grav_project Level 0
My Version of the Program

Denzel, Philipp

Email:
phdenzel@hispeed.ch

Structure of First Program

- Open files
- Read data into arrays
- Loop through all grid points (force)
- Nested loop through all grid centres (density)
- Calculate formula inside inner loop
- Write force into new file

Read Files

```
open(unit=8, file='/path/to/r_project.data', status='old',
    action='read')
! read radii into 1-D array
 do i = 1, N_r
     read(8, '(e20.10)') r_i
     r(i) = r_i
 end do
 close(unit=8)
```

for r(N_r), theta(N_theta), sigma(N_r,N_theta), dr(N_r), dtheta(N_theta)

Simple Implementation

```
! write force components for every corner in grid
do i = 1, N_r
   r_i = r(i)-.5*dr(i) ! shift r to the corners
   do j = 1, N_theta
      theta_j = theta(j)-.5*dtheta(j) ! shift theta to the corners
      ! sum up the forces onto the point (i, j)
      do iprime = 1, N_r
         do jprime = 1, N_theta
            force_point = -sigma(iprime,jprime)*r(iprime)*dr(iprime)*dtheta(jprime)/(r_i**2+r(iprime)
                 **2-2.*r_i*r(iprime)*cos(theta_j-theta(jprime)))**(1.5)
            f_r = f_r + force_point * (r_i-r(iprime)*cos(theta_j-theta(jprime)))
            f_theta = f_theta + force_point * r(iprime)*sin(theta_j-theta(jprime))
         end do
      end do
   write(11, '(e20.10)') f_r
   write(12, '(e20.10)') f_theta
   f_r = 0.
   f_theta = 0.
   end do
end do
```

Optimization Strategy

- Shift operations to outer loops, if possible
- Better programming syntax
- Idea: find faster methods, e.g. invsqrt (see later)

Write cos/sin Lookup Tables

Dimensions of the lookup table adjusted to the angular difference:

```
REAL(8), DIMENSION(1-N_theta:N_theta-1) :: cos_table,
    sin_table
! fill the sin/cos table
 do j = 1,N_theta
     theta_j = theta(j)-.5*dtheta(j)
     do jprime = 1,N_theta
         diff_theta = theta_j-theta(jprime)
         cos_table(j-jprime) = cos(diff_theta)
         sin_table(j-jprime) = sin(diff_theta)
     end do
 end do
```

Write Mass Lookup Tables

```
! fill the mass table
 do iprime = 1, N_r
     r_iprime = r(iprime)*dr(iprime) ! multiply dr already
         here to save some operations
     do jprime = 1,N_theta
         mass(iprime,jprime) = -sigma(iprime,jprime)*r_iprime*
             dtheta(jprime)
     end do
 end do
```

will change for every timestep

Current Version

Diffrence to the first version in capital letters:

```
! write force components for every corner in grid
 do i = 1, N_r
    r_i = r(i)-.5*dr(i) ! shift r to the corners
    R_I_SQUARED = R_I*R_I ! better than calculating it in formula
    do j = 1, N_theta
        theta_j = theta(j)-.5*dtheta(j) ! shift theta to the corners
        ! sum up the forces on the point (i, j)
        do iprime = 1, N_r
            R_IPRIME = R(IPRIME) ! slight performance improvement
            R_IPRIME_SQUARED = R_IPRIME*R_IPRIME ! slight performance improvement
            do jprime = 1, N_theta
                ! formula for the gravitational force split into four parts for faster calculation
                R_IPRIME_COS = R_IPRIME*COS_TABLE(J-JPRIME)
                DENOM_POINT = 1./SQRT(R_I_SQUARED+R_IPRIME_SQUARED-2.*R_I*R_IPRIME_COS)
                DENOM_POINT = DENOM_POINT * DENOM_POINT * DENOM_POINT
                force_point = MASS(IPRIME,JPRIME) * DENOM_POINT
                f_r = f_r + force_point * (R_I-R_IPRIME_COS)
                f_theta = f_theta + force_point * R_IPRIME*SIN_TABLE(J-JPRIME)
            end do
        end do
    write(11, '(e20.10)') f_r
    write(12, '(e20.10)') f_theta
    f_r = 0.
    f_theta = 0.
    end do
 end do
```

