Optimizing FLASH While Retaining Portability

Anshu Dubey
University of Chicago/ANL

Nov 20, 2012

The FLASH Code Contributors

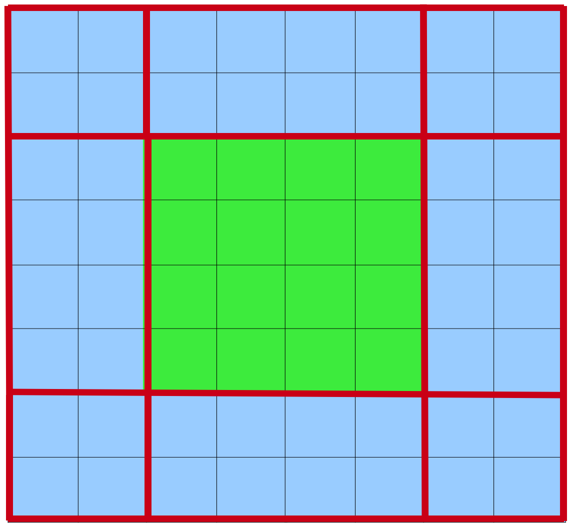
- Current Contributors in the Center:
  - John Bachan, Sean Couch, Chris Daley, Milad Fatenejad, Norbert Flocke, Carlo Graziani, Cal Jordan, Dongwook Lee, Min Long, Prateeti Mohapatra, Anthony Scopatz, Petros Tzeferacos, Klaus Weide,
- Current External Contributors:
  - Paul Ricker, John Zuhone, Marcos Vanella, Mats Holmstrom, Chris Orban, Igor Golovkin, Tommaso Vinci, William Gray, Christoph Federreth, Robi Banerjee, Richard Wunsch
- Past Major Contributors:
  - Katie Antypas, Alan Calder, Jonathan Dursi, Robert Fisher, Bruce Fryxell, Murali Ganapathy, Shravan Gopal, Nathan Hearn, 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

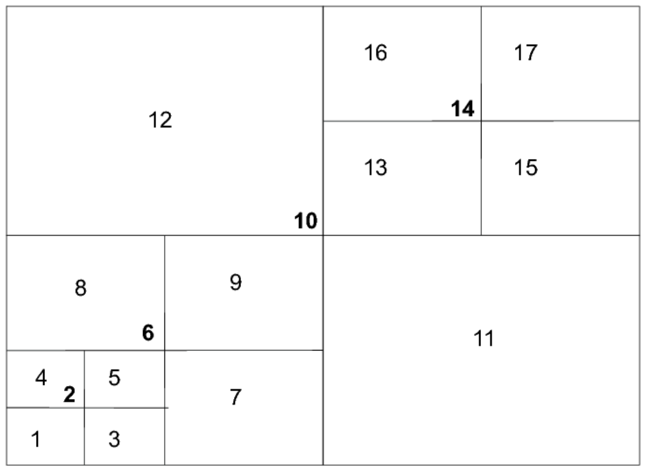
FLASH Basics

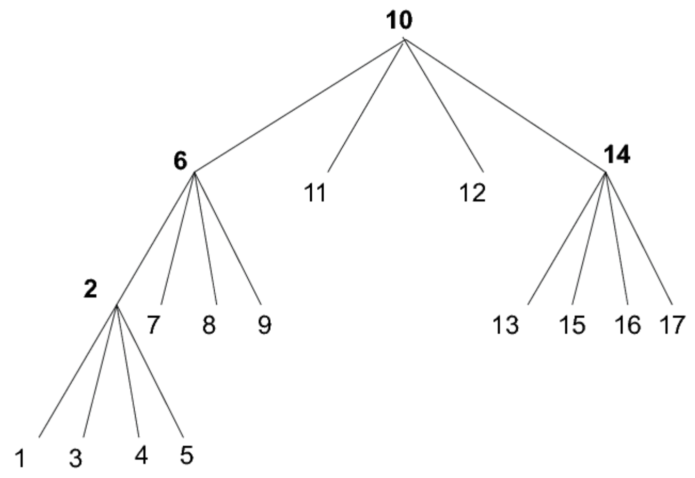
- More than 1.25 million lines of code
- Roughly 25% comments
- Infrastructure :
  - Mesh – UG, AMR, mesh replication
  - IO – HDF5, PnetCDF, Direct
  - Lagrangian Particles
  - Runtime support
  - Monitoring and diagnostics
- Capabilities
  - Compressible Hydro, MHD
  - Incompressible Hydro
  - Gravity
  - EOS, Material properties
  - Radiation transfer
  - Cosmology
  - Source terms
    - Radiation, Burn, Flame, Laser drive, Chemistry, Heat, Cool, Stir
  - Immersed boundaries

