Analytical Modeling of Parallel Performance

CS406/CS531
Week 6

Adapted from slides "Principles of Parallel Algorithm Design" by Ananth Grama and slides by John Mellor-Crummey

Topic Overview

- Performance metrics for parallel systems
- Scalability of parallel systems
- Cost optimality, scalability, efficiency
- Asymptotic isoefficiency
- Minimum execution time
- Other scalability metrics
  - —memory and time constrained scaling
  - —serial fraction
  - —Amdahl's law

Measuring Program Performance

- **Wall clock time**
  - —start time of the first process to end time of last process
  - —how does this scale?
    - when the number of processors is changed
    - when the program is ported to another machine
- **How much faster is the parallel version?**
  - —what do we compare with?
- **Operation counts, e.g. FLOPs**
  - —are these useful?

Asymptotic Execution Time

- **Sequential execution time**
  - —function of
    - input size
- **Parallel execution time**
  - —function of
    - input size
    - number of processors
    - communication parameters of target platform
- **Implications**
  - —must analyze parallel algorithms for a particular target platform
    - —communication characteristics can differ by more than O(1)
  - —parallel system = parallel algorithm + platform

Performance Metrics: Execution Time

- **Serial time: T_S**
  - **—time elapsed between the start and end of serial execution**
- **Parallel time: T_p**
  - **—time elapsed between first process start and last process end**

Total Parallel Overhead

- T_{all} =
  - —$\sum$ time spent collectively by each processor
    - – $p\ T_p$, where p is the number of processors
- Total parallel overhead: T_o
  - —time wasted by all processors combined
  - —$T_o = T_{all} - T_S$
  - —$T_o = pT_P - T_S$

Overheads in Parallel Programs

- **Extra computation**
  - —computation not performed by serial version
    - e.g., partially-replicated computation to reduce communication
- **Communication**
  - —data movement
- **Idling**
  - —load imbalance
  - —synchronization
  - —serialization
- **Contention**
  - —memory
  - —network

Performance Metrics: Speedup

- $S = T_S / T_P$

Example

- **Add n numbers using n processing elements**
- **If n is a power of two**
  - **can perform this operation in $\log n$ steps on n processors**
    - **propagate partial sums up a logical binary tree**

Performance analysis

- $T_S = \Theta(n)$
- **Assumptions**
  - **addition takes constant time t_c**
  - **communication of a single word takes time $t_s + t_w$**
- $T_P = \Theta(\log n)$
- **Speedup $S = \Theta(n / \log n)$**

A Note About T_s

- Might be many serial algorithms for a problem
- Different algorithms may have different asymptotic runtimes
- May be parallelizable to different degrees

Example: Sorting

Bubble sort

```
procedure bubbleSort( A : vector)
  n := length( A )
  do
    swapped := false
    n := n - 1
    for each i in 0 to n - 1
      if A[i] > A[i + 1]
        swap(A[i], A[i + 1]); swapped := true
  while (swapped)
end procedure
```

Odd-even sort

“parallel bubble sort”

```
procedure oddEvenSort( A : vector)
  n := length( A )
  do
    swapped := false
    for each i in 0 to n - 1 by 2 in parallel
      if A[i] > A[i + 1]
        swap(A[i], A[i + 1]); swapped := true
    for each i in 1 to n - 1 by 2 in parallel
      if A[i] > A[i + 1]
        swap(A[i], A[i + 1]); swapped := true
  while (swapped)
end procedure
```

Speedup of Odd-Even Parallel Sort

- Serial time for bubblesort: 150 seconds
- Odd-even parallel sort on 4 processors: 40 seconds
- Apparent speedup = 150/40 = 3.75
  - —is this a fair assessment?

- What if serial quicksort only took 30 seconds?
- Speedup of odd-even sort over quicksort = 30/40 = 0.75
  - —fairer assessment

Should consider the best serial program as the baseline for T_s

Performance Metrics: Speedup Bounds

- **Parallel program never terminates: speedup = 0**
- **Speedup > *p*?**
  - **—theoretically only if each processor spends less than time T_s / p**
  - **—but then single processor could be time sliced for < T_s**
    - **contradicts assumption of minimal T_s**
  - **—in practice: yes!**
  - **—examples**
    - **parallel version does less work than serial algorithm**
      - **searching for solution to n-queens problem**
        - **processor k could encounter a solution quickly**
    - **resource-based superlinearity**
      - **more processors: higher aggregate cache/memory bandwidth**
        - **improved cache-hit ratios can produce superlinearity**

