Lecture 14: Analytical Modeling of Parallel Programs, part 2

Concurrent and Multicore Programming

Department of Computer Science and Engineering
Yonghong Yan
yan@oakland.edu
www.secs.oakland.edu/~yan

Topic Overview

Review

- Scalability of Parallel Systems
  - Isoefficiency Metric of Scalability
- Minimum Execution Time and Minimum Cost-Optimal Execution Time
- Asymptotic Analysis of Parallel Programs
- Other Scalability Metrics
  - Scaled speedup, Serial fraction

Parallel Execution Time

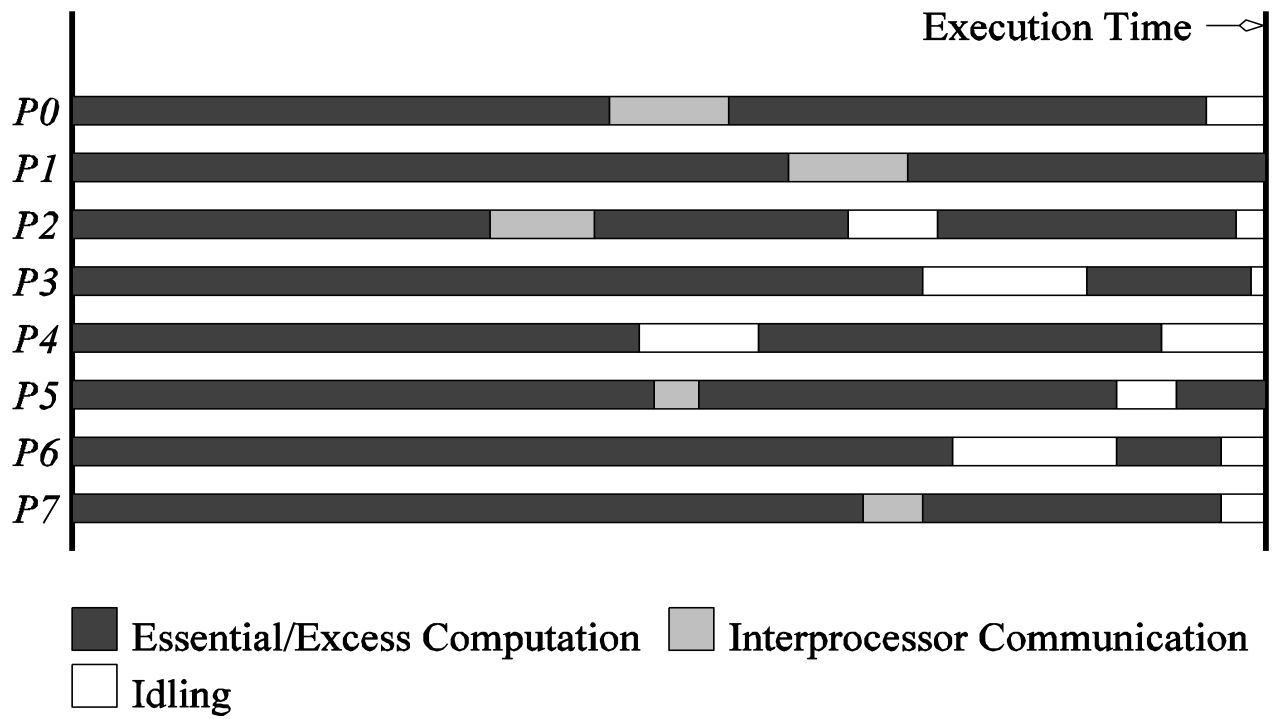
- Parallel execution time is a function of:
  - input size
  - **number of processors**
  - **communication parameters of target platform**

- Implications
  - must analyze parallel program for a particular target platform
    - communication characteristics can differ by more than O(1)
  - parallel program = parallel algorithm + platform

Overhead in Parallel Programs

If using two processors, shouldn't a program run twice as fast?

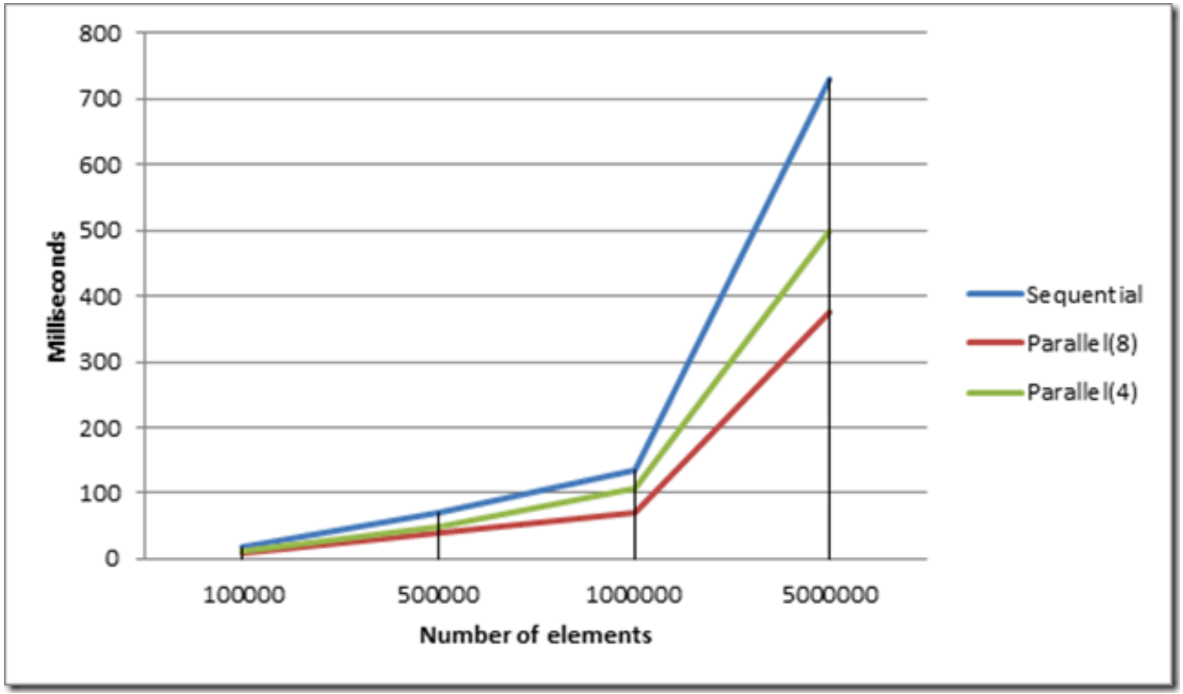
- Not all parts of the program are parallelized
- A number of overheads incurred when donig it in parallel



Performance Metrics: Execution Time

Does a parallel program run faster than its sequential version?

