# Scalability and Convergence report

Benjamin Russell - fdmw97

January 25, 2021

## Introduction

The scalability tests were carried out on Hamilton using the compile command:
icpc -std=c++0x -O3 -xHost -fopenmp step-4.cpp. The compute node used had 28 cores each with a clock speed of 2.40GHz. The convergence Tests were also performed on Hamilton using only one core and the compile commands:
icpc -std=c++0x -O3 -xHost -no-vec step-1.cpp *and* icpc -std=c++0x -O3 -xHost -fopenmp step-3.cpp repectively.

## Scalability Plot

For the speed-up tests I used a setup consisting of 729 particles in a random grid shape with masses normalised between 0.01 and 0.1 in order to minimise collisions (which could affect runtimes), All IO was disabled as well to prevent unnecessary slowdowns. The solution to step 4 clearly exhibits strong scaling behaviour as its speedup per core decreases the more cores are added due to serial parts of the code Fig. 1. Using non-linear least squares regression I fit a curve of Amdahl's Law. The fitted curve has an $f$ value of 0.158727 suggesting that my solution spends approximately 15% of its time in serial computation. This makes sense as I could not find a satisfactory way to parallelise collision which would account for this 15%.

## Convergence Plot

For the convergence study I used a setup consisting of 2 particles aligned on 2 axes spaced distance 1 on either side of the co-ordinate system's origin, both particles also had 0 velocity and masses of 1.0. I then varied the time-step size starting at 0.001 and halving for each subsequent test. The position at which the 2 particles merged was recorded. In order to avoid relying on an analytical solution I used the differences between subsequent measurements in order to determine the convergence order as this will also decrease in that order. Both scales were converted to log scales in order to better identify the trend and to calculate the gradient ($p$). Using least squares regression I was able to plot a trend line for both time-stepping methods and use this to determine $p$ as 2.157 for Runge-Kutta 2nd Order and 1.041 for the Explicit Euler Fig. 2. This suggests that my RK(2) implementation is 2nd order convergent and Explicit Euler is 1st order convergent. The first few data points do not exhibit much convergence which likely means that at time-step sizes $>=$ 0.01 the time-stepping schemes are not A-stable due to truncation errors.

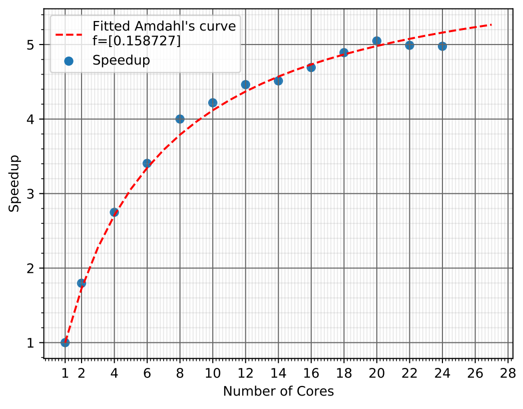

**Figure 1:** Graph of Speed-up against core count

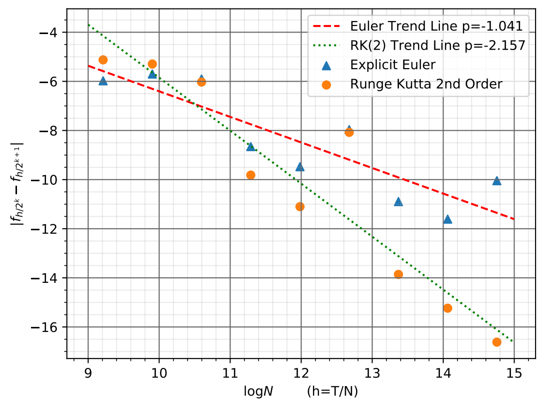

**Figure 2:** Graph of time-step against differences