The FLASH Framework: from Giga to Exa-scale

Anshu Dubey
University of Chicago

WOLFHPC
May 31, 2011

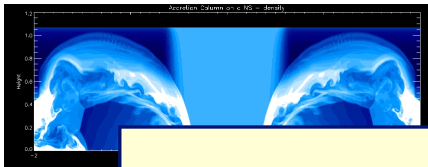
The FLASH Code Contributors

- Current Contributors in the Center:
  - John Bachan, Chris Daley, Milad Fatenejad, Norbert Flocke, Shravan Gopal,Dongwook Lee, Prateeti Mohapatra, Klaus Weide,
- Current External Contributors:
  - Paul Ricker, John Zuhone, Marcos Vanella, Mats Holmstrom
- Past Major Contributors:
  - Katie Antypas, Alan Calder, Jonathan Dursi, Robert Fisher, Murali Ganapathy, Timur Linde, Bronson Messer, Kevin Olson, Tomek Plewa, Lynn Reid, Katherine Riley, Andrew Siegel, Dan Sheeler, Frank Timmes, , Dean Townsley, Natalia Vladimirova, Greg Weirs, Mike Zingale

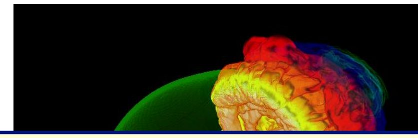
Four sections in the talk

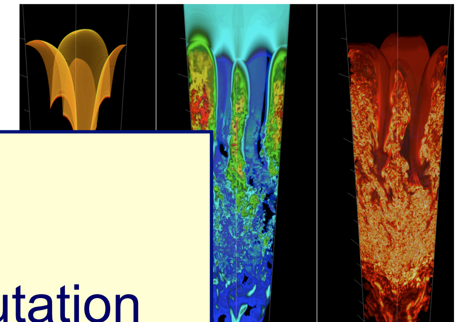
- **Section 1 : General information and evolution of the framework**
- Section 2 : The current code architecture
- Section 3 : History of simulations and the performance challenges at various stages of evolution
- Section 4 : Going to exa-scale

FLASH Capabilities Span a Broad Range…

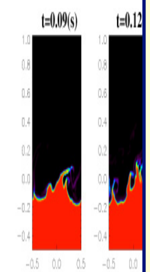


Shortly: Re





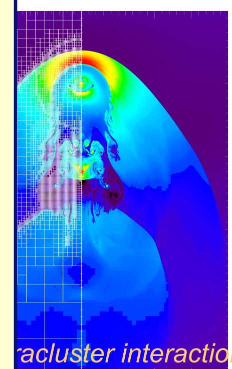
Burning



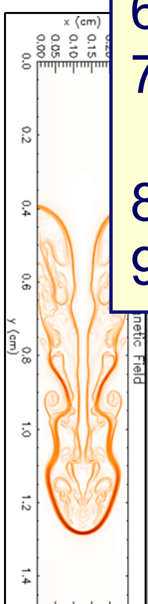
Wave break

The FLASH code

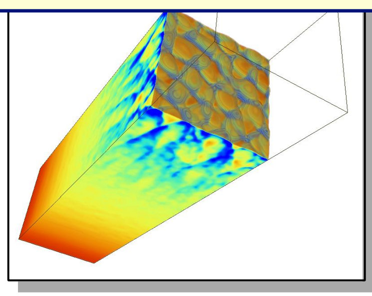
1. Parallel, adaptive-mesh refinement (AMR) code
2. Block structured AMR; a block is the unit of computation
3. Originally designed for compressible reactive flows
4. Can solve a broad range of (astro)physical problems
5. Portable: runs on many massively-parallel systems
6. Scales and performs well
7. Fully modular and extensible: components can be combined to create many different applications
8. Well defined auditing process
9. Extensive user base



