Gravity, Particles, and Cosmology with FLASH

Paul Ricker

University of Illinois at Urbana-Champaign
National Center for Supercomputing Applications

http://www.astro.uiuc.edu/~pmricker/research/codes/flashcosmo/

ILLINOIS

NCSA

Overview

- The big picture
  - Applications for gravity and particles
  - Equations to be solved
- Methods
  - Algorithms currently in FLASH
  - Performance
- Usage
  - Organization of the code modules
  - Initializing a particle application
  - Analyzing particle output
  - Examples

The big picture

Gravity applications

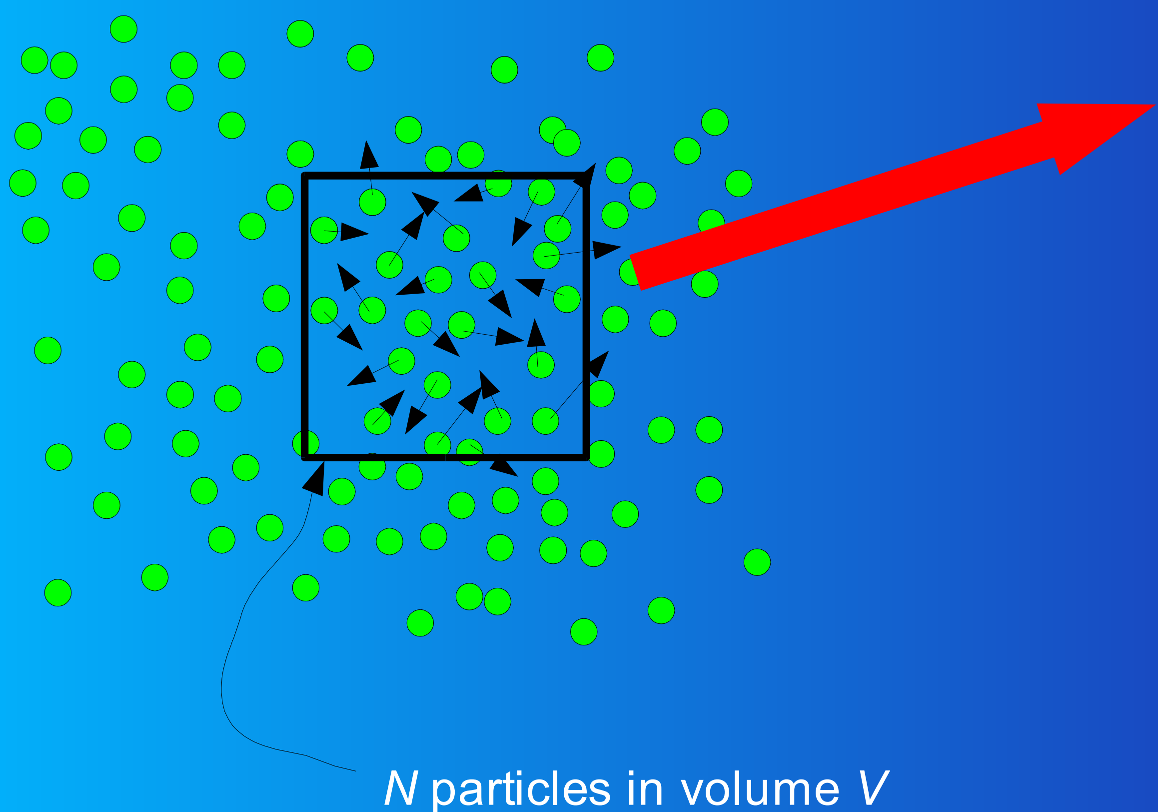
- Atmospheres (externally imposed fields)
  - Accretion (e.g., onto compact objects)
  - Buoyancy (e.g., bubbles, nuclear runaway)
- Nearly equilibrium self-gravitating systems
  - Stellar interiors
  - Intracluster medium
- Collapse and explosion problems
  - Galaxy formation
  - Star formation
  - Supernovae

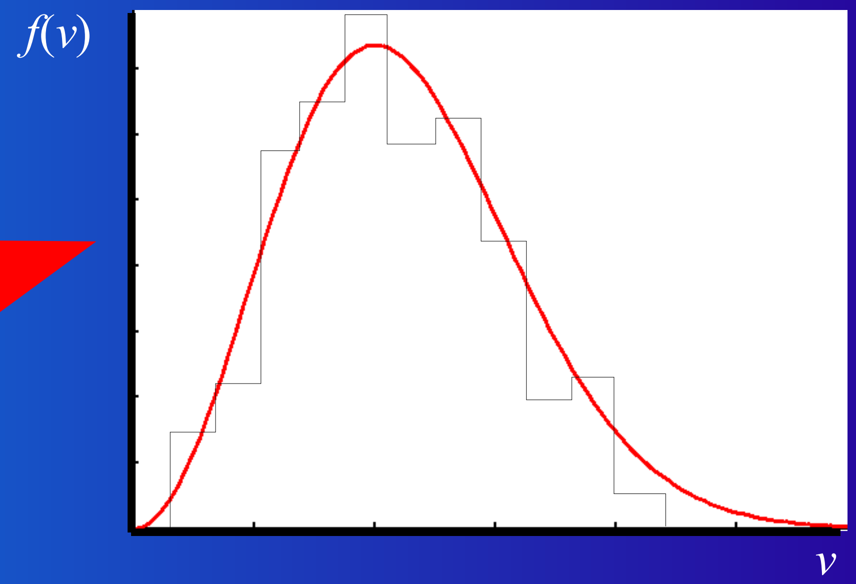
Particle applications

- Dark matter
  - Large-scale structure formation
  - Galaxies
  - Galaxy clusters
- Stars
  - Galaxies
  - Globular clusters
  - Compact objects
- Tracer particles
  - Supernova nucleosynthesis
  - Mixing problems
- Electrons and ions
  - Plasmas

Particle simulation

Monte Carlo sampling of particle distribution function (gas, dust, dark matter)





$$n(\mathbf{x}) = \int f(\mathbf{x}, v)\, dv \approx \frac{N}{V}$$

$$\frac{|f_N - f_{\text{true}}|}{|f_{\text{true}}|} \propto N^{-1/2}$$

Basic requirements:

- As $N \to \infty$, error ("shot noise") in approximate distribution function f_N goes to 0
- As $N \to \infty$, equation describing evolution of f_N becomes the Boltzmann equation

Equations – proper coordinates

Euler equations for gasdynamics:

Mass
$$\frac{\partial}{\partial t}\rho+\nabla\cdot\left(\rho\,\mathbf{v}\right)=0$$

Momentum
$$\frac{\partial}{\partial t}\left(\rho\,\mathbf{v}\right)+\nabla\cdot\left(\rho\,\mathbf{v}\,\mathbf{v}\right)+\nabla\,p=-\rho\,\nabla\,\phi$$

Energy
$$\frac{\partial}{\partial t}\left(\rho\,E\right)+\nabla\cdot\left[\left(\rho\,E+p\right)\mathbf{v}\right]=-\rho\,\mathbf{v}\cdot\nabla\,\phi$$

$$\rho\,E\equiv\frac{p}{\gamma-1}+\frac{1}{2}\rho\,v^2$$

Newton's laws for particles:

$$\frac{\partial\,\mathbf{x}}{\partial\,t}=\mathbf{v}$$

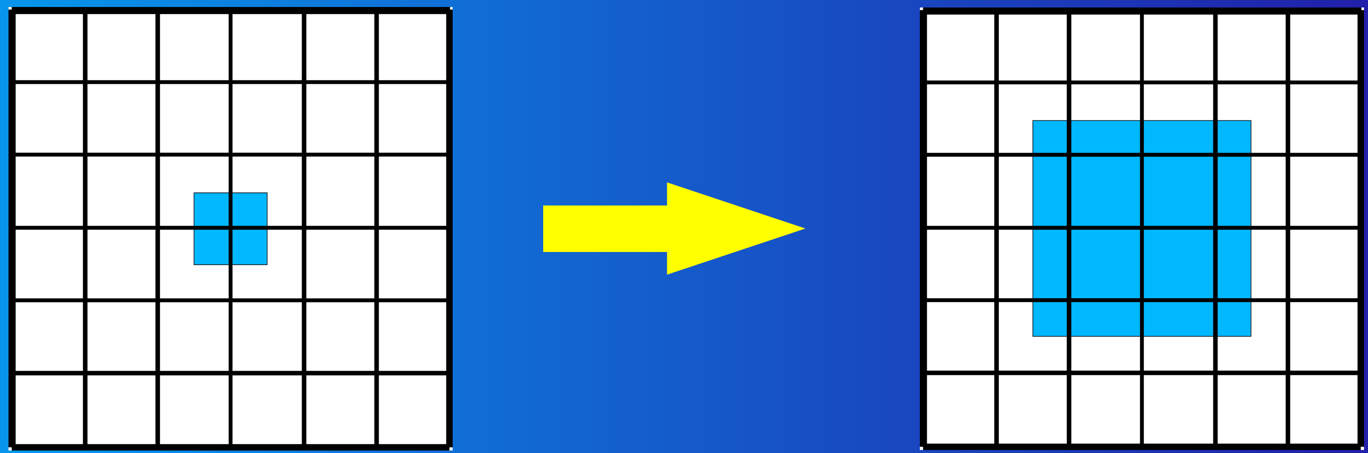
$$\frac{\partial\,\mathbf{v}}{\partial\,t}=-\nabla\,\phi$$

Poisson equation for gravitational potential:

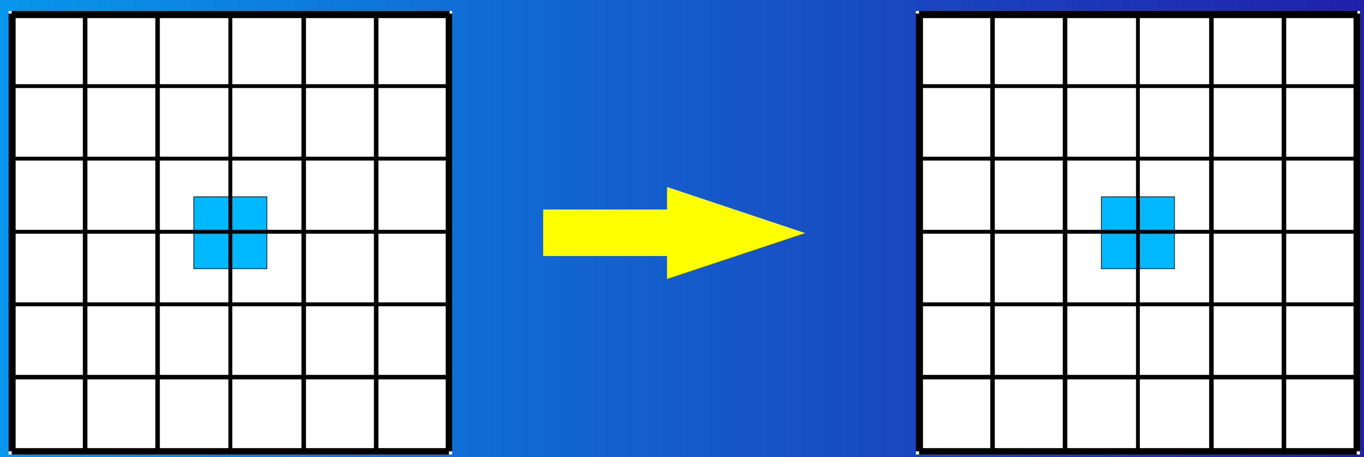
$$\nabla^2\phi=4\,\pi\,G\,\rho$$

Comoving coordinates

Proper coordinates **r** are Eulerian: $\mathbf{u} = \dot{\mathbf{r}}$



Comoving coordinates **x** scale out the expansion: $\mathbf{v} = \dot{\mathbf{x}}$



$$\mathbf{r} = a\,\mathbf{x} \;\;\Rightarrow\;\; \mathbf{u} = a\,\dot{\mathbf{x}} + \dot{a}\,\mathbf{x} = a\,\mathbf{v} + H\,\mathbf{r}$$

$H \equiv \dfrac{\dot{a}}{a}$ is the Hubble parameter

Fluid equations – comoving coordinates

Comoving variables: $\rho \equiv a^3 \tilde{\rho}, \quad p \equiv a\, \tilde{p}, \quad T \equiv \frac{\tilde{T}}{a^2}$

Fluid equations for gasdynamics:

Mass $$\frac{\partial}{\partial t}\rho + \nabla \cdot (\rho \mathbf{v}) = 0$$

Momentum $$\frac{\partial}{\partial t}(\rho \mathbf{v}_g) + \nabla \cdot (\rho \mathbf{v} \mathbf{v}) + \nabla p + 2\frac{\dot{a}}{a}\rho \mathbf{v} = -\rho \nabla \phi$$

Energy $$\frac{\partial}{\partial t}(\rho E) + \nabla \cdot \left[(\rho E + p)\mathbf{v}\right] + \frac{\dot{a}}{a}\left[\left(\frac{3\gamma - 1}{\gamma - 1}\right)p + 2\rho v^2\right] = \rho \frac{\partial \phi}{\partial t}$$

$$\rho E \equiv \frac{p}{\gamma - 1} + \frac{1}{2}\rho v^2 + \rho \phi$$

Particle/field equations – comoving

Particle equations:

$$\frac{\partial \mathbf{x}}{\partial t} = \mathbf{v}$$

$$\frac{\partial \mathbf{v}}{\partial t} + 2\frac{\dot{a}}{a}\mathbf{v} = -\nabla \phi$$

Gravity (Poisson & Friedmann equations):

$$\nabla^2 \phi = \frac{4\pi G}{a^3}\left[(\rho_{\mathrm{g}} + \rho_{\mathrm{dm}}) - \overline{(\rho_{\mathrm{g}} + \rho_{\mathrm{dm}})} \right]$$

$$H^2(t) \equiv \left(\frac{\dot{a}}{a} \right)^2 = H_0^2 \left(\frac{\Omega_{\mathrm{m}}}{a^3} + \frac{\Omega_{\mathrm{r}}}{a^4} + \Omega_\Lambda - \frac{\Omega_{\mathrm{c}}}{a^2} \right)$$