Asymptotic Parallel Efficiency

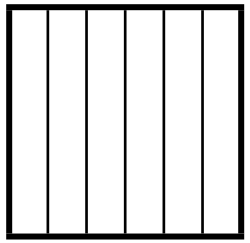
- **Fraction of time a processor performs useful work**
- $E = S / p = T_S / (p\ T_P)$
- **Bounds**
  - —**theoretically: $0 \leq E \leq 1$**
  - —**in practice: can have efficiency > 1 if superlinear speedup**
- **Previous example: add *n* numbers using *n* PEs**
  - —**speedup** $S = \Theta\ (n / \log n)$
  - —**efficiency** $E = S / n = \Theta\ (n / \log n) / n = \Theta\ (1 / \log n)$

Example: Edge Detection

- **Operation uses a *3* x *3* template to compute each pixel value**

-1	0	1
-2	0	2
-1	0	1

- **Serial time for an n x n image is given by $T_S = 9\ t_c n^2$**
- **Possible parallelization**
  - **partition image equally into vertical slabs, each with n^2 / p pixels**
  - **boundary of each slab is $2n$ pixels**
    - **number of pixel values that will have to be communicated**
  - **communication time = $2(t_s + t_w n)$**



- **Apply template to all n^2 / p pixels in time $T_S = 9\ t_c n^2 / p$**

Parallel Time, Speedup, and Efficiency

Edge detection (continued)

- **Parallel time:**

$$T_p = 9\,t_c n^2/p + 2(t_s + t_w n)$$

stencil cost (serial work)

communication cost (send n element msg to left and right neighbors)

- **Speedup:**

$$S = T_S / T_p$$

$$= 9\,t_c n^2 / (9\,t_c n^2/p + 2(t_s + t_w n))$$

- **Efficiency:**

$$E = S / p$$

$$= 9\,t_c n^2 / (p\,(9\,t_c n^2/p + 2(t_s + t_w n)))$$

$$= 1 / (1 + 2p(t_s + t_w n)/(9\,t_c n^2))$$

Cost Optimality

- **Cost of parallel system = pT_P**
  - **—sum of the work time for each processor**
  - **—*AKA work* or *processor-time product***
- **Parallel system is *cost-optimal* if**
  - **—Θ(solving a problem on a parallel computer) = Θ(serial)**
- **Since $E = T_S / pT_P$, for cost optimal systems $E = \Theta(1)$**

Considering Cost Optimality

Problem revisited: add *n* numbers

- **Is it cost-optimal on a parallel system using *n* PEs?**
- **As before,** T_P **=** $\log n$ for $p = n$
- **Cost of this system =** $p\ T_P = \Theta(n \log n)$
- **Serial runtime =** $\Theta(n)$
- **Algorithm is not cost optimal**
  - $E = \Theta(n / (n \log n)) = \Theta(1 / \log n)$

Impact of Non-Cost Optimality

- **Use n processing elements to sort a vector, time $(\log n)^2$**
  - **bitonic sort [Batcher 1968]**
- **Serial runtime of a comparison-based sort is $n \log n$**
- **Speedup = $n \log n \,/\, (\log n)^2 = n / \log n$**
- **pT_P product of this algorithm is $n (\log n)^2$**
  - **not cost optimal by a factor of $\log n$**
- **If $p < n$,**
  - **assign n tasks to p processors: $T_P = n (\log n)^2 / p$**
  - **speedup $= n \log n / (n (\log n)^2 / p) = p / \log n$**
  - **speedup $\downarrow$ as the problem size $n \uparrow$ for given p**
    - **significant cost associated with non-cost optimality**
    - **cost optimality is of practical importance**

