Parallel Algorithms

" A parallel algorithm is a solution method for a given problem destined to be performed on a parallel computer. "

Need for Parallelism

→ There is a physical limit on the computational speed and packing density.
→ The maximum theoretical speed is the speed of light beyond which the links connecting the processors start to bottleneck.
→ Thus by distributing several operations to multiple processors time taken for computation can be reduced.

Application

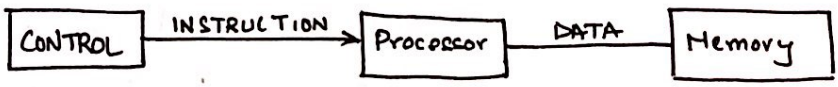
- <many>

Models of Computation

Any model for computation can be split into the instruction and the data And the interaction between them.

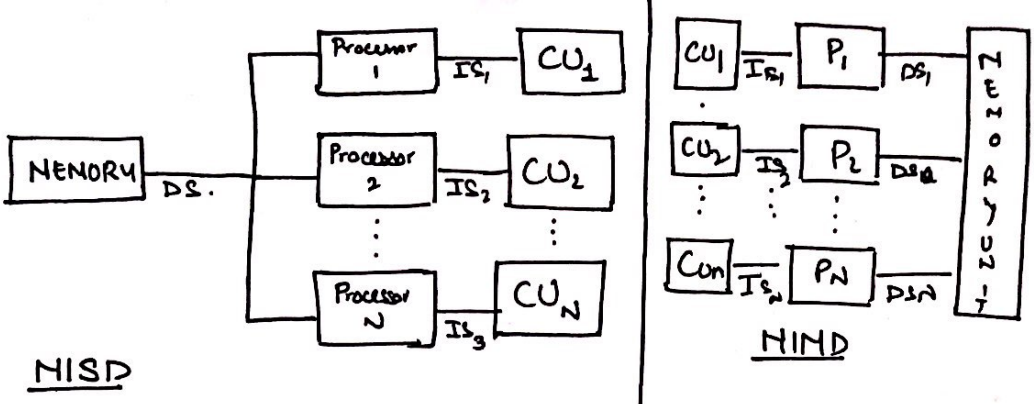
1. Single Instruction Single Data (SISD)

- A single processing unit consisting of a single input data stream.
- The control unit emits one instruction that operates on data from the memory
- Also called a sequential computer / Serial



2. Multiple Instruction Single Data (MISD)

- N processor with their own control units share a common memory
- Best used when performing multiple (diffrent) operations on the same data like classification.
- In practice these are very awkard to use.



MISD

MIMD

3. Multiple Instruction Multiple Data (MIMD)

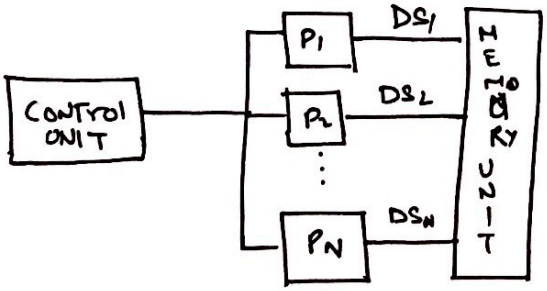
- These are the most general and powerful type of parallel processing systems.
- Each processor has its own instruction stream and can work independently of some data i.e the processors work asynchronous.
- The processors can be using shared memory i.e Multiprocessors. or using interprocro communication i.e Multicomputers (distributed).

4. Single Instruction Multiple data (SIMD)

- Each processor has its own local memory.
- All processors are under the control of a single control unit.
- All processes do the same operation on different data.
- Some processors can be set to an inactive state when they are not needed.
- The proceses can share data by Shared Memory or IPC.

4.1 Shared Memory SIMD

- Also called Parallel Random Access Machine (PRAM).
- Two processes use a bulliten board approach to share memory i writes to memory and then j reads it.
- This system is vulnerable to inbetween updates and race condition.
- There are 4 sub classes to SM-SIMD systems.



4.1.1 Exclusive Read Exclusive Writ (EREW)

- Access to the Memory is exclusive. No simultanious Read/Writ is allowed. N.
- Much like a critical Section.

4.1.2 Concurrent Read Exclusive Wreik (CREW)

- Writ access is exclusive but reading is concurrent.
- Much like Locks.

4.1.3 Exclusive Read Concurrent Writ (ERCW)

- exclusive access for reading but writing can be simultanious

4.1.4 Concurrent Read Concurrent Writ (CRCW)

- Anarchy.

- .3 and .4 are prone to writ conflicts and race conditions and requir a way to safeguard.

1. Oldest process goes first
2. Writ only when all valus are same (sync)
3. Ensure constant sum.

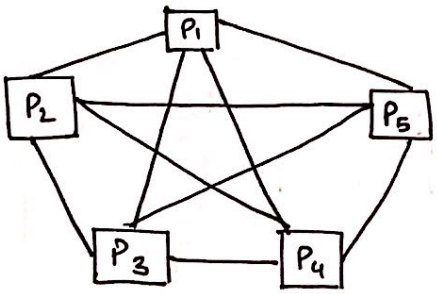
EREW is the weakest but safesest system.

