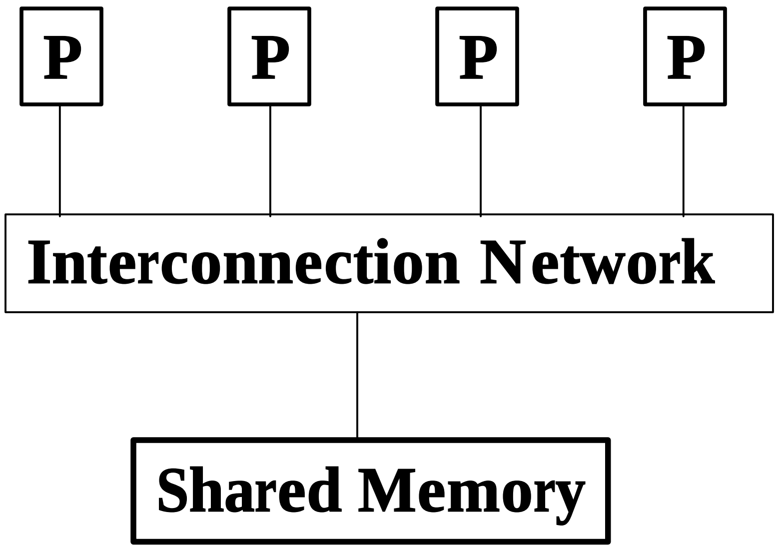
Requirements for applications

- ✍ **Exploring useful information from such data will efficient parallel algorithms.**

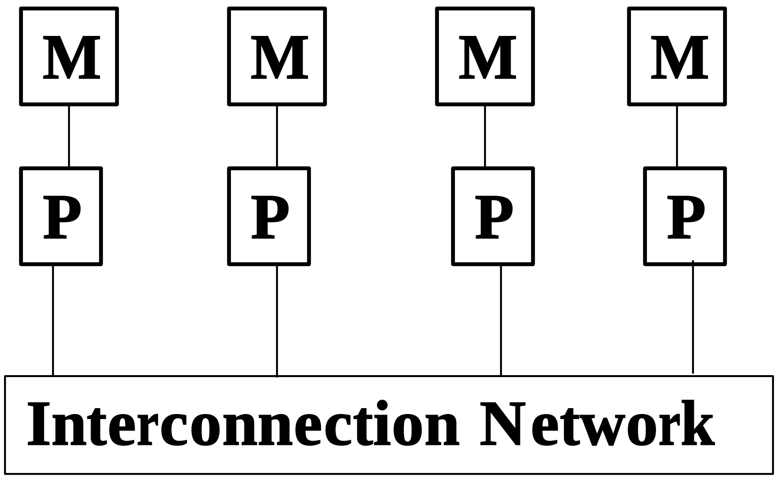
- ✍ **Running on high performance computing systems with powerful parallel I/O capabilities is very much essential**

- ✍ **Development parallel algorithms for clustering and classification for large data sets.**

General Purpose Parallel Computer



Shared Memory Architecture



Distributed Memory Architecture

Serial and Parallel Computing

SERIAL COMPUTING

- Fetch/Store
- Compute

PARALLEL COMPUTING

- Fetch/Store
- Compute/communicate
- Cooperative game

Serial and Parallel Algorithms - Evaluation

- **Serial Algorithm**
  - **Execution time as a function of size of input**
- **Parallel Algorithm**
  - **Execution time as a function of input size, parallel architecture and number of processors used**

Parallel System

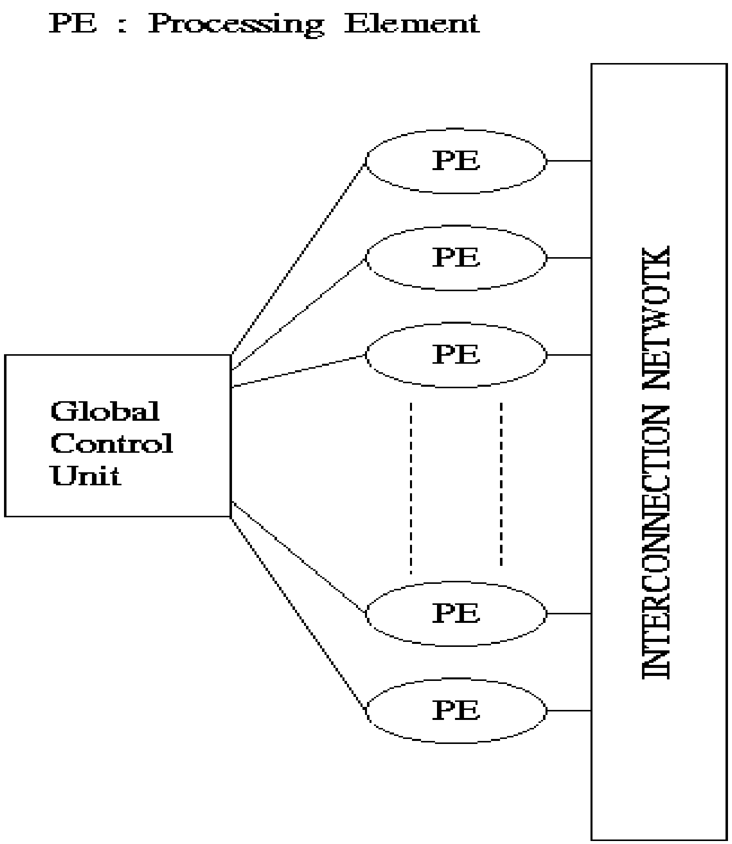
A parallel system is the combination of an algorithm and the parallel architecture on which its implemented

Issues in Parallel Computing

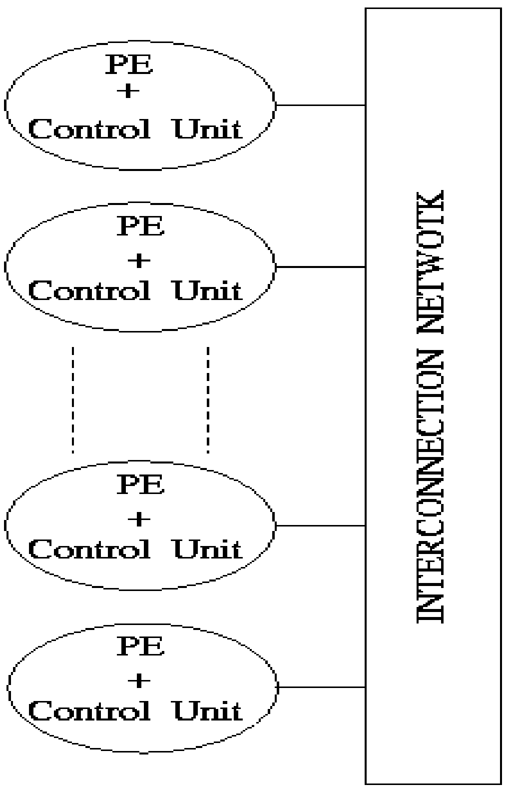
- Design of parallel computers
- Design of efficient parallel algorithms
- Parallel programming models
- Parallel computer language
- Methods for evaluating parallel algorithms
- Parallel programming tools
- Portable parallel programs