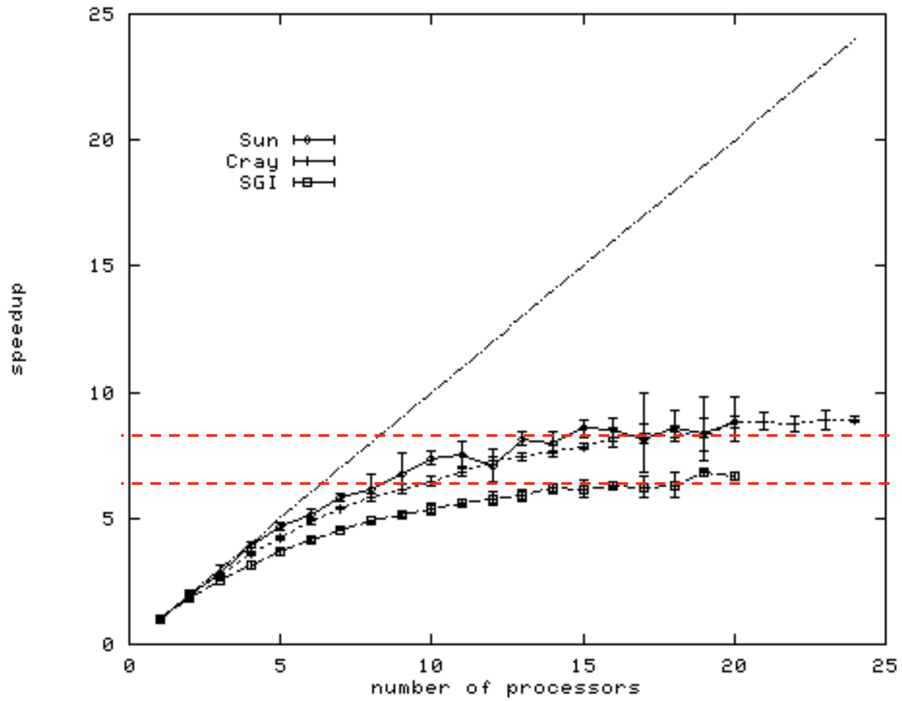
- Serial time: $\mathbf{T}_s$
  - time elapsed between the start and end of serial execution
- Parallel time: $\mathbf{T}_p$
  - time elapsed between first process start and last process end



Performance Metrics: Speedup

What is the benefit from increasing parallelism?

- Speedup (S): $\mathbf{T_S / T_P}$
  - The ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with p identical processing elements.



Performance Metrics: Efficiency

- Fraction of time for which a process perform useful work

$$E = S / p = T_S / (p\ T_P)$$

- Bounds
  - Theoretically, **$0 \leq E \leq 1$**
    - **The larger, the better**
    - **E=1: 0 overhead**
  - Practically, **E > 1 if superlinear speedup is achieved**

- Previous example: adding N numbers using N PEs
  - **Speedup:** $S = \Theta(N / \log N)$
  - **Efficiency:** $E = S/N = \Theta(N / \log N) / N = \Theta(1 / \log N)$
    - **Very low when N is big**

Performance Metrics: Cost

Product of parallel execution time and number of PEs: $p * T_P$

- The total amount of time by all PEs to solve the problem

- **Overhead: T_o**
  - $T_o = T_{all} - T_S$
  - $T_o = p\, T_P - T_S$

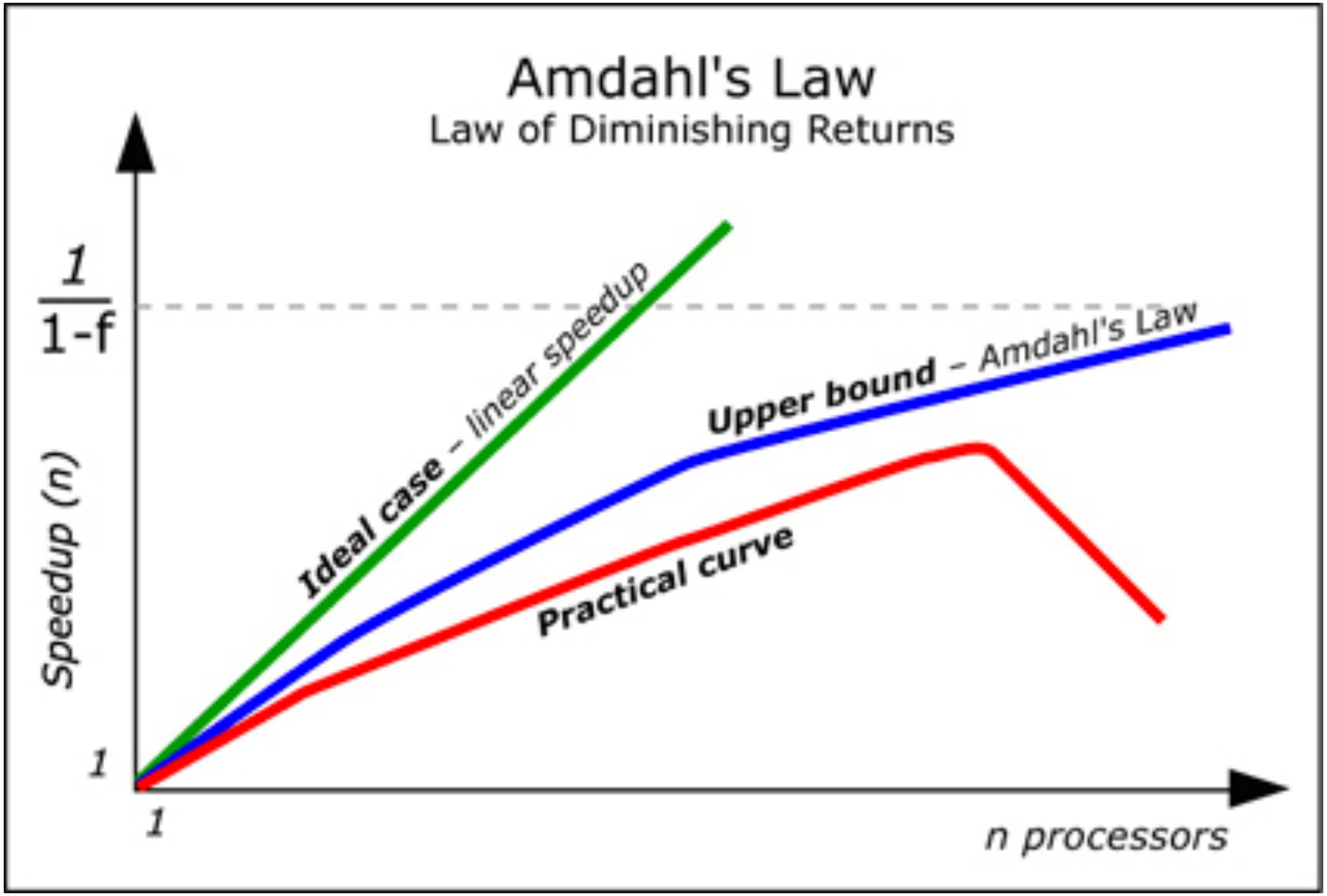
- ***Cost-optimal*** : parallel cost ≅ serial cost
  - ~0 overhead
  - **E = $\Theta(1)$, since E = $T_S / p * T_P$**

