RECAP

Computation →
Communication within the processor
(or cores)

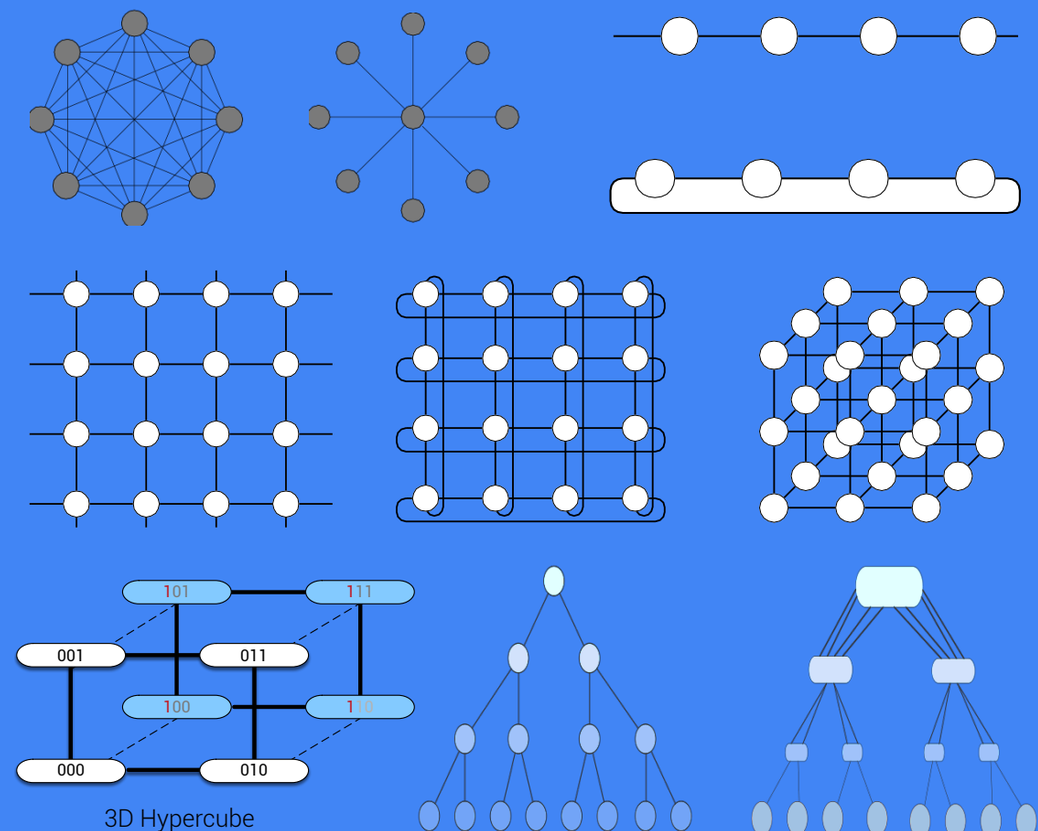
SIMD vs. MIMD (SPMD)

Buses, Crossbars

NUMA vs. UMA

RECAP

Communication (across machines) →
Network Topologies →



3D Hypercube

Degree.

Distance vs. Diameter

Connectivity vs. Bisection Width

Latency vs. Bandwidth

This Week

Measuring Performance

Basic Communication Operations
Communication Costs
Merging Networks →
Best of all involved

Student Presentation (next week)