Methods

Gravity algorithms in FLASH

- Externally imposed fields
  - Constant field ($\mathbf{g}$ = constant)
  - Point mass ($\mathbf{g} = -GM\mathbf{r}/r^3$)
  - Plane-parallel ($\mathbf{g} = -GM\mathbf{x}/r^3$)
- Self-gravity – Poisson equation
  - Multipole
    - Nearly spherically symmetric problems
    - Isolated boundary conditions
    - 1D spherical, 2D cylindrical, 3D Cartesian mesh geometries
  - Multigrid
    - Arbitrary source fields
    - Periodic or isolated boundary conditions (James algorithm)
    - 1D/2D/3D Cartesian mesh geometry

Multipole algorithm

Solve Poisson's equation in integral form:

$$\phi(\mathbf{x}) = -G \iiint d^3x' \frac{\rho(\mathbf{x}')}{|\mathbf{x}-\mathbf{x}'|} \qquad -\frac{G}{|\mathbf{x}-\mathbf{x}'|} = \text{Green's function}$$

Note that

$$\frac{1}{|\mathbf{x}-\mathbf{x}'|} = 4\pi \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \frac{1}{2l+1} \frac{r_<^l}{r_>^{l+1}} Y_{lm}^*(\theta',\varphi') Y_{lm}(\theta,\varphi)$$

where the Y_{lm} are spherical harmonic basis functions and

$$r_< \equiv \min\left[|\mathbf{x}|, |\mathbf{x}'|\right]$$

$$r_> \equiv \max\left[|\mathbf{x}|, |\mathbf{x}'|\right]$$

Thus

$$\phi(\mathbf{x}) = -G \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \frac{1}{2l+1} Y_{lm}(\theta,\varphi) \left[r^l \int_{r<r'} d^3x' \frac{\rho(\mathbf{x}')}{r'^{\,l+1}} Y_{lm}^*(\theta',\varphi') + \frac{1}{r^{l+1}} \int_{r>r'} d^3x' \rho(\mathbf{x}') r'^{\,l} Y_{lm}^*(\theta',\varphi') \right]$$

Multipole algorithm

Using definition of Y_{lm} in terms of Legendre polynomials P_{lm}, obtain

$$\phi(\mathbf{x}) = -G\sum_{l=0}^{\infty} P_{l0}(\cos\theta)\left[r^l \mu_{l0}^{eo}(r) + \frac{1}{r^{l+1}}\mu_{l0}^{ei}(r)\right] -$$

$$2G\sum_{l=1}^{\infty}\sum_{m=1}^{l} P_{lm}(\cos\theta)\Big[(r^l \cos m\varphi)\mu_{lm}^{eo}(r) + (r^l \sin m\varphi)\mu_{lm}^{oo}(r) +$$

$$\frac{\cos m\varphi}{r^{l+1}}\mu_{lm}^{ei}(r) + \frac{\sin m\varphi}{r^{l+1}}\mu_{lm}^{oi}(r)\Big]$$

where the even/odd, inner/outer moments of the density are given by

$$\mu_{lm}^{ei}(r) \equiv \frac{(l-m)!}{(l+m)!}\int_{r>r'} d^3x'\, r'^l \rho(\mathbf{x}') P_{lm}(\cos\theta')\cos m\varphi'$$

$$\mu_{lm}^{oi}(r) \equiv \frac{(l-m)!}{(l+m)!}\int_{r>r'} d^3x'\, r'^l \rho(\mathbf{x}') P_{lm}(\cos\theta')\sin m\varphi'$$

$$\mu_{lm}^{eo}(r) \equiv \frac{(l-m)!}{(l+m)!}\int_{r<r'} d^3x'\, r'^{-(l+1)} \rho(\mathbf{x}') P_{lm}(\cos\theta')\cos m\varphi'$$

$$\mu_{lm}^{oo}(r) \equiv \frac{(l-m)!}{(l+m)!}\int_{r<r'} d^3x'\, r'^{-(l+1)} \rho(\mathbf{x}') P_{lm}(\cos\theta')\sin m\varphi'$$

These moments can be evaluated by any appropriate O(N) quadrature method.

Relaxation methods

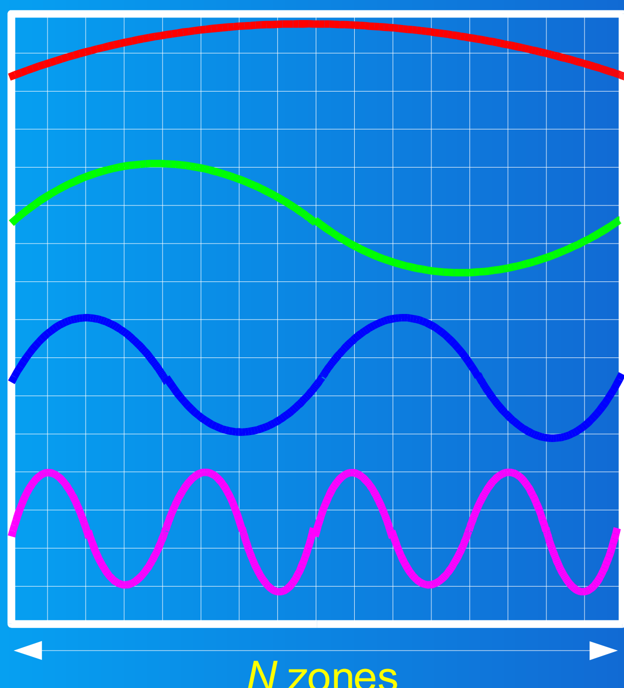
Treat differential form of Poisson equation as steady-state solution to a diffusion problem:

$$\nabla^2 \phi = \rho \qquad \rightarrow \qquad \frac{\partial \phi}{\partial t} = \nabla^2 \phi - \rho$$

"Time coordinate" is really an iteration coordinate. Difference explicitly, rearrange, and apply stability criterion to get Jacobi method:

$$\phi_i^{n+1} = \frac{1}{2}\left[\phi_{i+1}^n + \phi_{i-1}^n\right] - \frac{\Delta x^2}{2}\rho_i$$

Simple... but O(N^2)!



2*N* iterations to communicate across wavelength

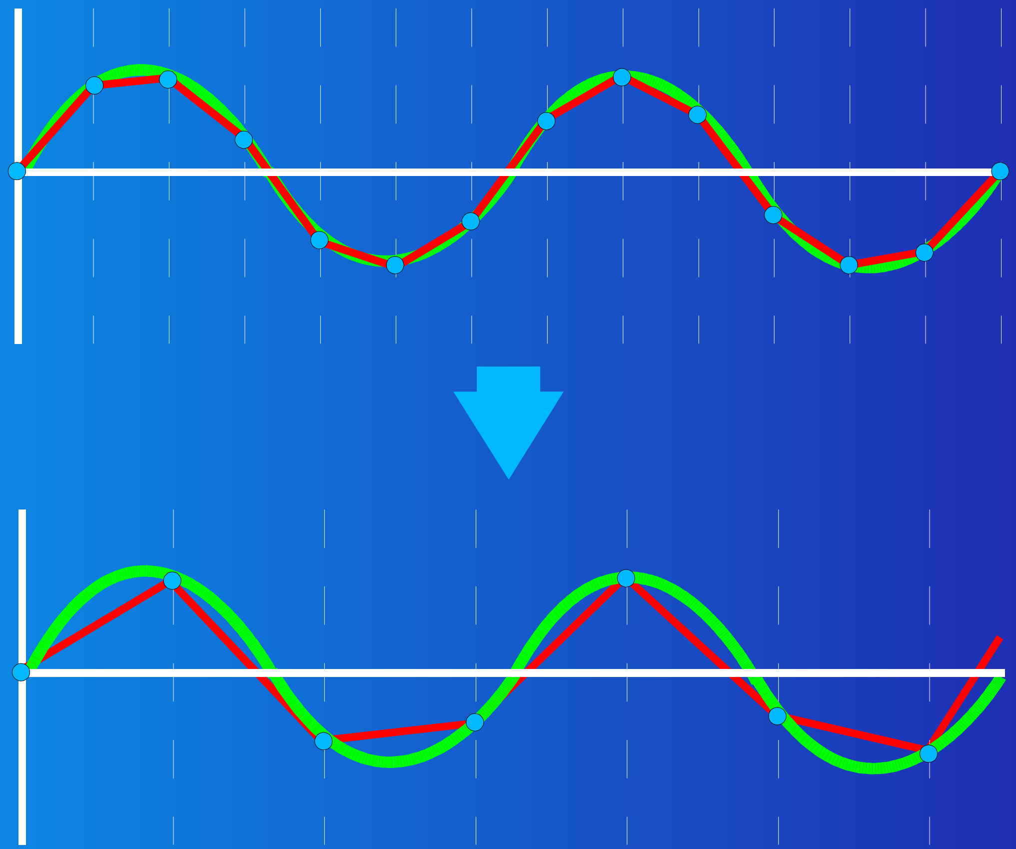
N iterations to communicate across wavelength

N/2 iterations to communicate across wavelength

N/4 iterations to communicate across wavelength

Multigrid – basic idea

- On a coarser mesh, a given error mode appears to be "higher frequency:"



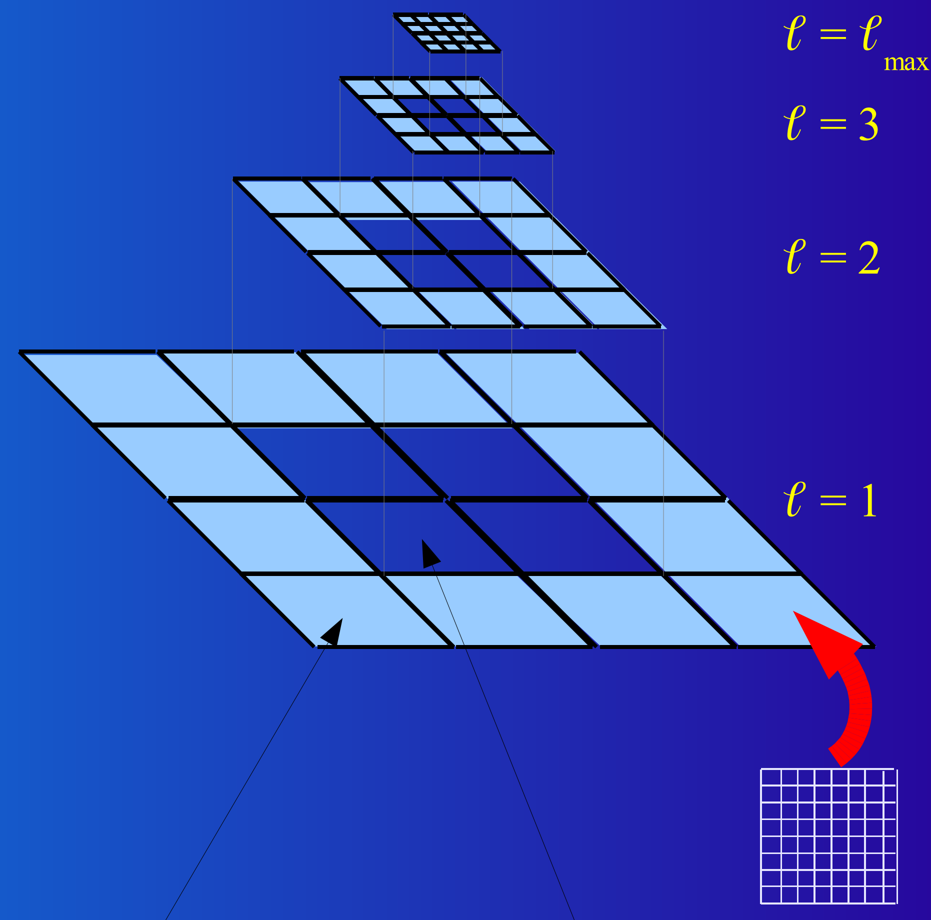
- A single multigrid iteration ("V cycle" in our case) comprises
  - Relaxation of long-wavelength error modes on coarse mesh
  - Relaxation of short-wavelength error modes on fine mesh
- Both wavelengths converge at the same rates – so O(*N* log *N*) performance

AMR multigrid – Poisson equation

Based on Martin & Cartwright (1996)

1. Compute residual of the source equation on all leaf blocks. **Residual operation uses flux conservation at fine-coarse boundaries.**

2. V-cycle:
   a) Zero the correction on level ℓ_{max}.
   b) For $\ell = \ell_{max}$ down to 2:
      i. Copy solution to a temporary variable.
      ii. Zero correction on level ℓ-1.
      iii. Relax correction equation on level ℓ.
      iv. Add correction to the solution on level ℓ.
      v. Compute residual of the correction equation on all blocks (leaf or not) on level ℓ. Restrict this residual to level ℓ-1.
      vi. Compute the residual of the source equation on all leaf blocks of level ℓ-1.
   c) Solve correction equation on level 1. Correct the solution on this level.

