Lagrangian Infrastructure & IO

FLASH Tutorial/Workshop
May 30 – June 1, 2012
Anshu Dubey

The University of Chicago
Components of the Framework

Four sub-units within Particles unit
- ParticlesMain – unit scope data, time advancement
- ParticlesInitialization – initializing the unit and particle positions
- ParticlesMapping – to & from the grid
- ParticlesForces – from & to other particles and from & to grid

One sub-unit in the Grid unit
- GridParticles
- Three sub-sub-units under it
  - GridParticlesMove – move the particles data structures when their positions change
  - GridParticlesMapFromMesh – interpolate grid variables from the cell or face center to the particle positions
  - GridParticlesMapToMesh – map the particle attribute to relevant cells in the grid variable
The Control Flow Between Them

- **Driver_initFlash**
  - **Particles Initialization**
  - **GridParticles MapFromMesh**
  - **Particles Mapping**

- **Driver_EvolveFlash**
  - **Particles Main**
  - **GridParticles MapFromMesh**
  - **Particles Mapping**

- **Particles Forces**
  - **Particles Mapping**

- **GridParticles Move**
  - **GridParticles MapToMesh**
  - **Particles Mapping**

* invokes
* follows

The University of Chicago
Passive particles trace and record the history of the flow

Active particles influence the simulation
- Massive (dark matter) or Charged (PIC)

All particles are stored in the same 2-D array:
- 1\textsuperscript{st} dim: Total number of particle properties ($\text{NPART_PROPS}$). A single property named $\text{TYPE_PART_PROP}$ indicates particle type.
- 2\textsuperscript{nd} dim: Maximum number of particles that are allowed on a single processor ($\text{pt\_maxPerProc}$).
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 ADVMETHOD advmethod

The *initmethod*, *mapmethod* and *advmethod* 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, Particles_mapFromMesh and Particles_advance).

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 $\rightarrow$ Particles)
  - Particles_mapToMeshOneBlk (Particles $\rightarrow$ 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”.

The University of Chicago
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.
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 $pt_{\text{maxPerProc}}$ 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:
  - $\text{refine\_on\_particle\_count} = .\text{true.} / .\text{false.}$
  - $\text{max\_particles\_per\_blk} = \text{Value}$
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.
Example 1

Add Passive particles:

REQUESTS Particles/ParticlesMain/passive/RungeKutta
PARTICLETYPEx passive INITMETHOD lattice MAPMETHOD quadratic
ADVMETHOD rungekutta
REQUESTS Particles/ParticlesInitialization/Lattice
REQUESTS Particles/ParticlesMapping/Quadratic
REQUESTS Particles/ParticlesMain/passive/RungeKutta
REQUIRES Grid/GridParticles

FLASH Simulation: Weakly compressible turbulence
Example 2

Add Active particles with your own custom initialization:

REQUIRES Particles/ParticlesMain/active/LeapfrogCosmo
PARTICLETYPEn darkmatter INITMETHOD custom MAPMETHOD leapfrog
REQUESTSES Particles/ParticlesMain/active/massive/Leapfrog

REQUESTS Particles/ParticlesMapping/meshWeighting/CIC
REQUIRES Grid/GridParticles/MapToMesh
REQUIRES Particles/ParticlesMapping/meshWeighting/MapToMesh
REQUIRES Particles/ParticlesForces/longRange/gravity/ParticleMesh
REQUESTS physics/Gravity/GravityMain/Poisson/Multigrid

Additional units for active particles subject to gravitational long range force.
Galaxy Cluster Simulation
External Contribution by Mats Holmström
Models ions as particles and electrons as massless fluid
Works only with uniform grid
Two basic operations
- Deposit charges and currents into the grid
  - Grid_mapParticlesToMesh
- Interpolate fields to particle positions
  - Grid_mapMeshToParticles
Time advancement using predictor-corrector leapfrog
Lagrangian Framework
File Types - Diagnostic Files

- **Log File: flash.log**
  - Generated by the Logfile module
  - Collects events during a run, and often provides more data than stdout/stderr
  - Can also put out individual process logfiles -- good for debugging

- **Dat File: flash.dat**
  - Collection of quantities generated per time step
  - Usually integrated over the physical domain

- **amr.log -- Paramesh only!**
  - Generated by Paramesh in the event of an error

- **Timer summaries: timer_summary_xxxxx**
  - Allows for the collection of individual processor timing data from FLASH’s timers, each processor writes out a file
  - Can be turned off by setting `eachProcWritesSummary` to false
File Types -- Large Files

