Advanced Algorithms

Piyush Kumar

(Lecture 10: Parallel Algorithms)

Courtesy Baker 05.

Parallel Models

- An abstract description of a real world parallel machine.
- Attempts to capture essential features (and suppress details?)
- What other models have we seen so far?

RAM?
External Memory Model?

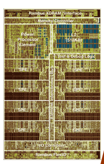
RAM

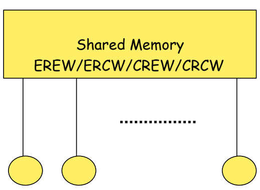
- Random Access Machine Model
  - Memory is a sequence of bits/words.
  - Each memory access takes O(1) time.
  - Basic operations take O(1) time: Add/Mul/Xor/Sub/AND/not…
  - Instructions can not be modified.
  - No consideration of memory hierarchies.
  - Has been very successful in modelling real world machines.

Parallel RAM aka PRAM

- Generalization of RAM
- P processors with their own programs (and unique id)
- MIMD processors : At each point in time the processors might be executing different instructions on different data.
- Shared Memory
- Instructions are synchronized among the processors

PRAM





EREW: A program isnt allowed to access the same memory location at the same time.

Variants of CRCW

- Common CRCW: CW iff processors write same value.
- Arbitrary CRCW
- Priority CRCW
- Combining CRCW

Why PRAM?

- Lot of literature available on algorithms for PRAM.
- One of the most "clean" models.
- Focuses on what communication is needed (and ignores the cost/means to do it)

PRAM Algorithm design.

- Problem 1: Produce the sum of an array of n numbers.
- RAM = ?
- PRAM = ?

Problem 2: Prefix Computation

Let X = $\{s_0, s_1, \ldots, s_{n-1}\}$ be in a set S

Let $\otimes$ be a *binary*, *associative*, *closed* operator with respect to S (usually $\Theta(1)$ time – MIN, MAX, AND, +, ...)

The result of $s_0 \otimes s_1 \otimes \ldots \otimes s_k$ is called the *k-th prefix*

Computing all such *n* prefixes is the *parallel prefix computation*

1st prefix	s_0
2nd prefix	$s_0 \otimes s_1$
3rd prefix	$s_0 \otimes s_1 \otimes s_2$
...	...
(*n*-1)th prefix	$s_0 \otimes s_1 \otimes \ldots \otimes s_{n-1}$

Prefix computation

- Suffix computation is a similar problem.
- Assumes Binary op takes O(1)
- In RAM = ?

Prefix Computation (Akl)

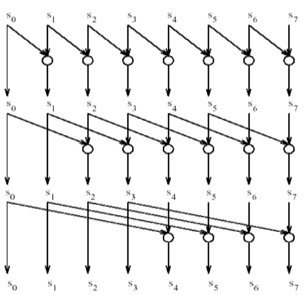
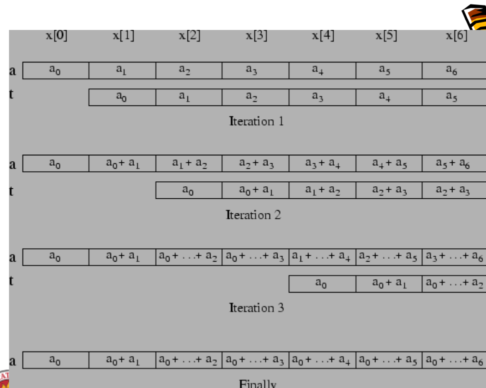


Figure 4.1: Prefix computation on the PRAM.



EREW PRAM Prefix computation

- Assume PRAM has n processors and n is a power of 2.
- Input: s_i for i = 0,1, ... , n-1.
- Algorithm Steps:

```
for j = 0  to (lg n) -1, do
    for  i = 2^j to  n-1 do
        h = i - 2^j
        s_i = s_h ⊗ s_i
    endfor
endfor
```

Total time in EREW PRAM?

Problem 3: Array packing

- Assume that we have
  - an array of n elements, $X = \{x_1, x_2, \ldots , x_n\}$
  - Some array elements are *marked* (or *distinguished*).
- The requirements of this problem are to
  - pack the marked elements in the front part of the array.
  - place the remaining elements in the back of the array.
- While not a requirement, it is also desirable to
  - maintain the original order between the marked elements
  - maintain the original order between the unmarked elements

In RAM?

- How would you do this?
- Inplace?
- Running time?
- Any ideas on how to do this in PRAM?