Performance Evaluation

Basic Messaging

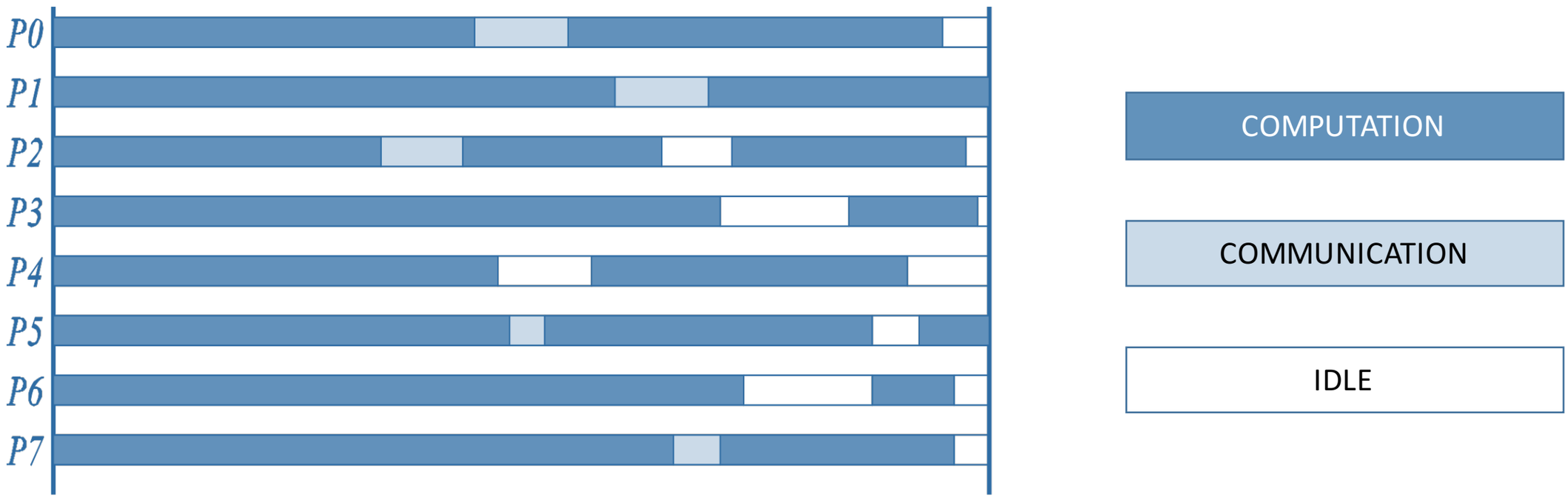
Serial vs. Parallel

- A sequential algorithm is evaluated by its runtime (in general, asymptotic runtime as a function of input size).
- The asymptotic runtime of a sequential program is identical on any serial platform.

- The parallel runtime of a program depends on the input size, the number of processors, and the communication parameters of the machine.
- A parallel system → algorithm + architecture.

Serial vs. Parallel (How do we quantify benefit)

- Wall clock time - the time from the start of the first processor to the stopping time of the last processor in a parallel ensemble
- If time take by a serial code is T, and we use p processors, would a parallelized version take T/p?