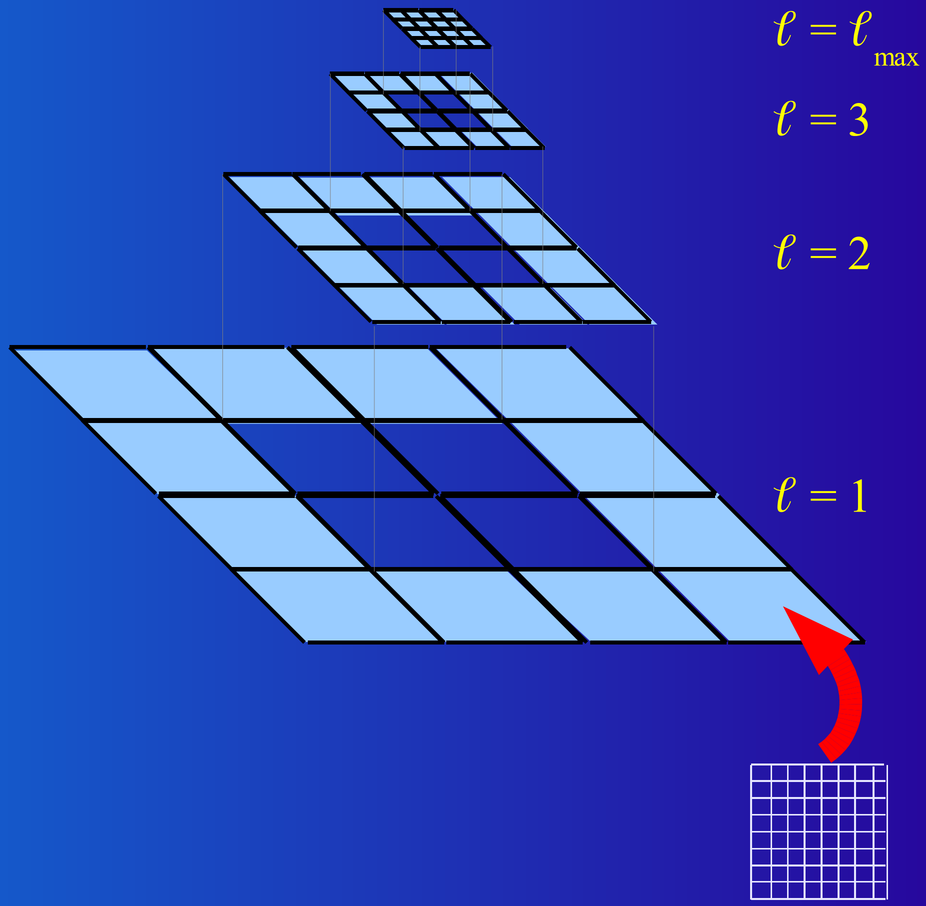


Leaf-node block stores:
- solution
- solution residual

Non-leaf-node block stores:
- correction
- correction residual

AMR multigrid – Poisson equation

2. d) For $\ell = 2$ up to ℓ_{max}:
    i. Prolongate correction from level ℓ-1 and add result to the correction on level ℓ.
    ii. Replace the stored residual on level ℓ with the new residual of the correction equation.
    iii. Zero a second temporary variable on levels ℓ-1 and ℓ.
    iv. Relax this variable against the residual on level ℓ.
    v. Add the result to the correction on level ℓ.
    vi. Copy back into the solution on all leaf blocks on level ℓ.
    vii. Add the correction to the solution on all leaf blocks on level ℓ.



Important to use quadratic ($O(\Delta x^3)$) interpolation when prolongating, or V-cycles will fail to converge (interpolation error will be comparable to differencing error).

Pieces of a particle simulation algorithm

Given particle positions $\mathbf{x}_i^n$ and velocities $\mathbf{v}_i^n$ at time t_n, compute values at time t_{n+1}:

1. Compute acceleration of each particle at t_n: $\mathbf{a}_i^n$
2. Using $\mathbf{a}_i^n$ and possibly acceleration values from previous steps, advance $\mathbf{v}_i^n$:
$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \Delta t\, \mathbf{a}_{\mathrm{eff},i}(\mathbf{a}_i^n, \ldots)$$
3. Using $\mathbf{v}_i^n$ and possibly velocity values from previous steps, advance $\mathbf{x}_i^n$:
$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \Delta t\, \mathbf{v}_{\mathrm{eff},i}(\mathbf{v}_i^n, \ldots)$$

Positions and velocities may be evaluated at different times (e.g., t_n and $t_{n+1/2}$).

Basic components:

- Field computation
  - Direct *N*-body (particle-particle)
  - Particle-mesh
  - Particle-particle-particle mesh (P^3M)
  - Treecodes
- Time integrator

Particle algorithms in FLASH

- Passive particles – particles move with gas
  - Time integrators
    - Euler
    - Predictor-corrector
- Active particles – move independently of gas
  - Time integrators
    - Euler
    - Leapfrog
    - Cosmological leapfrog
  - Long-range forces
    - Gravity (particle-mesh)
- Particle-mesh transfer operators
  - Nearest grid point (NGP)
  - Cloud in cell (CIC)
  - Triangle-shaped cloud (TSC)

Particle-mesh method

- Exploit fast mesh-based Poisson solvers
- First introduced for plasma simulation in the early 1960s
- Procedure:

1. Assign particle masses to mesh using a mass assignment operator $\rightarrow \rho_{ijk}$.
2. Solve $\nabla^2 \phi = 4\pi G \rho$ on mesh.
3. Finite-difference ϕ to get forces on mesh.
4. Interpolate forces to the particle positions using a force interpolation operator.
5. Advance the positions and velocities of the particles in time.
6. Repeat.

Particle-mesh transfer operators

Mass assignment operator:

$$\rho_{ijk} = \frac{m}{\Delta x \, \Delta y \, \Delta z} \sum_{p=1}^{N_p} W(x_i - x_p) \, W(y_j - y_p) \, W(z_k - z_p)$$

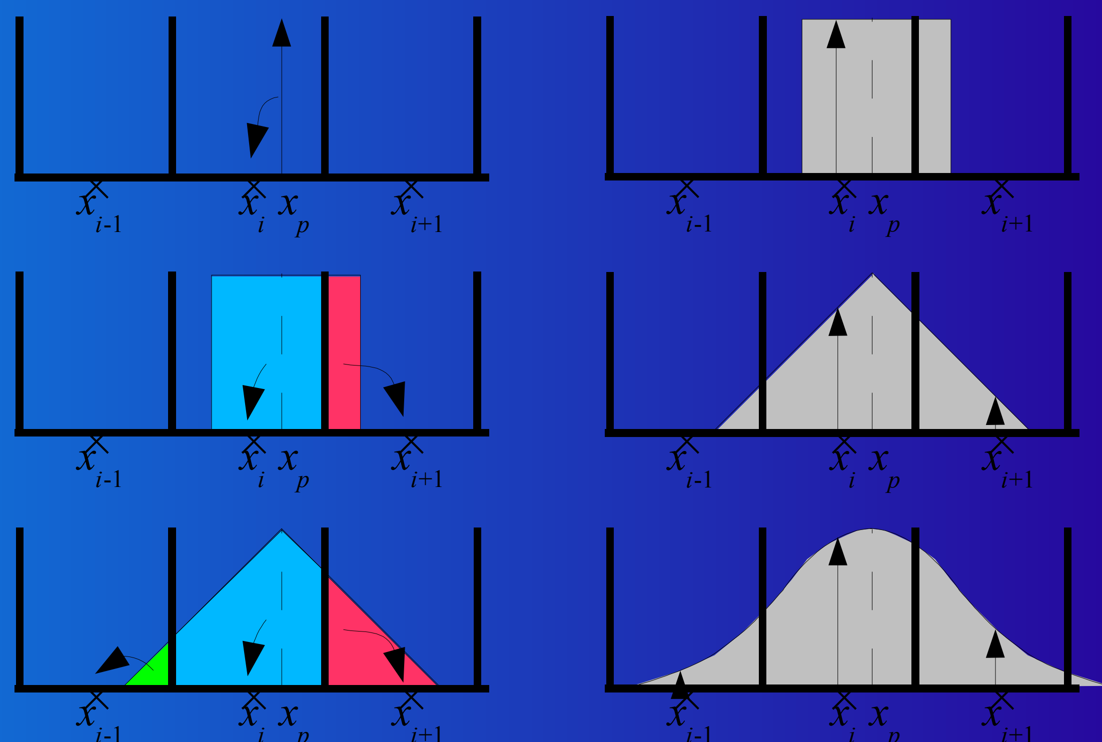
Force interpolation operator:

$$\boldsymbol{F}(x_p) = -m \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} \sum_{k=1}^{N_g} (\nabla \phi)_{ijk} \, W(x_p - x_i) \, W(y_p - y_j) \, W(z_p - z_k)$$

Nearest grid point (NGP)

Cloud in cell (CIC)

Triangle-shaped cloud (TSC)



Force smoothing in particle-mesh

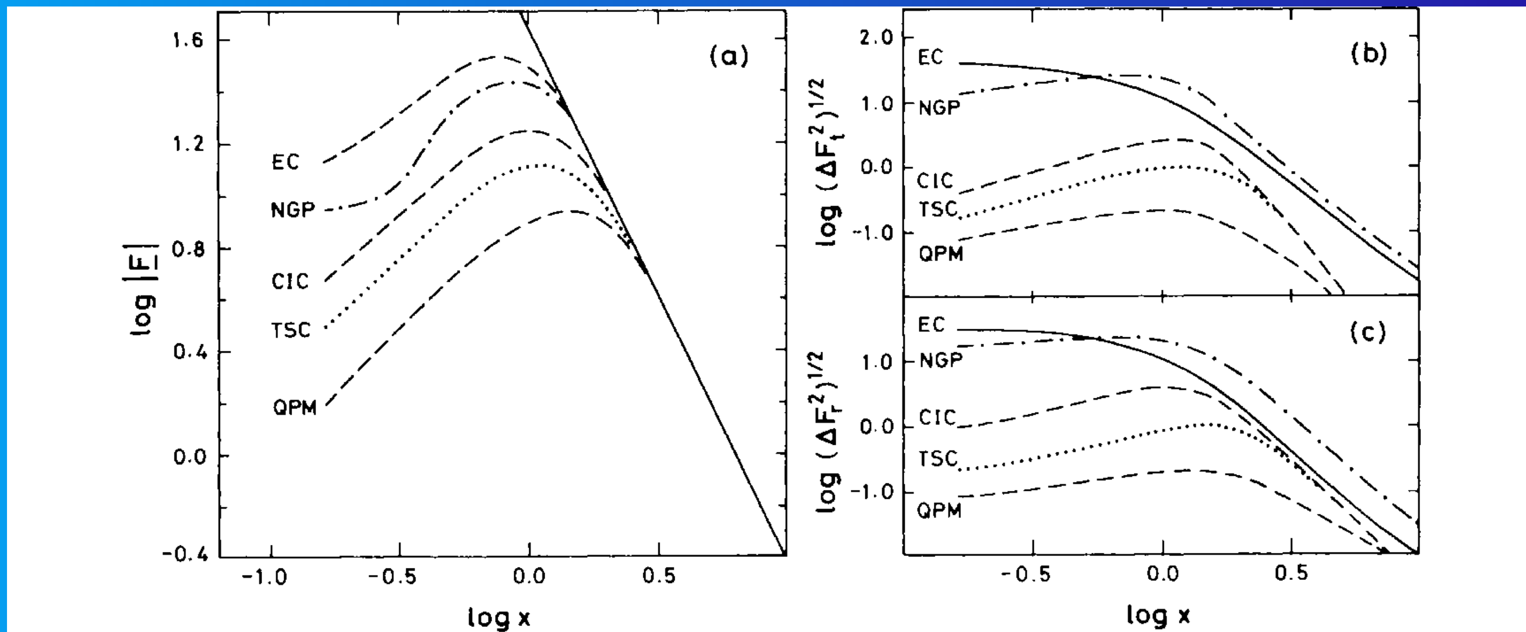
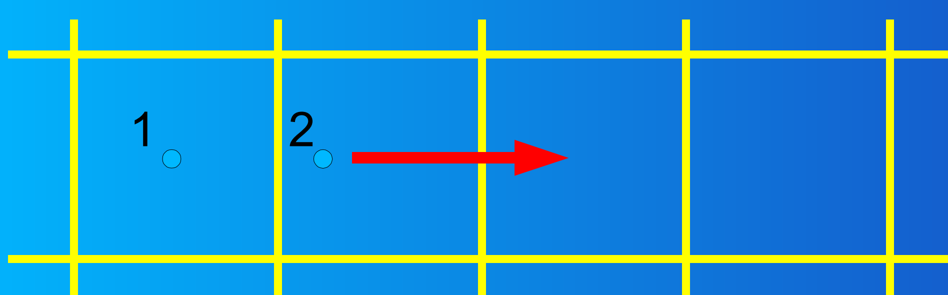
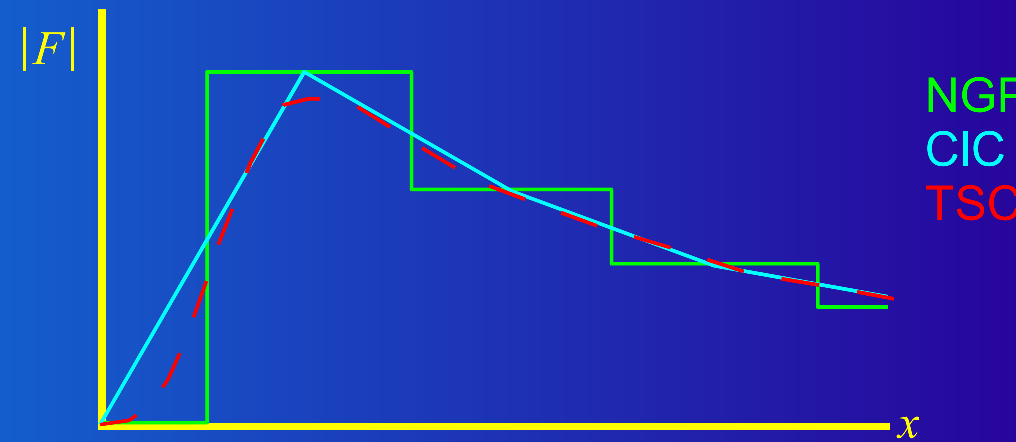


Figure 2 (*a*) The mean interparticle force as a function of separation in mesh spaces for several particle-mesh schemes. The solid line in (*a*) shows the unsoftened force. The other panels show the rms fluctuations in the tangential (*b*) and radial (*c*) directions [reproduced from Efstathiou et al. (1985), with permission].





