1 Introduction

2 Serial Optimisation

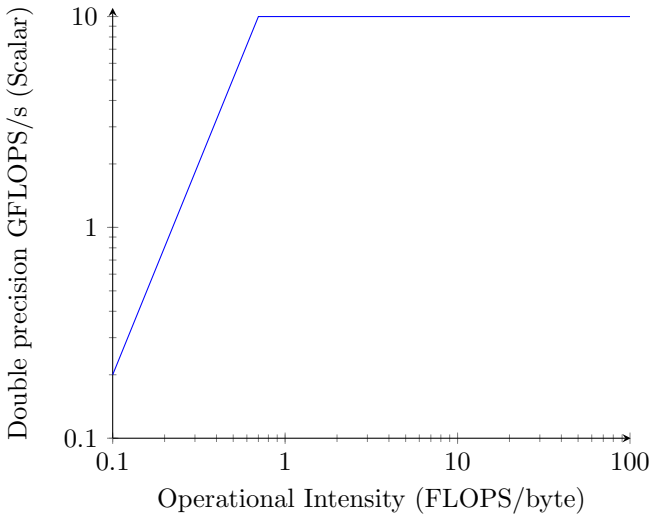
2.1 Reduce Memory Accesses

In the original implementation of the algorithm for each timestep the `cells` array was looped over the 4 times, in `propagate`, `rebound`, `collision` and `av_velocity`. This resulted in repeated stores and loads of the same sections of memory. To prevent this the first 3 separate functions (those in the `timestep` function) were fuzed into a single loop. This meant the same sections of memory were used closer together making it is more likely for them to still be in cache. The function `av_velocity` was then repeating calculations that already took place in collision and requiring an additional loop over cells. The result was therefore calculated in the single parse over the cells and returned from the timestep function. Similarly in propagate and collisions the values were switched between `tmp_cells` and `cells` multiple times. In the new implementation the `tmp_cells` array was used as the "answer" space and stored only the next timesteps cell values. This then only required a single write to `tmp_cells` each timestep. At the end of the timestep the `tmp_cells` and `cells` array's pointers were then swapped which set the `cells` array to the correct value without having to write directly to the array.

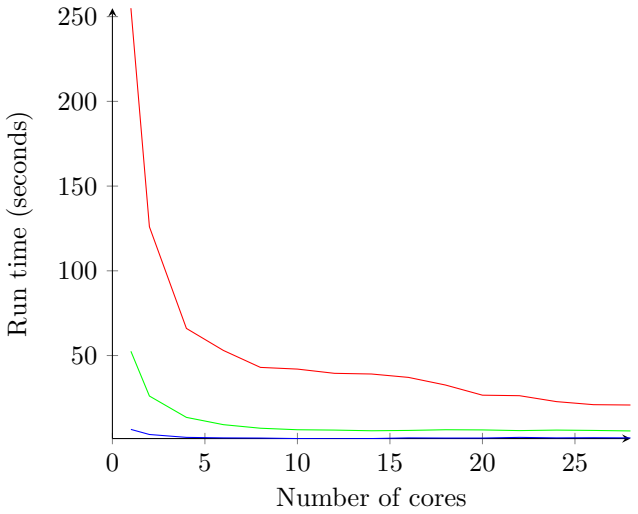
2.2 Vecotrisation

The first step to vectorizing the code is to ensure all the arrays used in timestep are aligned. This can be achieved by using `__mm_malloc` and then adding compiler directives which covey this alignment.

In the original implementation the speeds for each cell were stored in a structure (`t_speed`) and the whole grid of cells was an array of these structures. This implementation does not lend it self to vertorization as



3 Parallel



Results			
Number of cores	128x128	256x256	1024x1024
1	6.4	52.5	255
2	3.4	26.1	126
4	1.7	13.5	66
6	1.4	9.2	53
8	1.3	7.1	43
10	1.0	6.2	42
12	0.9	6.0	39.5
14	1.0	5.6	39.1
16	1.4	5.8	37.1
18	1.3	6.2	32.6
20	1.3	6.1	26.6
22	1.7	5.7	26.3
24	1.4	6.0	22.8
26	1.5	5.8	21.0
28	1.4	5.5	20.8

tableParallel scaling results from 1-28 cores