Parallel Complexity Theory

Lecture 6

Reminder: Complexity (1)

- Number of 'steps' or 'memory units' required to compute some result
  - In terms of input size
  - Using a single processor
- O(1) says that regardless of input size, we only need constant time
- O(n) says 1 time step per input value
- O(n^2) says n steps per input value

Comparatorial Networks (1)

- Networks of connected comparators

Reminder: Complexity (2)

- NP hard problems
  - NP="non-deterministic, polynomial time"
  - Every NP hard problem is in the same 'class'
    - Prove that one NP hard problem is NP then all are NP.
    - Examples:
      - SAT (prove that (A v B) ^ (¬C v D) ^ etc = true for some A,B,C,D,etc
      - In general: everything that says:
        - » Find the optimal X under conditions A,B,C,D,E…
        - » Example: TSP, register allocation, instruction/process/message/room/class/etc scheduling

Graph Accessibility Problem (GAP) (1)

- Graph G =(V,E)
  - Vertices numbered 0 – n-1
  - Shared memory machine is given:
    - 0 <= ***u*,*v*** < n
    - Adjacency matrix ***A***
      - In a *G*, two vertices are adjacent if they are joined by an edge
      - '1' in matrix thus says if two vertices are adjacent
        - » A[i,j] == true iff (i,j) ∈E
  - Problem: find a path over the edges that matches some condition. We'll use "length(path) < n"

GAP (2)

- Non deterministic:

```
x = start
while x != end
        y = random(graph-size)
        if (x,y) not in graph
                REJECT NTM
        x = y
ACCEPT NTM
```

Here NTM splices into graph-size new NTMs

GAP (3)

```
bool deterministic_gap(graph g, node p, int len)
{
        if (len == 0) return false;
        for each neighbour q of p in g
                if  not already traversed q
                        if deterministic_gap(g, q, len-1)
                                return true;
        return false;
}
```

With n nodes in g: $O(n!)$

GAP (3)

```
bool deterministic_gap(graph g, node p, int len)
{
        if (len == 0) return false;
        parfor each neighbour q of p in g
                if  not already traversed q
                        if deterministic_gap(g, q, len-1)
                                return true;
        return false;
}
```

- how many cpus ?
- how much speedup ?

Graph Accessibility Problem GAP (3)

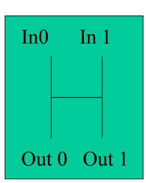
- PID = $0 \ldots n^3$ // get cpu-nr (need n^3 cpus !)
- $I=\lfloor PID/n^2 \rfloor$, $j=\lfloor (PID \% n^2)/n \rfloor$, $k=PID \% n$
- A[I,I] = true
- L=1
- while (L < n)
  - if (A[i,k] && A[k,j])
    - A[i,j] = true;
  - I = L; J = L;
  - L=L*2;

GAP (4)

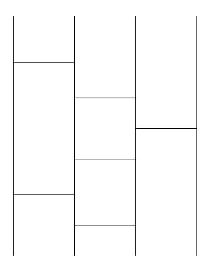
- Notes:
  - Run with n^3 processors delivers exponential speedup
  - Simulanious(!) writes to A[i,j]
  - Proof:
    - After the t'th iteration
      - $L = 2^t$
      - For all $0 <= x,y < n$, A[x,y]==true iff there is a path of length of at most L from x to y (by induction on 't')
      - After $\lceil \log n \rceil$ iterations, A[u,v] is true iff there a path from u to v
      - Therefore
        - » Running time is $O(\log n)$
        - » Word size needed is $O(\log n)$
        - » Space bound is $O(n^3)$

Combinatorial Networks (1)

- Comparator
  - Small switching unit that swaps outputs if (in 0 < in 1)
- Networks of comparators
  - Without feedback
  - Values are atomic units
  - Values travel in channels
  - Finite number of levels
    - Level = parallel comparators
    - Level 0 are the input values
- If outputs are de/ascending
  - Sorting network
- Important: depth and size (#comparators used)

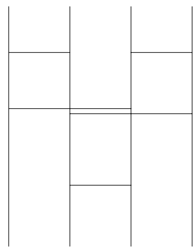


Combinatorial Networks (2)



*Question: size, depth ?

