Lecture 14: Analytical Modeling of Parallel Programs, part 2

Concurrent and Multicore Programming

Department of Computer Science and Engineering
Yonghong Yan
yan@oakland.edu
www.secs.oakland.edu/~yan
Topic Overview

Review

- Scalability of Parallel Systems
  - Isoefficiency Metric of Scalability
- Minimum Execution Time and Minimum Cost-Optimal Execution Time
- Asymptotic Analysis of Parallel Programs
- Other Scalability Metrics
  - Scaled speedup, Serial fraction
Parallel Execution Time

• Parallel execution time is a function of:
  – input size
  – number of processors
  – communication parameters of target platform

• Implications
  – must analyze parallel program for a particular target platform
    • communication characteristics can differ by more than $O(1)$
  – parallel program = parallel algorithm + platform
If using two processors, shouldn’t a program run twice as fast?

- Not all parts of the program are parallelized
- A number of overheads incurred when doing it in parallel
Performance Metrics: Execution Time

Does a parallel program run faster than its sequential version?

- **Serial time:** $T_s$
  - time elapsed between the start and end of serial execution

- **Parallel time:** $T_p$
  - time elapsed between first process start and last process end
Performance Metrics: Speedup

What is the benefit from increasing parallelism?

- **Speedup (S):** $T_S / T_P$
  - 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.

![Graph showing speedup vs. number of processors]
Performance Metrics: Efficiency

- Fraction of time for which a process performs useful work

\[ E = \frac{S}{\rho} = \frac{T_S}{(p \ T_P)} \]

- Bounds
  - Theoretically, \( 0 \leq E \leq 1 \)
    - The larger, the better
    - \( E=1: 0 \) overhead
  - Practically, \( E > 1 \) if superlinear speedup is achieved

- Previous example: adding \( N \) numbers using \( N \) PEs
  - Speedup: \( S = \Theta \left( \frac{N}{\log N} \right) \)
  - Efficiency: \( E = \frac{S}{N} = \Theta \left( \frac{N}{\log N} \right) / N = \Theta \left( \frac{1}{\log N} \right) \)
    - Very low when \( N \) is big
Performance Metrics: Cost

Product of parallel execution time and number of PEs: $p^* T_P$

• The total amount of time by all PEs to solve the problem

• Overhead: $T_o$
  - $T_o = T_{all} - T_S$
  - $T_o = p^* T_P - T_S$

• Cost-optimal: parallel cost $\approx$ serial cost
  - $\sim 0$ overhead
  - $E = \Theta(1)$, since $E = T_S / p^* T_P$
Topic Overview

• Introduction
• Performance Metrics for Parallel Systems
  – Execution Time, Overhead, Speedup, Efficiency, Cost
• Amdahl’s Law
• Scalability of Parallel Systems
  – Isoefficiency Metric of Scalability
• Minimum Execution Time and Minimum Cost-Optimal Execution Time
• Asymptotic Analysis of Parallel Programs
• Other Scalability Metrics
  – Scaled speedup, Serial fraction
Amdahl’s Law Speedup

\[ S(N) \leq \frac{1}{1 - F} \]
Speedup and Efficiency

Efficiency of example parallel program

- Ideal speedup
- Speedup (Sp=T1/Tp)
- Efficiency (Sp/p)
Scalability of Parallel Systems

• Strong scaling:
  – Scales with same problem size

• Weak scaling
  – Scales with increased problem size

• [https://www.sharcnet.ca/help/index.php/Measuring_Parallel_Scaling_Performance](https://www.sharcnet.ca/help/index.php/Measuring_Parallel_Scaling_Performance)
**Strong Scaling**

\[
S(p) = \frac{T(1)}{T(p)} \\
E(p) = \frac{S(p)}{p}
\]

For *ideal* parallel speedup we get:

\[
T(p) = \frac{T(1)}{p} \\
S(p) = \frac{T(1)}{T(p)} = p \\
E(p) = \frac{S(p)}{p} = 1 \text{ or } 100\%
\]

**Speedup**

- Ideal
- Super-linear
- Saturation
- Disaster

**Efficiency**

- Ideal
- 1

**Number of processors**
Weak Scalability of Parallel Systems

Extrapolate performance
• From small problems and small systems $\rightarrow$ larger problems on larger configurations

3 parallel algorithms for computing an n-point FFT on 64 PEs

Inferences from small datasets or small machines can be misleading
Scaling Characteristics of Parallel Programs

- **Efficiency:**
  \[ E = \frac{S}{p} = \frac{T_S}{pT_P} \]

