# CSCI 5451: Introduction to Parallel Computing

## Lecture 17: Advanced Analytical Modeling

computer science
& engineering

University of Minnesota
Driven to Discover®

# Lecture Overview

- Review of Asymptotic Function Growth Rates (Big-O, $\boldsymbol{\Omega}$, $\boldsymbol{\Theta}$)
- Scalability and the Definition of Work ($W$)
- Isoefficiency
- Optimal Computation Time
- Comparing Different Algorithms with Parallel Metrics

# Lecture Overview

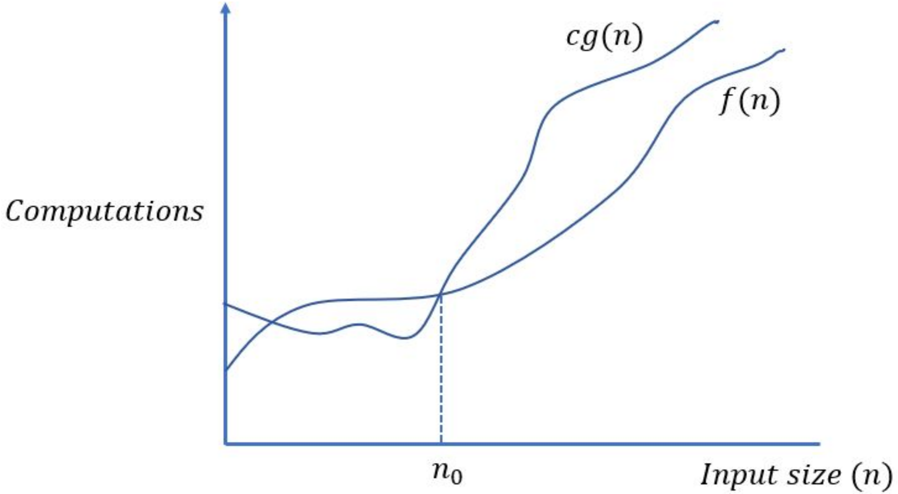
- ❑ **Review of Asymptotic Function Growth Rates (Big-O, Ω, Θ)**
- ❑ Scalability and the Definition of Work ($W$)
- ❑ Isoefficiency
- ❑ Optimal Computation Time
- ❑ Comparing Different Algorithms with Parallel Metrics

# Review of Big-O, Ω, Θ

- Big O (O)
  - Upper bound on function of interest, $f$
  - $f(n) = O(g(n)) \Rightarrow c \cdot g(n)$ is an upper bound for $f(n)$, $n > n_0$ for some $c$, $n_0$
- Big Ω (Omega)
  - Lower bound on function of interest, $f$
  - $f(n) = \Omega(g(n)) \Rightarrow c \cdot g(n)$ is a lower bound for $f(n)$, $n > n_0$ for some $c$, $n_0$
- Big Θ (Theta)
  - Tight bound on function of interest, $f$
  - $f(n) = \Theta(g(n)) \Rightarrow c_1 \cdot g(n)$ is a lower bound for $f(n)$, $c_2 \cdot g(n)$ an upper bound for $f(n)$, $n > n_0$ some $c_1$, $c_2$, $n_0$

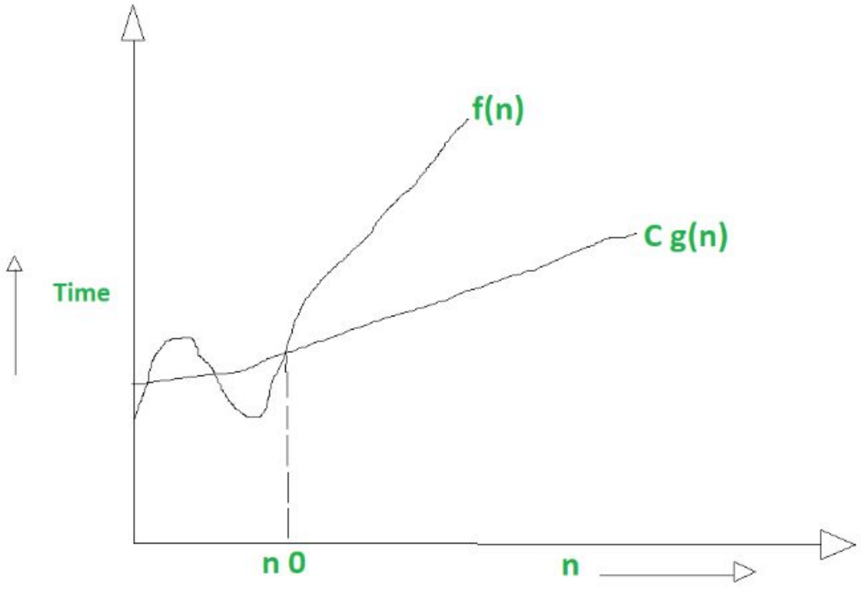
# Review of Big-O, Ω, Θ

- Big O (O)
  - Upper bound on function of interest, $f$
  - $f(n) = O(g(n)) \Rightarrow c \cdot g(n)$ is an upper bound for $f(n)$, $n > n_0$ for some $c$, $n_0$
- Big Ω (Omega)
  - Lower bound on function of interest, $f$
  - $f(n) = \Omega(g(n)) \Rightarrow c \cdot g(n)$ is a lower bound for $f(n)$, $n > n_0$ for some $c$, $n_0$
- Big Θ (Theta)
  - Tight bound on function of interest, $f$
  - $f(n) = \Theta(g(n)) \Rightarrow c_1 \cdot g(n)$ is a lower bound for $f(n)$, $c_2 \cdot g(n)$ an upper bound for $f(n)$, $n > n_0$ some $c_1$, $c_2$, $n_0$



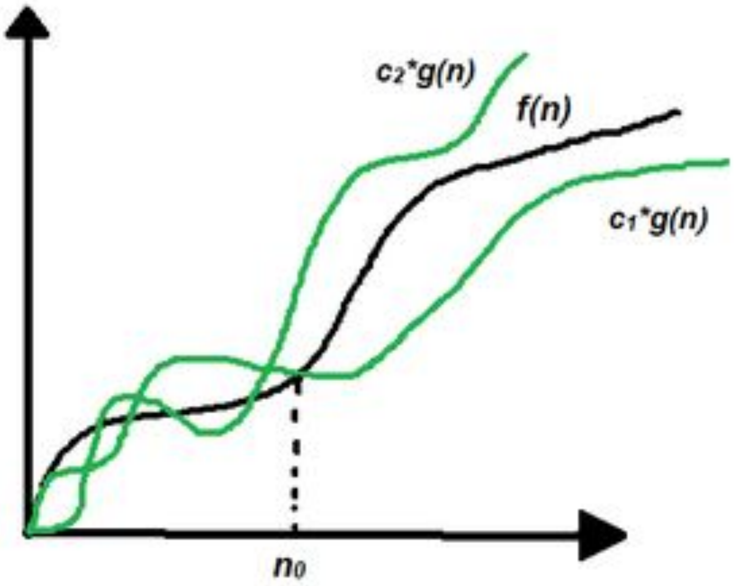
# Review of Big-O, $\Omega$, $\Theta$

- Big O (O)
  - Upper bound on function of interest, $f$
  - $f(n) = O(g(n)) \Rightarrow c \cdot g(n)$ is an upper bound for $f(n)$, $n > n_0$ for some $c$, $n_0$
- Big $\Omega$ (Omega)
  - Lower bound on function of interest, $f$
  - $f(n) = \Omega(g(n)) \Rightarrow c \cdot g(n)$ is a lower bound for $f(n)$, $n > n_0$ for some $c$, $n_0$
- Big $\Theta$ (Theta)
  - Tight bound on function of interest, $f$
  - $f(n) = \Theta(g(n)) \Rightarrow c_1 \cdot g(n)$ is a lower bound for $f(n)$, $c_2 \cdot g(n)$ an upper bound for $f(n)$, $n > n_0$ some $c_1$, $c_2$, $n_0$



