Compact Stars

via Smoothed Particle Hydrodynamics

Matthew Portman

Computational Science Research Center
San Diego State University

May 1, 2018

Outline

Compact Object Physics and Rotation

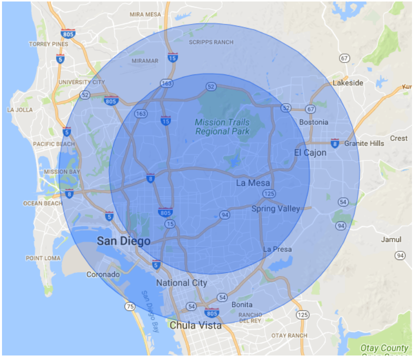
Compact Object Physics and Rotation

Compact Objects: A Primer

- Classical Mechanics and General Relativity.
- White Dwarfs, Black Holes

Compact Objects: A Primer

- Classical Mechanics and General Relativity.
- White Dwarfs, Black Holes
- ...And Neutron Stars.



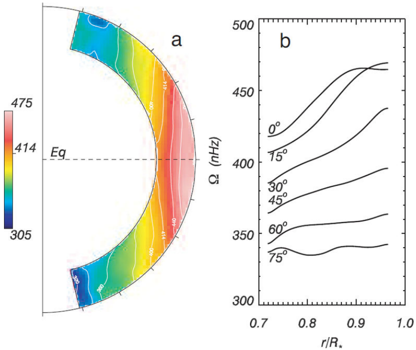
A neutron star the size of San Diego
Thanks to: Google Maps and obeattie.github.io

(Differential) Rotation

- Rotation in Compact Objects
- Differential Rotation AKA Rotating w.r.t. Rotation

$$j(\Omega) = A^2(\Omega_c - \Omega)$$
$$\frac{\Omega}{\Omega_c} = \frac{A^2}{A^2 + r^2 sin^2\theta}$$
$$A = aR_{\text{star}}$$

Muller and Eriguchi (1985)



Differential Rotation of the Sun (Brun et. al. 2004)

Smoothed Particle Hydrodynamics

SPH: What is it? How does it work? Let's find out!

Smoothed Particle Hydrodynamics (SPH)

- The N-Body Problem and the Mean-Field approximation... solving the Advection Equation.

Smoothed Particle Hydrodynamics (SPH)

- The N-Body Problem and the Mean-Field approximation... solving the Advection Equation.
- SPH Mathematics: the Basics

$$\mathcal{L}_{sph} = \sum_{i=1}^{N} m_i [\frac{1}{2} v_i^2 - u(\rho_i)]$$

$$A(r) = \int A(r') W(r - r', h) dr' \rightarrow A(r) = \sum_b m_b \frac{A_b}{\rho_b} W(r - r_b, h)$$

The Smoothing Kernel W

$$W = \frac{1}{\pi h^2} e^{-\frac{|r|^2}{2h^2}}; \quad W_{ij} = W(|r_i - r_j|; h)$$

Smoothed Particle Hydrodynamics (SPH) - More Physics

- After some Euler-Lagranging... (Monaghan 1992)

$$\rho(r) = \sum_j m_j W(|r - r_j|; h)$$

$$\frac{dv_i}{dt} = -\sum_j m_j (\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2}) \nabla_i W_{ij}$$

$$\frac{du_i}{dt} = \frac{1}{2} \sum_j m_j (\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2}) v_{ij} \cdot \nabla_i W_{ij}$$

Smoothed Particle Hydrodynamics (SPH) - More Physics

- After some Euler-Lagranging... (Monaghan 1992)

$$\rho(r) = \sum_j m_j W(|r - r_j|; h)$$

$$\frac{dv_i}{dt} = -\sum_j m_j \left(\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2}\right) \nabla_i W_{ij}$$

$$\frac{du_i}{dt} = \frac{1}{2} \sum_j m_j \left(\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2}\right) v_{ij} \cdot \nabla_i W_{ij}$$

- Other considerations include:

$$\Pi_{ij},\ h \to h(i),\ \text{and damping}$$

Implementation

Implementation

or: How I learned to Stop Doing Everything at Once

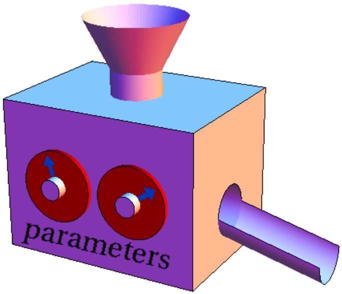
Initial Conditions and Parameters

- Spatial Discretization?
  Grid/Mesh? Boundaries?
- Initial Conditions: x, v, u, ρ
- Equation of State - Polytropic (simple)
- Parameters: k, ν, λ, a

$$P = k\rho^2$$

$$a_i = -nuv_i - \lambda x_i - a_{i,v1}$$

Monaghan and Price (2004)

