Performance analysis

Goals are

• to be able to understand better why your program has the performance it has, and

• what could be preventing its performance from being better.
Speedup

\[ \text{speedup} \leq \frac{T_s}{T_P(p)} \]

\[ \text{speedup} \leq \frac{\text{serial time}}{\text{parallel time}} \]

- Parallel time \( T_P(p) \) is the time it takes the parallel form of the program to run on \( p \) processors
Speedup

\[ speedup \leq \frac{T_s}{T_p(p)} \]

\[ speedup \leq \frac{\text{serial time}}{\text{parallel time}} \]

- Sequential time \( T_s \) is more problematic
  - Can be \( T_p(1) \), but this carries the overhead of extra code needed for parallelization. Even with one thread, OpenMP code will call libraries for threading. **One way to “cheat” on benchmarking.**
  - Should be the best possible sequential implementation: tuned, good or best compiler switches, etc.
  - Best possible sequential implementation may not exist for a problem size
The typical *speedup* curve - fixed problem size
A typical speedup curve - problem size grows with number of processors, if the program has good weak scaling.
What is execution time?

• Execution time can be modeled as the sum of:

1. Inherently sequential computation $\sigma(n)$
2. Potentially parallel computation $\phi(n)$
3. Communication time $\kappa(n,p)$
Components of execution time

Inherently Sequential Execution time

execution time

number of processors
Components of execution time

*Parallel time*

execution time

number of processors
Components of execution time

Communication time and other parallel overheads

\[ \kappa(P) \alpha \lfloor \log_2 P \rfloor \]

Execution time

Number of processors
Components of execution time

*Sequential time*

**Sequential time**

- **Execution time**
  - Speedup $< 1$
  - Speedup $= 1$
  - Maximum speedup

At some point, the decrease in parallel execution time of the parallel part is less than the increase in communication costs, leading to the knee in the curve.
Speedup as a function of these components

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

- Sequential time is
  i. the sequential computation \((\sigma(n))\)
  ii. the parallel computation \((\Phi(n))\)
- Parallel time is
  iii. the sequential computation time \((\sigma(n))\)
  iv. the parallel computation time \((\Phi(n)/p)\)
  v. the communication cost \((\kappa(n,p))\)
Efficiency

\[ \text{efficiency} \leq \frac{\text{sequential execution time}}{\text{num processors} \times \text{parallel execution time}} \]

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

\[ \epsilon(n, p) \leq \frac{\sigma(n) + \phi(n)}{p\sigma(n) + \phi(n) + p\kappa(n, p)} \]

Intuitively, \textit{efficiency} is how effectively the machines are being used by the parallel computation.

If the number of processors is doubled, for the efficiency to stay the same the parallel execution time \( T_p \) must be halved.

\[ 0 < \epsilon(n, p) < 1 \]

all terms > 0,
\[ \epsilon(n, p) > 0 \]

numerator \leq denominator \leq 1
Efficiency

denominator is the total processor time used in parallel execution

\[ \text{efficiency} \leq \frac{\text{sequential execution time}}{\text{num processors} \times \text{parallel execution time}} \]

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

\[ \epsilon(n, p) \leq \frac{\sigma(n) + \phi(n)}{p\sigma(n) + \phi(n) + p\kappa(n,p)} \]
Efficiency by amount of work

$\Phi$: amount of computation that can be done in parallel

$\kappa$: communication overhead

$\sigma$: sequential computation
Amdahl’s Law

• Developed by Gene Amdahl

• Basic idea: the parallel performance of a program is limited by the sequential portion of the program

• argument for fewer, faster processors

• Can be used to model performance on various sizes of machines, and to derive other useful relations.
Gene Amdahl

- Worked on IBM 704, 709, Stretch and 7030 machines
- Stretch was first transistorized computer, fastest from 1961 until CDC 6600 in 1964, 1.2 MIPS
- Multiprogramming, memory protection, generalized interrupts, the 8-bit byte, Instruction pipelining, prefetch and decoding introduced in this machine
- Worked on IBM System 360
Gene Amdahl

- In technical disagreement with IBM, set up Amdahl Computers to build plug-compatible machines -- later acquired by Hitachi
- Amdahl's law came from discussions with Dan Slotnick (Illiac IV architect at UIUC) and others about future of parallel processing
Power density too high to keep junctions at low temp

Courtesy, Intel
Oxen and killer micros

• Seymour Cray’s comments about preferring 2 oxen over 1000 chickens was in agreement with what Amdahl suggested.

