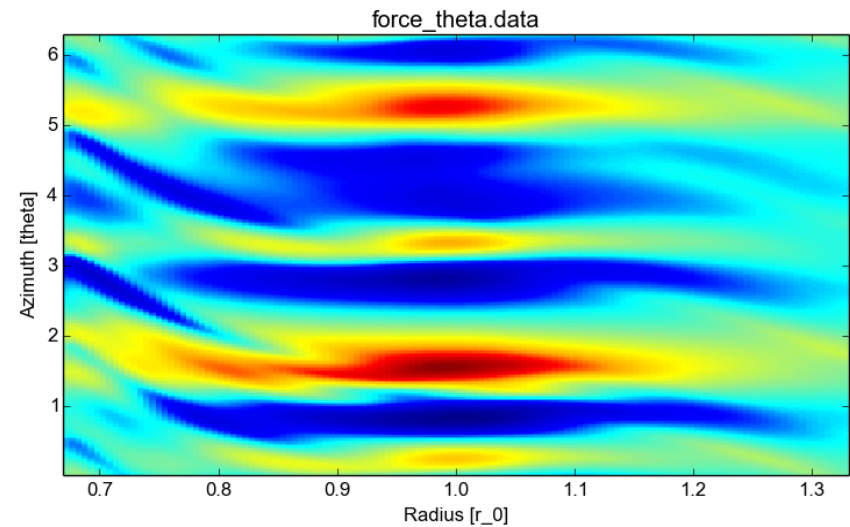
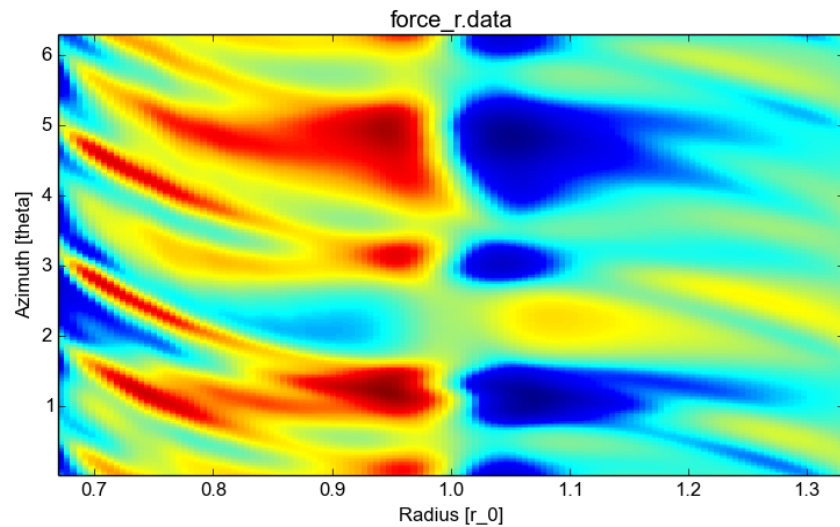
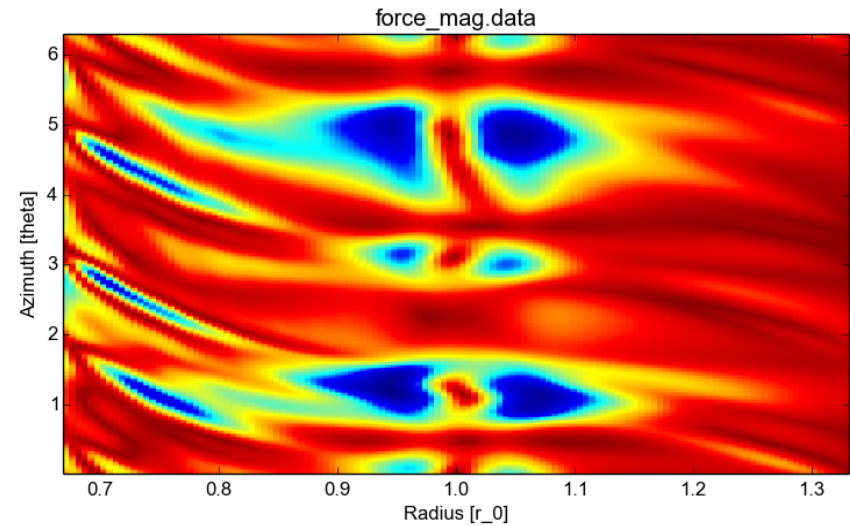
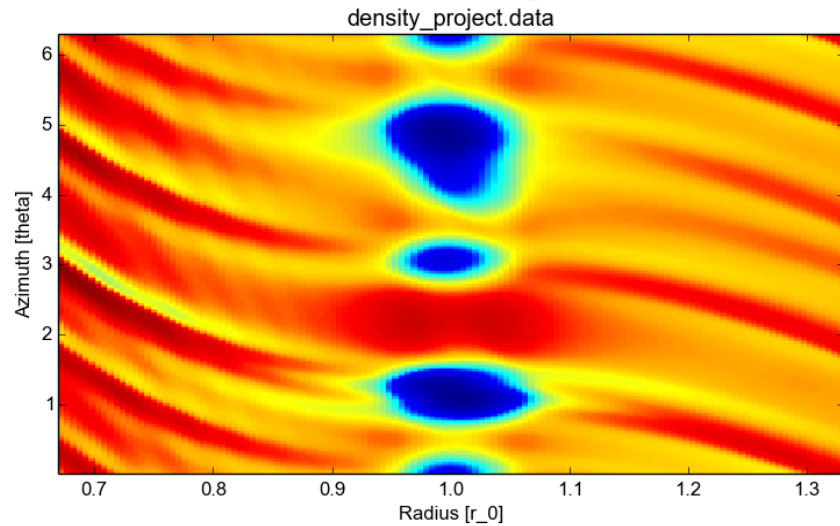