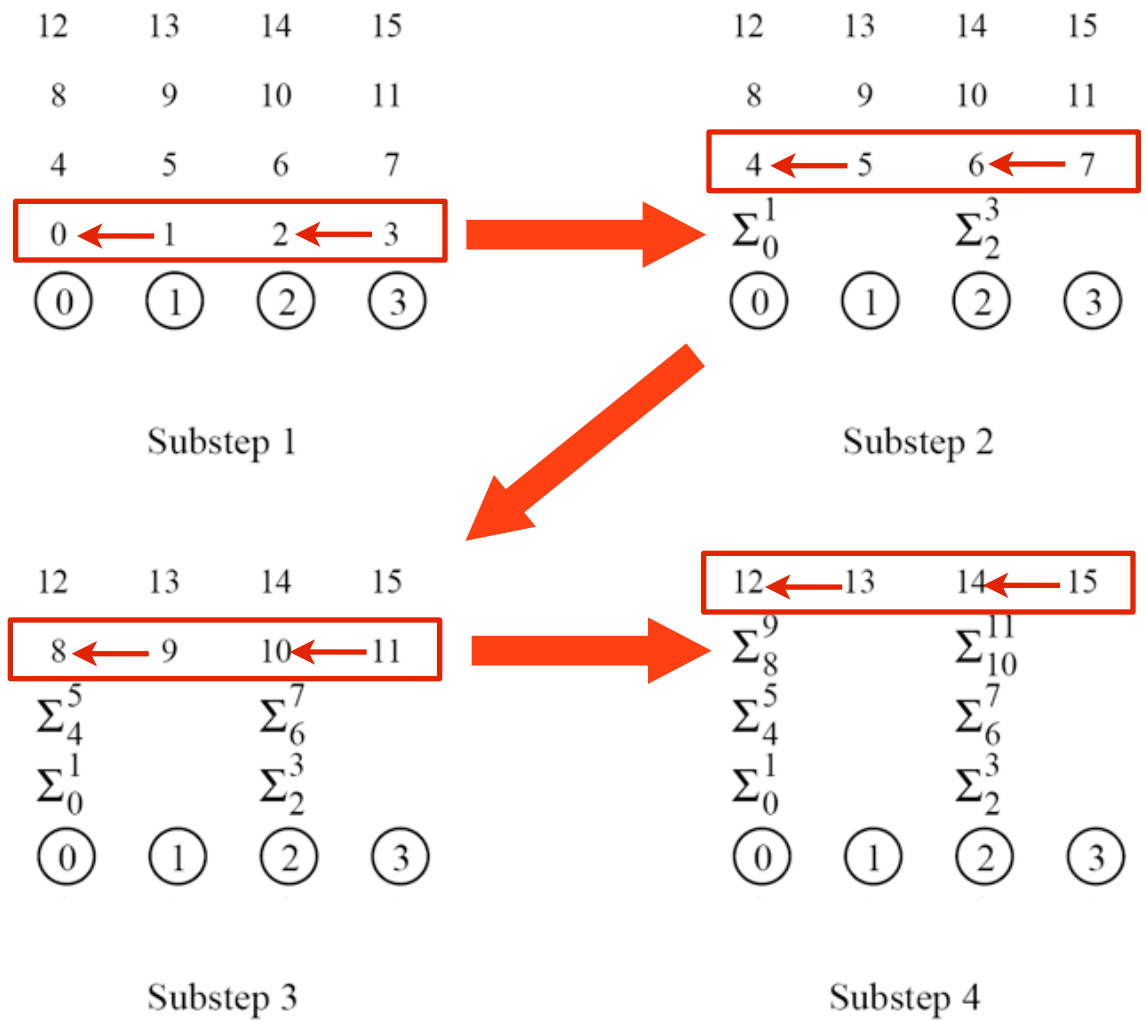
Effect of Granularity on Performance

- **Scaling down a parallel system**
  - —using fewer processors than the maximum possible
  - —usually improves parallel system efficiency
  - —naïve scaling down
    - consider each original processor as virtual processor
    - map virtual processors to scaled-down number of processors
- **Impact**
  - —# PE decreases by a factor of n / p
  - —computation for each PE increases by a factor of n / p
  - —communication cost depends upon what the VPs do