• Flynn’s *Attack of the killer micros*, Supercomputing talk in 1990 why special purpose vector machines would lose out to large numbers of more general purpose machines

• GPUs are can be thought of as a return from the dead of special purpose hardware
The first characteristic of interest is the fraction of the computational load which is associated with data management housekeeping. This fraction has been very nearly constant for about ten years, and accounts for 40% of the executed instructions in production runs. In an entirely dedicated special purpose environment this might be reduced by a factor of two, but it is highly improbable that it could be reduced by a factor of three. The nature of this overhead appears to be sequential so that it is unlikely to be amenable to parallel processing techniques. Overhead alone would then place an upper limit on throughput of five to seven times the sequential processing rate, even if the housekeeping were done in a separate processor. The non housekeeping part of the problem could exploit at most a processor of performance three to four times the performance of the housekeeping processor. A fairly obvious conclusion which can be drawn at this point is that the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude.
Amdahl’s law - key insight

With perfect utilization of parallelism on the parallel part of the job, must take at least $T_{\text{serial}}$ time to execute. This observation forms the motivation for Amdahl’s law.

As $p \to \infty$, $T_{\text{parallel}} \to 0$ and $\psi(\infty) \to (T_{\text{total work}})/T_{\text{serial}}$. Thus, $\psi$ is limited by the serial part of the program.

\[
\psi(1) = \frac{T_{\text{total work}}}{T_{\text{serial}} + T_{\text{parallel}}}
\]

\[
\psi(\infty) = \frac{T_{\text{total work}}}{T_{\text{serial}}}
\]

$\psi(p)$: speedup with $p$ processors
Two measures of speedup

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

Takes into account communication cost.

• \( \sigma(n) \) and \( \phi(n) \) are arguably fundamental properties of a program

• \( \kappa(n,p) \) is a property of both the program, the hardware, and the library implementations -- arguably a less fundamental concept.

• Can formulate a meaningful, but optimistic, approximation to the speedup without \( \kappa(n,p) \)

\[ \psi(n, p) \leq \frac{\sigma(n) + \phi(n)}{p\sigma(n) + \phi(n)} \]
Speedup in terms of the serial fraction of a program

Given this formulation on the previous slide, the fraction of the program that is serial in a sequential execution is

\[ f \leq \frac{\sigma(n)}{\sigma(n)+\phi(n)} \]

Speedup can be rewritten in terms of \( f \):

This gives us Amdahl’s Law.

\[ \psi(p) \leq \frac{1}{f+(1-f)/p} \]
\textbf{Amdahl's Law }\implies \textbf{speedup}

\begin{align*}
\text{speedup} & = \frac{1}{f + (1 - f)/p} \\
& = \frac{1}{\frac{\sigma(n)}{\sigma(n) + \phi(n)} + \left(1 - \frac{\sigma(n)}{\sigma(n) + \phi(n)}\right)/p} \\
& = \frac{\sigma(n) + \phi(n)}{\sigma(n) + \phi(n)} \cdot \frac{1}{\frac{\sigma(n)}{\sigma(n) + \phi(n)} + \left(1 - \frac{\sigma(n)}{\sigma(n) + \phi(n)}\right)/p} \\
& = \frac{\sigma(n) + \phi(n)}{\sigma(n) + \sigma(n) + \phi(n) - \sigma(n))/p} \\
& = \frac{\sigma(n) + \phi(n)}{\sigma(n) + \phi(n)/p}
\end{align*}
Example of using Amdahl’s Law

A program is 90% parallel. What speedup can be expected when running on four, eight and 16 processors?

\[
\psi(p) \leq 3.077 = \frac{1}{0.1 + (1 - 0.1)/4}
\]

\[
\psi(p) \leq 4.71 = \frac{1}{0.1 + (0.9)/8}
\]

\[
\psi(p) \leq 6.4 = \frac{1}{0.1 + (0.9)/16}
\]
What is the efficiency of this program?

\[ \varepsilon(p) \leq 0.769 = \frac{3.077}{4} \]

\[ \varepsilon(p) \leq 0.589 = \frac{4.71}{8} \]

\[ \varepsilon(p) \leq 0.4 = \frac{6.4}{16} \]

A 2\times increase in machine cost gives you a 1.4\times increase in performance.

