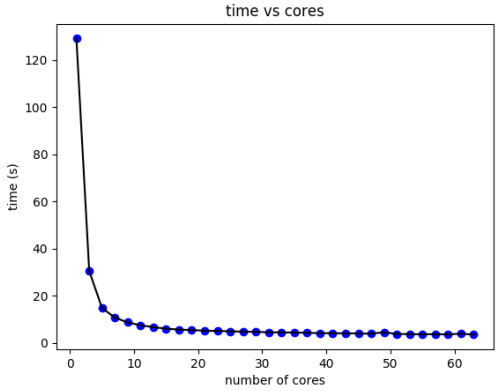
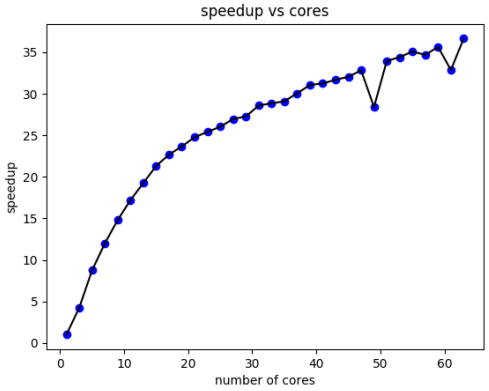
HPC I Homework 3

Morse, Michael December 11, 2017



(a) Time vs number of cores



(b) Speed-up vs number of cores

Figure 1: Plots for Round Robin Mandelbrot code

1 Problem 1

1.1 Strong Scaling

For my round robin Mandelbrot code I used two processors as the serial time T_1 as one processor was the master so only one was doing the work. The code was has nearly linear speedup up to around 20 cores then again after around 25 cores. I believe that this is around the time that communication between nodes comes into play. Unfortuntely, attempting to use the beginning point to estimate the serial fraction of the code by

$$S = (1 - F_s)p + F_s \tag{1}$$

was unsuccessful as F_s was given as $F_s \approx -0.7$ from both the slope and the y-intercept. This leads me to believe that the communication time is non-negligible. However, we can use Eqn.(9) from chapter 2 of Eijhout's book, and some algebra to write

$$\frac{1}{S} = F_p(\frac{1}{p}) + (1 - F_p) \tag{2}$$

where we have assumed that the communication time is a constant. Using scipy statistics we get that the code has a $F_p = .95 \pm 0.2$ with a $r^2 = .97$. We should expect that is code is almost perfectly parallelizable as the only parts done in serial are the check on the point and the initialization, both of which are not mathematically intensive. This can be seen in Fig.(5).

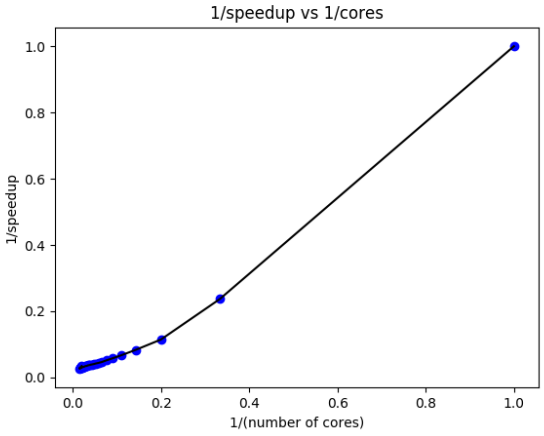


Figure 2: $^{1}/_{\text{Speed}-\text{up}}$ vs $^{1}/_{\text{numberofcores}}$

1.2 Karp-Flatt metric

We can use also calculate the serial fraction using the Karp-Flatt metric defined as

$$f_s = \frac{\frac{1}{s} - \frac{1}{p}}{1 - \frac{1}{p}} \tag{3}$$

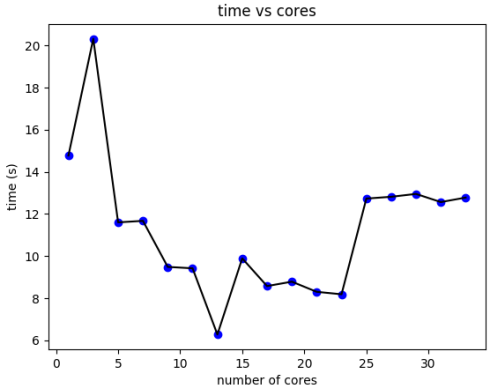
Taking the average of the f_s calculated for 2-36 cores we get that for the Mandelbrot set code $f_s = 0.02 \pm 0.17$ which is consistent with the f_s calculated in the first subsection of 0.05

