Chapter 2 Parallel Programming Concepts and Network Properties

Contents of this chapter

- Conditions of parallelism - Various dependencies
- Bernstein's conditions
- Hardware and Software parallelism
- Program and/or Data Partitioning and Scheduling

- Program flow mechanisms
- Network Architectures, Properties and Routing + **MINS**
- Performance Analysis - Effect of Granularity on the Time performance in multiprocessor systems

ANNEX

- Scheduling using Flow Models – Max-Flow Min-Cut Theorem
- Mapping tasks onto a Multiprocessor system – Decision making under data certainty

Reference: Kai Hwang's book

Chapter 2: Pages - 51 to 75

+ Material from Download zone as indicated.

Reading Assignment :

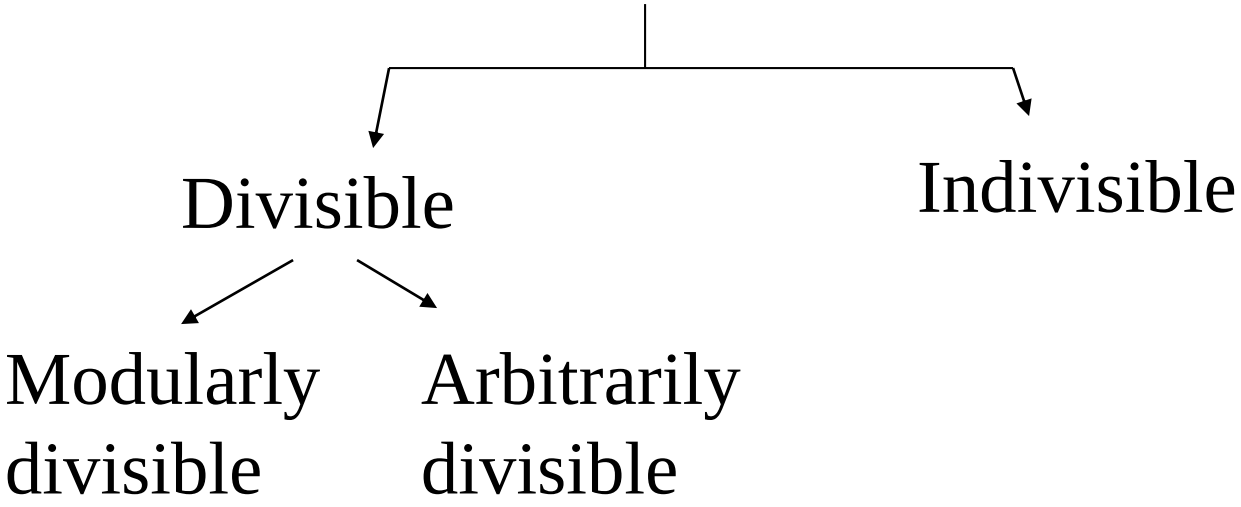
Section 2.3

pages - 70-75 (inclusive of Section 2.3.3)

Parallelism - Program
Data

- **This chapter deals with program parallelism and some aspects of data parallelism**

Data Parallelism : DATA



Indivisible: These are types of tasks that cannot be partitioned into sub-tasks. These have to processed in their entirety. Thus, a task in its entire form must be assigned to a processor in the system.

Gantt charts are used to represent the assignment of tasks onto the processors.

Note: The tasks could range from micro-instructions to a high level task and the resources could be a processor or an execution unit inside a processor

Modularly divisible: These tasks can be partitioned to a limited extent, however, inter-dependencies must be taken care while assigning them to different processors.

Graph representation is generally used. The nodes of the graph represent sub-tasks (with or without any weights) and the edges represent the dependencies (with or without weights)

To some extent (depending on the problem size) Gantt charts can be used here too.

Arbitrarily divisible: After partitioning the data, each segment can be independently processed on any node.

There is no data dependency between the segments.

Thus, using a parallel processor set-up, we can partition the data into equal sizes and can be processed on these nodes. There are no communication delays.

However, using a loosely coupled multiprocessor system (distributed system), need to account for the non-zero communication delays and heterogeneous nature of the system.

Divisible Load theory (DLT) proposes various strategies to schedule the divisible data onto the distributed system.

Objective: Minimize the total processing time of the data subject to the communication and computation delays.

Issues: Fault-tolerance, reliability, modelling the communication and computation delays, topology

Reference: *V. Bharadwaj, et. Al (1996), "Scheduling Divisible Loads in Parallel and Distributed Systems", IEEE Computer Society, Los Alamitos, CA, USA.*

Conditions of Parallelism

Attributes of parallelism - computational granularity, Time and space complexities, communication latencies, scheduling policies, load balancing, etc.

Data dependencies

- Dependence graph - precedence relations
- Five types are identified: *Flow dependence*, *Anti-dependence*, *Output dependence*, *I/O dependence*, *Unknown dependence*

Flow dependence: A statement S2 is FD on statement S1, if an execution path exists from S1 to S2 and if at least one output of S1 feeds in as input to S2. FD is denoted as **S1 -> S2**.