- **Parallel overhead:** \( T_o = p T_P - T_S \)
  - **Overhead increases as** \( p \) **increase**

- **Problem size:**
  - **Given problem size,** \( T_S \) **remains constant**

- **Efficiency increases if**
  - **The problem size increases and**
  - **Keeping the number of PEs constant.**
Example: Adding $n$ Numbers on $p$ PEs

- Addition = 1 time unit; communication = 1 time unit

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

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

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

Speedup tends to saturate and efficiency drops
Scaling and Efficiency

- **Fixed problem size (W)**
  - fixed problem size
  - # PEs increasing
  - all parallel systems

- **Fixed number of processors (p)**
  - problem size increasing
  - # PEs fixed
  - scalable parallel systems
Scaling Characteristics of Parallel Programs

- **Overhead** \( T_o = f(T_s, p) \), i.e. problem size and \( p \)
  - In many cases, \( T_o \) grows sublinearly with respect to \( T_s \)

- **Efficiency:**
  - Decreases as we increase \( p \) -> \( T_o \)
  - Increases as we increase problem size (\( T_s \))

\[
E = \frac{1}{1 + \frac{T_o}{T_s}}
\]

- **Keep efficiency constant**
  - Increase problem sizes and
  - proportionally increasing the number of PEs

- **Scalable parallel systems**
Scalability vs Cost-Optimality

• To maintain constant efficiency Θ(1)
  – Cost-optimal == E = Θ(1)

• Any scalable parallel system can be made cost-optimal
  – Requires appropriate choice of
    • Size of the computation
    • Number of PEs
Isoefficiency Metric of Scalability

Rate at which the problem size ($T_s$) must increase per additional PE ($T_0$) to keep the efficiency fixed

- The scalability of the system
  - The slower this rate, the better scalability
  - Rate $== 0$: strong scaling.
    - The same problem (same size) scales when increasing number of PEs

- To formalize this rate, we define
  - The problem size $W =$ the asymptotic number of operations associated with the best serial algorithm to solve the problem.
  - The serial execution time, $T_s$

$$E = \frac{1}{1 + \frac{T_0}{T_s}}$$
Isoefficiency Metric of Scalability

- Parallel overhead: $T_o(W,p)$
- 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 Metric of Scalability

- To maintain constant efficiency (between 0 and 1)

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

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

\[
W = \frac{E}{1 - E} T_o(W, p).
\]

- \( K = \frac{E}{1 - E} \) is a constant related to the desired efficiency

\[
W = KT_o(W, p).
\]

Ratio \( T_o / W \) should be maintained at a constant value.
Isoefficiency Metric of Scalability

\[ W = KT_o(W, p). \]

\[ W = \Phi (p) \] such that efficiency is constant

- \( W = \Phi (p) \) is called the *isoefficiency function*
  - Read as: what is the problem size when we have \( p \) PEs to maintain constant efficiency?
  - \( W_{p+1} - W_p = \Phi (p+1) - \Phi (p) \)
    - To maintain constant efficiency, how much to increase the problem size if adding one more PE?

- *isoefficiency function* determines the ease
  - With which a parallel system maintain a constant efficiency
  - Hence achieve speedups increasing in proportion to # PEs
Adding $n$ numbers using $p$ PEs

- Parallel overhead: $T_o = 2p \log p$
- $W = KT_o(W,p)$, substitute $T_o$
  - $W = K \cdot 2p \log p$
- $K \cdot 2p \log p$ is the isoefficiency function

The asymptotic isoefficiency function for this parallel system is $\Theta(p \log p)$

- To have the same efficiency on $p'$ processors as on $p$
  - problem size $n$ must increase by $(p' \log p') / (p \log p)$ when increasing PEs from $p$ to $p'$

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

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

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

• by \( (p' \log p') / (p \log p) \)

• If \( p = 8, p' = 16 \)
  \( 16 \log 16 / (8 \log 8) = 16 \cdot 4 / (8 \cdot 3) = 8 / 3 = 2.67 \)

• 10M on 8 cores
• 10 \cdot 2.67M on 16 cores

• \( A1 \cdot x + B1 \cdot y = C1 \rightarrow A2 \cdot x + A2 \cdot (B1 / A1) \cdot y = A2 \cdot (C1 / A1) \)
• \( A2 \cdot x + B2 \cdot y = C2 \)
Isoefficiency Example 2