4.2 Interconnections in SIMD

- Blocks of Memory are divide among N processors.
- Each processor must contain

1) a circuit cost of f(N-1) to decode a log(N-1) bit address
2) a f(M/N) cost of decoding a log(M/N) bit address.

1. Fully Connected

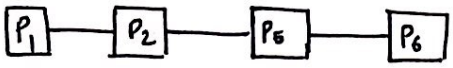


N-1 connection per node

$\frac{N(N-1)}{2}$ connections total.

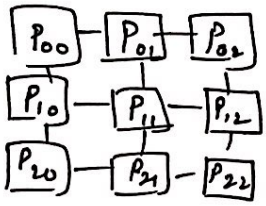
4.2.1 Linear Array

1. Linearly connected to its two neighbours
2. Serial execution



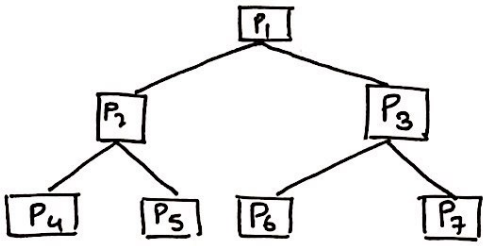
4.2.1 Two Dimensional Mesh

1. mxn dimensial arrangement of N prossesors $m = \sqrt{N}$.
2. All processors are equidistant to prevent abnormalities due to attenuation.
3. This is also called a mesh.



4.2.3 Tree Connection

1. Maintained as a complete binary tree. with d levels $d = \sqrt{N+1}$

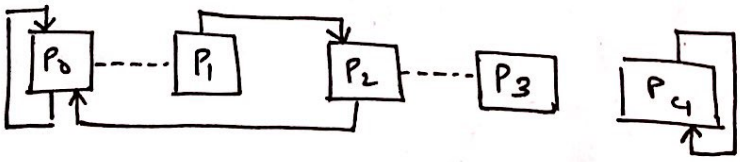


4.2.4 Perfect Shuffle

1. There are two types of links a) shuffle links b) exchange links
2. Suffle for P_j and P_i are.

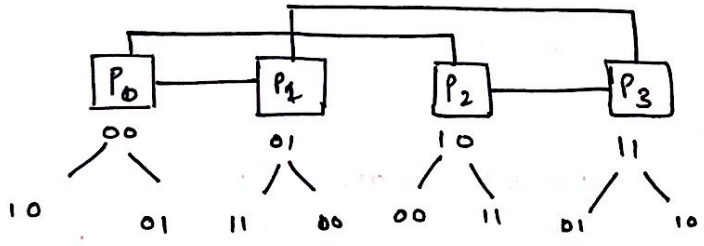
$$j = \begin{cases} 2i & 0 \leq i \leq N/2 - 1 \\ 2i + 1 - N & N/2 \leq i \leq N-1 \end{cases}$$

3. Exchang links are every even processor is connected to its successor.



4.2.5 Cube Connection

- A q dimensional cube where $N = 2^q$.
- To get the connections represent the pid as binary and one by one flip every bit.



Analyzing Parallel Algorithms

1. Running time

it has two components

1) computation time.
2) Routing time.

One key measure is the speed up.

$$\text{Speed up} = \frac{\text{Worst case time of serial (best}}{\text{WORST CASE of Parallel}}$$

2. Number of Processors

The higher the no. of processor faster is the execution but the expense is high as well.

$$\text{cost} = \text{parallel Run time} \times \text{No of processors}$$

Cost optimal: if the cost of P.A matches the lowerbound of a sequential one then it is said to be cost optimal.

$$\text{Efficiency} = \frac{\text{Worst case of Sequential}}{\text{cost of Parallel}}$$

Expressing Parallel Algos

- There are two types of statements, 1. which are serial like cases, if-elses etc and 2. parrallel.

1. For sequential we use standard notations.

2. For parallel we have two sub cases.

   1. Several step are done in parallel.

```
do steps i to j in parallel
        step i
        step i+1
          :
        step j
END
```

   2. Several Processors are to do some oprations parallel

```
for i= x to y do in parallel
    { task of Pi }
END
```

Dense Matrix Operations

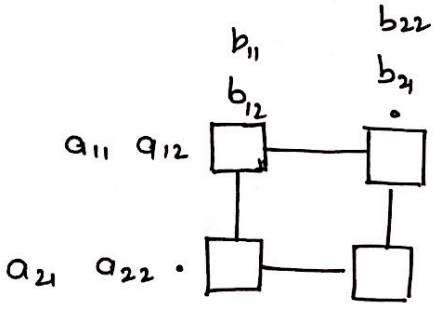
- The goal is to multiply two matrices A (m×n) and B (n×k) this gives a (m×k) matrix C.
- The serial naive approach is $O(N^3)$ and the best concieved algos offer $O(N^{2.7})$ and $O(N^{2.37})$ time but are fairly impractical.

```
MULTMAT (A,B,C)
for i = 1 to m do
    for j = 1 to k do
        c_ij = 0
        for s = 1 to n do
            c_ij += a_is * b_sj
```

