Analytical Modeling of Parallel Programs (Chapter 5)

Alexandre David
B2-206

Topic Overview

- Sources of overhead in parallel programs.
- Performance metrics for parallel systems.
- Effect of granularity on performance.
- Scalability of parallel systems.
- Minimum execution time and minimum cost-optimal execution time.
- Asymptotic analysis of parallel programs.
- Other scalability metrics.

Analytical Modeling – Basics

- A sequential algorithm is evaluated by its runtime in function of its input size.
  - $O(f(n))$, $\Omega(f(n))$, $\Theta(f(n))$.
- The asymptotic runtime is independent of the platform. Analysis "at a constant factor".
- A parallel algorithm has more parameters.
  - Which ones?

Analytical Modeling – Basics

- A parallel algorithm is evaluated by its runtime in function of
  - the input size,
  - the number of processors,
  - the communication parameters.
- Which performance measures?
- Compare to which (serial version) baseline?

Sources of Overhead in Parallel Programs

- Overheads: wasted computation, communication, idling, contention.
  - Inter-process interaction.
  - Load imbalance.
  - Dependencies.

Performance Metrics for Parallel Systems

- Execution time = time elapsed between
  - beginning and end of execution on a sequential computer.
  - beginning of first processor and end of the last processor on a parallel computer.

Performance Metrics for Parallel Systems

- Total parallel overhead.
  - Total time collectively spent by all processing elements = pT_P.
  - Time spent doing useful work (serial time) = T_S.
  - Overhead function: $T_O = pT_P - T_S$.

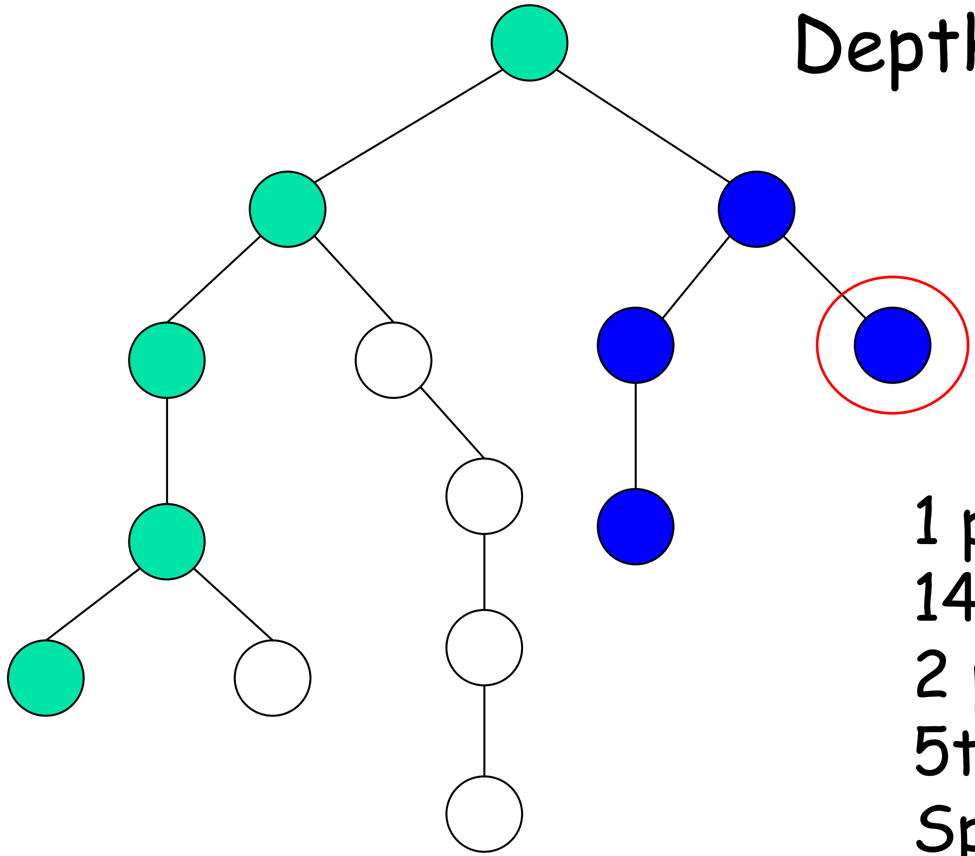
Performance Metrics for Parallel Systems

- What is the benefit of parallelism?
  - Speedup of course… let's define it.
- Speedup $S = T_S/T_P$.
- Example: Compute the sum of n elements.
  - Serial algorithm $\Theta(n)$.
  - Parallel algorithm $\Theta(\log n)$.
  - Speedup = $\Theta(n/\log n)$.
- Baseline (T_S) is for the best sequential algorithm available.

