COMS 4995-2: Advanced Algorithms (Spring'20) Apr 21, 2020

Lecture Lecture 23: Parallel models, MPC model

Instructor: *Alex Andoni* Scribes: *Wang Yao, Zhengping Jiang*

1 Introduction

Today's lecture is about

- PRAM
- MPC

2 Parallel Randomized Access Machine (PRAM)

2.1 PRAM:

- P processors
- Shared memory
- Synchronized

Now a natural question here is about confidence. Is it allowed for two processors to read the same location at the same time? Is it allowed for two processors to write the same location at the same time?

This leads to 4 different sub-type architectures: **E/C-R-E/C-W**, where E stands for exclusive and C stands for concurrent. Now the question is about the performances of these 4 architectures.

2.2 Performance Measure:

- Parallel time ($\approx$ Depth of circuit)
- Work ($\approx$ Size of circuit)

Example 1. *Computing XOR for n bits*

On EREW we need $O(\log n)$ time
Lower bound: Even on CRCW we need $\Omega(\frac{\log n}{\log\log n})$ time.

3 Massive Parallel Computing (MPC) $\approx$ Map Reduce System

3.1 MPC

- M machines

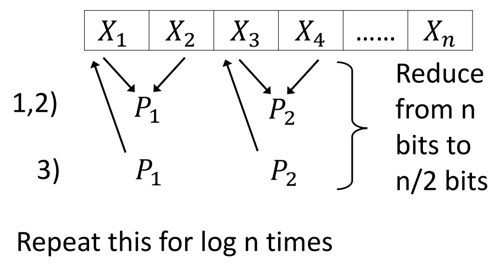

Figure 1:

- Each machine has space $O(S)$
  (Local space, the biggest difference between PRAM and MPC is the space is not shared.)
  $\rightarrow$ Total space $O(mS)$
- Computation proceeds in rounds
  - Do any local computation
  - Shuffle step, which means change information between machines
    (Shuffle has constraint that the amount of data in/out for each machine is $O(S)$)

3.2 Performance Measure:

- Number of rounds ($\approx$ Parallel time)
- Total space used ($\approx$ Work)

Related to BSP: Bulk-Synchronous-Parallel (Valiant)

Note: Can simulate "Shared Memory" by: Each machine be responsible for S addresses

Theorem 2. *Can simulate PRAM (at least EREW) where number of rounds $R = O(Parallel - time)$ (for any $S = \Omega(1)$)*

Proof. Why easy for EREW: each machine is responsible for S addresses, this means the number of requests is $\frac{O(S)}{round}$ □

Params:

- $S = O(1) \rightarrow$ Like PRAM (EREW)
- S can be much larger, total space $O(m \cdot S) \geq N$
- Ideally $m \cdot S = O(N)$
- $S = N^\delta$, $\delta =$ Small constant
  If each machine has a list of all other machines $\rightarrow S \geq m \rightarrow S \geq \sqrt{N}$, i.e. $\delta = \frac{1}{2}$

Question: When $\delta = \Omega(1)$ can we do better than PRAM ?

Problem 1: XOR of N bits

Note: At start each machine has S bits of inputs, split arbitrarily

PRAM: $O(\log N)$ Parallel time

$R = O(1)$ rounds, when $S = \sqrt{N}$

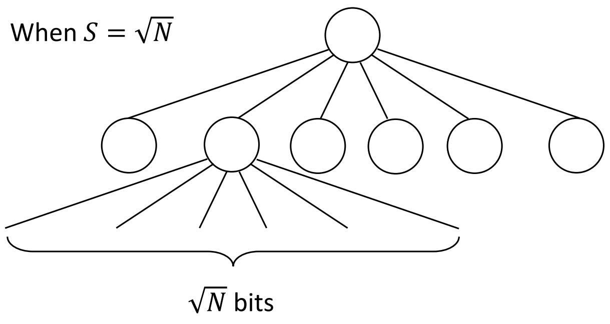

Figure 2:

Algorithm:[see Figure 2]

- XOR locally

- Compute answer to C

- C XORs its inputs

$R = O(\log_S N) = O(\frac{1}{\delta})$, When $S = N^{\delta}$, implement that in S-Arrays tree where number of rounds is the height of tree $= O(\log_S N)$

Theorem 3. *Can simulate CRCW with number of rounds $R = O(\textit{Parallel-time} \cdot \log_S N)$*

Problem 2: Prefix sum

Input: $(i, a_i), i = 1, 2, \cdots, n$
Output: $(i, \sigma_i), \sigma_i = \sum_{j=i}^{i}$

We'll solve in rounds $R = O(\log_S N)$

$M_l(i) =$ Ancestor of i at level $l, l = 1, 2, \cdots, n$

Algorithm:[see Figure 3]

- Send (i, a_i) to machine $M_L(i)$
- For $l = L \nearrow 2$ send sum of "local" numbers to parent
- For $l = 1 \searrow L-1$ send to child c: the sum of children $1, 2, \cdots, c-1$ plus σ_p where σ_p is the sum of a_is to the left of the node.[see Figure 4.]
- Machine $M_L(i)$ computes $\sigma_i = \sigma_p + a_j + a_{j+1} + \cdots + a_i$ and produces pair (i, a_i) [see Figure 5.]

