Performance Metrics

- Speedup
- Efficiency
- Work
- Scaled speedup
Speedup

- A problem is ideally solvable faster with multiple processors.
- Speedup = \frac{\text{time to solve sequentially}}{\text{time to solve in parallel}}

- Some qualification is necessary:
  - Are the sequential and parallel times necessarily using the same algorithm?
Algorithm Dependence

- Not all algorithms parallelize equally well.
- Parallel execution may introduce overheads not present in sequential execution of a given algorithm.
- Rigorous definition of speedup demands that parallel execution of a given algorithm be compared against serial execution using the “best” sequential algorithm.
- Often this is not done: instead, the same algorithm is used for both sequential and parallel.
Speedup as a function of the number of processors

- Let $T_p$ be the time required to solve a given problem using $p$ processors.
- Let $S_p$ be the speedup using $p$ processors.
- $S_p = T_1 / T_p$
- Ideally $S_p = p$
- The ideal is difficult to achieve.
Amdahl’s Law
(Gene Amdahl, 1967)

- Amdahl’s law expresses a reason why ideal speedup may not be achieved.
- Make the idealized assumption that the code to be executed consists of
  - A perfectly-parallel portion (capable of using all $p$ processors efficiently), and
  - A strictly-sequential portion (capable of using only 1 processor).
- Let $f$ be the fraction of instructions that fall into the strictly-sequential category.
Amdahl’s Law (2)

- Ideally, $f$ is low. If $f$ is 0, perfect speedup can be expected, while if $f$ is high, speedup will be near 1.

- What is unexpected is how quickly speedup drops off as a function of $f$. 
Amdahl’s Law (3)

Program Execution Profile

Sequential portion  Parallel portion

Sequential fraction = \( f = \frac{a}{{a + p \cdot b}} \)

\( T_p = a + b \)

\( T_1 = a + pb \)

Sequential fraction = \( f = \frac{a}{{a + p \cdot b}} \)
Amdahl’s Law (4)

Sequential fraction = f =  \( a / (a + p \times b) \)
Therefore f*(a + p*b) = a
Therefore b = \( (a/f - a)/p = (a/p)(1/f - 1) \)
Therefore \( b/a = (1/f - 1)/p \)

\[
\text{Speedup} = \frac{T_1}{T_p} = \frac{(a+pb)}{(a+b)} \\
= \frac{(1+pb/a)}{(1+b/a)} \\
= \frac{(1 + (1/f - 1)/(1+(1/f-1)/p)}{1+(1/f-1)/p)} \\
= \frac{(1/f)/(1+(1/f-1)/p)}
\]

\[\text{Speedup} = 1/(f + (1-f)/p)\]  
Amdahl’s Law, where f is sequential
Amdahl’s Law (5)

\[
\text{Speedup} = \frac{1}{f + \frac{1-f}{p}}
\]

Limiting cases and examples:

- \( p \to 1 \): Speedup \( \to 1 \)
- \( p \to \infty \): Speedup \( \to \frac{1}{f} \)
- \( f \to 0 \): Speedup \( \to p \)
- \( f \to 1 \): Speedup \( \to 1 \)
- \( f = 0.1 \): Speedup \( < 10 \)
Amdahl’s Law (6)

To lift the ceiling on speedup, we need to decrease $f$. 

\[
\text{Speedup} = \frac{1}{f} 
\]
Due to **overhead** of supporting more processors, speedup may actually **decrease** after a point:
Effort

- The effort used by a parallel processor in executing a program is the product of:
  - the elapsed time, and
  - the number of processors used

- Included in effort are processors that are used for some portion of the computation, but which are idle for other portions.
Effort (2)

- Ideally the effort is the same regardless of the number of processors.
- In practice, the effort tends to go up with more processors, due to:
  - Overhead in spawning parallel processes
  - Communication overhead
  - Some processors being idle part of the time
- Usually we are willing to sacrifice some effort to attain speedup.
Efficiency

- How well are the parallel processors being utilized?
  - If there are $p$ processors, with parallel execution time $T_p$, then the effort is $p \cdot T_p$.
  - The actual “work” done is $T_1$, the time it would take to do the work on one processor.
  - Therefore,
    \[
    \text{Efficiency} = \frac{T_1}{p \cdot T_p}
    \]
    which happens to be equal to Speedup / $p$. 
Ideal Efficiency

- The ideal efficiency is 1, with actual practice being somewhat worse, due to the additional effort mentioned earlier.

- We shouldn’t be lulled into thinking that efficiency is how busy the processors are, because they could be doing work that is parallel overhead.
Gustafson’s “Law”  
(John Gustafson, 1988)

- Gustafson tried to refute Amdahl’s law, which assumes that we are interested in applying ever larger numbers of processors to a fixed-sized problem.
- In practice, we are only interested in applying more processors as the size of the problem scales.
- Moreover, scaling the problem usually scales the parallel part disproportionately.
Gustafson’s “Law” (John Gustafson, 1988)

- Gustafson tried to refute Amdahl’s law, which assumes that we are interested in applying ever larger numbers of processors to a fixed-sized problem.
- In practice, we are only interested in applying more processors as the size of the problem scales.
- Moreover, scaling the problem usually scales the parallel part disproportionately.
Gustafson’s “Law”
(John Gustafson, 1988)

- Gustafson tried to refute Amdahl’s law, which assumes that we are interested in applying ever larger numbers of processors to a fixed-sized problem.
- In practice, we are only interested in applying more processors as the size of the problem scales.
- Moreover, scaling the problem usually scales the parallel part disproportionately.
Let $n$ be a measure of the problem size.

The execution of the program on a parallel computer is decomposed into

$$a(n) + b(n) = 1$$

where $a$ is the sequential fraction and $b$ the parallel fraction (ignoring overhead for now).

On a sequential computer, the relative time would be $a(n) + pb(n)$ where $p$ is the number of processors in the parallel case.
Gustafson’s Law (3)

- Speedup is therefore
  \[(a(n) + pb(n)) \text{ (relative to } a(n)+b(n) = 1)\]

  \[= a(n)+p*(1-a(n))\]

  where \(a(n)\) is the serial fraction.

- Assuming the serial fraction \(a(n)\) diminishes with problem size \(n\), then speedup approaches \(p\) as \(n \to \infty\) as desired.

- Thus Gustafson’s law seems to rescue parallel processing from Amdahl’s law.
Granularity Considerations

- Roughly speaking, **granularity** means the ratio of computation interval to communication time needed to achieve a reasonable speedup.

- If a process needs to communicate frequently with other processes, then the communication must be very fast or the process’ waiting time will absorb the speedup from parallel execution.
Granularity (2)

- Finer granularity is better, since it provides more ways to distribute the work.
- Imagine that the computation work load is a 10 kg. of material:
  - Sand = fine-grain
  - Cinder blocks (with or without warts) = coarse grain
- Which is easier to distribute?
Granularity (3)

- Fine-grain parallelism requires relatively-frequent communication compared to the computation interval.
- Consequently, fine-grain is more suited to shared memory than to distributed memory. Conversely, distributed memory requires relatively coarse grain to be effective.
- Because SIMD has less synchronization overhead, very-fine grain is more suited to SIMD than to MIMD.