# Review of Big-O, $\Omega$, $\Theta$

- Big O (O)
  - Upper bound on function of interest, $f$
  - $f(n) = O(g(n)) \Rightarrow c \cdot g(n)$ is an upper bound for $f(n)$, $n > n_0$ for some $c$, $n_0$
- Big $\Omega$ (Omega)
  - Lower bound on function of interest, $f$
  - $f(n) = \Omega(g(n)) \Rightarrow c \cdot g(n)$ is a lower bound for $f(n)$, $n > n_0$ for some $c$, $n_0$
- Big $\Theta$ (Theta)
  - Tight bound on function of interest, $f$
  - $f(n) = \Theta(g(n)) \Rightarrow c_1 \cdot g(n)$ is a lower bound for $f(n)$, $c_2 \cdot g(n)$ an upper bound for $f(n)$, $n > n_0$ some $c_1$, $c_2$, $n_0$

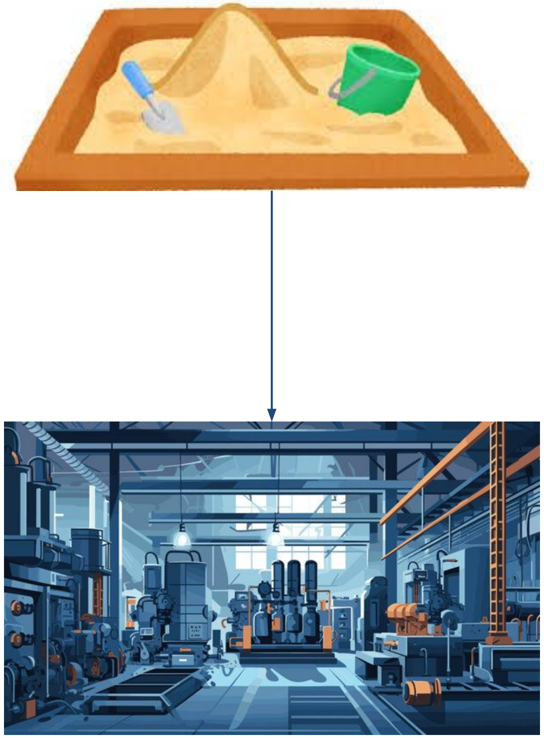


# Lecture Overview

- ❑ Review of Asymptotic Function Growth Rates (Big-O, $\Omega$, $\Theta$)
- ❑ **Scalability and the Definition of Work ($W$)**
- ❑ Isoefficiency
- ❑ Optimal Computation Time
- ❑ Comparing Different Algorithms with Parallel Metrics

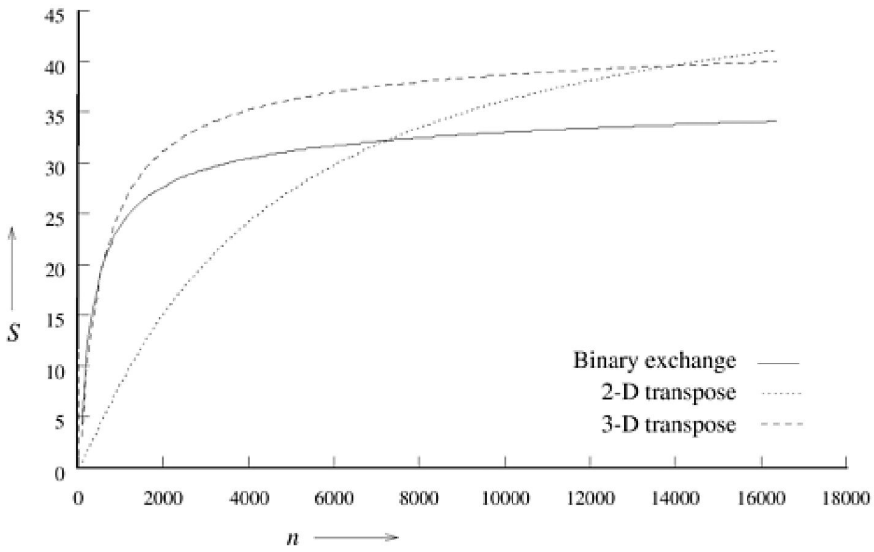
# Scalability

- Ultimately, we want to figure out how to go from programs that complete in minutes on dozens of processes to programs which take weeks on thousands of processes
- To do this, we must estimate how efficient our programs will be, as we will not be able to empirically determine their runtimes at such enormous scale



# Scalability

- We have to consider how the efficiency, parallel runtime & speedups of our program change as we vary the size of the input problem and the number of processing elements
- [Right] Speedup comparison of several different parallel algorithms for Fast Fourier Transform



# Revisiting Efficiency

We can rewrite efficiency in several different ways terms of $T_o$ and $T_s$

$$E = \frac{S}{P}$$

$$E = \frac{T_s}{p\,T_p}$$

$$E = \frac{1}{1 + \left(\frac{T_o}{T_s}\right)}$$

# Revisiting Efficiency

We can rewrite efficiency in several different ways terms of $T_o$ and $T_s$

$$E = \frac{S}{P}$$

$$E = \frac{T_s}{p\,T_p}$$

$$E = \frac{1}{1 + \left(\frac{T_o}{T_s}\right)}$$

Increasing the number of processors

- $T_o$ is an increasing function of $p$.
- Every program will have some serial portion taking up time $t_{serial}$
- All other $p-1$ processes will idle in this time
- Thus, in the best case, $T_o = \Omega(p)$
- Other costs (communication, excess computation) can further increase $T_o$
- If we fix the problem size, then $T_s$ will remain constant
- ***Increasing p will decrease efficiency E***

# Revisiting Efficiency

We can rewrite efficiency in several different ways terms of $T_o$ and $T_s$

$$E = \frac{S}{P}$$

$$E = \frac{T_s}{p\,T_p}$$

$$E = \frac{1}{1 + \left(\frac{T_o}{T_s}\right)}$$

## Increasing the problem size

- Increasing the problem size will increase the size of $T_s$
- Increasing the problem size ***can*** increase the overhead $T_o$
- ***A good parallel algorithm*** should have an efficiency that increases as the problem size grows.
- If your algorithm does not exhibit this, it will not scale to larger problems

# Revisiting Addition of N Numbers

Let's find some non-asymptotic metrics when summing *N* numbers

- Sum *n* numbers on *p* processors
- Assume each addition takes 1 unit time
- Each communication takes 1 unit time
- What are the values of
  - $T_s$ = ?
  - $T_p$ = ?
  - $T_o$ = ?
  - S = ?
  - E = ?

