Performance Metrics for Parallel Systems and Asymptotic Analysis of Parallel Programs

Ref: ``Introduction to Parallel Computing'',
Addison Wesley, 2003. A Grama, A Gupta, G Karypis, and V Kumar

Effect of Granularity on Performance

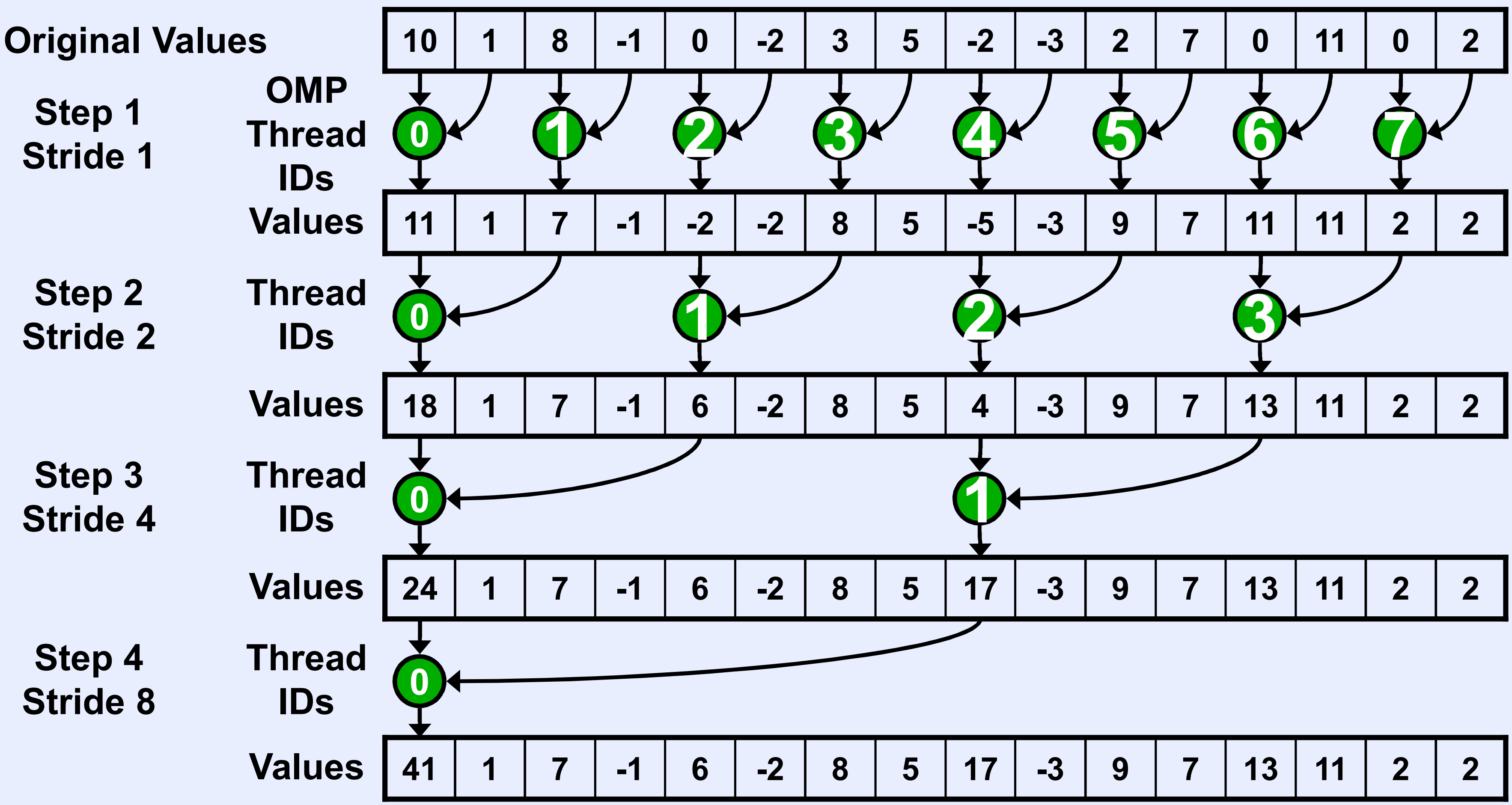
- Often, using fewer processors improves performance of parallel systems.
- Using fewer than the maximum possible number of processing elements to execute a parallel algorithm is called *scaling down* a parallel system.
- A naive way of scaling down is to think of each processor in the original case as a virtual processor and to assign virtual processors equally to scaled down processors.
- Since the number of processing elements decreases by a factor of n / p, the computation at each processing element increases by a factor of n / p.
- The communication cost should not increase by this factor since some of the virtual processors assigned to a physical processors might talk to each other. This is the basic reason for the improvement from building granularity.