Computation Block and Oct-tree

- Blocks consist of cells: guard cells and interior cells.
- For purposes of guard cell filling, guard cells are organized into guard cell regions.



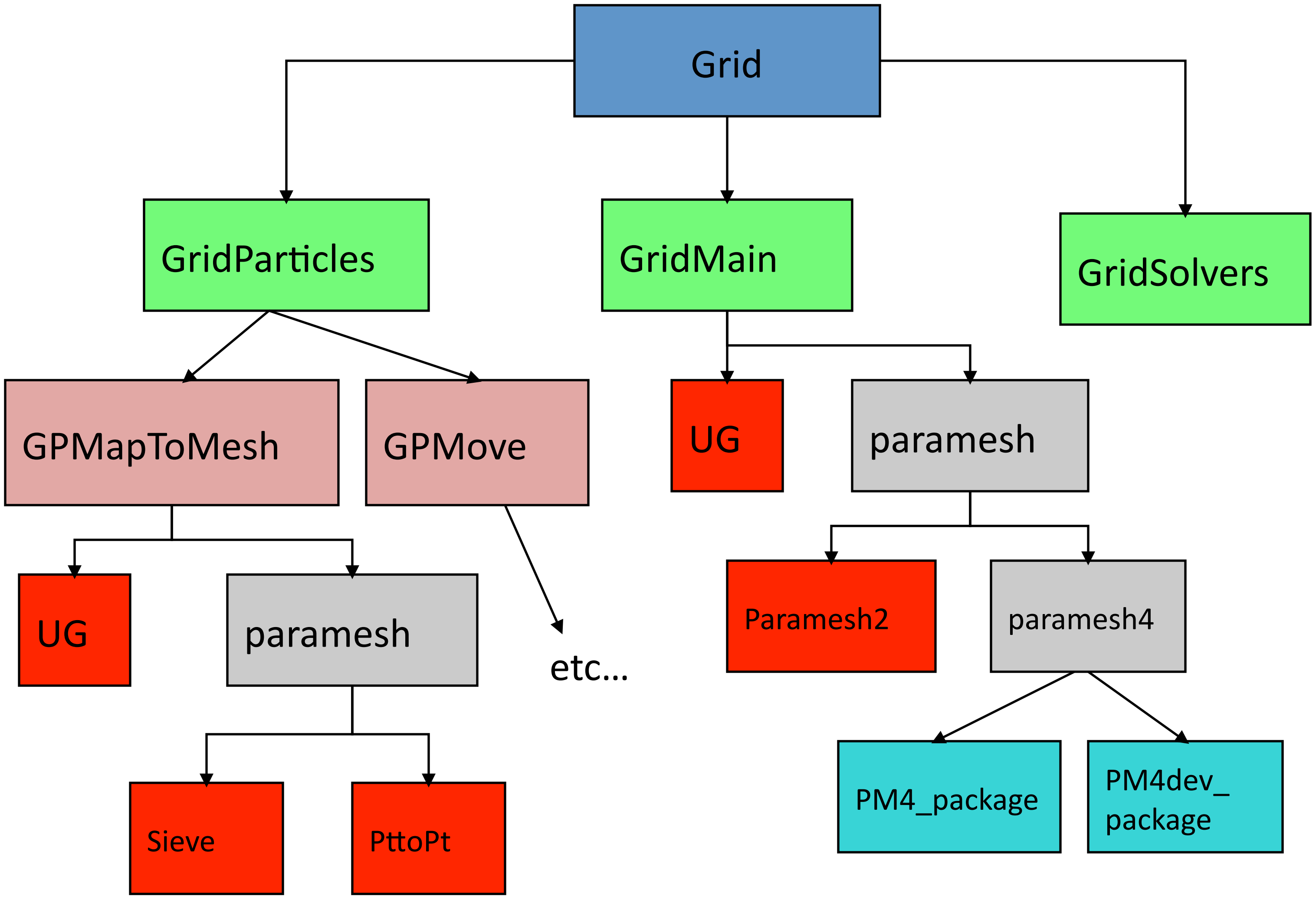




Architecture : Unit

- 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
    - Any sub-unit or a component of a subunit can have multiple alternative implementations
    - Different implementations can be picked for different applications

Example of a Unit – Grid (simplified and incomplete)



How We Optimize

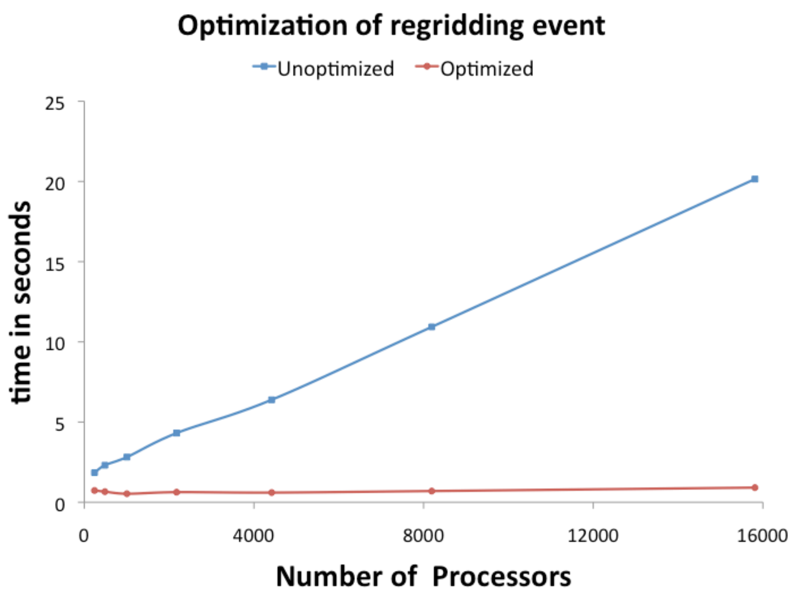
- Rarely too specific to the target machine
  - Portability is extremely important
    - Large user base, runs on all kinds of platforms
    - HPC platforms have short shelf life and the code base is large
    - Usually simulation campaigns run on multiple platforms
  - Bigger dividends come from:
    - Better algorithms
    - Eliminating weaknesses in implementations
    - Domain based problem specific intuitions
- Platform specific optimizations mostly limited to infrastructure
  - Alternative implementations
  - Runtime or configuration time knobs