EREW PRAM Algorithm

1. Set s_i in P_i to 1 if x_i is marked and set $s_i = 0$ otherwise.
2. Perform a prefix sum on $S = (s_1, s_2, \ldots, s_n)$ to obtain destination $d_i = s_i$ for each marked x_i .
3. All PEs set $m = s_n$, the total nr of marked elements.
4. P_i sets s_i to 0 if x_i is marked and otherwise sets $s_i = 1$.
5. Perform a prefix sum on S and set $d_i = s_i + m$ for each unmarked x_i .
6. Each P_i copies array element x_i into address d_i in X.

Array Packing

- Assume n processors are used above.
- Optimal prefix sums requires O(lg n) time.
- The *EREW broadcast* of s_n needed in Step 3 takes O(lg n) time using a binary tree in memory
- All and other steps require constant time.
- Runs in O(lg n) time and is cost optimal.
- Maintains original order in unmarked group as well

Notes:

- Algorithm illustrates usefulness of Prefix Sums
- There many applications for Array Packing algorithm

Problem 4: PRAM MergeSort

- RAM Merge Sort Recursion?
- PRAM Merge Sort recursion?
- Can we speed up the merging?
  - Merging n elements with n processors can be done in O(log n) time.
  - Assume all elements are distinct
  - Rank(a, A) = number of elements in A smaller than a. For example rank(8, {1,3,5,7,9}) = 4

PRAM Merging

```
for each  i  do in parallel
    compute rank(a_i, B) and  rank(b_i,A)
  for each i do in parallel
    begin
    sorted[i+rank(a_i , B)] := a_i
     sorted[i+rank( b_i , A)] := b_i
   end.
```

A = 2,3,10,15,16

Rank(2)=1 +1
Rank(3)=1 +2
Rank(10)=2 +3
Rank(15)=4 +4
Rank(16)=4 +5

B = 1,8,12,14,19

Rank(1)=0 +1
Rank(8)=2 +2
Rank(12)=3 +3
Rank(14)=3 +4
Rank(19)=5 +5

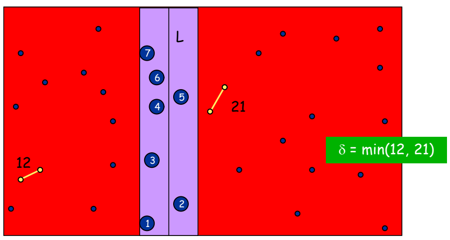
1	2	3	8	10	12	14	15	16	19

PRAM Merge Sort

- $T(n) = T(n/2) + O(\log n)$
- Using the idea of pipelined d&c PRAM Mergesort can be done in $O(\log n)$.
- D&C is one of the most powerful techniques to solve problems in parallel.

Problem 5: Closest Pair

- RAM Version ?



Closest Pair: RAM Version

```
Closest-Pair(p1, …, pn) {
   Compute separation line L such that half the points        O(n log n)
   are on one side and half on the other side.

   δ1 = Closest-Pair(left half)                                2T(n / 2)
   δ2 = Closest-Pair(right half)
   δ  = min(δ1, δ2)

   Delete all points further than δ from separation line L    O(n)

   Sort remaining points by y-coordinate.                      O(n log n)

   Scan points in y-order and compare distance between         O(n)
   each point and next 11 neighbors. If any of these
   distances is less than δ, update δ.

   return δ.
}
```

Closest Pair: PRAM Version?

```
Closest-Pair(p1, …, pn) {
   Compute separation line L such that half the points         O(1)
   are on one side and half on the other side.   Use sorted lists

   δ1 = Closest-Pair(left half)    } In parallel               T(n / 2)
   δ2 = Closest-Pair(right half)   }
   δ  = min(δ1, δ2)

   Delete all points further than δ from separation line L

   Sort remaining points by y-coordinate.                      O(log n)
                          (Use presorting and prefix computation.)

   Scan points in y-order and compare distance between         O(1)
   each point and next 11 neighbors.
   Find min of all these distances, update δ.                  O(log n)
                          (Again use prefix computation.)
   return δ.
}
```

Recurrence : $T(n) = T(n/2) + O(\log n)$

Other Interesting Algorithms

A List

- Approximation Algorithms
- Online Algorithms
- Learning Algorithms
- Network Algorithms
- Advanced Data Structures.
- Flow Algorithms.
- Algorithmic Game Theory
- Quantum Algorithms.
- Geometric Algorithms

Interesting Classes at FSU

In case you liked this class:

- Parallel Algorithms
- Computational Geometry
- Advanced Algorithms

Next Class

- Practice Problem Solving for Finals.
- Extra Office Hours :
  - Wednesday, I will be in office and accessible anytime for questions.