Recap from the previous lecture on Analytical Modeling

- Speedup: \( S_p = T_s / T_p (p) \)
- Efficiency \( E = S_p / p \)
- Parallel Cost: \( C = p * T_p \)
- Parallel Overhead: \( T_o = p * T_p - T_s \)
An example

- Adding \( n \) numbers using \( n \) processing elements

\[
\begin{array}{cccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
\sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} & \sum_{i=1}^{n} \\
\end{array}
\]

An example (II)

- Each step consists of one addition and one communication operation
  - Addition two numbers takes constant time, e.g. \( t_c \)
  - Communication takes constant time per step

\[ T_p = O(\log n) \]

- Sequential algorithm takes \( n \) times \( t_c \) time units
\[ T_s = O(n) \]

\[ S_p = O(n/\log n) \]

- Parallel Efficiency
\[ E = O \left( \frac{n}{\log n} \right) / n \]
\[ = O \left( \frac{1}{\log n} \right) \]
Cost optimality

• A parallel system is said to be cost optimal if the cost of solving a problem on a parallel computer has the same asymptotic growth as a function of the input size as the fastest known sequential algorithm on a single processing element.

• For adding \( n \) number on \( n \) processing elements:

  Serial runtime: \( O(n) \)

  Parallel cost: \( C = n \times T_p = O(n \log n) \)

\( \Rightarrow \) algorithm is not cost optimal

A modified example

• Adding \( n \) elements on \( p \) processing elements, with \( p < n \)

\begin{table}
\begin{tabular}{cccc}
rank & 0 & 1 & 2 & 3 \\
\hline
13 & 14 & 15 & 16 \\
9 & 10 & 11 & 12 \\
5 & 6 & 7 & 8 \\
1 & 2 & 3 & 4 \\
\end{tabular}
\end{table}

\begin{table}
\begin{tabular}{cccc}
rank & 0 & 1 & 2 & 3 \\
\hline
\sum_{i=1}^{10} & \sum_{i=11}^{16} & \sum_{i=13}^{15} & \sum_{i=14}^{16} \\
\sum_{i=1}^{9} & \sum_{i=11}^{16} & \sum_{i=12}^{15} & \sum_{i=16}^{16} \\
\sum_{i=2}^{9} & \sum_{i=12}^{16} & \sum_{i=13}^{15} & \sum_{i=16}^{16} \\
\sum_{i=3}^{10} & \sum_{i=13}^{16} & \sum_{i=14}^{15} & \sum_{i=16}^{16} \\
\sum_{i=4}^{8} & \sum_{i=14}^{16} & \sum_{i=15}^{15} & \sum_{i=16}^{16} \\
\sum_{i=5}^{7} & \sum_{i=15}^{16} & \sum_{i=16}^{16} & \sum_{i=16}^{16} \\
\sum_{i=6}^{6} & \sum_{i=16}^{16} & \sum_{i=16}^{16} & \sum_{i=16}^{16} \\
\sum_{i=7}^{4} & \sum_{i=16}^{16} & \sum_{i=16}^{16} & \sum_{i=16}^{16} \\
\sum_{i=8}^{3} & \sum_{i=16}^{16} & \sum_{i=16}^{16} & \sum_{i=16}^{16} \\
\sum_{i=9}^{2} & \sum_{i=16}^{16} & \sum_{i=16}^{16} & \sum_{i=16}^{16} \\
\sum_{i=10}^{1} & \sum_{i=16}^{16} & \sum_{i=16}^{16} & \sum_{i=16}^{16} \\
\sum_{i=11}^{0} & \sum_{i=16}^{16} & \sum_{i=16}^{16} & \sum_{i=16}^{16} \\
\sum_{i=12}^{1} & \sum_{i=16}^{16} & \sum_{i=16}^{16} & \sum_{i=16}^{16} \\
\sum_{i=13}^{2} & \sum_{i=16}^{16} & \sum_{i=16}^{16} & \sum_{i=16}^{16} \\
\sum_{i=14}^{3} & \sum_{i=16}^{16} & \sum_{i=16}^{16} & \sum_{i=16}^{16} \\
\sum_{i=15}^{4} & \sum_{i=16}^{16} & \sum_{i=16}^{16} & \sum_{i=16}^{16} \\
\sum_{i=16}^{5} & \sum_{i=16}^{16} & \sum_{i=16}^{16} & \sum_{i=16}^{16} \\
\end{tabular}
\end{table}
A second example

- Adding $n$ elements on $p$ processing elements, with $p < n$

  First log $p$ steps: $O \left( \frac{n}{p} \log p \right)$

  Second step: a single processing element is left with $\frac{n}{p}$ elements to add, adding $O \left( \frac{n}{p} \right)$ time

Thus, overall time of this parallel system:

$$T_p = O \left( \frac{n}{p} \log p \right)$$

$$C = O \left( p \times \left( \frac{n}{p} \log p \right) \right) = O \left( n \log p \right)$$

