Parallel Algorithms

Last time ...

- Introduction to Big Data
- Data Collection
- Assignment #1

Questions?

Today …

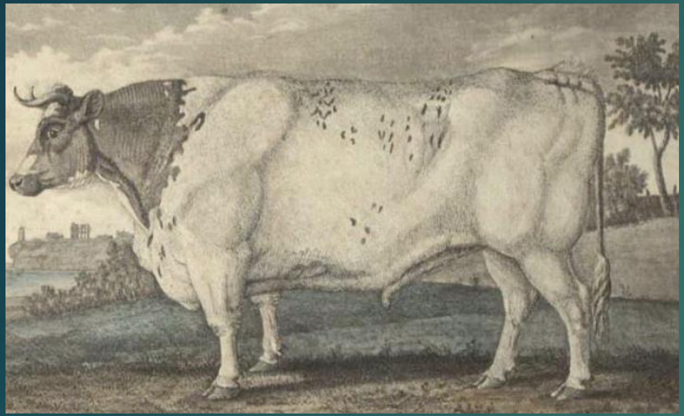
- New classroom – WEB L114
- Reminder: Assignment #1 questionnaire due today
- Permission codes

- Intro to Parallel Algorithms
  - Complexity analysis
  - Work depth
  - Communication costs

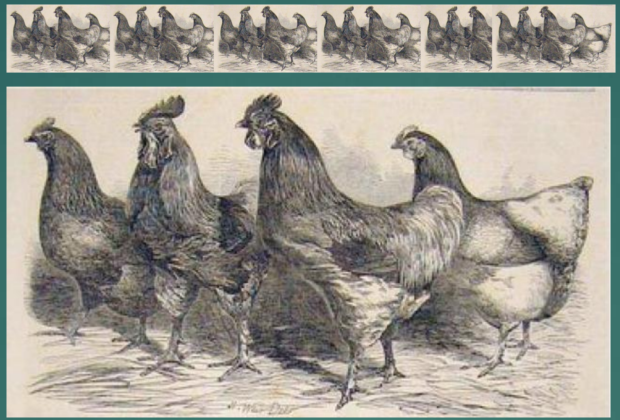
Parallel Thinking

THE MOST IMPORTANT GOAL OF TODAY'S LECTURE

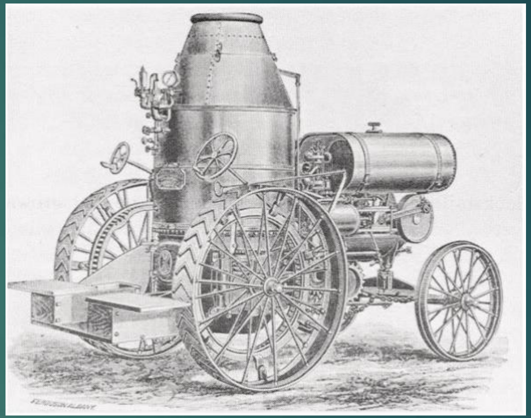
Parallelism & beyond …



1 ox: single core performance



1024 chickens: parallelism



tractor: better algorithms

If you were plowing a field, which would you rather use? Two strong oxen or 1024 chickens?