Sum on Virtual Processors

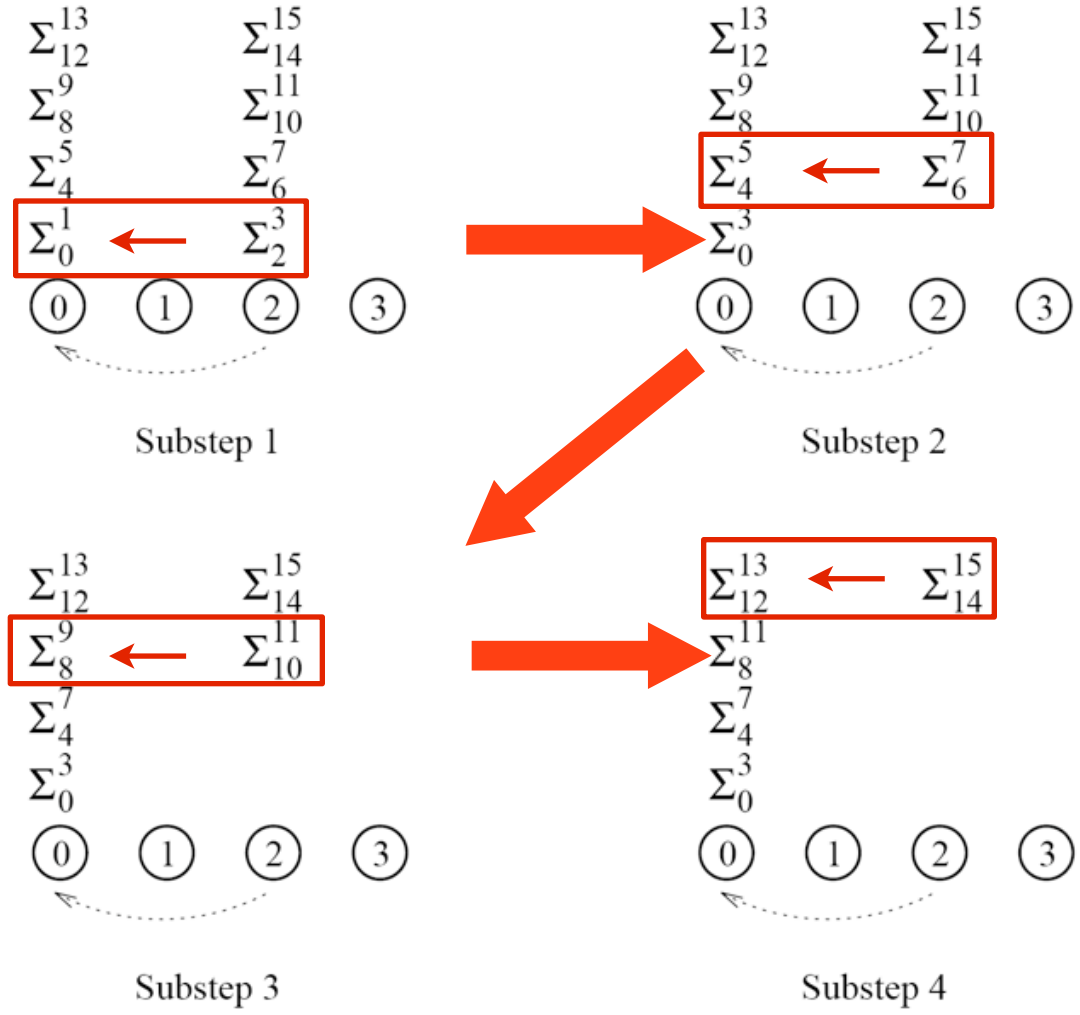
- **Add *n* numbers on *p* processing elements**
  - —***p* < *n***
  - —***n* and *p* are powers of 2**
- **Use parallel algorithm for *n* (virtual) processors**
  - **—assign each processor *n* / *p* virtual processors**

