UNIT-5
PARALLEL PROGRAMMING MODELS

1. A model for parallel programming is an abstraction and is machine architecture independent. Two categories:
   (1) Implicit parallelism - Compilers and run-time support system automatically specify parallelism.
   (2) Explicit parallelism - Specified by the programmer in source code using special language constructs, complex directives or library cells.

2. Shared variable model -

   ![Diagram of shared variables in a common memory]

   (IPC) Interprocess communication using shared variables.

   Critical section (CS) - It is a code of segment accessing shared variable, which must be executed by only one process at a time and which once started must be completed without interrupt.

   CS satisfies the following requirements:
   (1) Mutual exclusion
   (2) No deadlock in waiting
   (3) No preemption - No interrupt until completion
   (4) Eventual entry

3. Message passing model -

   ![Diagram of process communication]

   IPC using message passing.

   Message can be instructions, data, signals, interrupt etc.

   Synchronous message passing - A synchronous communication is not complete until the message has been received.
Asynchronous message passing - An asynchronous communication completes as soon as the message is on the way. Sends
sends queue of message.

Asynchronous message passing allows more parallelism than synchronous message passing.

3 Data Parallel Model -

It originates from vector programming where the programmer writes his application in terms of highly optimized vector operations.

SIMD implementation -
The programmer need not be aware of the existence of multiple processors but only of the availability of some fast built-in operations for some specific type of data.

MIMD implementation -

The processes are only synchronized at the beginning and end of parallel operations as opposed to matching through the same sequence of instructions all at the same pace.

Array language extension - It is represented by high level data types.

The array syntax enables the removal of some nested loops in the code and should reflect the architecture of the array process.

4 Comparison between explicit parallel programming models

<table>
<thead>
<tr>
<th>Main Feature</th>
<th>Data Parallel</th>
<th>Message Passing</th>
<th>Shared Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Flow</td>
<td>Single</td>
<td>Multiple</td>
<td>Multiple</td>
</tr>
<tr>
<td>Synchrony</td>
<td>highly synchronous</td>
<td>Asynchronous</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>Address Space</td>
<td>Single</td>
<td>multiple</td>
<td>multiple</td>
</tr>
<tr>
<td>Interactor</td>
<td>Implicit</td>
<td>Explicit</td>
<td>Explicit</td>
</tr>
<tr>
<td>Data allocation</td>
<td>Implicit</td>
<td>Explicit</td>
<td>semiexplicit</td>
</tr>
</tbody>
</table>
Object-Oriented model -
It provides suitable abstractions and software engineering models methods for structured application design.

→ Concurrent Object-oriented Programming (COOP) -
It combines concurrency and object-oriented programming.
E.g. JAVA, combine OOP with threads.

Systems where objects themselves are a concurrency primitive, such as when objects are combined with the actor model.

→ Actor Model -
Everything in the system is taken to be an actor. Actors are completely independent of other actors.
All actions taken by an actor upon receipt of a message are concurrent, no implicit serial ordering of the actions in a method.
Actor primitives are Create, Send-to, and Become.

→ Parallelism in COOP -
(1) Pipeline concurrency
(2) Divide and Conquer concurrency
(3) Co-operative problem solving

Functional and logic models -

→ Functional programming models -
Functional programming languages use declarative semantics and some form of lambda calculus to express the operation of a program.
Concurrency is done with pure functional languages.

→ Logic programming models -
Logic programming is suitable for knowledge processing dealing with large databases. This model adopts an implicit search strategy and support parallelism in the logic inference process.
Concurrency is done by parallelism (execute multiple predicates), or parallelism (execute multiple guards), through explicit mapping of predicates linked together through guard assignment variables.
1. **Parallel languages and Computers** -

- **Parallel Languages** -
  - Languages that use parallel programming, e.g., SISAL, PCW
  - Language features for parallelism (e.g.,
    - Optimization features (automated analysis, semi-automated analysis)
    - Availability features (scalability, compatibility, and portability)
    - Synchronization/Communication features (send/recv, remote procedure call)
    - Control of parallelism (by explicit/explicit parallelism)
    - Data parallelism features (mapping functions, virtual processor support)
    - Parvam Management features (high-level parallelism, automatic load balancing, dynamic process creation)

2. **Parallelizing Compilers** - Source Code
   - Flow Analysis
   - Program Optimization
   - Parallel Code Generation

3. **Parallel Programming Environment** -

   It is a bridge between a system developer's natural model of an application and an implementation of that application on available hardware.

   It consists of hardware platforms, language support, OS, and software tools and application packages.

   It must offer portable, efficient, and easy-to-use tools for development of applications.
Software tools and environments -

- Pure Parallel Languages
  - VINO, FORGE, SISAL, HyperTalk

Software Tool Type

- Basic
  - SPICES-2, SCHEDULE, POKER, MONMAC
- Limited
  - FAUST, Express, TOPSYS
- Well
  - PIE, PAT, Hyrias

Basic environment provides a simple program tracing facility for debugging and performance trace monitoring.

Limited integration provides tools for parallel debugging, performance monitoring or program visualization.

Well developed environments provide intensive tools for debugging, programs, interaction of textual/graphical representations of a parallel program, visualization support for performance monitoring, parallel I/O, parallel graphics etc.

Environment features -

1. Control flow graph generation
2. Integrated graphical maps
3. Parallel debugger at source code level
4. Performance monitoring by either software or hardware means
5. Performance prediction model
6. Parallel input/output for fast data movement
7. OS support for parallelism in front end and back end environment