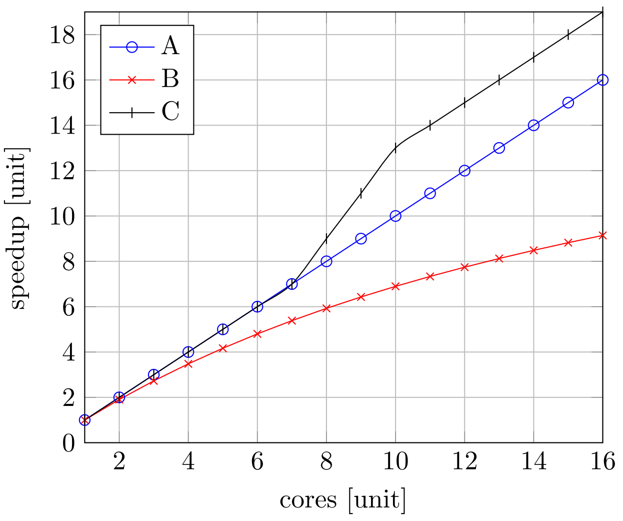
1 Exercise 1

The speedup of a parallel program is defined as

$$S_p = \frac{t_s}{t_p} \tag{1}$$

where t_s is the execution time of the fastest sequential implementation and t_p is the execution time of the parallel program using p processors. This figure shows a speedup graph for three fictional programs A, B, and C:

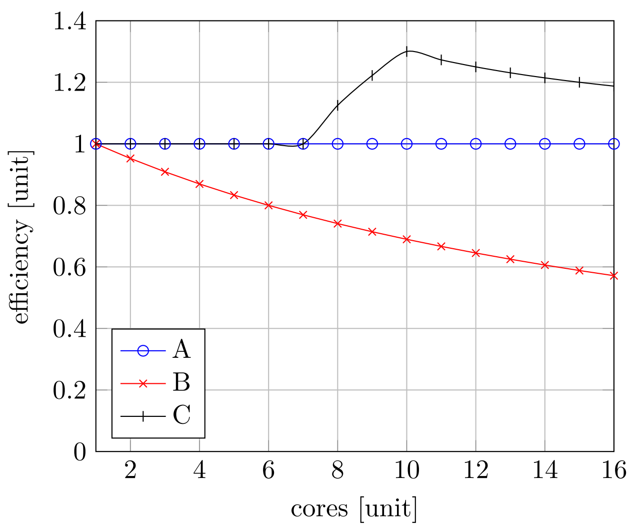


The figure shows several things: First, A is a program that scales linearly — for 16 cores, you get a speedup of 16. Next, there is B, which scales sub-linearly. This is most often the case, as parallelization does not come for free and incurs some overhead. Third, there is C, which shows super-linear scaling. While somewhat rare, this can happen for example if working with non-deterministic algorithms or due to the memory hierarchy (e.g. same amount of data divided among more cores $\rightarrow$ less data per core $\rightarrow$ might fit in the cache at some point).

Another very common metric for parallel programs is the efficiency, defined as

$$E_p = \frac{S_p}{p} \tag{2}$$

The next figure shows the efficiency for the same data as before:



Amdahl's law is defined as

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}} \tag{3}$$

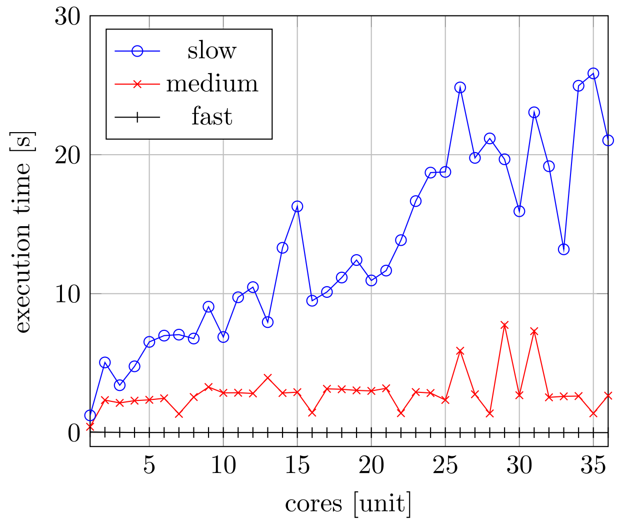
where α is the portion of a program that cannot be parallelized, $(1-\alpha)$ denotes the parallelizable part, and p is the number of cores. For example, the previous figures show speedups and efficiency for $\alpha = 1$ (A) and $\alpha = 0.95$ (B). Amdahl's law is immensely important because it puts the amount of unparallelizable code in your program in relation to the maximum speedup one can expect. Computation for the first example in Exercise 1:

$$\alpha = 0.1 \qquad S_6 = \frac{1}{\frac{1}{10} + \frac{\frac{9}{10}}{6}} = \frac{1}{\frac{1}{10} + \frac{9}{60}} = \frac{1}{\frac{5}{20}} = 4 \tag{4}$$

Note: The last question of Exercise 1 gives the complexity of the algorithm. This information is not required for computing Amdahl's law with the given numbers of $S_{64} = 10$ and can therefore be ignored.