Building Granularity: Example

- Consider the problem of adding n numbers on p processing elements such that $p < n$ and both n and p are powers of 2.

- Use the parallel algorithm for n processors, except, in this case, we think of them as virtual processors.
- Each of the p processors is now assigned n / p virtual processors.
- The first **log** p of the **log** n steps of the original algorithm are simulated in (n / p) **log** p steps on p processing elements.
- Subsequent **log** n - **log** p steps do not require any communication.
- A single processing element is left with n/p numbers taking time O(n/p).

Parallel Reduction SUM



Building Granularity: Example

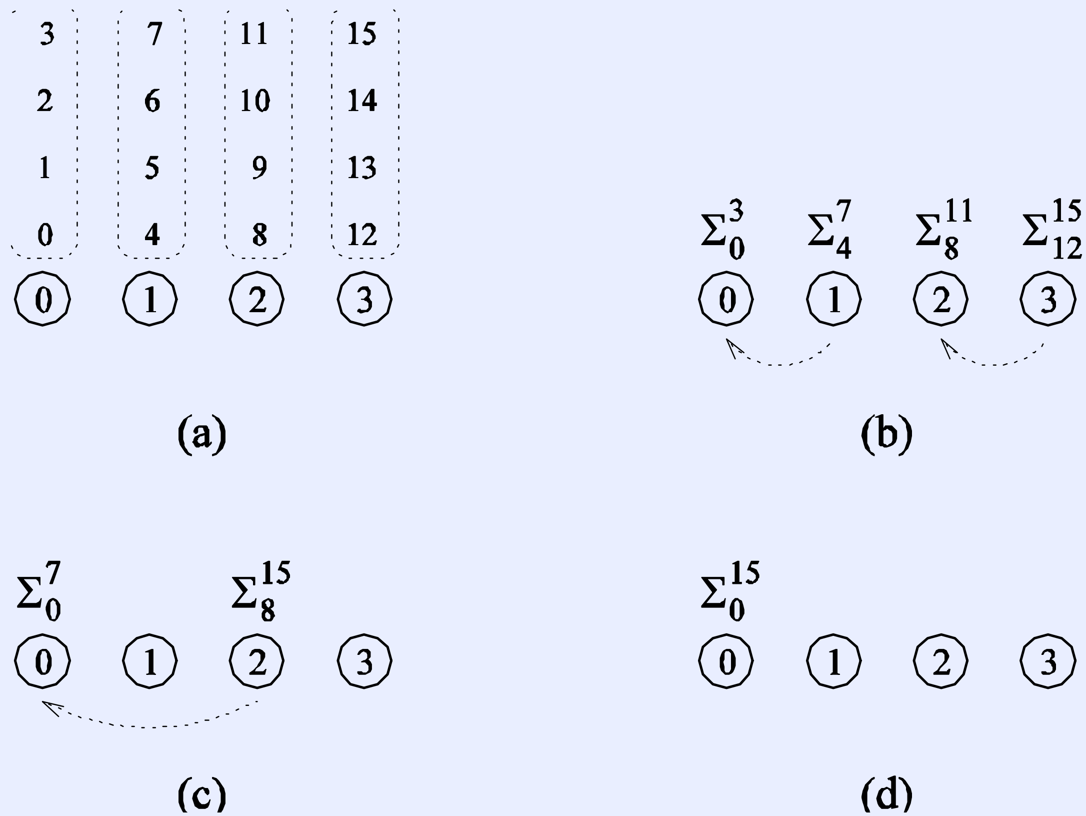
- The overall parallel execution time of this parallel system is
  $\Theta\left((\boldsymbol{n} / \boldsymbol{p}) \log \boldsymbol{p}\right)$.