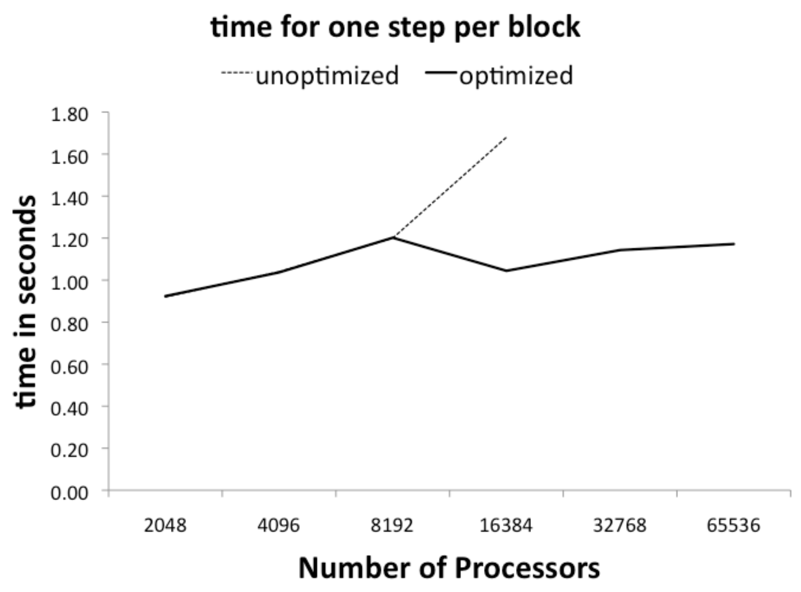
Example : Better Algorithms

- Lagrangian particles – non-native data structure, not automatically managed by mesh, but needs intimate mesh knowledge
  - Early implementation – replicate oct-tree meta-data
  - 32K procs BGL metadata wouldn't fit in memory
  - Three new parallel algorithms
    - Directional move for UG
    - Point-to-point for displacement in AMR
      - Makes use of view of the tree constructed for guard-cell fill
    - Sieve for migrating data in the re-gridding event
- Mapping Lagrangian quantities to mesh
  - Virtual particles

Example: Eliminate Weakness in Implementation

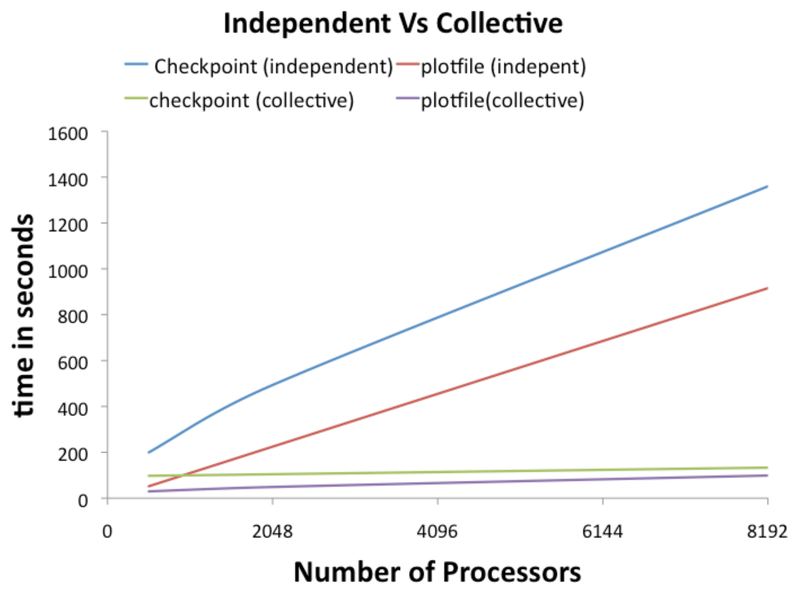
- PARAMESH has a “Digital Orrery” algorithm for updating some block neighbor information.
  - The “Digital Orrery” hands meta-information about blocks around until every PE has seen it.
- PARAMESH builds face neighbors information without the Orrery
- A few extra nearest neighbor communications get edge and vertex neighbors

