Optimization flow for the complete relative self gravity calculations
Computational Science II - University of Zurich

Mihai Tomozeiu

14th March 2015

1 Raw code - *gravity.f90*

The original code without any optimization is listed beneath:

```
program gravity
!Variable section
implicit none
INTEGER i, j, k, l
REAL a, G, dr, dth
double precision t_init, t_end, timed
REAL, dimension(1:128) :: radius
REAL, dimension(1:256) :: theta
REAL, dimension(1:128,1:256) :: density, acc_r, acc_t
REAL, dimension(1:2) :: acc


!Reading Section

OPEN(UNIT = 2, FILE ='/home/ics/mihai/git/Computational_Science_II/Data/r_proje
OPEN(UNIT = 3, FILE ='/home/ics/mihai/git/Computational_Science_II/Data/theta_p
OPEN(UNIT = 4, FILE ='/home/ics/mihai/git/Computational_Science_II/Data/density



do i = 1, 128
        read(2,*) a
        radius(i) = a

enddo




do i = 1, 256
        read(3,*) a
        theta(i) = a
enddo


do i = 1, 128
        do j = 1, 256
                read(4,*) a
                density(i,j) = a
        enddo
enddo

close(2)
close(3)
close(4)
OPEN(UNIT = 5, FILE='/home/ics/mihai/git/Computational_Science_II_Open/force_r_
```

```
OPEN(UNIT = 6, FILE='/home/ics/mihai/git/Computational_Science_II_Open/force_t_
!Calculations section

call cpu_time(t_init)

G   = 1
dr  = radius(2)-radius(1)
dth = theta(2) - theta(1)
do i = 1, Nr
      do j = 1, Nt


           acc(1) = 0
           acc(2) = 0
           do k = 1,Nr
              do l = 1, Nt
                 acc(1) = acc(1) + density(k,l)*dr*dth*radius(k)* &
                          (radius(i) +dr/2 -radius(k)*cos(theta(j)-theta(l)+dth/2
                          /(radius(k)**2 + (radius(i)+dr/2)**2 -2*radius(k) &
                          *(radius(i)+dr/2)*cos(theta(l)-theta(j)+dth/2))**(1.5)

                 acc(2) = acc(2) + density(k,l)*dr*dth*radius(k)* &
                          sin(theta(j)-theta(l)+dth/2) &
                          /(radius(k)**2 + (radius(i)+dr/2)**2 -2*radius(k) &
                          *(radius(i)+dr/2)*cos(theta(l)-theta(j)+dth/2))**(1.5)
              enddo
            enddo


            acc_r(i,j) = acc(1)
            acc_t(i,j) = acc(2)
            write(5,*) acc_r(i,j)
            write(6,*) acc_t(i,j)
        enddo
enddo

call cpu_time(t_end)




close(5)
close(6)

OPEN(UNIT = 7, FILE='/home/ics/mihai/git/Computational_Science_II_Open/time.dat
write(7,*) t_end - t_init, t_init, t_end

close(7)

end program gravity
```

A cpu time measurement on a single core on f3 login node of zbox for the code revealed a value of **409.2s**.

2 First set of pre-calculations - *gravity_precalc.f90*

First of all, the values 128 and 256 for r and θ were assigned to the variables nr and nt to allow consistent changes of these quantities in the calculation 'for's. The mass map $mass(k, l)$ associated with the grid was computed prior to the main 'four-for':

```
do k = 1, nr
        do l = 1, nt
                mass(k,l) = density(k,l)*radius(k)*dr*dth
        enddo
enddo
```

Next, all simple and complex trigonometric functions found inside the four-for were pre-computed:

```
do i = 1, nt
        cos_center(i) = cos(theta(i))
        cos_corner(i) = cos(theta(i)+dth/2)
        sin_center(i) = sin(theta(i))
        sin_corner(i) = sin(theta(i)+dth/2)
enddo
do l = 1, nt
        do j = 1, nt
                cos_diff(l,j) = cos(theta(l)-theta(j) + dth/2)
        enddo
enddo
```

Moreover, the value of the radii at the corners were determined using one for:

```
!Precalculating the corner radii
do i = 1, nr
        radius_corn(i) = radius(i) + dr/2
```