- **Checkpoint files: basename_filetype_chk_xxxx**
  - Contain everything you need to restart outside of a parfile
  - Large, but can save a lot of time and CPU hours
  - Can be set to “roll” via the rollingCheckpoint parameter

- **Plot Files: basename_filetype_plt_cnt_xxxx**
  - Contains specific Eulerian quantities specified in your parfile
  - Much smaller and faster to output than a checkpoint
  - By default double-sized floating point data is output in single precision

- **Particle files: basename_filetype_part_xxxx**
  - Contains header information, particle metadata and particle data
  - Typically very small and fast to output
Key Flash I/O Feature Overview

- Multiple I/O Modes
  - Serial, Parallel, Direct
- Multiple I/O Libraries supported
  - HDF5 in serial and parallel mode
  - PnetCDF
  - More can be brought in under FLASH’s architecture
- Transparent Restarting
- Arbitrary I/O File Splitting
- Integral Quantities
Serial I/O

- Each processor sends its data to the master who then writes the data to a file.

**Advantages**
- Don’t need a parallel file system
- Simple

**Disadvantages**
- Not scalable
- Not Efficient

The University of Chicago
Parallel I/O: Separate Files

- Each processor writes its own data to a separate file
- **Advantages**
  - Fast!
- **Disadvantages**
  - can quickly accumulate many files
  - hard to manage
  - requires post processing
Parallel I/O: Single-file

- Each processor writes its own data to the same file using MPI-IO mapping
- Advantages
  - single file
  - scalable
- Disadvantages
  - requires MPI-IO mapping or other higher level libraries
Parallel I/O Split File

- Hybridized model: parallel output to multiple files

- Advantages
  - Potentially more scalable than single file
  - Can take advantage of architecture

- Disadvantages
  - Requires MPI-IO mapping or other higher level libraries
  - Still have multiple files to deal with
Each processor writes to a section of a data array. Each must know its offset from the beginning of the array and the number of elements to write.
HDF5

- Library maintained by the HDF group
- Allows for serial and parallel operations
- Primary IO format for FLASH

**Pros:**
- Data is stored with metadata that increases portability
- Very flexible data format
- Handles large volumes of data well
- Most tools for working with FLASH files are written for this format

**Cons:**
- Can be slower than other IO libraries
- Lots of settings, can be confusing
HDF5: Notes on Parallel Mode

- Parallel HDF5 can be run using an independent access pattern or a collective access pattern.

- Collective operations can aggregate reads and writes from multiple processes so that the data can be written in one disk operation.

- This can lead to dramatic increases in speed.

- Collective mode may not play nice with other HDF5 features.
PnetCDF

- Library maintained by Argonne National Laboratory
- Allows for parallel operations, a CDF library can be used for serial tools.
- Every operation is run in collective mode

Pros:
- Very fast if collective operations are enabled, can be faster than HDF5
- Interface to files is simpler than HDF5

Cons:
- Not as flexible
- Most tools for FLASH do not support PnetCDF files
Direct IO

- Each processor performs a binary write to disk.
- Data split up into $n$ files where $n$ is the number of processors.

Pros:
- Always available.
- One of the fastest methods available.

Cons:
- No automated reader
- Files will be non-portable
- Can generate too many files

Warning:
- Method of Last Resort!
- Implementation within FLASH is only an example should this mode be necessary.
Flash Center IO Nightmare…

- Large 32,000 processor run on LLNL BG/L
- Parallel IO libraries not yet available
- Intensive I/O application
  - checkpoint files .7 TB, dumped every 4 hours, 200 dumps
    - used for restarting the run
    - full resolution snapshots of entire grid
  - plotfiles - 20GB each, 700 dumps
    - coarsened by a factor of two averaging
    - single precision
    - subset of grid variables
  - particle files 1400 particle files 470MB each
- 154 TB of disk capacity
- 74 million files!
- Unix tool problems
- 2 Years Later we were still trying to sift though data, sew files together
Integral Quantities

- Individual file output by the master PE
- Collects quantities integrated by volume over the grid
  - Cartesian geometries are supported along with 2D cylindrical
- Frequently overrode in individual simulations for additional functionality
- If modified, the user is responsible for all MPI needed to marshal data
  - Recommended that you use Flash_mpi.h and FLASH_REAL for MPI calls.
- Also a good place for step-by-step statistics for debugging
Questions?