Add solve $n$ linear equations on $p$ processing elements

- For Gaussian elimination, execution time $= O(n^3/p + n^2 + n \log p)$
- Total parallel work $= O(n^3 + pn^2 + pn \log p)$
- Overhead $T_o$ is $O(pn^2 + pn \log p)$
  — back substitution + pivot computation

- For isoefficiency, we want $W = K T_o(W, p)$
- Expressing overhead as a function of $W = n^3$ yields
  $T_o = O(pW^{2/3} + pW^{1/3} \log p)$
- Asymptotic isoefficiency $W = K(pW^{2/3} + pW^{1/3} \log p)$
- Want the same efficiency on $p'$ processors as on $p$
  — using first term $W = KpW^{2/3} \rightarrow W = K^3 p^3$
  — using second term $W = KpW^{1/3} \log p \rightarrow W = K^{3/2} (p \log p)^{3/2}$
  — first term dominates: work must increase by $(p')^3 / p^3$
  — problem size $n$ must increase by $p'/p$
Cost-Optimality and Isoefficiency

• A parallel system is cost-optimal if and only if
  – Parallel cost == total work
  • Efficiency = 1

• From this, we have:
  – i.e. work dominates overhead

• If we have an isoefficiency function $f(p)$
  – The relation $W = \Omega(f(p))$ must be satisfied to ensure the cost-optimality of a parallel system as it is scaled up

\[
pT_P = \Theta(W).
\]

\[
W + T_o(W, p) = \Theta(W)
\]

\[
T_o(W, p) = O(W)
\]

\[
W = \Omega(T_o(W, p))
\]
Lower Bound on the Isoefficiency Function

• For a problem consisting of $W$ units of work
  – No more than $W$ PEs can be used cost-optimally.

• To maintain fixed efficiency
  – The problem size must increase at least as fast as $\Theta(p)$

• Hence, $\Omega(p)$ is the asymptotic lower bound on the isoefficiency function
  – At least one additional computation item needs to be added to maintain constant efficiency
Degree of Concurrency and Isoefficiency

• Degree of concurrency
  – The maximum number of tasks that can be executed simultaneously at any time in a parallel algorithm
  – $C(W)$ is the degree of concurrency of a parallel algorithm

• For a problem of size $W$
  – No more than $C(W)$ processing elements can be employed effectively.
Degree of Concurrency and Isoefficiency: Example

Solving a system of equation using Gaussian elimination

• N variables, $W = \Theta(n^3)$
  – $n$ variables must be eliminated one after the other
  – Eliminating each variable requires $\Theta(n^2)$ computations.

• At most $\Theta(n^2)$ PEs can be kept busy at any time.

• Since $W = \Theta(n^3)$, the degree of concurrency $C(W) = \Theta(W^{2/3})$

• Given $p$ PEs
  – The problem size should be at least $\Omega(p^{3/2})$ to use them all.
Topic Overview

• Introduction

• Performance Metrics for Parallel Systems
  – Execution Time, Overhead, Speedup, Efficiency, Cost

• Amdahl’s Law

• Scalability of Parallel Systems
  – Isoefficiency Metric of Scalability

• Minimum Execution Time

• Asymptotic Analysis of Parallel Programs

• Other Scalability Metrics
  – Scaled speedup, Serial fraction
Minimum Execution Time

- Often, we are interested in the minimum time to solution.
- To determine the minimum execution time $T_P^{\min}$ for a given $W$,
  - Differentiating the expression for $T_P$ w.r.t. $p$ and equate it to 0
    \[
    \frac{d}{dp} T_P = 0
    \]
- If $p_0$ is the value of $p$ as determined by this equation,
  - $T_P(p_0)$ is the minimum parallel time.
Minimum Execution Time: Example

Adding n numbers

- Parallel execution time:
  \[ T_P = \frac{n}{p} + 2 \log p. \]

- Compute the derivative:
  \[ \frac{\partial}{\partial p} \left( \frac{n}{p} + 2 \log p \right) = -\frac{n}{p^2} + 2 \left( \frac{1}{p} \right) \]

- Set the derivative = 0, solve for p:
  \[ -\frac{n}{p^2} + 2 \left( \frac{1}{p} \right) = 0 \]

- The corresponding exe time:
  \[ T_P^{\text{min}} = 2 \log n. \]

Note that at this point, the formulation is not cost-optimal.
Minimum Cost-Optimal Parallel Time

• The minimum cost-optimal parallel time: $T_p^{cost\_opt}$
• If the isoeficiency function of a parallel system is $\Theta(f(p))$
  – Then a problem of size $W$ can be solved cost-optimally if and only if

\[ W = \Omega(f(p)) \]