Topic Overview

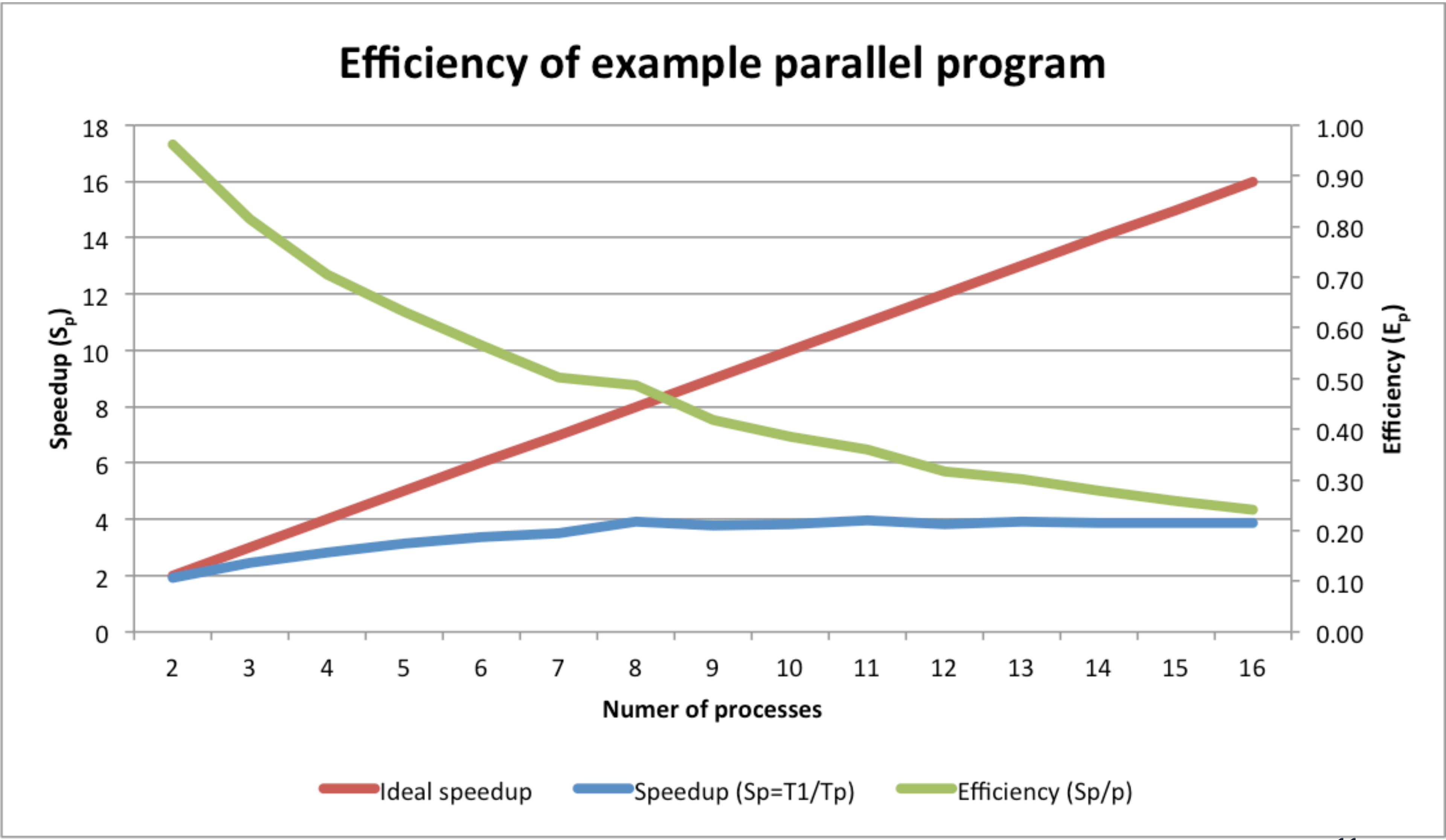
- Introduction
- Performance Metrics for Parallel Systems
  - Execution Time, Overhead, Speedup, Efficiency, Cost
- Amdahl's Law
- ☛ **Scalability of Parallel Systems**
  - **Isoefficiency Metric of Scalability**
- **Minimum Execution Time and Minimum Cost-Optimal Execution Time**
- **Asymptotic Analysis of Parallel Programs**
- **Other Scalability Metrics**
  - **Scaled speedup, Serial fraction**