And this is optimistic since communication costs are not considered.
Another Amdahl’s Law example

A program is 20% inherently serial. Given 2, 16 and infinite processors, how much speedup can we get?

\[
\psi(p) \leq 1.67 = \frac{1}{0.2 + (0.8)/2}
\]

\[
\psi(p) \leq 4 = \frac{1}{0.2 + (0.8)/16}
\]

\[
\psi(p) \leq 5 = \frac{1}{0.2 + (0.8)/\infty}
\]
Effect of Amdahl’s Law

Limitation of Amdahl’s Law

This result is a limit, not a realistic number.

The problem is that communication costs \(\kappa(n,p)\) are ignored, and this is an overhead that is worse than fixed (which \(f\) is), but actually grows with the number of processors.

Amdahl’s Law is too optimistic and may target the wrong problem
No communication overhead

- execution time

- speedup = 1

- maximum speedup

- number of processors
$O(\log_2 P)$ communication costs

Execution time

Speedup $= 1$

Maximum speedup

Number of processors
O(P) Communication Costs

- **Speedup** = 1
- Maximum speedup

- Execution time
- Number of processors
Amdahl Effect

• Complexity of $\phi(n)$ usually higher than complexity of $\kappa(n,p)$ (i.e. computational complexity usually higher than complexity of communication -- same is often true of $\sigma(n)$ as well.) $\phi(n)$ usually $O(n)$ or higher

• $\kappa(n,p)$ often $O(1)$ or $O(\log_2 P)$

• Increasing $n$ allows $\phi(n)$ to dominate $\kappa(n,p)$

• Thus, increasing the problem size $n$ increases the speedup $\Psi$ for a given number of processors

• Another “cheat” to get good results -- make $n$ large

• Most benchmarks have standard sized inputs to preclude this
Amdahl Effect

Speedup

Number of processors

- $n=100000$
- $n=10000$
- $n=1000$
Amdahl Effect both increases speedup and moves the knee of the curve to the right.
Summary

• Allows speedup to be computed for
  • fixed problem size \( n \)
  • varying number of processes
• Ignores communication costs
• Is optimistic, but gives an upper bound
Gustafson-Barsis’ Law

How does speedup scale with larger problem sizes?

Given a fixed amount of time, how much bigger of a problem can we solve by adding more processors?

Large problem sizes often correspond to better resolution and precision on the problem being solved.
Basic terms

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

Because \( \kappa(n,p) > 0 \),
\[ \psi(n, p) \leq \frac{\sigma(n) + \phi(n)}{\sigma(n) + \phi(n)/p} \]

Let \( s \) be the fraction of time in a parallel execution of the program that is spent performing sequential operations.

Then, \( (1-s) \) is the fraction of time spent in a parallel execution of the program performing parallel operations.
Note that Amdahl's Law looks at the sequential and parallel parts of the program for a given problem size, and the value of $f$ is the fraction in a sequential execution that is inherently sequential, and so

$$f \leq \frac{\sigma(n)}{\sigma(n)+\phi(n)}$$

$$\psi(p) \leq \frac{1}{f+(1-f)/p}$$

Note number of processors not mentioned for definition of $f$ because $f$ is for time in a sequential run.
Some definitions

The sequential part of a **parallel** computation:

\[
S = \frac{\sigma(n)}{\sigma(n) + \phi(n)/p}
\]

The parallel part of a parallel computation:

\[
(1 - s) = \frac{\phi(n)/p}{\sigma(n) + \phi(n)/p}
\]

And the speedup:

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

In terms of \(s\), \(\Psi(p) = p - (1-p)s\)
Difference between Gustafson-Barsis (G-B) Law and Amdahl’s Law

The serial portion in Amdahl’s law is a fraction of the total execution time of the program.

