CSE 6220 INTRODUCTION TO HIGH PERFORMANCE COMPUTING

PARALLEL ALGORITHMS: DESIGN GOALS AND ANALYSIS TECHNIQUES

Ümit V. Çatalyürek

School of Computational Science and Engineering

Georgia Institute of Technology

Measuring Algorithmic Performance

Sequential Computing

$T(n)$: sequential runtime

n = problem size

$T(n) = O(f(n))$ $[\leq c\, f(n)]$

$T(n) = \Omega(g(n))$ $[\geq d\, g(n)]$

$T(n) = \Theta(f(n))$ $[O(f(n))$ and $\Omega(f(n))]$

Parallel Computing

$T(n,p)$: parallel runtime

n = problem size

p = number of processors

Can different algorithms perform better for different values of p?

Speedup

- Speedup = $\dfrac{\text{Run-time of the best Sequential Alg.}}{\text{Run-time of the Parallel Alg.}}$

$$S(p) = \frac{T(n,1)}{T(n,p)}$$

Lemma 1: $S(p) \leq p$

Proof: By Contradiction.

Suppose

$$S(p) > p$$

$$\Rightarrow \frac{T(n,1)}{T(n,p)} > p$$

$$\Rightarrow T(n,1) > p \cdot T(n,p)$$

Lemma 1 – Proof (cont'd)

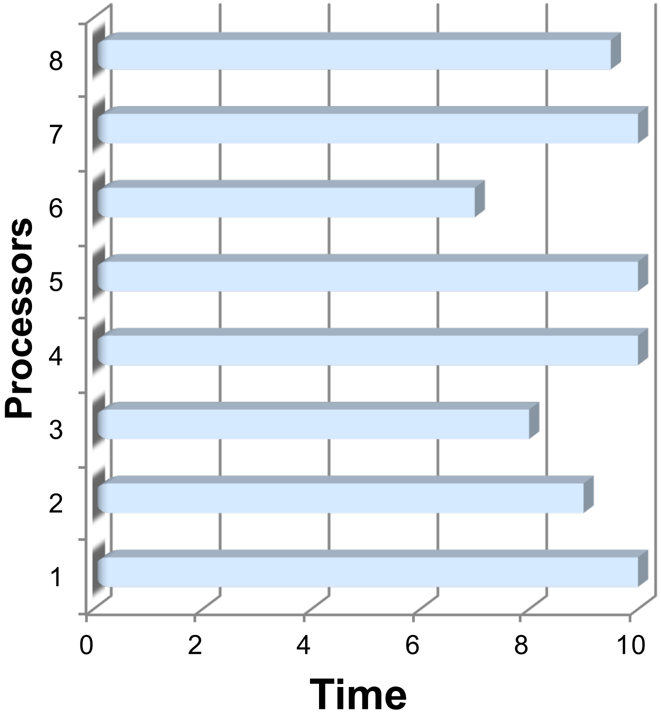
Design a new sequential alg. by simulating the actions of processors in the parallel alg. using a single processor.

Run-time of new sequential alg.

$$T(n,1) \le p \cdot T(n,p)$$

But this contradicts with

$$T(n,1) > p \cdot T(n,p) \quad \blacksquare$$



In Theory

"In theory, theory and practice are the same.
In practice, they are not."

In practice, Superlinear Speedup is possible

- Memory Access Issues
  - Cache vs. main memory

- $T(n,p)$ is for worst-case complexity
  - Examples:
    - Sorted list in sorting
    - Search

Parallel Efficiency

$$\text{Efficiency} = \frac{\text{Work done by the best seq. alg.}}{\text{Work done by the parallel alg.}}$$

$$E(p) = \frac{T(n,1)}{p \cdot T(n,p)} = \frac{S(p)}{p} \leq 1$$

Efficiency is a measure of how effectively resources of a parallel computer are utilized

Speedup vs Efficiency

What should be the aim when designing a parallel algorithm?

Example: Matrix Matrix Multiplication

better at 2.37

3-loop naïve sequential algorithm: $T(n,1) = O(n^3)$

Parallel Alg 1: $T(n,n^3) = O(\log n)$

Parallel Alg 2: $T(n,n^2) = O(n)$

Speedup vs Efficiency

	Speedup	Efficiency
Alg 1: $T(n,n^3) = O(\log n)$	$\frac{n^3}{\log n}$	
Alg 2: $T(n,n^2) = O(n)$		

Speedup vs Efficiency

What should be the aim when designing a parallel algorithm?