1.3 Weak Scaling

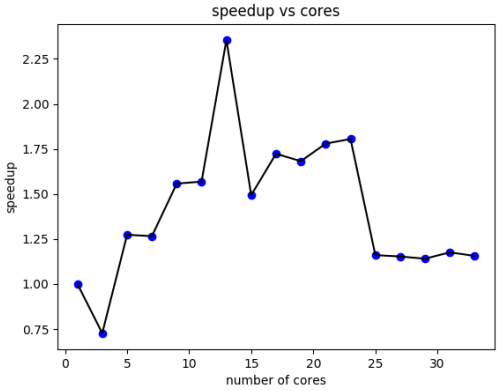
For the weak scaling we see that the time stays more or less constant going from 4 cores up to more then 30 cores. This can be understood as each core has the same amount of work to do no matter how many cores there are. This is similar to how I has written the part (b) Mandelbrot code in homework two. This allows for a more dense grid size to be completed in the time it takes one core to do a proportionally less dense grid.

The same results are seen for speed up. While there is some speedup it is not a trend that more cores causes more speedup. Again this is because the problem is scaling with the number of cores so speedup is not actually a good metric to use to gauge the effectiveness of the parallel code.

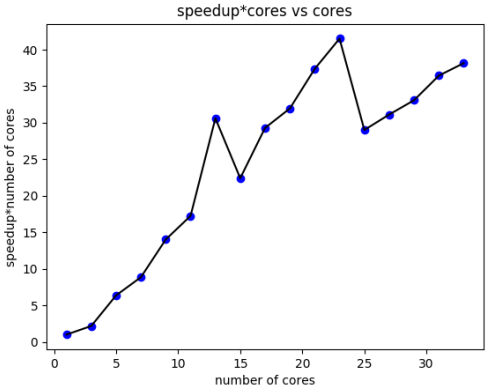
A better metric is Fig.(3c) where we multiple the speedup by the number of cores. This quantity now scales linearly as expected. 40 cores are doing 40 times the work of one core so the effective speedup is 40. The deviation from the graph being proportional is most likely due to the increase in communication time with an increase in cores.



(a) Time vs number of cores



(b) Speed-up vs number of cores



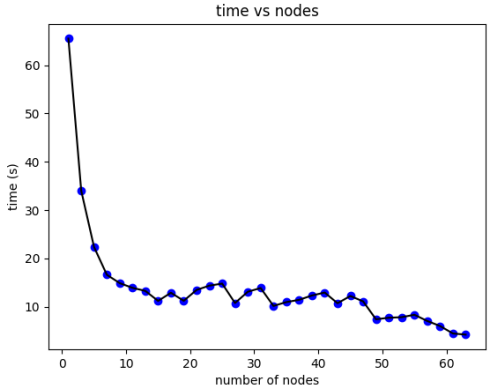
(c) Speed-up vs number of cores

Figure 3: Plots for Mandelbrot code, weak scaling

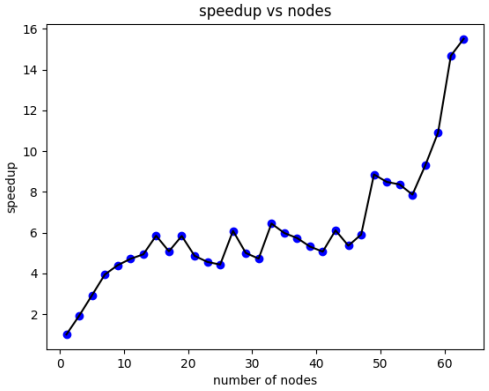
2 Problem 2

2.1 Strong Scaling

We can run the same type of analysis we did in question 1 on the pi code used as an example in *Using MPI* by Gropp. We notice first in Fig.(4) that the speed-up and time decrease lines are not as sharp as they are in question 1. I would postulate that this is because there is more work being done serially. If we calculate what the fraction of the code is parallel using Eqn.(**??**) and linear regression from scipy we find a result of $1.1 \pm .16$ with $r = .78$, which is a much greater range and less precision than in question 1.