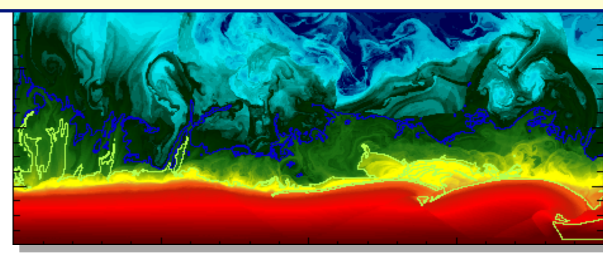
acluster interactions



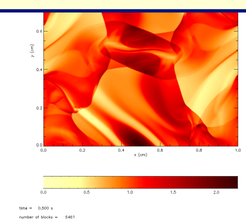
Magnetic Rayleigh-Taylor



Cellular detonation



Helium burning on neutron stars



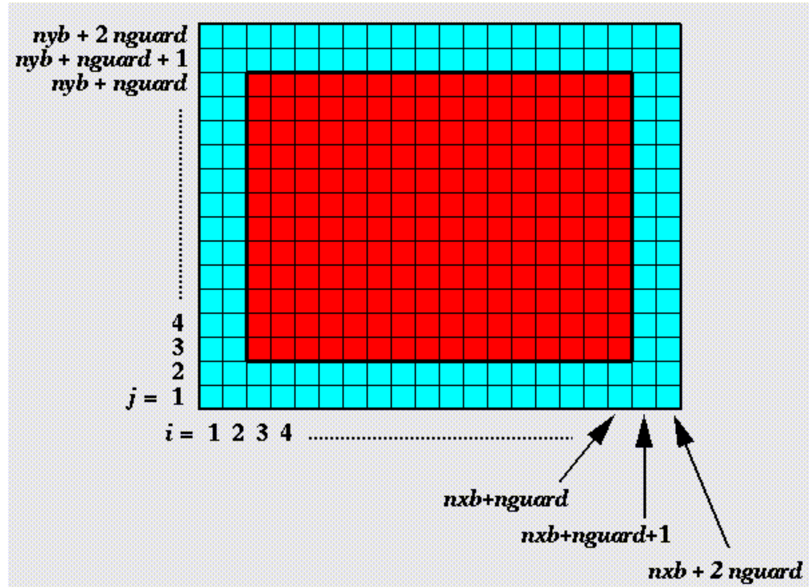
Orzag/Tang MHD vortex

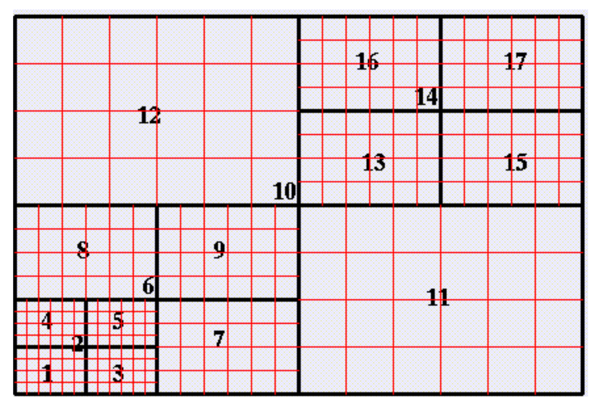
Richtmyer-Meshkov instability

Flash Center for Computational Science
The University of Chicago

Basic Computational Unit, Block

- The grid is composed of blocks
- Cover different fraction of the physical domain.
- In AMR blocks at different levels of refinement have different grid spacing.





FLASH Framework Evolution

Goal : To create robust, reliable, efficient and extensible code, that stands the test of time and users

Challenges

- Many code components started out stand-alone legacy codes
- Individual solvers have different characterisitics
- Complexity of physics dictates lateral interactions between components

History of architecture evolution

- FLASH0 : Smashing of Paramesh (AMR), Prometheus (shock hydrodynamics) and EOS/Burn (nuclear)
- FLASH1 : Introduction of modular architecture & inheritance
  - Configuration layer, alternative implementations of modules
- FLASH2 : Untangle modules
  - Attempt at encapsulation
  - Centralized repository of all data
- FLASH3 : Decentralize data management
  - Encapsulation accomplished
  - Formalization of unit API, and unit architecture
  - Introduction of sub-units
  - Formalization of multiple implementations of a unit or subunit
  - Resolution of lateral data movement issues

Data Management in Current Version

- Defined constants for globally known quantities
- Move from centralized database to ownership by individual units
  - Arbitration on data shared by two or more units
- Definition of scope for groups of data
  - Unit scope data module, one per implementation of the unit
  - Subunit scope data module, one per implementation of the subunit
  - All other data modules follow the general FLASH inheritance
    - The directory in which the module exists, and all of its subdirectories have access to the data modules
- Other units can access data through available accessor functions
- For large scale manipulations of data residing in two or more units, runtime control transfers back and forth between units
  - Avoids lateral transfer of large amounts of data
  - Avoids performance degradation