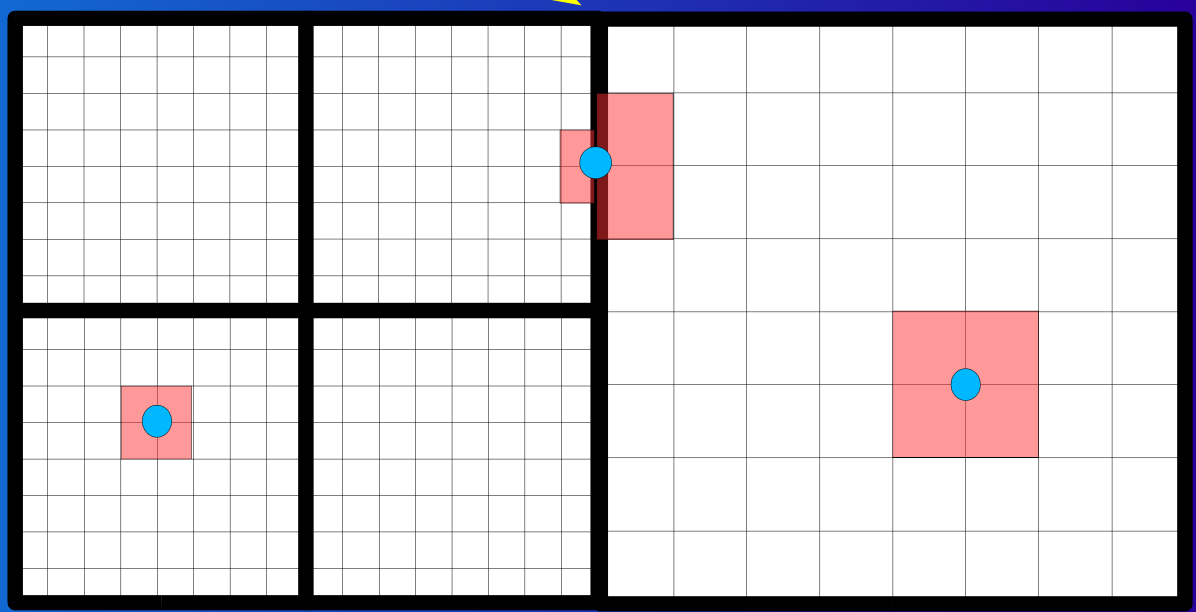
At jumps in refinement

- Multigrid solver
  - Quadratic interpolants
  - “Flux” matching
- Particle interpolation
  - Particle clouds change discontinuously at boundary
  - Nonzero self-forces
  - Few particles: refine on mesh particle density
  - Many particles: mean field dominates

$$\frac{\partial^2 \phi}{\partial x^2} = \rho \quad \rightarrow \quad g_{i+1/2,j} = \frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x}$$

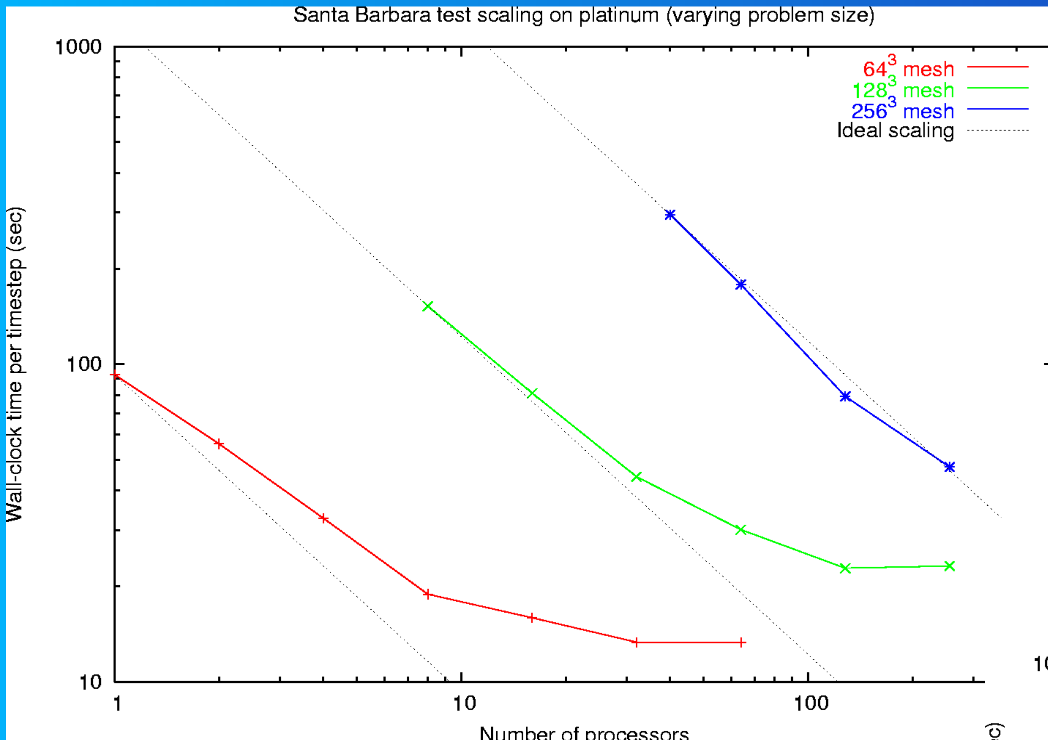
$$g^{\text{coarse}}_{1/2,j} := g^{\text{fine}}_{N+1/2,j}$$

$$R_{i,j} = \rho_{i,j} - \frac{g_{i+1/2,j} - g_{i-1/2,j}}{\Delta x}$$

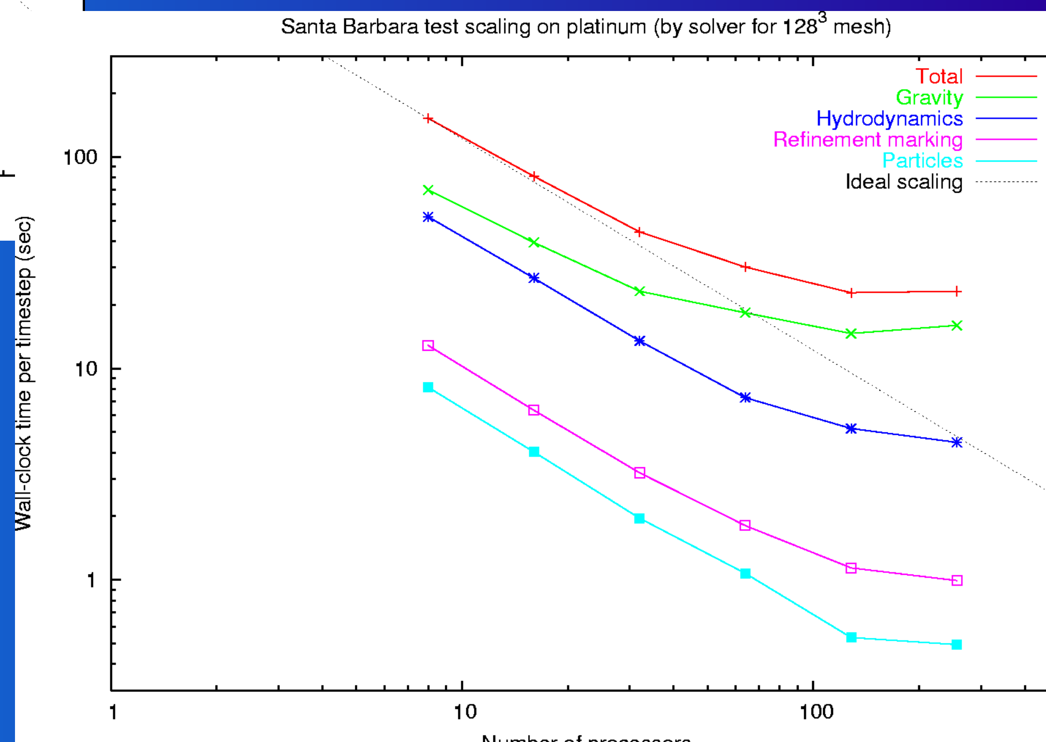


Scaling by solver

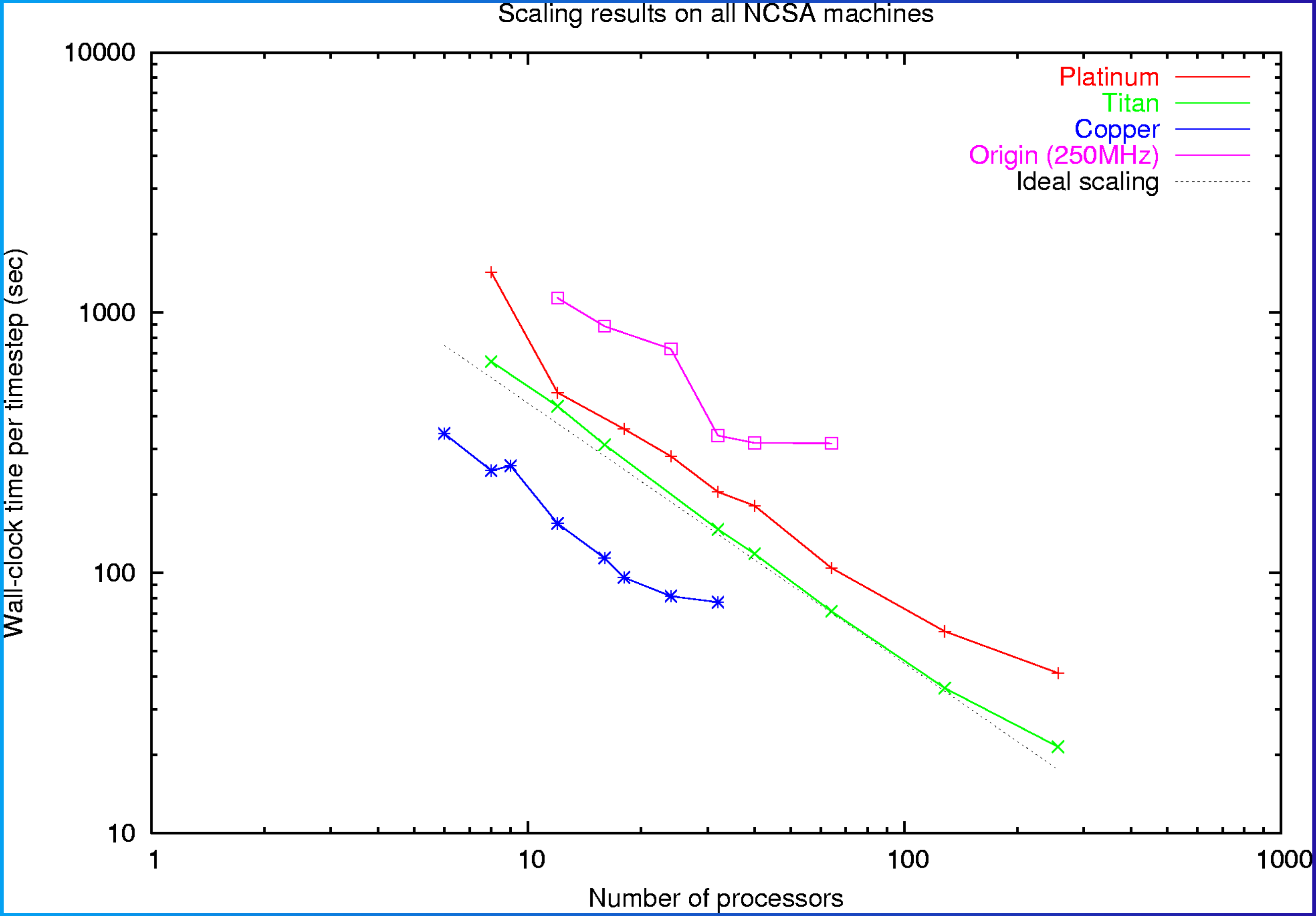
Uniform mesh on NCSA Platinum IA-32 cluster (1 GHz PIII CPUs)



- 16^3 zones per block
- No effort at storage balancing (uniform block weights)