(a) Time vs number of cores



(b) Speed-up vs number of cores

Figure 4: Plots for pi code

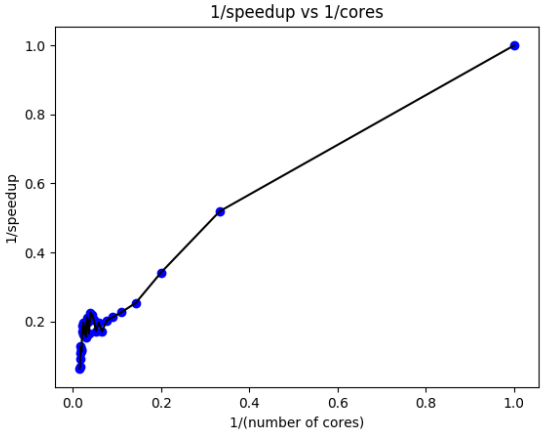


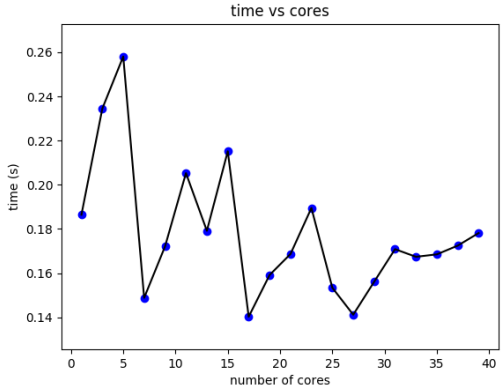
Figure 5: $^{1}/_{\text{Speed}-\text{up}}$ vs $^{1}/_{\text{numberofcores}}$

2.2 Karp-Flatt metric

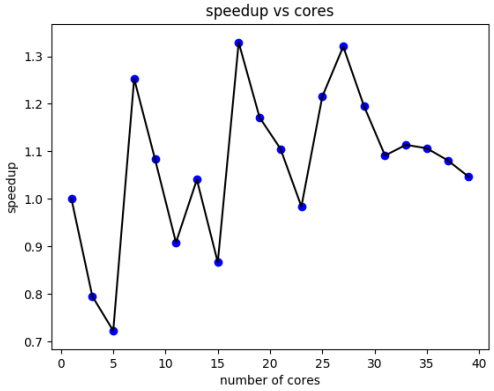
We can use also calculate the serial fraction using the Karp-Flatt metric defined as

$$f_s = \frac{\frac{1}{s} - \frac{1}{p}}{1 - \frac{1}{p}} \tag{4}$$

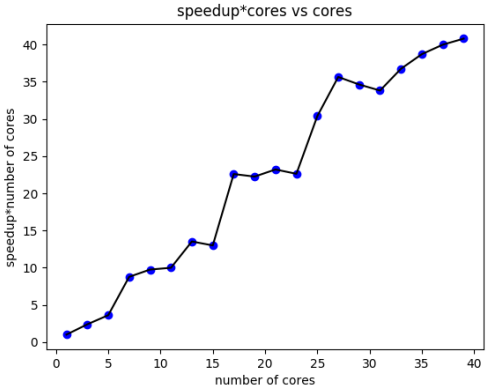
Taking the average of the f_s calculated for 2-36 cores we get that for the Mandelbrot set code $f_s = 0.17 \pm 0.15$ the results from this and the regression in the previous section agree within one standard deviation.



(a) Time vs number of cores



(b) Speed-up vs number of cores



(c) Speed-up vs number of cores

Figure 6: Plots for Mandelbrot code, weak scaling

2.3 Weak Scaling

Similar to what we saw for weak scaling in question 1 the time and speedup graphs in Fig.(6) do not show any clear trends. As discussed above this is not surprising as the time should stay relatively constant, up to an increase in communication time with more cores, due to the fact that each core has a constant amount of work. Fig.(6c) again shows that the better metric to use is speedup*cores as it will scale with the number of cores. The deviation from the graph being proportional is most likely due to the increase in communication time with an increase in cores.