The serial portion in G-B is a fraction of the parallel execution time of the program. To use G-B Law we assume work scales to maintain value of $s$. 

\[
f \leq \frac{\sigma(n)}{\sigma(n) + \phi(n)}
\]

\[
s = \frac{\sigma(n)}{\sigma(n) + \phi(n)/p}
\]
No communication overhead

Gustafson-Barsis $\Phi(n)/P$, $n$ scales with $P$

Amdahl’s Law $\Phi(n)/P$, $n$ constant

G-B, Amdahl’s law, sequential portion $\sigma(n)$. Note that as $n$ increases with $P$ for G-B, $\sigma(n)$ also increases (not shown here), but the ratio $s$ stays the same.
Deriving G-B Law

First, we show that the formula circled in blue leads to our speedup formula.
Deriving G-B Law

Second, we show that the formula circled in blue (that we just showed is equivalent to speedup) leads to the G-B Law formula.

\[
\psi(n, p) \leq \frac{(\sigma(n) + \frac{\phi(n)}{p}) (s + (1 - s)p)}{\sigma(n) + \frac{\phi(n)}{p}}
\]

\[
\psi(n, p) \leq s + (1 - s)p
\]

\[
\psi(n, p) \leq p + (1 - p)s
\]

\[
s + (1 - s)p = s + p - pS
\]

\[
= p + (1 - p)s
\]
An example

An application executing on 64 processors requires 220 seconds to run. It is experimentally determined through benchmarking that 5% of the time is spent in the serial code on a single processor. What is the scaled speedup of the application?

\[ s = 0.05, \text{ thus on 64 processors} \]
\[ \Psi = 64 + (1-64)(0.05) = 64 - 3.15 = 60.85 \]
Another way of looking at this result: given P processors, P amount of useful work can be done. However, on P-1 processors there is time wasted due to the sequential part that must be subtracted out from the useful work.

\[ s = 0.05, \text{ thus on 64 processors} \]
\[ \Psi = 64 + (1-64)(0.05) = 64 - 3.15 = 60.85 \]
Second example

You have money to buy a 16K (16,384) core distributed memory system, but you only want to spend the money if you can get decent performance on your application.

Allowing the problem to scale with increasing numbers of processors, what must $s$ be to get a scaled speedup of 15,000 on the machine, i.e. what fraction of the application's parallel execution time can be devoted to inherently serial computation?

\[
15,000 = 16,384 - 16,383s
\]

\[
\Rightarrow s = \frac{1,384}{16,383}
\]

\[
\Rightarrow s = 0.084
\]
Comparison with Amdahl’s Law result

\[ \psi(n,p) \leq p + (1 - p)s \]

\[ 15,000 = 16,384 - 16,383s \]

\[ \Rightarrow s = \frac{1,384}{16,383} \]

\[ \Rightarrow s = 0.084 \]

G-B almost 1% can be sequential

Amdahl's law (56 millionths)
Comparison with Amdahl’s Law

\( \psi(n,p) \leq p + (1 - p)s \)

\[ 15,000 = 16,384 - 16,383s \]

\[ \Rightarrow s = \frac{1,384}{16,383} \]

\[ \Rightarrow s = 0.084 \]

But then Amdahl's law doesn't allow the problem size to scale.
Non-scaled performance

\[ \sigma(1) = \sigma(p); \phi(1) = \phi(p) \]

Work is constant, speedup levels off at \(~256\) processors
Even though it is hard to see, as the parallel work increases proportionally to the number of processors, the speedup scales proportionally to the number of processors.

\[ \sigma(1) = \sigma(p); \ p \phi(1) = \phi(p) \]
performance

\[ \sigma(l) = \sigma(p); \quad p \cdot \phi(l) = \phi(p) \]

Note that the parallel work may (and usually does) increase faster than the problem size.
Scaled speedups, log scales

\[ \sigma(1) = \sigma(p); \ p \cdot \phi(1) = \phi(1) \]

The same chart as before, except log scales for parallel work and speedup.