- The cost is $\Theta(\boldsymbol{n} \log \boldsymbol{p})$, which is asymptotically higher than the $\Theta(\boldsymbol{n})$ cost of adding ***n*** numbers sequentially. Therefore, the parallel system is not cost-optimal.

Building Granularity: Example

Can we build granularity in the example in a cost-optimal fashion?

- Each processing element locally adds its ***n*** / ***p*** numbers in time Θ (***n*** / ***p***).



A cost-optimal way of computing the sum of 16 numbers using four processing elements.

The ***p*** partial sums on ***p*** processing elements can be added in time Θ(***n*** /***p***).

Building Granularity: Example

- The parallel runtime of this algorithm is

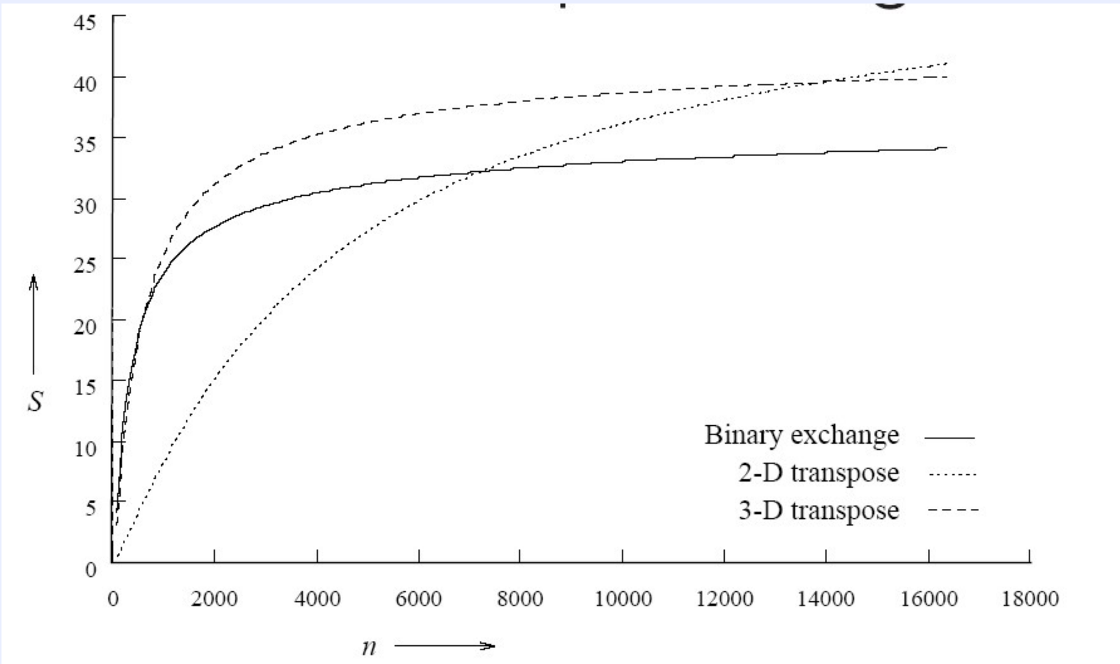
$$T_P = \Theta(n/p + \log p),$$

- The cost is $\Theta(n + p \log p)$
- This is cost-optimal, so long as $n = \Omega(p \log p)$

Scalability of Parallel Systems

How do we extrapolate performance from small problems and small systems to larger problems on larger configurations?

Consider three parallel algorithms for computing an ***n***-point Fast Fourier Transform (FFT) on 64 processing elements.



