Performance of Parallel Systems

• Parallelism adds one architectural dimension to those that are crucial for performance
  – traditional parameters: CPU speed, memory size, …
  – new parameter: number of processors $p$

• Suppose you have a parallel program on a parallel architecture
  – what kind of value you can expect for the run time $T_p$?
  – how good is $T_p$ with respect to the serial run time $T_s$

• Fundamental performance parameter is the speedup
  – $S = T_s / T_p$
Speedup

• Delicate points about speedup ($S$)
  – how do you get a fair measure of $T_s$?
  – what if the best serial algorithm is not known/too slow?
• Theoretically, speedup can never exceed $p$
  – speedup of $p$ obtained if each processor runs for no more than $T_s/p$
• practically, one can see superlinear speedup
  – possible reasons: nonoptimal serial algorithm, or
  – architectural features favoring parallel execution
    (example: program fitting into physical memory)
Efficiency and cost

- Efficiency ($E$) is defined as
  $$E = S / p$$

- Cost is defined as $T_P * p$ and represents the sum of the time spent by each processor executing the program.
  - Efficiency can also be expressed as ratio of serial cost to parallel cost
    $$E = T_c / p.T_p$$
  - A parallel system is said to be *cost-optimal* if the parallel cost is proportional to $T_S$
  - Since efficiency is the ratio of sequential cost to parallel cost, a cost-optimal parallel system has an efficiency of $O(1)$
Example: speedup

$\sum_{0}^{15}$

\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad 12 \quad 13 \quad 14 \quad 15 \]

- Sum of 16 numbers on a 16-processor on a hypercube
  - $T_S = \Theta (n)$, $T_P = \Theta (\log n)$
  - $S = \Theta (n / \log n)$
- Efficiency:
  - $E = S/p = 1/\log n$
- Cost: $\Theta (n \log n) \neq \Theta (n)$ => Not cost-optimal
Effect of granularity and data allocation on cost

• Most of the time there are only $p$ processors to process $n$ inputs, with $p < n$
  – mapping $n$ computations on $p$ processes is called *scaling down* the system
  – granularity of computation is increased because processors perform $n/p$ times more work

• if a system is cost-optimal with $n$ processors, it is also so when $p$ processors do the work of $n$
  – if mapping performed appropriately, both computation time and comm. time (and total run time) increase by $n/p$
Example: effect of data mapping

- When scaling down a parallel system, the choice of how to distribute data and labor is cost critical.

**Naïve approach**

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

\[ \text{Cost} = \Theta \left( n \log p \right) \]

Not cost-optimal

**Smart approach**

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

\[ \text{Cost} = \Theta \left( n + p \log p \right) \]

Cost-optimal if \( n = \Omega(p \log p) \)
Scalability

- The number of processors $p$ is an upper bound on the speedup of a parallel system.
- The scalability of a system is an indication of how good is his speedup when increasing $p$.
  - A system is scalable if efficiency can be kept at a fixed value by increasing both $p$ and problem size $n$. 
Example: Scalability

- Using the "good" algorithm:
  - $T_P = n/p + 2 \log p$
  - $S = T_S / T_P = np / (n + 2 \ p \ \log p)$
  - $E = S / p = n / (n + 2 \ \ p \ \log p)$

- Speedup limited by
  - Amdahl’s Law
  - overhead due to:
    - interprocessor communication
    - load imbalance
    - extra computation with respect to serial algorithm

<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>.80</td>
<td>.57</td>
<td>.33</td>
<td>.17</td>
</tr>
<tr>
<td>192</td>
<td>1.0</td>
<td>.92</td>
<td>.80</td>
<td>.60</td>
<td>.38</td>
</tr>
<tr>
<td>320</td>
<td>1.0</td>
<td>.95</td>
<td>.87</td>
<td>.71</td>
<td>.50</td>
</tr>
<tr>
<td>512</td>
<td>1.0</td>
<td>.97</td>
<td>.91</td>
<td>.80</td>
<td>.62</td>
</tr>
</tbody>
</table>
The overhead function

- The size \( W \) of a problem is defined as the number of basic computation steps in the best sequential algorithm on a single processor. Note that:
  \[
  W = T_s
  \]
  if we assume it takes one unit of time for each basic operation.

- The total overhead or overhead function \( T_0(W, p) \) of a parallel system is the part of the cost \( (pT_P) \) not incurred by the fastest serial algorithm:
  \[
  T_0 = p T_P - W
  \]
Example: overhead

- Using the “smart” algorithm:
  - $T_p = n/p + 2 \log p$
  - number of steps in sequential algorithm: $n - 1 \approx n$
  - $T_0 = p \left( \frac{n}{p} + 2 \log p \right) - n = 2p \log p$