# Revisiting Addition of N Numbers

Let's find some non-asymptotic metrics when summing *N* numbers

- Sum *n* numbers on *p* processors
- Assume each addition takes 1 unit time
- Each communication takes 1 unit time
- What are the values of
  - $T_s$ = ?
  - $T_p$ = ?
  - $T_o$ = ?
  - S = ?
  - E = ?

$$T_s = n$$

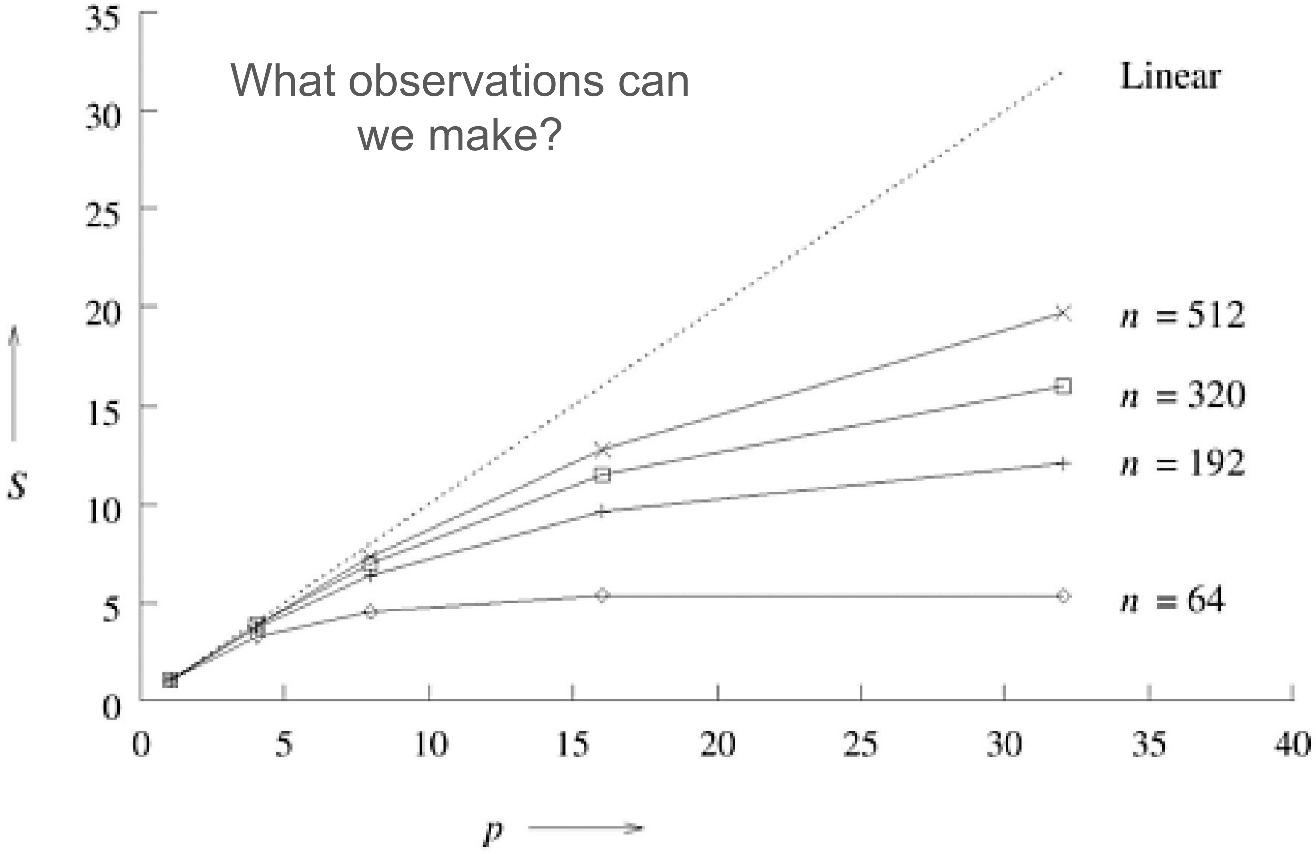
$$T_p = \frac{n}{p} + 2\log p$$

$$T_o = 2p\log p$$

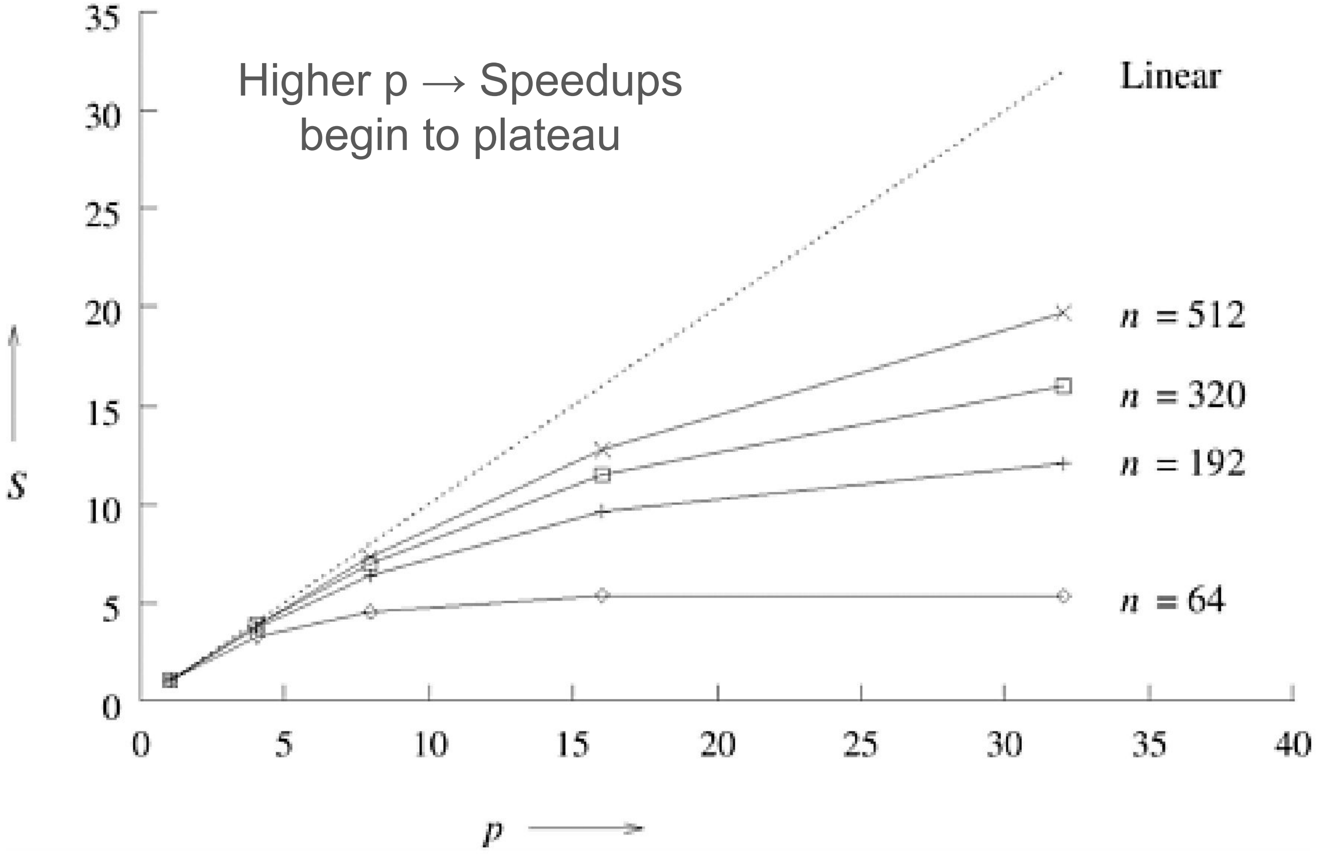
$$S = \frac{n}{\frac{n}{p} + 2\log p}$$

$$E = \frac{1}{1 + \left(\frac{2p\log p}{n}\right)}$$

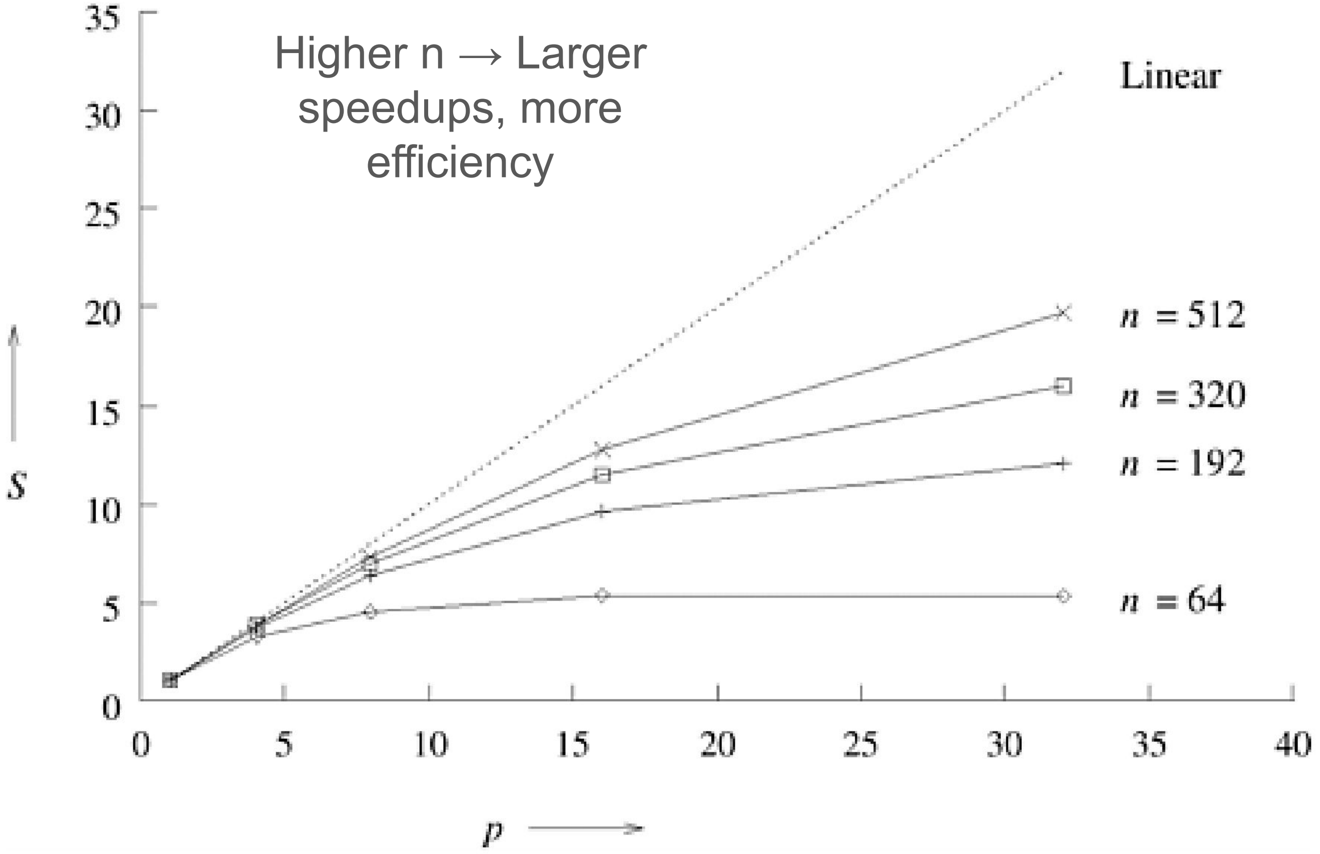
# Revisiting Addition of N Numbers



# Revisiting Addition of N Numbers



# Revisiting Addition of N Numbers



# Revisiting Addition of N Numbers

| $n$ | $p = 1$ | $p = 4$ | $p = 8$ | $p = 16$ | $p = 32$ |
|---|---|---|---|---|---|
| 64 | 1.0 | ***0.80*** | 0.57 | 0.33 | 0.17 |
| 192 | 1.0 | 0.92 | ***0.80*** | 0.60 | 0.38 |
| 320 | 1.0 | 0.95 | 0.87 | 0.71 | 0.50 |