InvSqrt

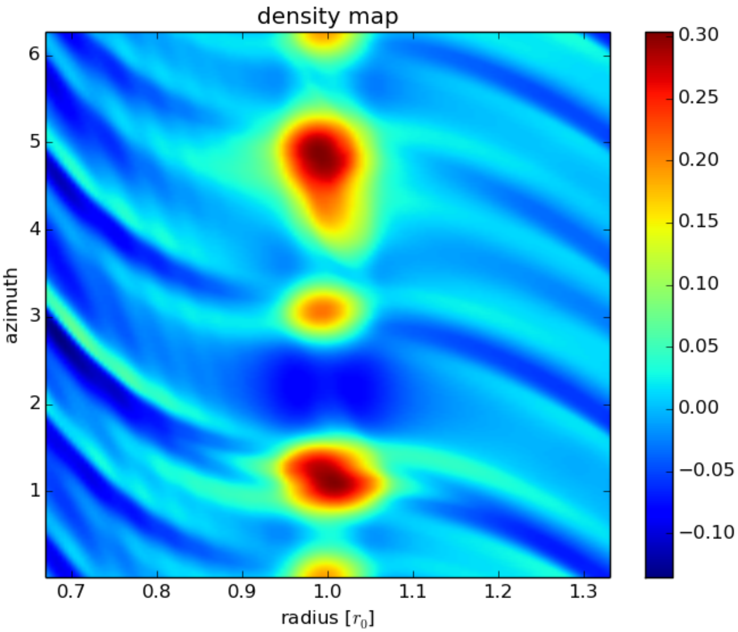
http://en.wikipedia.org/wiki/Fast_inverse_square_root