Architectural models of Parallel Computers

SIMD



MIMD



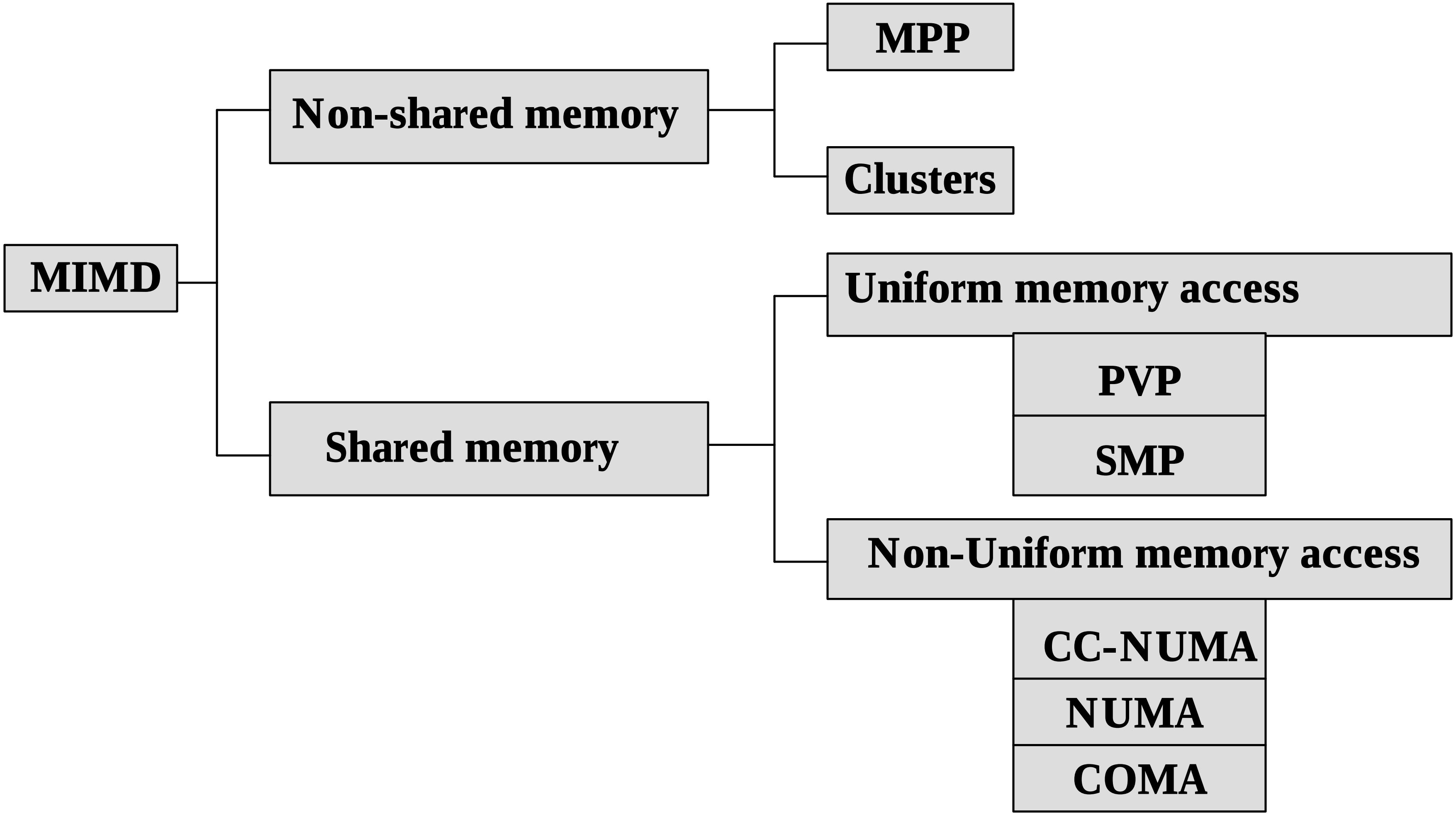
SIMD Features

- **Implementing a fast, globally accessible shared memory takes a major hardware effort**
- **SIMD algorithms for certain class of applications are good choice for performance**
- **SIMD machines are inherently synchronous**
- **There is one common memory for the whole machine**
- **Cost of message passing is very less**

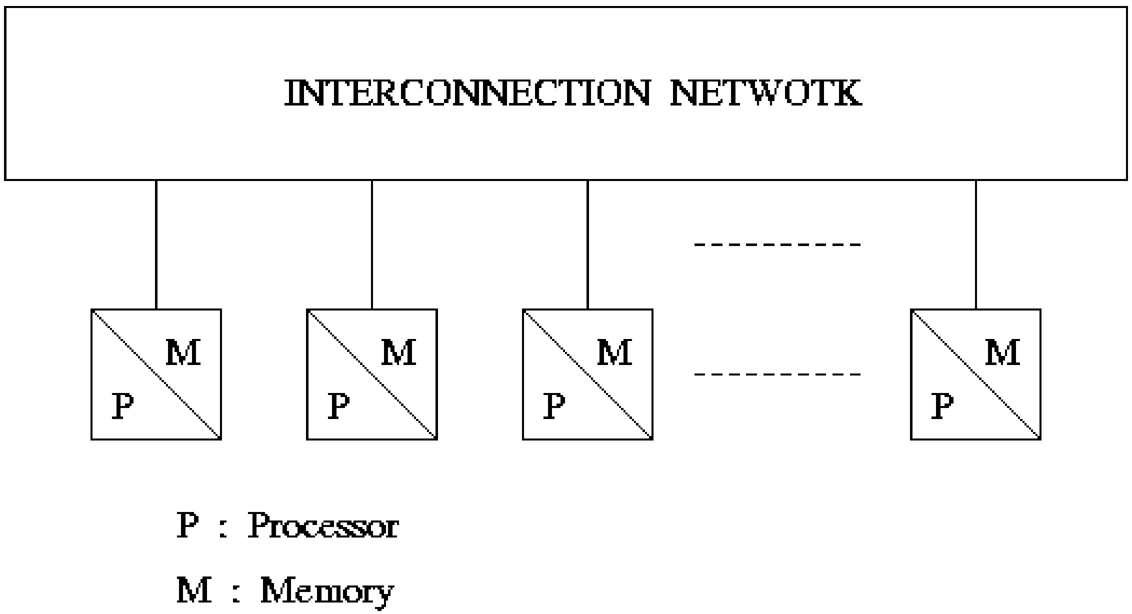
MIMD Features

- MIMD architecture is more general purpose
- MIMD needs clever use of synchronization that comes from message passing to prevent the race condition
- Designing efficient message passing algorithm is hard because the data must be distributed in a way that minimizes communication traffic
- Cost of message passing is very high