Amdahl's Law Speedup

$$S(N) \leq \frac{1}{1-F}$$



Speedup and Efficiency



Scalability of Parallel Systems

- **Strong scaling:**
  - **Scales with same problem size**

- **Weak scaling**
  - **Scales with increased problem size**

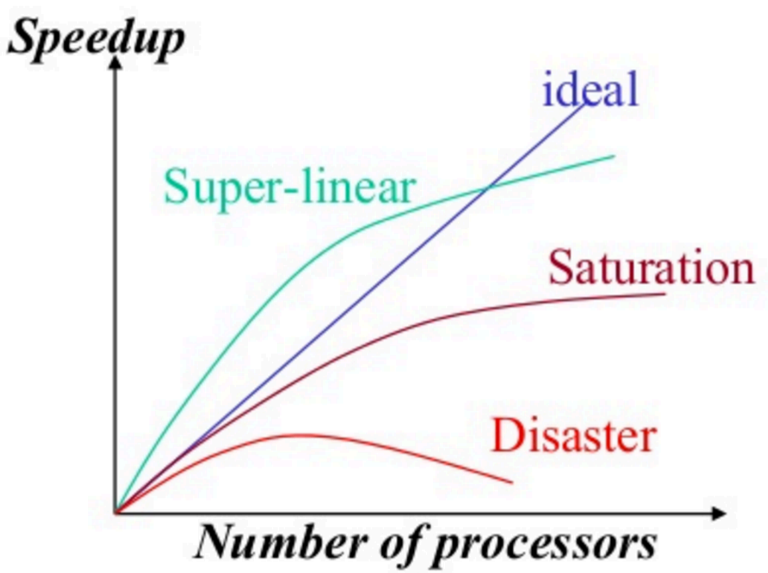
- http://www.mcs.anl.gov/~itf/dbpp/text/node30.html
- https://www.sharcnet.ca/help/index.php/Measuring_Parallel_Scaling_Performance

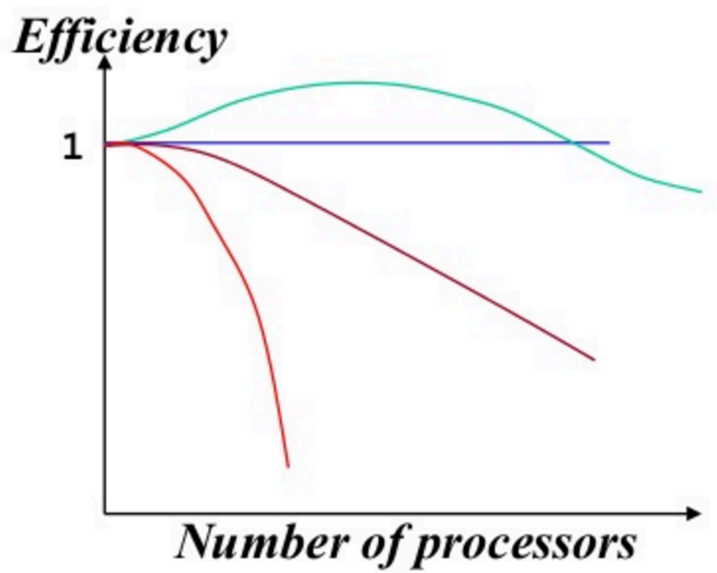
Strong Scaling

```
S(p) = T(1)/T(p)
E(p) = S(p)/p
```

for *ideal* parallel speedup we get:

```
T(p) = T(1)/p
S(p) = T(1)/T(p) = p
E(p) = S(p)/p = 1  or 100%
```



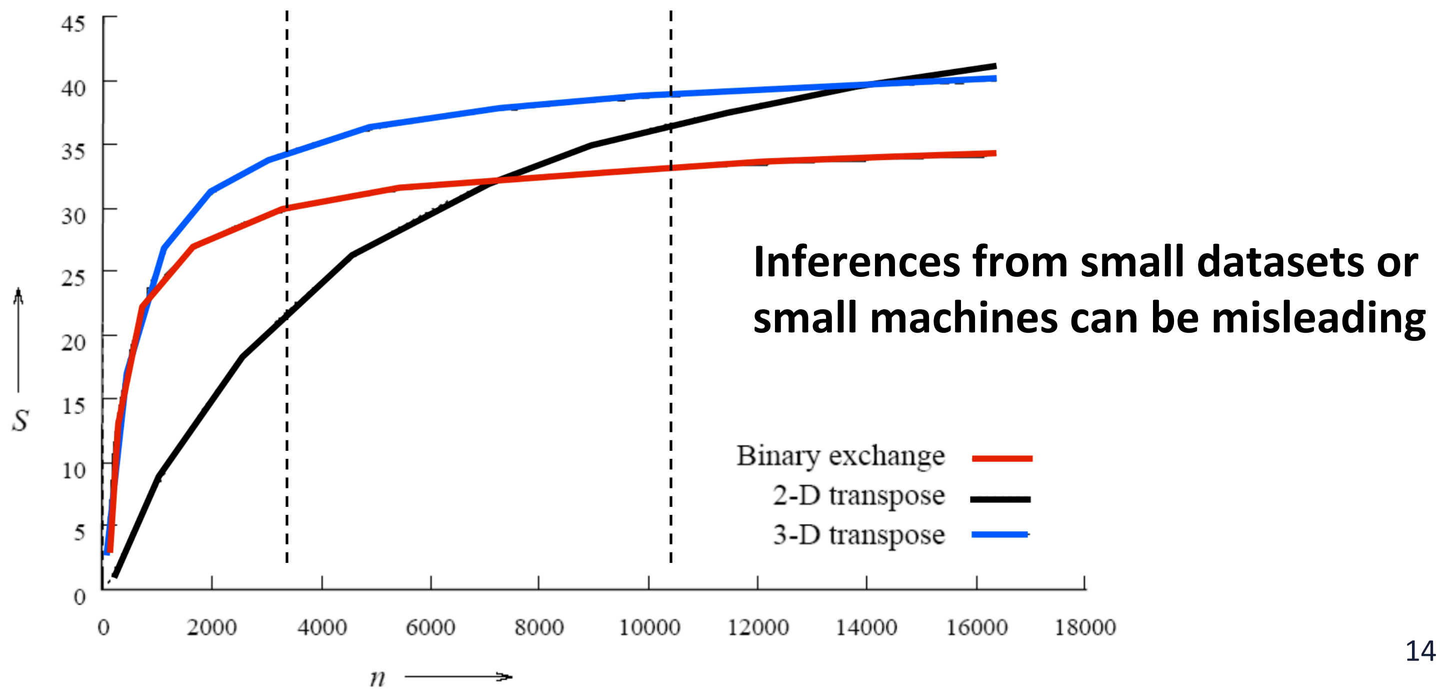


Weak Scalability of Parallel Systems

Extrapolate performance

- From small problems and small systems → larger problems on larger configurations