Example of Unit Design

- Non trivial to design several of the physics units in ways that meet modularity and performance constraints.
- Eos (equation of state) unit is a good example
  - Individual mesh points are independent of each other
  - There are several reusable calculations
  - Other physics units demand great flexibility from it
    - single grid point at a time
    - only the interior cells, or only the ghost cells
    - a row at a time, a column at a time or the entire block at once
    - different grid data structures, and different modes at different times
  - Implementations range from simple ideal gas law to table look up and iterations for degenerate matter and plasma, with widely differing relative contribution in the overall execution time
  - Relative values of overall energy and internal energy play role in accuracy of results
  - Sometimes several derivative quantities are desired as output

EOS interface Design

- Hierarchy in complexity of interfaces
  - For single point calculation scalar input and output
  - For sections of a block or full block vectorized input and output
    - wrappers to vectorize and configure the data
    - returning derivative quantities if desired
- Different levels in the hierarchy give different degrees of control to the client routines
  - Most of the complexity is completely hidden from casual users
  - More sophisticated users can bypass the wrappers for greater control
- Done with elaborate machinery of masks and defined constants

FLASH Physics Capabilities
Hydrodynamics (shocks, MHD, RHD, 2T+rad); Flux-Limited Diffusion; Laser Energy Deposition; Multimaterial EOS & Opacities; Gravity; Nuclear Burning; Material Properties; Source Terms; Cosmology, Particles

Infrastructure

- Abstraction of mesh management
  - Made possible through sub-units
  - Simulations can choose mesh at configuration time
    - Paramesh and Chombo for AMR; Chombo or homegrown UG for uniform mesh
- IO options
  - HDF5 and PnetCDF
  - Direct IO as a last resort
- Hierarchical support for logging progress of a simulation
  - global and local log-files
- Scalable parallel algorithms for solvers
  - Hybridization of multigrid
  - Particles mapping and movement algorithms