Combinatorial Networks (3)



*Question: size, depth ?

Min-max, max-min theorem (1)

- Every mixed min-max, max-min sorting network can be rewritten to pure min-max
- Proof
  - Represent comparators as <a,b,c> tuples where a = level and b,c output channels
    - Min-max=b<c, max-min=b>c
  - Represent sorter as list C: 0…s, with s comparators.

Min-max, max-min theorem (2)

// read c?=c? and b? as b?

```
for i:=0 to s
        if bi>ci then
                for j:=i to s
                        Lj=<aj,bj,cj>
                        if bj==b_i then bj=c_i in Lj
                        else if bj==ci then bj=bi in Lj

                        if cj==bi then cj=ci in Lj
                        else if cj=ci then cj=bi in Lj
```

Min-max, max-min theorem (3)

- Claim 1: after the (i-1)th iteration we still sort
  - bi < ci
    - No change made
  - bi > ci
    - Swap outputs whenever bi or ci is used ***below*** level I
    - Level i can ***only*** sort
- By induction we therefore still sort. ■
- Claim 2: we end up with only min-max: trivial induction on 'i' ■
- Claim 3: we still sort in the same way: given ascending inputs we perform no actions and therefore sort in the same way. ■

The zero-one principle (1)

- An n-input network is a sorter if it sorts all sequences of 0-1.
  - Impact: no need to test for $\mathbf{Z}$, 0/1s suffice
- Proof:
  - Define $h_a(x) = 1$ if x >= a, else 0
  - Claim: if for inputs $x_1, x_2 \dots x_n$ a channel carries value B,at level j, then for inputs $h_a(x_1)$, $h_a(x_2) \dots h_a(x_n)$, it carries $h_a(B)$

The zero-one principle (2)

- Claim: if for inputs x1,x2…xn a channel carries value B,at level j, then for inputs h_a(x1), h_a(x2)… h_a(xn) , it carries $h_a(B)$
  - J=0: h_a(x1) = $h_a(B)$
  - J>0: consider channel i on level j with value B on input x
    - Write V(i,j) = B for input x
    - Write $V_a(i,j)$ for input $h_a(x)$
- Suppose no comparator at level j, then by induction (as on the previous j we were OK)
  - $V_a(i,j) = V_a(B)$

The zero-one principle (3)

- Suppose exists comparator between channels i and k at level j
  - i<k (=min-max comparator)
    - $V_a(i) = \min(V_a(i,j-1), V_a(k,j-1))$
    - $V_a(i) = \min(h_a(B_i), h_a(B_k))$ // by induction
    - $V_a(i) = \min(B_i, B_k)$ // by def. Of h_a
    - $V_a(i) = h_a(B)$

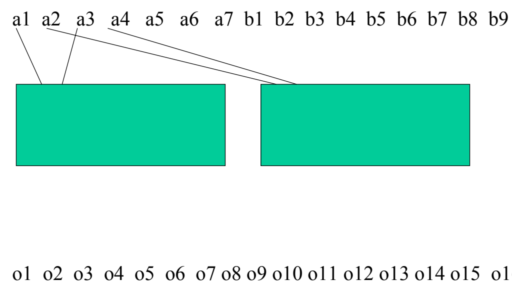
The zero-one principle (4)

- Proof by contradiction for network K:
  - Assume outputs y_{1-n} for inputs x_{1-n}
  - Assume not properly sorted: $y_i > y_{i+1}$
  - Look at $h_a(y_{1-n})$ for inputs $h_a(x_{1-n})$
    - Chooze $a=(y_i+y_i+1)/2$
      - $h_a(y_i) = 1$
      - $h_a(y_{i+1}) = 0$
    - Which can't happen by definition of h_a when applied to a sorting network therefore K not a sorting network. Therefore K must be a sorter.