3 parallel algorithms for computing an n-point FFT on 64 PEs



Scaling Characteristics of Parallel Programs

- Efficiency: $E = \frac{S}{p} = \frac{T_S}{pT_P}$

- Parallel overhead: $T_o = p\,T_P - T_S$
  - **Overhead increases as *p* increase**

$$E = \frac{1}{1 + \frac{T_o}{T_S}}$$

- Problem size:
  - Given problem size, T_S remains constant

- **Efficiency increases if**
  - **The problem size increases and**
  - **Keeping the number of PEs constant.**

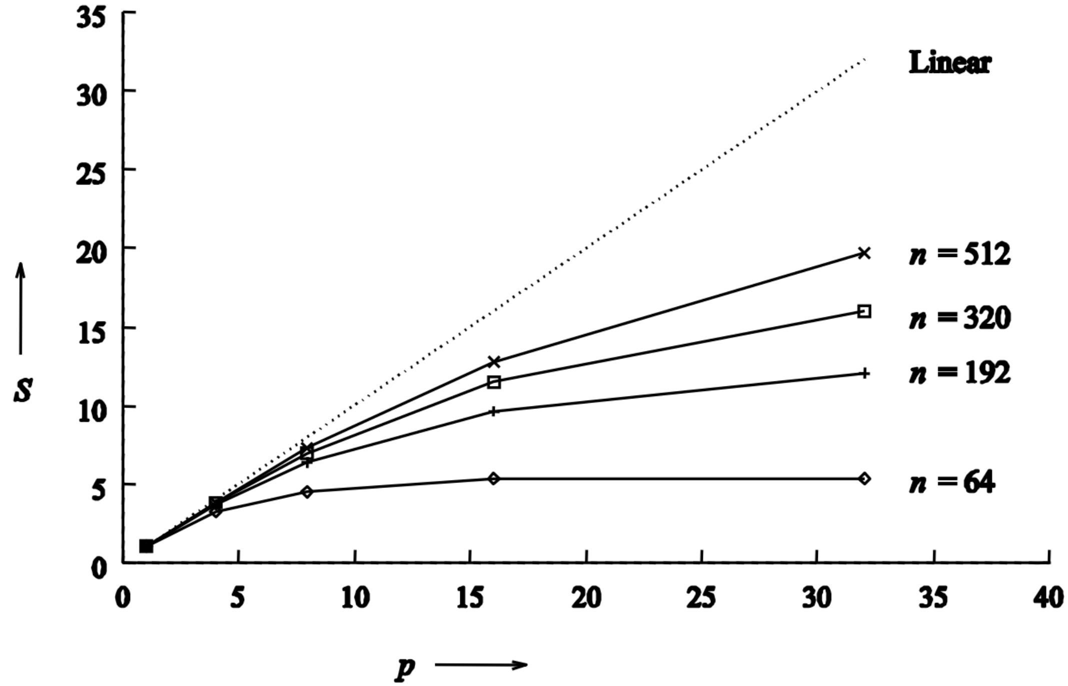
Example: Adding *n* Numbers on *p* PEs

- Addition = 1 time unit; communication = 1 time unit

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

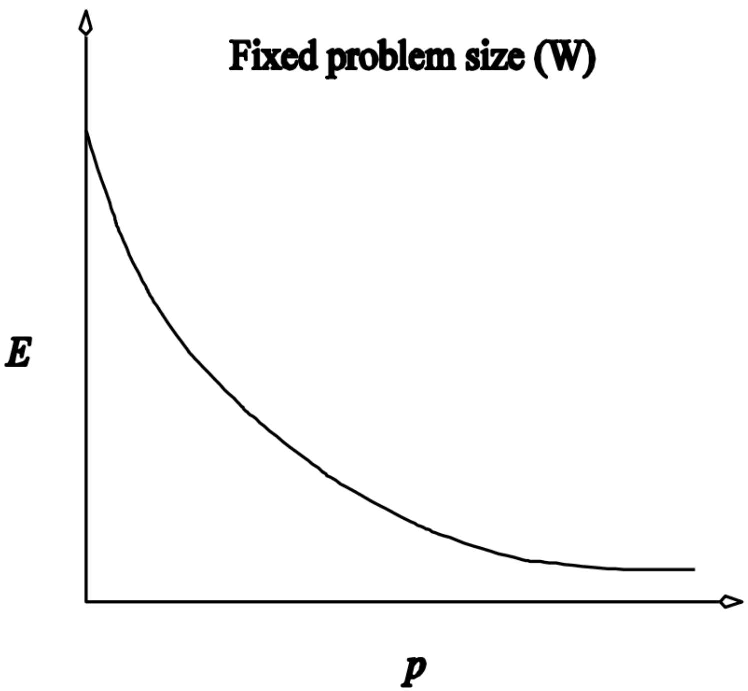
$$S = \frac{n}{\frac{n}{p} + 2\log p}$$

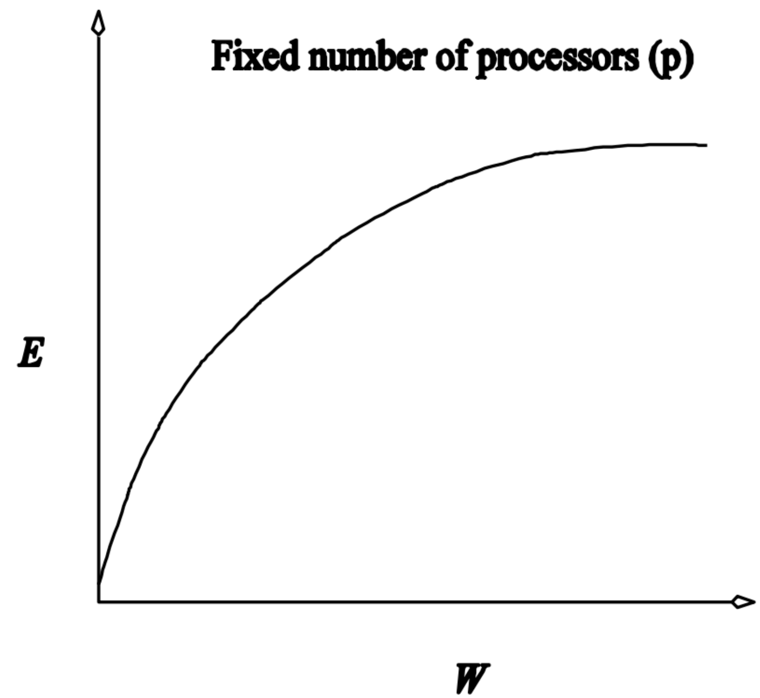
$$E = \frac{1}{1 + \frac{2p\log p}{n}}$$