Seymour Cray

Consider an array A with n elements,

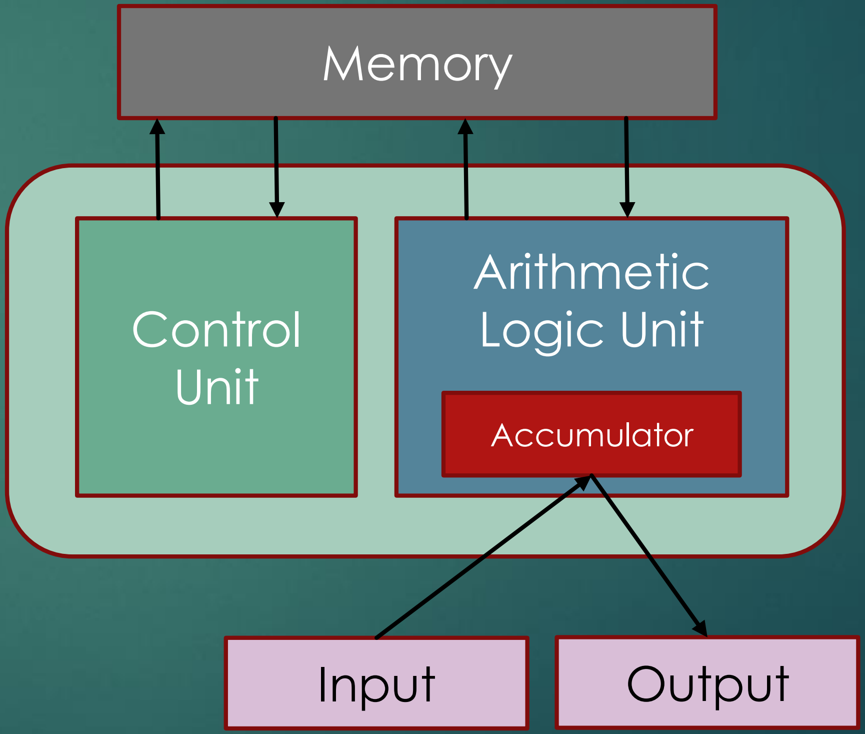
Goal: to compute,

$$x = \sum_{1}^{n} \sqrt{A_i}$$

Machine Model
Programming Model
Performance analysis

Von Neumann architecture

- Central Processing Unit (CPU, Core)
- Memory
- Input/Output (I/O)
- One instruction per unit/time
- **Sequential**



Characterizing algorithm performance

- O-notation
  - Given an input of size n, let $T(n)$ be the total time, and $S(n)$ the necessary storage
  - Given a problem, is there a way to compute lower bounds on storage and time → **Algorithmic Complexity**
  - $T(n) = O(f(n))$ means

    $T(n) \leq cf(n)$, where c is some unknown positive constant

    compare algorithms by comparing $f(n)$.

Scalability

- Scale Vertically → scale-up
  - Add resources to a single node
  - CPU, memory, disks,
- Scale Horizontally → scale-out
  - Add more nodes to the system

Parallel Performance

- Speedup

  best sequential time/time on p processors

- Efficiency

  $\text{speedup}/p, \qquad (< 1)$

Scalability

Amdahl's Law

- Sequential bottlenecks:

  Let s be the percentage of the overall work that is sequential

- Then, the speedup is given by

$$S = \frac{1}{s + \frac{1-s}{p}} \leq \frac{1}{s}$$

Gustafson

Sequential part should be independent of the problem size

Increase problem size, with increasing number of processors

Strong & Weak Scalability

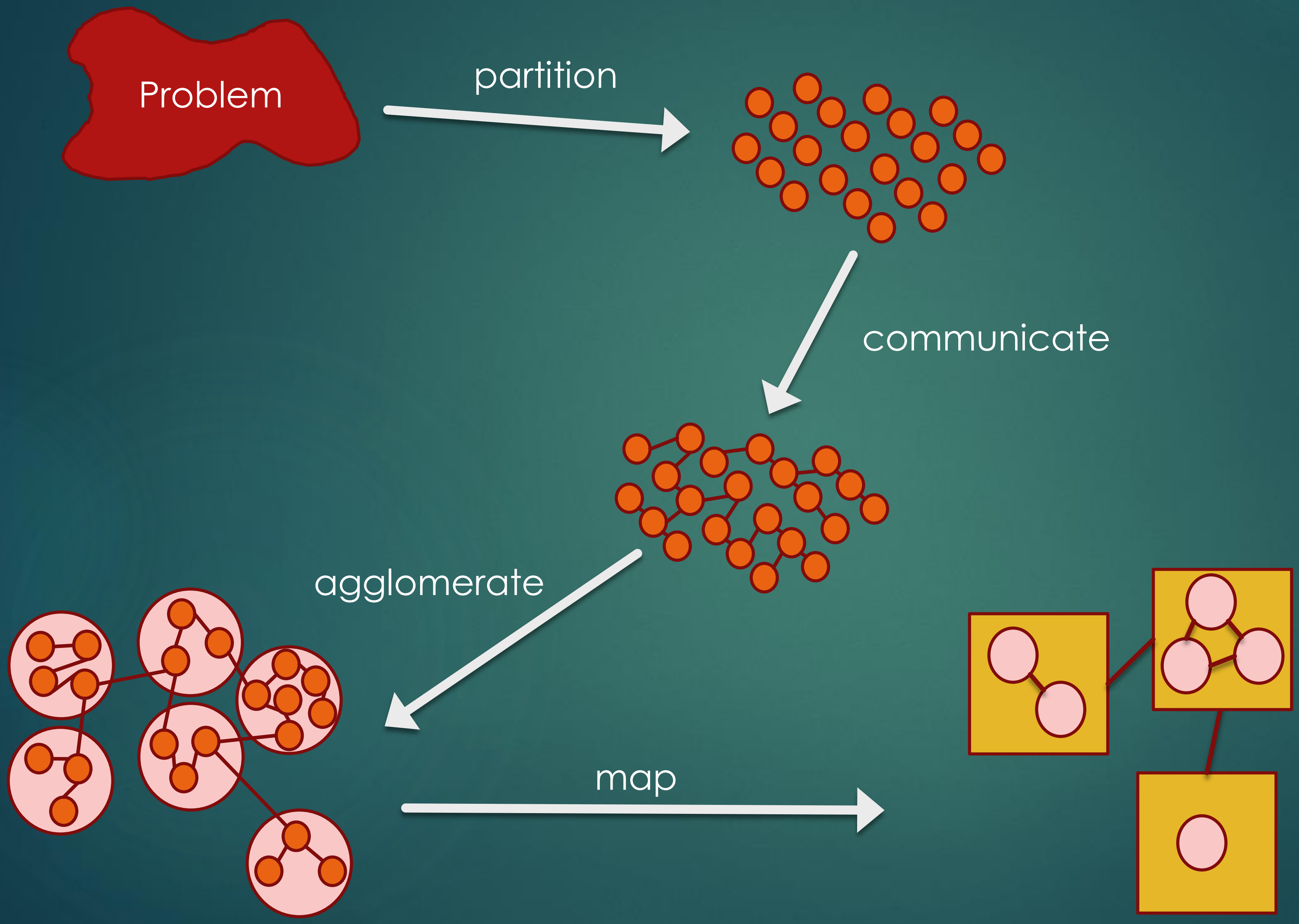
Increasing number of cores

Strong (fixed-sized) scalability — keep problem size fixed

Weak (fixed-sized) scalability — keep problem size/core fixed

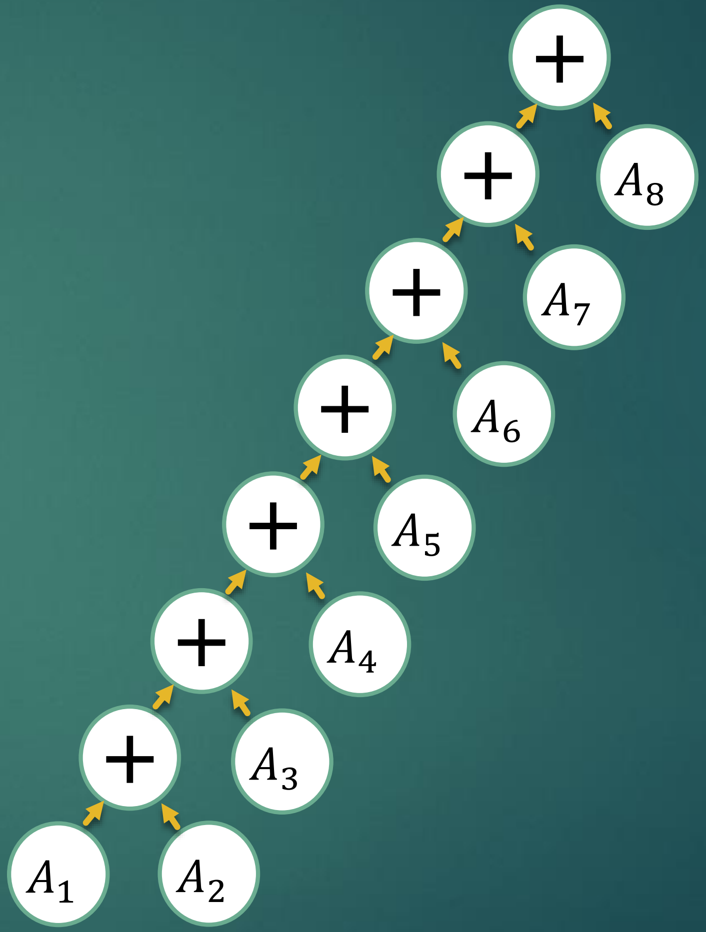