• In other words, for cost optimality, $p = O(f^{-1}(W))$
• For cost-optimal systems, $T_p = \Theta(W/p)$, therefore,

\[ T_p^{cost\_opt} = \Omega \left( \frac{W}{f^{-1}(W)} \right). \]
Adding n numbers

- The isoeficiency function $f(p)$ is $\Theta(p \log p)$.
- From this, we have $p \approx n / \log n$.
- At this processor count, the parallel runtime is:

$$T^{\text{cost-opt}}_p = \log n + \log \left( \frac{n}{\log n} \right) = 2 \log n - \log \log n.$$

- Note that both $T^{\text{min}}_p$ and $T^{\text{cost-opt}}_p$ for adding $n$ numbers are $\Theta(\log n)$. This may not always be the case.
Topic Overview

- Introduction
- Performance Metrics for Parallel Systems
  - Execution Time, Overhead, Speedup, Efficiency, Cost
- Amdahl’s Law
- Scalability of Parallel Systems
  - Isoefficiency Metric of Scalability
- Minimum Execution Time
- Asymptotic Analysis of Parallel Programs
- Other Scalability Metrics
  - Scaled speedup, Serial fraction
Asymptotic Analysis of Parallel Programs

Sorting a list of $n$ numbers.

- The fastest serial programs: $\Theta(n \log n)$.
- Four parallel algorithms, A1, A2, A3, and A4

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$n^2$</td>
<td>log $n$</td>
<td>$n$</td>
<td>$\sqrt{n}$</td>
</tr>
<tr>
<td>$T_P$</td>
<td>1</td>
<td>$n$</td>
<td>$\sqrt{n}$</td>
<td>$\sqrt{n} \log n$</td>
</tr>
<tr>
<td>$S$</td>
<td>$n \log n$</td>
<td>log $n$</td>
<td>$\sqrt{n} \log n$</td>
<td>$\sqrt{n}$</td>
</tr>
<tr>
<td>$E$</td>
<td>$\frac{\log n}{n}$</td>
<td>1</td>
<td>$\frac{\log n}{\sqrt{n}}$</td>
<td>1</td>
</tr>
<tr>
<td>$pT_P$</td>
<td>$n^2$</td>
<td>$n \log n$</td>
<td>$n^{1.5}$</td>
<td>$n \log n$</td>
</tr>
</tbody>
</table>
Asymptotic Analysis of Parallel Programs

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$n^2$</td>
<td>$\log n$</td>
<td>$n$</td>
<td>$\sqrt{n}$</td>
</tr>
<tr>
<td>$T_P$</td>
<td>1</td>
<td>$n$</td>
<td>$\sqrt{n}$</td>
<td>$\sqrt{n \log n}$</td>
</tr>
<tr>
<td>$S$</td>
<td>$n \log n$</td>
<td>$\log n$</td>
<td>$\sqrt{n \log n}$</td>
<td>$\sqrt{n}$</td>
</tr>
<tr>
<td>$E$</td>
<td>$\frac{\log n}{n}$</td>
<td>1</td>
<td>$\frac{\log n}{\sqrt{n}}$</td>
<td>1</td>
</tr>
<tr>
<td>$pT_P$</td>
<td>$n^2$</td>
<td>$n \log n$</td>
<td>$n^{1.5}$</td>
<td>$n \log n$</td>
</tr>
</tbody>
</table>

- If metric is speed ($T_P$), algorithm A1 is the best, followed by A3, A4, and A2.
- In terms of efficiency ($E$), A2 and A4 are the best, followed by A3 and A1.
- In terms of cost ($pT_P$), algorithms A2 and A4 are cost optimal, A1 and A3 are not.

- It is important to identify the analysis objectives and to use appropriate metrics!
Topic Overview

• Introduction
• Performance Metrics for Parallel Systems
  – Execution Time, Overhead, Speedup, Efficiency, Cost
• Amdahl’s Law
• Scalability of Parallel Systems
  – Isoefficiency Metric of Scalability
• Minimum Execution Time
• Asymptotic Analysis of Parallel Programs
  ➡️ Other Scalability Metrics
  – Scaled speedup, Serial fraction
Other Scalability Metrics

• A number of other metrics have been proposed, dictated by specific needs of applications.
  – For real-time applications, the objective is to scale up a system to accomplish a task in a specified time bound.
  – In memory constrained environments, metrics operate at the limit of memory and estimate performance under this problem growth rate.
Other Scalability Metrics: Scaled Speedup

• Speedup obtained when the problem size is increased linearly with the number of processing elements.
  – Per-PE problem size the same

• If scaled speedup is close to linear, the system is considered scalable.
  – Weak scaling

• If the isoefficiency is near linear, scaled speedup curve is close to linear as well.