Speedup tends to saturate and efficiency drops

Scaling and Efficiency





fixed problem size
 # PEs increasing

all parallel systems

problem size increasing
 # PEs fixed

scalable parallel systems

Scaling Characteristics of Parallel Programs

- Overhead $T_o = f(T_s, p)$, i.e. problem size and p
  - In many cases, T_o grows sublinearly with respect to T_s
- Efficiency:
  - Decreases as we increase p -> T_0
  - Increases as we increase problem size (Ts)

$$E = \frac{1}{1 + \frac{T_o}{T_S}}$$

- **Keep efficiency constant**
  - **Increase problem sizes and**
  - **proportionally increasing the number of PEs**
- ***Scalable*** **parallel systems**

Scalability vs Cost-Optimality

- To maintain constant efficiency $\Theta(1)$
  - Cost-optimal == $E = \Theta(1)$
- Any scalable parallel system can be made cost-optimal
  - Requires appropriate choice of
    - Size of the computation
    - Number of PEs

Isoefficiency Metric of Scalability

Rate at which the problem size (T_s) must increase per additional PE (T_0) to keep the efficiency fixed

$$E = \frac{1}{1 + \frac{T_o}{T_S}}$$

- The scalability of the system
  - The slower this rate, the better scalability
  - Rate == 0: strong scaling.
    - The same problem (same size) scales when increasing number of PEs
- To formalize this rate, we define
  - The problem size ***W*** = the asymptotic number of operations associated with the best serial algorithm to solve the problem.
    - The serial execution time, T_s

Isoefficiency Metric of Scalability

- Parallel overhead: $T_o(W,p)$
- Parallel execution time:

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

- Speedup:

$$\begin{aligned} S &= \frac{W}{T_P} \\ &= \frac{Wp}{W + T_o(W,p)}. \end{aligned}$$

- Efficiency

$$\begin{aligned} E &= \frac{S}{p} \\ &= \frac{W}{W + T_o(W,p)} \\ &= \frac{1}{1 + T_o(W,p)/W}. \end{aligned}$$

Isoefficiency Metric of Scalability

- **To maintain constant efficiency (between 0 and 1)**

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

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

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

- ***K*** = ***E*** / (1 – ***E***) is a constant related to the desired efficiency

$$W = KT_o(W,p).$$

Ratio T_o / *W* should be maintained at a constant value.

Isoefficiency Metric of Scalability

$$W = KT_o(W, p).$$

➡ **$W = \Phi(p)$** such that efficiency is constant

- **$W = \Phi(p)$** is called the ***isoefficiency function***
  - Read as: what is the problem size when we have ***p*** PEs to maintain constant efficiency?
  - **$W_{p+1} - W_p = \Phi(p+1) - \Phi(p)$**
    - **To maintain constant efficiency, how much to increase the problem size if adding one more PE?**

- ***isoefficiency function*** determines the ease
  - With which a parallel system maintain a constant efficiency
  - Hence achieve speedups increasing in proportion to # PEs

Isoefficiency Example 1

Adding *n* numbers using *p* PEs

- Parallel overhead: ***T_o = 2p log p***
- ***W = KT_0(W,p) ,*** substitute ***T_0***
  - **W = K *2*p*log p**
- ***K *2*p*log p is* the isoefficiency function**

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

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

$$E = \frac{1}{1 + \frac{2p\log p}{n}}$$

- The asymptotic isoefficiency function for this parallel system is ***Θ(p*log p)***
- **To have the same efficiency on p' processors as on p**
  - **problem size *n* must increase by (p' log p') / (p log p) when increasing PEs from p to p'**

Examples

- by (p’ log p’) / (p log p)

- If p = 8, p’ = 16
- 16*log16/(8*log8) = 16*4/(8*3) = 8/3 = 2.67

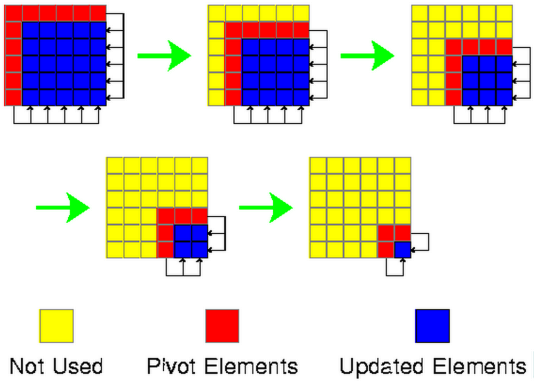
- 10M on 8 cores
- 10*2.67M on 16 cores

- A1*x + B1*y = C1 → A2*x + A2*(B1/A1)*y = A2*(C1/A1)
- A2*x + B2*y = C2

Isoefficiency Example 2

Add solve *n* linear equations on *p* processing elements

- **For Gaussian elimination, execution time =** $O(n^3/p + n^2 + n \log p)$
- **Total parallel work =** $O(n^3 + pn^2 + pn \log p)$
- **Overhead T_o is** $O(pn^2 + pn \log p)$
  - —*back substitution + pivot computation*



- **For isoefficiency, we want** $W = K\, T_o(W, p)$
- **Expressing overhead as a function of W = n^3 yields**

  $T_o = O(pW^{2/3} + pW^{1/3} \log p)$

- **Asymptotic isoefficiency** $W = K(pW^{2/3} + pW^{1/3} \log p)$
- **Want the same efficiency on p' processors as on p**
  - **—using first term W =** $KpW^{2/3} \rightarrow W = K^3p^3$
  - **—using second term W =** $KpW^{1/3} \log p \rightarrow W = K^{3/2} (p \log p)^{3/2}$
  - **—first term dominates: work must increase by** $(p')^3 / p^3$
    - **problem size n must increase by** p' / p