Computational Science II



Preliminary Optimisation Report

By: Sara Vossoughi

$$\sum \sum \frac{G(r', \theta') r' \cdot \Delta r' \cdot \Delta \theta'}{(r^2 + r'^2 - 2 \cdot r \cdot r' \cos(\theta - \theta'))^{(2/3)}} \begin{cases} u_r = r - r' \cdot \cos(\theta - \theta') \\ u_\theta = r' \cdot \sin(\theta - \theta') \end{cases}$$

Runtime Performance*

$O(n^2)$

worst performance: $\sim 69.0s^{**}$



$O(n^2)$

best performance: $\sim 11.8s$

* 3 year old intel laptop with four 2.2GHz cores (i7 2620M)

** With inner loop time measurements, the process did not complete

Optimizations

First Culprit: Trigonometry

- Over 1 billion calls to sine and cosine
- Only 256 distinct angles $(i \cdot \Delta\theta)$
- Symmetry of sine and cosine

Moving operations from inner to outer loop

- Certain calculations are constant within the inner loop

Choosing Cheaper Operations

- IAND vs. Modulo
- Multiplications vs. power operator $(x^{**}y)$
- Sqrt(x) over power operator

Optimizations Continued

- CosCache is an array of 64 ($N_{\text{theta}}/4$) pre-computed cosine values
- Fewer memory requirements, more logic
- Can possibly be further optimised with hardcoded constants ($N_{\text{theta}}/2$) with larger array
- SELECT CASE did not provide an improvement

```
sara@Altair: ~/2015/computationalScience2/Compu
contains

real(8) function cosFast(x)
  integer :: x, y
  y = MODULO(x, N_theta)
  IF (y < N_theta/4) THEN
    cosFast = cosCache(y)
  ELSE IF (y < N_theta/2) THEN
    cosFast = -cosCache((N_theta/2-y) -1)
  ELSE IF (y < 3*N_theta/4) THEN
    cosFast = -cosCache(y - N_theta/2 )
  ELSE
    cosFast = cosCache(N_theta-y -1)
  END IF
end function cosFast

real(8) function sinFast(x)
  integer :: x
  sinFast = cosFast(x - N_theta/4)
end function sinFast

107,0-1 33%
```

Optimization

```
sara@Altair: ~/2015/computationalScience2/Computational_Science_II/Sara
d_rd_theta = r_step * theta_step ! pre-compute, causes slowdown
do out_i=0, (N_r*N_theta)-1 !32768 loops
  r=r_prime(out_i/N_theta)-(r_step/2.0)
  r2=r*r ! pre-compute, causes slowdown
  theta = theta_prime(MODULO(out_i, N_theta))-(theta_step/2.0)
  force_r(out_i)=0
  force_theta(out_i)=0
  do in_i=0, (N_r*N_theta)-1 !another 32768 loops
    r_p = r_prime(in_i/N_theta) !current r-prime
    theta_p = theta_prime(IAND(in_i, N_theta-1))!current theta-prime
                                                    44,1          16%
```

- Moving variables outside the inner loop
- Pre-computing values
- Using cheaper operations IAND

Results*

- Trigonometric caching (256 distinct values): 68s → 20s
- Compiler optimization settings (-O3 & -O): 20s → 11.8s
- Trigonometric symmetry: 20s → 25s
- Moving instructions to outer loops: +1-2s ??
- Using cheaper operations: inconclusive measurements/no change

* FORTRAN noob, your mileage may vary

Conclusions

Best performances achieved

- Compiler tweaking (-O3 or -O flags)
- Caching trigonometric functions

No change or slowdown

- Trigonometric symmetry costs a slowdown
- Moving computations to outer loop had little effect
- Bitwise AND vs. Modulo computation in inner loop had no effect

This is largely a trial and error endeavor.