| 512 | 1.0 | 0.97 | 0.91 | ***0.80*** | 0.62 |

# Revisiting Addition of N Numbers

For a given number of processors, *p*, and efficiency, *E*, we can determine the problem size needed (in this case *n*)

| *n* | *p* = 1 | *p* = 4 | *p* = 8 | *p* = 16 | *p* = 32 |
|---|---|---|---|---|---|
| 64 | 1.0 | ***0.80*** | 0.57 | 0.33 | 0.17 |
| 192 | 1.0 | 0.92 | ***0.80*** | 0.60 | 0.38 |
| 320 | 1.0 | 0.95 | 0.87 | 0.71 | 0.50 |
| 512 | 1.0 | 0.97 | 0.91 | ***0.80*** | 0.62 |

# Revisiting Addition of N Numbers

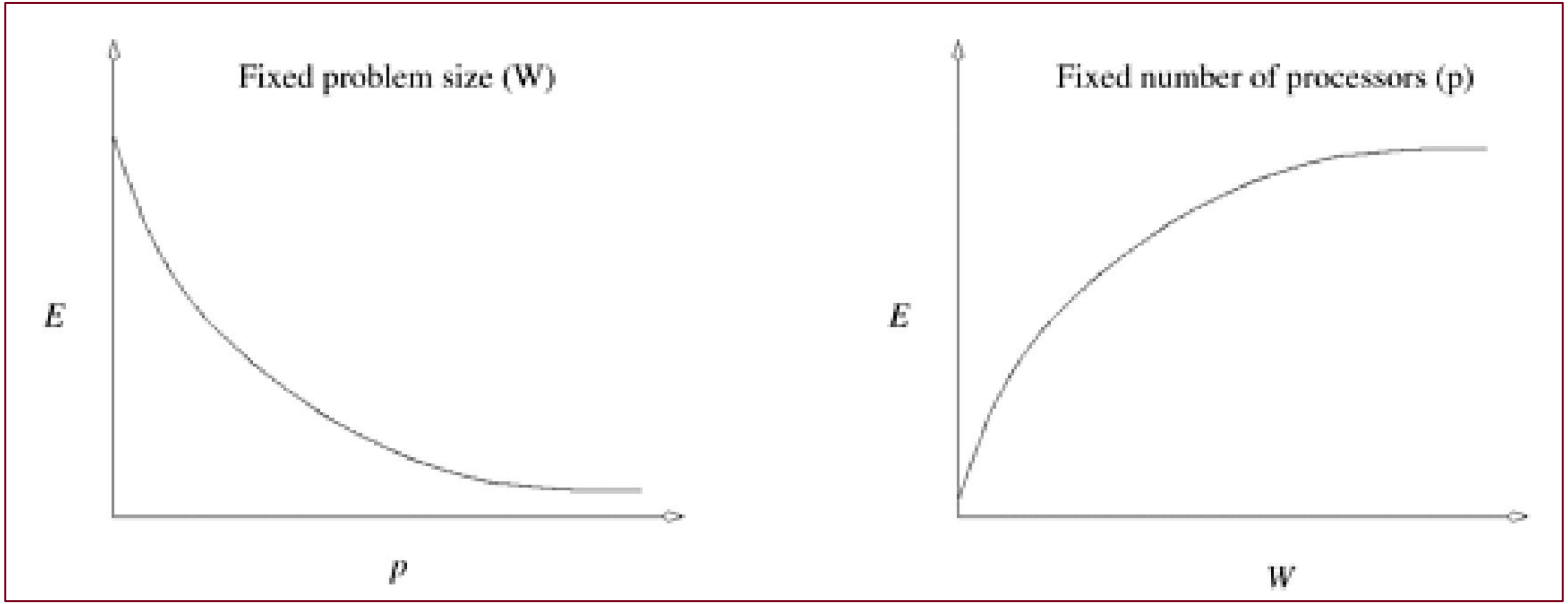
**IsoEfficiency**

For a given number of processors, *p*, and efficiency, *E*, we can determine the problem size needed (in this case *n*)

| ***n*** | ***p* = 1** | ***p* = 4** | ***p* = 8** | ***p* = 16** | ***p* = 32** |
|---|---|---|---|---|---|
| 64 | 1.0 | ***0.80*** | 0.57 | 0.33 | 0.17 |
| 192 | 1.0 | 0.92 | ***0.80*** | 0.60 | 0.38 |
| 320 | 1.0 | 0.95 | 0.87 | 0.71 | 0.50 |
| 512 | 1.0 | 0.97 | 0.91 | ***0.80*** | 0.62 |