With the use of the pre-calculated quantities, the main part was transformed into:

```
do i = 1, nr
        do j = 1, nt

        !acceleration calculations for a single cell
        !projection Cartesian
          acc(1) = 0
          acc(2) = 0
          do k = 1,nr
              do l = 1, nt
                  acc(1) = acc(1) + mass(k,l)* &
                          (radius(k)*cos_center(l) - &
```

```
                        radius_corn(i)*cos_corner(j)) &
                        /(radius(k)**2 + radius_corn(i)**2 -2*radius(k) &
                        *radius_corn(i)*cos_diff(l,j))**(1.5)

            acc(2) = acc(2) + mass(k,l)* &
                        (radius(k)*sin_center(l) - &
                         radius_corn(i)*sin_corner(j)) &
                         /(radius(k)**2 + radius_corn(i)**2 -2*radius(k) &
                         *radius_corn(i)*cos_diff(l,j))**(1.5)
          enddo
        enddo

      !projection Polar
        acc_r(i,j) = acc(1)*cos_center(j) + acc(2)*sin_center(j)
        acc_t(i,j) = acc(2)*cos_center(j) - acc(1)*sin_center(j)
        write(5,*) acc_r(i,j)
        write(6,*) acc_t(i,j)
```

The measured cpu time for the resulting **gravity$_{\mathbf{p}}$recalc.f90** was **235s**. Thus, the use of the first set of precomputed quantities led to a reductio of the cpu time of 43%.

3 Reformulation of the acceleration calculations - $gravity\_reduct.f90$

The next step is to rewrite the acceleration equations in terms of radial ratio $radius(k)/radius\_corn(i)$.

```
acc(1) = acc(1) + mass(k,l)* &
         (radius(k)/radius_corn(i)*cos_center(l) - &
         cos_corner(j)) &
         /((radius(k)/radius_corn(i))**2 +1 -2*radius(k) &
         /radius_corn(i)*cos_diff(l,j))**(1.5)/radius_corn(i)**2
acc(2) = acc(2) + mass(k,l)* &
         (radius(k)/radius_corn(i)*sin_center(l) - &
         sin_corner(j)) &
         /((radius(k)/radius_corn(i))**2 + 1 -2*radius(k) &
         /radius_corn(i)*cos_diff(l,j))**(1.5)/radius_corn(i)**2
```

Since for this step no precalculations and operation optimization were done the cpu time associated to the calculations grew by 6% to the value of 249 s. The time increase is due to the extra operations introduced, including the extra divisions which are slow. In the next two sections time cost improvements were made with another wave of precalculated quantities and a reduction of the number of calculations per cell.

4 Second set of pre-calculations - $gravity\_precalc\_2.f90$

The ratio of the radii defined as $R = radius(k)/radius\_corn(i)$ appears in the acceleration calculations as standalone or its square. Moreover, the square appears only when added to unity therefore the addition of one was removed from the four-for and included in the precalculation fors over k and i.

```
do k = 1,nr
        do i = 1, nr
                ratio(k,i) = radius(k)/radius_corn(i)
                ratio_f(k,i) = ratio(k,i)*ratio(k,i) + 1
        enddo
enddo
```

Moreover, the multiplication by a factor of two with the cosine difference was introduced into the precalculation fors:

```
do l = 1, nt
        do j = 1, nt
                cos_diff(l,j) = 2*cos(theta(l)-theta(j) + dth/2)
        enddo
enddo
```

The previous modifications bring the acceleration part of the code to the following form:

```
acc(1) = acc(1) + mass(k,l)* &
         (ratio(k,i)*cos_center(l) - &
         cos_corner(j)) &
         /(ratio_f(k,i) -ratio(k,i) &
         *cos_diff(l,j))**(1.5)/radius_corn(i)**2
acc(2) = acc(2) + mass(k,l)* &
         (ratio(k,i)*sin_center(l) - &
         sin_corner(j)) &
         /(ratio_f(k,i) -ratio(k,i) &
         *cos_diff(l,j))**(1.5)/radius_corn(i)**2
```

These changes lead to a minor reduction of the computational time by 3% to **242s**.