- Example: Matrix Matrix Multiplication
- $T(n,1) = \Theta(n^3)$ $\leq c\, f(n)$ $\approx c\, f(n)$
- Alg 1: $T(n,n^3) = \Theta(\log n)$
- Alg 2: $T(n,n^2) = \Theta(n)$

Speedup vs Efficiency

	Speedup	Efficiency
Alg 1: $T(n,n^3)=\Theta(\log n)$	$\frac{n^3}{\log n}$	$\frac{n^3}{n^3 \log n} = \frac{1}{\log n}$
Alg 2: $T(n,n^2)=\Theta(n)$		

Speedup vs Efficiency

	Speedup	Efficiency
Alg 1: $T(n,n^3) = \Theta(\log n)$	$\Theta\left(\frac{n^3}{\log n}\right)$ ★	$\Theta\left(\frac{1}{\log n}\right)$
Alg 2: $T(n,n^2) = \Theta(n)$	$\Theta\left(n^2\right)$	$\Theta\left(1\right)$ ★

Lemma 2: (Brent's Lemma)

- Let $T(n, p_1)$ be the run-time of a parallel alg. designed to run on p_1 processors.
- The same alg. can be run on $p_2 < p_1$ processors without loss of efficiency, i.e., $E(p_2) = E(p_1)$.

- This is also called efficiency scaling.

Lemma 2: (Brent's Lemma)

Proof:

To run on p_2 processors, let each processor simulate $\frac{p_1}{p_2}$ processors.

$$T(n, p_2) = \frac{p_1}{p_2} T(n, p_1)$$

$$E(p_2) = \frac{T(n,1)}{p_2 \cdot T(n, p_2)} = \frac{T(n,1)}{p_2 \cdot \frac{p_1}{p_2} T(n, p_1)} = E(p_1)$$

Lemma 2: (Brent's Lemma)

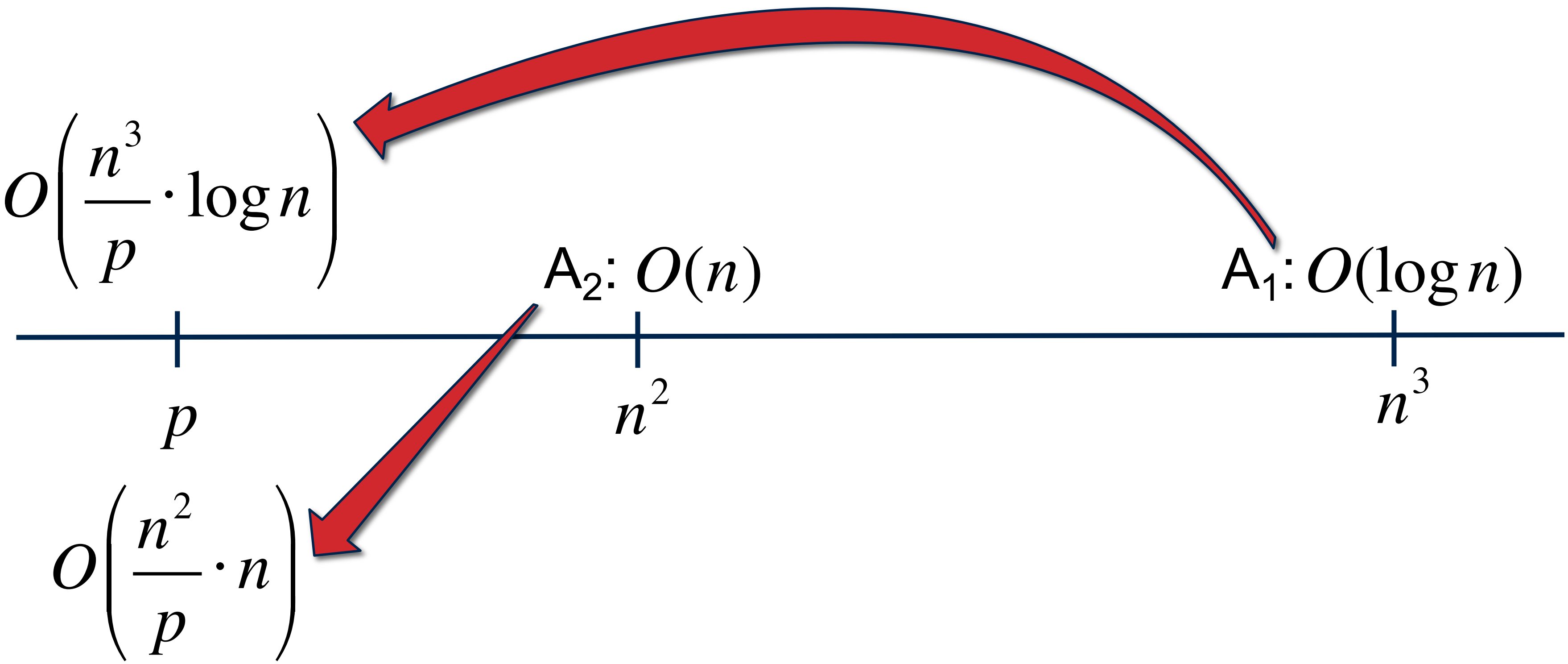
Proof: To run on p_2 processors, let each processor simulate $\left\lceil \frac{p_1}{p_2} \right\rceil$ processors.