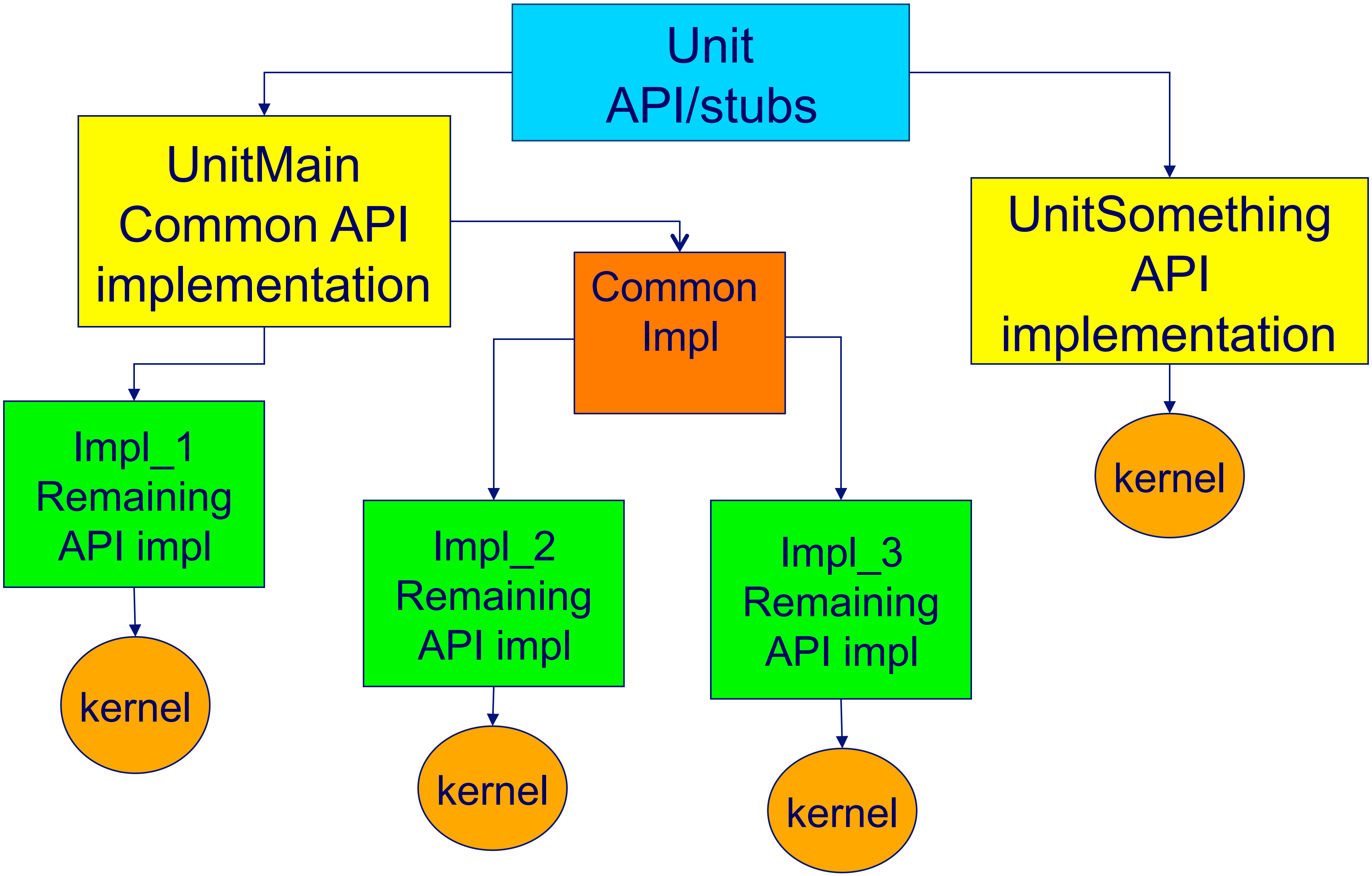
Four sections in the talk

- Section 1 : General information and evolution of the framework
- **Section 2 : The current code architecture**
- Section 3 : History of simulations and the performance challenges at various stages of evolution
- Section 4 : Going to exa-scale

