Parallel Complexity Theory and Combinatorial Networks

Lecture 6
Sequential 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

- Trick: draw the iteration space, the surface area is the complexity formula
Sequential Complexity (2)

• NP hard problems
  – \( P = \text{Polynomial time} \)
    - \( \sum_{i=0..n} c_i \times x^i \)
  – NP=\text{Non-deterministic turing machine, 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 \((A \vee B) \land (\neg C \vee D) \land \text{etc} = \text{true}\) 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
Parallel Complexity

• $O(N,P)$
  – With input size $N$ and $P$ processors
  – $(N,1)$ for sequential algorithms
  – $(1,P*P)$
    • Problem can be solved in constant time with $P*P$ processors
  – $(N,\log(P))$
    • ?

• Trick: draw iteration space and collapse the axis that is run in parallel
Graph Accessibility Problem (GAP) (1)

- Graph \( G = (V, E) \)
  - Vertices numbered 0 – n-1
  - Shared memory machine is given:
    - \( 0 \leq u, v < n \)
    - 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
        - \( A[i,j] = \text{true} \) iff \((i,j) \in E\)
  - Problem: find a path over the edges that matches some condition. We’ll use “length(path) < n”
GAP (2)

- Non deterministic:

```plaintext
x = start; path = <x>;
while x != end
    y = random(graph-size)
    if (x,y) not in graph
        REJECT NTM
    path += y;
    if not condition(path)
        REJECT NTM
    x = y
// now loop again to see if 'path' is minimal to all others.
// ....
ACCEPT NTM
```

Here NTM splices into graph-size new NTMs
bool deterministic_gap(graph g, node p, path z) 
{
    if (p == end) return true;
    for each neighbor q of p in g
        if q not in z:
            m = z + q;
            if (condition(m) and
deterministic_gap(g, q, m)
                return true;
    return false;
}
With n nodes in g: O(n!)}
bool deterministic_gap(graph g, node p, path z)
{
    if (p == end) return true;
    parfor each neighbor q of p in g
        if not q in z:
            m = z + q;
            if condition(m) and
deterministic_gap(g, q, m)
                return true;
    return false;
}
- how many cpus ?
- how much speedup ?
Graph Accessibility Problem

GAP (3) – PRAM model

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

GAP (4)

• Notes:
  – Run with $n^3$ processors delivers exponential speedup
  – Simultaneous (!) writes to $A[i,j]$ 
  – Proof:
    • After the $t$'th iteration
      – $L = 2^t$
      – For all $0 \leq x, y < n$, $A[x,y] = \text{true}$ 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 $O(n^3)$
Comparator-ial Networks (1)

- Networks of connected comparators
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 (time before sth drops out)
  – size (#comparators used)
*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)

// input: array of comparators, depth 's'
for i=0 to s
    // if max-min ISO min-max
    if \( b_i > c_i \) then
        for j=i to s
            \( L_j = <a_j, b_j, c_j> \) // level \( a \), output \( b \), output \( c \)
            if \( b_j == b_i \) then \( b_j = c_i \) in \( L_j \)
            else if \( b_j == c_i \) then \( b_j = b_i \) in \( L_j \)
            if \( c_j == b_i \) then \( c_j = c_i \) in \( L_j \)
            else if \( c_j == c_i \) then \( c_j = b_i \) in \( L_j \)
Min-max, max-min theorem (3)

• Claim 1: after the (i-1)\textsuperscript{th} iteration we still sort
  - \( b_i < c_i \)
    • No change made
  - \( b_i > c_i \)
    • Swap outputs whenever \( b_i \) or \( c_i \) is used \textit{below} level I
    • Level i can \textit{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 \( \mathbb{Z} \), 0/1s suffice

• Proof:
  – Define \( h_a(x) = 1 \), if \( x \geq a \), else 0
The zero-one principle (2)