MIMD Classification

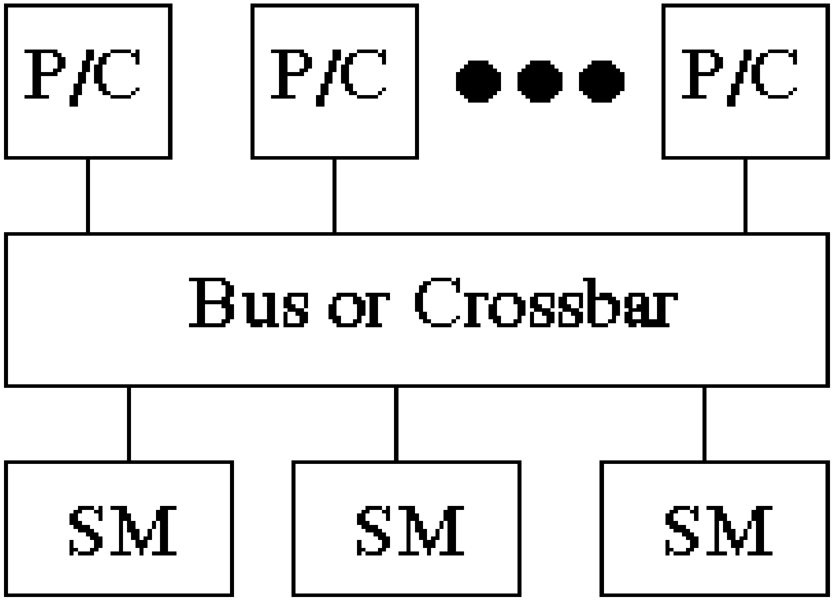


Message Passing Architecture



MIMD message-passing computers are referred as multicomputers

Symmetric Multiprocessors (SMPs)



P / C : Microprocessor and cache; SM : Shared memory

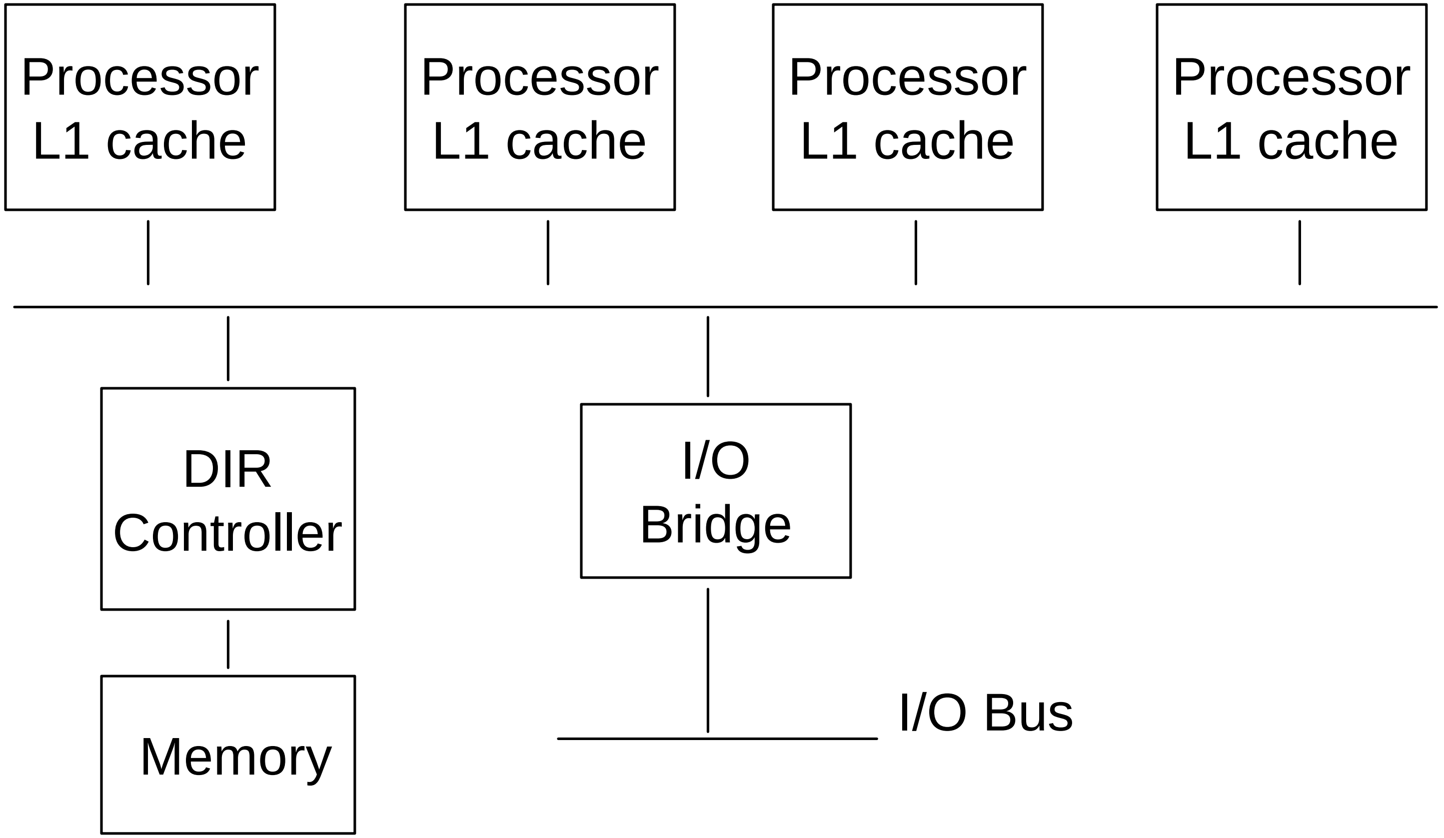
Symmetric Multiprocessors (SMPs)

- **Uses commodity microprocessors with on-chip and off-chip caches.**
- **Processors are connectecd to a shared memory through a high-speed snoopy bus**
- **On Some SMPs, a crossbar switch is used in addition to the bus.**
- **Scalable up to:**
  - **4-8 processors (non-back planed based)**
  - **few tens of processors (back plane based)**

Symmetric Multiprocessors (SMPs)

- **All processors see same image of all system resources**
- **Equal priority for all processors (except for master or boot CPU)**
- **Memory coherency maintained by HW**
- **Multiple I/O Buses for greater Input Output**

Symmetric Multiprocessors (SMPs)



Symmetric Multiprocessors (SMPs)