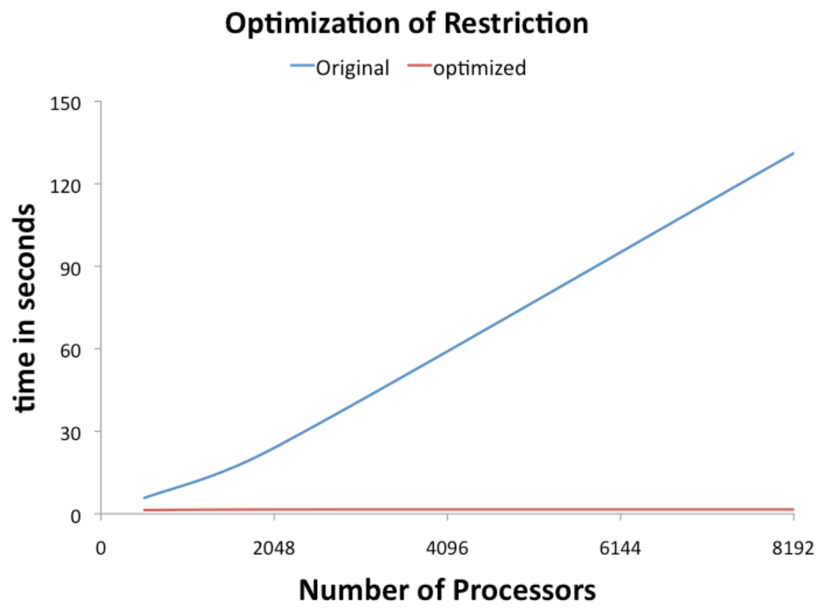




Example: Eliminate Weakness in Implementation-IO

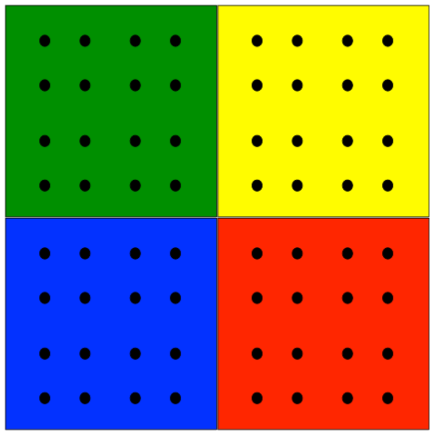
- Motivating factor: up to 35% runtime spent in I/O!
- Limiting factors :
  - Collective I/O Silent error, data corruption
    - Fix in the library
  - FLASH AMR data restriction – reuse of metadata
  - Modification of FLASH file format



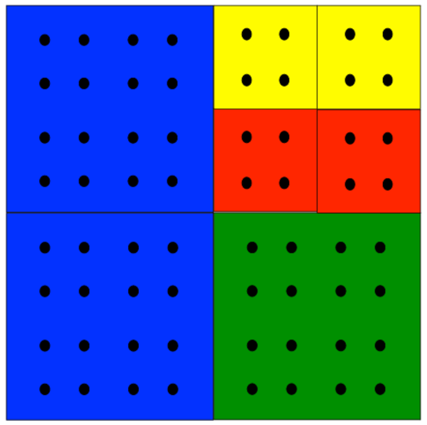


Example: Utilize Domain Knowledge

- The GCD Simulation – computational challenges
  - The difference in scales
    - The whole star - 8 km resolution, flame thickness ranges from 1 mm to a few cm.
  - The Multi-pole Self Gravity solver needs many moments to keep the star from oscillating
  - Over-refinement, too many blocks
  - Load imbalance in the Flame



- The initialization challenge
  - Particles distributed based on density
  - As bubble grows, all new blocks get added along the bubble
  - The regions outside the bubble don't refine; so blocks concentrate on a few processors : Memory crunch