Speedup

- Theoretically, speedup can never exceed *p*. If > *p*, then you found a better sequential algorithm… Best: $T_P=T_S/p$.
- In practice, super-linear speedup is observed. How?
  - Serial algorithm does more work?
  - Effects from caches.
  - Exploratory decompositions.

Speedup – Example



Depth-first Search

1 processing element: $14t_c$.
2 processing elements: $5t_c$.
Speedup: 2.8.

Performance Metrics

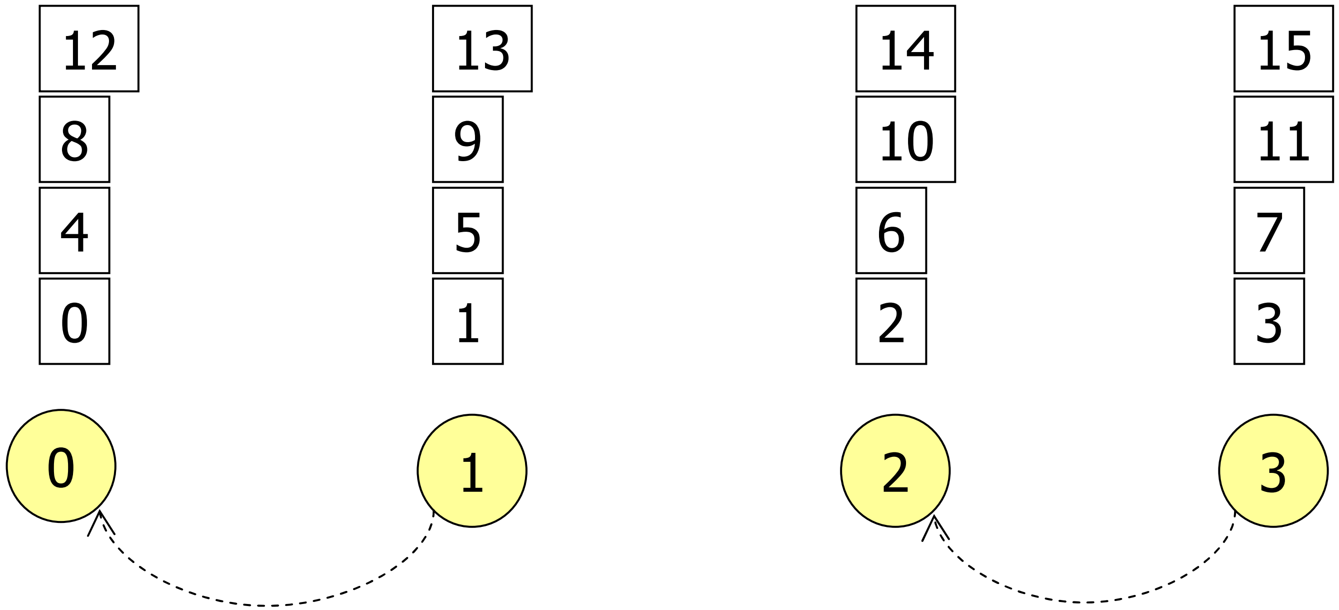
- Efficiency $E=S/p$.
  - Measure time spent in doing useful work.
  - Previous sum example: $E = \Theta(1/\log n)$.
- Cost $C=pT_P$.
  - A.k.a. work or processor-time product.
  - Note: $E=T_S/C$.
  - Cost optimal if E is a constant.