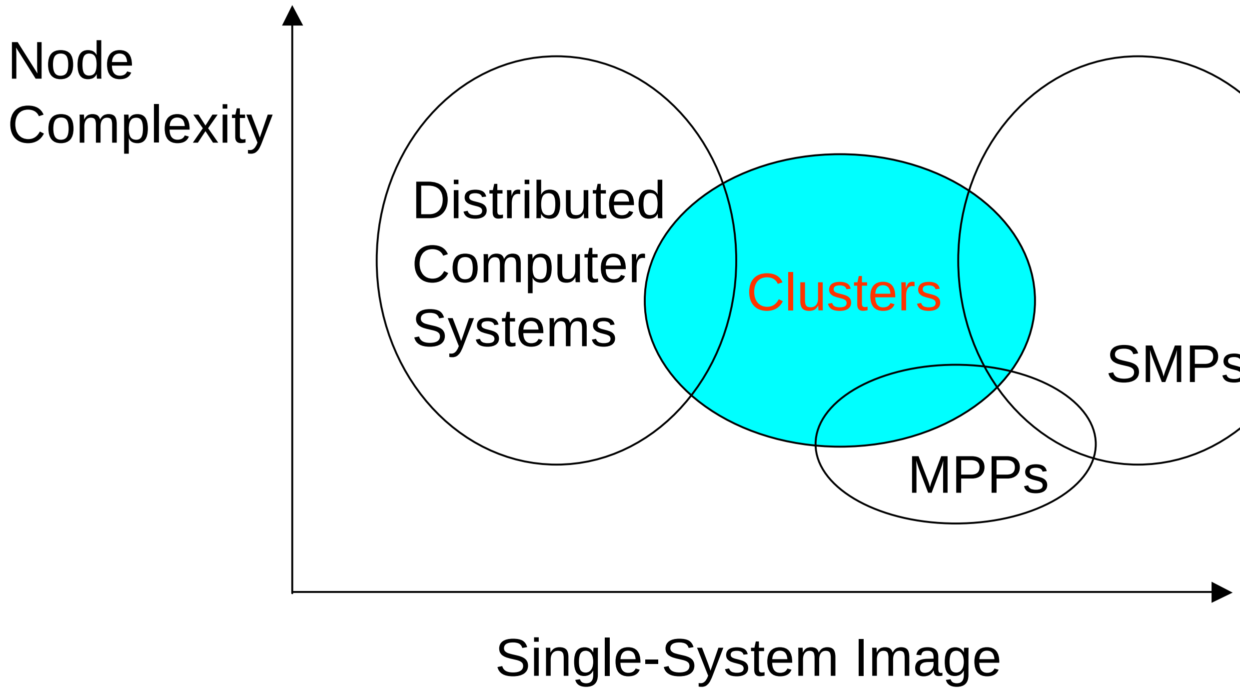
Issues

- **Bus based architecture :**
  - **Inadequate beyond 8-16 processors**
- **Crossbar based architecture**
  - **multistage approach considering I/Os required in hardware**
- **Clock distribution and HF design issues for backplanes**
- **Limitation is mainly caused by using a centralized shared memory and a bus or cross bar interconnect which are both difficult to scale once built.**

Symmetric Multiprocessors (SMPs)

- Heavily used in commercial applications (data bases, on-line transaction systems)
- System is symmetric (every processor has equal equal access to the shared memory, the I/O devices, and the operating systems.
- Being symmetric, a higher degree of parallelism can be achieved.

Better Performance for clusters

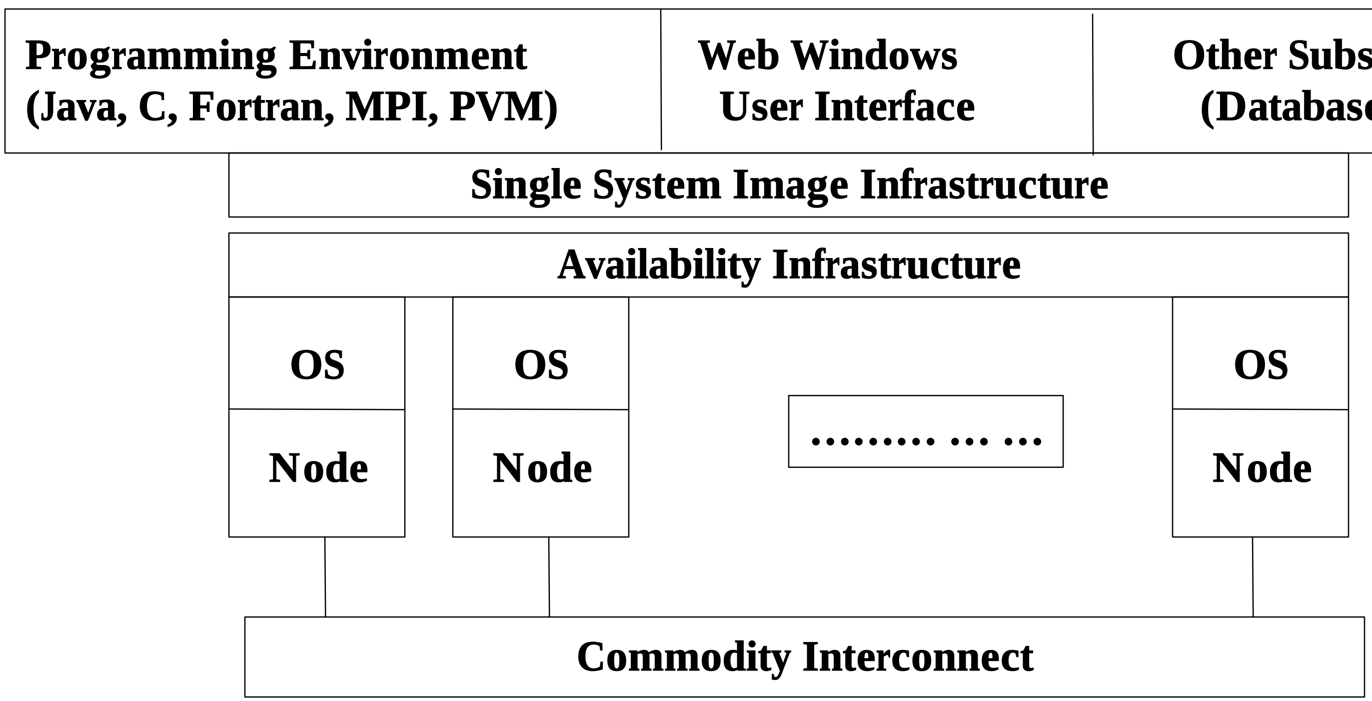


Overlapped design space of clusters, MPPs, SMPs, and distributed computer systems

Clusters

A cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected <u>stand-alone computers</u> cooperatively working together as a <u>single</u>, integrated computing resource.

Cluster Architecture



Clusters Features

- Collection of nodes physically connected over commodity/ proprietary network
- Network is a decisive factors for scalability issues (especially for fine grain applications)
- Each node is usable as a separate entity
- Built in reliability and redundancy
- Cost/performance

Different about clusters?