$$T(n,p_2) = \left\lceil \frac{p_1}{p_2} \right\rceil T(n,p_1)$$

$$E(p_2) = \frac{T(n,1)}{p_2 \cdot \left\lceil \frac{p_1}{p_2} \right\rceil T(n,p_1)}$$

$$\geq \frac{T(n,1)}{p_2 \cdot \left(\frac{p_1}{p_2} + 1 \right) T(n,p_1)} = \frac{T(n,1)}{\left(p_1 + p_2\right) T(n,p_1)} \geq \frac{1}{2} E(p_1) = \Theta(E(p_1))$$

Speedup vs Efficiency



A$_2$ is faster!

Goals of Parallel Algorithm Design

- Speed: Design alg. to minimize $T(n,p)$ using the smallest possible value p.
- Efficiency: Design alg. to maximize $E(p)$ s.t. p is the largest possible.

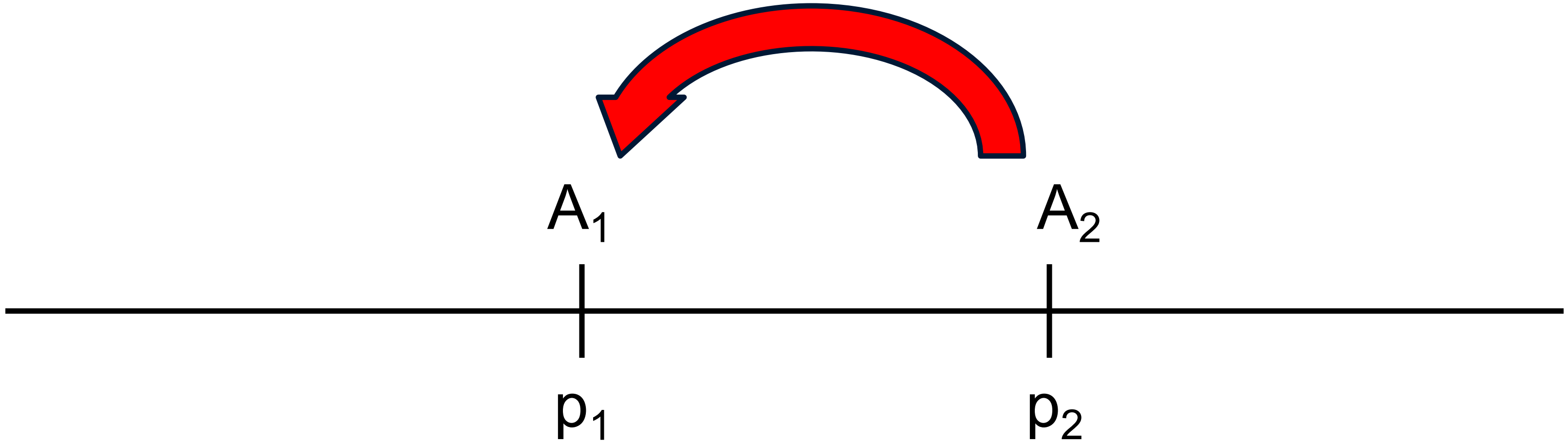
Goals of Parallel Algorithm Design

- A_1: Designed for speed and uses p_1 processors
- A_2: Designed for efficiency and uses a maximum of p_2 processors