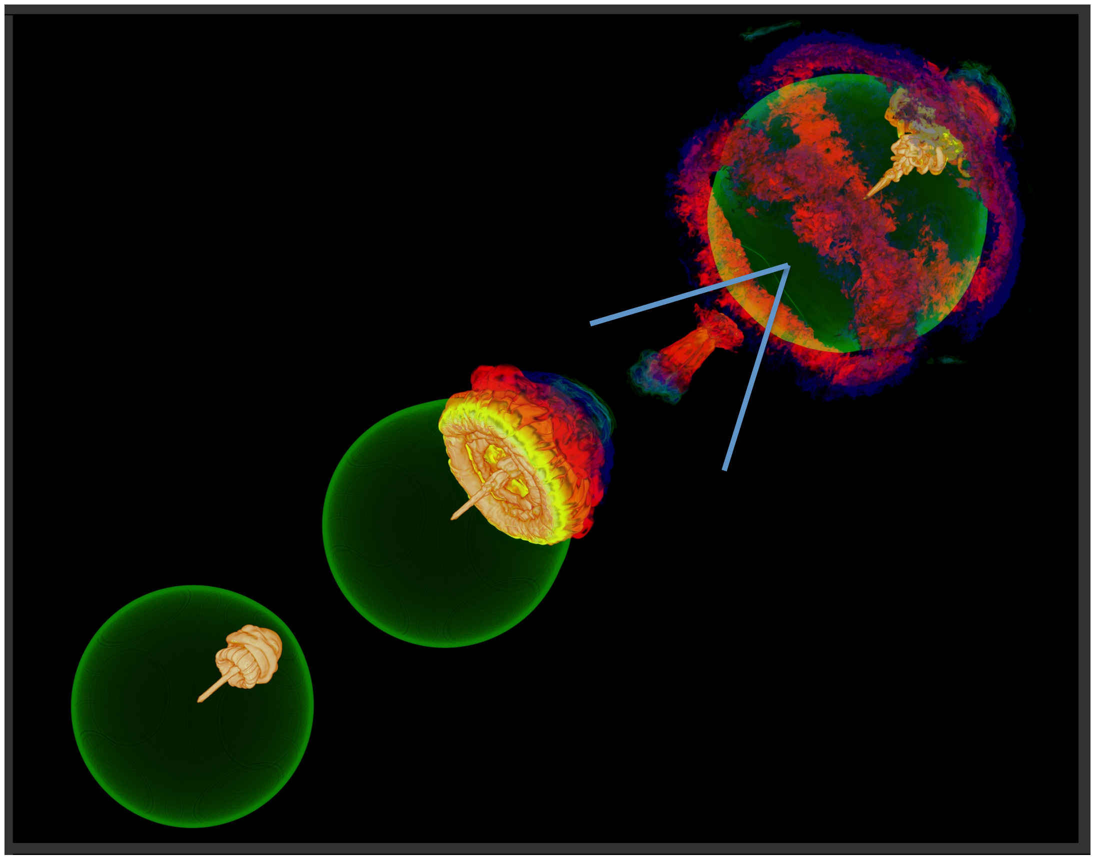


Optimizations

- Gravity Solver
  - Make the bubble rise along the Z axis.
    - Higher moments need fewer terms because of symmetry
    - Reduce the subsampling
    - Reduced the gravity cost by more than a factor of 10
- The Flame
  - Pre-compute the NSE states
    - Changed the computation from iteration to table lookup
    - Saving of another order of magnitude in flame calculation
- The Lagrangian Tracer Particles
    - Force blocks to refine, even though hydro doesn't require it

The Ultimate Black Magic – Refinement Criterion

- When the bubble is rising to the surface, same criterion applied all over the domain
  - Not much happening anywhere except close to the bubble
- Later when the fronts are moving around, there is also action elsewhere
  - We are not interested in that action
  - Increases number of blocks
  - Application won't fit
- Define regions (a cone here) to refine.
  - Sometimes force refinement to capture action at very small scales



Current Optimizations

- MPI + OpenMP Hybrid mode
  - Coarse – give a separate block to each thread
  - Fine – apply directives to performance critical loops
    - Another general possibility-fragment blocks
      - Not possible without modifications because of guard-cell overheads
    - Replace in-place calculation with a separate destination blocks
      - Not production ready
- Communication reduction through mesh replication
  - Mostly used for radiation with flux limited diffusion
- Runtime knobs for trial and error for optimal ranks / thread combination

Future Optimizations

- Requirements
  - Maintainable code, support large user community
  - Reliable results within quantified limits
  - Retain code portability and performance
  - Measurable and predictable performance
- The challenges in meeting the requirements
  - Ongoing
    - Modularity and performance
    - Readabale/maintainable code and portability
  - New ones
    - Easy adaptability to new and heterogeneous architectures and complex multiphysics capabilities
    - Regression test based verification and tolerance for non reproducibility