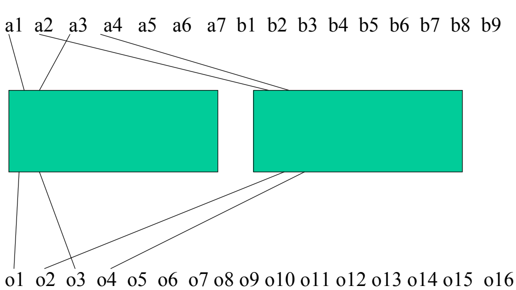
Batcher's merging network (1)

- Merge two sorted sequences <$a_1 \ldots a_n$> and <$b_1 \ldots b_n$>
  - N=1 then use 1 comparator
  - N>1
    - first merge(a_x,b_x) where x odd and merge(a_y,b_y) where y even
    - Merge two sub results by adding a layer of comparators connecting channel 2i and 2i+1(with 1<= i <n/2)

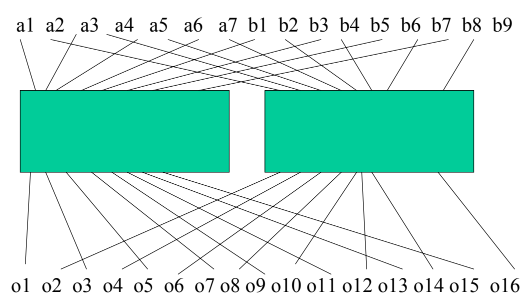
Batcher's merging network (2)



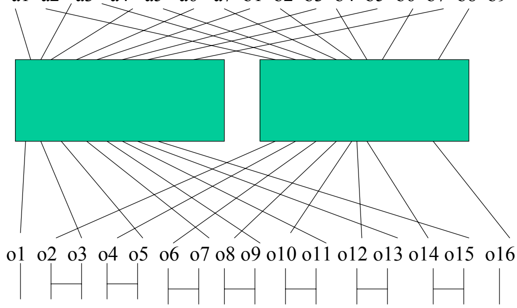
Batcher's merging network (3)



Batcher's merging network (4)



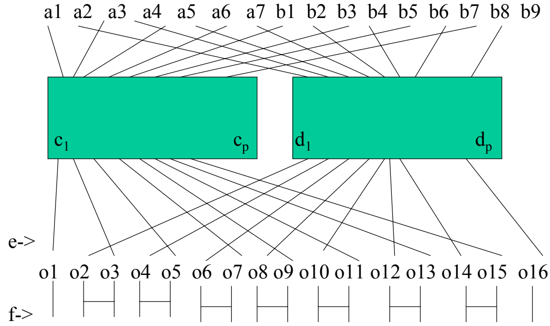
Batcher's merging network (5)



Batcher's merging network (6)

- Does this merge OK ?
- Proof:
  - Assume:
    - n inputs
    - recursive sub-merge was OK for odd/even
      - got <$c_1 \ldots c_p$> and <$d_1 \ldots d_p$> with p=n/2
    - Input-a <$a_1 \ldots a_n$> : g*'0', (p-g)*'1'
    - Input-b <$b_1 \ldots b_n$> : h*'0', (p-h)*'1',
    - Output <$f_1 \ldots f_n$>

Batcher's merging network (7)



Batcher's merging network (8)

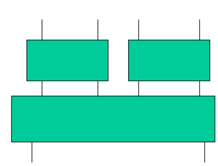
- By induction, 'e' then consists of ((g/2)+(h/2)) * '0' and (p-(g/2)-(h/2))*'1'
  - Same for 'd'
- 3 cases:
  - c has the same number of zeros as d
    - e= 000000010111111111 → g=h → must be even(g+h) → there must be a comparator to fix this
    - E=000000000000000000 → ok
    - E=111111111111111111 → OK
  - c has one more zero than d → e=sorted, therefore f too
  - c has two more zeros than d → e=sorted, therefore f too
- Now merges all 0/1 seq. therefore zero-one principle applies