Sum Reduction on Virtual Processors



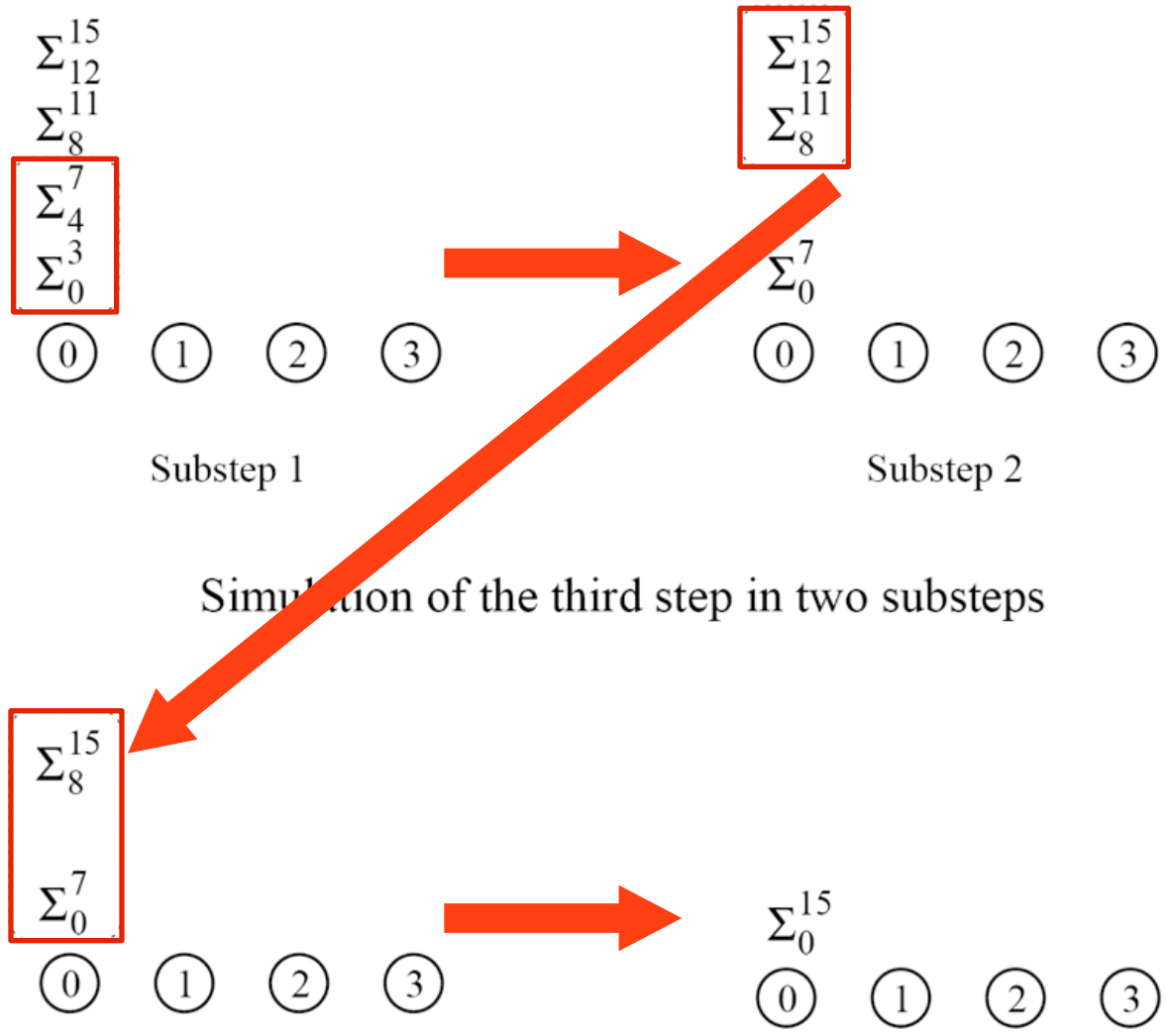
Four processors simulating the first communication step of 16 processors

Sum Reduction on Virtual Processors



Four processors simulating the second communication step of 16 processors

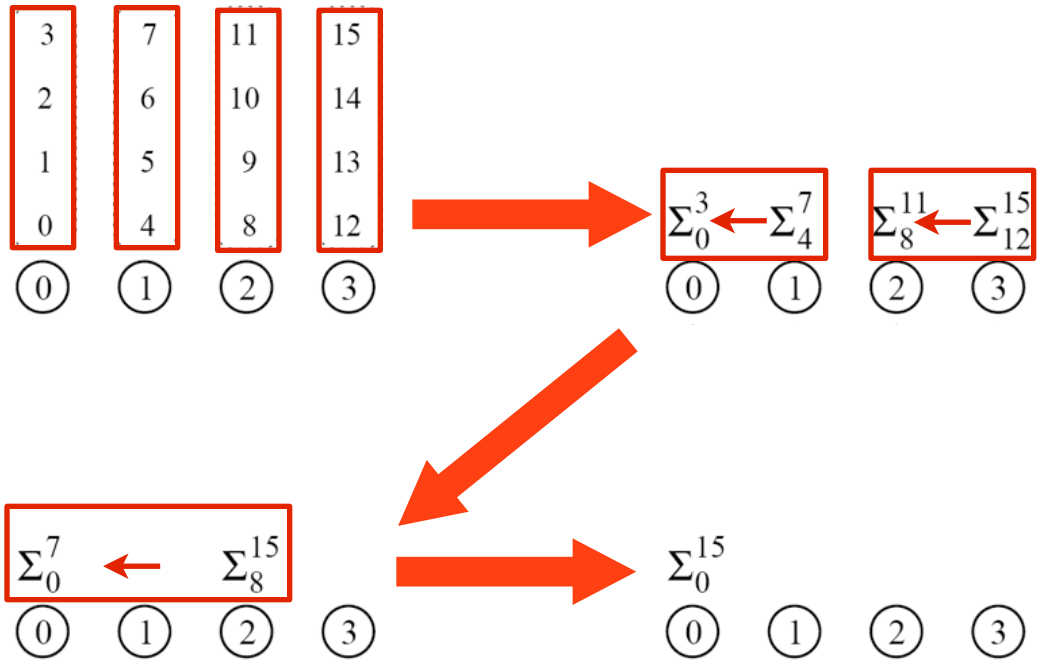
Sum Reduction on Virtual Processors



Sum Reduction on Virtual Processors

- **Execution cost**
  - —log p **of the** log n **steps of the original algorithm are simulated in** (n / p) log p **steps on** p **processing elements**
  - **—final log n/p steps: no communication needed**
- **Overall parallel execution time** $T_p = \Theta$ ((n / p) log p)
- **Total cost =** $p\ T_p = \Theta$ (n log p)
  - **—asymptotically >** Θ (n) **cost of adding** n **numbers sequentially**
  - **—thus, the parallel system is not cost-optimal**

