Parallel Programming
with MPI and OpenMP

Michael J. Quinn
Chapter 7

Performance Analysis
Learning Objectives

• Predict performance of parallel programs
• Understand barriers to higher performance
Outline

• General speedup formula
• Amdahl’s Law
• Gustafson–Barsis’ Law
• Karp–Flatt metric
• Isoefficiency metric
Speedup Formula

\[
\text{Speedup} = \frac{\text{Sequential execution time}}{\text{Parallel execution time}}
\]
Execution Time Components

- Inherently sequential computations: $\sigma(n)$
- Potentially parallel computations: $\varphi(n)$
- Communication operations: $\kappa(n, \rho)$
Speedup Expression

\[ \psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n) / p + \kappa(n, p)} \]
\( \varphi(n) / \rho \)
κ(n, p)
\( \varphi(n)/\rho + \kappa(n, \rho) \)
Speedup Plot

“elbowing out”
Efficiency

\[
\text{Speedup} = \frac{\text{Sequential execution time}}{\text{Processors} \times \text{Parallel execution time}}
\]

\[
\text{Speedup} = \frac{\text{Speedup}}{\text{Processors}}
\]
\[ 0 \leq \varepsilon(n, p) \leq 1 \]

\[ \varepsilon(n, p) \leq \frac{\sigma(n) + \varphi(n)}{p \sigma(n) + \varphi(n) + p \kappa(n, p)} \]

All terms > 0 \( \Rightarrow \) \( \varepsilon(n, p) > 0 \)

Denominator > numerator \( \Rightarrow \) \( \varepsilon(n, p) < 1 \)
Amdahl’s Law

\[ \psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n) / p + \kappa(n, p)} \]

\[ \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n) / p} \]

Let \( f = \sigma(n)/(\sigma(n) + \varphi(n)) \)

\[ \psi \leq \frac{1}{f + (1 - f) / p} \]
Example 1

- 95% of a program’s execution time occurs inside a loop that can be executed in parallel. What is the maximum speedup we should expect from a parallel version of the program executing on 8 CPUs?

$$\psi \leq \frac{1}{0.05 + (1-0.05)/8} \approx 5.9$$
Example 2

• 20% of a program’s execution time is spent within inherently sequential code. What is the limit to the speedup achievable by a parallel version of the program?

\[ \lim_{p \to \infty} \frac{1}{0.2 + \frac{1 - 0.2}{p}} = \frac{1}{0.2} = 5 \]
Pop Quiz

• An oceanographer gives you a serial program and asks you how much faster it might run on 8 processors. You can only find one function amenable to a parallel solution. Benchmarking on a single processor reveals 80% of the execution time is spent inside this function. What is the best speedup a parallel version is likely to achieve on 8 processors?
Pop Quiz

• A computer animation program generates a feature movie frame-by-frame. Each frame can be generated independently and is output to its own file. If it takes 99 seconds to render a frame and 1 second to output it, how much speedup can be achieved by rendering the movie on 100 processors?
Limitations of Amdahl’s Law

- Ignores $\kappa(n,p)$
- Overestimates speedup achievable
Amdahl Effect

- Typically \( \kappa(n, p) \) has lower complexity than \( \varphi(n)/p \)
- As \( n \) increases, \( \varphi(n)/p \) dominates \( \kappa(n, p) \)
- As \( n \) increases, speedup increases
Illustration of Amdahl Effect
Review of Amdahl’s Law

• Treats problem size as a constant
• Shows how execution time decreases as number of processors increases
Another Perspective

• We often use faster computers to solve larger problem instances
• Let’s treat time as a constant and allow problem size to increase with number of processors
Gustafson–Barsis’s Law

\[ \psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n) / p} \]

Let \( s = \sigma(n)/(\sigma(n)+\varphi(n)/p) \)

\[ \psi \leq p + (1 - p)s \]
Gustafson–Barsis’s Law

- Begin with parallel execution time
- Estimate sequential execution time to solve same problem
- Problem size is an increasing function of $\rho$
- Predicts scaled speedup
Example 1

- An application running on 10 processors spends 3% of its time in serial code. What is the scaled speedup of the application?

\[ \psi = 10 + (1-10)(0.03) = 10 - 0.27 = 9.73 \]

...except 9 do not have to execute serial code

Execution on 1 CPU takes 10 times as long...
Example 2

• What is the maximum fraction of a program’s parallel execution time that can be spent in serial code if it is to achieve a scaled speedup of 7 on 8 processors?

\[ 7 = 8 + (1 - 8)s \Rightarrow s \approx 0.14 \]
Pop Quiz

• A parallel program executing on 32 processors spends 5% of its time in sequential code. What is the scaled speedup of this program?
The Karp–Flatt Metric

• Amdahl’s Law and Gustafson–Barsis’ Law ignore $\kappa(n,p)$
• They can overestimate speedup or scaled speedup
• Karp and Flatt proposed another metric
Experimentally Determined Serial Fraction

\[ e = \frac{\sigma(n) + \kappa(n, p)}{\sigma(n) + \phi(n)} \]

Inherently serial component of parallel computation + processor communication and synchronization overhead

Single processor execution time

\[ e = \frac{1/\psi - 1/p}{1 - 1/p} \]
Experimentally Determined Serial Fraction

• Takes into account parallel overhead
• Detects other sources of overhead or inefficiency ignored in speedup model
  – Process startup time
  – Process synchronization time
  – Imbalanced workload
  – Architectural overhead
Example 1