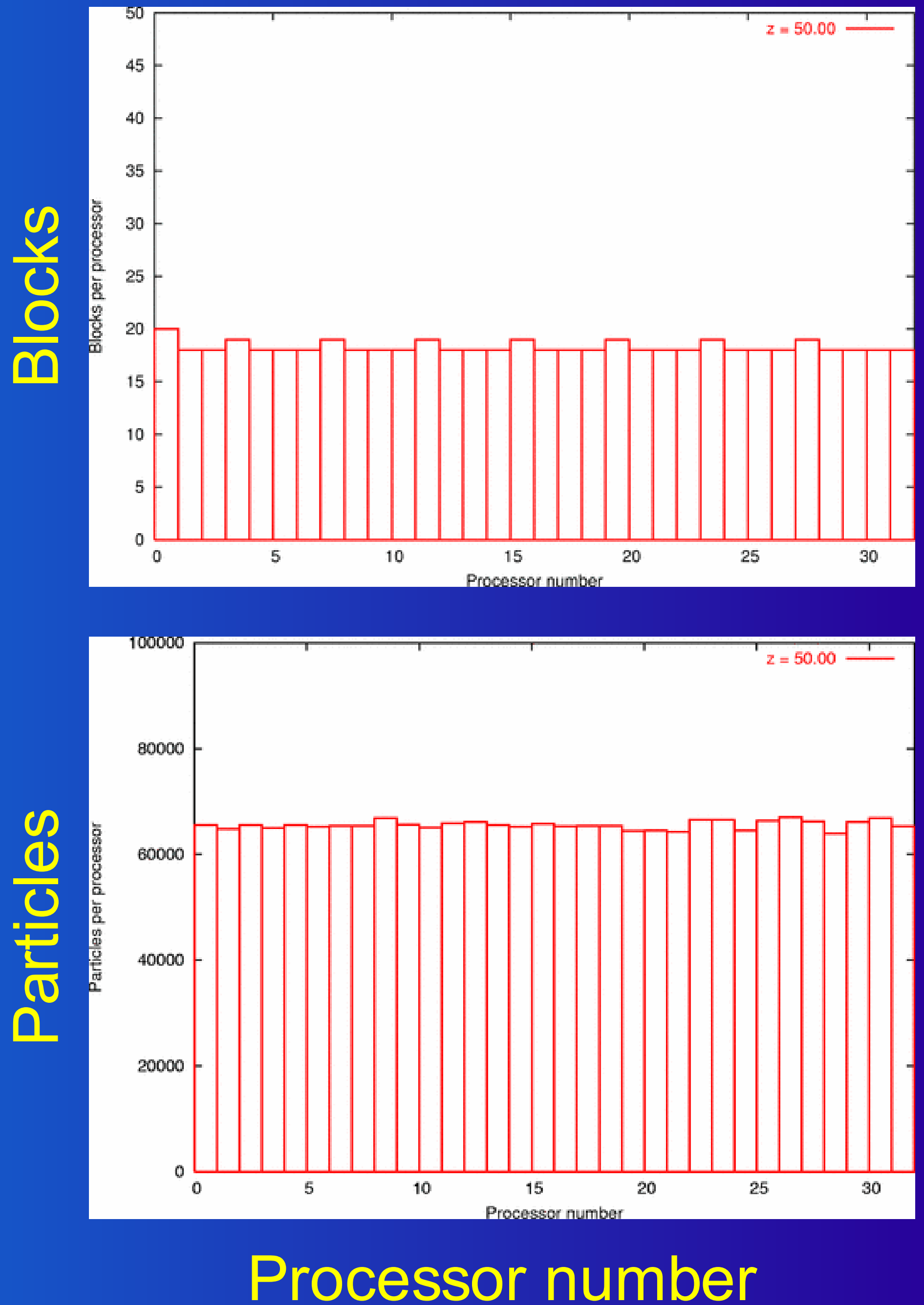


Scaling by platform



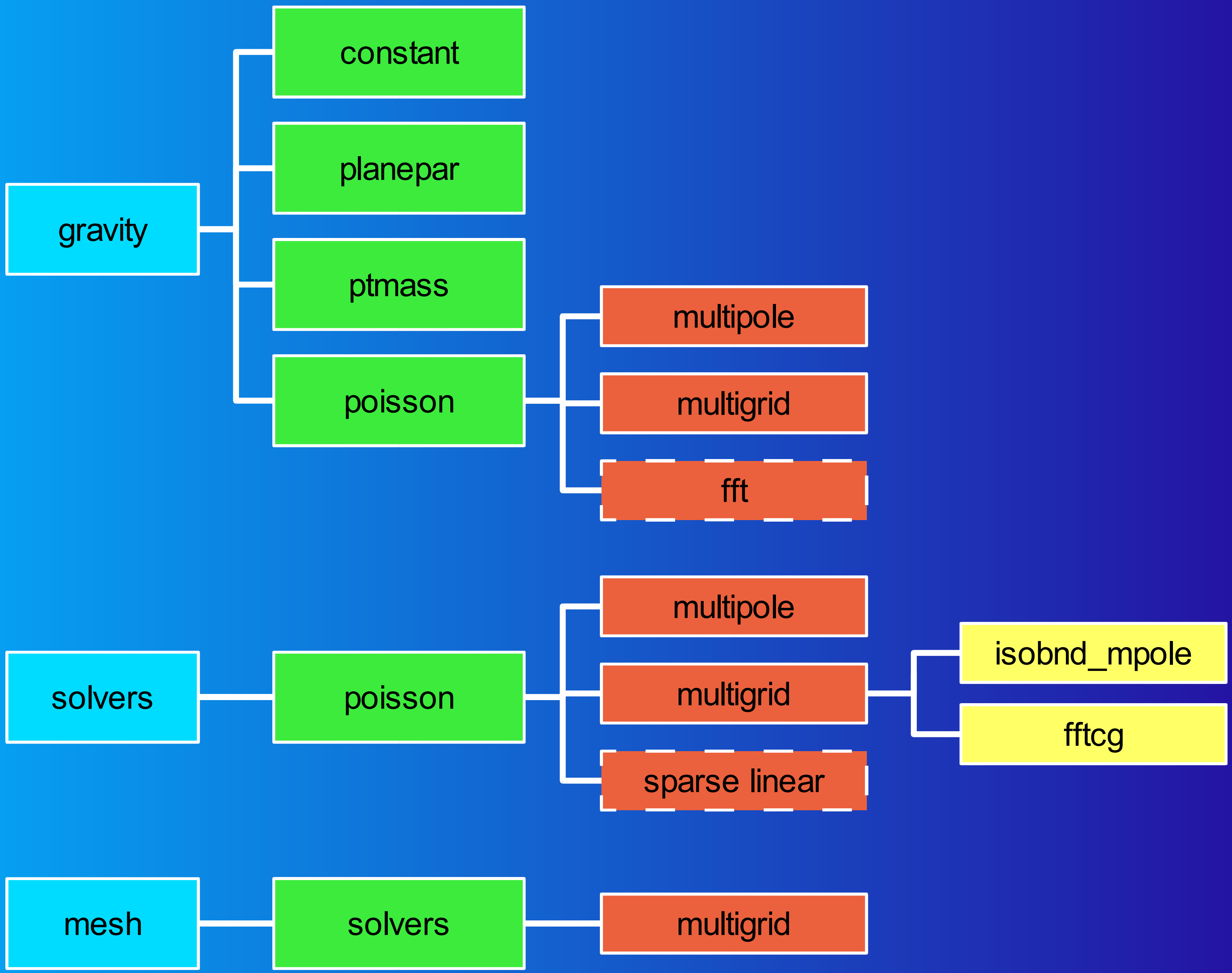
Storage balancing

- The problem
  - Particles start uniform, end clustered
  - Locality: particles stored with blocks
  - Uniform block weights distribute blocks well, but not particles
- A solution
  - Weight blocks by number of particles they contain
  - Shifts storage imbalance to blocks
  - Density-based refinement optimizes both distributions



Usage

FLASH gravity-related modules



Methods supplied by `Gravity` F90 module

- Usage:

  `use Gravity, ONLY:` <*list of gravity methods to import*>

- Types of sub-modules in FLASH
  - Compute acceleration “natively”
  - Compute potential, then difference to get acceleration
- Methods generic to all gravity sub-modules (not called by user modules)
  - `InitGravity()`

    Initialize the gravity module (called by `init_flash()`)
  - `GravityTimestep(dt_grav, dt_minloc, block)`

    Compute any timestep restrictions imposed by gravity module (called by `timestep()`)

Methods supplied by `Gravity` F90 module

- Methods oriented toward computing gravitational acceleration
- No mesh variables generically defined; declared by physics FLASH modules
  - `GravAccelAllBlocks(pot_var, acc_var, component)`

    Compute a component of the acceleration on all blocks.
  - `GravAccelOneBlock(pot_var, acc_var, component, block)`

    Compute a component of the acceleration on a single block.
  - `GravAccelOneLevel(pot_var, acc_var, component, level)`

    Compute a component of the acceleration on all blocks at a single level of refinement.
  - `GravAccelOneRow(j, k, direction, block, pot_var, g, n)`

    Compute a component of the acceleration along a single row of a single block; return as an array (`g`).
  - `GravAccelOneZone(x, y, z, g)`

    Compute all components of the acceleration at a single position in space.

Methods supplied by `Gravity` F90 module

- Methods oriented toward computing gravitational potential
- Generically defined mesh variables: `gpot`, `gpol`
  - `GravPotentialAllBlocks(pot_var)`

    Compute potential on the entire mesh.
  - `GravPotentialOneBlock(pot_var, block)`

    (Planned) return potential on a single block.
  - `GravPotentialOneLevel(pot_var, level)`

    (Planned) return potential on all blocks at a given level of refinement.

Gravity FLASH module configuration

```
#       Parameters:

D         grav_boundary       External boundary condition to use for Poisson solver
D           &                 (applied to all boundaries):  0 = isolated,
D           &                 1 = periodic, 2 = Dirichlet
D         grav_boundary_type  String-valued version of grav_boundary.  Accepts:
D           &                 "isolated", "periodic", "dirichlet".
D           &                 If grav_boundary is set, its value overrides this
D           &                 setting.
D         igrav               Gravity switch:  if /= 0, turn on gravity

DEFAULT multigrid

EXCLUSIVE multigrid fft multipole

PARAMETER grav_boundary_type STRING  "isolated" # string-valued boundary parm
PARAMETER grav_boundary      INTEGER -1         # integer-valued boundary parm
PARAMETER igrav              INTEGER 1          # if /= 0, turn on gravity

#       Self-gravity requires that the density be defined as a variable.
#       This should be ignored by setup if the hydro module (which also
#       defines "dens") is included.

VARIABLE gpot NOADVECT NORENORM NOCONSERVE # grav. potential at current step
VARIABLE gpol NOADVECT NORENORM NOCONSERVE # grav. potential at previous step
VARIABLE dens ADVECT NORENORM CONSERVE     # density
```

Multigrid FLASH module configurations

solvers/poisson/multigrid/isobnd_mpole (isolated boundaries via James method)

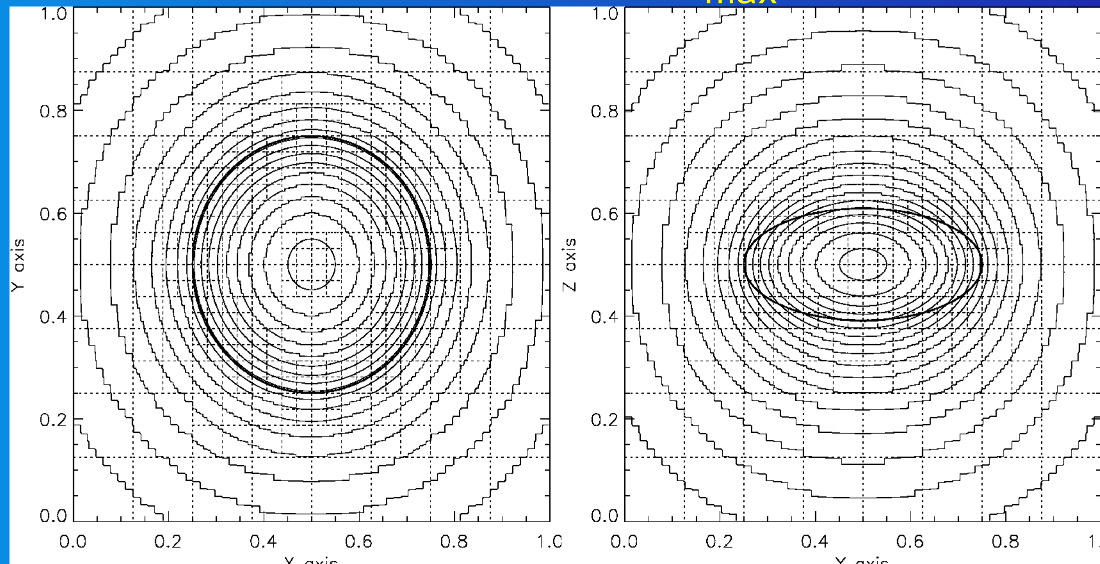
```
D         mpole_lmax    Maximum multipole moment to use

PARAMETER mpole_lmax    INTEGER 0
```

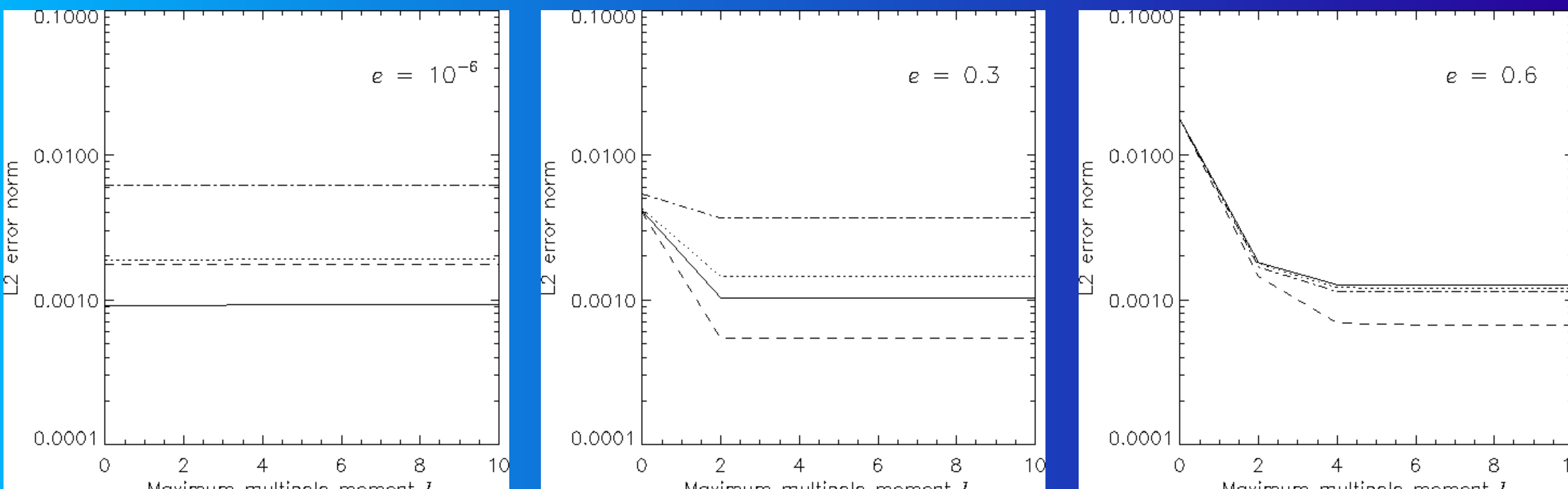
mesh/solvers/multigrid

```
D         mgrid_max_residual_norm Maximum ratio of the residual norm to that of the right-hand side
D         mgrid_max_iter_change   Maximum change in the residual norm from one iteration to the next
D         mgrid_max_vcycles       Maximum number of V-cycles to take
D         mgrid_npresmooth        Number of pre-smoothing iterations to perform on each level
D         mgrid_npostsmooth       Number of post-smoothing iterations to perform on each level
D         mgrid_smooth_tol        Convergence criterion (coarse grid smoother)
D         mgrid_solve_max_iter    Maximum number of iterations for solution (coarse grid smoother)
D         mgrid_print_norm        If .true., print residual norm to stdout after each V-cycle
D         quadrant                In 2d cylindrical coords, assume symmetry about y=0

PARAMETER mgrid_max_residual_norm REAL    1.E-6
PARAMETER mgrid_max_iter_change   REAL    1.E-3
PARAMETER mgrid_max_vcycles       INTEGER 100
PARAMETER mgrid_npresmooth        INTEGER 1
PARAMETER mgrid_npostsmooth       INTEGER 8
PARAMETER mgrid_smooth_tol        REAL    5.E-3
PARAMETER mgrid_solve_max_iter    INTEGER 5000
PARAMETER mgrid_print_norm        BOOLEAN FALSE
PARAMETER quadrant                BOOLEAN FALSE

#       Work variables needed by multigrid solver

VARIABLE mgw1 NOADVECT NORENORM NOCONSERVE
VARIABLE mgw2 NOADVECT NORENORM NOCONSERVE
VARIABLE mgw3 NOADVECT NORENORM NOCONSERVE
VARIABLE mgw4 NOADVECT NORENORM NOCONSERVE
VARIABLE mgw5 NOADVECT NORENORM NOCONSERVE
VARIABLE mgw6 NOADVECT NORENORM NOCONSERVE
VARIABLE mgw7 NOADVECT NORENORM NOCONSERVE
```