$$p_1 \overset{?}{\gtrless} p_2$$

Goals of Parallel Algorithm Design

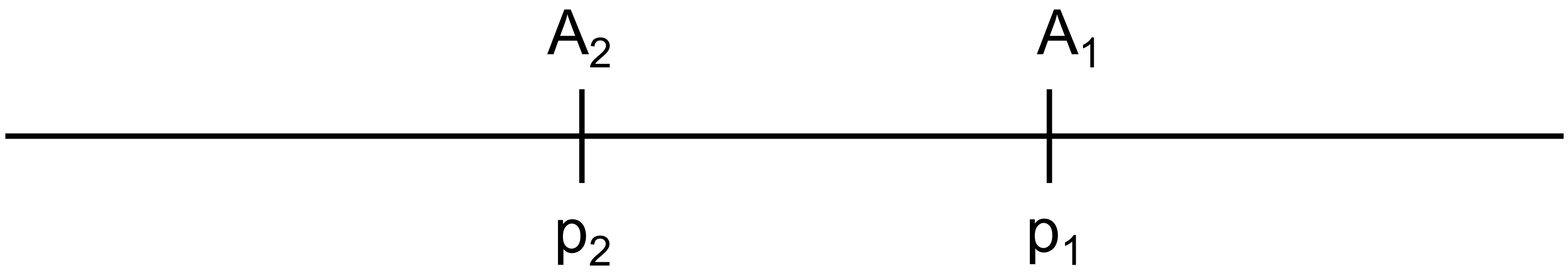
- Otherwise:



We can use Brent's Lemma and scale A_2
A_2 will run faster than A_1 on p_1 processors

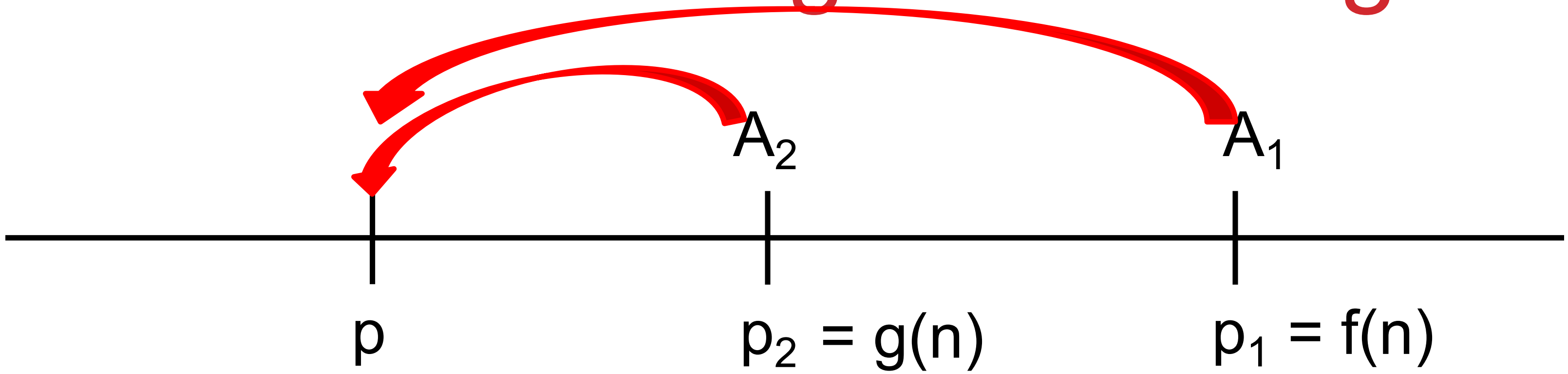
Contradicts A_1 is designed for speed and
A_2 is designed for efficiency

Goals of Parallel Algorithm Design



- A_2 (A_1) can run on less number of processors with their respective efficiencies preserved
- Only guarantee on p_2, A_1 will be less efficient than A_2
- A_2 will be more efficient, and hence provide better speedup, on any number of processors $\leq p_2$

