in 2 or less tries you'll get good hash function.

QUIZ 1 DISCUSSION (SKIPPED)

Lecture 13: 19th February 2020

Parallel Computing

Computer Architecture

└ increasing clock frequency but memory performance doesn't scale up then processors sits idle (Memory Wall)

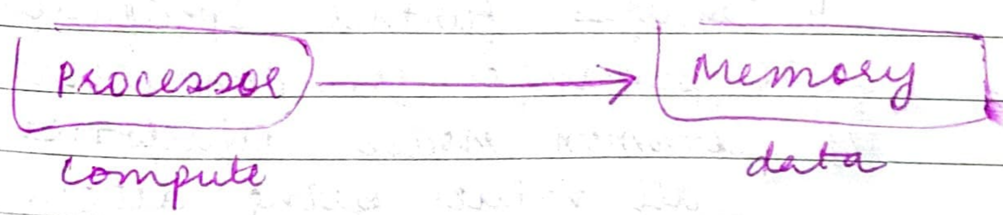
└ if ↑se clock frequency, not able to catch up with instructn level parallelism (ILP wall)

└ dissipated power will become much more significant (Power Wall)

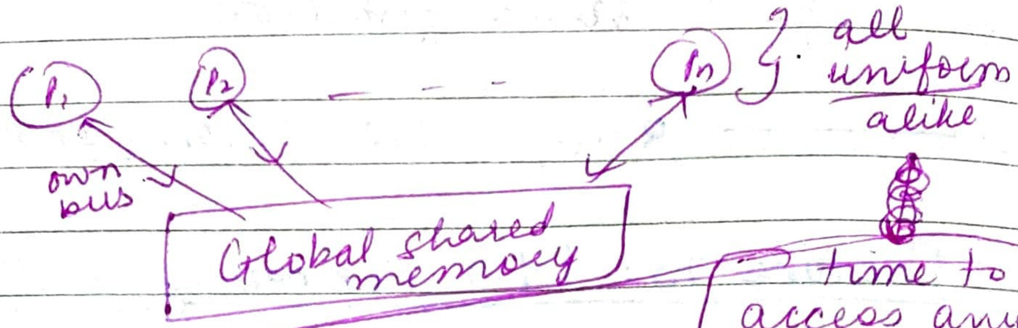
MW + PW + ILP = Brick Wall

The PRAM Model (Parallel RAM)

(1) └ RAM Model or von Neumann Model



② PRAM



Assumption: time to access any part of memory is uniform

— Model PRAM

① n identical processors
② common access shared memory
③ synchronous time
④ access to shared memory has same unit cost
⑤ diff models to provide semantics for concurrent access to shared memory.

EREW = Exclusive Read, E write
CREW : Concurrent Read, E write
CRCW :

└ requires further specification of semantics for concurrent write

① Common Model – CW allowed if all values being attempted are equal eg. boolean OR of n bits
② Arbitrary = some processor succeeds and its write takes effect

Date
Page

i.e. arbitrary one will take effect.

③ Priority – assumes that processors have numbers that can be used to decide which succeed
└ Priority Rule

Example 1: Matrix Multiplication

DO THIS FROM BOOK / SLIDES

$$C[i,j] = \Sigma A[i,k] \cdot B[k,j]$$

computation of each element in matrix C are independent.

Another approach: Cannon's algo.

DO DIVIDE & CONQUER ALGOs

Example 2: Prefix Computation

Sequential = $O(n)$

→ associative operator '$\circ$'

in any order
$a \circ (b \circ c) = (a \circ b) \circ c$

```
S[1] = A[1]
for i = 2 to n do
    S[i] = S[i-1] o A[i]
```

Date
Page

Why cannot the seq program used in parallel
- because each prefix index is not ~~idependent~~ independent
  you need s[i-1] to find s[i]

Need a new algorithm approach
- Balanced Binary Tree design paradigm
  - can be used to solve a lot of problems.
  - dependencies of one index on another can be largely avoided.

→ Logical (not physical) arrangement of processors in form of a complete binary tree (processors at internal nodes)

↓

~~Inputs~~ Operations are executed at interal nodes and inputs (of an operation at a node) are values at children of this node.

→ main input at leaves
first element – first leaf
last " last leaf & so on

→ an operation associated c̄ each processor at internal node —

these operations need not be identical.

→ since input to each node is its children — unless values at children are not known — parent cannot compute — so there is some waiting involved.

→ So parents of leaf nodes are ready to execute because they have values available at their children.

→ so computation is going ~~…~~ (tree traversal) from leaf to root and at every time step all nodes at one level can do their computation

Upward Traversal,
- data flow from children to root
- used in finding
  1. max
  2. expression evaluation } do these

Downward Traversal
- ~~root to leaf~~
- useful for element broadcast
  - root processor informs other processors about something

→ Prefix Computation uses both upward and downward traversal

even indices
↳ ~~copy~~ parent ~~pass~~ processors informs right child (parallely)

odd indices
↳ ~~cop~~ left uncle processor informs nephew & nephews adds it to its value

OR

↳ subtract sibling from parent

using recursion as part of downward traversal
where output of a particular level in binary tree can be used to create output at next level starting from root.

Lecture 14: 2nd March 2021

Step 1: for ($i = 1$ to $n/2$) { // parallel
$b_i = a_{2i-1} \circ a_{2i}$
}

~~Step 2: Save these in c_i~~

Step 2: find b_i recursively & store in c_i

Step 3: for ($i = 1$ to n) { // parallell
if (i is even) $s_i = c_{i/2}$
else if ($i = 1$) $s_1 = c_1$
else if (i is odd) $s_i = c_{(i-1)/2} \circ a_i$
}

Date
Page

Analysis : ① Time

- Each operation — 1 unit
- as many processors

Step 1 : $O(1)$ — $n/2$ processors
Step 2 : Recursive call — $T(n/2)$
$= O(\log n)$
Step 3 : $O(1)$ — n processors

So, $T(n) = O(\log n)$

② Work done
└ sum of work done by each processor

step 1 = $\frac{n}{2} \times O(1) = O(n)$
step 3

step 2 = ~~W(n)~~ $W(n/2)$
$= O(\log n)$

$$\boxed{W(n) = O(n)}$$

W(n) indicates if algorithm is doing about same amt of operations as best known sequential algorithm. Such a parallel algo is called an optimal algorithms.

~~15~~
~~16~~
~~17~~
~~18~~
19

$A_{\text{Parallel} \to \text{Seq}}$ $\quad$ A_{seq}

if $\boxed{T_{P \to S} = T_S}$ ← work optimal

OTHER DESIGN PARADIGMS

Partitioning

└ similar to divide and conquer

D&C
- Divide
- Solve
- combine

Partitioning
- Divide
- Solve

eg. quicksort is example of sequential partitioning

In parallel algos, each subproblem that we get out of divide step can be treated independently & solved in parallel.

eg. Parallel merging, searching

MERGING IN PARALLEL

Two sorted arrays A and B to be merged into C.

Rank (x, A) = no. of elements smaller than x in A

x ↓ ele, A ↓ sorted array