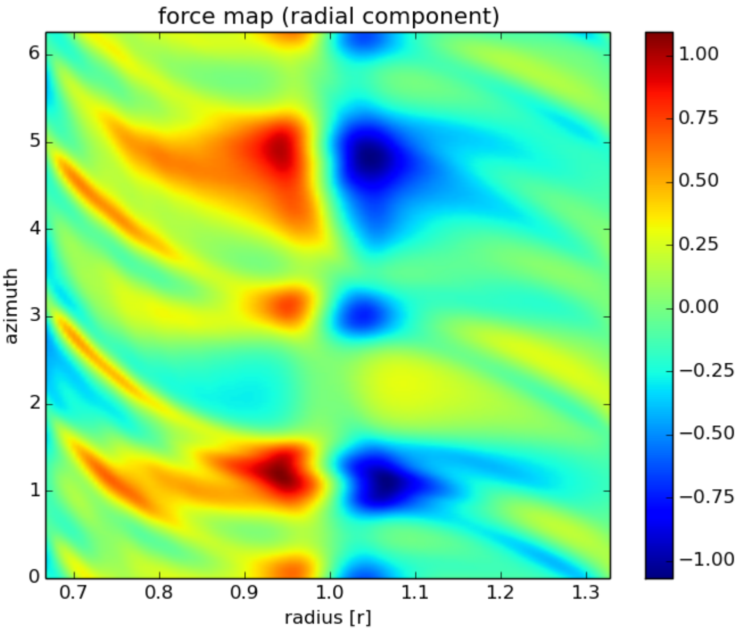
Custom InvSqrt

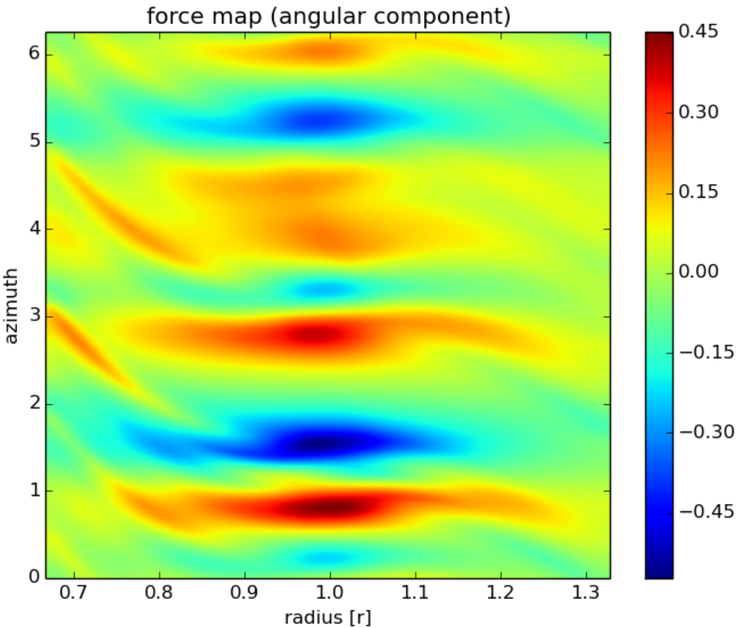
```
  REAL(8) FUNCTION InvSqrt (x)
   IMPLICIT NONE
   TYPE casting
      REAL(8) :: x
   END TYPE casting
   REAL(8), INTENT(in) :: x
   ! casting
   TYPE(casting), TARGET :: pointerTo
   ! Encode data as an array of integers
   INTEGER(8), DIMENSION(:), ALLOCATABLE :: enc
   INTEGER(8) :: length
   INTEGER(8) :: magic_number = 6910469410427058089
   REAL(8) :: xhalf
   xhalf = .5*x
   ! transfer to heap
   pointerTo%x = x
   ! encode a memory section from a type to other
   length = size(transfer(pointerTo, enc))
   allocate(enc(length))
   ! encoded to integer
   enc = transfer(pointerTo, enc) ! evil floating point bit level hacking
   enc(1) = magic_number - rshift(enc(1),1) ! wtf! (for int64: 0x5fe6eb50c7b537a9 = 6910469410427058089)
   ! decode
   pointerTo = transfer(enc, pointerTo)
   ! dealloc
   deallocate(enc)

   InvSqrt = pointerTo%x*(1.5 - xhalf*pointerTo%x*pointerTo%x)
END FUNCTION InvSqrt
```

Alternative

Intel Programming Environment: SSE instructions rsqrtss/rsqrtps


density map
azimuth
radius [r_0]
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1
2
3
4
5
6
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10


force map (radial component)
azimuth
radius [r]
0
1
2
3
4
5
6
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


force map (angular component)
azimuth
radius [r]
0
1
2
3
4
5
6
0.7
0.8
0.9
1.0
1.1
1.2
1.3
0.45
0.30
0.15
0.00
−0.15
−0.30
−0.45

Timings

```
------------------------------------------------------------------------------------------------
 OPTIMAL TIME STANDS AT:   10.2038803 sec -> current standard

 without mass array    255.950150
                                     (slight performance improvements by avoiding too many array calls)
    with mass array    152.530518
                                     (half mass calc operations; inner loop)
sin/cos inside loop    119.338303
                                     (one cos calculation eliminated; inner loop)
   with denominator    57.8149910
                                     (denominator using sqrt*sqrt*sqrt instead **1.5)
with sin/cos tables    17.7189388
                                     (calculating cos/sin in external loop)
         mass table    15.7216377
                                     (calculating the mass in external loop)
  with r_iprime_cos    14.8603277
                                     (saves a few calculations in the inner loop since it appears twice there)
 optimizer flag -O1    10.2038803
                                     (remembered that gnu compilers have no optimization default)
     custom invsqrt    84.1394730
                                     (sadly much slower than intrinsic 1./sqrt,
                                           but the error is actually not that bad)
------------------------------------------------------------------------------------------------
```