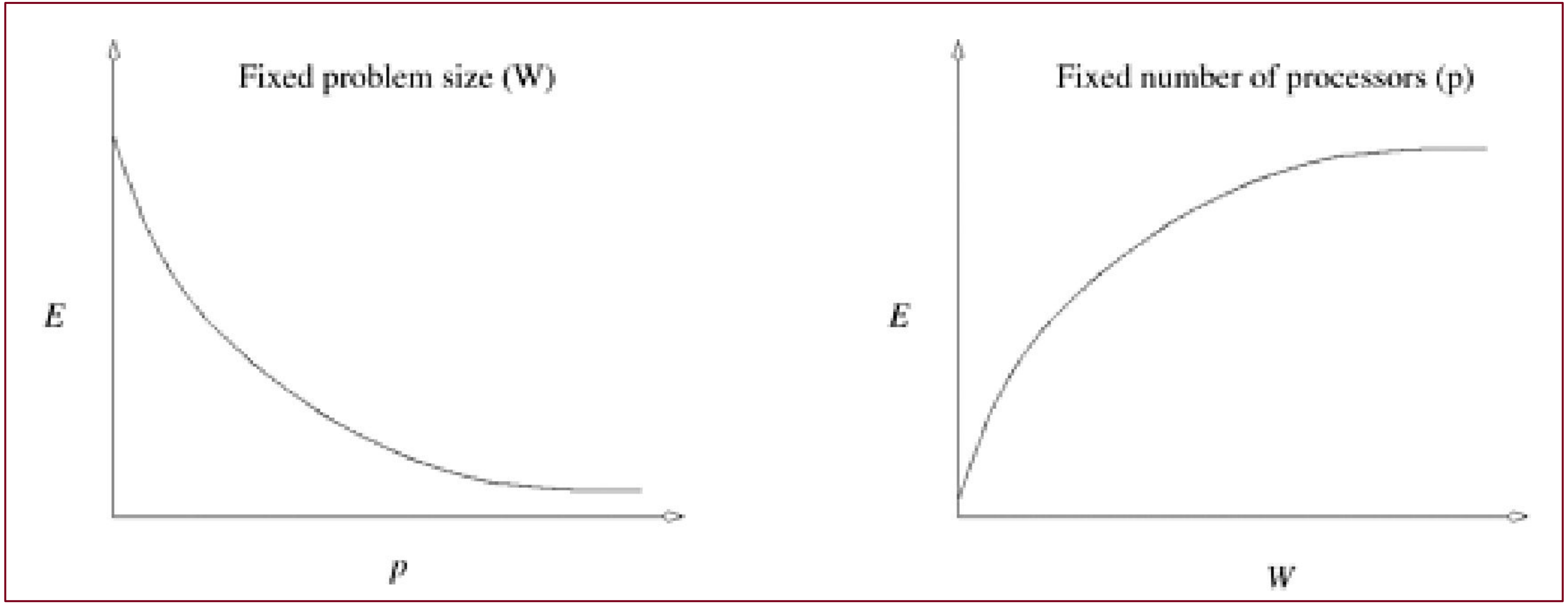
# Revisiting Addition of N Numbers

General Observation #1:
With a fixed problem size, increasing the amount of processors **decreases** efficiency



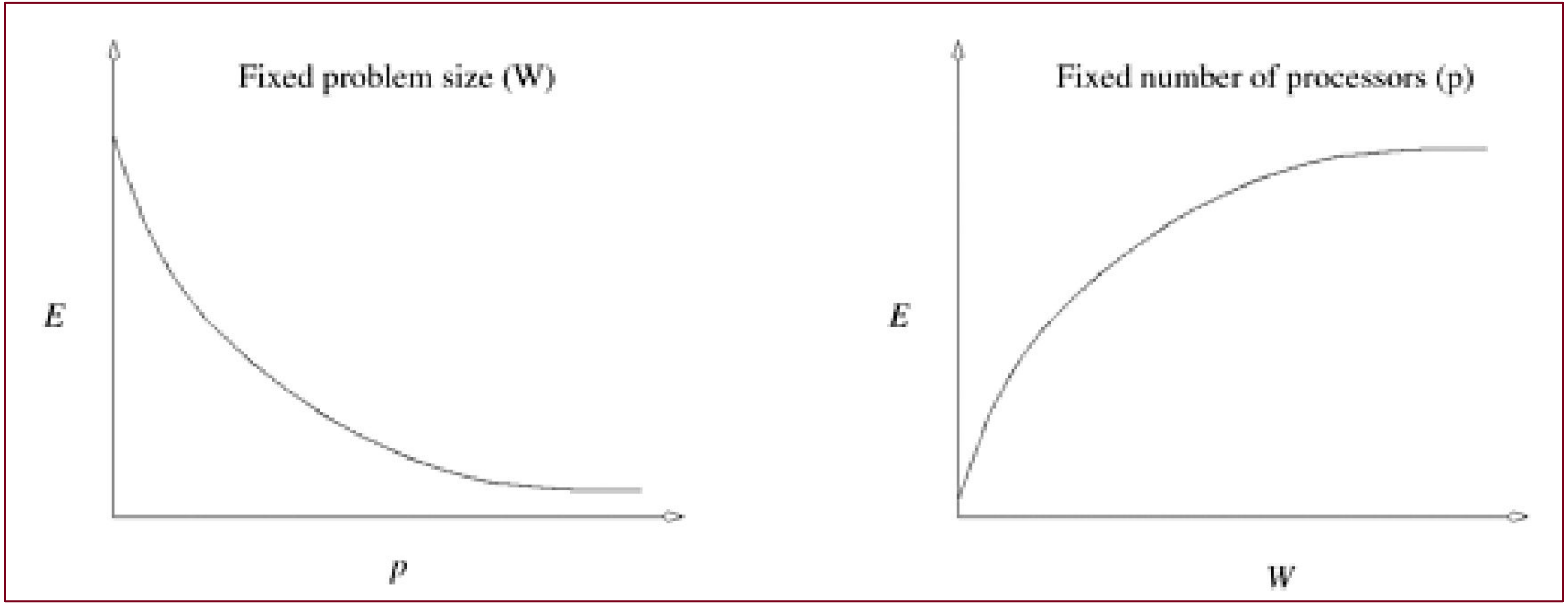
# Revisiting Addition of N Numbers

General Observation #2:
With a fixed number of processors, increasing the problem size **often increases** efficiency



