Overview: Performance Measures and Models

- parallel speedup and efficiency
- parallel overheads
- scalability (strong/weak)
- Amdahl’s Law (strong scaling law)
- Gustafson’s Law (weak scaling law)
- measuring time

Ref: Schmidt et. al. Section 2.5; Grama et. al. chapter 5; Wilkinson & Allen chapter 1

Parallel Speedup and Efficiency

- Speedup is a measure of the relative performance between a single and a multiprocessor parallel system when solving a fixed size problem

$$S_p = \frac{\text{execution time on single processor}}{\text{execution time using } p \text{ processors}} = \frac{t_{\text{seq}}}{t_{\text{par}}}$$

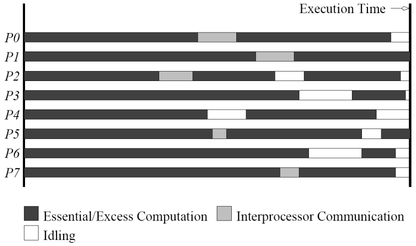
- (should we use walltime or CPU time?)
- t_{seq} typically defined as the time for the *fastest* known sequential algorithm
  - sometimes (but not always) we need a different algorithm for parallelization
- ideally, $S_p = p$ (aka linear speedup)
- can super-linear speed-up ($S_p > p$) happen in practice ? Yes
  - Examples: super-linear complexity; cache memory effects
- Efficiency is a measure of how far we are from ideal speedup. Defined as:

$$E_p = \frac{S_p}{p}$$

- clearly, $0 < E_p \leq 1$. Optimally, $E_p = 1$

Parallel Overheads

- can we expect $S_p = p$ for arbitrarily large p? **No!!!**
- why not? Parallelization-related overheads (examples):
  - interprocessor communication and synchronization
  - idling (caused typically by load imbalance, data dependencies, serial parts)
  - excess computation (e.g., higher #iters. with p, communication-avoiding algs)



in practice, one leverages performance analysis tools (e.g., Intel ITAC) to obtain Gantt charts like the one on the left; see also here for tools available on Gadi

- a problem that can be solved without communication is called embarrassingly parallel. Clearly, will have $E_p \approx 1$ for large p
- however, even under this scenario E_p will always drop for some (large) p due to resource underutilization caused by very little data winded up on each processor

---

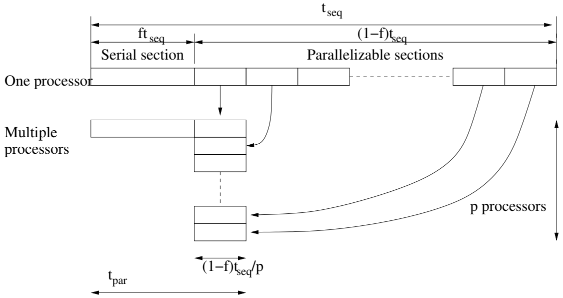
Scalability

scalability is a very broad term, used in many different contexts, which relates to the ability of a parallel system (algorithm + code + hardware) to exploit efficiently increasing computational resources

- hardware scalability: does increasing the size of the hardware give increased performance? e.g., aggregated memory bandwidth is typically limited as we scale p in shared-memory multiprocessors
- algorithmic scalability: at which rate does the complexity of an algorithm (number of operations and memory) grow with increasing problem size?
  - Example: for two dense $N \times N$ matrices, doubling the value of N increases the cost of matrix addition by a factor of 4, but the cost of matrix multiplication by a factor of 8 (i.e., $O(N^2)$ versus $O(N^3)$ complexity)
- strong parallel scalability: at which rate the efficiency of a parallel algorithm decays with increasing number of processors and fixed problem size?
- weak parallel scalability (previous two combined): at which rate the efficiency of parallel algorithm decays as we increase BOTH the number of processors and problem size?

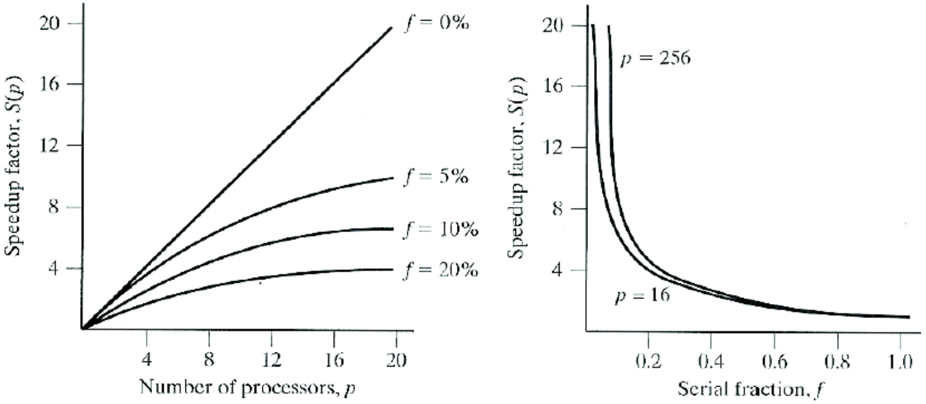
Amdahl's Law: definition

- considers "sequential parts" as the only source of overhead
- to what extent is S_p limited by this factor ?
- let f the (sequential) fraction of a computation that cannot be split into parallel tasks. Then, max speedup achievable for arbitrary large p is $\frac{1}{f}$!!!
  - $t_{\text{par}} = f t_{\text{seq}} + \frac{(1-f)t_{\text{seq}}}{p}$
  - $S_p = \frac{t_{\text{seq}}}{t_{\text{par}}} = \frac{p}{pf+(1-f)}$
  - $\lim_{p\to\infty} S_p = \frac{1}{f}$



- it is a strong scaling law, assumes fixed problem size

Amdahl's Law: Speedup Curves



Amdahl's law with fixed f and ↑ p (left), and fixed p and ↑ f (right)

Gustafson's Law

- Amdahl's law was thought to show that large p would never pay off
- However, it assumes fixed problem size executed on more and more processors
- In practice, this is not the case. One typically tailors problem size to p (weak scaling)
- A more realistic assumption is that parallel fraction can be arbitrarily extended
- Assume that the sequential portion of a parallel code is independent on p, and that the problem size can be scaled s.t. the parallelizable portion is p times larger. Then, the scaled speed-up:

$$S_p^{\text{scaled}} = \frac{T_{\text{seq}}^{\text{scaled}}}{T_{\text{par}}} = \frac{T_{\text{seq}} f + p T_{\text{seq}}(1-f)}{T_{\text{seq}} f + T_{\text{seq}}(1-f)} = \frac{f+p(1-f)}{f+(1-f)} = f + p(1-f) = p - f(p-1),$$

  is now an unbounded linear function with p (with slope depending on f)
- it is a weak scaling law, assumes problem size scaled in proportion with p

Measuring Time

- in order to evaluate performance of parallel algorithms we need to accurately measure computation times
- broadly speaking, there are two kind of times: wall clock time (i.e., elapsed time) and CPU time
- we will use wall clock times all the way through in this course (as, among others, we also want to measure e.g., overhead of system calls required to implement communication)
- two important timer parameters are timer resolution (t_R) and overhead (t_O)
- t_R is the smallest unit of time that can be accurately measured by the timer
  - the lower the t_R the higher the resolution
  - if the event to be time is shorter than timer resolution, we can't measure it!
- t_O relates to the instructions which are executed and included in the measured time and not strictly related to the event being measured
- t_R and t_O can be estimated measuring (differences between) repeated calls to a timer function (Lab #1)