Parallel Programming

- Partition Work   Data & Tasks
- Determine Communication
- Agglomeration to number of available processors
- Map to processors
- Tune for architecture


Problem
partition
communicate
agglomerate
map

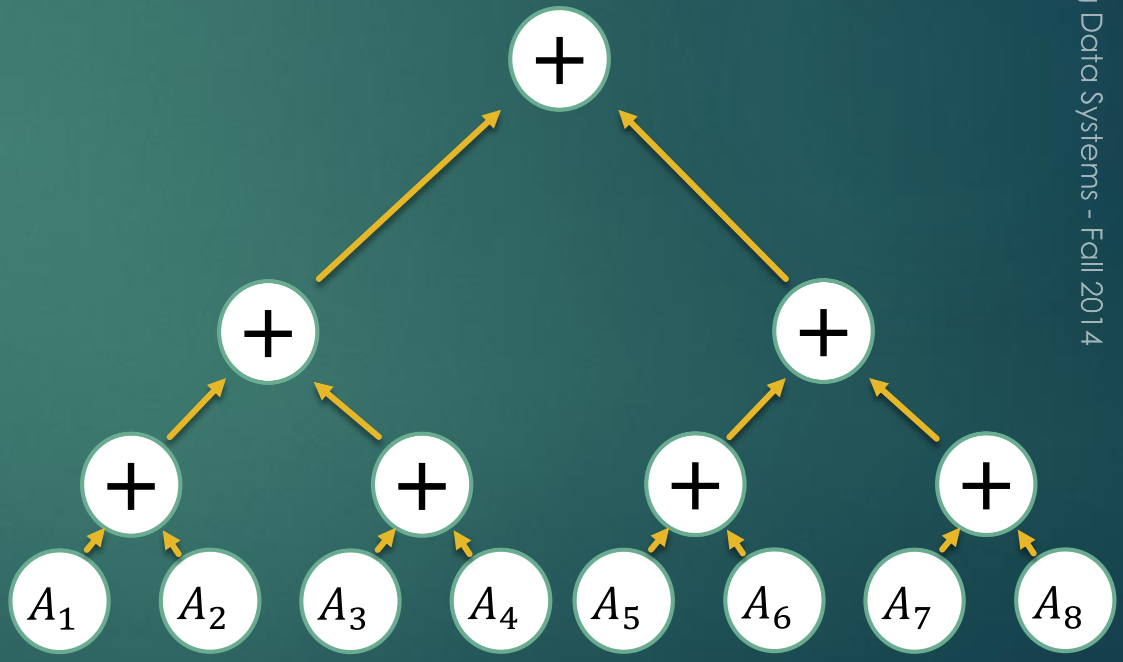
Work/Depth Models

- Abstract programming model
- Exposes the parallelism
  - Compute work W and depth D
  - D - longest chain of dependencies
  - $P = W/D$
- Directed Acyclic Graphs
- Concepts
  - parallel for (data decomposition)
  - recursion (divide and conquer)