A comparison of the speedups obtained by the binary-exchange, 2-D transpose and 3-D transpose algorithms on 64 processing elements. Clearly, it is difficult to infer scaling characteristics from observations on small datasets on small machines.

Scaling Characteristics

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

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

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

or

- The total overhead function T_o is an increasing function of p .

Scaling Characteristics

- 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.

Scaling Characteristics

- Consider the problem of adding n numbers on p processing elements.

- We have seen that:

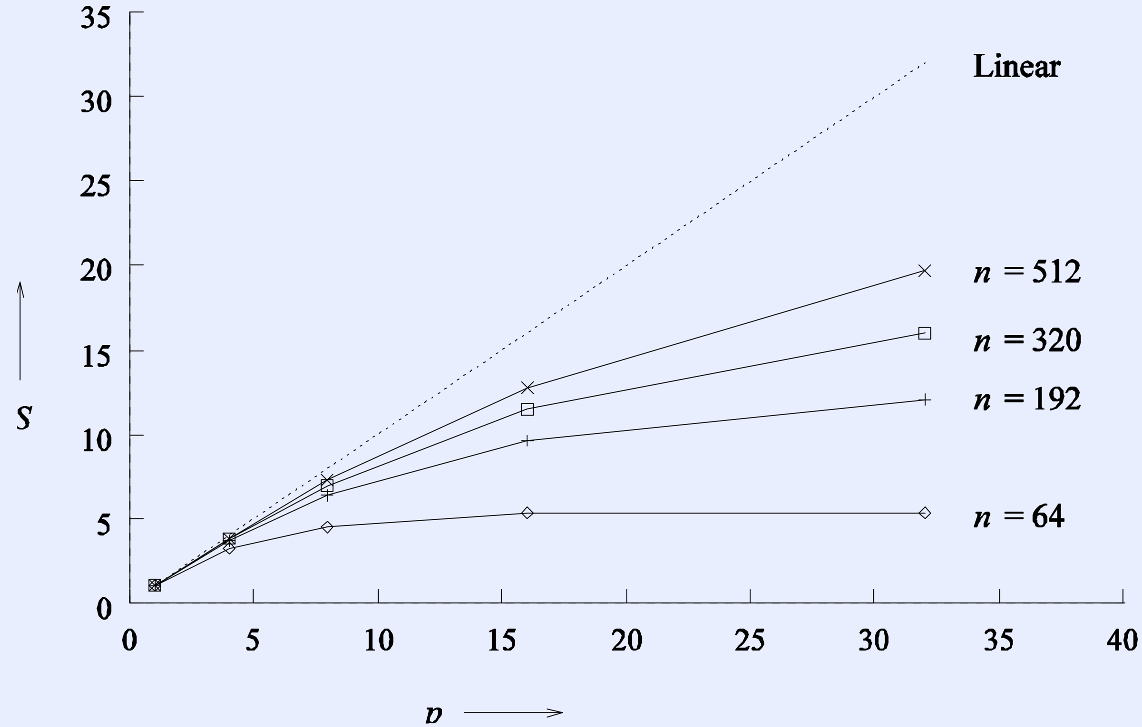
$$T_P = \frac{n}{p} + 2\log p$$

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

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

Scaling Characteristics of Parallel Programs

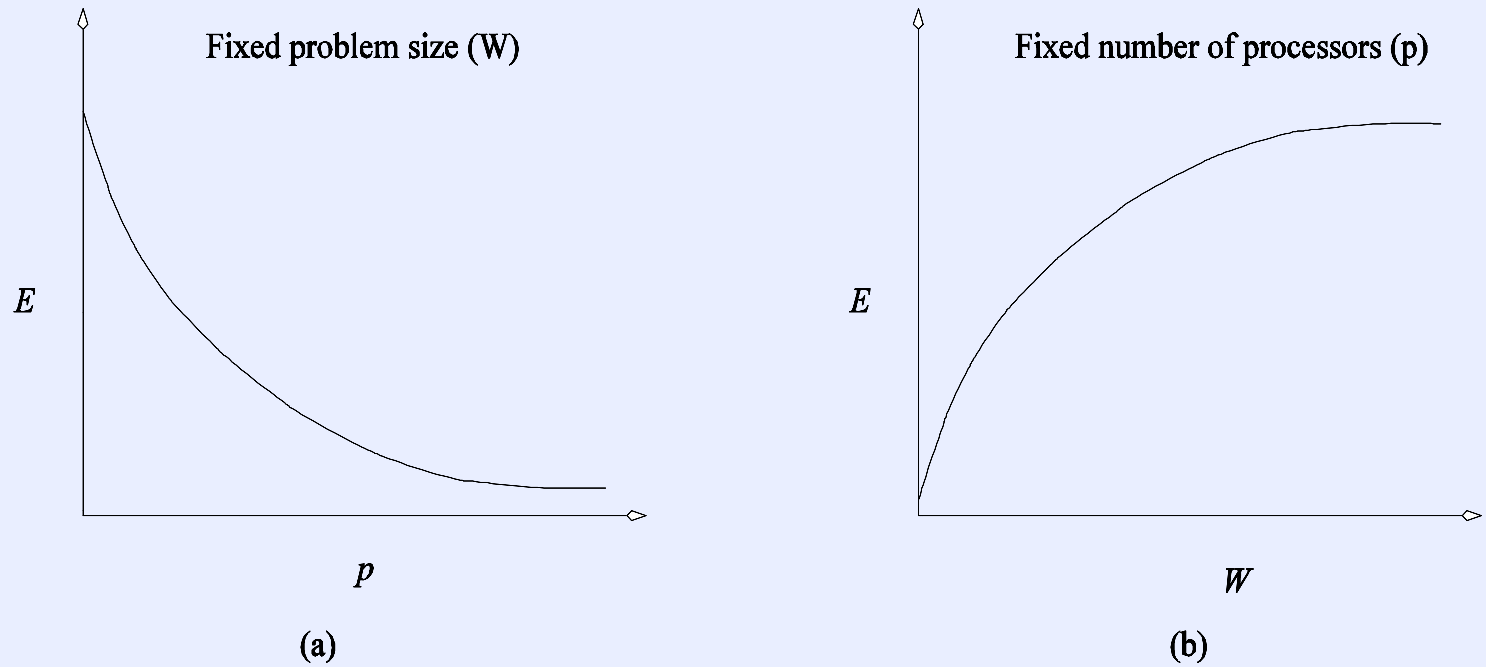
Plotting the speedup for various input sizes gives us:



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.

Isoefficiency Metric of Scalability



Variation of efficiency: (a) as the number of processing elements is increased for a given problem size; and (b) as the problem size is increased for a given number of processing elements. The phenomenon illustrated in graph (b) is not common to all parallel systems.

Isoefficiency Metric of Scalability

- What is the rate at which the problem size must increase with respect to the number of processing elements to keep the efficiency fixed?
- This rate determines the scalability of the system. The slower this rate, the better.
- Before we formalize this rate, we define the problem size ***W*** as the asymptotic number of operations associated with the best serial algorithm to solve the problem.

Isoefficiency Metric of Scalability

- We can write parallel runtime as:

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

- The resulting expression for speedup is

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

Finally, we write the expression for efficiency as

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

Isoefficiency Metric of Scalability

- For scalable parallel systems, efficiency can be maintained at a fixed value (between 0 and 1) if the ratio T_o / W is maintained at a constant value.
- For a desired value E of efficiency,

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

- If $K = E / (1 - E)$ is a constant depending on the efficiency to be maintained, since T_o is a function of W and p, we have

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

Isoefficiency Metric of Scalability

- The problem size ***W*** can usually be obtained as a function of ***p*** by algebraic manipulations to keep efficiency constant.

- This function is called the *isoefficiency function*.

- This function determines the ease with which a parallel system can maintain a constant efficiency and hence achieve speedups increasing in proportion to the number of processing elements