Goals of Parallel Algorithm Design



- We use larger number processors to solve larger problems.
- Think our matrix multiplication example
  - Where $f(n) = O(n^3)$ and $g(n) = O(n^2)$
  - Say we want to solve a problem that is twice of the problem we solve on p processor

Matrix Multiplication

Sequential: $T(n,1) = O(n^3)$

Alg 1: $T(n,n^3) = O(\log n)$ Not Efficient

Alg 2: $T(n,n^2) = O(n)$

Alg 3: $T(n,n\ ) = O(n^2)$

$$E(p) = \frac{T(n,1)}{p \cdot T(n,p)} = \Theta(1)$$

Which one you would use: Alg 2 or Alg 3?

Scalability

- Increase the problem size retain same runtime?
  - Fixed-time scalability

- Example: (Again) Matrix Multiplication

Scalability: Matrix Multiplication Example

If there is an efficient algorithm

- $T(n, n^3) = O(1), \quad \leq n^3$
- $T(n, p\ ) = O(\frac{n^3}{p})$
- Increase problem size by 2x
  - Work increases 8x,
  - Let's use 8x processors
- $T(2n, 8p\ ) = O\left(\frac{8n^3}{8p}\right) =$

  $O(\frac{n^3}{p})$

Inefficient algorithm

- $T(n, n^3) = O(\log n)$
- $T(n, p\ ) = O(\frac{n^3 \log n}{p})$
- Increase problem size by 2x
  - Work increases 8x,
  - Let's use 8x processors
- $T(2n, 8p\ ) = O\left(\frac{8n^3 \log 2n}{8p}\right) =$

  $O\left(\frac{n^3\ (1+\log n)}{p}\right)$

Scalability

What if we have two efficient algorithms?

A_2: $T(n, n^2) = O(n)$ $\quad\quad$ $p \leq n^2$

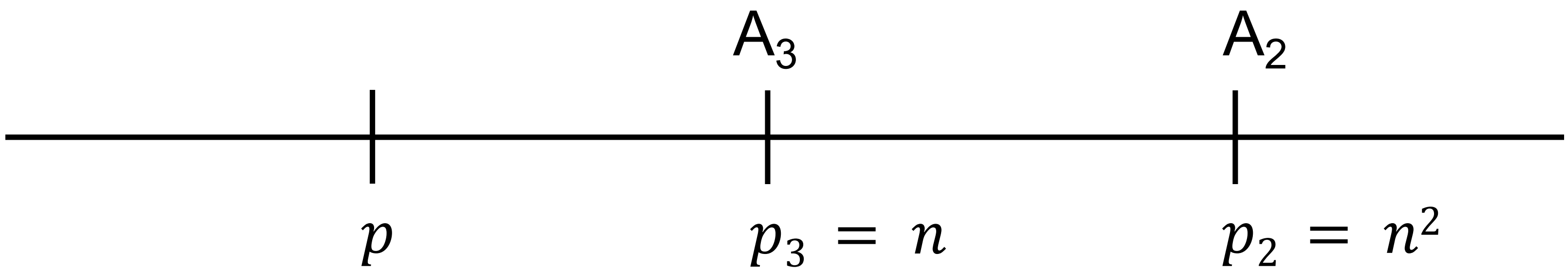
A_3: $T(n, n\ ) = O(n^2)$ $\quad\quad$ $p \leq n$

If we double the problem size n,

A_2: Can quadruple no. of processors $\Rightarrow$ doubles run-time.

A_3: Can double no. of processors $\Rightarrow$ quadruples run-time.

Scalability



- Let's say you are multiplying 2 1000×1000 matrices.
- problem size $n = 1000$ and say $p = 100$,
  - $p_3 = 1000$, $p_2 = 1000000$
- Now let's say we want to solve problem 2x, $n = 2000$
- Work is increased 8x (matrix multiplication), let's use 8x processors
- $p = 800$, $p_3 = 2000$, $p_2 = 4000000$, since $p < p_3$ and $p < p_2$ you can use either algorithm
- If we double again?
- $p = 6400$, $p_3 = 4000$, $p_2 = 16000000$, now $p > p_3$ so we cannot use A_3, so A_2 better!