Summing Cost Optimally

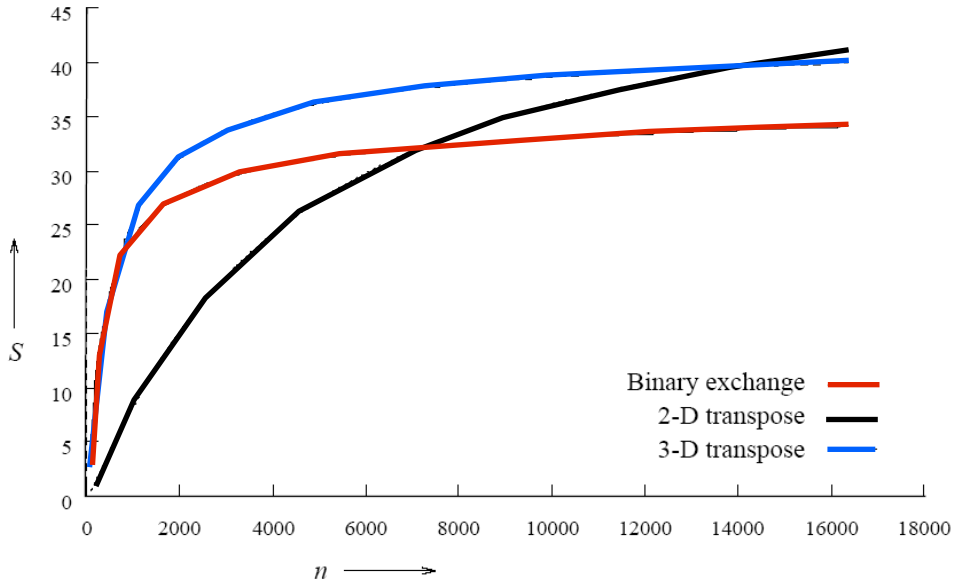


Scalability of Parallel Systems

Extrapolating performance

small problems & systems → larger problems on larger configurations

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



Inferences from small datasets or small machines can be misleading

Scalable Parallel Systems

- **Total overhead function** $T_o(T_s, p)$
  - **best serial execution time** T_s
  - **number of processors** p

recall that $T_o = pT_p - T_s$

- **Efficiency** $E = T_s / pT_p = T_s / (T_o + T_s) = 1 / (1 + T_o / T_s)$
- **Often,** $T_o(T_s, p) / T_s < 1$ (T_o **grows sublinearly w.r.t.** T_s)
  - **in this case, efficiency increases if**
    - **problem size is increased**
    - **number of processors is constant**
- **For such systems, can keep efficiency constant by**
  - **increasing the problem size**
  - **proportionally increasing the number of processors**
- **Such systems: *scalable* parallel systems**