- Parallel Matrix–Matrix Multiplication

1. A m×k grid of processors with input at the left and top boundary
2. Rows of A correspond to the rows of the grid.
3. Cols of B correspond to the cols of the grid.
4. Initially C is initialized to 0.
5. Whenever processor P_{ij} gets input a,b it multiplies them and adds to C_{ij}.

```
MeshMatrixMult (A,B,C)
  for i = 1 to m do in parallel
    for j = 1 to k do in parallel
        c_ij ← 0
        while P_ij has input a,b
            c_ij += a×b
            if i < m send b to P_{i+1,j}
            if j < k send a to P_{i,j+1}
```



The • are delay bits make them = 0 if doing imp.

Complexity: at worst an element a or b will take m+k+n-2 step to propogate.

∴ Time Complexity = $O[\max(m,k,n)] = O(n)$

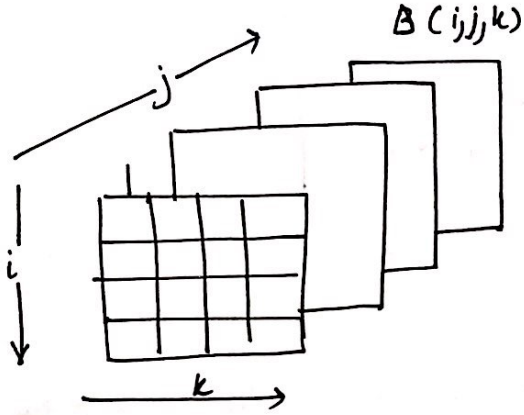
Cost = $O(n) \times O(m \times n) \quad O(nmk)$

Hyper cube Multiplication

- We have $N = 2^q$ processors for $q \geq 1$.
- we can multiply a $n \times n$ matrix with $N = n^3 = 2^{3q}$
  i.e for a 4×4 matrix we will have 64 process in a 2^6 $q=6$ dimension hyper cube.
- This does the n^2 internal loop of n^3 mulliplication simultaniously.
- A and B are distributed over the n^3 processors as

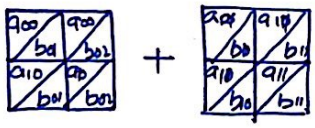
$$A(i,j,k) = a_{ji}$$

$$B(i,j,k) = b_{ik}$$



i.e ~~copy data~~

if $n=2$ $N=8$ $q=1$



$k=0$ $j=1$

= mulliply values.

Hyper Cube Mult (A,B,C):

1. for $m = 3q-1$ down to $2q$ do
   for all r in $\{N, r=0\}$ in Parallel
   $A_r^m = A$
   $B_r^m = B_r$
   ] distribute the data
2. for $m = q-1$ down to 0
   for all r in $\{N, r_m = r_{2q+m}\}$ in Parallel
   $A_{r^m} \leftarrow A_r$
3. For $m = 2q-1$ down to q
   for all r in $\{N, r_m = r_{q+m}\}$ in Parallel
   $B_r^m \leftarrow B_r$
   ] distribute
4. For all $r=1$ to N in Parallel
   $C \leftarrow A \times B$
   ] Mult
5. For $m = 2q$ to $3q-1$ do
   for $r=1$ to N in parallel
   $C \leftarrow C_r + C_r^m$
   ] Add

Complexity -

1. The distribution, & merg take $\log(n)$ time to propagate and merge
2. Mulliplication is constant

$\therefore$ time = $O(\log(n))$

cost = $O(n^3 \log(n))$

additions =

$$\sum_{i=1}^{s-1} 2^i = 2^s - 1$$

$n-1 < 2^s - 1$

$n \leq 2^s$

$s = \log(n)$

CRCW Multiplication

- TL; DR: PARALLELIZE EVERYTHING!!!!
- In parallelized implementation of the naive algorithm the addition and multiplication become constant time operations.

```
CRCW MULT (A,B,C)
for i=1 to m in parallel
    for j = 1 to k in parallel
        for s= 1 to n in parallel
            cij= 0
            cij = aisx bsj
```

Complexity -

- The multiplication and addition steps are constant

time =	O(1)
cost =	O(n³)

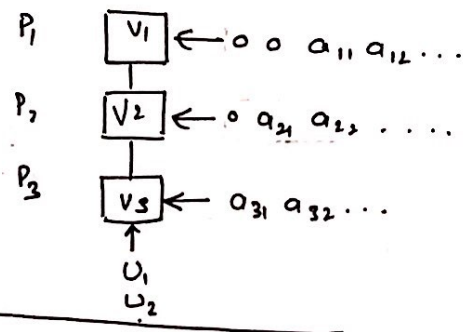
Matrix Vector Multiplication

given a mxn matrix A and nx1 vector u we wish to find mx1 vector V which is Axu.