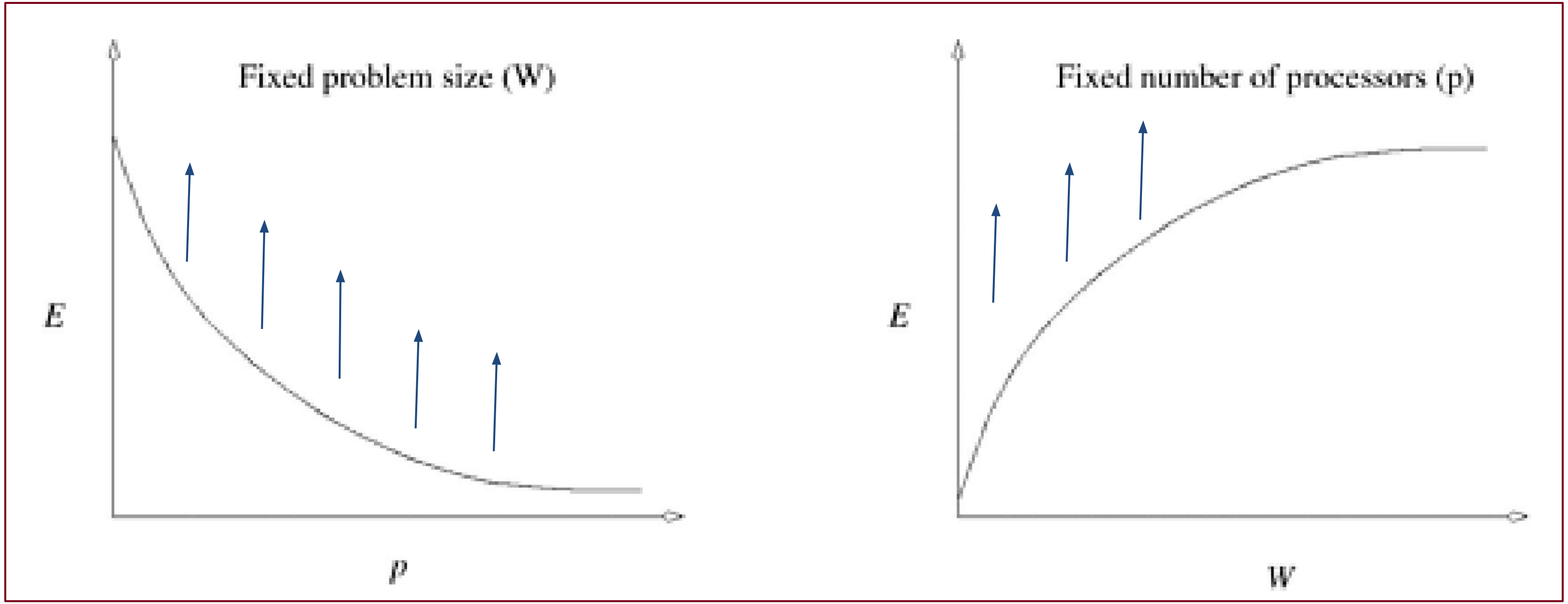
# Revisiting Addition of N Numbers

What would be more ideal versions of the below curves? In other words, how would we want efficiency to change w.r.t. to *W,p?*



# Revisiting Addition of N Numbers

What would be more ideal versions of the below curves? In other words, how would we want efficiency to change w.r. to *W,p*?



# Formally Defining Work $W$

- Up to this point, we have defined problem sizes & work in informal terms
- Naive way to define this → amount of input data $n$
- Problem with this?

# Formally Defining Work *W*

- Up to this point, we have defined problem sizes & work in informal terms
- Naive way to define this → amount of input data *n*
- Problem with this?
  - Doesn't reflect runtime
  - Consider multiplying two nxn input matrices
  - Input is $n^2$, runtime is $n^3$
- Problem Size (or Work) should reflect the serial execution complexity

# Formally Defining Work *W*

- Up to this point, we have defined problem sizes & work in informal terms
- Naive way to define this → amount of input data *n*
- Problem with this?
  - Doesn't reflect runtime
  - Consider multiplying two nxn input matrices
  - Input is $n^2$, runtime is $n^3$
- Problem Size (or Work) should reflect the serial execution complexity

Precise definition:
The problem size ***W*** is defined as the number of computation steps in the ***best*** sequential algorithm to solve the problem on a single processing element

# Redefining Metrics in Terms of W

$$T_P = \frac{W + T_o(W,p)}{p}$$

$$S = \frac{W}{T_P} = \frac{Wp}{W + T_o(W,p)}$$

$$E = \frac{S}{p} = \frac{W}{W + T_o(W,p)} = \frac{1}{1 + \frac{T_o(W,p)}{W}}$$

# Deriving W as a function of *p*, *T<sub>o</sub>*, *E*

$$E = \frac{1}{1 + \frac{T_o(W,p)}{W}},$$

$$\frac{T_o(W,p)}{W} = \frac{1-E}{E},$$

$$W = \frac{E}{1-E} T_o(W,p),$$

$$W = K\, T_o(W,p)$$

# Deriving W as a function of *p*, $T_o$, *E*

$$E = \frac{1}{1 + \frac{T_o(W,p)}{W}},$$

$$\frac{T_o(W,p)}{W} = \frac{1-E}{E},$$

$$W = \frac{E}{1-E} T_o(W,p),$$

$$W = K\, T_o(W,p)$$

For a constant efficiency, ***E***, we can define the Work, ***W***, as a function of the overhead $T_o$

# Deriving W as a function of *p*, $T_o$, *E*

$$E = \frac{1}{1 + \frac{T_o(W,p)}{W}},$$

$$\frac{T_o(W,p)}{W} = \frac{1-E}{E},$$

$$W = \frac{E}{1-E} T_o(W,p),$$

$$W = K\, T_o(W,p)$$

For a constant efficiency, ***E***, we can define the Work, ***W***, as a function of the overhead

**IsoEfficiency**

# Lecture Overview

- Review of Asymptotic Function Growth Rates (Big-O, $\Omega$, $\Theta$)
- Scalability and the Definition of Work ($W$)
- **Isoefficiency**
- Optimal Computation Time
- Comparing Different Algorithms with Parallel Metrics

# Isoefficiency

- This tells us f we change the number of processors, by what amount must we change the work to keep efficiency constant
- Usefulness → Understanding how well your program will scale when going from smaller problems to bigger ones

$$\boxed{W = K\,T_o(W, p)}$$

# Comparing Isoefficiency Functions

For each of the two overhead functions at right

- What are their isoefficiency functions?

$$\text{(A)} \quad T_o = \log p$$

$$\text{(B)} \quad T_o = p$$

- Which has a better isoefficiency function and why?

# Comparing Isoefficiency Functions

For each of the two overhead functions at right

- What are their isoefficiency functions?

$$(A) \quad W = K \log p$$

$$(B) \quad W = Kp$$

- Which has a better isoefficiency function and why?
  - A has better scaling. If we increase p from 2 to 64, then maintaining efficiency on (A), requires increasing the work by 6x, whereas on (B) we must increase work by 32x

$$(A) \quad T_o = \log p$$

$$(B) \quad T_o = p$$

# Isoefficiency (Adding n numbers)

- The overhead function for adding n numbers is

$$T_o = 2p\log p$$

- The isoefficiency function is

$$W = 2kp\log p$$

- Asymptotically this is $\Theta(p\log p)$
- In other words, if we increase the number of processors from p to p', how much will we need to increase the amount of work by to maintain constant efficiency?

