CS377
Parallel Programming

Lecture 1
Introduction

Marc L. Smith

Preliminaries

- Introductions
- Why are you taking this course?
- Discuss syllabus: online

Computing Platform

- Linux command: **`$ lscpu`**
- SP 307 (Asprey) and 309 Linux workstations are:
  - single-socket
  - 12 cores per socket
  - 2 threads per core
  - 20 online CPUs (0-19)
  - why not 24 total cores? (run: **`$ lscpu -e`**)
- SC 006 (Agile) workstations have different specs
- Your computer / mobile device also has parallel computing

A motivating example

- Find the max of n numbers
  - How would you do it?
  - Assumptions?
  - How many comparisons?
    (we **are** computer scientists!)

Find max of n numbers

- Case 1: sorted (ascending)
  - no comparisons -- just return last number in array
  - O(1) comparisons
  - It's easy when numbers already sorted!
  - Hidden cost?

Find max of n numbers

- Case 2: unsorted
  - sort array first, then return max
  - O(n log(n)) comparisons
  - a lot of work just to return the max!

Find max of n numbers

- Case 3: unsorted
  - Compare unsorted numbers from first to last, keep track of max as you go
  - O(n) comparisons
  - sure beats sorting them all first!

Find max of n numbers

- Let's pause for a moment...
  - the unsorted cases are the interesting ones...
  - it takes n-1 comparisons to find the max of n unsorted numbers ~ O(n)
  - can we do better than O(n)?
    (and what do we mean by better?)

Find max of n numbers

- Let's look at the problem again
  - just the problem
  - no assumptions about its solution
- Did we make assumptions previously?
  - yes: single processor
  - what if we had more processors?

Find max of n numbers

array A:

8	6	4	2	1	3	5	7

Max = ?

All numbers must be compared at least once to find the max, but the pairs we choose to compare, and the order of those comparisons, depends on the algorithm.

Order. Traditional (imperative) programming causes us to become (necessarily) obsessed with order...

It's time to break out of the sequential box.