Other ways to Quantify

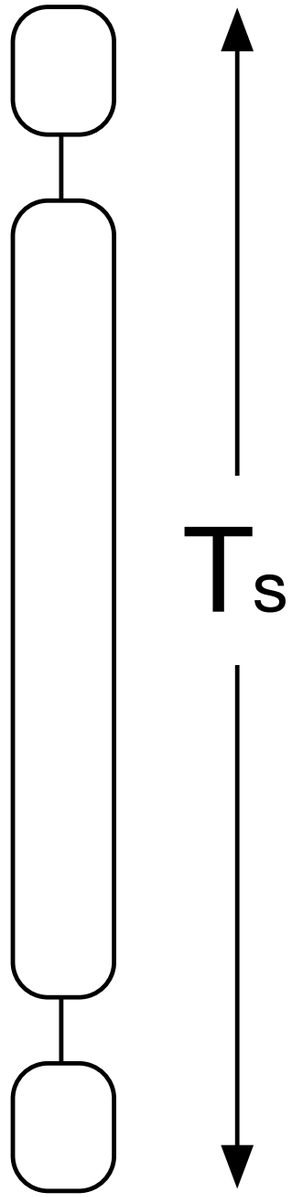
Serial → T_S

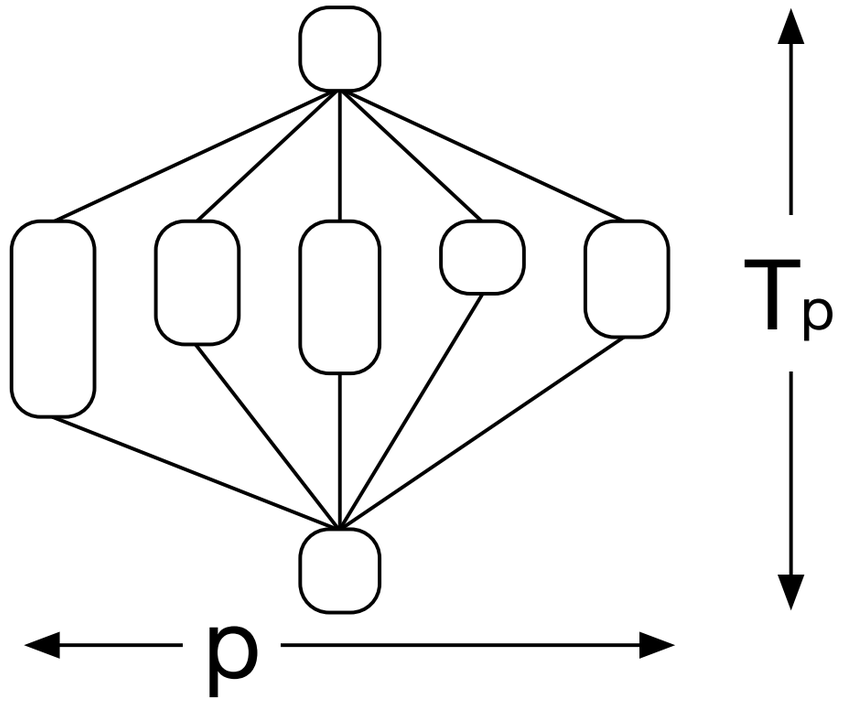
Parallel T_{ALL} → pT_P

Overhead

T_o → $T_{ALL} - T_S$

T_o → $pT_P - T_S$





Example 1

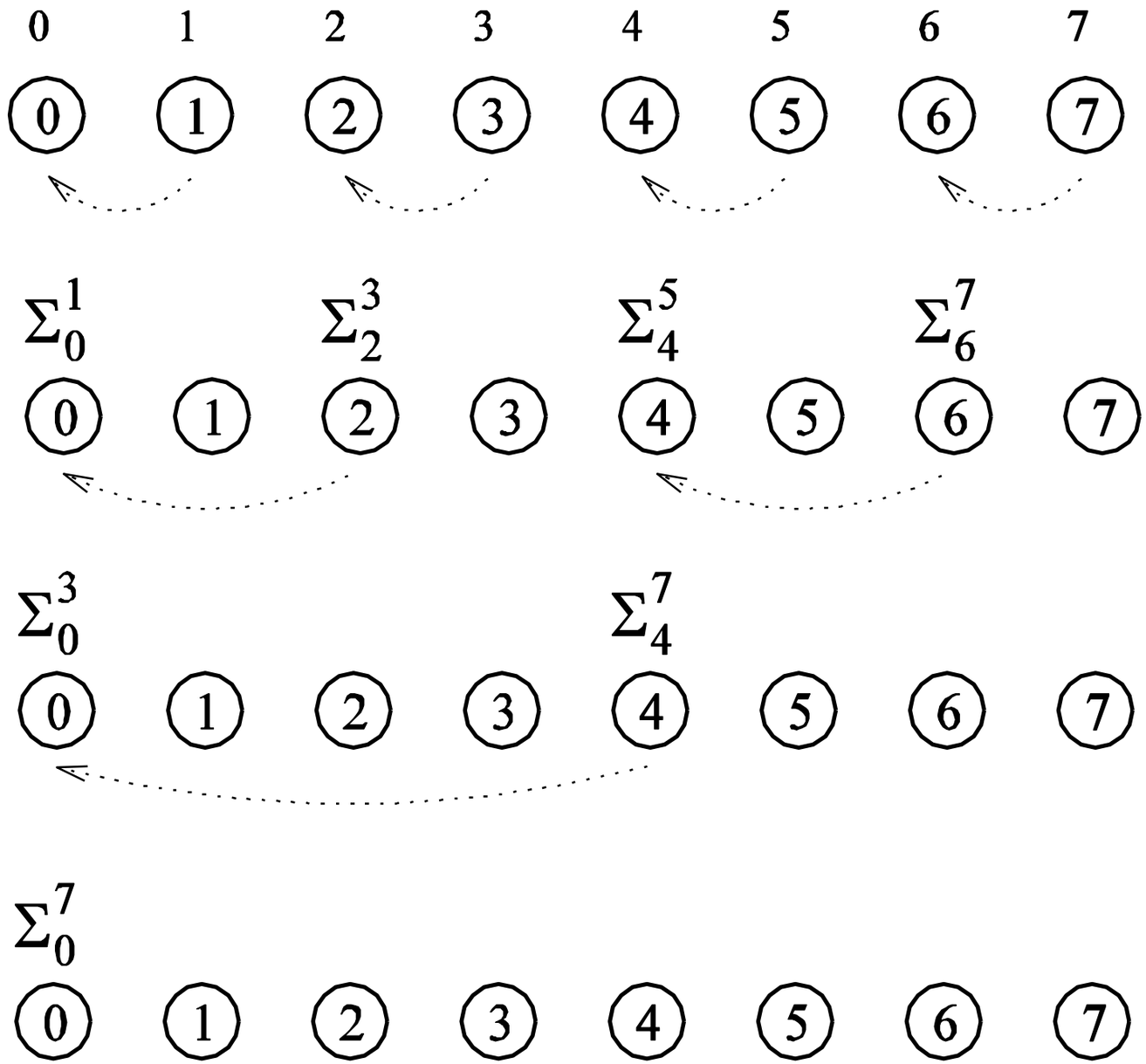
Add n numbers

Ts → O(n)

How would one do this in **log n** steps

T_P = O(log n)

S → O (n / log(n))



What's the benefit: SPEEDUP

Speedup (S) is 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.

Speedup (Example)

- Consider the problem of parallel bubble sort.
- The serial time for bubble-sort is 150 seconds.
- The parallel time for odd-even sort (efficient parallelization of bubble sort) is 40 seconds.
- The speedup would appear to be 150/40 = 3.75.
- But is this really a fair assessment of the system?
- What if serial quicksort only took 30 seconds? In this case, the speedup is 30/40 = 0.75. This is a more realistic assessment of the system.

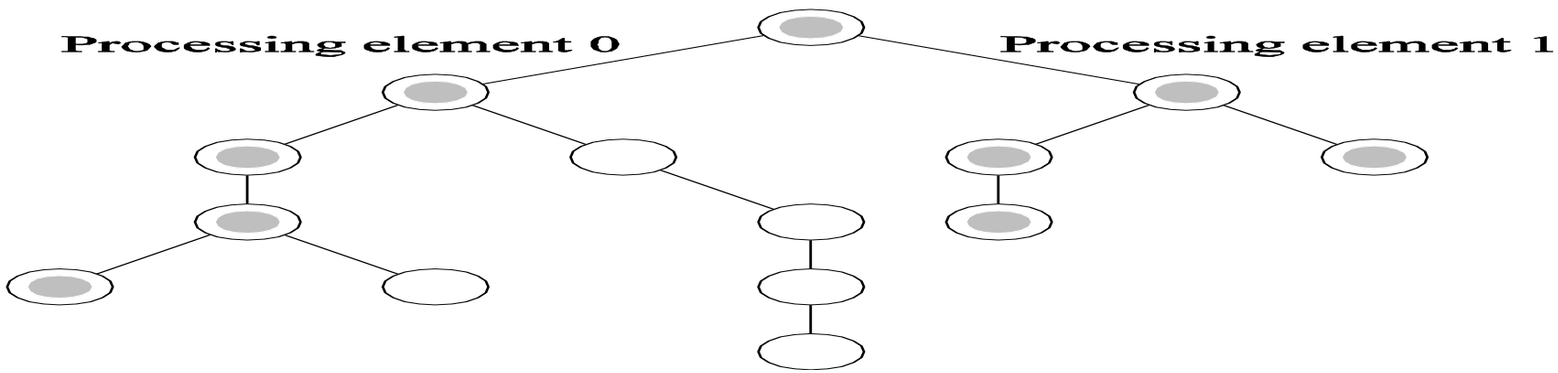
Speedup (Max and Min)

- Speedup can be as low as 0 (the parallel program never terminates).

- Max Speedup?
  - Theoretically → p (number of processors)
  - Can it be greater than p?
    - SUPERLINEAR SPEEDUP
    - Parallel does less work than serial → S > p
    - Parallel can use more resources than serial → S > p

Example S > p

One reason for superlinearity is that the parallel version does less work than corresponding serial algorithm.



Searching an unstructured tree for a node with a given label, `S', on two processing elements using depth-first traversal. The two-processor version with processor 0 searching the left subtree and processor 1 searching the right subtree expands only the shaded nodes before the solution is found. The corresponding serial formulation expands the entire tree. It is clear that the serial algorithm does more work than the parallel algorithm.

Efficiency

- Speedup → Serial / Parallel

- Efficiency → Are you keeping your processors busy?

$$\mathrm{E} = \left(\frac{Speedup}{processors}\right) \rightarrow \left(\frac{T_s/T_p}{p}\right) \rightarrow \left(\frac{T_s}{pT_p}\right)$$

What is the Efficiency of the number adder?