Example: Maclaurin spheroid potential

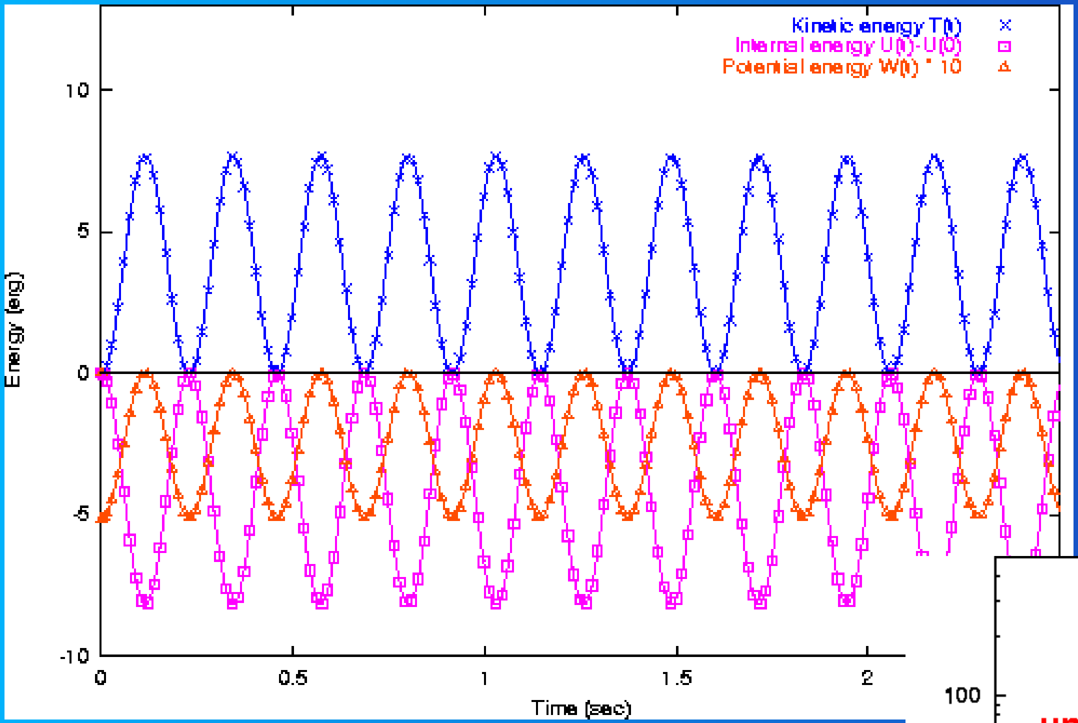
Eccentricity 0.9, $\ell_{max} = 10$

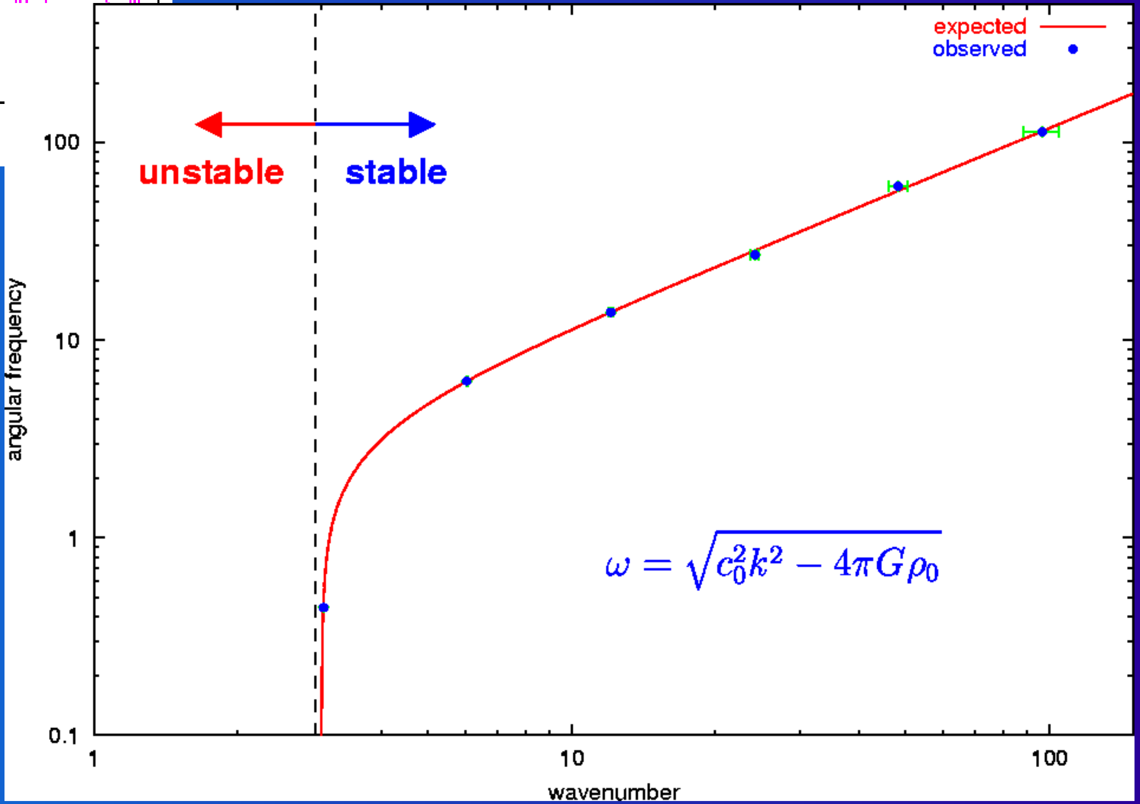




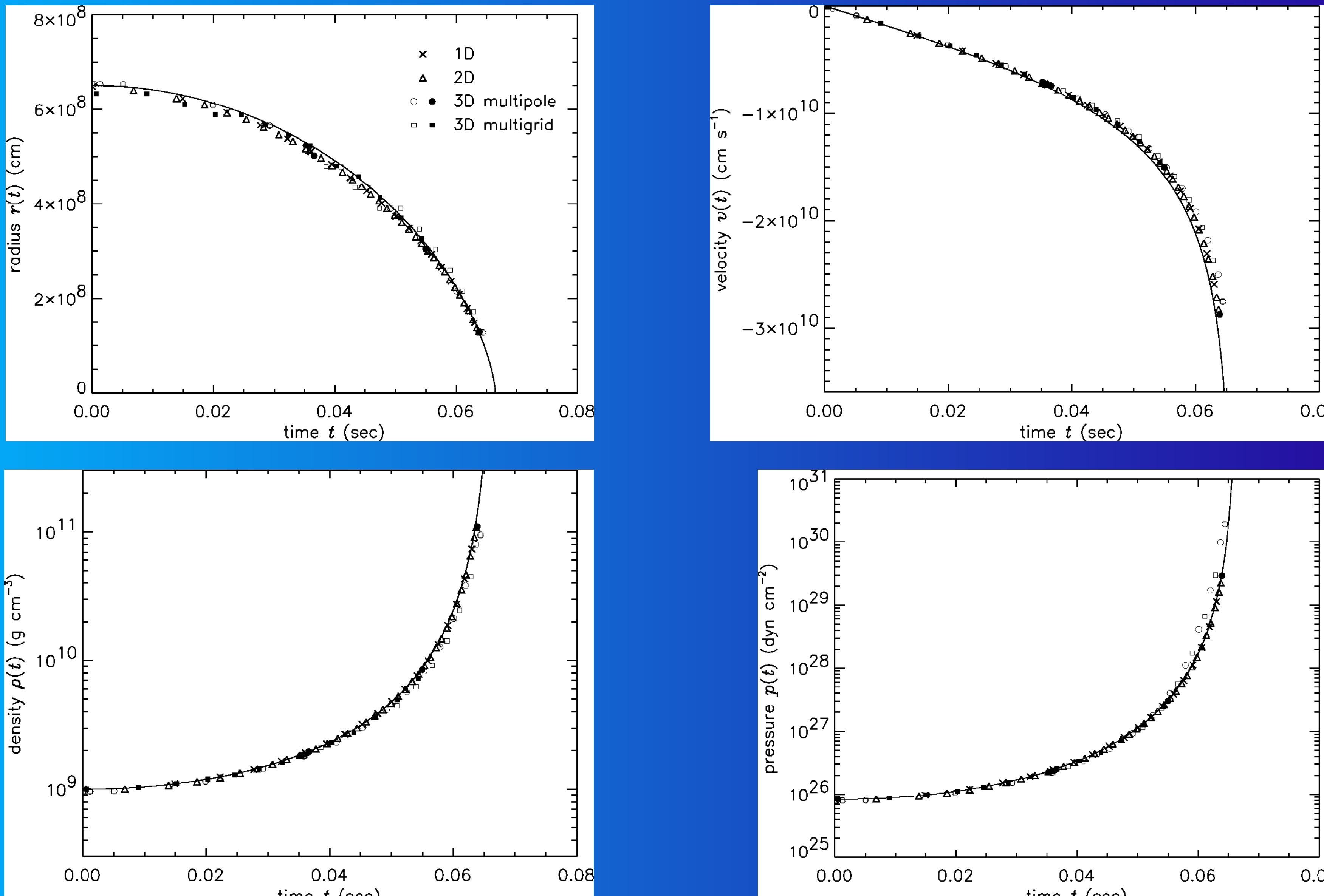
Varying eccentricity and multipole order

Example: Jeans instability



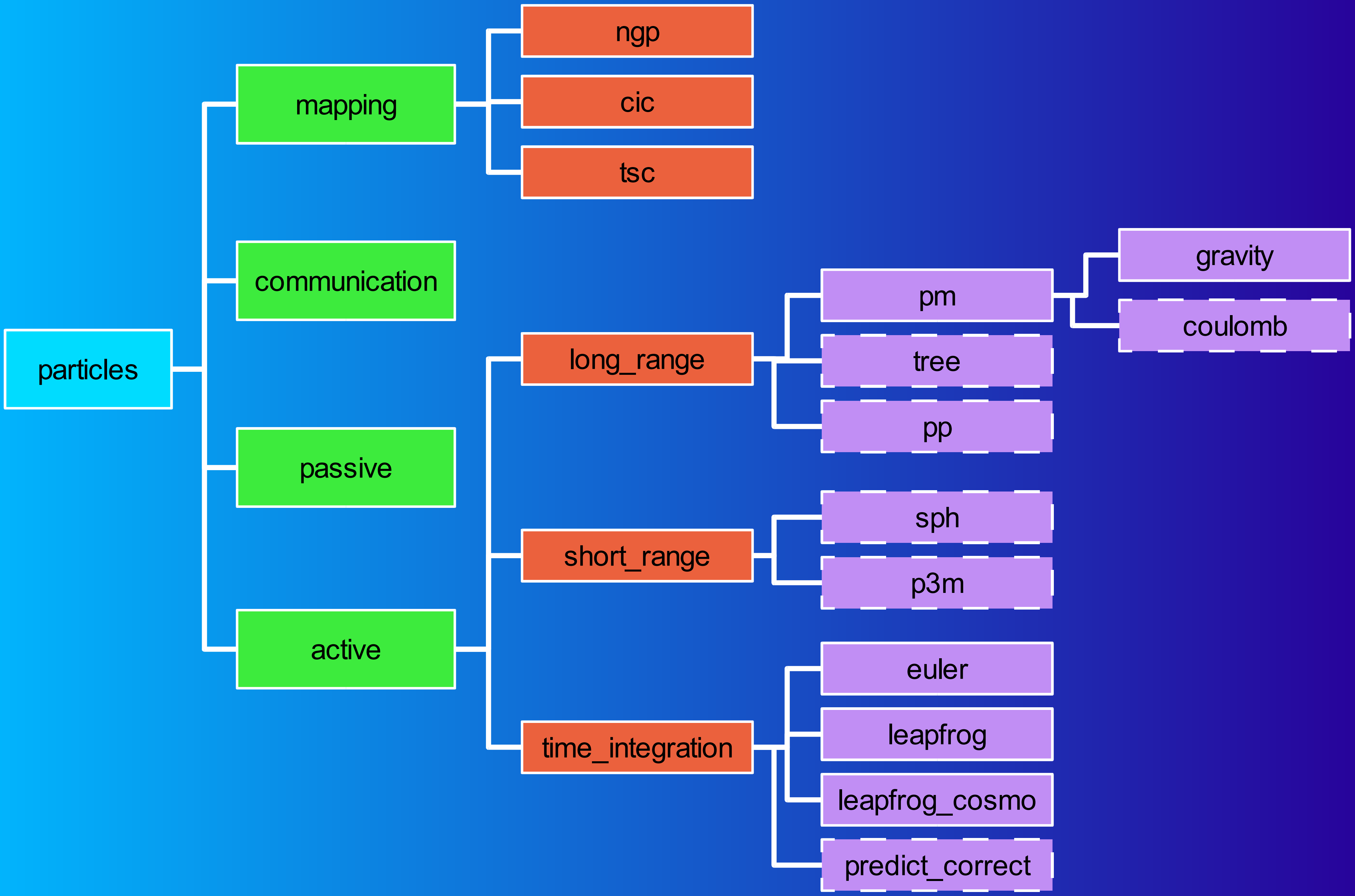


Example: dust cloud collapse



Colgate & White (1966), Mönchmeyer & Müller (1989)

FLASH particle module



Methods supplied by `Particles` F90 module

- Usage:

  `use ParticleModule, ONLY:` <*list of methods to import*>

- Methods:
  - `InitParticles()`

    Initialize the particle module (called by `init_flash()`)
  - `InitParticlePositions(block)`

    Initialize particle positions for a given block.
  - `AdvanceParticles()`

    Time advancement of particle positions.
  - `MapMeshToParticles(p_attrib, mesh_var, zero_mode)`
  - `MapParticlesToMesh(mesh_var, p_attrib, zero_mode)`

    Apply particle-mesh transfer operators.
  - `ReDistributeParticles(mesh_mod_flag)`

    Distribute particles among processors so that they are colocated with the leaf-node blocks that contain them.
  - `ParticleTimestep(dt_part, dt_minloc, block)`

    Compute particle timestep restriction.

