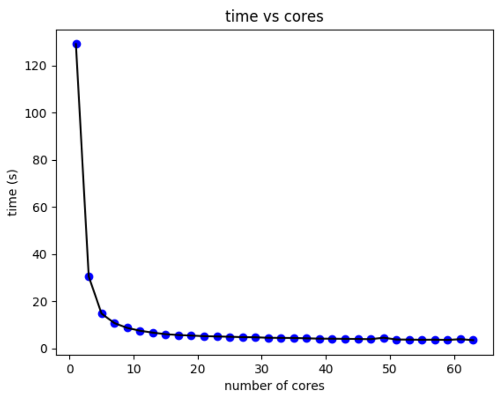
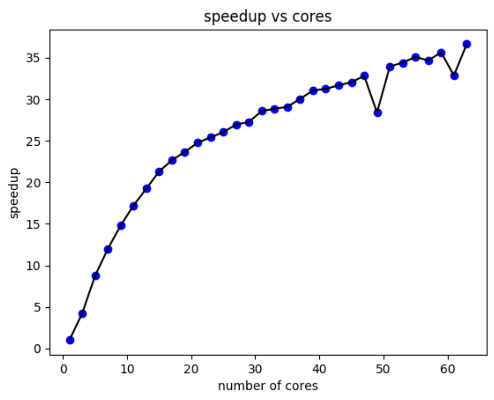
HPC I Homework 3

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 $F_p = .95$ 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.(4).

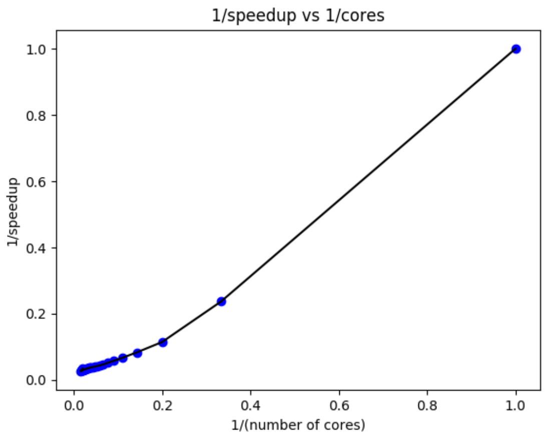


Figure 2: $^1/_{\text{Speed − up}}$ vs $^1/_{\text{numberofcores}}$

1.2 Karp-Flatt metric

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

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

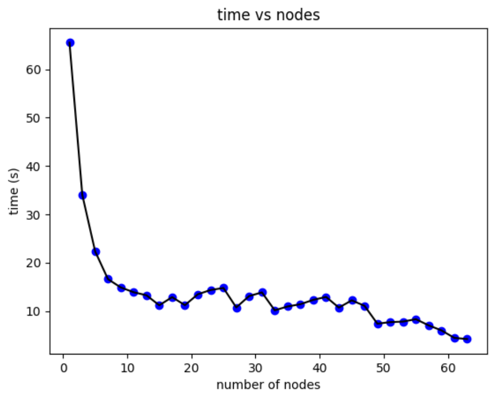
Taking the average of the f_s calculated for 2-36 cores we get that for the Mandelbrot set code $f_s = 0.02 \pm 0.17$ which is consistent with the f_s calculated in the first subsection of 0.05

1.3 Weak Scaling

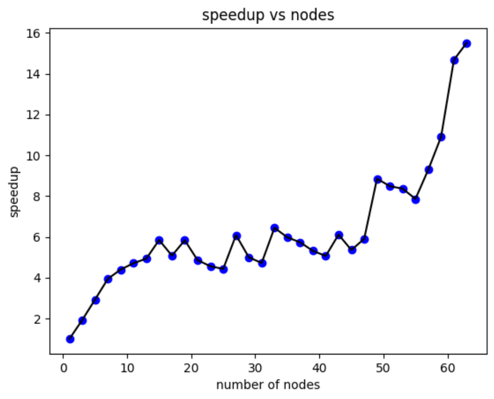
2 Problem 2

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.(3) 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 and as there is



(a) Time vs number of cores



(b) Speed-up vs number of cores

Figure 3: Plots for pi code

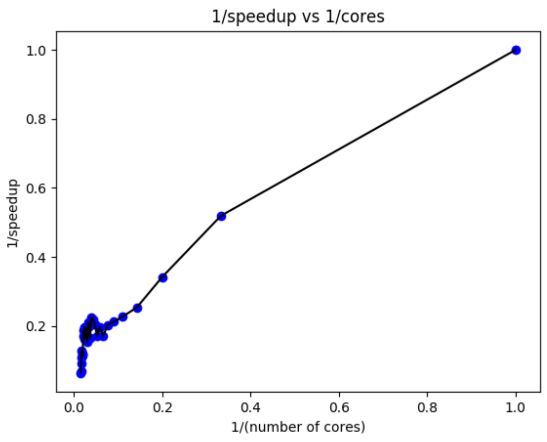


Figure 4: $^1/_{\text{Speed}-\text{up}}$ vs $^1/_{\text{numberofcores}}$