5 First operation optimization - *gravity_optimiz.f90*

In this step the greatest improvement in the computational time was achieved. The acceleration calculation part was transformed into:

```
!reducing double operations
rad     = ratio_f(k,i) -ratio(k,i) &
          *cos_diff(l,j)
den     = rad * sqrt(rad)
den_inv = 1./den
comm    = mass(k,l)*radius_corn_2_inv(i)*den_inv
!acc calculations

acc(1) = acc(1) + comm* &
         (ratio(k,i)*cos_center(l) - &
         cos_corner(j))
```

```
            acc(2) = acc(2) + comm* &
                     (ratio(k,i)*sin_center(l) - &
                     sin_corner(j))
```

First of all the redundant double operations were removed. What is common to both projections of accelerations was calculated separately and then introduced in the acceleration equations. In the denominator there are a quadratic function in terms of the ratio and the square of the corner radius (i). The inverse of the square of the corner radius was precomputed in order to remove the time consuming divisions and the extra square. This was easily done with a single for along the radius indices:

```
do i = 1, nr
        radius_corn(i) = radius(i) + dr/2
        radius_corn_2_inv(i) = 1./(radius_corn(i)*radius_corn(i))
enddo
```

The same for as the one precomputing the corner radii was used. The quadratic function dependent on all four indices was first computed and assigned to variable *rad*. Next the quantity was multiplied with its square root to avoid the direct 3/2 power law. Next it was inverted and assigned to variable *den_inv*. The value then was multimplied with $mass(k,l)$ and the inverted square corner radius. This multiplication assigned to variable *com* was used as a prefactor in computing the values of the projections of the acceleration.

The computational time was reduced by 91% to **22.8s**. The main contribution to the improvement is the minimization of the number of divisions in the four-for.

6 Exploiting the trigonometric symmetry - *gravity_theta_symmetry.f*

In this section the length of one of the four-fors is halved (the first for along the values of theta). This decrease of the number of steps can be done by exploiting the following trigonometric identities:

$$cos(\theta + \pi) = cos(\theta) \cdot cos(\pi) - sin(\theta) \cdot sin(\pi) = -cos(\theta) \tag{1}$$

$$sin(\theta + \pi) = sin(\theta) \cdot cos(\pi) + sin(\pi) \cdot cos(\theta) = -sin(\theta) \tag{2}$$

Thus, the difference in the calculations of two values of θ different by π is a set of factors of -1.

The four-for has the following form:

```
do i = 1, nr
   do j = 1, nt/2

   !acceleration calculations for a single cell
   !projection Cartesian
      acc(1) = 0
      acc(2) = 0
      acc(3) = 0
      acc(4) = 0
```

```
    do k = 1,nr
       do l = 1, nt
          !reducing double operations
          rad     = ratio_f(k,i) -ratio(k,i) &
                    *cos_diff(l,j)
          den     = rad * sqrt(rad)
          den_inv = 1./den
          comm    = mass(k,l)*radius_corn_2_inv(i)*den_inv
          !acc calculations

          acc(1) = acc(1) + comm* &
                   (ratio(k,i)*cos_center(l) - &
                   cos_corner(j))


          acc(2) = acc(2) + comm* &
                  (ratio(k,i)*sin_center(l) - &
                  sin_corner(j))
          !exploting symmetry in theta along j
          rad     = ratio_f(k,i) +ratio(k,i) &
                    *cos_diff(l,j)
          den     = rad * sqrt(rad)
          den_inv = 1./den
          comm    = mass(k,l)*radius_corn_2_inv(i)*den_inv

          acc(3) = acc(3) + comm* &
                   (ratio(k,i)*cos_center(l) + &
                   cos_corner(j))


          acc(4) = acc(4) + comm* &
                   (ratio(k,i)*sin_center(l) + &
                   sin_corner(j))




       enddo
    enddo

    !projection Polar
    acc_r(i,j) = acc(1)*cos_center(j) + acc(2)*sin_center(j)
    acc_t(i,j) = acc(2)*cos_center(j) - acc(1)*sin_center(j)
    !the other pi

    acc_r(i,j+nt/2) = -acc(3)*cos_center(j) - acc(4)*sin_center(j)
    acc_t(i,j+nt/2) = -acc(4)*cos_center(j) + acc(3)*sin_center(j)


enddo
```

enddo

With the j-for halved, the values associated with j in $[nt/2+1..nt]$ are computed along with the ones from the first half by changing the sign in front of any term containing a *sin* or *cos* trigonometric function of a linear function in $\theta(j)$. This change brings a decrease of the computational time equal to 3% of the previous version cpu-time. The value of the time needed to complete the calculations was **22.1s**.