Scaled speedup close to ideal
The effect of un-modeled $\log_2 P$ communication

This is clearly an important effect that is not being modeled.
The Karp-Flatt Metric

• Takes into account communication costs

• \[ T(n,p) = \sigma(n) + \phi(n)/p + \kappa(n,p) \]

• Serial time \[ T(n,1) = \sigma(n) + \phi(n) \]

• The experimentally determined serial fraction \( e \) of the parallel computation is

\[
e = (\sigma(n) + \kappa(n,p))/T(n,1)
\]
\[ e = \frac{\sigma(n) + \kappa(n,p)}{T(n,1)} \]

- \( e \) is the fraction of the one processor execution time that is serial on all \( p \) processors.

- Communication cost mandates measuring at a given processor count.

- This is because communication cost is a function of theoretical limits and implementation.
The experimentally determined serial fraction \( e \) of the parallel computation is
\[
e = \frac{\sigma(n) + \kappa(n,p)}{T(n,1)}
\]
\[
e \cdot T(n,1) = \sigma(n) + \kappa(n,p)
\]

The parallel execution time
\[
T(n,p) = \sigma(n) + \frac{\phi(n)}{p} + \kappa(n,p)
\]
can now be rewritten as
\[
T(n,p) = T(n,1) \cdot e + T(n,1)(1 - e)/p
\]

Let \( \psi \) represent \( \psi(n,p) \), and
\[
\psi = \frac{T(n,1)}{T(n,p)}
\]
then
\[
T(n,1) = T(n, p)\psi.
\]

Therefore
\[
T(n,p) = T(n,p)\psi e + T(n,p)\psi(1-e)/p
\]
The experimentally determined serial fraction $e$ of the parallel computation is

$$e = \frac{\sigma(n) + \kappa(n,p)}{T(n,1)}$$

$$e \cdot T(n,1) = \sigma(n) + \kappa(n,p)$$

The parallel execution time

$$T(n,p) = \sigma(n) + \phi(n)/p + \kappa(n,p)$$

can now be rewritten as

$$T(n,p) = T(n,1) \cdot e + T(n,1)(1 - e)/p$$

Let $\psi$ represent $\psi(n,p)$, and

$$\psi = \frac{T(n,1)}{T(n,p)}$$

then

$$T(n,1) = T(n, p)\psi.$$ 

Therefore

$$T(n,p) = T(n,p)\psi e + T(n,p)\psi(1-e)/p$$
The experimentally determined serial fraction $e$ of the parallel computation is

$$e = (\sigma(n) + \kappa(n,p))/T(n,1)$$
$$e \cdot T(n,1) = \sigma(n) + \kappa(n,p)$$

The parallel execution time

$$T(n,p) = \sigma(n) + \phi(n)/p + \kappa(n,p)$$

can now be rewritten as

$$T(n,p) = T(n,1) \cdot e + T(n,1)(1 - e)/p.$$ 

Let $\psi$ represent $\psi(n,p)$, and

$$\psi = T(n,1)/T(n,p)$$

then

$$T(n,1) = T(n, p)\psi.$$ 

Therefore

$$T(n,p) = T(n,p)\psi e + T(n,p)\psi(1-e)/p.$$
The experimentally determined serial fraction $e$ of the parallel computation is

$$e = (\sigma(n) + \kappa(n,p))/T(n,1)$$

$$e \cdot T(n,1) = \sigma(n) + \kappa(n,p)$$

The parallel execution time

$$T(n,p) = \sigma(n) + \phi(n)/p + \kappa(n,p)$$

can now be rewritten as

$$T(n,p) = T(n,1) \cdot e + T(n,1)(1 - e)/p$$

Let $\psi$ represent $\psi(n,p)$, and

$$\psi = T(n,1)/T(n,p)$$

then

$$T(n,1) = T(n, p)\psi.$$ 

Therefore

$$T(n,p) = T(n,p)\psi e + \frac{T(n,p)\psi (1-e)}{p}$$
Karp-Flatt Metric

\[ T(n,p) = T(n,p)\psi e + T(n,p)\psi(1-e)/p \Rightarrow \]
\[ 1 = \psi e + \psi(1-e)/p \Rightarrow \]
\[ 1/\psi = e + (1-e)/p \Rightarrow \]
\[ 1/\psi = e + 1/p - e/p \Rightarrow \]
\[ 1/\psi = e(1-1/p) +1/p \Rightarrow \]

\[ e = \frac{1/\psi - 1/p}{1-1/p} \]
What is it good for?