Effect of Granularity on Performance

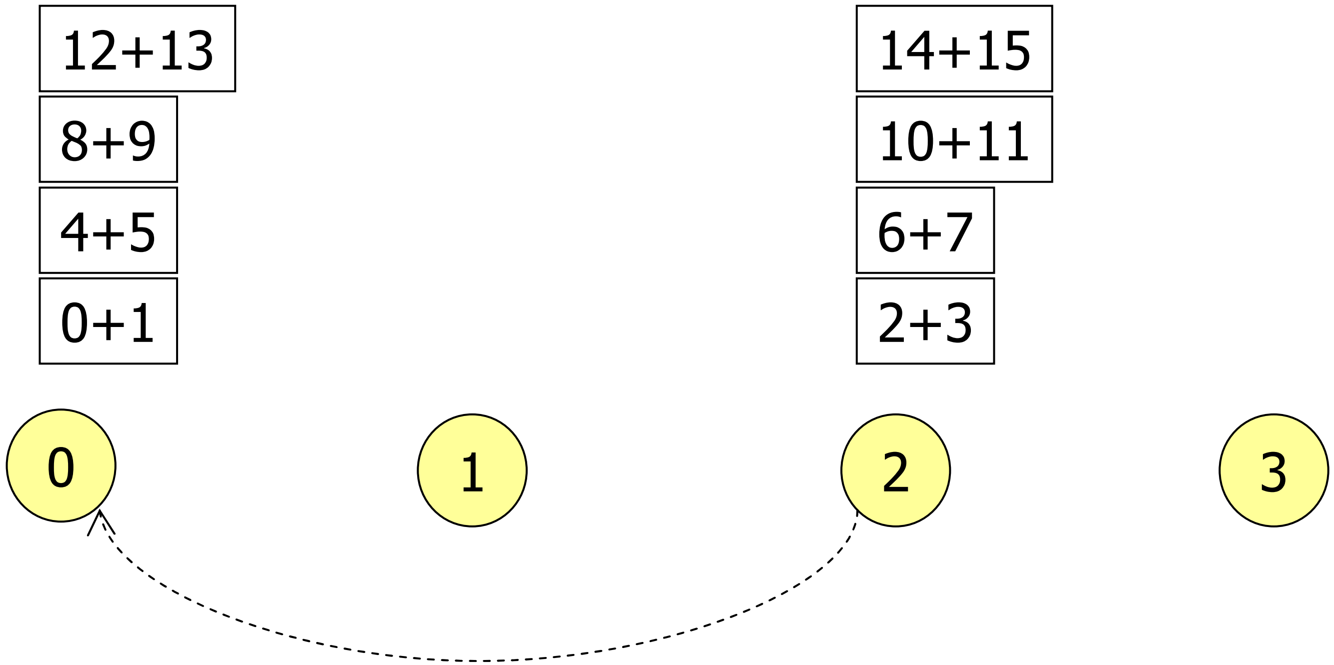
- Scaling down: To use fewer processing elements than the maximum possible.
- Naïve way to scale down:
  - Assign the work of n/p processing element to every processing element.
    - Computation increases by n/p.
    - Communication growth $\leq n/p$.
  - If a parallel system with n processing elements is cost optimal, then it is still cost optimal with p.

If it is not cost optimal, it may still not be cost optimal after the granularity increase.