1. Linear array Multiplication

- Similar to mesh matrix.
- we have a m element vector of processor with rows of matrix as side inputs and the vector as col input.

```
Lineal MatVect(A,U,V)
  for i=1 to m do in parallel
      Vi ←0
      while Pi has input aand u
          Vi+= axu
          if i>1 send u to Pi-1
```

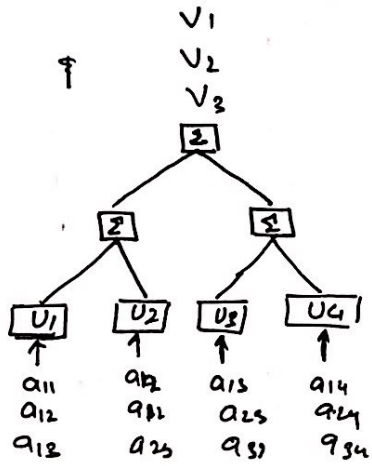


Complexity :

time =	O(m+n-1)	max propagation
cost =	O(m x (max(m,n)) =	O(n²).

Tree Multiplication

- This improves the O(n) prop time to log(n) by restructuring to a tree SIMD.
- The leaves store the vector U and get the matrix as input
- at the leaf Multiply the values and propagate up.
- at any inner node, add the incoming value and propagate up.




```
TreeMatVect(A, U, V)
  do 1 and 2 in parallel:
      1) for i=1 to n in parallel
           for j=1 to m
              send U_i x a_ji to parent

      2) for i=n+1 to 2n-1 in parallel
           while P_i has two inputs
              find the sum
              if i < 2n-1 send result to parent
              else output result.
```

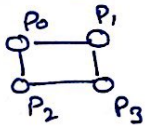
Complexity: log(n) time to get output and m multiplication happen

Time =	$O(m + \log n - 1)$
cost =	$O(mn^2)$

Parallel Sorting Algorithms

1. Hyper Quick Sort

- The quick sort paradigm is done on a hyper cube.
- The array is divided amongst each node of the cube.
- Each node is sorted locally.
- Select a median from the 0^{th} node and broad cast to all other nodes.
- Split the list locally based on this pivot
- Swap the halves across the highest dimension
- Repeat till the higes dimension is 0

  i.e if there's 4 dimensions 

  then ↕ first and then ↔
- Sort all the segments.

Complexity

Time: The splitting phase will take $n/p \log p$ time for each n/p segments it'll be done $\log p$ times.

The broadcasing overhead of the system will be $\log^2(p)$

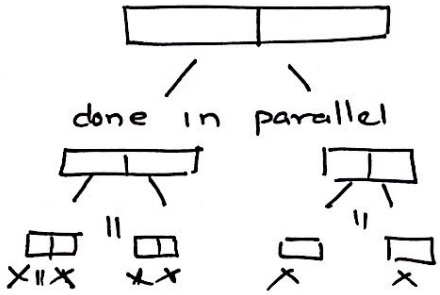
The cost of individually sorting the segments is $n/p \log n/p$

$$\Rightarrow \quad t(n) = \underbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}_{\text{sort.}} + \underbrace{\Theta\left(\frac{n}{p}\log p\right) + \log^2 p}_{\text{split \& Broad Cast}}$$

$$\text{cost} = t(n) * p$$

Parallel Merge Sort

- The parallelizes the spitting process of the Merge sort procedure



// the recursion stops at N=2 because the Merge can be directly be called.

```
Parallel Merge Sort (A):
    S ← 2
    while S ≤ N do in parallel with N/s processors
            merge S size blocks of A
        S ← S×2.
```

Complexity

Time : assume at worst $N = 2^k$ (because the bin tree Nature)

k is the levels of recursion = $\log(n)$

each loop call has a merge call with geometrically increasing size = 2^i

∴ Total work done =

$$\sum_{i=1}^{k} 2^i = \sum_{i=0}^{k} -1$$

$$= 2^{k+1} - 1 - 1$$

$$= 2 \cdot 2^k - 2 = 2(2^k - 1)$$

$$= 2(2^{\log N} - 1)$$

$$= 2n - 1$$

$$= O(n)$$

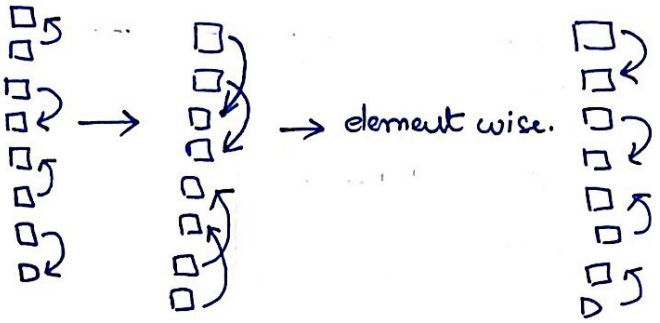
cost: the lowest level will call for $N/2 = O(N)$ processors

∴ cost = $O(n^2)$.