# Isoefficiency (Adding n numbers)

- The overhead function for adding n numbers is

$$T_o = 2p\log p$$

- The isoefficiency function is

$$W = 2kp\log p$$

- Asymptotically this is $\Theta(p\log p)$
- In other words, if we increase the number of processors from p to p', how much will we need to increase the amount of work by to maintain constant efficiency?

$$\frac{p' \log p'}{p \log p}$$

# Isoefficiency (Complex Example)

- In the simple example of adding $n$ numbers, the overhead function is simple & only a function of $p$
- Most overhead functions will be a function of both $p$ and $W$
- It can be more difficult to define a closed form expression of $W$
- As such, we instead compute the isoefficiency function for each separate component - only considering the function that grows largest as a function of $p$

$$T_o = p^{3/2} + p^{3/4}W^{3/4}$$

# Isoefficiency (Complex Example)

- In the simple example of adding $n$ numbers, the overhead function is simple & only a function of $p$
- Most overhead functions will be a function of both $p$ and $W$
- It can be more difficult to define a closed form expression of $W$
- As such, we instead compute the isoefficiency function for each separate component - only considering the function that grows largest as a function of $p$

$$T_o = \boxed{p^{3/2}} + p^{3/4}W^{3/4}$$

$$\boxed{W = Kp^{3/2}}$$

# Isoefficiency (Complex Example)

- In the simple example of adding *n* numbers, the overhead function is simple & only a function of *p*
- Most overhead functions will be a function of both *p* and *W*
- It can be more difficult to define a closed form expression of *W*
- As such, we instead compute the isoefficiency function for each separate component - only considering the function that grows largest as a function of *p*

$$T_o = \boxed{p^{3/2}} + \boxed{p^{3/4}W^{3/4}}$$

$$\boxed{W = Kp^{3/2}}$$

$$W = Kp^{3/4}W^{3/4}$$

$$W^{1/4} = Kp^{3/4}$$

$$W = K^4p^3$$

# Isoefficiency (Complex Example)

- In the simple example of adding $n$ numbers, the overhead function is simple & only a function of $p$
- Most overhead functions will be a function of both $p$ and $W$
- It can be more difficult to define a closed form expression of $W$
- As such, we instead compute the isoefficiency function for each separate component - only considering the function that grows largest as a function of $p$

**Take largest**

$$T_o = \boxed{p^{3/2}} + \boxed{p^{3/4}W^{3/4}}$$

$$W = Kp^{3/4}W^{3/4}$$

$$W^{1/4} = Kp^{3/4}$$

$$\boxed{W = K^4p^3}$$

# Lecture Overview

- ❑ Review of Asymptotic Function Growth Rates (Big-O, $\mathbf{\Omega}$, $\mathbf{\Theta}$)
- ❑ Scalability and the Definition of Work ($W$)
- ❑ Isoefficiency
- ❑ **Optimal Computation Time**
- ❑ Comparing Different Algorithms with Parallel Metrics

# Optimal Computation Time

- A common question we may ask is what the minimum parallel runtime ($T_p$) is and what the minimum number of processors are needed to achieve this minimum runtime?
- How can we achieve this?

# Optimal Computation Time

- A common question we may ask is what the minimum parallel runtime ($T_p$) is and what the minimum number of processors are needed to achieve this minimum runtime?
- How can we achieve this?
  - Take the derivative of $T_p$ with respect to *p*
  - Find $p_0$ where this derivative is 0
  - Use this *p* as the input to $T_p$

$$\frac{d}{dp}T_P = 0$$

# Optimal Computation Time (Adding n numbers)

$$\boxed{T_p = \frac{n}{p} + 2\log p}$$

# Optimal Computation Time (Adding n numbers)

$$\boxed{T_p = \frac{n}{p} + 2\log p}$$

$$\frac{d}{dp} T_P = -\frac{n}{p_0^2} + \frac{2}{p_0} = 0,$$

$$-n + 2p_0 = 0,$$

$$p_0 = \frac{n}{2},$$

$$T_P^{min} = n \log n - n \log 2 + 2$$

# Optimal Computation Time (Complex Example)

$$\boxed{T_P = \frac{W}{p} + p^{1/2} + \frac{W^{3/4}}{p^{1/4}}}$$

# Optimal Computation Time (Complex Example)

$$\frac{d}{dp}T_P = -\frac{W}{p_0^2} + \frac{1}{2p_0^{1/2}} - \frac{W^{3/4}}{4p_0^{5/4}} = 0,$$

$$\boxed{T_P = \frac{W}{p} + p^{1/2} + \frac{W^{3/4}}{p^{1/4}}}$$

# Optimal Computation Time (Complex Example)

$$\boxed{T_P = \frac{W}{p} + p^{1/2} + \frac{W^{3/4}}{p^{1/4}}}$$

$$\frac{d}{dp}T_P = -\frac{W}{p_0^2} + \frac{1}{2p_0^{1/2}} - \frac{W^{3/4}}{4p_0^{5/4}} = 0,$$

$$-W + \frac{1}{2}p_0^{3/2} - \frac{1}{4}W^{3/4}p_0^{3/4} = 0,$$

# Optimal Computation Time (Complex Example)

$$\boxed{T_P = \frac{W}{p} + p^{1/2} + \frac{W^{3/4}}{p^{1/4}}}$$

$$\frac{d}{dp}T_P = -\frac{W}{p_0^2} + \frac{1}{2p_0^{1/2}} - \frac{W^{3/4}}{4p_0^{5/4}} = 0,$$

$$-W + \frac{1}{2}p_0^{3/2} - \frac{1}{4}W^{3/4}p_0^{3/4} = 0,$$

$$p_0^{3/4} = \frac{1}{4}W^{3/4} \pm \left(\frac{1}{16}W^{3/2} + 2W\right)^{1/2}$$

# Optimal Computation Time (Complex Example)

$$T_P = \frac{W}{p} + p^{1/2} + \frac{W^{3/4}}{p^{1/4}}$$

$$\frac{d}{dp}T_P = -\frac{W}{p_0^2} + \frac{1}{2p_0^{1/2}} - \frac{W^{3/4}}{4p_0^{5/4}} = 0,$$

$$-W + \frac{1}{2}p_0^{3/2} - \frac{1}{4}W^{3/4}p_0^{3/4} = 0,$$

$$p_0^{3/4} = \frac{1}{4}W^{3/4} \pm \left(\frac{1}{16}W^{3/2} + 2W\right)^{1/2}$$

$$= \Theta(W^{3/4}),$$

# Optimal Computation Time (Complex Example)

$$\boxed{T_P = \frac{W}{p} + p^{1/2} + \frac{W^{3/4}}{p^{1/4}}}$$

$$\frac{d}{dp}T_P = -\frac{W}{p_0^2} + \frac{1}{2p_0^{1/2}} - \frac{W^{3/4}}{4p_0^{5/4}} = 0,$$

$$-W + \frac{1}{2}p_0^{3/2} - \frac{1}{4}W^{3/4}p_0^{3/4} = 0,$$

$$p_0^{3/4} = \frac{1}{4}W^{3/4} \pm \left(\frac{1}{16}W^{3/2} + 2W\right)^{1/2}$$

$$= \Theta(W^{3/4}),$$

$$p_0 = \Theta(W),$$

# Optimal Computation Time (Complex Example)

$$\boxed{T_P = \frac{W}{p} + p^{1/2} + \frac{W^{3/4}}{p^{1/4}}}$$

$$\frac{d}{dp}T_P = -\frac{W}{p_0^2} + \frac{1}{2p_0^{1/2}} - \frac{W^{3/4}}{4p_0^{5/4}} = 0,$$

$$-W + \frac{1}{2}p_0^{3/2} - \frac{1}{4}W^{3/4}p_0^{3/4} = 0,$$

$$p_0^{3/4} = \frac{1}{4}W^{3/4} \pm \left(\frac{1}{16}W^{3/2} + 2W\right)^{1/2}$$