Cost-Optimality and Isoefficiency

- A parallel system is cost-optimal if and only if
  - Parallel cost == total work
    - Efficiency = 1

$$pT_P = \Theta(W).$$

- From this, we have:
  - i.e. work dominates overhead

$$W + T_o(W,p) = \Theta(W)$$

$$T_o(W,p) = O(W)$$

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

- If we have an isoefficiency function ***f***(***p***)
  - The relation $W = \Omega(f(p))$ must be satisfied to ensure the cost-optimality of a parallel system as it is scaled up

Lower Bound on the Isoefficiency Function

- For a problem consisting of W units of work
  - No more than W PEs can be used cost-optimally.
- To maintain fixed efficiency
  - The problem size must increase at least as fast as $\Theta(p)$
- Hence, $\Omega(p)$ is the asymptotic lower bound on the isoefficiency function
  - At least one additional computation item needs to be added to maintain constant efficiency

Degree of Concurrency and Isoefficiency

- Degree of concurrency
  - The maximum number of tasks that can be executed simultaneously at any time in a parallel algorithm
  - $C(W)$ is the degree of concurrency of a parallel algorithm
- For a problem of size W
  - No more than $C(W)$ processing elements can be employed effectively.

Degree of Concurrency and Isoefficiency: Example

Solving a system of equation using Gaussian elimination

- N variables, $W = \Theta(n^3)$
  - n variables must be eliminated one after the other
  - Eliminating each variable requires $\Theta(n^2)$ computations.
- At most $\Theta(n^2)$ PEs can be kept busy at any time.
- Since $W = \Theta(n^3)$, the degree of concurrency $C(W) = \Theta(W^{2/3})$
- Given p PEs
  - The problem size should be at least $\Omega(p^{3/2})$ to use them all.

Topic Overview

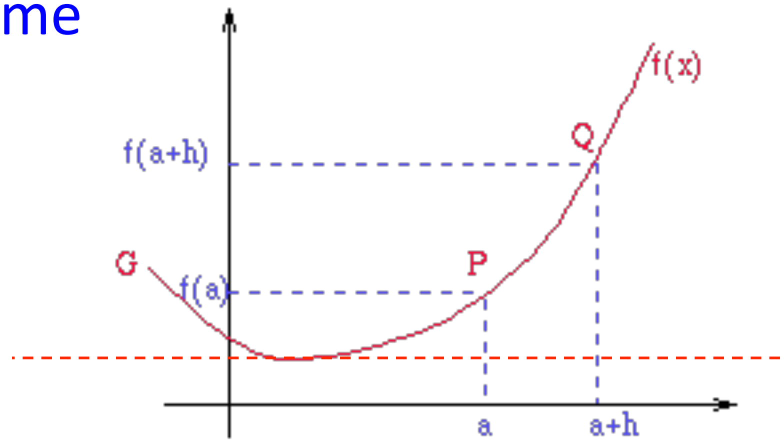
- Introduction
- Performance Metrics for Parallel Systems
  - Execution Time, Overhead, Speedup, Efficiency, Cost
- Amdahl's Law
- Scalability of Parallel Systems
  - Isoefficiency Metric of Scalability
- ☛ Minimum Execution Time
- Asymptotic Analysis of Parallel Programs
- Other Scalability Metrics
  - Scaled speedup, Serial fraction

Minimum Execution Time

- Often, we are interested in the minimum time to solution
- To determine the minimum exe time T_P^{min} for a given W
  - Differentiating the expression for T_P w.r.t. p and equate it to 0

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

- If p_0 is the value of p as determined by this equation
  - $T_P(p_0)$ is the minimum parallel time



Minimum Execution Time: Example

Adding n numbers

- Parallel execution time: $T_P = \frac{n}{p} + 2\log p.$
- Compute the derivative: $\frac{\partial}{\partial p}\left(\frac{n}{p} + 2\log p\right) = -\frac{n}{p^2} + 2\left(\frac{1}{p}\right)$
- Set the derivative = 0, solve for p: $-\frac{n}{p^2} + 2\left(\frac{1}{p}\right) = 0$
- The corresponding exe time:

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

$$p = n/2$$

$$T_P^{min} = 2\log n.$$

Note that at this point, the formulation is not cost-optimal.

Minimum Cost-Optimal Parallel Time

- The minimum cost-optimal parallel time: $T_P^{cost\_opt}$
- If the isoefficiency function of a parallel system is $\Theta(f(p))$
  - Then a problem of size W can be solved cost-optimally if and only if

$$W = \Omega(f(p))$$

- In other words, for cost optimality, $p = O(f^{-1}(W))$
- For cost-optimal systems, $T_P = \Theta(W/p)$, therefore,

$$T_P^{cost\_opt} = \Omega\left(\frac{W}{f^{-1}(W)}\right).$$

Minimum Cost-Optimal Parallel Time: Example

Adding n numbers

- The isoefficiency function $f(p)$ is $\Theta(p \log p)$.
- From this, we have $p \approx n / \log n$.
- At this processor count, the parallel runtime is:

$$T_P^{cost\_opt} = \log n + \log\left(\frac{n}{\log n}\right)$$
$$= 2\log n - \log\log n.$$

- Note that both T_P^{min} and $T_P^{cost\_opt}$ for adding n numbers are $\Theta(\log n)$. This may not always be the case.

Topic Overview

- Introduction
- Performance Metrics for Parallel Systems
  - Execution Time, Overhead, Speedup, Efficiency, Cost
- Amdahl's Law
- Scalability of Parallel Systems
  - Isoefficiency Metric of Scalability
- Minimum Execution Time
- ☛ Asymptotic Analysis of Parallel Programs
- Other Scalability Metrics
  - Scaled speedup, Serial fraction

Asymptotic Analysis of Parallel Programs

Sorting a list of *n* numbers.

- The fastest serial programs: Θ(***n* log *n***).
- Four parallel algorithms, A1, A2, A3, and 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	$n\log n$	$n^{1.5}$	$n\log n$

Asymptotic Analysis of Parallel Programs

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	$n\log n$	$n^{1.5}$	$n\log n$

- If metric is speed (T_p), algorithm A1 is the best, followed by A3, A4, and A2
- In terms of efficiency (E), A2 and A4 are the best, followed by A3 and A1.
- In terms of cost(pT_p), algorithms A2 and A4 are cost optimal, A1 and A3 are not.