S → $\frac{Ts}{Tp}$

Efficiency = S / p

In adder n = p

→ $\Theta\left(\frac{n/\log n}{n}\right)$

→ $\Theta\left(\frac{1}{\log n}\right)$

Cost

- Speedup → Serial / Parallel
- Efficiency → Speedup / Number of Processors

$$\text{COST} \rightarrow p \times T_P$$

- Cost reflects the sum of the time that each processing element spends solving the problem.
- A parallel system is said to be *cost-optimal* if the cost of solving a problem on a parallel computer is asymptotically identical to serial cost.

Since $\boldsymbol{E} = \boldsymbol{T_S} / \boldsymbol{p\,T_P}$, for cost optimal systems, $\boldsymbol{E} = O(1)$.

Cost of a Parallel System: Example

Consider the problem of adding numbers

We have, $T_P = \log n$ (for $p = n$).

The cost of this system is given by $p\,T_P = n \log n$.

Since the serial runtime of this operation is $\Theta(n)$, the algorithm is not cost optimal.

Consider a parallel sorting algorithm that uses n processing elements to sort the list in time $(\log n)^2$.

- Since the serial runtime of a (comparison-based) sort is $T_s = n \log n$, the speedup and efficiency of this algorithm are given by n / $\log n$ and 1 / $\log n$, respectively.
- COST of this algorithm is n $(\log n)^2$.
- This algorithm is not cost optimal but only by a factor of $\log n$.

- If $p < n$, assigning n tasks to p processors gives $T_P = n$ $(\log n)^2$ / p .
- The corresponding speedup of this formulation is p / $\log n$.
- This speedup goes down as the problem size n is increased for a given p !

Non-Optimal Cost

- Assume n = 1024 and p = 32 processors
- Speedup → p / log(n) → 32/10 = 3.2
- If n = 10^6 → S = 32/~20 = 1.6
- If n = 10^9 → S = 32/~30 = ~1.0

SIGNIFICANT COSTS WITH NOT BEING COST OPTIMAL

Cost of solving in parallel has the same asymptotic growth as the serial version

Granularity

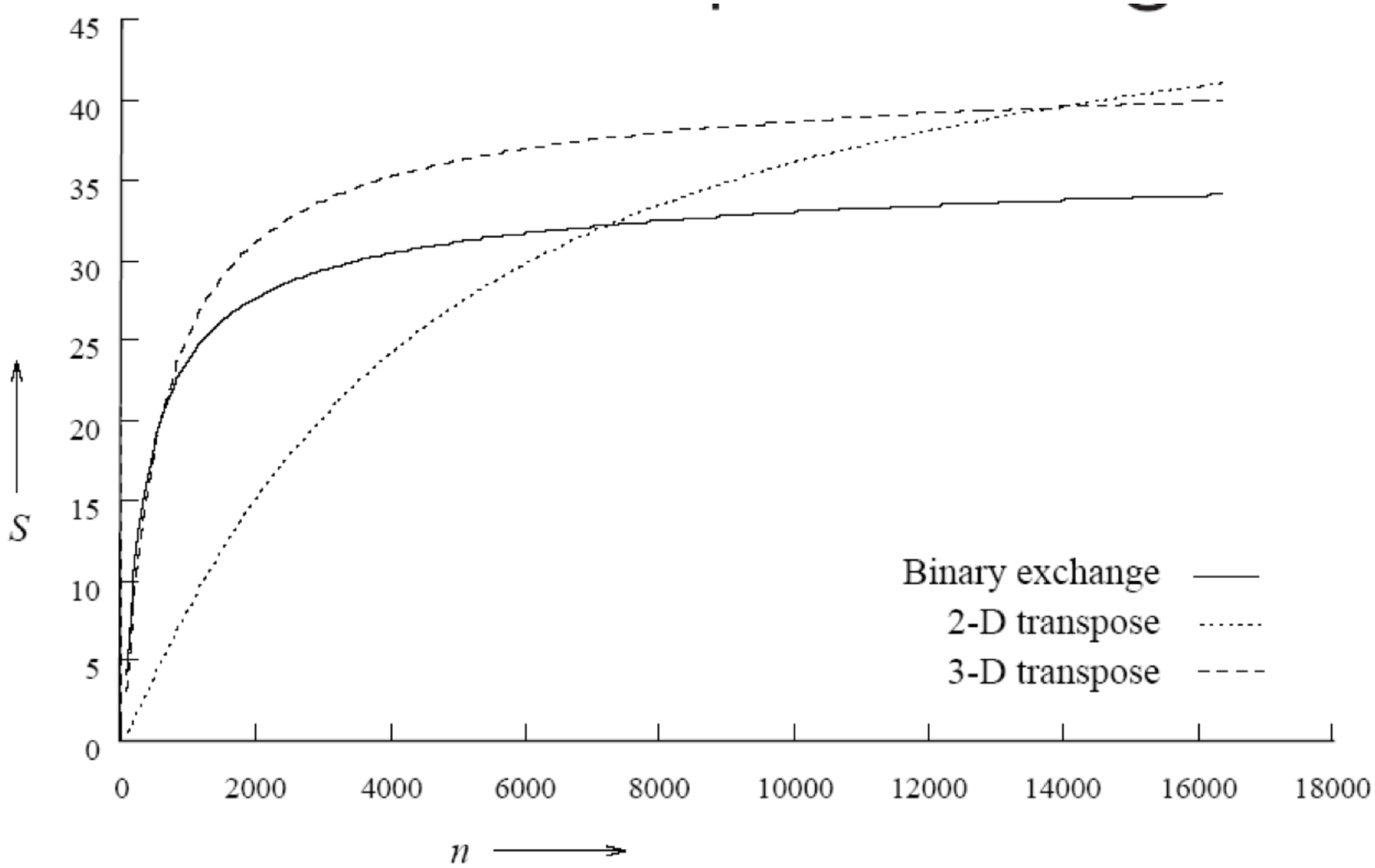
- The adder example was not optimal
  - One processor per element