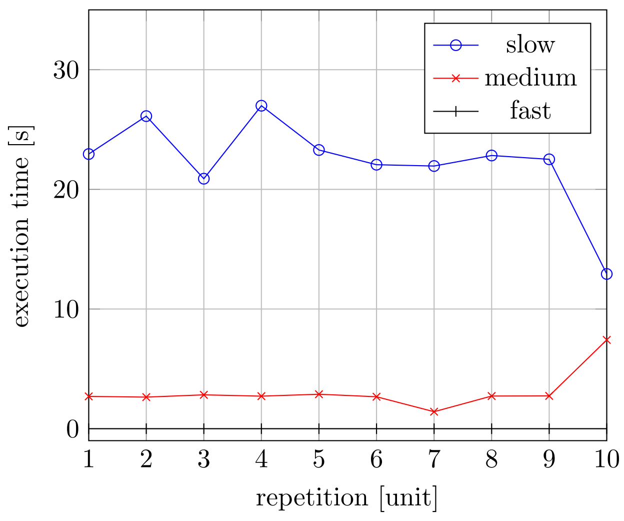
2 Exercise 2

The following figure illustrates the execution times for the three programs to be measured (a table of the raw data follows below) on a system with two Intel Xeon E5-2699 v3 processors and a total of 36 cores, with gcc 6.3.0 and compile flags `-O3 -fopenmp`:



The data shows that first, the `slow` program is not only slower than `medium` and `fast`, it also does not decrease its execution time for increasing numbers of cores (also referred to as *'it does not scale'*) but actually tends to take more execution time with increasing numbers of cores. Contrary to that, the `medium` program seems to have roughly the same execution time regardless of the number of cores used. Finally, program `fast` scales with the number of cores (visible from the first few rows of the raw data), and is also overall much faster than the `medium` or `slow` programs.

Finally, the next figure shows the execution times for the maximum number of cores (36) of the three programs, each measured 10 times:



The figure illustrates that the execution time varies a lot and that the data should probably be represented using statistics (e.g. median, mean, variance, etc.) and represented using graphs such as boxplots. However, this exceeds the scope of this first assignment.

Table 1 towards the end of this document shows the execution times for all three programs.

Table 1: Execution times of programs `slow`, `medium`, and `fast`

cores	slow	medium	fast
1	1.24	0.42	0.05
2	5.05	2.34	0.03
3	3.41	2.14	0.02
4	4.77	2.30	0.01
5	6.53	2.36	0.01
6	6.98	2.47	0.01
7	7.05	1.35	0.01
8	6.77	2.56	0.01
9	9.06	3.28	0.01
10	6.88	2.86	0.01
11	9.74	2.87	0.01
12	10.47	2.82	0.01
13	7.95	3.94	0.01
14	13.30	2.85	0.01
15	16.28	2.91	0.00
16	9.49	1.44	0.00
17	10.12	3.15	0.00
18	11.16	3.12	0.00
19	12.42	3.04	0.00
20	10.95	3.00	0.00
21	11.66	3.19	0.00
22	13.85	1.40	0.00
23	16.66	2.92	0.00
24	18.71	2.85	0.00
25	18.76	2.35	0.00
26	24.84	5.89	0.00
27	19.77	2.76	0.00
28	21.17	1.38	0.00
29	19.67	7.75	0.00
30	15.93	2.69	0.00
31	23.05	7.30	0.00
32	19.17	2.54	0.00
33	13.19	2.61	0.00
34	24.96	2.63	0.00
35	25.85	1.39	0.00
36	21.03	2.66	0.00

3 What's the point of all this?

So why do we show you a four-page PDF for such a simple assignment, you might ask? Well, the point is not only to discuss the basic concepts of parallelism, but to give you an example of how to properly present data. Take a good look at the graphs, note that they all have axis labels like 'speedup' or 'execution time' along with the unit of measurement in brackets (e.g. '[s]' for seconds or '[unit]' for unitless quantities). **Never** show graphs without proper axis labels. Also, note how all y-axes start at 0 in order to not give a biased illustration of the data (as the marketing departments of many companies like to do...). Furthermore, note the different colors for each line in a figure and the marks for each line (circles, crosses, etc.), which clearly show every data point — obviously, there is no data for e.g. 4.5 cores. However, if all of these marks were gone, you would not be able to tell for which numbers of cores the measurements were actually taken, and for which numbers of cores the data was just interpolated.

These graphs are obviously not perfect for illustrating the collected data, but they should serve as a starting point.