- Claim: if for inputs $x_1, x_2 \ldots x_n$ a channel carries value $B$, at level $j$, then for inputs $h_a(x_1), h_a(x_2)\ldots h_a(x_n)$, it carries $h_a(B)$
  - $J = 0$: $h_a(x_1) = h_a(B)$
  - $J > 0$: 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)
    - $V_a(i, j) = \min(V_a(i, j-1), V_a(k, j-1))$
    - $V_a(i, j) = \min(h_a(B_i), h_a(B_k))$  // by induction
    - $V_a(i, j) = \min(B_i, B_k)$  // by def. Of $h_a$
    - $V_a(i, j) = h_a(B)$
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 $h_a(x_{1-n})$
  - Choose $a = (y_i + y_{i+1})/2$
    - $h_a(y_i) = 1$
    - $h_a(y_{i+1}) = 0$
  - Which can’t happen by definition of $h_a$ when applied to a sorting network, therefore K not a sorting network.
    - = Contradiction
    - -> Assumption must be wrong
      » Therefore K must be a sorter.
Batcher’s merging network (1)

- Merge two *sorted* sequences \(<a_1...a_n>\) and \(<b_1...b_n>\)
  - \(N=1\) 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 \(1 \leq i < n/2\))
Batcher’s merging network (2)

a1 a2 a3 a4 a5 a6 a7 b1 b2 b3 b4 b5 b6 b7 b8 b9

o1 o2 o3 o4 o5 o6 o7 o8 o9 o10 o11 o12 o13 o14 o15 o16
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 \(<c_1...c_p>\) and \(<d_1...d_p>\) with \(p=n/2\)
    • Input-a \(<a_1...a_n>\) : \(g*'0', (p-g)*'1'\)
    • Input-b \(<b_1...b_n>\) : \(h*'0', (p-h)*'1'\),
    • Output \(<f_1...f_n>\)
Batcher’s merging network (7)
Batcher’s merging network (8)

• By induction, ‘e’ then consists of $((g/2)+(h/2)) \times ‘0’ \text{ and } (p-(g/2)-(h/2))\times’1’$
  – Same for ‘f’

• 3 cases:
  – c has the same number of zeros as d
    • e= 0000000101111111111 → g=h → must be even(g+h) → there must be a comparator to fix this
    • E=000000000000000000 → ok
    • E=11111111111111111 → 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 $p$
  – $\text{depth}(1) = 1$, $\text{size}(1) = 1$
  – $\text{depth}(p) = \text{depth}(p/2) + 1$
  – $\text{size}(p) = 2*\text{size}(p/2) + p - 1$
Batcher’s sorting network

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

\[
SDepth(n) = SDepth(n/2) + \text{merge\_depth}(n/2) \\
= SDepth(n/2) + \log(n) \\
= \log(n)*((\log(n)+1)/2)
\]
Parallel Prefix using Fischer & Ladners algorithm (1)

- Given an addition gate
- Given n inputs $x_{1-n}$ and n outputs $y_{1-n}$
  - Compute $y(i) = \sum x_{1..i}$ for $i=1…n$
- Naive solution:
  (how many times is $(x_1 + x_2)$ 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)

\[
\begin{align*}
D(n) &= D(n/2) + 2 \\
&= 2 \times \log(n) \\
S(n) &= S(n/2) + n - 1 \\
&= 2n - 2 - \log(n)
\end{align*}
\]
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 clock-tick after ‘set’
Networks using Feedback (2)

• Parallel prefix sum:
  – Compute \( y(i) = \sum_{1..i} x \) for \( i=1\ldots n \)
  • Try and create a circuit that does:
    – \( y(i) = \text{buffered}(y(i)) + 0 \)
    – \( \text{buffered}(y(i)) = y(i-1) + x[i] \)
Networks using Feedback (3)

- Parallel prefix sum:

\[
\begin{align*}
X[0] & + 0 \\
X[3] & + \\
X[1] & + \\
X[4] & + \\
X[2] & + \\
X[5] & + \\
\end{align*}
\]
Networks using Feedback (4)

• Parallel prefix sum:

\[
\begin{align*}
X[0] &+ 0 \\
X[3] &+ X[0] + 0 \\
X[4] &+ X[1] + x[0] + 0 \\
\end{align*}
\]
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)

X1: P[1..n/2]

X2: P[n/2 .. N]
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 at least one mapping.
Permutator (6)

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