- Commodity parts
- Incremental Scalability
- Independent Failure
- Complete Operating System on every node
- Good Price Performance Ratio

Cluster Challenges

- Single System Image
- Programming Environments (MPI/PVM)
- Compilers
- Process/thread migration, global PID
- Global File System
- Scalable I/O Services
- Network Services

Parallel I/O

- **Parallel File System**
- **Parallel read / write**
- **Parallel I/O architecture for storage subsystem**

Conclusion: A way to achieve high I/O throughput

PARAM 10000 - A 100 GF Parallel Supercomputer

Developed by - Centre for Development of Advanced Computing, India

40 Sun Enterprise Ultra450 Nodes

No. of CPUs per node 4 @300MHz

Networks

- **Fast Ethernet**
- **PARAMNet**
- **Myrinet**

Parallel Computing Environments

- **PVM**
- **MPI**
- **OpenMP**

File Servers	**4 @ 4GB RAM**
Compute Nodes	**36 @ 2GB RAM**
OS	**Solaris 2.7**

Issues in Parallel Computing on Clusters

- **Productivity**
- **Reliability**
- **Availability**
- **Usability**
- **Scalability**
- **Available Utilization**
- **Performance / cost ratio**

Requirements for Applications

- Parallel I/O
- Optimized libraries
- Low latency and High bandwidth networks
- Scalability of a parallel system

Important Issues in Parallel Programming

- ✍ **Partitioning of data**
- ✍ **Mapping of data onto the processors**
- ✍ **Reproducibility of results**
- ✍ **Synchronization**
- ✍ **Scalability and Predictability of performance**

Success depends on the combination of

- ✍ **Architecture, Compiler, Choice of Right Algorithm, Programming Language**

- ✍ **Design of software, Principles of Design of algorithm, Portability, Maintainability, Performance analysis measures, and Efficient implementation**

Designing Parallel Algorithms

- **Detect and exploit any inherent parallelism in an existing sequential Algorithm**
- **Invent a new parallel algorithm**
- **Adopt another parallel algorithm that solves a similar problem**

Principles of Parallel Algorithms and Design

Questions to be answered

- ✍ **How to partition the data?**
- ✍ **Which data is going to be partitioned?**
- ✍ **How many types of concurrency?**
- ✍ **What are the key principles of designing parallel algorithms?**
- ✍ **What are the overheads in the algorithm design?**
- ✍ **How the mapping for balancing the load is done effectively?**

Two keysteps

- Discuss methods for mapping the tasks to processors so that the processors are efficiently utilized.
- Different decompositions and mapping may yield good performance on different computers for a given problem.

It is therefore crucial for programmers to understand the relationship between the underlying machine model and the parallel program to develop efficient programs.

Parallel Algorithms - Characteristics

- **A parallel algorithm is a recipe that tells us how to solve a given problem using multiprocessors**
- **Methods for handling and reducing interactions among tasks so that the processors are all doing useful work** most of the time **is important for performance**
- **Parallel algorithms has the added dimensions of concurrency which is of paramount importance in parallel programming.**
- **The maximum number of tasks that can be executed at any time in a parallel algorithm is called *degree of concurrency***

Types of Parallelism

- Data parallelism
- Task parallelism
- Combination of Data and Task parallelism
- Stream parallelism

Types of Parallelism - Data Parallelism

- **Identical operations being applied concurrently on different data items is called data parallelism.**
- **It applies the SAME OPERATION in parallel on different elements of a data set.**
- **It uses a simpler model and reduce the programmer's work.**

Example

- ✍ **Problem of adding *n* x *n* matrices.**
- ✍ **Structured grid computations in CFD.**
- ✍ **Genetic algorithms.**

Types of Parallelism - Data Parallelism

- **For most of the application problems, the degree of data parallelism with the size of the problem.**
- **More number of processors can be used to solve large size problems.**
- **f90 and HPF data parallel language**

Responsibility of programmer

- **Specifying the distribution of data structures**

Types of Parallelism - Task Parallelism

- **Many tasks are executed concurrently is called task parallelism.**
- **This can be done (visualized) by a task graph. In this graph, the node represent a task to be executed. Edges represent the dependencies between the tasks.**
- **Sometimes, a task in the task graph can be executed as long as all preceding tasks have been completed.**
- **Let the programmer define different types of processes. These processes communicate and synchronize with each other through MPI or other mechanisms.**

Types of Parallelism - Task Parallelism

Programmer's responsibility

- **Programmer must deal explicitly with process creation, communication and synchronization.**

Task parallelism

Example

Vehicle relational database to process the following query

(MODEL = "-------" AND YEAR = "-------")
AND (COLOR = "Green" OR COLOR ="Black")

Types of Parallelism - Data and Task Parallelism

Integration of Task and Data Parallelism

✍ **Two Approaches**

- **Add task parallel constructs to data parallel constructs.**
- **Add data parallel constructs to task parallel construct**

✍ **Approach to Integration**

- **Language based approaches.**
- **Library based approaches.**

Types of Parallelism - Data and Task Parallelism

Example

- **Multi disciplinary optimization application for aircraft design.**
- **Need for supporting task parallel constructs and communication between data parallel modules**
- **Optimizer initiates and monitors the application's execution until the result satisfy some objective function (such as minimal aircraft weight)**

Types of Parallelism - Data and Task Parallelism

Advantages

- Generality
- Ability to increase scalability by exploiting both forms of parallelism in a application.
- Ability to co-ordinate multidisciplinary applications.

Problems

- Differences in parallel program structure
- Address space organization
- Language implementation

Types of Parallelism - Stream Parallelism

- **Stream parallelism refers to the simultaneous execution of different programs on a data stream. It is also referred to as *pipelining*.**

- **The computation is parallelized by executing a different program at each processor and sending intermediate results to the next processor.**

- **The result is a pipeline of data flow between processors.**

Types of Parallelism - Stream Parallelism

- Many problems exhibit a combination of data, task and stream parallelism.
- The amount of stream parallelism available in a problem is usually independent of the size of the problem.
- The amount of data and task parallelism in a problem usually increases with the size of the problem.
- Combinations of task and data parallelism often allow us to utilize the coarse granularity inherent in task parallelism with the fine granularity in data parallelism to effectively utilize a large number of processors.

Decomposition Techniques

The process of splitting the computations in a problem into a set of concurrent tasks is referred to as <u>decomposition</u>.

- **Decomposing a problem effectively is of paramount importance in parallel computing.**

- **Without a good decomposition, we may not be able to achieve a high degree of concurrency.**

- **Decomposing a problem must ensure good load balance.**

<u>What is meant by good decomposition?</u>

- **It should lead to high degree of concurrency**
- **The interaction among tasks should be title as possible. These objectives often conflict with each other.**
- **Parallel algorithm design has helped in the formulation of certain heuristics for decomposition.**

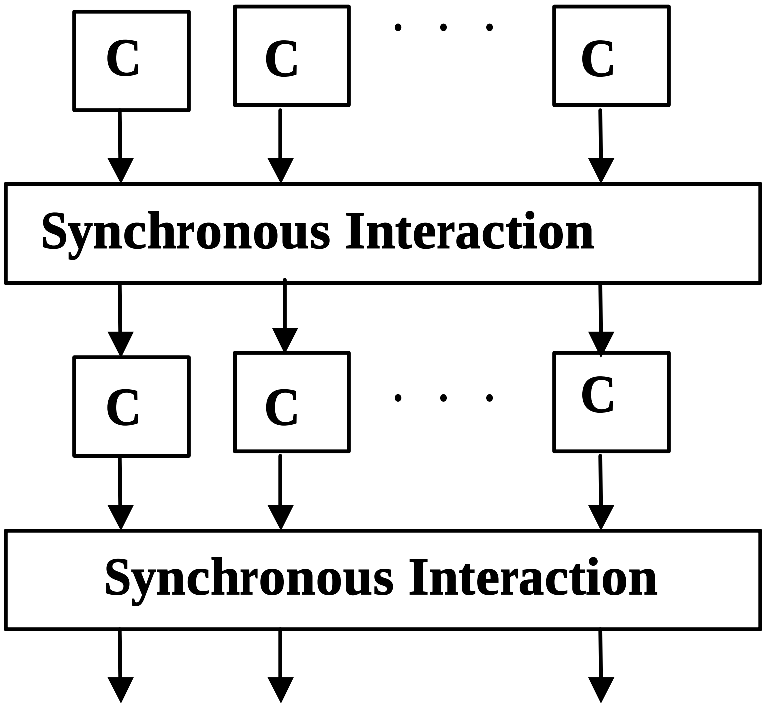
Parallel Programming Paradigm

- ✍ **Phase parallel**
- ✍ **Divide and conquer**
- ✍ **Pipeline**
- ✍ **Process farm**
- ✍ **Work pool**