Work/Depth Models

- Abstract programming model
- Exposes the parallelism
  - Compute work W and depth D
  - D - longest chain of dependencies
  - $P = W/D$
- Directed Acyclic Graphs
- Concepts
  - parallel for (data decomposition)
  - recursion (divide and conquer)



Parallel vs sequential for

- Dependent statements
  - $W = \sum W_i$
  - $D = \sum D_i$
- Independent statements
  - $W = \sum W_i$
  - $D = \max(D_i)$

Parallel Sum – language model

```
// Recursive implementation
Algorithm SUM(a, n)
// Input: array a of length n = 2^k, k = log n
    parallel_for i ← [0,n/2)
        b(i) ← a(2i) + a(2i+1)
    return SUM(b); // W(n/2), D(n/2)
```

Complexity:

$$D(n) = D\left(\frac{n}{2}\right) + O(1) = O(\log n)$$

$$W(n) = W\left(\frac{n}{2}\right) + O(n) = O(n)$$

Prefix Sums

SCAN

Prefix sum or a scan

- $[4 \quad 2 \quad 1 \quad 8] \rightarrow [4 \quad 4+2 \quad 4+2+1 \quad 4+2+1+8]$
  $\rightarrow [4 \quad 6 \quad 7 \quad 15]$

- Obvious sequential algorithm

$$D_s(n) = O(n)$$

- Not parallelizable, need to change algorithm

- Remark: algorithm generalizes to any associative operation

Parallel prefix sum

- Input: $\{x_i\}, \quad i = 1, \ldots, n$
- Output: $\{s_i\}, \quad i = 1, \ldots, n \quad | \quad s_i = \sum_1^{i-1} x_k$

- Sequential algorithm: best $T_s(n) = O(n)$
- Parallel
  - Work/Depth $W(n) = O(n), \ D(n) = O(\log n)$
  - Recursive and iterative implementation
- Binary tree (like sum)
  - Two tree traversals: bottom-up and top-down

Key Idea

- Divide & Conquer + Recursion
  - Break the scan in parts
  - Compute subcomponents
  - Combine
- Implementation
  - Recursive or Iterative

Divide & Conquer

Recursive Scan

```
s = Rec_Scan(a,n)
// s[k]= a[0]+a[1]+…+a[k-1], k=0,…,n-1
1.  if n=1 { s[0]←a[0]; return; } //base case
2.  parfor (i=0; i<n/2; ++i)
       b[i] ← a[2i] + a[2i+1];
3.  c ← Rec_Scan (b, n/2);
4.  s[0] ← a[0];
5.  parfor (i=0; i<n; ++i)
        if isOdd(i)
            s[i] ← c[i/2];
        elseif isEven(i)
            s[i] ← c[i/2] + a[i];
```

Iterative ?

Parallel Select

- Select numbers < pivot

$$A \leftarrow [1 \quad 2 \quad 3 \quad 0 \quad 4 \quad 0 \quad 2 \quad 3 \quad 0 \quad 1 \quad 3 \quad 4]$$

pivot ← 2

$$[1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0]$$

$$[1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 3 \quad 4 \quad 5 \quad 5 \quad 5]$$

Parallel Select the $a_i < \mathrm{pv}$

```
[l,m] ← select_lower (a, n, pv)
// t = t[0,…,n-1]
parfor (i=0; i<n; ++i)   t[i] ← a[i] < pv;
s ← scan (t);  m ← s(n-1);
parfor (i=0; i<n; ++i)   if t[i]  l[s[i] – 1] ← a[i];
```

$$W(n) = O(n)$$
$$D(n) = O(\log n)$$