



# The Center for Astrophysical Thermonuclear Flashes

---

## Particles

Chris Daley



An Advanced Simulation & Computing (ASC)  
Academic Strategic Alliances Program (ASAP) Center  
at The University of Chicago





# Particle flavors

---

- Passive particles trace hydrodynamic flow in the simulation:
  - Velocities obtained from values on the grid.
  
- Active particles influence the simulation:
  - e.g. forces between particles (N-Body problem).
  
- All particles are stored in the same 2-D array:
  - 1<sup>st</sup> dim: Total number of particle properties (*NPART\_PROPS*) . A single property named *TYPE\_PART\_PROP* indicates particle type.
  - 2<sup>nd</sup> dim: Maximum number of particles that are allowed on a single processor (*pt\_maxPerProc*).



# Particle behaviors

---

- Particle behavior controlled by implementations of:
  - Time advancement
  - Initialization
  - Mapping (Bidirectional for active particles)
  
- Include the FLASH sub-units providing the desired behavior in your Simulation Config file.
  
- Register particle behavior with a particular particle type using PARTICLETYPE keyword in your Simulation Config file.



# PARTICLETYPE keyword

---

- PARTICLETYPE *name* INITMETHOD *initmethod* MAPMETHOD *mapmethod*
  
- The *initmethod* and *mapmethod* strings must correspond to pre-processor definitions from the file `Particles.h`.
  - We use these definitions to select the functions that are called for each particle type (see logic in the wrapper functions `Particles_initPositions` and `Particles_mapFromMesh`).
  
- PARTICLETYPE keyword is not fool-proof!
  - Your responsibility to ensure PARTICLETYPE arguments are consistent with the units being included.
  - Glance over the setup generated files: `Particles_specifyMethods.F90` and `setup_units`.



# Initialization

---

- The wrapper function `Particles_initPositions` calls the specified initialization function for each particle type.
  
- We have initialization functions named `pt_initPositionsLattice` and `pt_initPositionsWithDensity`.
  - These correspond to *initmethod* strings of:
    - “lattice”: Regularly spaced particle distribution.
    - “with\_density”: Density of particles is proportional to the density on the grid.
  
- You can use your own initialization function:
  - Name it `pt_initPositions` and place in simulation directory.
  - Use an *initmethod* string of “custom” for each particle type that should use this distribution.



# Mapping

---

- Converts grid based quantities into similar attributes defined on particles (and vice versa for active particles).
  - `Particles_mapFromMesh` (Mesh → Particles)
  - `Particles_mapToMeshOneBlk` (Particles → Mesh)
  
- FLASH supplies the following mapping schemes:
  - Quadratic: Second-order interpolation.
    - Only available for passive particles.
  - Weighted: A linear weighting from nearby points.
    - Default weighting is Cloud-In-Cell (CIC).
  
- Use *mapmethod* strings of “quadratic” or “weighted”.



# Time advancement

---

- Different time integration schemes for passive and active particles.
  - Only one type of passive and one type of active scheme may be selected in a simulation.
  
- Advancement of particles' position may require particles move to another block (may be on another processor).
  - Movement is handled by Grid/GridParticles subunit.
    - Also handles particle movement that occurs as a result of refinement / derefinement.



# Example 1

---

Add Passive particles:

REQUESTS Particles/ParticlesMain/passive/RungeKutta

PARTICLETYPE passive INITMETHOD lattice MAPMETHOD quadratic

REQUESTS Particles/ParticlesInitialization/Lattice

REQUESTS Particles/ParticlesMapping/Quadratic

REQUIRES Grid/GridParticles





## Example 2

---

Add Active particles with your own custom initialization:

REQUIRES Particles/ParticlesMain/active/LeapfrogCosmo

PARTICLETYPE darkmatter INITMETHOD custom MAPMETHOD weighted

REQUESTS Particles/ParticlesMapping/meshWeighting/CIC

Additional units for  
active particles subject  
to gravitational long  
range force.

REQUIRES Grid/GridParticles/MapToMesh

REQUIRES Particles/ParticlesMapping/meshWeighting/MapToMesh

REQUIRES Particles/ParticlesForces/longRange/gravity/ParticleMesh

REQUESTS physics/Gravity/GravityMain/Poisson/Multigrid



# Particle attributes

---

- Additional properties can be defined for each particle:

PARTICLEPROP *property-name*

- The new particle property may be used to sample the state of mesh variables:

PARTICLEMAP TO *property-name* FROM *VARTYPE variable-name*

(Here, *VARTYPE* can be GRIDVAR, FACEX, FACEY, FACEZ, VARIABLE, MASS\_SCALAR, SPECIES)

- We map from *variable-name* to *property-name* before we write a checkpoint file or a particle file.

- Example: To sample the value of a mass scalar named val1:

MASS\_SCALAR val1

PARTICLEPROP pval1

PARTICLEMAP TO pval1 FROM MASS\_SCALAR val1



# Particle based refinement

---

- Possible to refine the AMR grid according to the number of particles in each block.
  - May be necessary to avoid exceeding *pt\_maxPerProc* in simulations that have significant particle clustering.
  
- This can be used as the sole refinement criterion or it can be used in conjunction with the standard mesh refinement criterion.
  
- Use the following runtime parameters:
  - *refine\_on\_particle\_count = .true. / .false.*
  - *max\_particles\_per\_blk = Value*



# Useful runtime parameters

---

Particle options that can be set in flash.par:

*useParticles*: Logical value that specifies whether to use particles.

*pt\_maxPerProc*: Maximum number of particles that may exist on a single processor. Used to size particles array.

*refine\_on\_particle\_count*: Logical value that specifies whether particle count should be used as a refinement criterion.

*max\_particles\_per\_blk*: Refinement criterion for *refine\_on\_particle\_count*. It is the maximum number of particles that may exist on any block.



# Particles

---

Questions?