Example: Speed and Efficiency Analysis

Given n numbers, compute their sum using p processors.

- Serial runtime: $T(n,1) = \Theta(n)$
- Parallel runtime: $T(n,p) = \Theta\left(\frac{n}{p} + \log p\right)$

Speed

- Find p s.t. $T(n,p)$ is minimized

$$\frac{d}{dp}\left(\frac{n}{p} + \log p\right) = 0$$

$$-\frac{n}{p^2} + \frac{1}{p} = 0$$

$$\Rightarrow p = n$$

Efficiency

- What p can we use while maintaining $E(p) = \Theta(1)$

$$E(p) = \frac{T(n,1)}{pT(n,p)} = \Theta(1)$$

$$E(p) = \frac{n}{p\left(\frac{n}{p} + \log p\right)} = \Theta(1)$$

$$\Longrightarrow \frac{n}{n + p\log p} = \Theta(1)$$

$$\Longrightarrow p\log p = O(n)$$

Efficiency

- What p can we use while maintaining $E(p) = \Theta(1)$

$$\Longrightarrow p \log p = O(n)$$

Guess: $$\Longrightarrow p = O\left(\frac{n}{\log n}\right) \approx \frac{1}{\tau}\frac{n}{\log n}$$

Substitute p

$$O\left(\frac{n}{\log n} \cdot \log \frac{n}{\log n}\right)$$

$$= O\left(\frac{n}{\log n} \cdot (\log n - \log \log n)\right)$$

$$= O\left(\frac{n}{\log n} \cdot \log n\right) = O(n)$$

Another Example

- $T(n,p) = \Theta\left(\frac{n^2}{p} + \sqrt{n}\right), p \leq n^2$

- $T(n,1) = \Theta(n^2)$

Speed

- Find p s.t. $T(n,p)$ is minimized
- Choose $p = n^2$ for speed
- $T(n,n^2) = \Theta\left(\frac{n^2}{n^2} + \sqrt{n}\right) = \Theta(\sqrt{n})$
- Efficiency?
- $E(n^2) = \frac{\Theta(n^2)}{n^2\Theta(\sqrt{n})} = \Theta\left(\frac{1}{\sqrt{n}}\right)$

Efficiency

- What p can we use while maintaining $E(p) = \Theta(1)$

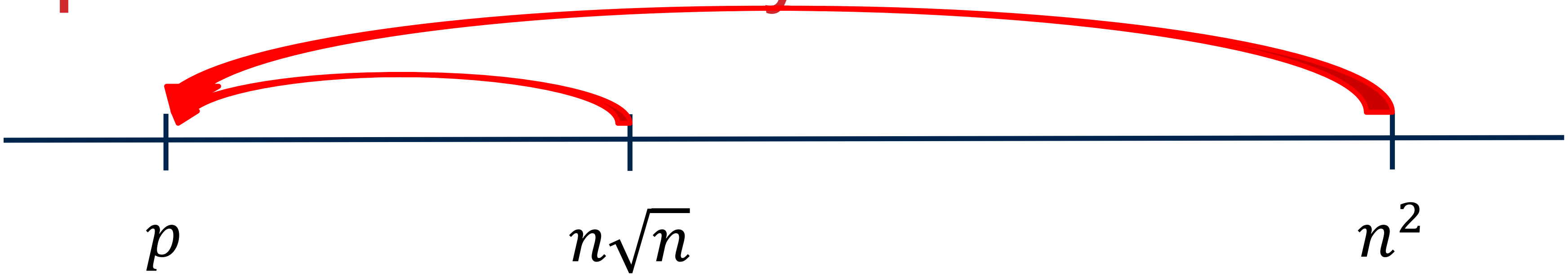
$$E(p) = \frac{\Theta(n^2)}{p \cdot \Theta\left(\frac{n^2}{p} + \sqrt{n}\right)} = \Theta(1)$$

$$\Rightarrow \frac{n^2}{n^2 + p\sqrt{n}} = \Theta(1)$$

$$\Rightarrow p\sqrt{n} = O(n^2)$$

$$\Rightarrow p = O(n\sqrt{n})$$

Speed vs. Efficiency



- Work-Optimal or Work-Efficient Algorithm