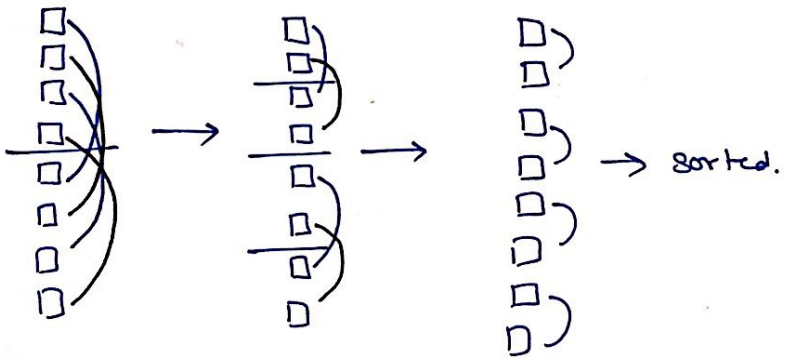
Bitonic Merge Sort

1. This is a two step process
    1. Build Bitonic Sequence
    2. Build sorted array from Bitonic Sequence.

2. Building the Bitonic Sequence.
    1. Consider consequtive elements and apply the rule.
    2. Compare over alternate element



3. Build Sorted list
    1. Compare each corresponding element of each half.
    2. Compare each alternate element of each half.
    3. Compare each consecutive element.



Complexity

Time: To build the sorted list we need $\log(n)$ comparisions.

$$T(n) = \log(n) + T(n/2) = \log(n)\,[\log(n)+1]/2 = \log^2(n)$$

if N is Not a pow of 2 then add int_MAX / some flags to make it a pow of 2

Odd-Even Transposition

- We use n processors to sort a Sn sized list.
- The process is simple and has two alternating steps when $x_{i+1} < x_i$ then swap.
- This is first done with odd no places & then with even.

```
ODDEVENTRASP (S)
  for j = 1 to n/2 do
(1) for i = 1,3... 2⌊n/2⌋ - 1 do in Parallel
        if xi > xi+1
          xi ↔ xi+1
(2) for i = 2,4,..., 2⌊n-1/2⌋ dom Parallel
        if xi > xi+1
          xi ↔ xi+1
```

Complexity

1. Both (1) and (2) are constant time operations.
2. The operations are done for $\lceil n/2 \rceil$ times

Time = $O(n)$

cost = $O(n^2)$

Enumeration Sort

→ CRCW sort

→ There are n^2 processors to sort on an array $S = \{S_1 \ldots S_n\}$

→ Uses the logic of Enumeration.

- Position of S_i is determined by computing C_i i.e the number of elements smaller than it.
- Once all C_i have been computed, S_i is placed in position $C_i + 1$.

#> Input array = S

#> Count = C

I^{th} row of processors in charge of S_i to compute C_i.

```
CRCW_Sort(S)
1. for i=1 to n do in parallel :
       for j = 1 to n do in parallel :
           if (Si > Sj) or (Ci = Sj and i > j)
               P(i,j) writes 1 to Ci
           else
               P(i,j) writes 0 to Ci
2. for i=1 to n do in Parallel :
       P(i,1) stores Si to (Ci+1)th of S
```

Complexity :

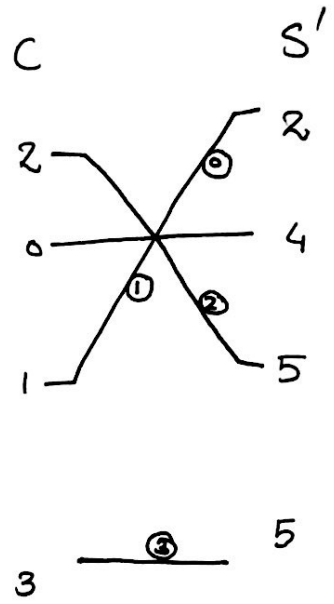
1. Both 1. and 2. are constant time operations
2. There are n^2 processors being used.

Time =	$O(1)$
Cost =	$O(n^2)$

Eg > $S = \{5, 2, 4, 5\}$

$n = 4 \quad P_n = 16$

S	P	p	1	2	3
5	p	5,5 •	5,2 1	5,4 1	5,5 0
2	2	2,5 0	2,2 0	2,4 0	2,5 •
4	3	4,5 0	4,2 1	4,4 •	4,5 0
5	3	5,5 1	5,2 1	5,4 1	5,5 0



Searching -

• Sorted Sequences

A : EREW Searching -

- Uses N processor ($KN \leq n$).
- Each processor is assigned a n/N length sub sequence.
- The key value is broadcasted to all processor in $O(\log(n))$ time.
- Each processor does Binary Search on its sub sequence.

> EREW_Search (S, x) :
>
> 1. Boadcast (x)
> 2. $P_i = \{ S_{(i-1)(n/N)+1}, \ldots, S_{(i)(n/N)} \}$
> 3. Each P_i does Binary search.

