**Serial optimisations and OpenMP Report**

b.A description of your Serial optimisations and OpenMP design;

c.Comparisons of your parallel performance vs. serial performance;

d.Analysis of effectiveness of different optimisations you tried;

e.Make it clear what your best performance is for the “256x256” case;

Serial Optimisations and OpenMP design

Parallel vs serial performance

Effectiveness of optimisations

Best performance (‘256x256’)

List of optimisations attempted:

Without –O3 flag:

Reynolds number: 3.718483826704E+01

Elapsed time: 493.078442 (s)

Elapsed user CPU time: 491.907218 (s)

Elapsed system CPU time: 0.004999 (s)

With –O3 flag

Reynolds number: 3.718483826704E+01

Elapsed time: 213.117505 (s)

Elapsed user CPU time: 212.609678 (s)

Elapsed system CPU time: 0.005999 (s)

1. Hoisting code (Repeated hence wasted calculations)
   1. Collisions function – c\_sq, w0, w1, w2. However saw no time saving, compiler most likely already hoists variables at compile time.
2. Cache Thrashing
   1. If data is constantly being loaded into cache and out again in the same code block. The data being accessed is too large to be stored within local cache.
3. Vectorisation
4. Combining for loops:
   1. In initialise(), there exists two sets of double for loops that iterate over the same value. Therefore combine computation in a single for loop.

Code Structure:

Main()

Initialise()

* Get values from input files
* Initialise values in array
* For nx, for ny

For(params.maxIters)

Timestep()

* Accelerate\_flow()
  + For nx
* Propagate()
  + For ny, for nx
* Rebound()
  + For ny, for nx
* Collision()
  + For ny, for nx

Av\_velocity()

* For ny, for nx