$\implies$ algorithm is still not cost optimal
**Alternative distribution of numbers**

- First step: add \( n/p \) numbers in \( O(n/p) \) time
- Second step: add \( p \) numbers on \( p \) processing elements in \( O(\log p) \) time (see the first example)

\[
T_p = O\left(\frac{n}{p} + \log p\right)
\]

\[
C = O\left(n + p \log p\right) = O(n) \text{ if } n \gg p \Rightarrow \text{cost optimal!}
\]

- The manner in which computation is mapped onto processing elements may determine whether a parallel system is cost optimal

**Scalability of parallel systems**

- Problem: using the performance of small test cases, can you make a prediction for the performance of large test cases?
- Example: cost-optimal algorithm to add \( n \) numbers on \( p \) processes
  - Replace asymptotic calculations by constants
    - One add operation takes one time unit
    - One communication operation takes one time unit
Scalability of parallel systems

- In that case:
  \[ T_p = \frac{n}{p} + \log p + \log p \]
  \[ S_p = \frac{n}{\left( \frac{n}{p} + 2 \log p \right)} \]
  \[ E = \frac{1}{1 + \left( 2p \log p / n \right)} \]

Scalability of parallel systems

- For a given problem size, as we increase the number of processing elements, the overall efficiency of the parallel system goes down.

- In many cases, the efficiency of a parallel system increases if the problem size is increased while keeping the number of processing elements constant.

- Question: is it possible to keep \( E_p \) constant when increasing the number of processing elements?
Scalability of parallel systems

- **Scalability** of a parallel system: ability to maintain a fixed efficiency by simultaneously increasing no. of processes and problem size
- It is often useful to determine the rate at which the problem size $n$ has to increase with respect to the number of processing elements $p$ to maintain a fixed efficiency

<table>
<thead>
<tr>
<th>n</th>
<th>p=1</th>
<th>p=4</th>
<th>p=8</th>
<th>p=16</th>
<th>p=32</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>1.0</td>
<td>0.8</td>
<td>0.57</td>
<td>0.33</td>
<td>0.17</td>
</tr>
<tr>
<td>192</td>
<td>1.0</td>
<td>0.92</td>
<td>0.80</td>
<td>0.60</td>
<td>0.38</td>
</tr>
<tr>
<td>320</td>
<td>1.0</td>
<td>0.95</td>
<td>0.87</td>
<td>0.71</td>
<td>0.50</td>
</tr>
<tr>
<td>512</td>
<td>1.0</td>
<td>0.97</td>
<td>0.91</td>
<td>0.80</td>
<td>0.62</td>
</tr>
</tbody>
</table>

Scalability of parallel systems

- What is the problem size?
  - Often provided as a parameter of input size
  - Problematic, since neither actual memory utilization nor the number of compute operations per input element are well defined
  - Example: extent of a vector, vs. of a 2-D matrix
    - Doubling input size of a 2-D matrix can lead to 8-fold increase in compute time for matrix-matrix multiply operation ( $O(n^3)$ for an $n \times n$ matrix )
  - Better: problem size and the amount of computation should be linearly correlated
- **Definition**: $W$ is the total number of basic compute operations required to solve the problem
  - $W$ is defined in terms of sequential time complexity

UNIVERSITY of HOUSTON
Isoefficiency Function

- Reformulating Parallel Runtime, Speedup and Efficiency in terms of $W$

  since $T_o = p \ T_p - T_s$ and $T_s = W$

  $T_p = (W + T_o (W,p)) / p$

  $S_p = W / T_p$

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

  $E = S / p$

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

- For a scalable system to maintain a constant $E$, $T_o/W$ has to remain constant

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

  Or

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

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

  With $K = \frac{E}{1-E}$
Isoefficiency function

• For the example adding n number on p processes, recall that
  \[ E = \frac{1}{1 + \left(2p \log p / n\right)} \]

• Thus
  \[ T_o(W,p) \approx 2p \log p \]

• The Isoefficiency function of this parallel system is therefore
  \[ W = K \cdot 2p \log p \]

• Asymptotic isoefficiency function: \(O(p \log p)\)

• Increasing the number of processing elements from \(p\) to \(p'\) the problem size has to be increased by a factor of \((p' \log p') / (p \log p)\)

Isoefficiency function

• Isoefficiency function determines the ‘ease’ with which a parallel system can maintain a constant efficiency in proportion to the number of processing elements
  - Small isoefficiency means small increments in problem size are sufficient for efficient utilization of a system
  - Large isoefficiency indicates a poorly scalable system

• Isoefficiency function does not exist for unscalable systems
  - Efficiency can not be kept constant
Minimum Execution Time

• To determine the minimum execution time, differentiate $T_p$ with respect to $p$ and set the derivative to zero.

$$\frac{dT_p}{dp} = 0$$

e.g. for adding $n$ numbers on $p$ processes with

$T_p = \frac{n}{p} \times 2 \log p$

The derivative is

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

$p = \frac{n}{2}$

And thus $T_p^{min} = 2 \log n$