Complexity -

Step 1. takes $O(\log N)$ time

Binary Search takes $O(\log(n/N))$ time

$\therefore$

$$\text{Time} = O(\log(N))$$

$$\text{cost} = O(N \log(N))$$

B : CREW Searching -

- There are two ways
  1. Same as EREW but now step 1. will take $O(1)$ time thus the total time will be $O(\log(n/N))$.
  2. Using an n+1nary Search.

- Each processor splits the seq into two parts, keep and discard.
- let g be the smallest int s.t $n \leq (N+1)^g - 1$ or $g = \left\lceil \frac{\log(n+1)}{\log(N+1)} \right\rceil$
- Each seq of len $(N+1)^g - 1$ is searched.
- P_i compares x to S_j ; $j = i(N+1)^{g-1}$.
- Each processor P_i keeps a c_i to show which part to keep. initially c_i is irrelevant. $c_0 = r$ and $c_{N+1} =$ left are kept as well.
- P_i assigns a value to c_i ; if $c_i \neq c_{i-1}$ for some i ; $1 \leq i \leq N$ then S_q and S_r is kept. $q = (i-1)(N+1)^{g-1}$; $r = i(N+1)^{g-1} - 1$.

```
CREW_Searching (S, x)
1.  q = 1
    r = n
2.  k = 0
    g = ⌈log(n+1) / log(N+1)⌉
3.  while q ≤ r and k = 0 do:
        3.1 j_0 = q - 1
        3.2 for i = 1 to N do in Parallel:
                a) j_i = (q-1) + i(N+1)^(g-1)
                   //compare S_j ? x
                b) if j_i ≤ r:
                       if S_{j_i} == x:
                           k = j_i
                       else if S_{j_i} > x
                           c_i = left
                       else c_i = right.
                   else:
                       j_i = r+1
                       c_i = left
                c) if c_i ≠ c_{i-1}
                       q = j_{i-1} + 1
                       r = j_i - 1
                d) if i == N and c_i ≠ c_{i-1}:
                       q = j_i + 1
        3.3 g ← g-1
```

Analysis

- 1, 2, 3.1 & 3.3 are constants ; 3.2 takes constant as well.

Time : $O(\log(n+1)/\log(N+1))$	cost = $O(N \log_{N+1}(n+1))$

Searching A Random Sequence

- SM-SIMD computers

- The general algorithm for search is the same i.e

```
Sm Search (S, x, k)
1. for i = 1 to N do in parallel:
        Read X
2. for i = 1 to N do in parallel:
        S_i = { S_{i-1 N/N} ... S_{i N/N} }
        Sequential Search (S_i, x, k)
3. for i = 1 to N do in parallel:
        if k_i > 0  k ← k_i
```

• EREW model

Step 1 is done using broad cast ∴ takes $O(\log(N))$
Step 2 is done using standard linear search ∴ $O(n/N)$
Step 3 is done using the store procedure ∴ $O(\log(N))$

time: $t(n) = O(\log N) + O(n/N)$
cost: $C(n) = O(N \log N) + O(n)$

• ERCW model

1 WHY DOES THIS EXIST.

Step 1 is Broadcast ∴ $O(\log(N))$
Step 2 is same $O(n/N)$
Step 3 is concurrent ∴ $O(1)$

time = $t(n) = O(\log N) + O(n/N)$
cost = $C(n) = O(N \log N) + O(n)$

• CREW Model

Step 1 is constant ∴ $O(1)$
Step 2 is same ∴ $O(n/N)$
Step 3 is STORE ∴ $O(\log n)$

time = $O(\log N) + O(n/N)$

cost = $C(n) = O(N \log N) + O(n)$

CRCW Model

Step 1 is constant $\therefore O(1)$

Step 2 is same $O(n/N)$

Step 3 is constant $\therefore O(1)$

$$\text{time} = t(n) = O(n/N)$$

$$\text{cost} = C(n) = O(n)$$

This model is cost optimal.

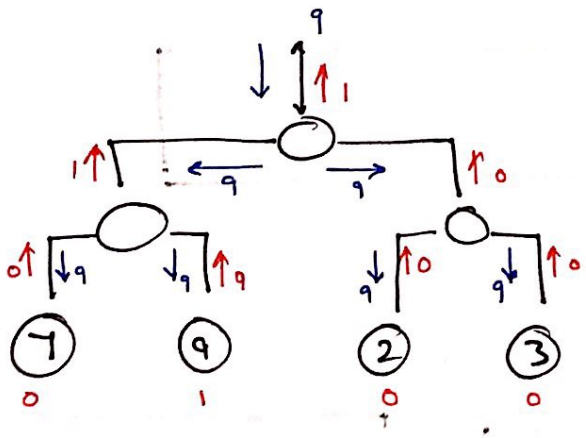
+ querying in the above takes a multiple of q times log N time this can be improved to be of the order of $\log n + q - 1$ using the tree method.