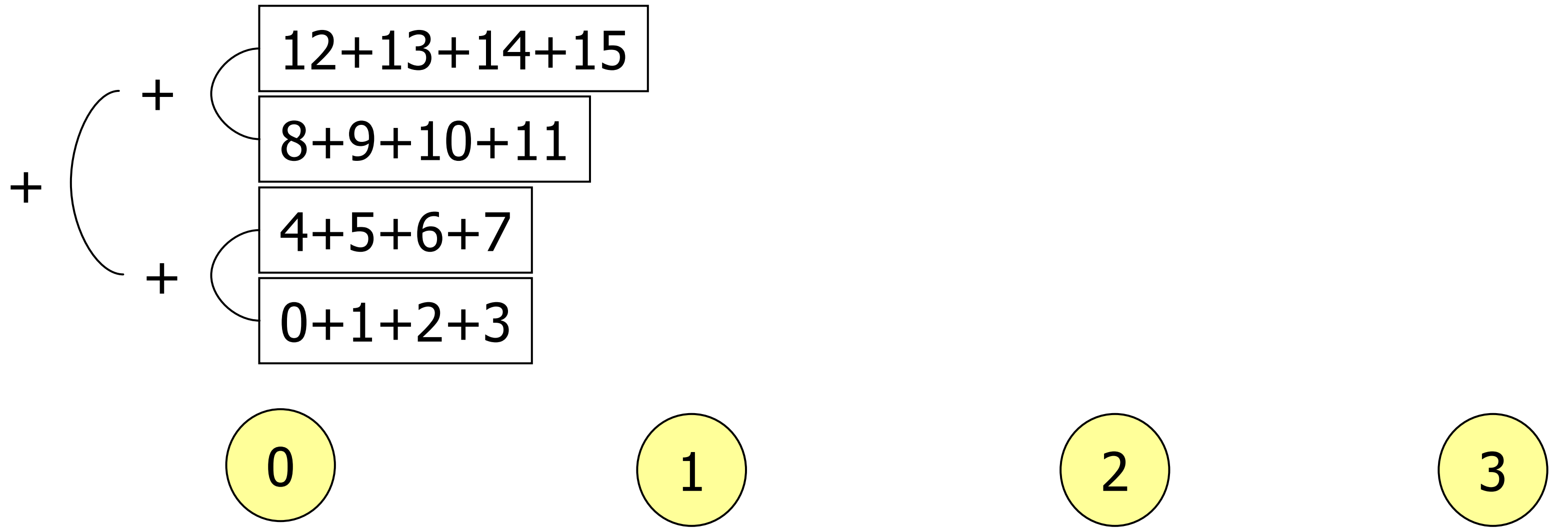
Adding n Numbers – Bad Way



Adding n Numbers – Bad Way

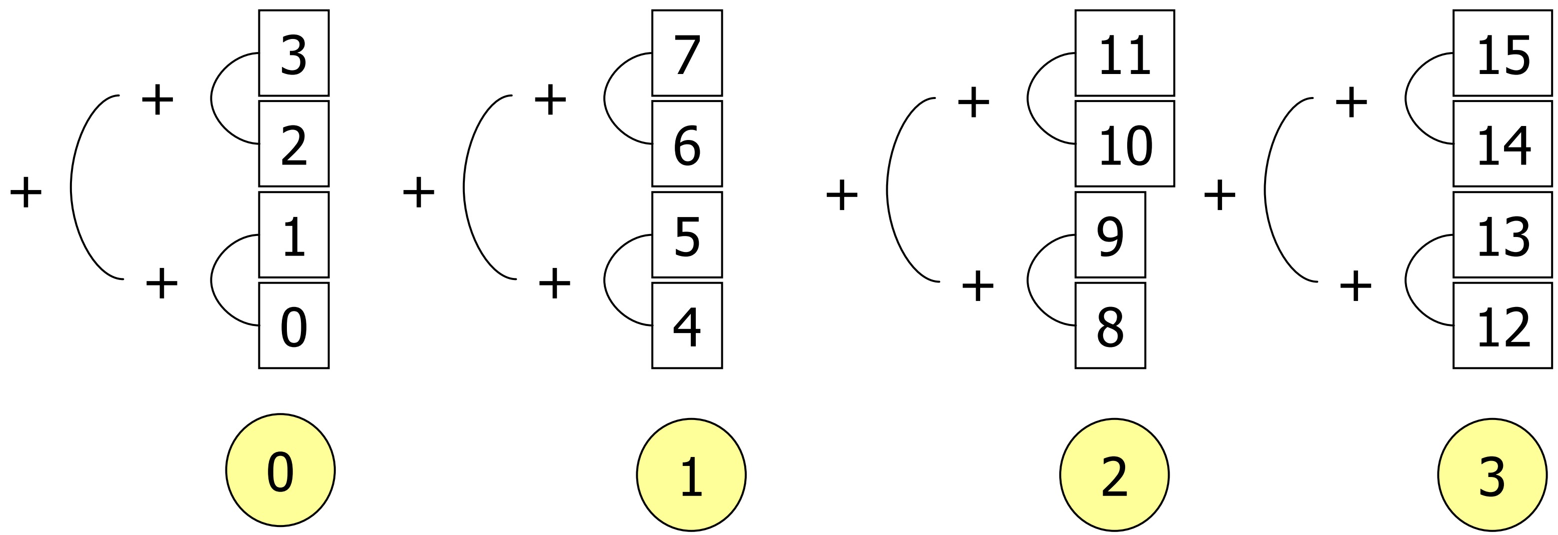


Adding n Numbers – Bad Way

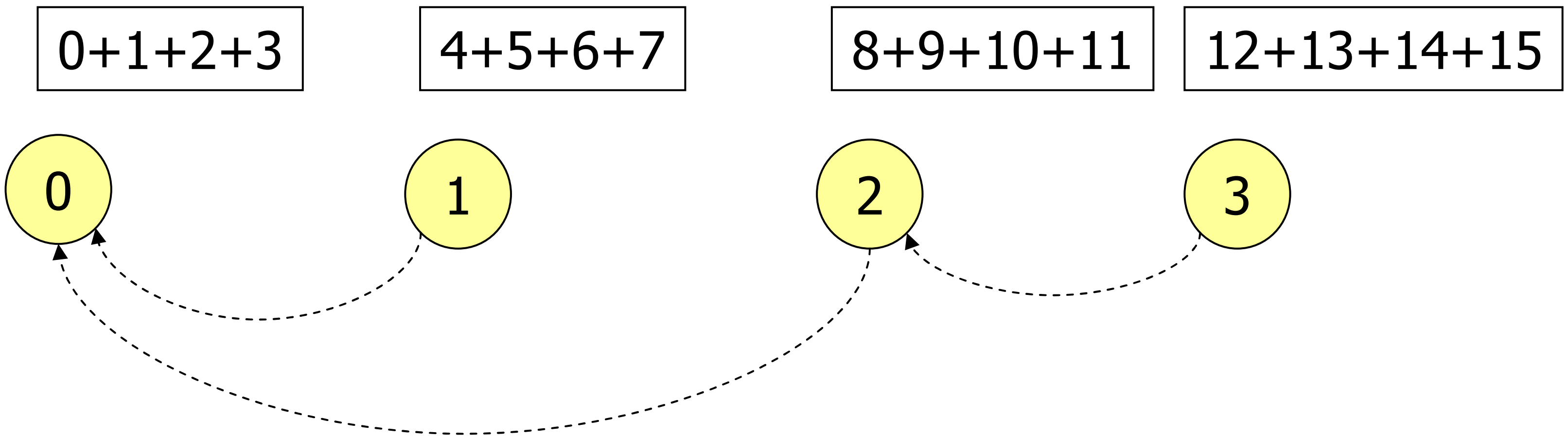


Bad way: $T=\Theta((n/p)\log p)$

Adding n Numbers – Good Way



Adding n Numbers – Good Way