- It is important to identify the analysis objectives and to use appropriate metrics!

Topic Overview

- Introduction
- Performance Metrics for Parallel Systems
  - Execution Time, Overhead, Speedup, Efficiency, Cost
- Amdahl's Law
- Scalability of Parallel Systems
  - Isoefficiency Metric of Scalability
- Minimum Execution Time
- Asymptotic Analysis of Parallel Programs
- ☛ Other Scalability Metrics
  - Scaled speedup, Serial fraction

Other Scalability Metrics

- A number of other metrics have been proposed, dictated by specific needs of applications.
  - For real-time applications, the objective is to scale up a system to accomplish a task in a specified time bound.
  - In memory constrained environments, metrics operate at the limit of memory and estimate performance under this problem growth rate.

Other Scalability Metrics: Scaled Speedup

- Speedup obtained when the problem size is increased linearly with the number of processing elements.
  - Per-PE problem size the same
- If scaled speedup is close to linear, the system is considered scalable.
  - Weak scaling
- If the isoefficiency is near linear, scaled speedup curve is close to linear as well.
- If the aggregate memory grows linearly in ***p***, scaled speedup increases problem size to fill memory.
- Alternately, the size of the problem is increased subject to an upper-bound on parallel execution time.

Scaled Speedup: Example

n x n matrix vector multiplication

- Serial execution time: $t_c n^2$
- Parallel Efficiency: $$S = \frac{t_c n^3}{t_c \frac{n^3}{p} + t_s \log p + 2 t_w \frac{n^2}{\sqrt{p}}}$$
- Total memory requirement of this algorithm is $\Theta(n^2)$.

Scaled Speedup: Example

Consider the case of memory-constrained scaling.

- We have m= Θ($\boldsymbol{n}^2$) = Θ($\boldsymbol{p}$).
- Memory constrained scaled speedup is given by

$$S' = \frac{t_c c \times p}{t_c \frac{c \times p}{p} + t_s \log p + t_w \sqrt{c \times p}}$$

$$S' = O(\sqrt{p})$$

- This is not a particularly scalable system

Scaled Speedup: Example (continued)

Consider the case of time-constrained scaling.

- We have $\boldsymbol{T_P} = O(\boldsymbol{n}^2)$.
- Since this is constrained to be constant, $\boldsymbol{n}^2 = O(\boldsymbol{p})$.

- Note that in this case, time-constrained speedup is identical to memory constrained speedup.
- This is not surprising, since the memory and time complexity of the operation are identical.
  - $O(\boldsymbol{n}^2)$

Scaled Speedup: Example

$n \times n$ matrix multiplication

- The serial execution time: $t_c n^3$.
- The parallel execution time: $$T_P = t_c \frac{n^3}{p} + t_s \log p + 2t_w \frac{n^2}{\sqrt{p}}$$
- Speedup: $$S = \frac{t_c n^3}{t_c \frac{n^3}{p} + t_s \log p + 2t_w \frac{n^2}{\sqrt{p}}}$$

Scaled Speedup: Example (continued)

Consider memory-constrained scaled speedup.

- We have memory complexity m= $\Theta(\boldsymbol{n}^2) = \Theta(\boldsymbol{p})$, or $\boldsymbol{n}^2 = \boldsymbol{c} \times \boldsymbol{p}$.
- At this growth rate, scaled speedup ***S'*** is given by:

$$S' = \frac{t_c (c \times p)^{1.5}}{t_c \frac{(c \times p)^{1.5}}{p} + t_s \log p + 2 t_w \frac{c \times p}{\sqrt{p}}} = O(p)$$

- Note that this is scalable.

Scaled Speedup: Example (continued)

Consider time-constrained scaled speedup.

- We have $T_p = O(1) = O(n^3 / p)$, or $n^3 = c \times p$.
- Time-constrained speedup ***S"*** is given by:

$$S'' = \frac{t_c c \times p}{t_c \frac{c \times p}{p} + t_s \log p + 2 t_w \frac{(c \times p)^{2/3}}{\sqrt{p}}} = O(p^{5/6})$$

- Memory constrained scaling yields better performance.

Serial Fraction *f*

- If a computation can be divided into a totally parallel and a totally serial component,, we have:

$$W = T_{ser} + T_{par}.$$

- From this, we have,

$$T_P = T_{ser} + \frac{T_{par}}{p}.$$

$$T_P = T_{ser} + \frac{W - T_{ser}}{p}$$

Serial Fraction *f*

- The serial fraction ***f*** of a parallel program is defined as:

$$f = \frac{T_{ser}}{W}.$$

- Therefore, we have:

$$T_P = f \times W + \frac{W - f \times W}{p}$$

$$\frac{T_P}{W} = f + \frac{1 - f}{p}$$

Serial Fraction

- Since $\boldsymbol{S = W / T_P}$, we have

$$\frac{1}{S} = f + \frac{1-f}{p}.$$

- From this, we have:

$$f = \frac{1/S - 1/p}{1 - 1/p}.$$

- If $\boldsymbol{f}$ increases with the number of processors, this is an indicator of rising overhead, and thus an indicator of poor scalability.

Serial Fraction: Example

Consider the problem of examining the serial component of the matrix-vector product.

$$f = \frac{\frac{t_c \frac{n^2}{p} + t_s \log p + t_w n}{t_c n^2}}{1 - 1/p}$$

We have:

$$f = \frac{t_s p \log p + t_w n p}{t_c n^2} \times \frac{1}{p - 1}$$

$$f \approx \frac{t_s \log p + t_w n}{t_c n^2}$$

Here, the denominator is the serial execution time and the numerator is the overhead.

References

- Adapted from slides "Principles of Parallel Algorithm Design" by Ananth Grama
- "Analytical Modeling of Parallel Systems", Chapter 5 in Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar, Introduction to Parallel Computing'', " Addison Wesley, 2003.
- Grama, Ananth Y.; Gupta, A.; Kumar, V., "Isoefficiency: measuring the scalability of parallel algorithms and architectures," in Parallel & Distributed Technology: Systems & Applications, IEEE , vol.1, no.3, pp.12-21, Aug. 1993, doi: 10.1109/88.242438, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=242438&isnumber=6234