Remark :

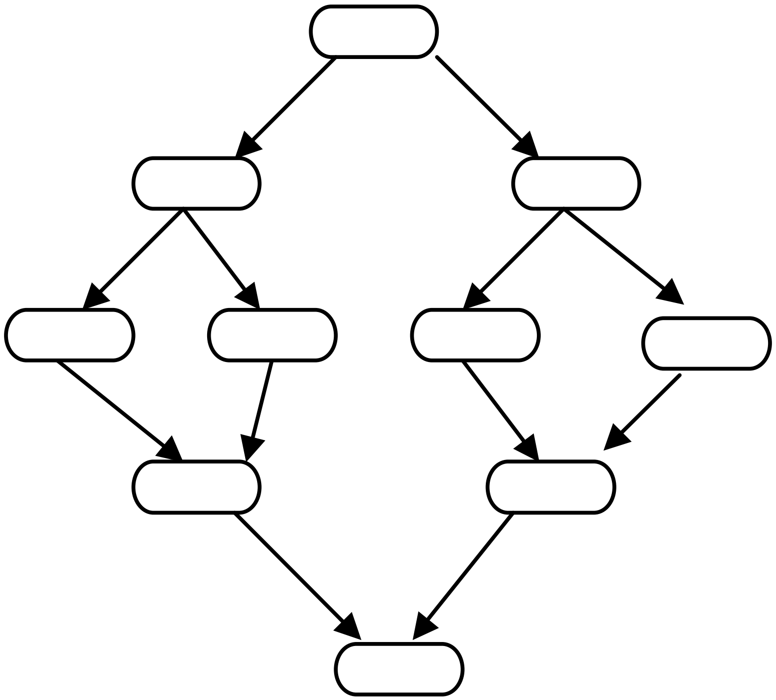
The parallel program consists of number of super steps, and each super step has two phases : *computation phase and interaction phase*

Phase Parallel Model



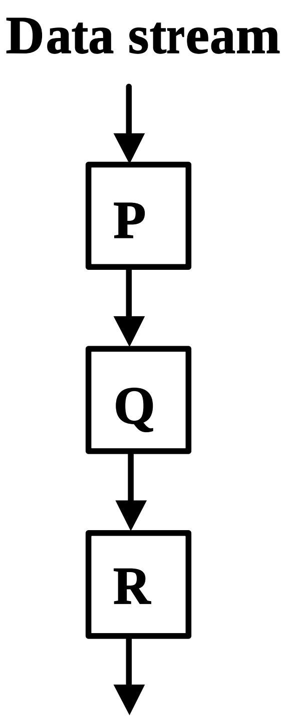
- The phase-parallel model offers a paradigm that is widely used in parallel programming.
- The parallel program consists of a number of supersteps, and each has two phases.
- In a computation phase, multiple processes each perform an independent computation *C*.
- In the subsequent interaction phase, the processes perform one or more synchronous interaction operations, such as a barrier or a blocking communication.
- Then next superstep is executed.

Divide and Conquer



- A parent process divides its workload into several smaller pieces and assigns them to a number of child processes.
- The child processes then compute their workload in parallel and the results are merged by the parent.
- The dividing and the merging procedures are done recursively.
- This paradigm is very natural for computations such as quick sort. Its disadvantage is the difficulty in achieving good load balance.

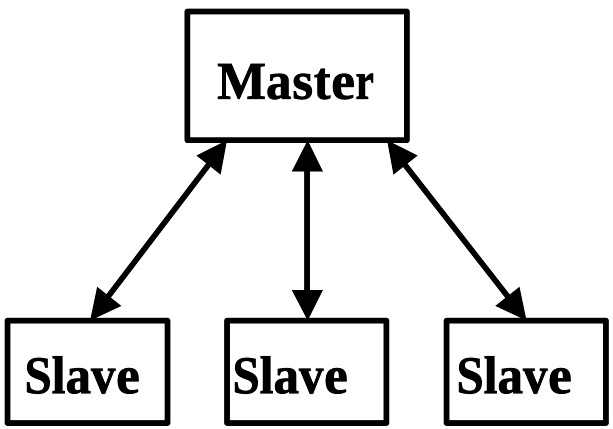
Pipeline



- In pipeline paradigm, a number of processes form a virtual pipeline.
- A continuous data stream is fed into the pipeline, and the processes execute at different pipeline stages simultaneously in an overlapped fashion.

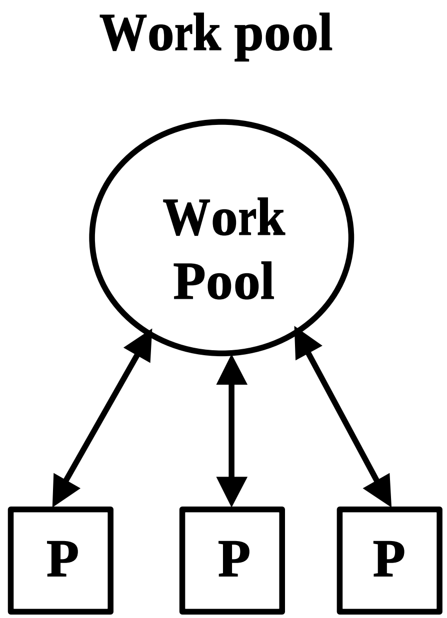
Process Farm

Data stream



- This paradigm is also known as the master-slave paradigm.
- A master process executes the essentially sequential part of the parallel program and spawns a number of slave processes to execute the parallel workload.
- When a slave finishes its workload, it informs the master which assigns a new workload to the slave.
- This is a very simple paradigm, where the coordination is done by the master.

Work Pool



- This paradigm is often used in a shared variable model.
- A pool of works is realized in a global data structure.
- A number of processes are created. Initially, there may be just one piece of work in the pool.
- Any free process fetches a piece of work from the pool and executes it, producing zero, one, or more new work pieces put into the pool.
- The parallel program ends when the work pool becomes empty.
- This paradigm facilitates load balancing, as the workload is dynamically allocated to free processes.

Implicit parallelism

- If the programmer does not explicitly specify parallelism, but let the compiler and the run-time support system automatically exploit it.