Batcher's merging network (9)

- Given 2 sorted seq. of length p
  - depth(1) = 1, size(1) = 1
  - depth(p) = depth(p/2) + 1
  - size(p) = 2*size(p/2) + p - 1

Batcher's sorting network

- Steps:
  - Sort 1st half, sort 2nd half of input numbers
  - Recursively merge the results of those two

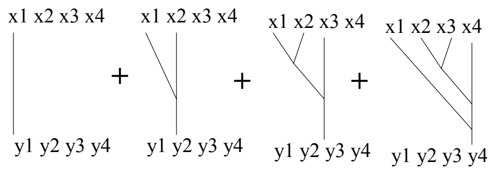


SDepth(n)=SDepth(n/2)+ merge_depth(n/2)
=SDepth(n/2) + log(n)
=log(n)*(log(n)+1)/2

Parallel Prefix using Fischer&Ladners algorithm (1)

- Given an addition gate
- Given n inputs $x_{1\text{-}n}$ and n outputs $y_{1\text{-}n}$
  - Compute y(i)= sum $x_{1..i}$ for i=1…n
- Naiive solution:

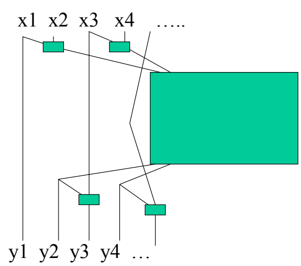
  (how many times is (x1+x2) computed ?)



Parallel Prefix using Fischer&Ladners algorithm (2)

- On input $x_{1\ldots n}$
  - first compute x_i+x_{i+1} for each odd(i)
  - Next compute the prefix sum of these results
    - Use sub network with n/2 inputs
      - i-th output of subnetwork becomes 2*i-th output
      - ((2*I)+1)-th output becomes the I-th output of the subnet + ((2*I)+1)-th input

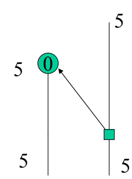
Parallel Prefix using Fischer&Ladners algorithm (3)



D(n)=D(n/2)+2
= 2*log(n)
S(n) = S(n/2)+n-1
= 2n-2-log(n)

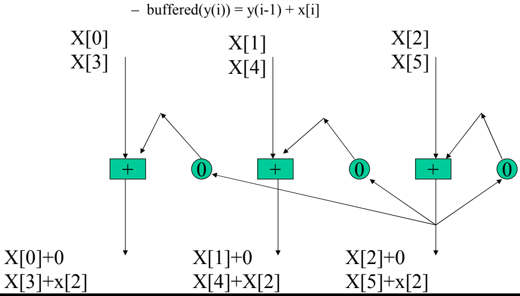
Networks using Feedback (1)

- Feedback is memory
  - Store previously computed values
    - Less recompute == less combinators ?
    - Less recompute == faster ?
  - Feedback is stored in 'buffer nodes
    - Buffer is initialized with some value (memory empty)
    - Releases same value each clocktick after 'set'



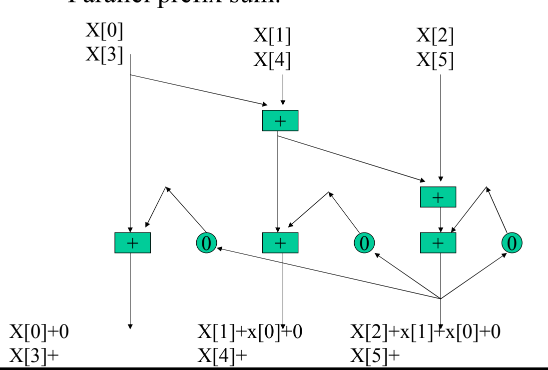
Networks using Feedback (2)