Antidependence: Statement S2 is antidependent on S1 if S2 follows S1 in program order and if the output of S2 overlaps the input of S1. This is denoted as a crossed directed arrow between S1 and S2, as **S1 -/-> S2**.

Output dependence: Two statements are OD, if they produce the same output variable. A circle on the directed arrow denotes this relation as, **S1-o-> S2**.

I/O dependence: Read and write are I/O statements; I/O dependence occurs when same file is referenced by both I/O statements.

Unknown dependence: Dependence relationships cannot be determined in the following situations:

- indirect addressing
- subscript does not contain loop index variable
- a variable appears more than once with subscripts having different coefficients of the loop variable (X_{mj})
- subscript is non-linear in the loop index variable

Refer to Example 2.1 on page 52. This example verifies all the dependencies defined earlier - *A must read exercise!*

Control Dependence: When the order of the execution cannot be determined before run time, such a situation arises. Conditional statements like **IF** will not be resolved until run time.

```
Do 20 I=1,N          control independent loop
      A(I) = C(I)
      IF (A(I).LT.0) A(I)=1
20 Continue
```

```
Do 10 I=1,N          control dependent loop
        IF (A(I-1).EQ.0) A(I)=0
10 Continue
```

Control dependence often prohibits parallelism from being exploited. To circumvent this problem, compiler techniques must be developed.

Resource dependence: This is concerned with the conflicts in using shared resources, such as integer units, floating point units, registers, memory areas, etc.

- ALU dependence, Storage dependence

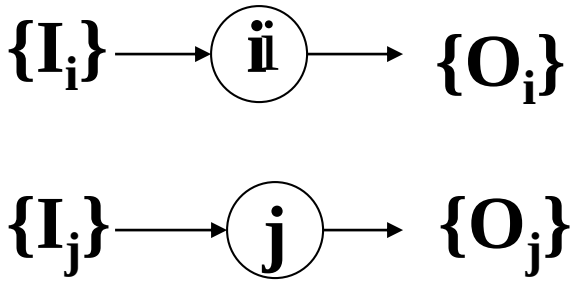
Bernstein's Conditions (1966)

These conditions are used to test whether two processes can be executed in parallel or not.

I_i: Set of input variables of a process P_i
O_i: Set of output variables

Two processes can execute in parallel, denoted as $P_i \parallel P_j$, if

$I_i \cap O_j = \phi$
$I_j \cap O_i = \phi$
$O_i \cap O_j = \phi$



These three conditions are referred to as *Bernstein's* conditions.

In terms of data dependencies, Bernstien's conditions mean that the two processes must be *flow-independent, anti-independent, output independent* for parallel execution

In general, a set of processes $P_1,\ldots,P_k$, can be executed in parallel, *if and only if* Bernstein's conditions are satisfied pairwise, i.e., for all $i \neq j$, P_i *and* P_j *must be executed in parallel.*

Refer to an example on page 55 - Example 2.2 - Detection of parallelism in a program using Bernstein's condition.

Trace the example carefully to understand figures 2.2 (a) and (b) on page 56

In general, violations of any one or more of the 3n(n-1)/2 Bernstein's conditions among n processes prohibits parallelism collectively or partially.

In general, data dependency prohibits parallelism.

Some properties of || relation:

- In general, || relation is commutative
- || relation is not transitive
  (See example: (P1 || P5) and (P5 || P2), but P1 and P2 are not independent

The above two implies, || is not an equivalence relation

- Associativity property holds
- $I_i \cap I_j \neq \phi$ does not prevent parallelism to be expolited

Hardware Parallelism(HP)

- Defined and decided by the machine architecture
- Function of cost and performance trade-offs

How do we characterize the parallelism?

One way is by the number of instruction issues per machine cycle.

If a processor issues k instructions per machine cycle, then it is called as a *k-issue* processor.

A conventional processor takes one or two cycles for issuing a single instruction, and these are referred to as *one-issue* processor.

Some existing architectures:

1). Intel i960CA is a three-issue processor
2). IBM RISC/System 6000 is a four-issue processor

Note: A multiprocessor system built with an ***n k-issue*** processors should be able to handle a maximum of ***nk*** threads of instructions simultaneously.

Software parallelism(SP)

This is defined entirely by the *control and data* dependence of programs.

The degree of parallelism is revealed via dependence flow graph.

SP is a function of algorithm, programming style, and compiler optimization.

Refer to an example on Page 58 - Fig. 2.3(a) and (b) to understand how HP and SP are determined.

Software parallelism :

* Control parallelism
* Data parallelism

Control Parallelism: Two or more operations in parallel pipelining is an example; handled largely by hardware

Data Parallelism: Used in MIMD and SIMD systems; parallelism is exploited in direct proportion to the amount of data available.

To solve the mismatch problem between h/w and s/w parallelism, compiler support is very much essential.

Program Partitioning

Grain size / Granularity: Measure of the amount of computation involved in a process.

Sometimes, it is also referred to as the size of the task that can be assigned onto a a processor for execution.

The simplest measure is to count the number of instructions in a grain. Grain size also refers to the program segment that can be executed in parallel.