Data structures supplied by `Particles` F90 module

- Usage:

  `use ParticleData, ONLY:` <*list of particle data to import*>

- Particles not fully integrated with dBase (yet)
  - Direct access to particle data structures
  - Particle attribute keys managed by dBase
- Basic particle data structure:

```
type particle_type
  integer :: intAttributes(MaxIntProperties)
  real    :: realAttributes(MaxRealProperties)
end type particle_type
type(particle_type) pointer :: particles(:)
```

- Use `RunningParticles` (logical) to test if particles are in code and turned on
- `pden` mesh variable defined for active particles to hold mesh density

Particles FLASH module configuration

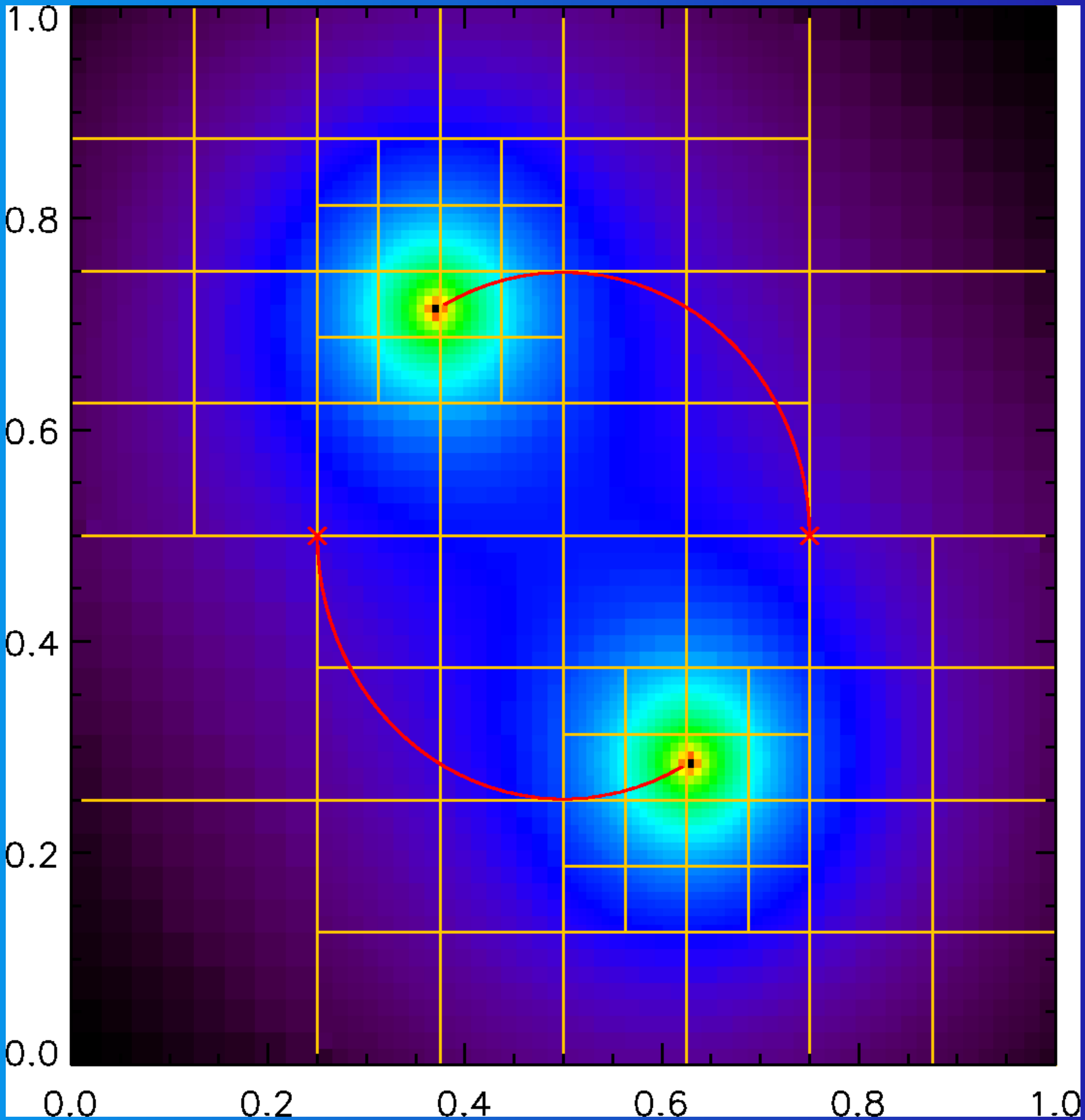
```
#  Configuration file for the particle module

DEFAULT communication

REQUIRES driver
REQUIRES particles/communication
REQUIRES particles/mapping

EXCLUSIVE active passive

#  Parameters:

D   MaxParticlesPerProc      Number of particles to track per processor
D   ipart                    Whether to advance particles or not
D   part_dt_factor           Factor multiplying dx/|v| in setting particle timestep limit

PARAMETER MaxParticlesPerProc  INTEGER 1
PARAMETER ipart                INTEGER 0
PARAMETER part_dt_factor       REAL    0.5

#  Particle properties/attributes

PROPERTY particle_x     REAL
PROPERTY particle_y     REAL
PROPERTY particle_z     REAL
PROPERTY particle_x_vel REAL
PROPERTY particle_y_vel REAL
PROPERTY particle_z_vel REAL
PROPERTY particle_tag   INTEGER
PROPERTY particle_block INTEGER
```

Data structures supplied by `Particles` F90 module

- Example:

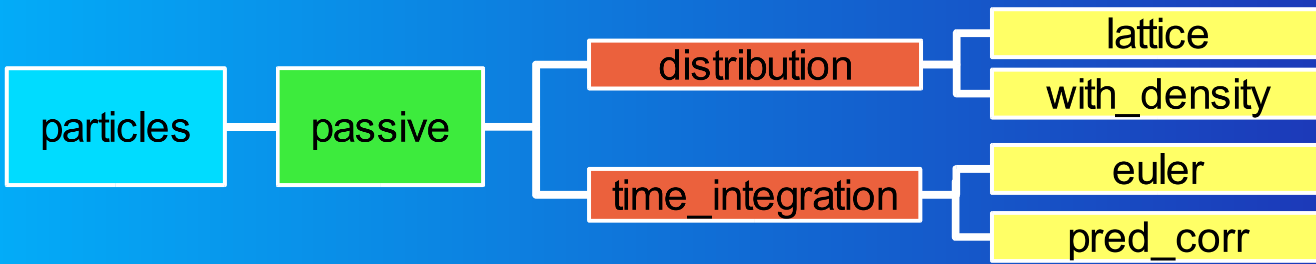
```
use ParticleData, ONLY:  particles, ipx, &
                         is_empty
integer :: ipflavor

ipflavor = dBaseKey("particle_flavor")
if (.not. is_empty(10)) then
  print *, particles(10)%realAttributes(ipx), &
           particles(10)%realAttributes(ipflavor)
endif
```

Example: orbit problem



Example: tracer particles

- FLASH 2.4 modifies passive particles thusly:



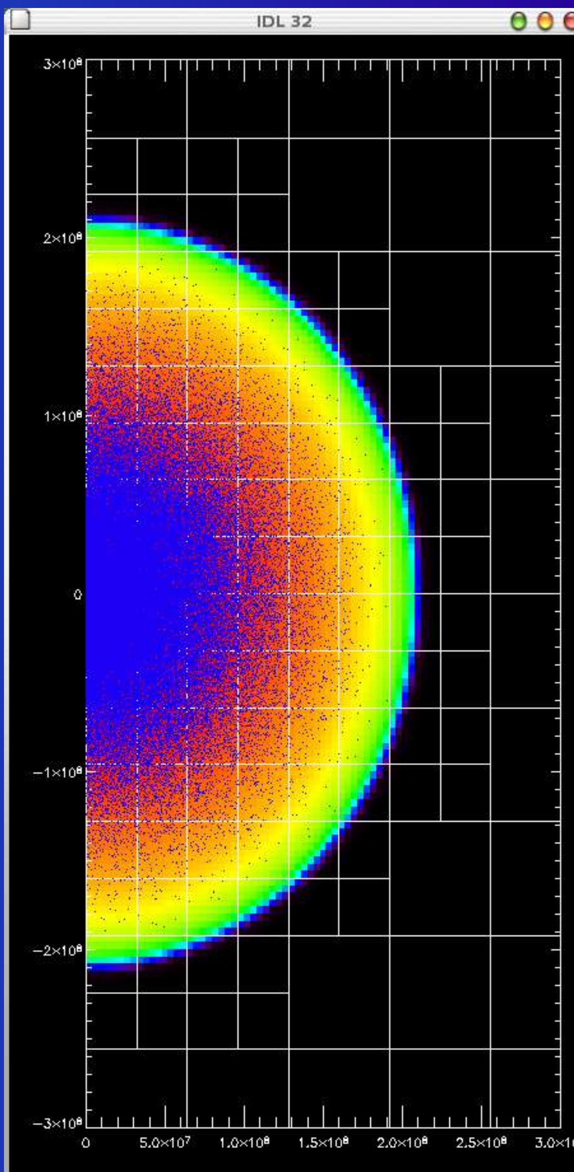
- Use `distribution/with_density` to distribute particles randomly according to the mesh density
- `init_from_scratch()` includes

```
use ParticleModule
```

... set up mesh using repeated init_block() calls ...

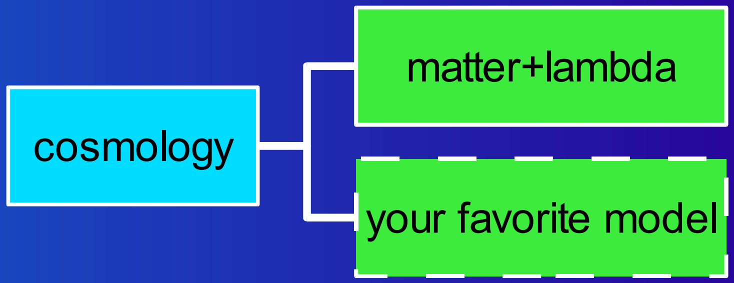
```
do block_no = 1, lnblocks
  if (dbaseNodeType(block_no) == 1) &
    call InitParticlePositions (block_no)
enddo
call ReDistributeParticles (.true.)
```

- Use similar approach for particle clouds with specified density profiles (e.g., isothermal sphere)



Cosmology FLASH module

```
#  Configuration file for the cosmology module.

DEFAULT matter+lambda

#  Parameters:

D  OmegaMatter          Ratio of total mass density to closure density at the
D  &                    present epoch
D  OmegaBaryon          Ratio of baryonic mass density to closure density at the
D  &                    present epoch (must be <= OmegaMatter!)
D  CosmologicalConstant Ratio of the mass density equivalent in the
D  &                    cosmological constant (or dark energy) to the closure
D  &                    density at the present epoch
D  OmegaRadiation       Ratio of total radiation density to closure density at
D  &                    the present epoch
D  HubbleConstant       Value of the Hubble constant (\dot{a}/a) in sec^-1 at
D  &                    the present epoch
D  MaxScaleChange       Maximum permitted fractional change in the scale
D  &                    factor during each timestep

PARAMETER OmegaMatter          REAL 0.3
PARAMETER OmegaBaryon          REAL 0.05
PARAMETER CosmologicalConstant REAL 0.7
PARAMETER OmegaRadiation       REAL 5.E-5
PARAMETER HubbleConstant       REAL 2.1065E-18
PARAMETER MaxScaleChange       REAL 1.E99
```



Methods supplied by `Cosmology` F90 module

- Usage:

  `use Cosmology, ONLY:`   <*list of methods to import*>

- Methods (sample):
  - `InitCosmologicalModel()` (2.4)

    Initialize the cosmology module (called by `init_flash()`)
  - `SolveFriedmannEquation(t, dt)`

    Advance scale factor from time `t` to `t+dt`.
  - `RedshiftHydroQuantities()`

    Apply operator-split redshift terms in the comoving Euler equations.
  - `MassToLength(mass, lambda)`

    Compute comoving diameter of a sphere containing the given mass.
  - `RedshiftToTime(z, t)`

    Compute age of the Universe for a given redshift.
  - `ExpansionTimestep()`

    Compute cosmological timestep restriction ($|\Delta a/a|$ < `MaxScaleChange`).
- Access old and updated scale factor and redshift through dBase

```
redshift = dBasePropertyReal("Redshift")
scale    = dBasePropertyReal("ScaleFactor")
```

Initializing particle problems

- General strategy
  - In `init_from_scratch()`, set up mesh (at least approximately)
  - Initialize particles once blocks are distributed; use `FindLeafBlockForPosn()` to determine which processor "owns" the particle
  - `Call ReDistributeParticles(.true.)` to force particle redistribution
- Some options
  - Read particle positions and velocities from a file
  - Processor 0 randomly chooses positions from user-supplied or mesh-variable distribution and sends them to processors
  - All processors randomly choose positions (rescale # of particles)
  - All processors loop through blocks, select known particle positions

Analyzing particle output

- XFLASH (IDL)
  - Can plot particle positions directly
- From command line in IDL using FIDLR routines (FIDLR2)

```
IDL> read_amr_hdf5, “pan1d_hdf5_chk_0000”, $
IDL>   particles=p, int_prop_names=inames, $
IDL>   real_prop_names=rnames
IDL> print, p[10].realAttributes[where(rnames $
IDL>   eq “particle_x”)]
       3.5360313e+23
```

- From command line in IDL using FIDLR routines (FIDLR3)

```
IDL> read_amr, “pan1d_hdf5_chk_0000”, particles=p
IDL> print, p[10].particle_x
       3.5360313e+23
```

