Limitations to ideal parallelization

- Communication
- Load imbalance
- Serial portions of code
- Extra computations not in sequential code
- Memory bandwidth limitations
- OpenMP thread creation/scheduling
- Conflicts in shared caches
Execution Time

- \[ t = t_{\text{comp}} + t_{\text{comm}} \text{ (non-overlapped)} \]
- Plot against \( p \)
- Execution rate (flop rate)
  - What is reasonable? How does it compare to peak flop rate? How does it compare to peak memory rate for DRAM or cache?
- Be aware of timer skew and granularity
- Can also measure time breakdown (\( t_{\text{comp}} \) and \( t_{\text{comm}} \) and components of them)
- Other measurements
  - Hardware performance counters (PAPI)
  - MPI performance measurement and visualization (VAMPIR)
Speedup

- \( S = \frac{t_1}{t_p} \)

- Plot against \( p \)

- \( t_1 \) is time for “best” sequential solution
  - What is good speedup depends on what the code is doing (i.e., the problem being solved); an embarrassingly parallel code should have perfect speedup
  - Beware of speedup reported using bad sequential codes

- Relative speedup, e.g. \( S = \frac{t_4}{t_p} \)

- Superlinear speedup is possible
  - Due to caches (smaller local problem sizes)
  - Due to parallel problem having less work than sequential problem
We're giving you a bonus. You're doing a crappy job, but you're doing it incredibly fast.
Speedup often depends on problem size
Efficiency

- \[ E = \left( \frac{t_1}{p} \right) / t_p = S_p / p \]
- Plot against \( p \); efficiency generally decreases as \( p \) increases
- Can also define relative efficiency
- How do you maximize throughput?
  - Consider how to minimize the time to run \( n \) parallel jobs
  - Maximizing throughput is at odds with minimizing runtime of a particular job
Load Balance

- $b = \frac{\text{perfectly balanced time}}{\text{actual time}}$
- $b = \frac{\text{average } t_i}{\text{max } t_i}$
- $b$ is an upper bound on the efficiency
Amdahl's Law (1967)
Amdahl's Law

- $f = \text{sequential fraction of sequential program}$
- Speedup is bounded by $1/f$

\[
\begin{align*}
    t_1 &= f + g = 1 \\
    t_p &= f + g/p \\
    S &= \frac{t_1}{t_p} = \frac{1}{f + \frac{1-f}{p}}
\end{align*}
\]
Scaled Speedup

- Speedup when the problem size is increased proportionally with the number of processors
  - for $p$ processors, the problem size is proportional to $p$ (amount of work is proportional to $p$)
- Plot against $p$
- Also called *weak speedup* (regular scalability is called *strong speedup*)
- We can also define *scaled efficiency*
Gustafson's Law

- Analogue of Amdahl's Law for scaled problem sizes
- $F =$ sequential fraction of parallel program
- What is the speedup as a function of $p$?

$$
t_1 = F + (1 - F)p
$$

$$
t_p = F + (1 - F)p/p
$$

$$
S = \frac{t_1}{t_p} = \frac{F + (1 - F)p}{F + (1 - F)} = F + (1 - F)p
$$
Scalability

- Scalability is the ability of a program to continue speeding up when \( p \) is increased (defined for strong scalability and weak scalability)
  - e.g., a program has strong scalability up to 64 processors

- Scalability also applies to algorithms
  - Scalable algorithm = amount of work increases proportionally with problem size (or almost proportionally)
log-log plots

• \( t = 2n^3 \)
  Take log of both slides:
  \( \log t = 3 \log n + \log 2 \)
  which has the form \( y = mx + b \) which is a straight line with slope \( m \) and intercept \( b \)

• \( t = 2n^2 \)
  \( \log t = 2 \log n + \log 2 \)
log plots

• How would you plot $t = 2^n$?
• What does it look like on a log-log plot?
• On what kind of plot is this function a straight line?

• Log and log-log plots can be useful when plotting functions that change rapidly or functions over a very large domain
• Log and log-log plots can be used to estimate function parameters (when the functional form is known)