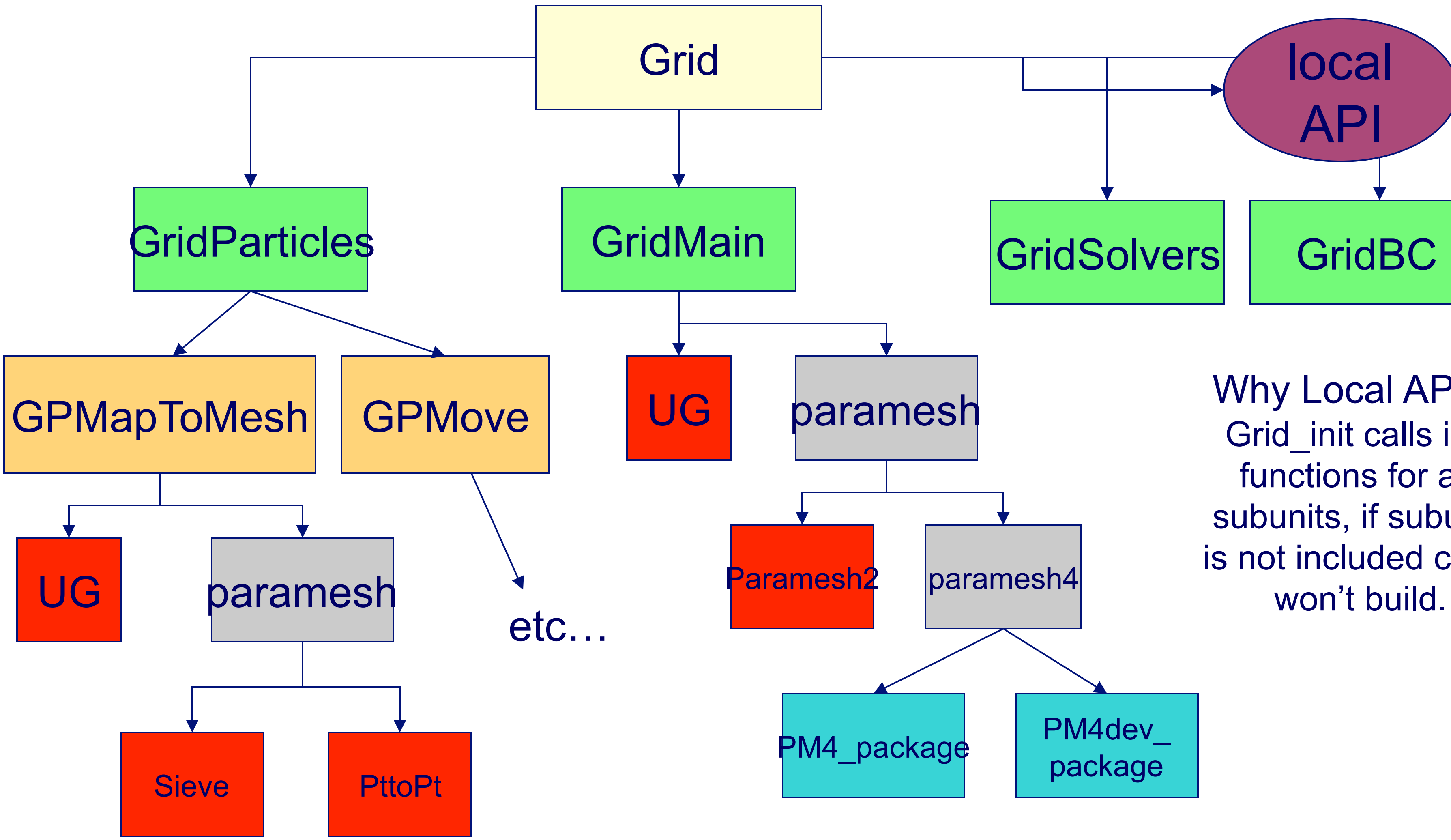
Architecture : Unit

- FLASH basic architecture unit
  - Component of the FLASH code providing a particular functionality
  - Different combinations of units are used for particular problem setups
  - Publishes a public interface (API) for other units' use.
  - Ex: Driver, Grid, Hydro, IO etc
- Interaction between units governed by the Driver
- Not all units are included in all applications
  - Not all subunits of an included unit need to be included in all applications
- An object oriented framework imposed upon F90 code through a combination of configuration setup tool, FLASH specific Config files, unix directory structure, naming convention, inheritance rules and F90 data modules and interfaces

Unit Hierarchy



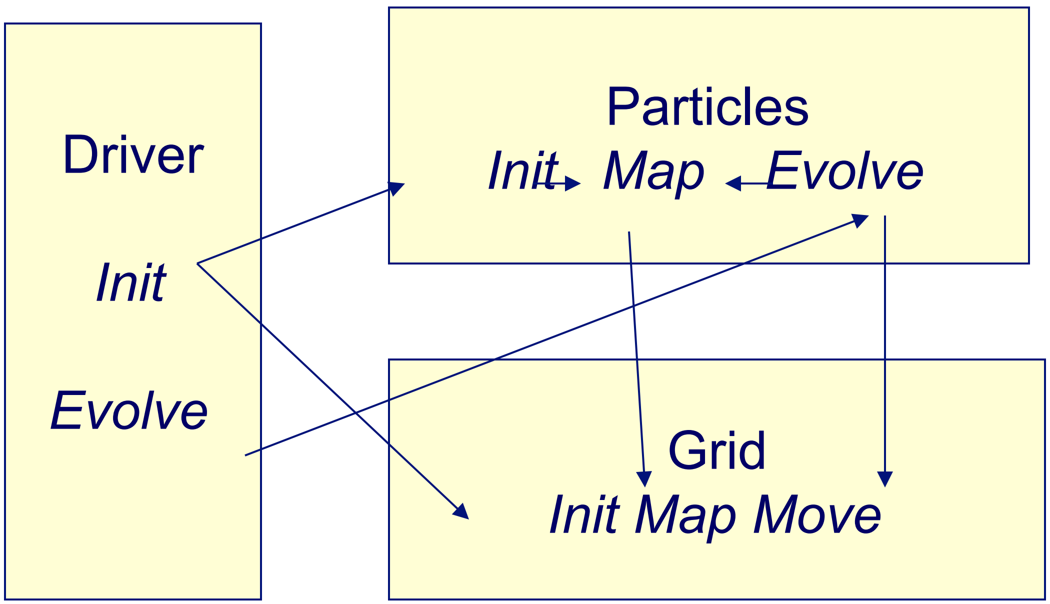
Example of a Unit – Grid (simplified)



Why Local API ?
Grid_init calls init functions for all subunits, if subunit is not included code won't build.