- Fine grain, medium grain, and coarse grain

Latency: This is a time measure of the communication overhead incurred between machine subsystems.

Examples: memory latency, synchronization latency

Levels of Parallelism

- Instruction level - *fine grain size*
- Loop level - *fine grain size*
- Procedure level - *medium grain size*
- Subprogram level - *medium/coarse grain size*
- Job level - *coarse level*

Instruction level: (***Slides 24 – 26 Reading Assignment***)

- Depending on individual programs, we can have fine grain size between 2 to 1000's in number.
- Assisted largely by compiler (implicit parallelism)

Loop level:

- Typical loop has 100's of instructions.
- If successive loop iterations are control independent, then they can be vectorized for pipeline execution or can be executed on a SIMD machine in a lock-step fashion
- Most optimized program construct to execute on a parallel or vector computer

Procedure level:

- This is at the medium grain size level; less than 2000 instr.
- SIMD is less common here
- Multitasking belongs to this category
- Cumbersome task to extract the dependencies at this level; lots of effort from the programmer is expected to restructure/organize the program

Subprogram level:

- Typical of exploiting parallelism by the algorithm designers and no compiler support is provided

Job level:

- Independent jobs on different processors
- Coarse grain size level
- Time-sharing and space sharing is exploited at this level;

Note: Fine grain provides a higher degree of parallelism however, the penalty is the higher communication overhead

Refer to Fig. 2.5 on page 62 for the hierarchy

Communication latency : This is the latency incurred with the inter-processor communication. IPC is also affected by the communication patterns besides the delay in the path.

In general, n tasks communicating with each other may require n(n-1)/2 links among them, which implies that the communication latency grows *quadratically*. This naturally sets a bound on the number of processors that can be used in the system.

- Broadcasting, multicasting are some common types

Grain packing and scheduling

- *How can a program be partitioned into parallel branches, modules, microtasks or grains, such that the finish time of the entire program is a minimum?*

- *What is the optimal size of the concurrent grains in a computation?*

Kruatrachue and Lewis (KL, 1988) approach for grain packing and scheduling for parallel programming

Refer to Example 2.4 and Example 2.5; Pages 64-70

Program flow mechanisms

Reading Assignment : *Pages - 70-75 (inclusive of Section 2.3.3)*

Network Architectures, Properties and Routing

Several computers can form a network interconnected via a medium and this interconnection can assume any specific or an arbitrary topology.

- Static and dynamic networks

Some common terminology and definitions :

1. ***Distance (u,v)*** specifies the distance between two nodes u and v in some metric that depends on the problem.

Example: The distance metric could be the shortest path between the nodes u and v

Note: In the rest of the definitions, we shall assume that the distance(u,v) is the shortest path between two nodes u and v, for the ease of understanding

2. Two paths between u and v are said to be **node-disjoint** if they have no other common nodes

3. Diameter (d) of a network is defined as,

d = max{Distance(u,v)}, where the max is determined over all pairs u,v $\in$ V

It is a measure of the network performance in terms of worst-case communication delay.

4. Degree, *deg(u)* of a node u is the number of links incident on it. (in-degree and out-degree in a directed G)

If deg(u) = δ, for all the nodes in the network, then the network graph is called as *δ-regular*.

5. For a regular network, the **cost** is defined as C = d.δ

6. ***Packing density*** of a network is defined as the ratio of the number of nodes to its cost. (nodes per unit cost)

Thus, higher the packing density, smaller the chip area required for its VLSI layout.

7. ***Node-connectivity (K)*** is the number of nodes whose removal results in a disconnected network.

Node-connectivity is a measure of fault-tolerance.

8. *f-fault diameter* of a network is defined as its worst-case diameter by removing at most ***f*** nodes

<u>Question</u>: Verify the above-mentioned quantities for some standard graphs - mesh, hypercube, ring, etc

---

Multistage Interconnection Networks (MINS)

• Exclusively used to interconnect a set of processors and memory modules - Various types (CLOS, baseline, omega)

Note: This portion of the material is my own compilation. ***Follow the slides presented in the class. U can get these from Course web site!***

If time permits, we will perform *blocking probability computations*

For Exams: It is sufficient if you go through my slides for this portion of the material that is discussed in the class.

Effect of Granularity on the Time Performance in Multiprocessor Systems

Download the slides from the web for this portion.

Catch the link:

“ ** Notes on the effect of granularity on the time performance (Chapter 2) “

ANNEX
(A) Scheduling using Flow Models – Scheduling DFGs

Use of flow models can solve our scheduling problem

Max-Flow Min-Cut Theorem

We will see a 2-processor assignment problem and learn how flow models derive an optimal, if not a feasible schedule!

See the PDF – download from Chapter 2 from our course page *Download Zone*!

(B) Mapping tasks onto a Multiprocessor system – Decision making under data certainty

We will formulate and understand how task mapping is carried out in a multiprocessor environment;

We will study the approach of decision making using the estimates put forth about the computing system by the applications/tasks;

Importance of this study – reducing the surprises for the Compiler!

Download the slides – Chapter 2 *Download Zone*!