Explicit Parallelism

- It means that parallelism is explicitly specified in the source code by the programming using special language constructs, complex directives, or library cells.

Implicit Parallelism: Parallelizing Compilers

- ✍ **Automatic parallelization of sequential programs**
  - **Dependency Analysis**
  - **Data dependency**
  - **Control dependency**

Remark

- ✍ **Users belief is influenced by the currently disappointing performance of automatic tools (Implicit parallelism) and partly by a theoretical results obtained**

Effectiveness of Parallelizing Compilers

✍ **Question :**

- **Are parallelizing compilers effective in generalizing efficient code from sequential programs?**
  - **Some performance studies indicate that may not be a effective**
  - **User direction and Run-Time Parallelization techniques are needed**

Implicit Parallelism

✍ **Bernstein's Theorem**

- **It is difficult to decide whether two operations in an imperative sequential program can be executed in parallel**
- **An implication of this theorem is that there is no automatic technique, compiler time or runtime that can exploit all parallelism in a sequential program**

Implicit Parallel Programming Models

- ✍ **To overcome this theoretical limitation, two solutions have been suggested**

  - **The first solution is to abolish the imperative style altogether, and to use a programming language which makes parallelism recognition easier**

  - **The second solution is to use explicit parallelism**

Explicit Parallel Programming Models

Three dominant parallel programming models are :

- ✍ **Data-parallel model**
- ✍ **Message-passing model**
- ✍ **Shared-variable model**

Explicit Parallel Programming Models

Main Features	Data-Parallel	Message-Passing	Shared-Variable
Control flow (threading)	Single	Multiple	Multiple
Synchrony	Loosely synchronous	Asynchronous	Asynchronous
Address space	Single	Multiple	Multiple
Interaction	Implicit	Explicit	Explicit
Data allocation	Implicit or semiexplicit	Explicit	Implicit or semiexplicit

Explicit Parallel Programming Models

The data parallel model

- Applies to either SIMD or SPMD models
- The idea is to execute the same instruction or program segment over different data sets simultaneously on multiple computing nodes
- It has a single thread of control and massive parallelism is exploited at data set level.
- Example: f90/HPF languages

Explicit Parallel Programming Models

Data parallelism

- Assumes a single address space, and data allocation is not required
- To achieve high performance, data parallel languages such as HPF use explicit data allocation directives
- A data parallel program is single threaded and loosely synchronous
- No need for explicit synchronization free from all deadlocks and livelocks
- Performance may not be good for unstructured irregular computations

Explicit Parallel Programming Models

Message – Passing

✍ **Message passing has the following characteristics :**

- **Multithreading**
- **Asynchronous parallelism (MPI reduce)**
- **Separate address spaces (Interaction by MPI/PVM)**
- **Explicit interaction**
- **Explicit allocation by user**

Message – Passing

- Programs are multithreading and asynchronous requiring explicit synchronization
- More flexible than the data parallel model, but it still lacks support for the work pool paradigm.
- PVM and MPI can be used
- Message passing programs exploit large-grain parallelism

Shared Variable Model

- It has a single address space (Similar to data parallel)
- It is multithreading and asynchronous (Similar to message-passing model)
- Data resides in single shared address space, thus does not have to be explicitly allocated
- Workload can be either explicitly or implicitly allocated
- Communication is done implicitly through shared reads and writes of variables. However synchronization is explicit

Explicit Parallel Programming Models

Shared variable model

- The shared-variable model assumes the existence of a single, shared address space where all shared data reside
- Programs are multithreading and asynchronous, requiring explicit synchronizations
- Efficient parallel programs that are loosely synchronous and have regular communication patterns, the shared variable approach is not easier than the message passing model

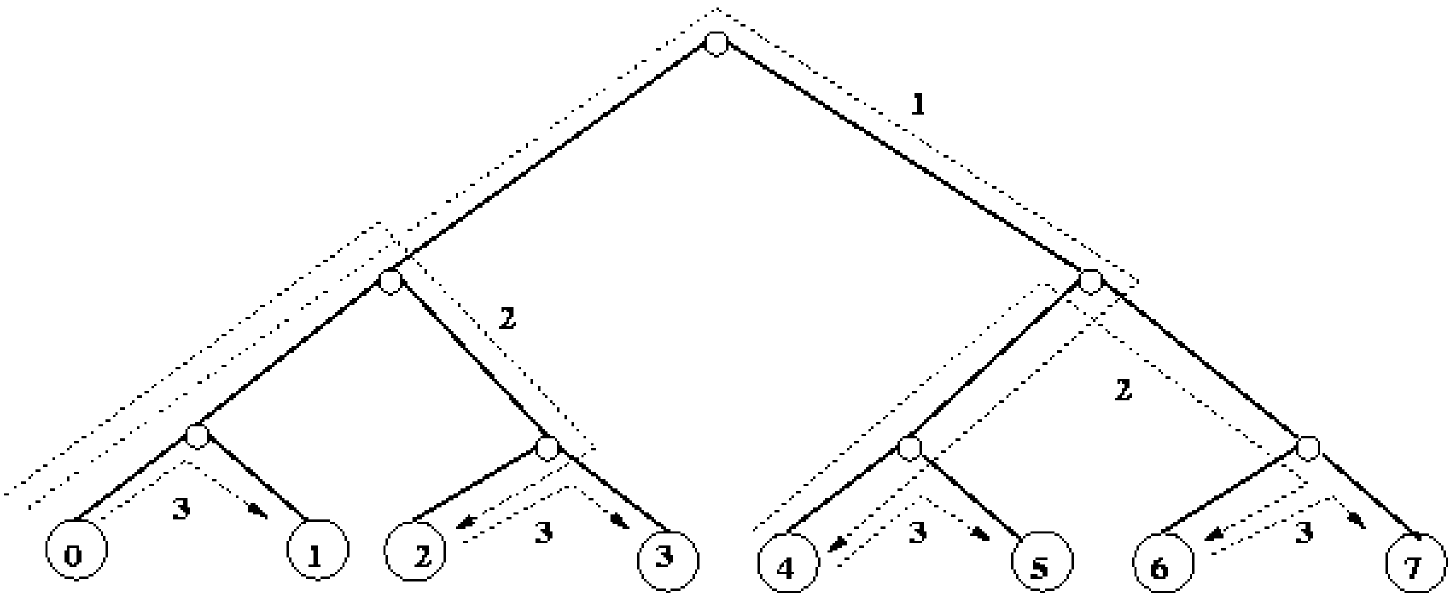
Other Parallel Programming Models

- **Functional programming**
- **Logic programming**
- **Computing by learning**
- **Object oriented programming**

Basic Communication Operations

- **One-to-All Broadcast**
- **One-to-All Personalized Communication**
- **All-to-All Broadcast**
- **All-to-All personalized Communication**
- **Circular Shift**
- **Reduction**
- **Prefix Sum**

One-to-all broadcast on an eight-processor tree



How do we measure the performance of a computer system?