Functional Component in Multiple Units

- Example Particles
  - Position initialization and time integration in Particles unit
  - Data movement in Grid unit
  - Mapping divided between Grid and Particles
- Solve the problem by moving control back and forth between units



Four sections in the talk

- Section 1 : General information and evolution of the framework
- Section 2 : The current code architecture
- **Section 3 : History of simulations and the performance challenges at various stages of evolution**
- Section 4 : Going to exa-scale

Performance Challenges

The Machines

- Cutting edge == less well tested systems software
- Highly specialized hardware
- A new generation every few years
- Parallel I/O always a challenge
- Availability is limited
- Stress testing the code before big runs is extremely challenging (or impossible)

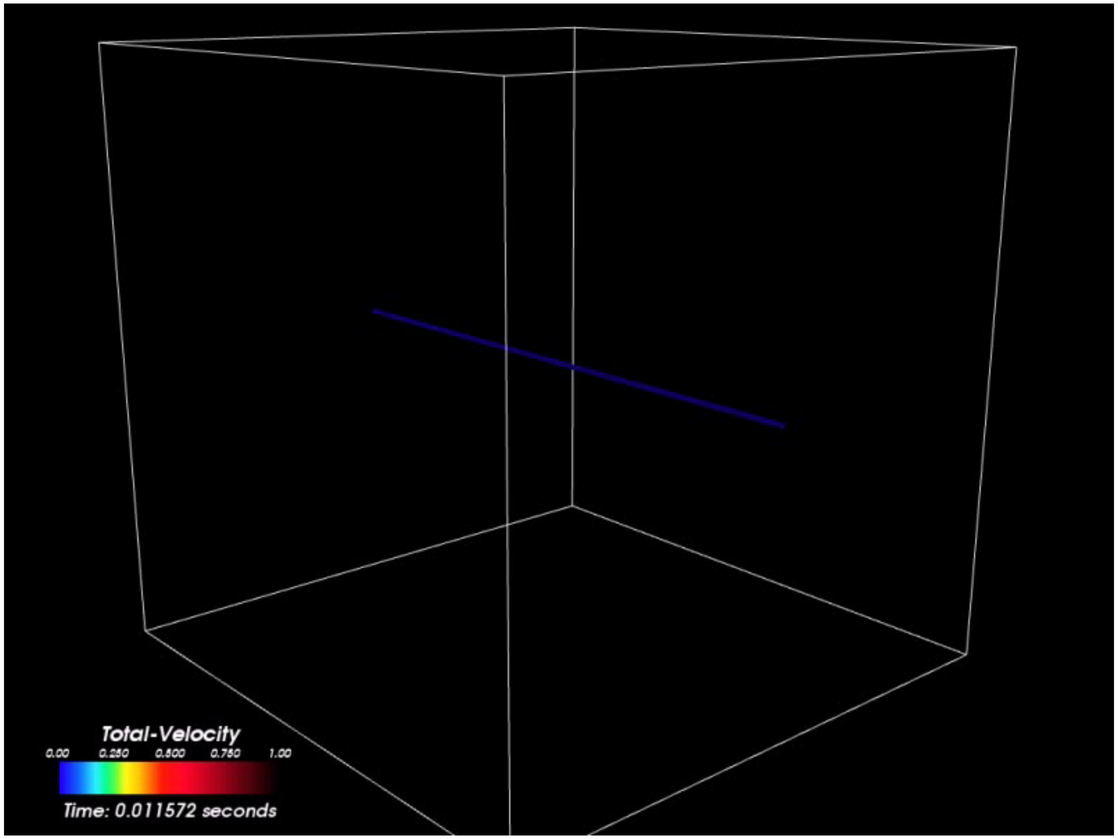
The Code

- More than half a million lines
- Multiphysics with AMR
- Public code with reasonably large user base
- Must run on multiple platforms
- Must be efficient on most platforms