7 Third set of precalcutations - $gravity\_precalc\_3.f90$

In the previous changes made to the code that implied precalculations of quantities, the sets were limited to a for or two. In the current section, three-fors are used for precalculations of some common quantities.

```
! 3 do precalculation
do k = 1, nr
          do i = 1, nr
                    do l = 1, nt
                              surf(k,l,i) = mass(k,l)*radius_corn_2_inv(i)


                              !projection for 1 -- nt/2
                              proj_1(k,i,l) = ratio(k,i)*cos_center(l)
                              proj_2(k,i,l) = ratio(k,i)*sin_center(l)

                    enddo
```

One of the quantities has the units of a surface density, it is effectively the mass of the cell divided by the square of the radius at the corner i. The other two are parts of the projections which are independent of j.

With these values precomputed the resulting computations inside the four-for are:

```
do k = 1,nr
    do l = 1, nt
          !reducing double operations
          rad      = ratio_f(k,i) -ratio(k,i) &
                     *cos_diff(l,j)
          den      = rad * sqrt(rad)
          comm     = surf(k,l,i)/den
          !acc calculations



          acc(1) = acc(1) + comm* &
                   (proj_1(k,i,l) - &
                   cos_corner(j))
```

```
acc(2) = acc(2) + comm* &
         (proj_2(k,i,l) - &
         sin_corner(j))

!exploiting symmetry in theta along j

rad    = ratio_f(k,i) +ratio(k,i) &
           *cos_diff(l,j)


den    = rad * sqrt(rad)
```

These changes further reduce the computational time by 13% to a value of **19.3s**.

8 Second optimization - *gravity_optimiz_2.f90*

For the final optimization the core product of the four-for $prod = ratio(k,i) * cos\_diff(l,j)$ common to all four projection calculations has been computed prior to the rad quantity estimate.

This change avoids an extra multiplication and reduces the cpu time by 5%. The final measured value for the computing time was **18.3s**.