• If the aggregate memory grows linearly in $p$, scaled speedup increases problem size to fill memory.

• Alternately, the size of the problem is increased subject to an upper-bound on parallel execution time.
Scaled Speedup: Example

\[ n \times n \text{ matrix vector multiplication} \]

• Serial execution time: \( t_c n^2 \)

• Parallel Efficiency:

\[
S = \frac{t_c n^3}{t_c \frac{n^3}{p} + t_s \log p + 2t_w \frac{n^2}{\sqrt{p}}}
\]

• Total memory requirement of this algorithm is \( \Theta(n^2) \).
Consider the case of memory-constrained scaling.

- We have $m = \Theta(n^2) = \Theta(p)$.
- Memory constrained scaled speedup is given by

$$S' = \frac{t_c c \times p}{t_c \frac{c\times p}{p} + t_s \log p + t_w \sqrt{c \times p}}$$

$$S' = O(\sqrt{p})$$

- This is not a particularly scalable system
Consider the case of time-constrained scaling.

- We have \( T_p = O(n^2) \).
- Since this is constrained to be constant, \( n^2 = O(p) \).
- Note that in this case, time-constrained speedup is identical to memory constrained speedup.
- This is not surprising, since the memory and time complexity of the operation are identical.
  - \( O(n^2) \)
Scaled Speedup: Example

\[ n \times n \text{ matrix multiplication} \]

- The serial execution time: \( t_c n^3 \).
- The parallel execution time:
  \[
  T_P = t_c \frac{n^3}{p} + t_s \log p + 2t_w \frac{n^2}{\sqrt{p}}
  \]
- Speedup:
  \[
  S = \frac{t_c n^3}{t_c \frac{n^3}{p} + t_s \log p + 2t_w \frac{n^2}{\sqrt{p}}}
  \]
Scaled Speedup: Example (continued)

Consider memory-constrained scaled speedup.

- We have memory complexity \( m = \Theta(n^2) = \Theta(p) \), or \( n^2 = c \times p \).

- At this growth rate, scaled speedup \( S' \) is given by:

\[
S' = \frac{t_c(c \times p)^{1.5}}{t_c \frac{(c \times p)^{1.5}}{p} + t_s \log p + 2t_w \frac{c \times p}{\sqrt{p}}} = O(p)
\]

- Note that this is scalable.
Consider time-constrained scaled speedup.

• We have $T_p = O(1) = O(n^3 / p)$, or $n^3 = c \times p$.

• Time-constrained speedup $S''$ is given by:

$$S'' = \frac{t_c c \times p}{t_c \frac{c \times p}{p} + t_s \log p + 2t_w \frac{(c \times p)^{2/3}}{\sqrt{p}}} = O(p^{5/6})$$

• Memory constrained scaling yields better performance.
Serial Fraction $f$

- If a computation can be divided into a totally parallel and a totally serial component, we have:

$$W = T_{ser} + T_{par}.$$ 

- From this, we have,

$$T_P = T_{ser} + \frac{T_{par}}{p}.$$ 

$$T_P = T_{ser} + \frac{W - T_{ser}}{p}.$$
Serial Fraction $f$

• The serial fraction $f$ of a parallel program is defined as:

$$f = \frac{T_{ser}}{W}.$$ 

• Therefore, we have:

$$T_P = f \times W + \frac{W - f \times W}{p}$$

$$\frac{T_P}{W} = f + \frac{1 - f}{p}$$
Serial Fraction

• Since $S = \frac{W}{T_p}$, we have

$$\frac{1}{S} = f + \frac{1 - f}{p}.$$ 

• From this, we have:

$$f = \frac{1/S - 1/p}{1 - 1/p}.$$ 

• If $f$ increases with the number of processors, this is an indicator of rising overhead, and thus an indicator of poor scalability.
Serial Fraction: Example

Consider the problem of examining the serial component of the matrix-vector product.

\[ f = \frac{t_c n^2 + t_s \log p + t_w n}{t_c n^2} \times \frac{1}{1 - 1/p} \]

We have:

\[ f \approx \frac{t_s \log p + t_w n}{t_c n^2} \]

Here, the denominator is the serial execution time and the numerator is the overhead.
• Adapted from slides “Principles of Parallel Algorithm Design” by Ananth Grama