# PRAM Algorithms

Sathish Vadhiyar

# PRAM Model - Introduction

- Parallel Random Access Machine
- Helps to write precursor parallel algorithm without any architecture constraints
- Allows parallel-algorithm designers to treat processing power as unlimited
- Ignores complexity of inter-process communication

# Benefits of PRAM

- Can be a suitable basis for the design of a parallel program targeted to a real machine
- Base algorithm can help establish tight lower and upper complexity bounds for practical implementations
- Assumptions made in PRAM model for ideal parallelism can help architecture designers to develop innovative designs
- Can be suitable for modern day architectures, e.g., GPUs

# PRAM Architecture Model

- Consists of control unit, global memory, and an unbounded set of processors, each with own private memory
- An active processor reads from global memory, performs computation, writes to global memory
- Execute in SIMD model

# Different Models

- Various PRAM models differ in how they handle read or write conflicts

1. EREW – Exclusive Read Exclusive Write
2. CREW – Concurrent Read Exclusive Write
3. CRCW
    1. COMMON – All processors writing to same global memory must write the same value
    2. ARBITRARY – one of the competing processor's value is arbitrarily chosen
    3. PRIORITY – processor with the lowest index writes its value

# Mapping Between Models

- Any PRAM model/algorithm can execute any other PRAM model/algorithm
- For example, possible to convert PRIORITY PRAM to EREW PRAM

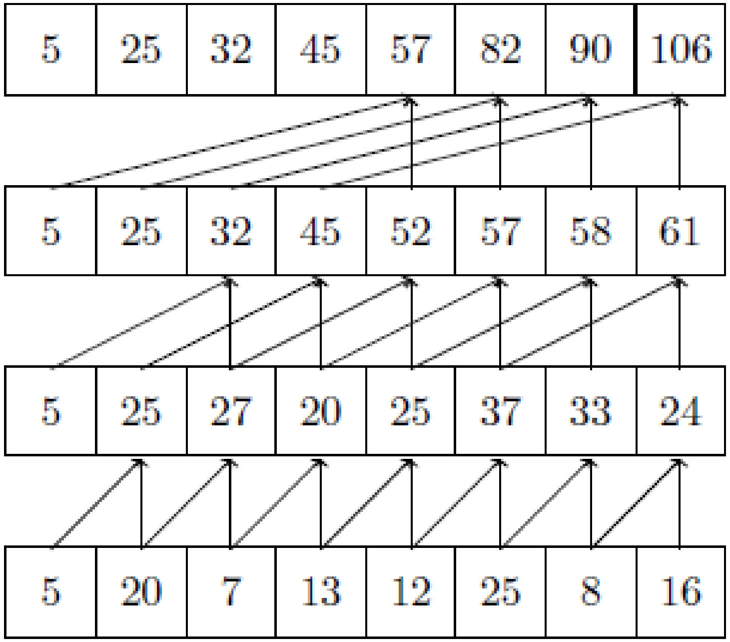
# Steps in PRAM Algorithm & Example: Reduction

- PRAM algorithms have two phases:
- Phase 1: Sufficient number of processors are activated
- Phase 2: Activated processors perform the computations in parallel
- For example, binary tree reduction can be implemented using n/2 processors
- EREW PRAM suffices for reduction

# Example: Prefix Sum Calculations

- $$B[i] = \sum_{j=0}^{i} A[j]$$
- Can be used for separating an array into two categories, lock-free synchronization in shared memory architectures etc.
- CREW PRAM algorithm for prefix sum calculations.
- Can use n/2 processors. Takes O(logn) time

# CREW PRAM for Prefix Sum




- Distance between the elements that are summed are doubled in every iteration

# Example: Merging Two Sorted Lists

- Most PRAM algorithms achieve low time complexity by performing more operations than an optimal RAM algorithm
- For example, a RAM algorithm requires at most n-1 comparisons to merge two sorted lists of n/2 elements. Time complexity is O(n)
- CREW PRAM algorithm:
- Assign each list element its own processor – n processors

# Example: Merging Two Sorted Lists

- The processor knows the index of the element in its own list
- Finds the index in the other list using binary search
- Adds the two indices to obtain the final position
- The total number of operations had increased to O(nlogn)
- Not **cost-optimal**

# Example: Enumeration sort

- Computes the final position of each element by comparing it with the other elements and counting the number of elements having smaller value
- A special CRCW PRAM can perform the sort in O(1) time
- Spawn $n^2$ processors corresponding to $n^2$ comparisons
- Special CRCW PRAM – If multiple processors simultaneously write values to a single memory location, the sum of the values is assigned to that location

# Example: Enumeration sort

- So, each processor compares a[i] and a[j]. If a[i] > a[j], writes position[i] = 1, else writes position[i]=0
- So the summation of all positions will give the final position – constant time algorithm
- But not cost-optimal – takes $O(n^2)$ comparisons, but a sequential algorithm does $O(n\log n)$ comparisons

# Summary

- PRAM algorithms mostly theoretical
- But can be used as a basis for developing efficient parallel algorithm for practical machines
- Can also motivate building specialized machines