The final form of the entire code can be seen in the following pages:

```
program gravity
!Variable section
implicit none
INTEGER i, j, k, l, nr, nt
REAL a, G, dr, dth, rad, den ,den_inv, comm, prod
double precision t_init, t_end, timed
REAL, dimension(1:128) :: radius, radius_corn, radius_corn_2_inv
REAL, dimension(1:256) :: theta, cos_center, cos_corner, sin_center, sin_corner
REAL, dimension(1:128,1:256) :: density, acc_r, acc_t, mass
REAL, dimension(1:256,1:256) :: cos_diff
REAL, dimension(1:128,1:128) :: ratio, ratio_f
REAL, dimension(1:4) :: acc
REAL, dimension(1:128,1:128,1:256) :: proj_1, proj_2, proj_3, proj_4
REAL, dimension(1:128,1:256,1:128) :: surf

nr = 128
nt = 256


!Reading Section

OPEN(UNIT = 2, FILE ='/home/ics/mihai/git/Computational_Science_II/Data/r_proje
OPEN(UNIT = 3, FILE ='/home/ics/mihai/git/Computational_Science_II/Data/theta_p
OPEN(UNIT = 4, FILE ='/home/ics/mihai/git/Computational_Science_II/Data/density
```

```
do i = 1, nr
        read(2,*) a
        radius(i) = a

enddo




do i = 1, nt
        read(3,*) a
        theta(i) = a
enddo


do i = 1, nr
        do j = 1, nt
                read(4,*) a
                density(i,j) = a
        enddo
enddo

close(2)
close(3)
close(4)
OPEN(UNIT = 5, FILE='/home/ics/mihai/git/Computational_Science_II_Open/acc_r_pr
OPEN(UNIT = 6, FILE='/home/ics/mihai/git/Computational_Science_II_Open/acc_t_pr
!Calculations section



call cpu_time(t_init)

G   = 1
dr  = radius(2)-radius(1)
dth = theta(2) - theta(1)


!Precalculating the trig. functions
do i = 1, nt
        cos_center(i) = cos(theta(i))
        cos_corner(i) = cos(theta(i)+dth/2)
        sin_center(i) = sin(theta(i))
        sin_corner(i) = sin(theta(i)+dth/2)
enddo
do l = 1, nt
        do j = 1, nt
                cos_diff(l,j) = 2*cos(theta(l)-theta(j) + dth/2)
        enddo
```

```
enddo


!Precalculating the corner radii
do i = 1, nr
        radius_corn(i) = radius(i) + dr/2
        radius_corn_2_inv(i) = 1./(radius_corn(i)*radius_corn(i))
enddo

!Precalculating the mass

do k = 1, nr
        do l = 1, nt
                mass(k,l) = density(k,l)*radius(k)*dr*dth
        enddo
enddo

!Precalculating the radius ratio functions

do k = 1,nr
        do i = 1, nr
                ratio(k,i) = radius(k)/radius_corn(i)
                ratio_f(k,i) = ratio(k,i)*ratio(k,i) + 1
        enddo
enddo

! 3 do precalculation
do k = 1, nr
        do i = 1, nr
                do l = 1, nt
                        surf(k,l,i) = mass(k,l)*radius_corn_2_inv(i)


                        !projection for 1 -- nt/2
                        proj_1(k,i,l) = ratio(k,i)*cos_center(l)
                        proj_2(k,i,l) = ratio(k,i)*sin_center(l)

                enddo

        enddo
enddo


do i = 1, nr
        do j = 1, nt/2

        !acceleration calculations for a single cell
        !projection Cartesian
                acc(1) = 0
                acc(2) = 0
```

```fortran
                acc(3) = 0
                acc(4) = 0
                do k = 1,nr
                        do l = 1, nt
                                !reducing double operations
                                prod    = ratio(k,i)*cos_diff(l,j)

                                rad     = ratio_f(k,i) - prod

                                den     = rad * sqrt(rad)
                                comm    = surf(k,l,i)/den
                                !acc calculations

                                acc(1) = acc(1) + comm* &
                                        (proj_1(k,i,l) - &
                                        cos_corner(j))

                                acc(2) = acc(2) + comm* &
                                        (proj_2(k,i,l) - &
                                        sin_corner(j))
                                !exploiting symmetry in theta along j
                                rad     = ratio_f(k,i) + prod
                                den     = rad * sqrt(rad)
                                comm    = surf(k,l,i)/den

                                acc(3) = acc(3) + comm* &
                                        (proj_1(k,i,l) + &
                                        cos_corner(j))

                                acc(4) = acc(4) + comm* &
                                        (proj_2(k,i,l) + &
                                        sin_corner(j))

                        enddo
                enddo

        !projection Polar
                acc_r(i,j) = acc(1)*cos_center(j) + acc(2)*sin_center(j)
                acc_t(i,j) = acc(2)*cos_center(j) - acc(1)*sin_center(j)
        !the other pi

                acc_r(i,j+nt/2) = -acc(3)*cos_center(j) - acc(4)*sin_center(j)
                acc_t(i,j+nt/2) = -acc(4)*cos_center(j) + acc(3)*sin_center(j)


        enddo
enddo

do i = 1, nr
        do j = 1, nt
```

```
                write(5,*) acc_r(i,j)
                write(6,*) acc_t(i,j)
        enddo
enddo


call cpu_time(t_end)



close(5)
close(6)

OPEN(UNIT = 7, FILE='/home/ics/mihai/git/Computational_Science_II_Open/time_opt

write(7,*) t_end - t_init, t_init, t_end

close(7)

end program gravity
```

9 Summary of time measurements, number of operations and results

The results of the *gravity_optimiz_2.f90* code are displayed in the figure 1.