<table>
<thead>
<tr>
<th>p</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>ψ</td>
<td>1.8</td>
<td>2.5</td>
<td>3.1</td>
<td>3.6</td>
<td>4.0</td>
<td>4.4</td>
<td>4.7</td>
</tr>
</tbody>
</table>

What is the primary reason for speedup of only 4.7 on 8 CPUs?

| e  | 0.1| 0.1| 0.1| 0.1| 0.1| 0.1| 0.1|

Since $e$ is constant, large serial fraction is the primary reason.
Example 2

<table>
<thead>
<tr>
<th>p</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>ψ</td>
<td>1.9</td>
<td>2.6</td>
<td>3.2</td>
<td>3.7</td>
<td>4.1</td>
<td>4.5</td>
<td>4.7</td>
</tr>
</tbody>
</table>

What is the primary reason for speedup of only 4.7 on 8 CPUs?

| e  | 0.070 | 0.075 | 0.080 | 0.085 | 0.090 | 0.095 | 0.100 |

Since $e$ is steadily increasing, overhead is the primary reason.
Pop Quiz

- Is this program likely to achieve a speedup of 10 on 12 processors?
Isoefficiency Metric

- Parallel system: parallel program executing on a parallel computer
- Scalability of a parallel system: measure of its ability to increase performance as number of processors increases
- A scalable system maintains efficiency as processors are added
- Isoefficiency: way to measure scalability
Isoefficiency Derivation Steps

• Begin with speedup formula
• Compute total amount of overhead
• Assume efficiency remains constant
• Determine relation between sequential execution time and overhead
Deriving Isoefficiency Relation

Determine overhead

\[ T_0(n, p) = (p - 1)\sigma(n) + p\kappa(n, p) \]

Substitute overhead into speedup equation

\[ \psi(n, p) \leq \frac{p(\sigma(n) + \varphi(n))}{\sigma(n) + \varphi(n) + T_0(n, p)} \]

Substitute \( T(n, 1) = \sigma(n) + \varphi(n) \). Assume efficiency is constant.

\[ T(n, 1) \geq C T_0(n, p) \quad \text{Isoefficiency Relation} \]
Scalability Function

- Suppose isoefficiency relation is $n \geq f(p)$
- Let $M(n)$ denote memory required for problem of size $n$
- $M(f(p))/p$ shows how memory usage per processor must increase to maintain same efficiency
- We call $M(f(p))/p$ the scalability function
Meaning of Scalability Function

• To maintain efficiency when increasing $\rho$, we must increase $n$

• Maximum problem size limited by available memory, which is linear in $\rho$

• Scalability function shows how memory usage per processor must grow to maintain efficiency

• Scalability function a constant means parallel system is perfectly scalable
Interpreting Scalability Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Number of processors</th>
<th>Memory needed per processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_p$</td>
<td></td>
<td>Cannot maintain efficiency</td>
</tr>
<tr>
<td>$C_p \log p$</td>
<td></td>
<td>Cannot maintain efficiency</td>
</tr>
<tr>
<td>$C \log p$</td>
<td></td>
<td>Can maintain efficiency</td>
</tr>
<tr>
<td>$C$</td>
<td></td>
<td>Can maintain efficiency</td>
</tr>
</tbody>
</table>

Number of processors

Memory Size

Memory needed per processor
Example 1: Reduction

• Sequential algorithm complexity
  \( T(n, 1) = \Theta(n) \)

• Parallel algorithm
  – Computational complexity = \( \Theta(n/p) \)
  – Communication complexity = \( \Theta(\log p) \)

• Parallel overhead
  \( T_0(n, p) = \Theta(p \log p) \)
Reduction (continued)

- Isoefficiency relation: \( n \geq C_p \log p \)
- We ask: To maintain same level of efficiency, how must \( n \) increase when \( p \) increases?
- \( M(n) = n \)

\[
M(Cp \log p) / p = Cp \log p / p = C \log p
\]

- The system has good scalability
Example 2: Floyd’s Algorithm

- Sequential time complexity: $\Theta(n^3)$
- Parallel computation time: $\Theta(n^3/p)$
- Parallel communication time: $\Theta(n^2\log p)$
- Parallel overhead: $T_0(n,p) = \Theta(pn^2\log p)$
Floyd’s Algorithm (continued)

• Isoefficiency relation
  \[ n^3 \geq C (p \ n^3 \log p) \Rightarrow n \geq C \ p \log p \]

• \( M(n) = n^2 \)

\[
M(Cp\log p)/p = C^2 \ p^2 \log^2 p / p = C^2 \ p \log^2 p
\]

• The parallel system has poor scalability
Example 3: Finite Difference

- Sequential time complexity per iteration: $\Theta(n^2)$
- Parallel communication complexity per iteration: $\Theta(n/\sqrt{p})$
- Parallel overhead: $\Theta(n \sqrt{p})$
Finite Difference (continued)

- Isoefficiency relation
  \[ n^2 \geq C \sqrt[p]{p} \Rightarrow n \geq C \sqrt[p]{p} \]
- \[ M(n) = n^2 \]

\[
M(C \sqrt{p}) / p = C^2 p / p = C^2
\]

- This algorithm is perfectly scalable
Summary (1/3)

• Performance terms
  – Speedup
  – Efficiency

• Model of speedup
  – Serial component
  – Parallel component
  – Communication component
Summary (2/3)

• What prevents linear speedup?
  – Serial operations
  – Communication operations
  – Process start–up
  – Imbalanced workloads
  – Architectural limitations
Summary (3/3)

• Analyzing parallel performance
  – Amdahl’s Law
  – Gustafson–Barsis’ Law
  – Karp–Flatt metric
  – Isoefficiency metric