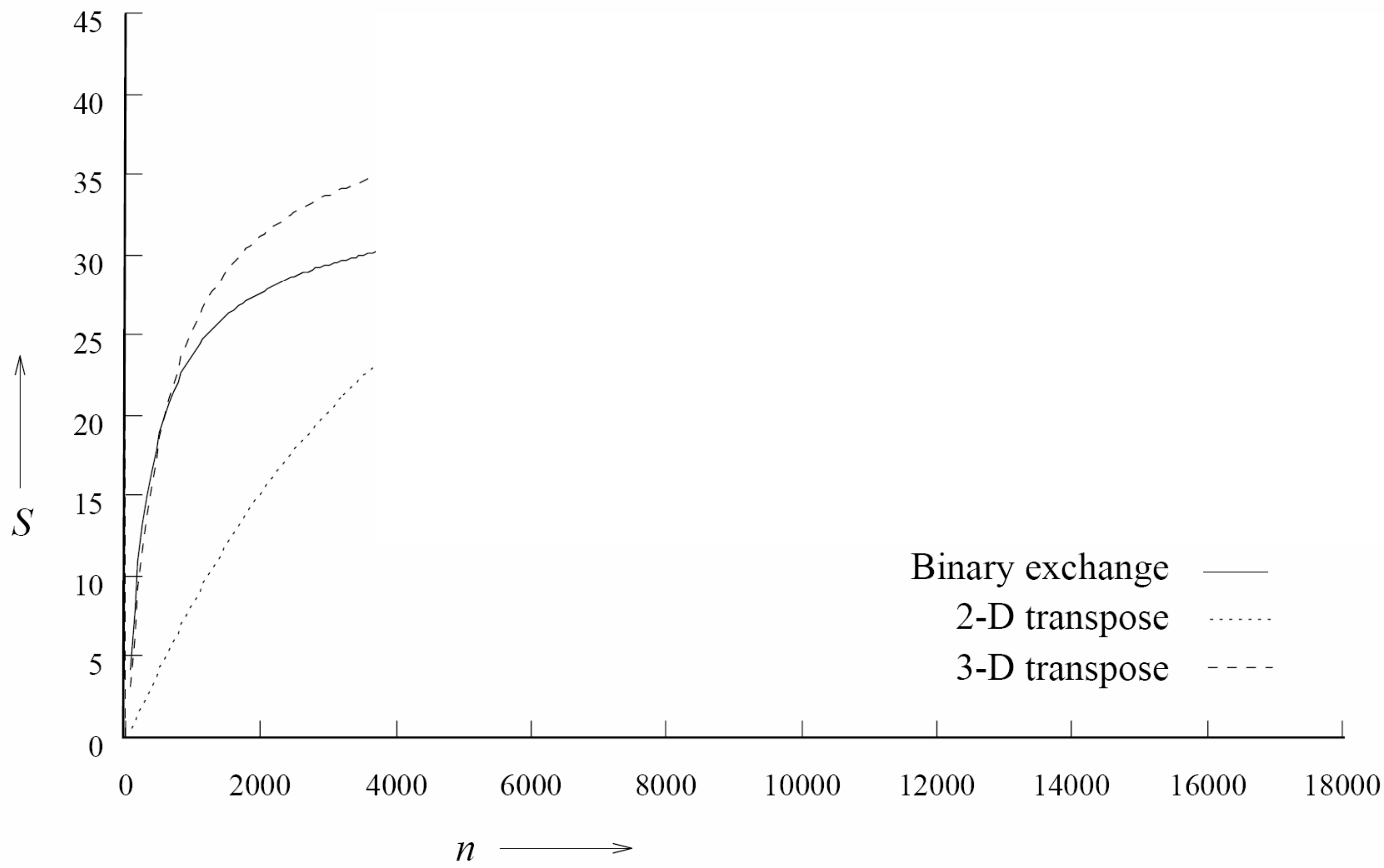


Much less communication. $T=\Theta(n/p + \log p)$.

Scalability of Parallel Systems

- In practice: Develop and test on small systems with small problems.
- Problem: What happens for the real large problems on large systems?
  - Difficult to extrapolate results.

Problem with Extrapolation



Scaling Characteristics of Parallel Programs

- Rewrite efficiency (E):

$$\begin{cases} E = \dfrac{S}{p} = \dfrac{T_S}{pT_p} \\ pT_p = T_0 + T_S \end{cases} \Rightarrow E = \dfrac{1}{1 + \dfrac{T_0}{T_S}}$$

- What does it tell us?