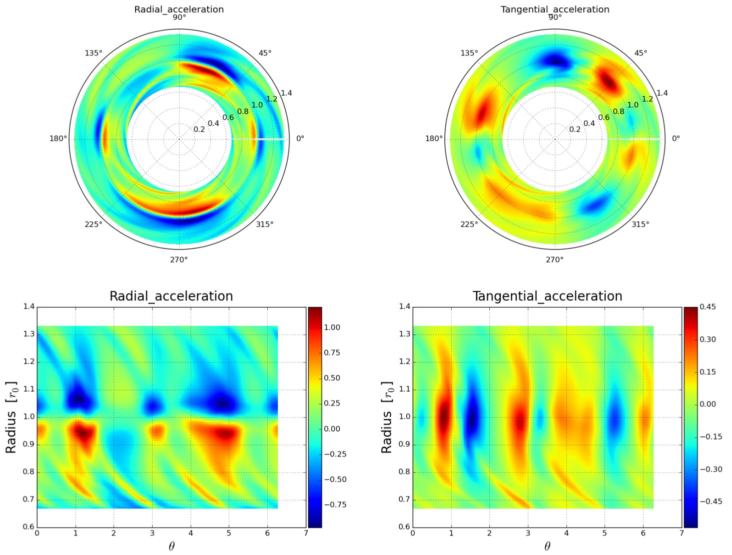

Figure 1: The results of $gravity\_optimiz\_2.f90$ are displayed. The first/second column corresponds to the radial/tangetial component of the acceleration due to relative self-gravity. First row displayes the values on a polar map, while the second displays them on a rectangular grid for easier visualization.

Table 1: Occurances of the mathematical operations and number of variable assignments (var) that depend on the number of angles and radii.

Operation type	Number count
+	$\frac{7}{2} \cdot N_r^2 \cdot N_t^2 + N_r \cdot N_t + N_r^2 + N_t^2 + 2 \cdot N_t$
-	$\frac{3}{2} \cdot N_r^2 \cdot N_t^2 + N_r \cdot N_t + N_t^2$
*	$\frac{7}{2} \cdot N_r^2 \cdot N_t^2 + 3 \cdot N_r^2 \cdot N_t + 6 \cdot N_r \cdot N_t + N_r^2 + N_t^2 + N_r$
/	$N_r^2 \cdot N_t^2 + N_r^2 + N_r$
cos	$2 \cdot N_t$
sin	$2 \cdot N_t$
sqrt	$N_r^2 \cdot N_t^2$
var	$\frac{11}{2} \cdot N_r^2 \cdot N_t^2 + 3 \cdot N_r^2 \cdot N_t + 5 \cdot N_r \cdot N_t + 2 \cdot N_r^2 + N_t^2 + 4 \cdot N_t + 2 \cdot N_r$

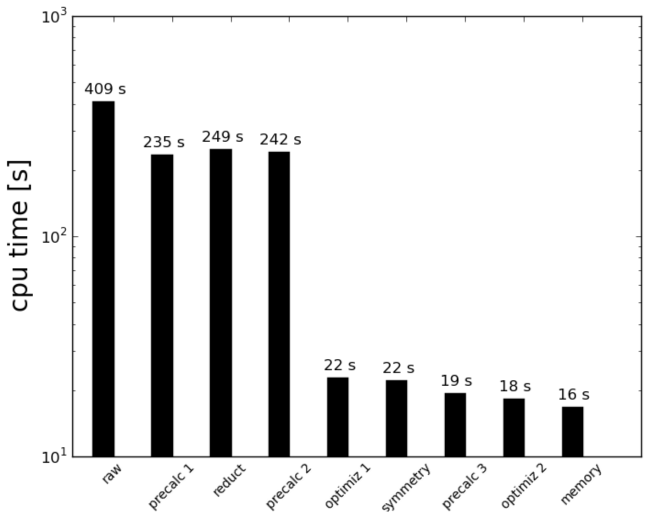

Figure 2: The computing time for each version of the code run on zbox f3 is presented in a logarithmic plot.

The operations counts for the last version of the code are enummerated in table 1. Finally the changes of the cpu time for the evolving code can be seen in figure 2.