Number of rounds $R = O(L) = O(\log_S N) = O(\frac{1}{\sigma})$

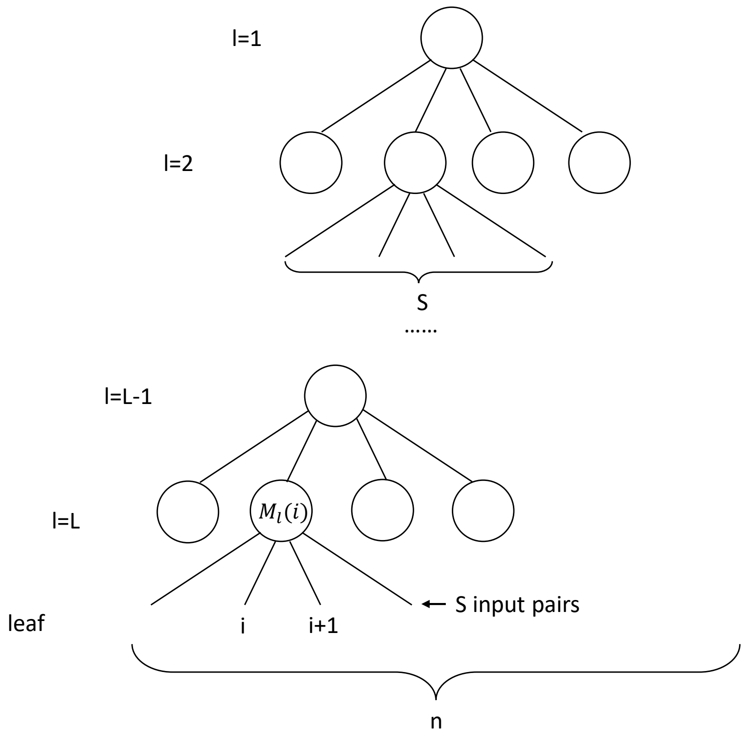

Figure 3:

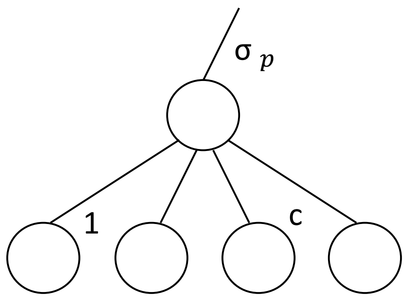

Figure 4:

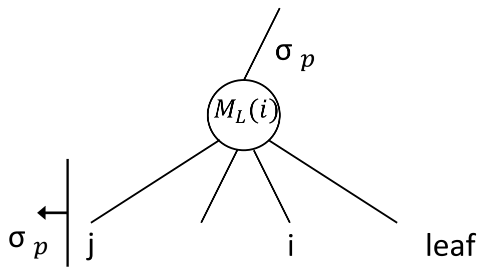

Figure 5:

Problem 3: Sorting

Input: $(i, a_i), i = 1, 2, \cdots, n$
Output: $(i, a_i, r), i = 1, 2, \cdots, n, r =$ rank of a_i in $\{a_j\}_{j=1,2,\cdots,n}$

$S = O(n^{\frac{3}{4}})$, $\delta = \frac{3}{4}$, $m = O(n^{\frac{1}{4}})$, we can also do like $S \geq N^{\delta}$

Idea: Quick-sort with many pivots $\rightarrow n^{\frac{1}{4}}$

1. For each $i \in [n]$ call i a pre-pivot with probability $\frac{n^{1/3}}{n}$
   $\rightarrow$ number of pre-pivots is $O(n^{1/3})$
2. Send pre-pivots to every machine
3. For every machine and every pre-pivots p, compute "local rank" $\rightarrow$ number of elements $\leq p$
4. Every machine sends "local rank" of every pre-pivot to C
5. C computes rank of each pre-pivot
6. C chooses set of $n^{1/4}$ pre-pivots, called $\mathcal{P}$, with ranks $i \cdot n^{3/4} \pm O(n^{3/4})$, $i = 1, 2, \cdots, n^{1/4}$
   $\rightarrow$ For averaging purpose
7. C sends pivots $\mathcal{P}$, together with their ranks to all machines
8. Each machine for each (i, a_i) finds j s.t. $p_j \leq a_i < p_{j+1}$ ($p_0 = -\infty$, $p_{n^{1/4}+1} = +\infty$), send a_i to machine j
9. Machine j has all a_is s.t. $p_j \leq a_i < p_{j+1}$, sort locally, assign ranks to a_i using rank of p_j