Analyzing particle output

- From F90/C program using FLASH HDF5 reading routines

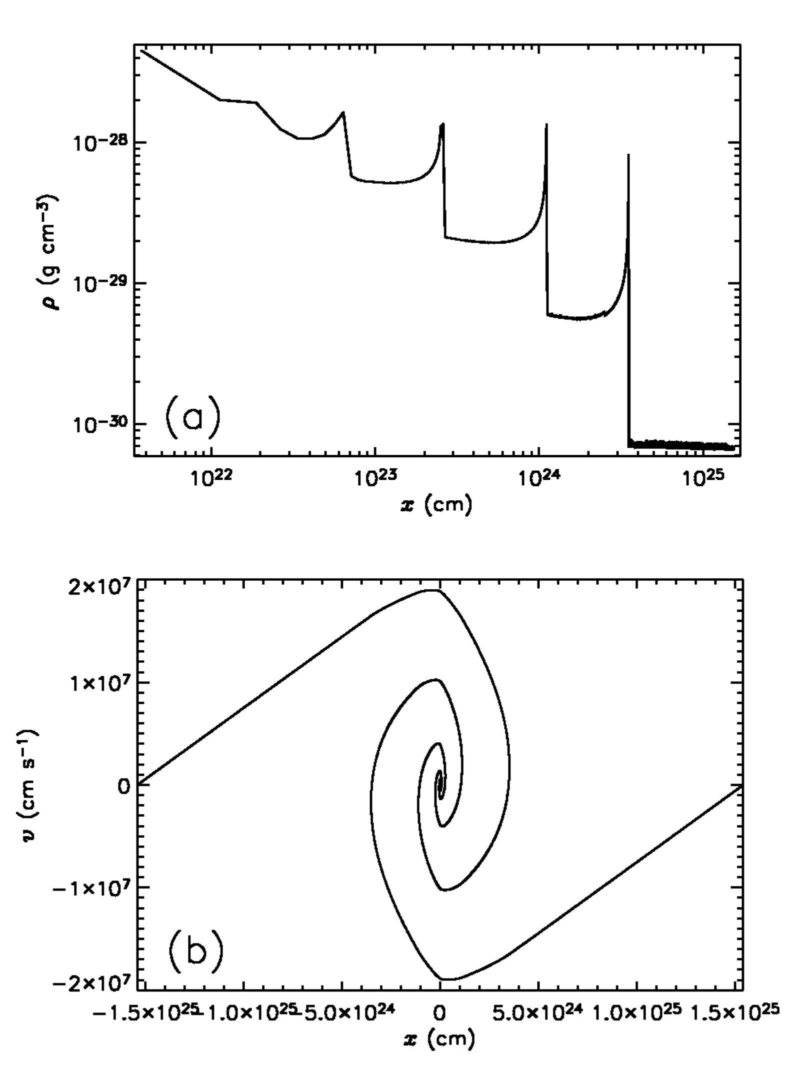
```
integer :: file_id, np
integer, parameter :: MAXINT = 12, &
                      MAXREAL = 12 ! must agree with file
type particle_type
  integer :: int(MAXINT)
  real    :: real(MAXREAL)
end type particle_type
type (particle_type) :: p
character(len=24) :: inames(MAXINT), rnames(MAXREAL)

call h5_open_file_for_read (file_id, “myfile”)
call get_numparticles (file_id, np)
call h5_read_particles (file_id, np, 1, 9, inames, rnames, p)
print *, p
call h5_close_file (file_id)
```

Example: Zel'dovich pancake

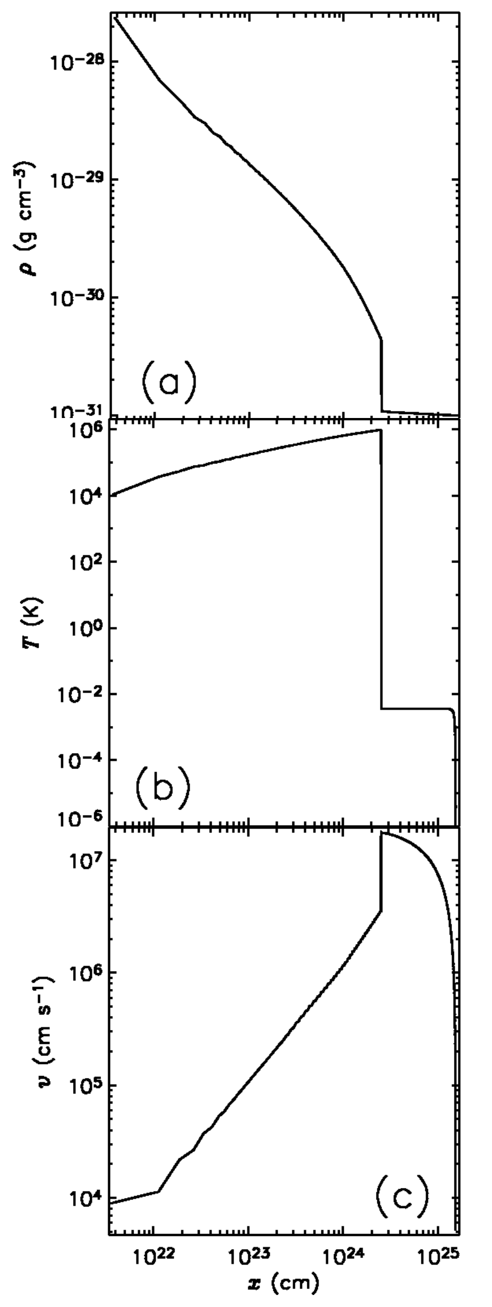
`pancake` problem

z = 0, caustic at z = 5
10 levels of refinement

- Single-mode planar collapse
- Collisionless dark matter forms caustics
- Gas forms strong shocks (Mach > 1000)
- Versions:
  - 2.3: create grid of particles, initialize block-by-block
  - 2.4: initialize all at once, discard particles not owned by us



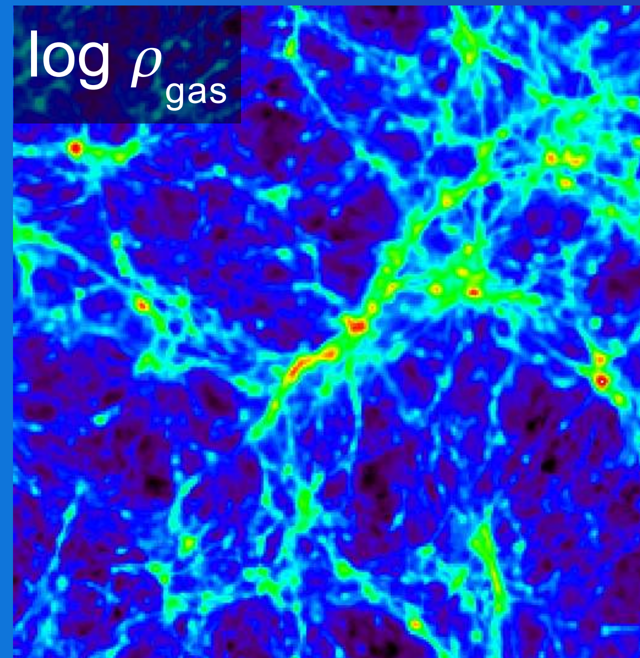
Dark matter

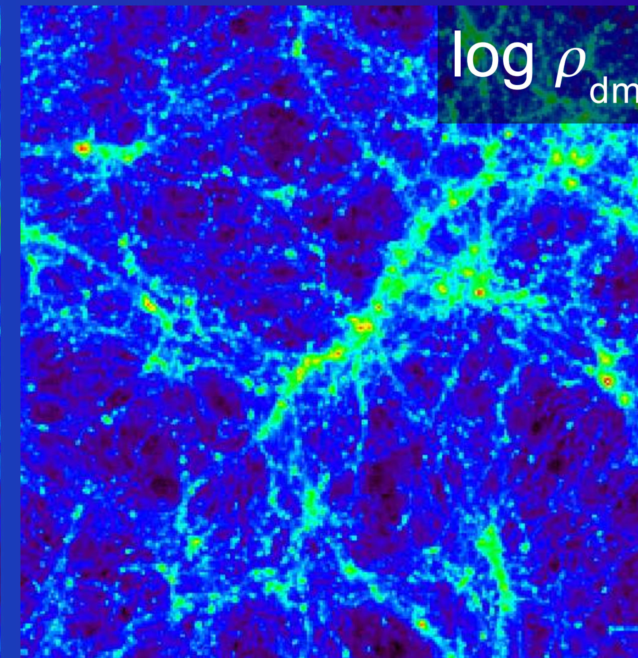


Gas

Example: Santa Barbara cluster

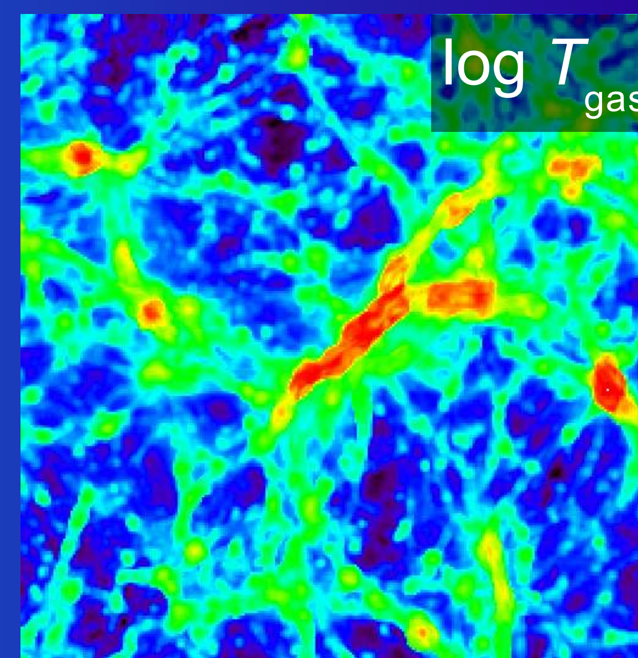
- Standard cold dark matter test problem (Frenk et al. 1999)
- Constrained realization of a 10^{15} $M_{\odot}$ cluster
- 256^3 particles
- $(64\ \text{Mpc})^3$ volume
- Initialization from file
  - Processor #0 reads, sends particles to owning processors





256^3 mesh

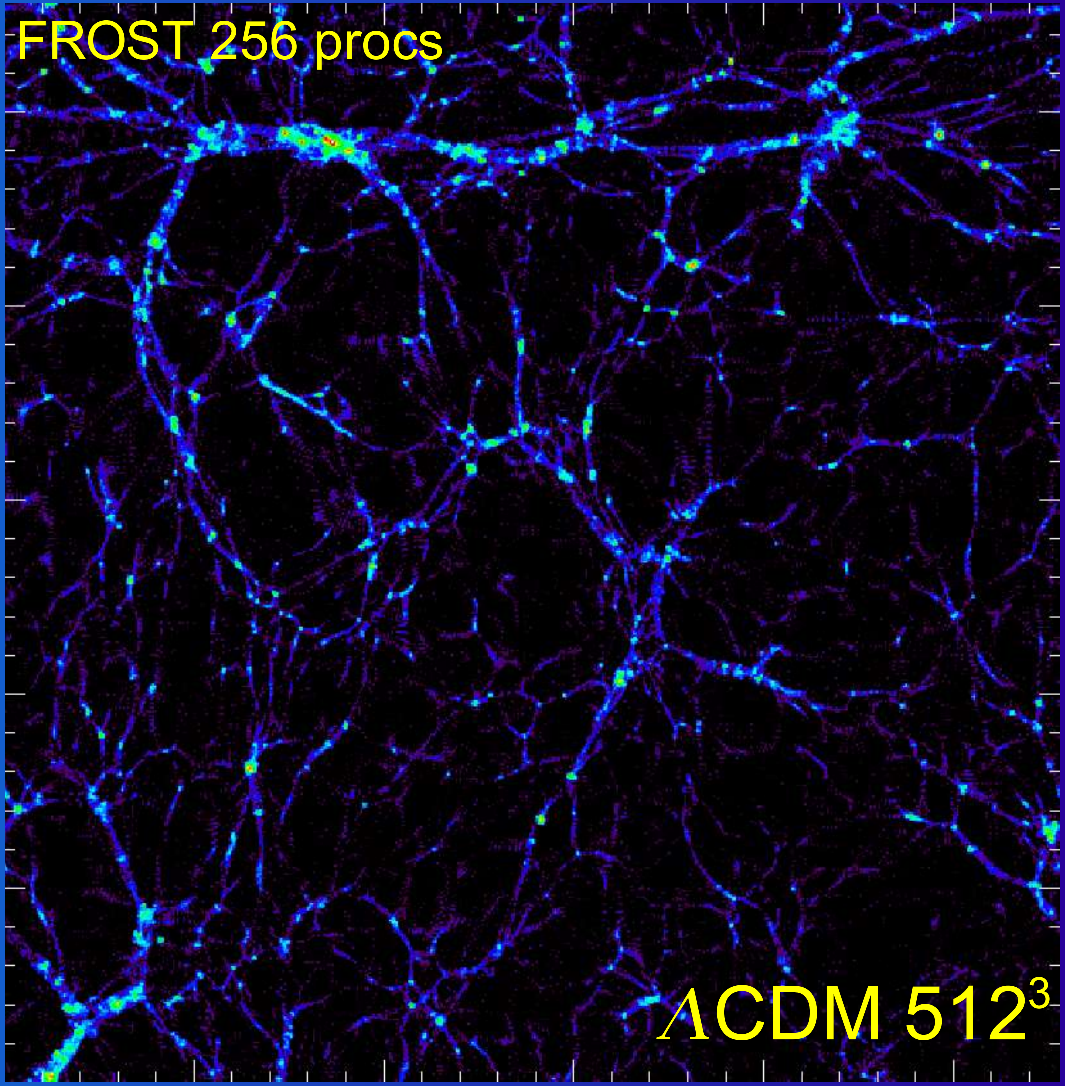
$z = 1.78$



Example: ΛCDM model

Code comparison with LANL PM code (MC^2), GADGET, TreePM, Warren tree code (HOT) (Heitmann et al. 2004)

- Initialization from file
- Two box sizes
  - $L = 64h^{-1}$ Mpc
  - $L = 256h^{-1}$ Mpc
- 1024^3 particles downsampled to 512^3 and 256^3



Summary

- The big picture
  - Applications for gravity and particles
  - Equations to be solved
- Methods
  - Algorithms currently in FLASH
  - Performance
- Usage
  - Organization of the code modules
  - Initializing a particle application
  - Analyzing particle output
  - Examples