ALPA LECTURE 3: PRAM MODEL

Outline

- More pointer jumping (Euler Tour)
- Divide and Parallelize
- Work-Depth Paradigm and Brent's Theorem
- Relative Power of PRAM models

More Pointer Jumping: Euler Tour

Similar data structures, but more complicated than lists

Problem: Given a binary tree with n nodes: each node i has three fields, *parent*, *right* & *left*. Determine the depth of all nodes in the tree.

Naive solution: Work from root down to leaves (increment a counter as you go along)

Drawback: What if tree is not balanced?

Solution: Use an Euler Tour

Basic Background

- Definition: Euler tour = cycle that traverses each edge exactly once (nodes may be visited multiple times)

- Remark: A connected directed graph admits an ET iff in-degree = out-degree.

- Hence the directed version of an undirected, connected graph has an ET.

Use ET for solving depth problem

For each node, associate three processors, A, B and C, and set up a linked list as follows:

- Node's A processor points to A processor of left child, if it exists, otherwise to own B processor.
- B processor points to A processor of right child if it exists, otherwise to own C processor.
- If a node is a left child of its parent, C processor points to B processor of parent, otherwise to C processor of parent.
- Root's C processor set to nil.

Question

What values should be placed in A, B and C processors, so that parallel prefix of the linked list gives depth of node?

- A processors get 1
- B processors get 0
- C processors get -1

Divide & Parallelize

Scan on an array

Input: Vector $x[1, \ldots n]$, (for $n = 2^k$) of elements of type T, binary associative operator, $\oplus : T \times T \rightarrow T$

Output: Vector $s[1, \ldots n]$ of type T, where $S[i] = \oplus_{j=1}^{i} x[j]$

Solution: ($\oplus$ is op)

```
if n = 1 then s[1] := x[1]
return s
endif
forall i = 1 ... n/2
    do y[i] := x[2i-1] op x[2i] enddo
z[1, ... n/2] := Scan(y[1 ... n/2])
forall i = 1 ... n do
    if even i then s[i] := z[i/2]
    elseif i= 1 then s[1] := x[1]
    else s[i] := z[(i-1)/2] op x[i]
    endif
enddo
return s
```

Work-Time Paradigm (recap)

- Algorithm/Program = sequence of steps
- Step = parallel operations $\Rightarrow$ forall construct (on as many processors as needed)
- Tackle actual number of physical processors later

Two complexity measures

- Step complexity, $S(n)$
- Work complexity, $W(n)$, total number of operations

$$W(n) = \sum_{i=1}^{S(n)} W_i(n)$$

Analysis of array scan

- $S(n) = \lg(n)$
- $W(n) = \Theta(n)$

Algorithm is not work-optimal.

Brent's Theorem

Theorem: [Brent 74] A WT algorithm with step complexity $S(n)$ and work complexity $W(n)$ can be simulated on a p-processor PRAM in no more than $\left\lfloor \frac{W(n)}{p} \right\rfloor + S(n)$ steps

Proof For each step i (for $1 \leq i \leq S(n)$, let $W_i(n)$ be the number of operations. Simulate each step on p processors in $\left\lceil \frac{W_i(n)}{p} \right\rceil$ time (load balanced). Hence total time is:

$$T = \sum_{i=1}^{S(n)} \left\lceil \frac{W_i(n)}{p} \right\rceil \leq \sum_{i=1}^{S(n)} \left(\left\lfloor \frac{W_i(n)}{p} \right\rfloor + 1 \right) \leq \left\lfloor \frac{W(n)}{p} \right\rfloor + S(n)$$

Implications

Efficiency improvement (by load balancing)

- Using a run time system/scheduler (à la Cilk [MIT], Athapascan [IMAG]
- At compile time (à la automatic parallelization)
- At algorithm design time (gives limits of parallelization – scalability)

Return to scan

Efficiency improvement (by load balancing)

$$S(n) = \lg(n) \qquad W(n) = \Theta(n)$$

Question: How many processors can we have without sacrificing running time?

Answer: As long as $\left\lfloor \frac{W(n)}{p} \right\rfloor = \Theta(S(n))$

We can retain $O(\lg n)$ running time, but on only $\frac{n}{\lg n}$ processors.

How? Careful scheduling at algorithm design time

Return to scan

Split array into blocks of $\lg n$ elements, an put one block per processor (so $p = \frac{n}{\lg n}$).

1. Each processor locally scans its block sequentially
2. The processors use the previous parallel (naive) algorithm on the last element of their local result, getting the last element(s) in the final result.
3. Each processor uses its local result to update the remainder of its result (again sequentially)

Analysis

1. $\lg n$
2. $\lg\left(\frac{n}{\lg n}\right) = \lg n - \lg^2 n = O(\lg n)$
3. $\lg n$

Scalability "analysis"

Recap: deinition If $W(n) = T(n,1)$ is the same as best sequential algorithm $T_S(n)$ then algorithm is work optimal

$$T(n,p) = O\left(\frac{T_S(n)}{p} + S(n)\right) = O\left(\frac{W(n)}{p} + S(n)\right)$$

Speedup $\mathcal{S} = \dfrac{T_S(n)}{T(n,p)} = \Omega\left(\dfrac{W(n)}{\frac{W(n)}{p} + S(n)}\right) = \Omega\left(\dfrac{pW(n)}{W(n) + pS(n)}\right)$

$\mathcal{S}$ is $\Theta(p)$ if $p = O(\frac{W(n)}{S(n)})$. Common sense (corollary of Brent's theorem) Of two work efficient parallel algorithm's, the one with the smaller step complexity is more scalable

Comparison of PRAM sub-models: ER vs CR

We know ER $\subseteq$ CR. Is the inclusion strict, i.e., is ER $\subset$ CR?

Yes

We know EW $\subseteq$ CW. Is the inclusion strict?

Yes