- Takes into account the parallel overhead ($\kappa(n,p)$) ignored by Amdahl’s Law and Gustafson-Barsis.
- Helps us to detect other sources of inefficiency ignored in these (sometimes too simple) models of execution time.
  - $\phi(n)/p$ may not be accurate because of load balance issues or work not dividing evenly into $c \cdot p$ chunks.
  - other interactions with the system may be causing problems.
- Can determine if the efficiency drop with increasing $p$ for a fixed size problem is
  a. because of limited parallelism
  b. because of increases in algorithmic or architectural overhead
Example

Benchmarking a program on 1, 2, ..., 8 processors produces the following speedups:

<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>$\psi$</td>
<td>1.82</td>
<td>2.5</td>
<td>3.08</td>
<td>3.57</td>
<td>4</td>
<td>4.38</td>
<td>4.71</td>
</tr>
</tbody>
</table>

Why is the speedup only 4.71 on 8 processors?

<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>$\psi$</td>
<td>1.82</td>
<td>2.5</td>
<td>3.08</td>
<td>3.57</td>
<td>4</td>
<td>4.38</td>
<td>4.71</td>
</tr>
<tr>
<td>$e$</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

\[ e = \frac{(1/3.57 - 1/5)/(1-1/5)}{0.08} = 0.1 \]
Example 2

Benchmarking a program on 1, 2, ..., 8 processors produces the following speedups:

<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>$\psi$</td>
<td>1.87</td>
<td>2.61</td>
<td>3.23</td>
<td>3.73</td>
<td>4.14</td>
<td>4.46</td>
<td>4.71</td>
</tr>
</tbody>
</table>

Why is the speedup only 4.71 on 8 processors?

<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>$\psi$</td>
<td>1.87</td>
<td>2.61</td>
<td>3.23</td>
<td>3.73</td>
<td>4.14</td>
<td>4.46</td>
<td>4.71</td>
</tr>
<tr>
<td>$e$</td>
<td>0.07</td>
<td>0.07</td>
<td>0.08</td>
<td>0.08</td>
<td>0.09</td>
<td>0.09</td>
<td>0.1</td>
</tr>
</tbody>
</table>

$e$ is increasing: speedup problem is increasing serial overhead (process startup, communication, algorithmic issues, the architecture of the parallel system, etc.)
Which has the efficiency problem?

![Graph showing speedup 1 and speedup 2 over different values.](image)
Very easy to see using $e$
Isoefficiency Metric Overview

• 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

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

Substitute overhead into speedup equation

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

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

\[ T(n, 1) \geq C T_o(n, p) \]

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 \( p \), we must increase \( n \)

• Maximum problem size limited by available memory, which is linear in \( p \)

• 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

- Memory Size per node
- Number of processors

Can maintain efficiency
- Cannot maintain efficiency

Memory needed per processor

Lines:
- Cp
- Clogp
- Cplogp
- C
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 \( T_0(n,p) = \Theta(p \log p) \)
    - \( p \) term because \( p \) processors involved in the reduction for \( \log p \) time.
Reduction (continued)

• Isoefficiency relation: \( n \geq C \, p \, \log p \)

• We ask: To maintain same level of efficiency, how must \( n \), the problem size, 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
  \[ n^3 \geq C(p \ n^2 \ \log p) \Rightarrow n \geq C \ p \ \log p \]

• \( M(n) = n^2 \)

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

• 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
  \[ n^2 \geq Cn\sqrt{p} \Rightarrow n \geq C\sqrt{p} \]

- \( M(n) = n^2 \)

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

- 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