Unit - 6 PAA

→ Dynamic multithreading : It allows the programmer to specify parallelism in the application without worrying about communication ~~protocols~~ protocols load balancing and other vagries of static - thread programming.

It support two feature.

(1) Nested parallelism: It allows a subroutine to be spawned allowing the caller to proceed while the spawned subroutine is computing its result.

(2) parallel loops : It is like an ordinary for loop except that the iteration of the loop can execute concurrently

Spawn : If spawn precedes a procedure call, then the procedure instance that execute the spawn (the parent) may continue to execute in parallel with the spawned subroutine (the child) instead of waiting for the child to complete.

Sync :- It bindicate that the procedure must wait for all of its spawned children to complete

parallel : it indicate loop body can be executed in parallel

eg Fibonacci(n)

if n<2 then return n;
x = spawn fibonacci(n-1); // parallel execution
y = spawn fibonacci(n-2);
sync;
return x+y;

* multithreaded computation can be represented using Directed acyclic graph G = (V, E) DAG.

V = instruction   E = dependencies b/w instructions.

An edge (u,v).E . instruction u must execute before instruction v.   ↳ can't done parallel

Strand : A sequence of instructions containing no parallel control (spawn, sync, return from spawn, parallel) can be grouped into a single strand.

A strand of max. length will be called thread.

Performance Measure

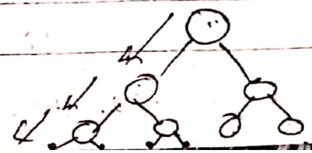
Work : Total time to execute the entire computation on one processor.

Work = sum of the times taken by each thread

Span : is the longest time to execute the threads along any path of the computational DAG.

Performance not only depend on work and span but also depend on no. of processor and how scheduling is perform.

eg.



work = 11
span = 4

T_1 = running time on single processor

T_p = running time on P processors

T_∞ = ——— infinite processors.

(I) Speedup on P processor: how many time faster the computation is than on 1 processor.

$$= \frac{T_1}{T_p}$$

(II) work law =

$$T_p \geq \frac{T_1}{P}$$

[Speed up of an algorithm on P processor can be no better than the run time with a single processor divided by the no. of processor P.

(III) Span law

$$T_p \geq T_\infty$$

[T_p can't run faster than infinite no. of processor]

(IV) parallelism.

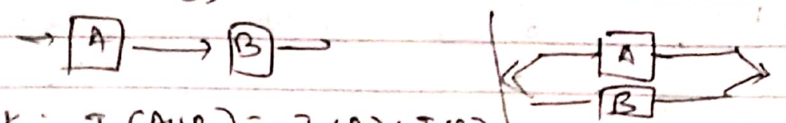
$$\frac{T_1}{T_\infty}$$

(V) Slackness.

$$\frac{T_1}{P.T_\infty}$$

Analysing Multithreaded Algorithm.

Series | parallel



Work: $T_1(A \cup B) = T_1(A) + T_1(B)$ | Work = $T_1(A) + T_1(B)$

Span: $T_\infty(A \cup B) = T_\infty(A) + T_\infty(B)$ | Span = $\max(T_\infty(A), T_\infty(B))$

parallel loops - parallel execution of iteration.

Race Condition

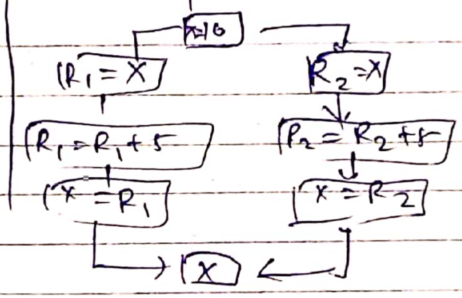
Deterministic Algo. → Same o/p for same i/p

Non deterministic algo - diff. o/p for same i/p.

Determinacy race occurs when two logical parallel instruction. access the same memory location, and atleast one of them performs a write operation.

Algo.

```
= Race_condition()
    x ← 10
    parallel for i ← 1 to 2 do
        x ← x+5
    end
```



Solution :- parallel loop should be independent

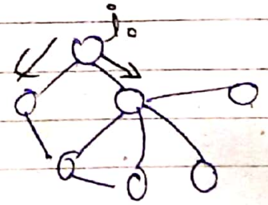
:- Spawn and sync, child should be independent of the parent

Distributed Algorithm

run on multi processor environment, processor have local memory and they communicate with each other via message passing.

→ It require communication cost.

Distributed BFS



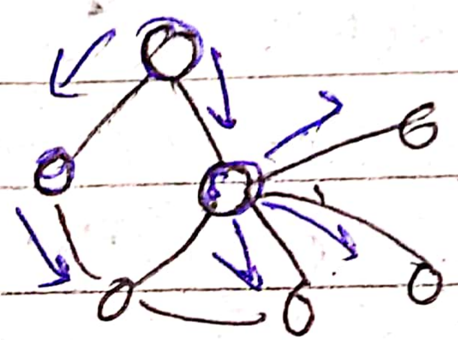
Round 1: i_0 send search msg to its neighbours.

Check unmarked then mark itself and update parent.

If marked cancel msg.

Scanned with CamScanner

Round 2 : The node who receive the Msg in round 1 Sends Search Msg to its outgoing neighbor..



At last they merge to form final set N.