Scaling Characteristics of Parallel Programs

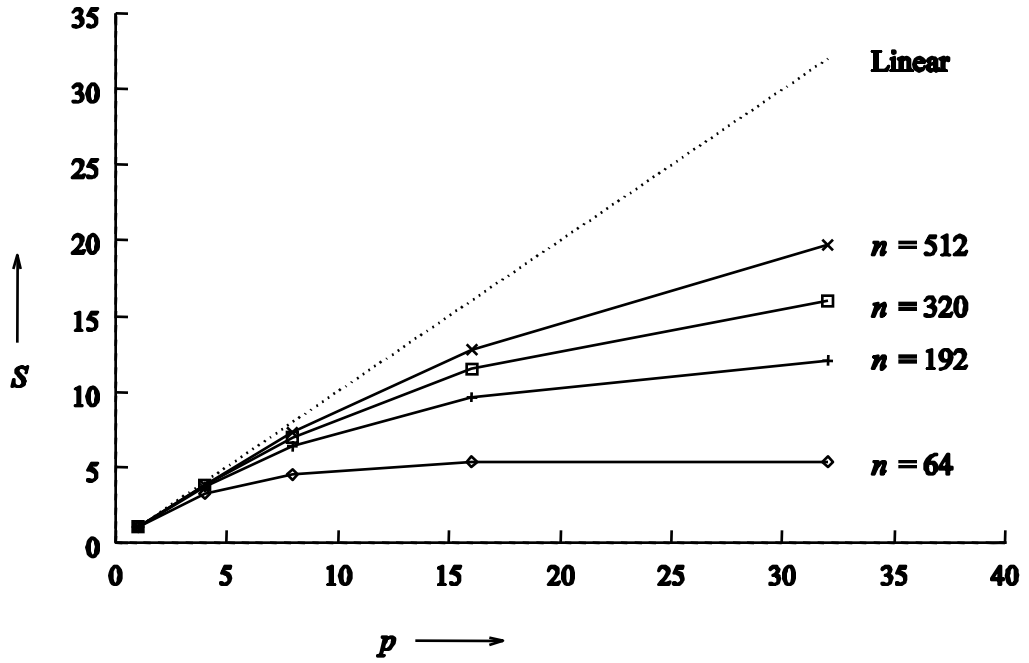
Adding *n* numbers on *p* processors

Assumption: addition = 1 time unit; communication = 1 time unit

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

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

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



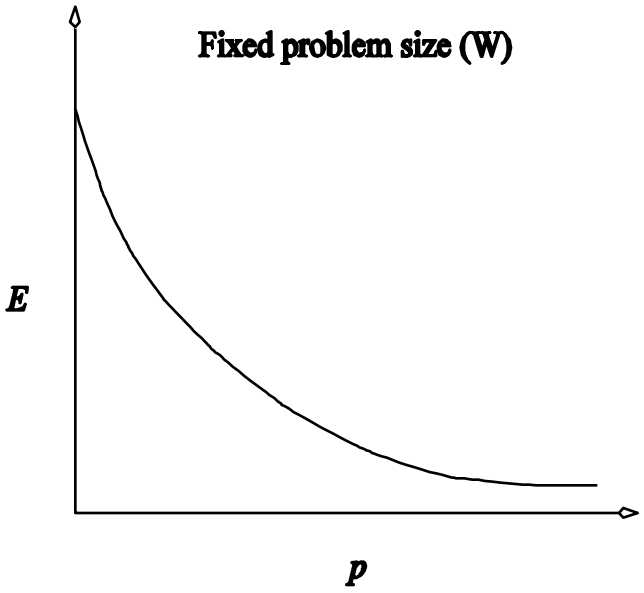
Speedup tends to saturate and efficiency drops off

Scaling Characteristics of Parallel Pgms

Scalability and cost-optimality are related

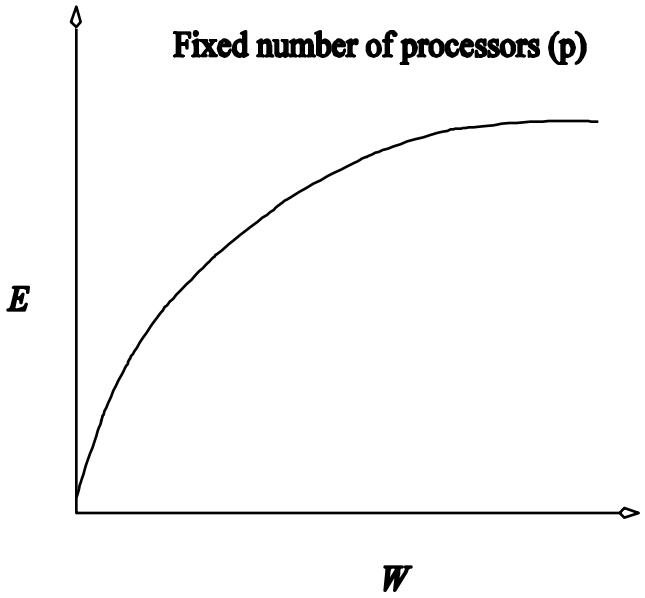
- **In cost-optimal parallel systems, efficiency = $\Theta(1)$**
- **Any scalable parallel system can be made cost-optimal**
  - **—requires appropriate choice of**
    - **size of the computation**
    - **number of processors**

Scaling and Efficiency



fixed problem size
 # PEs increasing

all parallel systems



problem size increasing
 # PEs fixed

scalable parallel systems

Isoefficiency

Rate at which problem size must increase per additional processor to keep efficiency fixed

- **Determines the ease with which a parallel system**
  - **—can maintain a constant efficiency**
  - **—achieve speedups increasing in proportion to # PEs**
  - **—the lower the rate, the better**

