Colorado State University

CS 575 Parallel Processing
Week 3: PRAM

Sanjay Rajopadhye
Colorado State University

Outline

- Discussion 1 recap
- PRAM Algorithms
  - Basic model & variants
  - Basic algorithmic techniques
  - List/pointer based algorithm (from the readings)
  - Array based algorithms

Discussion 1

- Main task: adding *n* numbers of 8 processors
  - Machine models: distributed/shared memory
- Distributed memory: initial distribution:
  - If one node has all the data it is faster to just add up everything on this processor so assume data is distributed to processors
  - Locally add *n*/8 numbers (*n* T_{comp}/8 time)
  - … then the 8 processors add up 8 numbers

Discussion 1 (contd)

- Reduction of 8 numbers on 8 processors (topology dependent)
  - Ring: $4\left(T_{comp}+T_{mess}\right)$
  - Grid: $4\left(T_{comp}+T_{mess}\right)$
  - Fully connected: $3\left(T_{comp}+T_{mess}\right)$
  - Shared memory: $3\left(T_{comp}+T_{mess}\right)$
- Machine parameters
  - T_{comp} = 1cycle
  - T_{share} = 100 cycles
  - T_{mess} = 10^6 cycles: not (yet) affine model of Ch 2

PRAM Algorithms

- Simplest (ideal) model of parallel machines
- Develop a theory of solving problems on such machines
- Two parameter analysis
  - Problem size *N*
  - Number of processors *P*
- Why do this analysis
  - Training
  - Lower bounds

Machine model

- A machine with *P* processors
- Accessing a shared memory of (arbitrary size)
- Ideal memory (one unit time to access any word in memory)
  - Ignore issues of locality (orthogonal to PRAMs)
- Conflicts: what if two processors access the same location
  - answer to this question yields sub-models

Exclusive/Concurrent access

- Exclusive access:
  - Simultaneous access to the same memory location are forbidden
    - Machine does not allow it
    - Algorithms must respect it (we need to prove it)

Most "restrictive" PRAM machine = EREW

- Concurrent accesses are allowed
  - For reads
  - What about concurrent writes

Concurrent Write Arbitration

How to resolve concurrent writes

- Common: insist that whenever this happens all processors doing so must be writing the same value
- Arbitrary: one of the writers succeeds
  - don't know which one, and algorithm must work regardless of who wins
- Priority: e.g., lowest index processor wins
- Combining: values are combined using an op (usually associative/commutative – reduction)

List Ranking (pointer jumping)

```
1  for each processor i  in parallel
2     do if next [i] = NIL
3        then d[i]  ← 0;
4        else d[i] ← 1;
5  while there exists i such that next[i] != NIL
6     for each processor i in parallel
7        do if next[i] != NIL
8           then   d[i] ← d[i] + d[next[i]];
9                  next[i] ← next[next[i]];
```


Specification & Correctness

- What we desire to compute: at the end we must ensure

$$d[i] = \begin{cases} 0 & \text{if } next[i] = \text{NIL} \\ d[next[i]] + 1 & \text{if } next[i] ! = \text{NIL} \end{cases}$$

- Invariant for the while loop (lines 6-9): adding up the d[i] values in any sub list headed by i, yields the rank of the object i in the original list

Analysis (Time & Work)

- Execution time of ListRank is O(lg *N*)
- Work of the parallel algorithm is O(*N* lg *N*)
- Work Efficiency
  - Ratio of work of best sequential algorithm to work of parallel algorithm
  - 1/lg *N* asymptotically approaches zero!

Parallel prefix on lists

```
1  for each processor i  in parallel
2      do if y[i] = x[i];
3  while there exists i such that next[i] != NIL
4      for each processor i in parallel
5          do if next[i] != NIL
6              then  y[next[i]] ← y[i] + y[next[i]];
7                    next[i] ← next[next[i]];
```

Parallel Prefix on Arrays

- Instead of lists you have an array X[1…N] and want to produce an Array Y[1…N] (so Quiz 2 was a prefix-max)
- Use divide & conquer: two strategies
  - Split in middle
  - Zip/unzip
- Analysis questions:
  - How many operations (work)
  - How many steps (time)

Kogge Stone 1974

- Compute the elements of the recurrence

$$y(i) = a_3 y(i-3) + a_2 y(i-2) + a_1 y(i-1) + a_0$$

- Define: $Y_i = [y_i, y_{i-1}, \ldots y_{i-3}]^T$
- papers (PPoPP 2013 is most recent)
- Another application: IIR filter/linear recurrences

Depth finding (not Euler tour)

Problem: find depth of all nodes in binary tree

- Sequential algorithm: $\Theta(N)$
- Parallel algorithm:
  - my depth = 1 + parent's depth (// depth first traversal)
  - Steps? Time? Worst case?
- But we have a hammer!!

Brent's Theorem

(real) Problem: dag scheduling, under the guise of simulating a combinational circuit

- Bounded fan-in and CREW
- Basic idea is simple (see figure)

Theorem: a dag with depth = d size= s can be simulated by a CREW PRAM in time = $O(n/p + d)$

- Importance: idea can be used to get work-efficient algorithms for many other problems.

Fast min: power of CW

```
1   for each processor i  in parallel do M[i] = 0;
2   for each processor <i,j>  in parallel do
3      if (C[i]<C[j]) then M[j] = 1;
4   for each processor <i,j> in parallel do
5      if (M[i] == 0 && i<j) then M[j] = 1;
6   for each processor i  in parallel do
7      if (M[i] == 0) then answer = C[i];
```