$$= \Theta(W^{3/4}),$$

$$p_0 = \Theta(W),$$

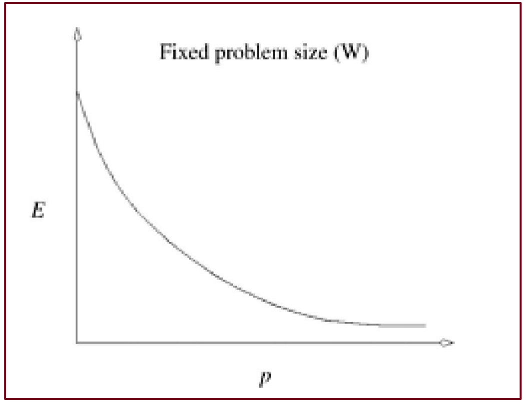
$$T_p^{min} = \Theta(W^{1/2}).$$

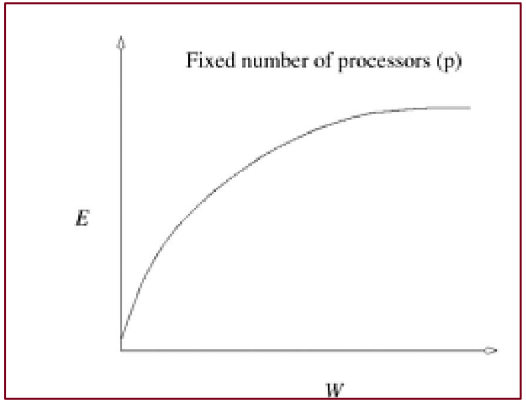
# Lecture Overview

- Review of Asymptotic Function Growth Rates (Big-O, $\boldsymbol{\Omega}$, $\boldsymbol{\Theta}$)
- Scalability and the Definition of Work ($W$)
- Isoefficiency
- Optimal Computation Time
- **Comparing Different Algorithms with Parallel Metrics**

# General Usefulness of these Metrics

- These metrics (E, S, isoefficiency, $T_o$, etc.) are useful tools for comparing different parallel approaches - we can determine which approaches are more scalable
- They expand on runtime analysis to show which algorithms will make ***better*** use of the processors we utilize, rather than just how much faster the algorithm itself will run



# General Usefulness of these Metrics

Consider the below set of algorithms

| Algorithm | A1 | A2 | A3 | A4 |
|---|---|---|---|---|
| $p$ | $n^2$ | $\log n$ | $n$ | $\sqrt{n}$ |
| $T_P$ | 1 | $n$ | $\sqrt{n}$ | $\sqrt{n}\log n$ |
| $S$ | $n \log n$ | $\log n$ | $\sqrt{n}\log n$ | $\sqrt{n}$ |
| $E$ | $\frac{\log n}{n}$ | 1 | $\frac{\log n}{\sqrt{n}}$ | 1 |
| $pT_P$ | $n^2$ | $n \log n$ | $n^{1.5}$ | $n \log n$ |

# General Usefulness of these Metrics

Which is fastest?

| Algorithm | A1 | A2 | A3 | A4 |
|---|---|---|---|---|
| $p$ | $n^2$ | $\log n$ | $n$ | $\sqrt{n}$ |
| $T_P$ | 1 | $n$ | $\sqrt{n}$ | $\sqrt{n}\log n$ |
| $S$ | $n \log n$ | $\log n$ | $\sqrt{n}\log n$ | $\sqrt{n}$ |
| $E$ | $\frac{\log n}{n}$ | 1 | $\frac{\log n}{\sqrt{n}}$ | 1 |
| $pT_P$ | $n^2$ | $n \log n$ | $n^{1.5}$ | $n \log n$ |

# General Usefulness of these Metrics

Which is fastest? (A1)

| Algorithm | A1 | A2 | A3 | A4 |
|---|---|---|---|---|
| $p$ | $n^2$ | $\log n$ | $n$ | $\sqrt{n}$ |
| $T_P$ | 1 | $n$ | $\sqrt{n}$ | $\sqrt{n}\log n$ |
| $S$ | $n \log n$ | $\log n$ | $\sqrt{n}\log n$ | $\sqrt{n}$ |
| $E$ | $\frac{\log n}{n}$ | 1 | $\frac{\log n}{\sqrt{n}}$ | 1 |
| $pT_P$ | $n^2$ | $n \log n$ | $n^{1.5}$ | $n \log n$ |

# General Usefulness of these Metrics

Should we use A1?

| Algorithm | A1 | A2 | A3 | A4 |
|---|---|---|---|---|
| $p$ | $n^2$ | $\log n$ | $n$ | $\sqrt{n}$ |
| $T_P$ | 1 | $n$ | $\sqrt{n}$ | $\sqrt{n} \log n$ |
| $S$ | $n \log n$ | $\log n$ | $\sqrt{n} \log n$ | $\sqrt{n}$ |
| $E$ | $\frac{\log n}{n}$ | 1 | $\frac{\log n}{\sqrt{n}}$ | 1 |
| $pT_P$ | $n^2$ | $n \log n$ | $n^{1.5}$ | $n \log n$ |

# General Usefulness of these Metrics

Should we use A1? No - requiring *p* processors is not practical

| Algorithm | A1 | A2 | A3 | A4 |
|---|---|---|---|---|
| $p$ | $n^2$ | $\log n$ | $n$ | $\sqrt{n}$ |
| $T_P$ | $1$ | $n$ | $\sqrt{n}$ | $\sqrt{n}\log n$ |
| $S$ | $n \log n$ | $\log n$ | $\sqrt{n}\log n$ | $\sqrt{n}$ |
| $E$ | $\frac{\log n}{n}$ | $1$ | $\frac{\log n}{\sqrt{n}}$ | $1$ |
| $pT_P$ | $n^2$ | $n \log n$ | $n^{1.5}$ | $n \log n$ |

# General Usefulness of these Metrics

Which are cost optimal?

| Algorithm | A1 | A2 | A3 | A4 |
|---|---|---|---|---|
| $p$ | $n^2$ | $\log n$ | $n$ | $\sqrt{n}$ |
| $T_P$ | 1 | $n$ | $\sqrt{n}$ | $\sqrt{n} \log n$ |
| $S$ | $n \log n$ | $\log n$ | $\sqrt{n} \log n$ | $\sqrt{n}$ |
| $E$ | $\frac{\log n}{n}$ | 1 | $\frac{\log n}{\sqrt{n}}$ | 1 |
| $pT_P$ | $n^2$ | $n \log n$ | $n^{1.5}$ | $n \log n$ |

# General Usefulness of these Metrics

Which are cost optimal? (A2, A4)

| Algorithm | A1 | A2 | A3 | A4 |
|---|---|---|---|---|
| $p$ | $n^2$ | $\log n$ | $n$ | $\sqrt{n}$ |
| $T_P$ | 1 | $n$ | $\sqrt{n}$ | $\sqrt{n}\log n$ |
| $S$ | $n \log n$ | $\log n$ | $\sqrt{n}\log n$ | $\sqrt{n}$ |
| $E$ | $\frac{\log n}{n}$ | 1 | $\frac{\log n}{\sqrt{n}}$ | 1 |
| $pT_P$ | $n^2$ | $n \log n$ | $n^{1.5}$ | $n \log n$ |

# General Usefulness of these Metrics

What should we use and why?

| Algorithm | A1 | A2 | A3 | A4 |
|---|---|---|---|---|
| $p$ | $n^2$ | $\log n$ | $n$ | $\sqrt{n}$ |
| $T_P$ | 1 | $n$ | $\sqrt{n}$ | $\sqrt{n}\log n$ |
| $S$ | $n \log n$ | $\log n$ | $\sqrt{n}\log n$ | $\sqrt{n}$ |
| $E$ | $\frac{\log n}{n}$ | 1 | $\frac{\log n}{\sqrt{n}}$ | 1 |
| $pT_P$ | $n^2$ | $n \log n$ | $n^{1.5}$ | $n \log n$ |