- In practice: assign large pieces to each element → Increase GRANULARITY

Scaling Characteristics of Parallel Programs

- For a given problem size (i.e., the value of T_S remains constant), as we increase the number of processing elements, T_o increases.

- The overall efficiency of the parallel program goes down. This is the case for all parallel programs.


45
40
35
30
25
20
15
10
5
0
S
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
n
Binary exchange
2-D transpose
3-D transpose

Scaling Characteristics of Parallel Programs

- The efficiency of a parallel program can be written as:

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

or

$$E = \frac{1}{1 + \frac{T_o}{T_S}}. \qquad (4)$$

- The total overhead function $\boldsymbol{T_o}$ is an increasing function of $\boldsymbol{p}$.

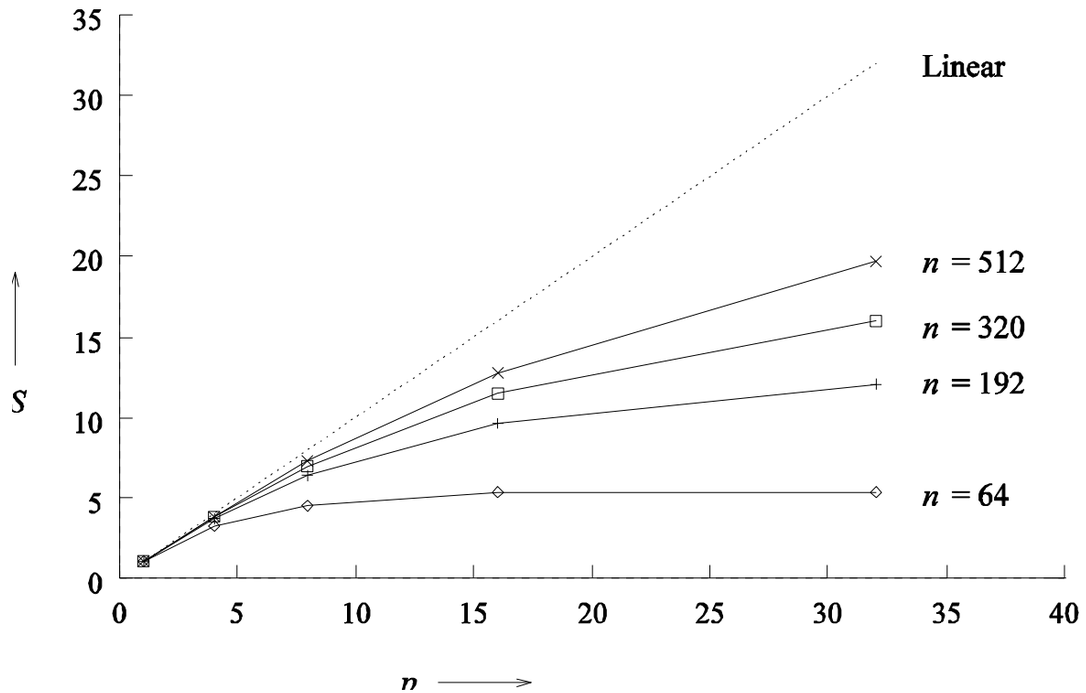
Adder Example (n elements, p processors)