- **Many people believe that execution time is the only reliable metric to measure computer performance**

Approach

- **Run the user's application elapsed time and measure wall clock time**

Remarks

- **This approach is some times difficult to apply and it could permit misleading interpretations.**
- **Pitfalls of using execution time as performance metric.**
  - **Execution time alone does not give the user much clue to a true performance of the parallel machine**

Types of performance requirement

Six types of performance requirements are posed by users:

- **Executive time and throughput**
- **Processing speed**
- **System throughput**
- **Utilization**
- **Cost effectiveness**
- **Performance / Cost ratio**

Remarks : These requirements could lead to quite different conclusions for the same application on the same computer platform

Remarks

- Higher Utilization corresponds to higher Gflop/s per dollar, provided if CPU-hours are changed at a fixed rate.
- A low utilization always indicates a poor program or compiler.
- Good program could have a long execution time due to a large workload, or a low speed due to a slow machine.
- Utilization factor varies from 5% to 38%. Generally the utilization drops as more nodes are used.
- Utilization values generated from the vendor's benchmark programs are often highly optimized.

Performance Metrics of Parallel Systems

Speedup : Speedup T_p is defined as the ratio of the serial runtime of the **best** sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on p processor

The p processors used by the parallel algorithm are assumed to be **identical** to the one used by the sequential algorithm

Cost : Cost of solving a problem on a parallel system is the product of parallel runtime and the number of processors used

$$E = p.S_p$$

Performance Metrics of Parallel Systems

<u>Efficiency</u> : Ratio of speedup to the number of processors.

Efficiency can also be expressed as the ratio of the execution time of the fastest known sequential algorithm for solving a problem to the cost of solving the same problem on *p* processors

The <u>cost</u> of solving a problem on a single processor is the execution time of the known best sequential algorithm

<u>Cost Optimal</u> : A parallel system is said to be cost-optimal if the cost of solving a problem on parallel computer is proportional to the execution time of the fastest known sequential algorithm on a single processor.

Performance Metrics of Parallel Systems

Speedup metrics

Three performance models based on three speedup metrics are commonly used.

- Amdahl's law -- Fixed problem size
- Gustafson's law -- Fixed time speedup
- Sun-Ni's law -- Memory Bounding speedup

Three approaches to scalability analysis are based on

- Maintaining a constant efficiency,
- A constant speed, and
- A constant utilization

<u>Amdahl's law</u> : Fixed Problem Size

Consider a problem with a fixed workload *W*. Assume that the workload can be divided into two parts

$$W = ?\; W + (1 - ?)\; W$$

where ? percent of W executed sequentially, and the remaining 1- ? percent can be executed by *p* nodes simultaneously.

Assume all overheads are ignored, a fixed load speedup is defined by

$$S_p = \frac{W}{?\; W + (1 - ?)\; W / p} = \frac{p}{1 + (p - 1)?} \longrightarrow \frac{1}{?} \text{ as } p \longrightarrow ?$$

Amdahl's law implications

1. For a given workload, the maximal speedup has an upper bound of 1/ ? .
2. In other words, the sequential component of the program is bottleneck.
3. When ? increases the speedup decreases proportionally.
4. To achieve good speedup, it is important to make the sequential bottleneck ? as small as possible.

For fixed load speedup S_p (with all overheads T_0) becomes

$$S_p = \frac{W}{? \; W + (1- ?)\; W/p + T_0} = \frac{1}{?_0 + T_0 \,/\, W} \quad \text{as } p \longrightarrow ?$$

Gustafson's Law : Scaling for Higher Accuracy

- **The problem size (workload) is fixed and cannot scale to match the available computing power as the machine size increases. Thus, Amdahl's law leads to a diminishing return when a larger system is employed to solve a small problem.**

- **The sequential bottleneck in Amdahl's law can be alleviated by removing the restriction of a fixed problem size.**

- **Gustafson's proposed *a* fixed time concept that achieves an improved speedup by scaling problem size with the increase in machine size.**

Gustafson's Law : Scaling for Higher Accuracy

The <u>fixed-time speedup</u> with scaled workload is defined as

$$S_p^* = \frac{\text{Sequential time for scaled-up workload}}{\text{Parallel time for scaled-up workload}} = \frac{? W + (1- ?)p}{W}$$

$$S_p^* = ? + (1- ?)\, p$$

- **It states that the fixed time speedup is a linear function of *p*, if the workload is scaled up to maintain a fixed execution time.**
- **Achieves an improved speedup by scaling the problem size with the increase in machine size.**

Sun and Ni's law : Memory Bound Speed up

Motivation

- The idea is to solve the largest possible problem, limited only by the available memory capacity.

- This also demands a scaled workload, providing higher speedup, greater accuracy, and better resource utilization

- Use concept of Amdahl's law and Gustafson's law to maximize the use of both CPU and memory capacities

<u>Sun and Ni's law</u> : Memory Bound Speed up (S_p*)

- **Let *M* be the memory capacity of a single node. On an *p*-node parallel system, the total memory is *pM*. Given a memory-bounded problem, assume it uses all the memory capacity *M* on one node and execute in *W* seconds. Now the workload on one node is *W* is given by ? *W* + (1- ?)*W***
- **When p nodes are used, assume that the parallel portion of the workload can be scaled up F(p) times.**
- **Scaled work load is W is given by ? W + (1- ?) F(p) W. (Here the factor G(p) reflects the increase in workload as the memory capacity increases p times).**

$$S_p^* = \frac{?\,W + (1-\,?\,)\,F(p)\,W}{?\,W + (1-\,?\,)\,F(p)\,W/p} = \frac{?\, + (1-\,?\,)\,F(p)}{?\, + (1-\,?\,)\,F(p)/p}$$

Conclusions

Clusters are promising

- Solve parallel processing paradox
- Offer incremental growth and matches with funding pattern
- New trends in hardware and software technologies are likely to make clusters more promising.

Success depends on the combination of

- Architecture, Compiler, Choice of Right Algorithm, Programming Language
- Design of software, Principles of Design of algorithm, Portability, Maintainability, Performance analysis measures, and Efficient implementation

References

- **Albert Y.H. Zomaya, Parallel and distributed Computing Handbook, McGraw-Hill Series on Computing Engineering, New York (1996).**
- **Ernst L. Leiss, Parallel and Vector Computing A practical Introduction, McGraw-Hill Series on Computer Engineering, New York (1995).**
- **Ian T. Foster, Designing and Building Parallel Programs, Concepts and tools for Parallel Software Engineering, Addison-Wesley Publishing Company (1995).**
- **Kai Hwang, Zhiwei Xu, Scalable Parallel Computing (Technology Architecture Programming) McGraw Hill New York (1997)**
- **Vipin Kumar, Ananth Grama, Anshul Gupta, George Karypis, Introduction to Parallel Computing, Design and Analysis of Algorithms, Redwood City, CA, Benjmann/Cummings (1994).**

Final Words

Acknowledgements

- **Centre for Development of Advanced Computing (C-DAC)**
- **Computer Service Center, IIT Delhi**
- **Department of Computer Science & Engineering, IIT Delhi**

More Information can be found at

http://www.cse.iitd.ac.in/~dheerajb/links.htm