- Parallel perfix sum:
  - Compute y(i)= sum $x_{1..i}$ for i=1…n
    - Try and create a circuit that does:
      - y(i) = buffered(y(i)) + 0
      - buffered(y(i)) = y(i-1) + x[i]



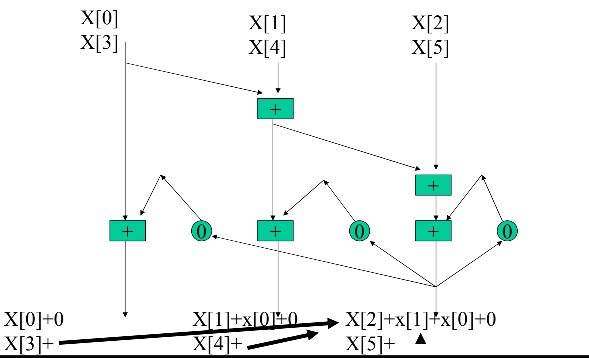
Networks using Feedback (3)

- Parallel prefix sum:



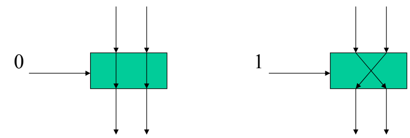
Networks using Feedback (4)

- Parallel perfix sum:



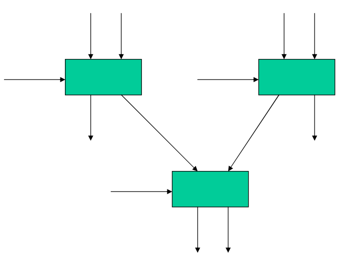
Permutator (1)

- Need to generate all permutations of N values
  - 1, 2, 3, 4
  - 4, 3, 2, 1
  - 3, 2, 1, 4
  - Etc
- N numbers = N! permutations
- Use a 'boolean switch' component

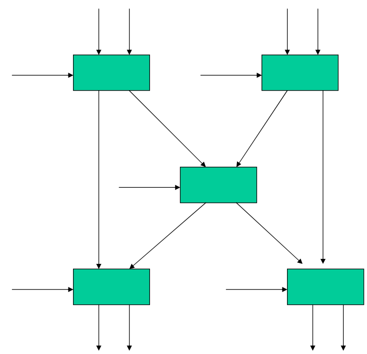


Permutator (2)

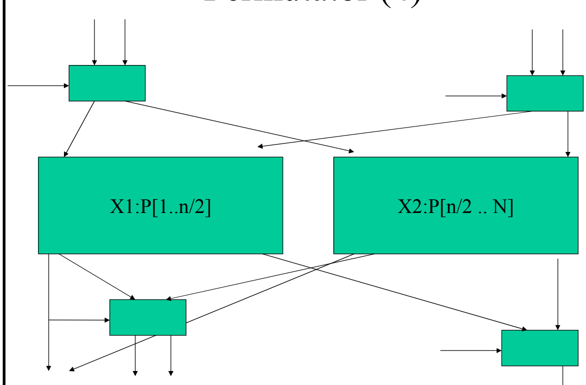
Is this a full permutator ?



Permutator (3)



Permutator (4)



Permutator (5)

- General:
  - Permutate all even inputs
  - Permutate all odd inputs
  - Permute one even with one odd output
- Does this create all permutations ?
  - Proof, given random wanted permutation:
    - N=1, trivially ok
    - N>1, route one output to one input via X1, take a neighbour-input and route via X2, continue until done
      - If routing conflict, start over with any other output
      - Though N=1 case, there is atleast one mapping.

Permutator (6)

- How many switches will I need to permute 5 values ?

See U in 2005 !

- No lecture on 20th or exercises on the 24th
- 10 january next lecture