One Possible Approach

Provide some foothold for abstractions

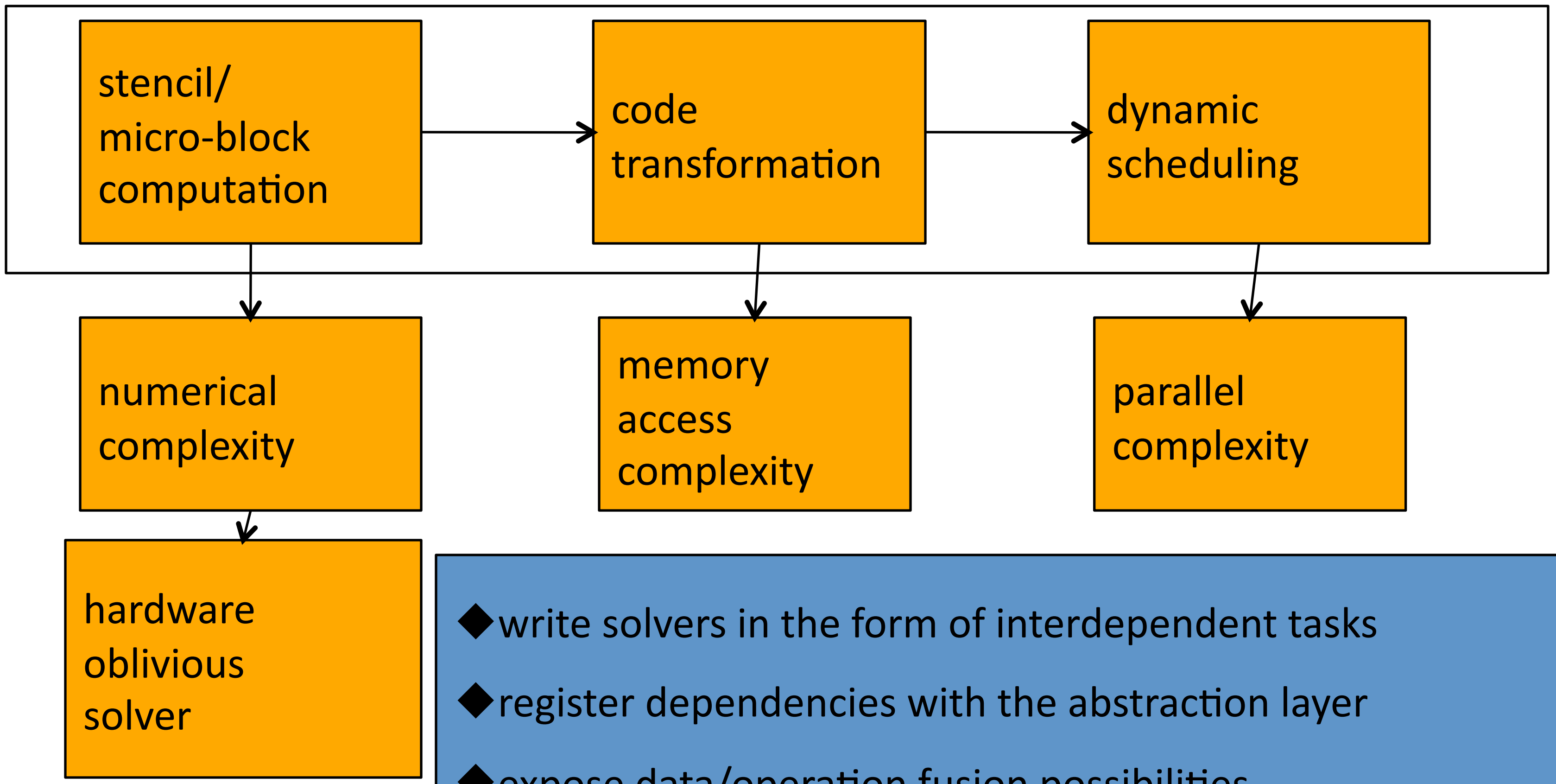
- Separate complexity : Example of a deterministic code
  - Physical model, and its numerical algorithms
  - Implementation – data structures and therefore memory access patterns
  - Parallelism
- Expose parallelism opportunities
  - Domain decomposition
  - Task parallelism
  - Operation parallelism

Implications of Data Structures

- At present we get a block, cell coordinates and other relevant grid meta data explicitly
  - Assumption is solver knows what it wants and gets it
  - Solver has to make receiving data structures conform to the mesh -> has to know them
- Solver could specify mesh data it wants for each cell and ask the grid to fill it
  - higher meta-data overheads if done on a per cell basis
  - if not on per cell basis then solver has to decide on loop bounds for calculations

The abstraction layers should do appropriate fusions and code transformations and use dynamic runtime management to orchestrate the computation for performance