- Assume a unit time to add two numbers
- First Local summation → n / p time
- Communication → log p →→ subsequent summation → log p

- Tp = n / p + 2 log p
- S = Ts / Tp → $\frac{n}{\frac{n}{p} + 2 \log p}$
- E = S / p → $\frac{1}{1 + \frac{2p \log p}{n}}$

Scaling Characteristics of Parallel Programs: Example (continued)

Plotting the speedup for various input sizes gives us:



Speedup versus the number of processing elements for adding a list of numbers.

Speedup tends to saturate and efficiency drops as a consequence of Amdahl's law

Weak vs. Strong Scaling

- Say you have a task X (matrix multiplication)

- **WEAK →** Increase the size of the problem but keep the resource fixed.
  Size/Resource is fixed.
  Bigger Matrix → Same Processor
  Gives you an idea on how your algorithm behaves with fixed size/core

- **STRONG →** Do the job faster but keep the problem fixed.
  Size/Resource varies.
  Faster Multiplication
  Gives you an idea on how your problem behaves with a fixed total size

Basic Communication Operations: Introduction

- Many interactions in practical parallel programs occur in well-defined patterns involving groups of processors.
- Efficient implementations of these operations can improve performance, reduce development effort and cost, and improve software quality.
- Efficient implementations must leverage underlying architecture. For this reason, we refer to specific architectures here.
- We select a descriptive set of architectures to illustrate the process of algorithm design.

Basic Communication Operations: Introduction

- Group communication operations are built using point-to-point messaging primitives.

- We assume that the network is bidirectional and that communication is single-ported.

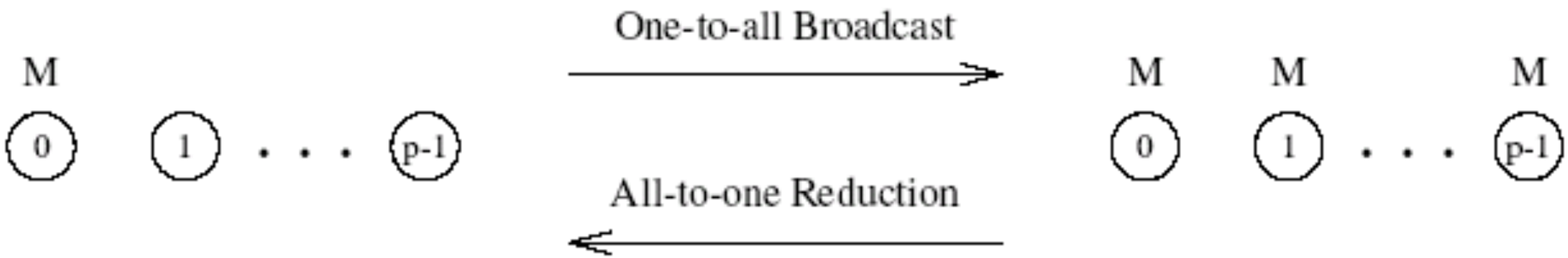
One-to-All Broadcast and All-to-One Reduction

- One processor has a piece of data (of size *m*) it needs to send to everyone.
- The dual of one-to-all broadcast is *all-to-one reduction*.

- In all-to-one reduction, each processor has *m* units of data. These data items must be combined piece-wise (using some associative operator, such as addition or min), and the result made available at a target processor.

One-to-All Broadcast and All-to-One Reduction on Rings

- Simplest way is to send *p-1* messages from the source to the other *p-1* processors - this is not very efficient.
- Use recursive doubling: source sends a message to a selected processor. We now have two independent problems derined over halves of machines.
- Reduction can be performed in an identical fashion by inverting the process.



Next

Messaging Algorithms and their analysis