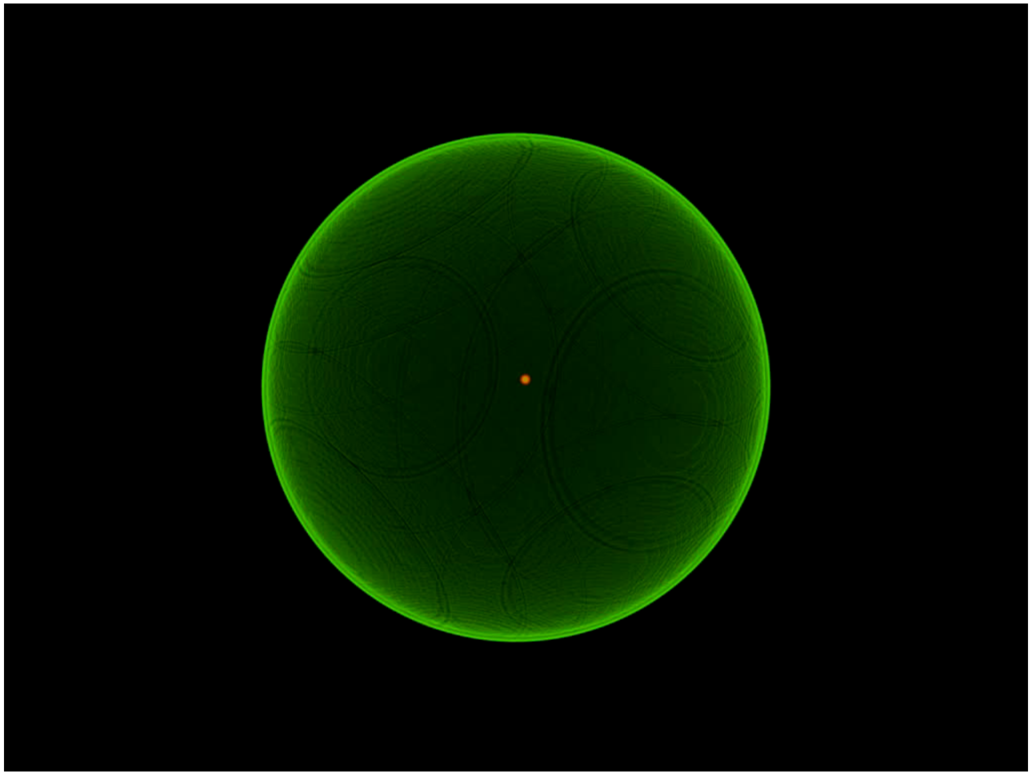
The layered architecture of the code comes to the rescue
MPI optimizations at infrastructure level
Memory optimizations at wrapper level
Memory and flop optimizations at kernel level

BGL : 32 K nodes

- Weakly compressible turbulence simulation
- Lagrangian particles frame unscalable
  - The metadata duplicated on all processors
  - Limited memory, wouldn't fit.
- Designed a suite of new algorithms for data movement



The GCD Application



Application Description

- Start the simulation with an off center bubble
- The bubble rises to the surface, developing Rayleigh-Taylor instabilities
- The material cannot escape because of the gravity, so it races around the star
- At the opposite end, the fronts collide to initiate detonation

Performance Issues

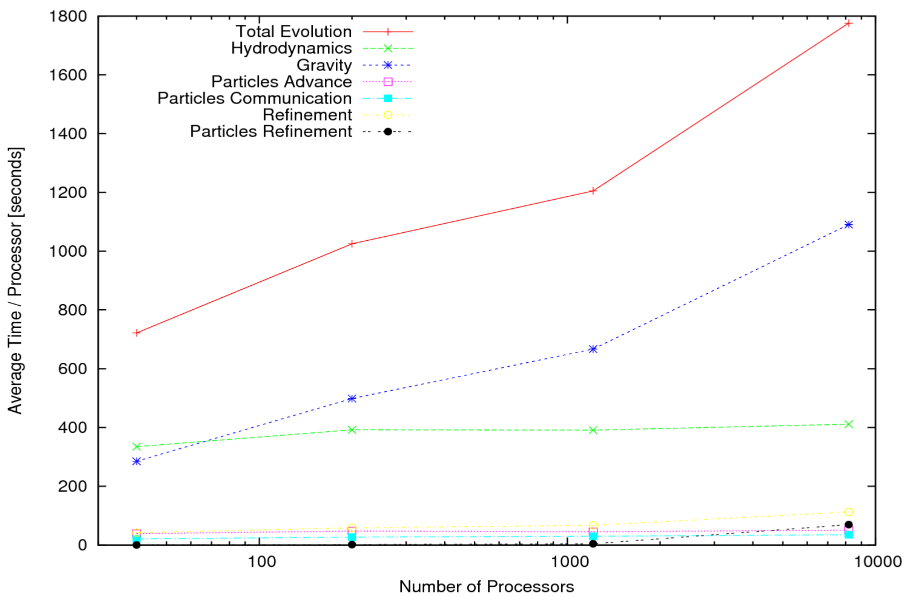
- Load imbalance in flame
- Too much time in gravity
- Memory limitation from particles
- Memory limitation from refinement