Mapping to Programming Abstractions

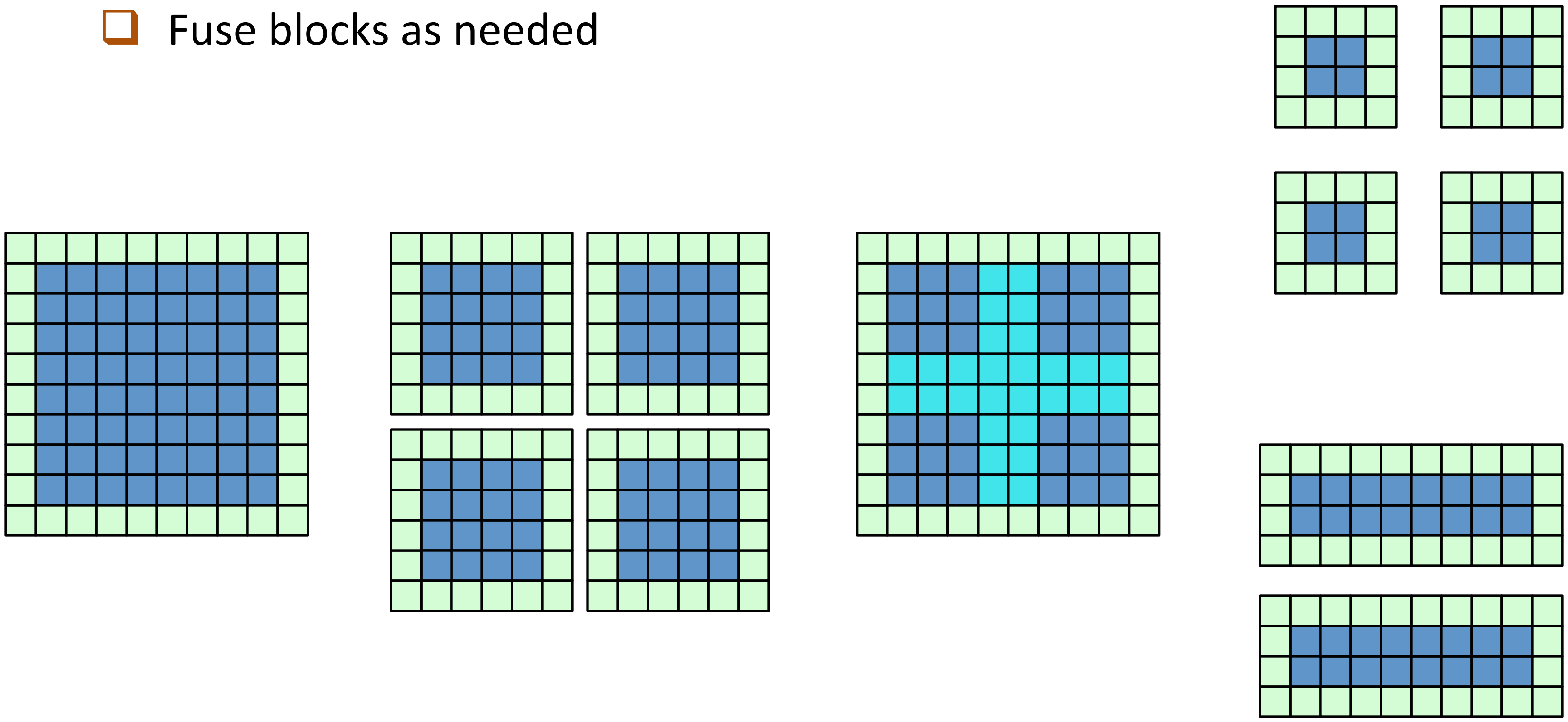


- ◆write solvers in the form of interdependent tasks
- ◆register dependencies with the abstraction layer
- ◆expose data/operation fusion possibilities

Need programming languages with richer collection of data structures and high level constructs that allow expression of computations with much less detail

Specific Projects : Memory

- Memory optimization
    - Replace in-place computations
    - Make blocks arbitrarily small
    - Stage data for cache
    - Fuse blocks as needed



Specific Projects

- Fault Tolerance – soft errors
  - AMR can provide low fidelity solution simultaneous to high fidelity solution
  - In non-critical parts of the domain use for reconstruction
  - Expected outcome
    - Help devise the interface for notification to app
    - Study the overall degradation in solution
    - Find the limits of error tolerance
- Communication reduction
  - Selective mesh replication
- Different refinement patterns for different operators
  - Especially beneficial if only one operator is more demanding of resolution