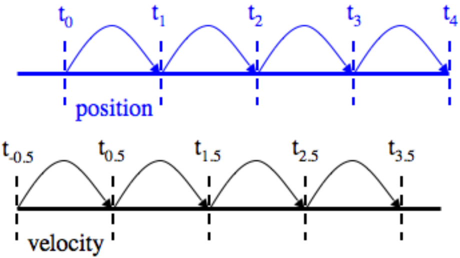


Nykamp DQ, "Function machine parameters." From Math Insight.

Leapfrogging

- Leapfrogging Through Time

$$v_{n+\frac{1}{2}} = v_{n-\frac{1}{2}} + a_n \Delta t$$
$$x_{n+1} = \Delta t v_{n+\frac{1}{2}} + x_n$$



Leapfrogging for position and velocity

Parallel Approach

Parallel Approach feat. CUDA

Parallelization

- Solving the Advection Equation: Forward Differencing $\rightarrow$ Leap-Frog

for $t = 1$ **to** $MaxTime$ **do**
    Update $\vec{x}$;
    Update $\vec{v}$;
    Initialize $\vec{a}$;
    for $i = 1$ **to** N **do**
        for $j = i + 1$ **to** N **do**
            $\vec{a}_i \mathrel{+}= ...\nabla W_{ij}$;
            $\vec{a}_j \mathrel{-}= ...\nabla W_{ij}$;
            $ut_i \mathrel{+}= ... \cdot \nabla W_{ij}$;
            $ut_j \mathrel{+}= ... \cdot \nabla W_{ij}$;
        end
    end
end

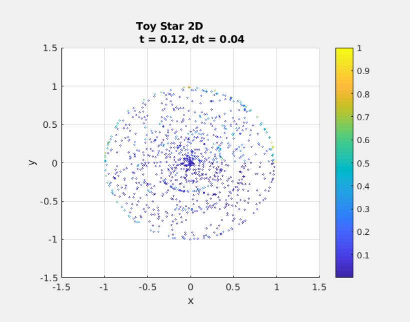
Implementation

```
! PARALLELIZING -------------------------------

  tblock = dim3(32,32,1)
  grid = dim3(ceiling(real(maxn)/tblock%x),ceiling(real(maxn)/tblock%y),1)

  pos_d(:,:) = pos(:,:)

  CALL gweighf<<<grid, tblock>>>(pos_d, hsm, grad_d)

  grad(:,:) = grad_d(:,:)

  do i = 1,maxn

    ratioi = P(i)/(rho(i)*rho(i))

    do j = i+1, maxn

      ratioj = P(j)/(rho(j)*rho(j))

! Monaghan & Price (2004)
! Acceleration due to pressure

      temp = m(j)*(ratioi + ratioj)

      accel(:,i) = accel(:,i) - temp*grad(1,j)
      accel(:,j) = accel(:,j) + temp*grad(2,j)

! Thermal Energy per unit mass/dt.

      temp = temp*abs(dot_product(vel(:,i)-vel(:,j),grad(:,j)))

      ut(i) = ut(i) + temp
      ut(j) = ut(j) + temp
! Assume no thermal conduction between adjacent particles. Just another term that would have to be added.
    enddo

  enddo
```

Results

Normalized x, y, Energy/time:

- Without rotation (of any kind)
- Without differential rotation
- With both kinds of rotation



Conclusions and Extensions

- Energy and (Angular) Momentum Conservation.
- Boundary Conditions and the extent of usefulness of the 'toy' model.
- Better approximation to self-gravity and relativistic approaches.
- Extending to 3D.

Conclusions and Extensions for Parallelization

- Expected Speed-up
- Parameterization of the model
- Extending to 3D in problem space and CUDA

Thanks to:

- CSRC and SDSU
- Dr. Peter Blomgren
- Dr. Fridolin Weber
- Phillip Mocz

This work was supported by a STEM scholarship award funded by the National Science Foundation grant DUE-1259951, PHY-1714068, and the Computational Science Research Center at SDSU.
Further references available by request.