- **To formalize this rate, define**

  problem size *W* = asymptotic # of operations for best serial algorithm to solve the problem

Scalability Metrics as Functions of *W*

Parallel execution time

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

Speedup

$$S = \frac{W}{T_P}$$

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

Efficiency

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

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

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

Isoefficiency for Scalable Parallel Systems

- **To maintain efficiency at a fixed value (between 0 and 1)**
  - **maintain ratio $T_o(W,p) / W$ a constant value**

- **For a desired value E of efficiency**

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

- **Let $K = E / (1 - E)$ be a constant related to the desired efficiency**
- **Since T_o is a function of W and p, we have**

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

Asymptotic Isoefficiency (Example 1)

Add *n* numbers on *p* processing elements

- **Overhead T_o is approximately** $2p \log p$
  - —*log p* **levels in tree; communication and addition @ each**
- **For isoefficiency, we want** $W = K T_o(W,p)$
- **Substituting T_o by** $2p \log p$ **, we get** $W = K\, 2p \log p$
- **This yields asymptotic isoefficiency** $\theta(p \log p)$.
- **Want the same efficiency on *p'* processors as on *p***
  - —**as the # PEs increases from *p* to *p'***
  - —**problem size n must increase by ($p' \log p'$) / ($p \log p$)**

Isoefficiency for Gaussian Elimination

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 = n³ 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**

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

- **From this, we have:**

$$\begin{aligned} W + T_o(W,p) &= \Theta(W) \\ T_o(W,p) &= O(W) \\ W &= \Omega(T_o(W,p)) \end{aligned}$$

- **If we have an isoefficiency function *f*(*p*),**
  - **— *W* = Ω(*f*(*p*)) must be satisfied to ensure cost-optimality of a parallel system as it is scaled up**

Lower Bound on Isoefficiency

- **For a problem consisting of *W* units of work**
  - **—no more than *W* processing elements can be used cost-optimally**

- **To maintain a fixed efficiency**
  - **—problem size must increase at least as fast as *Θ(p)***
  - **—*Ω(p)* is the asymptotic lower bound on the isoefficiency function**

Degree of Concurrency

- **Definition: degree of concurrency**
  - **—max # tasks that can execute simultaneously at *any* time in a parallel algorithm**
- **Let $C(W)$ be the degree of concurrency of a parallel algorithm**
- **For a problem of size W, no more than $C(W)$ PEs will be useful**

Degree of Concurrency Example

Solving a system of equations using Gaussian elimination

- $W = \Theta(n^3)$
  - —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** $C(W)$ **is** $\Theta(W^{2/3})$
- **Given** p **processing elements,**
  - —**problem size should be at least** $\Omega(p^{3/2})$ **to use them all**

Minimum Execution Time

- **Often, interested in the minimum time to solution**
- **Determine the min parallel runtime T_P^{min} for a given W**
  - **differentiate the expression for T_P w.r.t. p and equate it to zero**

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

- **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

Consider the minimum execution time for 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** $-\frac{n}{p^2} + 2\left(\frac{1}{p}\right) = 0$

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

$$p = n/2$$

- **The corresponding runtime is**

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

Parallel & Serial Run Times

- **Serial runtime can be divided two components**
  - **—a totally serial component**
  - **—a parallelizable component**

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

- **From this, we have,**

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

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

Serial Fraction

- **Serial fraction *f* of a parallel program**

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

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

- **Therefore, we have:**

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

Serial Fraction

- **Since** $S = W / T_P$ **, we have**

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

- **From this, we have:**

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

Amdahl's Law

Law of diminishing returns

- **Considers overall speedup of a parallel program**
- **Program = 2 parts:**
  - **serial fraction *f*, parallel fraction *1-f***
  - $T_p(p) = fT_s + (1-f)T_s/p$
- **Amdahl's law for program speedup**

$$\begin{aligned} S(p) &= T_s / T_p(p) \\ &= T_s / (fT_s + (1-f)T_s / p) \\ &= 1/(f + (1-f)/p) \end{aligned}$$

$$\lim_{p \to \infty} \frac{1}{f + (1-f)/p} = \frac{1}{f}$$

- **Once (1 – *f*) / *p* is small compared to *f***
  - **price/performance ratio falls rapidly as *p* is increased.**

Gene Amdahl, "Validity of the single processor approach to achieving large-scale computing capabilities", *AFIPS Conference Proceedings*, 30:483-485, 1967.

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.
- http://en.wikipedia.org/wiki/Bubble_sort