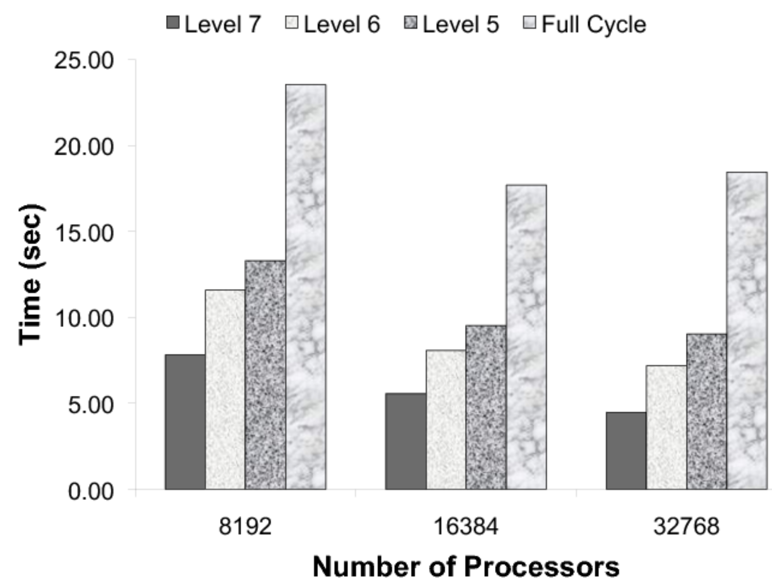
Optimizations

- Trade-off between accuracy and time
- Refinement criterion
- Table lookup instead of calculations

Multigrid optimization

- Motivation: Weak scaling results from a galaxy cluster simulation





- Gravity (and specifically the Multigrid solver) is the bottleneck
- Multigrid V-Cycles cause processor starvation on coarse grids
- The Solution : Switch to exact solution at a predetermined level
    - Re-arrangement of grid needed; parallel algorithm for mapping

Four sections in the talk

- Section 1 : General information and evolution of the framework
- Section 2 : The current code architecture
- Section 3 : History of simulations and the performance challenges at various stages of evolution
- **Section 4 : Going to exa-scale**

Exascale Through Co-Design

Inter-node Challenges

Challenges

- Parallel IO
  - Analysis memory snapshot a large fraction of total system memory
- Higher degree of macro parallelism
  - Load balance
  - Meta-data handling
- Higher fidelity physics dictates greater coupling
  - Implicit/semi-implicit treatment

Possibilities

- Different approach through data staging
  - Critical vs. non-critical data
  - Combine with *in situ* analysis
- New parallel algorithms
  - Trade-off between duplication and communication
  - Possibly more hierarchy
- Investigate different class of numerical algorithms
  - Less deterministic

Intra-Node and Resiliency Challenges

Challenges	Possibilities
Intra-Node	
❑ Memory intensive computations	❑ Aggressive reuse of memory
❑ Increasing limits on available memory per process	❑ Distinguish between cores
❑ Bigger working sets	❑ New algorithms
	❑ Programming model
Faults	
❑ Frequent failures	❑ Stochastic algorithms
❑ Silent errors	❑ Redundancy

Code Maintenance and Co-Design

- Code verification and regression testing
  - Expect more non-determinism and async execution models to get performance and scalability
  - But to do regression testing without reproduci...
  - Will study approaches to selectable d...
    - Changes to compiler or ru...
    - Changes in algori...

Performance vs. Maintainability vs. Portability

Auto-tuning, code to code translation, annotations

- Ti... ...science advances into the ... obsolescence of code modules)
  - ...g of new algorithms / implementation coming about because of new knowledge/insights

Co-Design Needs from Application

- Greater encapsulation
  - Minimize common data
  - Maximize code sections that are re-entrant
  - Increase isolation between layers
  - Separate code functionalities such that different optimizations are applicable to different layers
- Minimize kernel dependency on programming models
- Expose optimization and fault tolerance possibilities
  - Be clearer about dependencies
  - Identify critical sections Vs the non critical sections
  - Define more compact working sets
- Explore more inherently robust alternative algorithms
  - Stochastic Vs deterministic

What We Need to Achieve

- Measurable and predictable performance
- Reliable results within quantified limits
- Retain code portability and performance
  - Standardized interfaces for common functionalities
  - Libraries and middleware
  - Auto-tuning or code to code translation
- Memory management
  - Memory bound application
- IO management
  - Large volumes of analysis data
  - Currently one snapshot roughly 1/10th of memory footprint
  - Analysis a judicious combination of in-situ and post processing

Questions ?