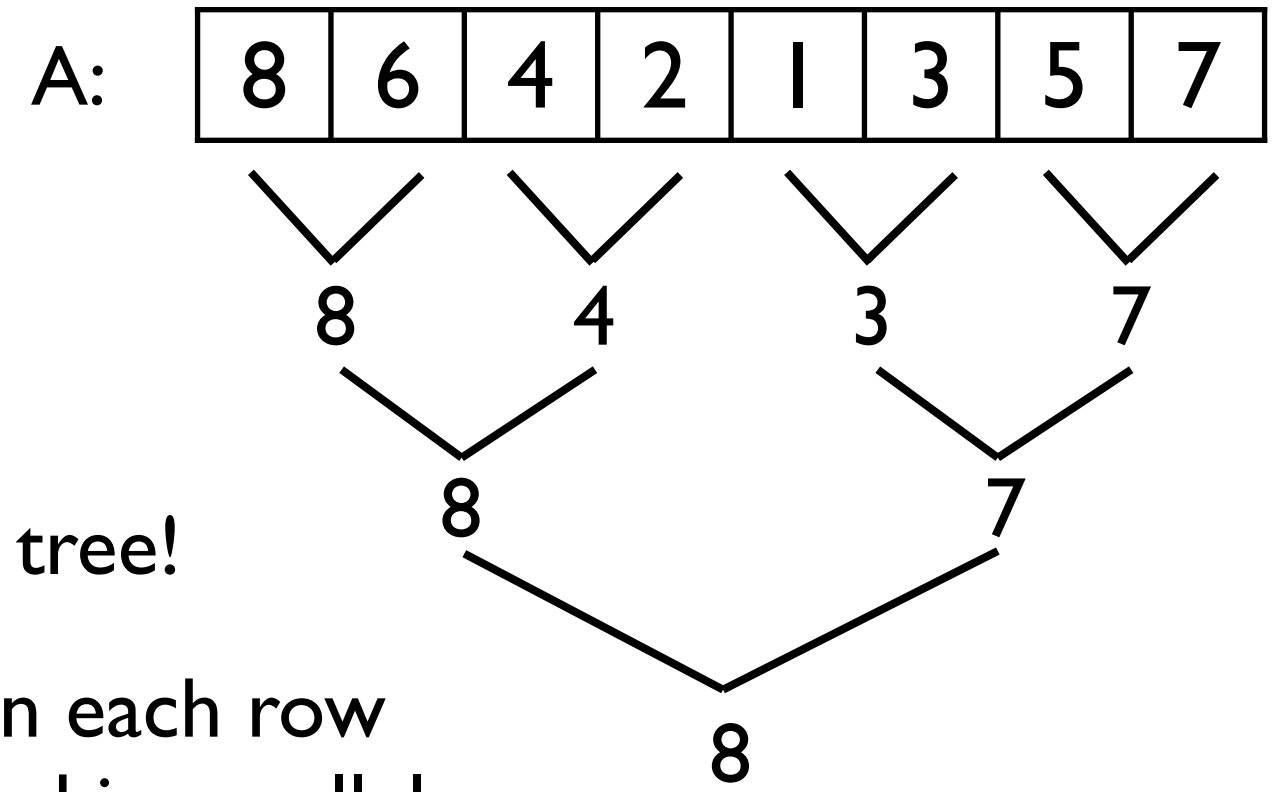
Find max of n numbers

- Why were we counting comparisons?
  - a measure independent of machine speed
- Is this still what we want if not sequential? (i.e., we can do >1 comparisons concurrently)
- We need new measures!
- Like what? (time, speedup, cost, work, efficiency)

Find max of N numbers

- Let $T_1(N)$ be the Best Sequential Algorithm
- Let $T_P(N)$ be the Time for Parallel Algorithm (P processors)
- The Speedup $S_P(N)$ is $T_1(N)/T_P(N)$
- The Cost $C_P(N)$ is $PT_P(N)$, assuming P processors
- The Work $W_P(N)$ is the summation of the number of steps taken by each of the processors. It is often, but not always, the same as Cost.
- The Cost Efficiency $CE_P(N)$ (often called efficiency $E_P(N)$) is $S_P(N)/P = C_1(N) / C_P(N) = T_1(N) / (PT_P(N))$
- The Work Efficiency $WE_P(N)$ is $W_1(N) / W_P(N) = T_1(N) / W_P(N)$

Find max of n numbers



A binary tree!

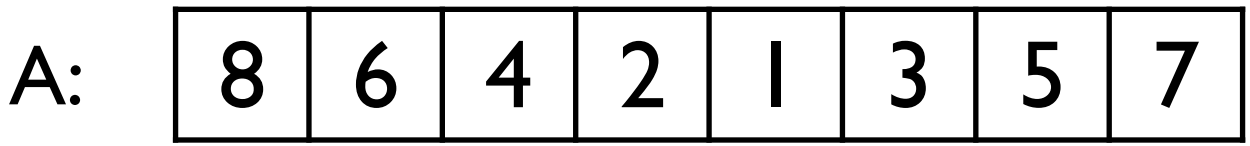
Comparisons in each row
can be performed in parallel

Time = # rows = log(n)

How many
processors needed?

Tradeoffs of time, cost, efficiency...

Find max of n numbers



This problem can be solved in
Time = O(1)

How?
Next time...

Reading Assignment 1

- Read Sutter and Larus article from September 2005 issue of ACM Queue
- Write one-page summary (no more, no less)
  - to discuss during next class
  - points in article most striking to you

Programming Assignment 1

- Write a C program that finds the max of n numbers
- Why? (it's a familiar problem)
- Goals
  - implement sequential solution to max
  - use a real editor: vim (learn it!)
  - compile / execute C program (not C++)

Programming Assignment 1

- Write a C program that finds the max of n numbers
  - main() function
    - initializes array (read values from stdin)
    - prints array
    - calls max() and prints result

Programming Assignment 1

- Write a C program that finds the max of n numbers
  - max() function
    - takes array of integers as parameter
    - iterates through array to find max
    - returns value of max element in given array