On Tree Model

- There are n leaves of the processors. (n^2)
- Each leaf has one entry.
- The root handles all inp/out.
- The internal node only act as merge/split cells.

1. Root Reads x & passes it to children which propagat x to all leaves $(O(\log(n))$
2. The leaves simultaneous do a comparison of x with their value producing a binary $(O(1))$
3. The output of leaves is combined and sent up to the root $(O(\log N))$

Eg



Mesh Search

- Constant wire length
- area used is O(n)
- running time and manhanona is $O(\sqrt{n})$.

Querying

- n^2 processors, $P_{i,j}$ has element $S_{i,j}$.

Two step process:

- Unfolding
  - P(1,1) reads x, if matched produce an output $b_{1,1}=1$ else $b_{1,1}=0$
  - communicate $(b_{1,1}, x)$ to P(1,2) if $X = S_{1,2}$ or $b_{1,1}=1$ $b_{1,2}=1$ else 0.
  - Now this is done in || for $P_{1,1}$ P_{12} to P_{21} & P_{22}
- Folding
  - Reverse of unfolding, results are merged back into Root (P_{11})

Mesh Search (S, x, and):

1. $P_{1,1}$ reads input
   if $x = S_{11}$ $b_{11}=1$ else $b_{11}=0$
2. { unfolding }
   for i = 1 to $\sqrt{n}-1$ do:
   for j = 1 to i do in parallel:
   P(j,i) transmits (b_{ji}, x) to $P(j,i+1)$
   if $x = S_{j,i+1}$ or $b_{ji}=1$ $b_{j,i+1}=1$ else 0.
   for j = 1 to i+1 do in parallel:
   P(i,j) transmits (b_{ij}, x) to $P(i+1,j)$
   if $x = S_{i,j+1}$ or $b_{ij}=1$; $b_{i+1,j}=1$ else 0.
3. { folding }
   for i = $\sqrt{n}$ down to 2 do:
   for j = 1 to 1 do in parallel:
   P_{ji} sends B_{ji} to $P_{j,i-1}$
   for j = 1 to i-1 do in parallel:
   $B_{j,i-1} = B_{ji}$
   if $B_{j,i-1}=1$ or $B_{ji}=1$ $B_{i,i-1}=1$ else 0.
   for j = 1 to i-1 do in parallel:
   P_{ij} send B_{ij} to $P_{i-1,j}$
   for j = 1 to i-2 do in parallel:
   $b_{i-1,j} = b_{ij}$
   if $B_{i-1,i-1}=1$ or $B_{i,i-1}=1$ $B_{i-1,i-1}=1$ else 0
4. Produce output

Complexity-

- Steps 1 and 4 are constant time
- Steps 2 & 3 have $\sqrt{n}$ iterations

Hence $\boxed{\text{time } t(n) = O(\sqrt{n})}$

$\boxed{\text{Cost } c(n) = O(n^2\sqrt{n})}$

Selection

- Given a sequence S of n elements and k $1 \le k \le n$ we have to find the k^{th} smallest element.

Sequential Select (S, k):

1. if $|S| < Q$ sort S and return the k^{th} element
   else subdivide S into $|S|/Q$ chunks.
2. Sort each subseq & get its median.
3. call SeqSel recursively to find m i.e median of $|S|/Q$ medians.
4. create 3 subseq S_1 S_2 S_3 for less than, GT and Eq to m.
5. if $|S_1| \ge k$
   call SeqSel on S_1
   if $|S_1| + |S_2| \ge k$ return m
   else call SeqSel on S_3 to find $(k - |S_1| - |S_2|^{th})$.

Complexity-

1. = c_1
2. = $c_2 n$
3. = $t(n/Q)$
4. = c_3
5. = $t(3n/4)$

$\therefore$ $t(n) = c_4 n + t(n/Q) + t(3n/4)$

$t(n) = c_4 n + c_5 t(n/20)$

$t(n) = c_5 n$

$\therefore$ $\boxed{\text{time} = O(n)}$

Parallel Selection

- uses some N no of processors.
- Each processor has n and for some $N = n^{1-x}$ $0<x<1$.
- Each of n^{1-x} processors has n^x elements.
- Each processor can call Selseq.
- M is a N len array.

Parallel Select (S, k):

1. if $|S| \leq 4$: return the k^{th} element
   else :
   subdivide S into $|S|^{1-x}$ subseq. $1 \leq i \leq |S|^{i-x}$ $S_i = |S|^x$.
   assign S_i to P_i.
2. for $i = 1$ to $|S|^{1-x}$ do in parallel :
   P_i gets median of S_i
   P_i stores m[i] in M(i).
3. obtain m of M.
4. make 3 seqs :
   $L = \{ S_i \in S ; S_i < m \}$
   $E = \{ S_i \in S ; S_i = m \}$
   $G = \{ S_i \in S , S_i > m \}$
5. if $|L| \geq k$ call Parallel Select on L
   else if $|L|+|E| \geq k$ return m.
   else call Parallel Select on $(G, k - |L| - |E|)$.