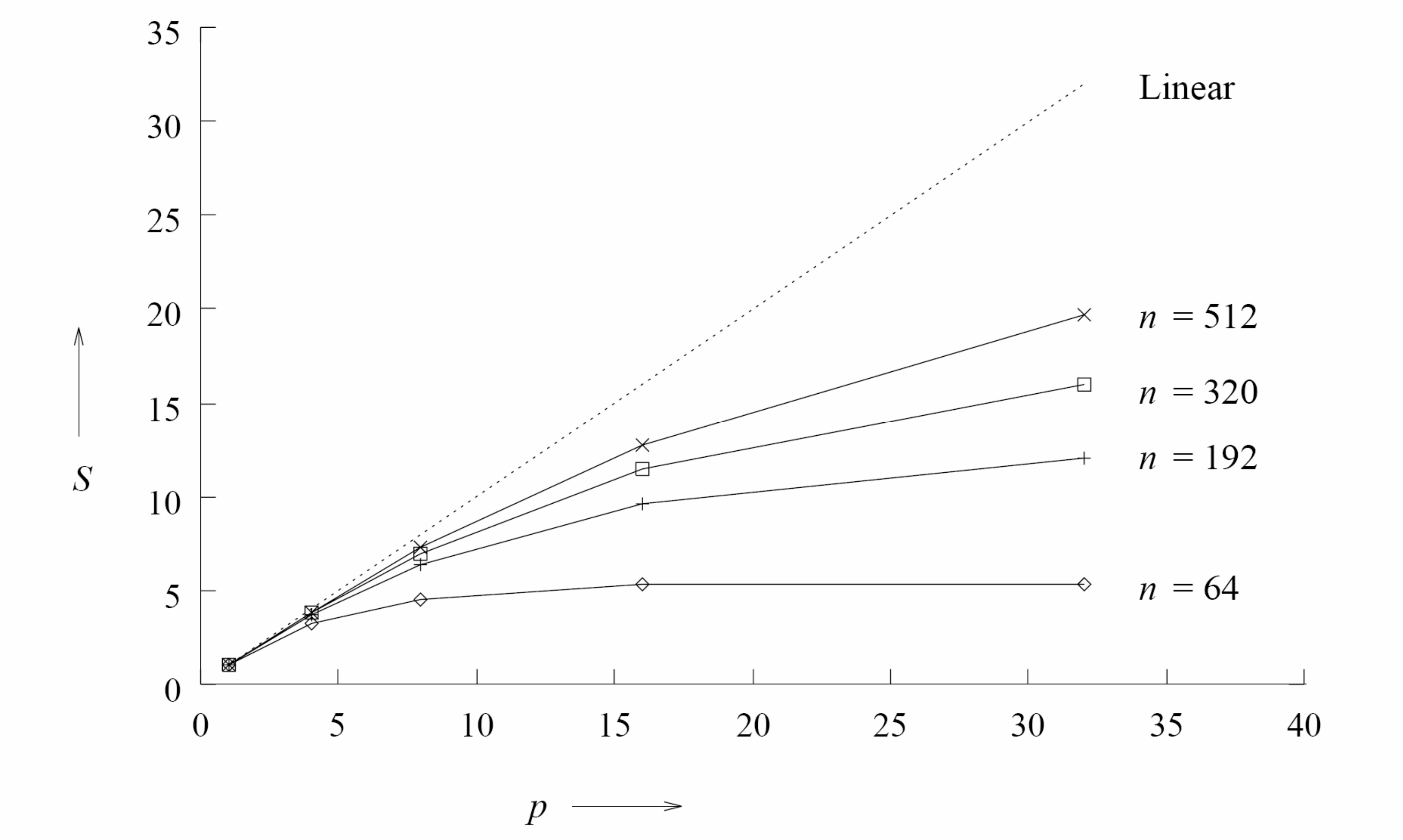
Example: Adding Numbers

Table 5.1 Efficiency as a function of n and p for adding n numbers on p processing elements.

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

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

Speedup


35
30
25
20
15
10
5
0
0
5
10
15
20
25
30
35
40
Linear
n = 512
n = 320
n = 192
n = 64
S
p

Scalable Parallel System

- Can maintain its efficiency constant when increasing the number of processors and the size of the problem.
- In many cases $T_0=f(T_S,p)$ and grows sub-linearly with T_S. It can be possible to increase p and T_S and keep E constant.
- Scalability measures the ability to increase speedup in function of *p*.

Cost-Optimality

- Cost optimal parallel systems have efficiency $\Theta(1)$.
- So scalability and cost-optimality are linked.
- Adding number example: becomes cost-optimal when $n=\Omega(p \log p)$.