Complexity

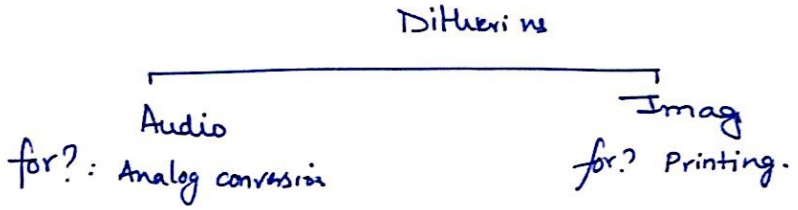
Time = $t(n) = O(n^x)$

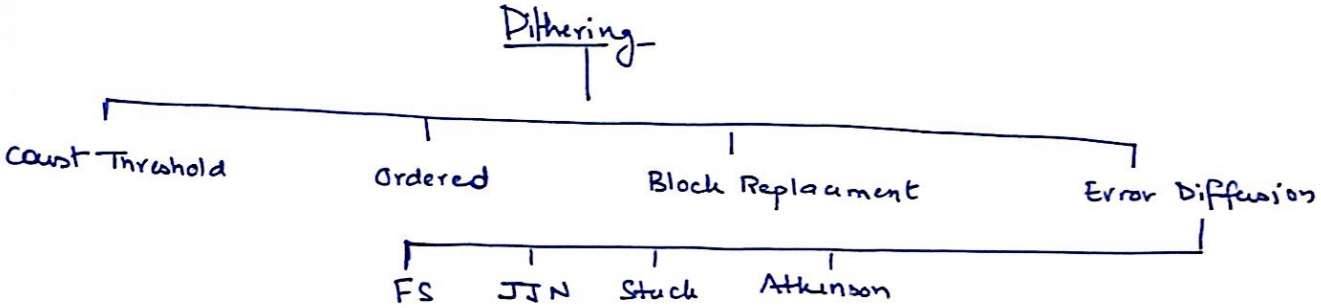
cost = $c(n) = O(n)$

$\therefore$ $x = 1 - \log(N) / \log(n)$

Image Dithering

- creating approx of peaks by spreading them over a region to create the illusion of continuity





Const threshold

↳ fixed value.

Random

> random

Ordered

follows a fixed pattern to turn pixels.

$$P_{ij} > T[i \% R, j \% R]$$

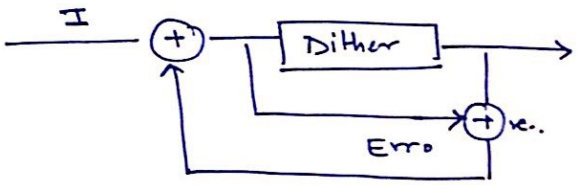
Block

There are predetermined Blocks.

Each pixel is replaced by one of those causes an upscaling effect.

Error diffusion

Errors of app^n are carried over.



15 puzzle

- Just like 8 puzzle but bigger.
- can extend to be an n puzzle version.
- Given : Goal, Init, allowed moves.
- NP complete → No Poly time soln.

Test for solvability :

1) if N is odd, then solvable if there even inversions
2) if N is even,
   → odd inv, blank @ even row
   → even inv, blank @ odd.

Inversion any [7|4] case. (any pair).

Algos,

→ B2B.
→ A*.
→ IDFS..
→ IDA*.

→ ||

→ || DFS, each process takes a diff path.
→ || IDA*
→ || Birectional search.

DB Query

- I/O Bottlenecks
- Disk to mem
- Wasting resources.

When to do ||?

→ || cost > serial i.e. omal query
→ operators that aren't ||izable.

DB

- Shared Mem
  + Multiple Proc
  + single memory
  + Any proc can access mem
  + queries can decompose
  — simple
  — Balances load
  — $$
  — ! Extensible
- Shared Disk
  + multiple mem
  + dedicated mem
  — low $
  — Load balance
  — Extensible
  — migratable
  — Complex
- Shared Nothing
  + decoded Ram + Rom
  + local
  — Available
  — Extensible
  — complex
  — NON load Balance
- Hybrid

operations

→ Partitioning
→ relations
→ Joins.

- Round Robin
- Range Index
- Hash

|| DES

- collection of simulated objects & a seq of event s
- changes to system only happen at discrete times
- The passage of time is modelled using a simulation clock.
- There is a list of events, scheduled & Not occurred.
- PDES is diff due to use of state variables, event list etc.
  - maintain causality while keeping || sm.
  - Synchronizing : If two events effect object @ same time they need to sync.

    Two approaches
    - ↳ conservative.
    - ↳ optimistic

Dense LU Factorization

- The Factorization of a matrix into the for

$$A_{nxn} = LU$$

- Invertible
- all leading matrices must be non zero → 
  - ↳ solved with reordering.

2 way

Doolittle & Gaussian.
- ↳ 1 convert to row echelon. U=
- ↳ 2 ways to find L

|| sm

- give a row to a processor.