Scalable System

- Efficiency can be kept constant when
  - the number of processors increases and
  - the problem size increases.
- At which rate the problem size should increase with the number of processors?
  - The rate determines the degree of scalability.
- In complexity problem size = size of the input. Here = number of basic operations to solve the problem. Noted W.

Rewrite Formulas

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 Function

- For scalable systems efficiency can be kept constant if T_0/W is kept constant.

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

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

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

$$W = KT_0(W,p)$$

Example

- Adding number: We saw that $T_0=2p\log p$.
- We get $W=K\,2p\log p$.
- If we increate p to p', the problem size must be increased by $(p'\log p')/(p\log p)$ to keep the same efficiency.
  - Increase p by p'/p.
  - Increase n by $(p'\log p')/(p\log p)$.

Example

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

$$W = Kp^{3/2}.$$

$$\begin{aligned} W &= Kp^{3/4}W^{3/4} \\ W^{1/4} &= Kp^{3/4} \\ W &= K^4p^3 \end{aligned}$$

Isoefficiency = $\Theta(p^3)$.

Why?

- After isoefficiency analysis, we can test our parallel program with few processors and then predict what will happen for larger systems.

Link to Cost-Optimality

A parallel system is cost-optimal iff
$pT_P = \Theta(W)$.

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

A parallel system is cost-optimal iff its overhead (T_0) does not exceed (asymptotically) the problem size.

Lower Bounds

- For a problem consisting of W units of work, $p \leq W$ processors can be used optimally.
- $W=\Omega(p)$ is the lower bound.
- For a degree of concurrency C(W), $p \leq C(W)$.
  - $C(W)=\Theta(W)$ for optimality (necessary condition).

Example

- Gaussian elimination: $W=\Theta(n^3)$.
  - But eliminate *n* variables consecutively with $\Theta(n^2)$ operations $\rightarrow C(W) = O(n^2) = O(W^{2/3})$.
  - Use all the processors: $C(W)=\Theta(p) \rightarrow W=\Omega(p^{3/2})$.

Minimum Execution Time

- If T_P $\searrow\nearrow$ in function of p, we want its minimum. Find p_0 s.t. $dT_P/dp=0$.
- Adding n numbers: $T_P=n/p+2\log p$.
  $\rightarrow p_0=n/2$.
  $\rightarrow T_P^{min}=2\log n$.
- Fastest but not necessary cost-optimal.

Cost-Optimal Minimum Execution Time

- If we solve cost-optimally, what is the minimum execution time?
- We saw that if isoefficiency function = $\Theta(f(p))$ then a problem of size W can be solved optimally iff $p=\Omega(f^{-1}(W))$.
- Cost-optimal system: $T_P=\Theta(W/p)$ $\rightarrow T_P^{cost\_opt}=\Omega(W/f^{-1}(W))$.

Example: Adding Numbers

- Isoefficiency function $f(p)=\Theta(p \log p)$.
  $W=n=f(p)=p \log p \rightarrow \log n=\log p \log\log p$.
  We have approximately $p=n/\log n=f^{-1}(n)$.
- $T_P^{cost\_opt}=\Omega(W/f^{-1}(W))$
  $=\Omega(n/\log n * \log(n/\log n) / (n/\log n))$
  $= \Omega(\log(n/\log n))= \Omega(\log n - \log\log n)= \Omega(\log n)$.
- $T_P=\Theta(n/p+\log p)=\Theta(\log n+\log(n/\log n))$
  $=\Theta(2\log n-\log\log n)= \Theta(\log n)$.
- For this example $T_P^{cost\_opt}= \Theta(T_P^{min})$.

Remark

- If $p_0 > C(W)$ then its value is meaningless. T_P^{min} is obtained for $p=C(W)$.

Asymptotic Analysis of Parallel Programs

Table 5.2 Comparison of four different algorithms for sorting a given list of numbers. The table shows number of processing elements, parallel runtime, speedup, efficiency and the pT_P product.

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

Other Scalability Metrics

- Scaled speedup: speedup when problem size increases linearly in function of p.
  - Motivation: constraints such as memory linear in function of p.
  - Time and memory constrained.