Modular GRMHD: Con2Prim Routines

Samuel Cupp
in collaboration with Terrence Pierre Jacques,
Leo Werneck, Zach Etienne

University of Idaho

November 2022

Motivation

- Many different general relativistic (magneto)hydrodynamic codes implement similar or even identical pieces of code. Some examples are
  - routines to convert between conservative and primitive variables
  - Reconstruction routines (PPM, WENO, etc.)

Motivation

- Many different general relativistic (magneto)hydrodynamic codes implement similar or even identical pieces of code. Some examples are
  - routines to convert between conservative and primitive variables
  - Reconstruction routines (PPM, WENO, etc.)
- GRMHD codes are often written in a way that is difficult for new users/developers–particularly students–to quickly understand

Motivation

- Many different general relativistic (magneto)hydrodynamic codes implement similar or even identical pieces of code. Some examples are
  - routines to convert between conservative and primitive variables
  - Reconstruction routines (PPM, WENO, etc.)
- GRMHD codes are often written in a way that is difficult for new users/developers–particularly students–to quickly understand
- separating different pieces of the code as much as is reasonable helps keep the intention of each piece of code clear, and makes the usage and dependencies of individual functions clearer

Motivation

- Many different general relativistic (magneto)hydrodynamic codes implement similar or even identical pieces of code. Some examples are
  - routines to convert between conservative and primitive variables
  - Reconstruction routines (PPM, WENO, etc.)
- GRMHD codes are often written in a way that is difficult for new users/developers–particularly students–to quickly understand
- separating different pieces of the code as much as is reasonable helps keep the intention of each piece of code clear, and makes the usage and dependencies of individual functions clearer
- IllinoisGRMHD already exists in two different infrastructures (Cactus, NRPy+), and a unified code structure across these different ecosystems will help streamline maintenance and future development of the code

Goal: Modularize IllinoisGRMHD

- We aim to modularize IllinoisGRMHD following four key tenets:

Goal: Modularize IllinoisGRMHD

- We aim to modularize IllinoisGRMHD following four key tenets:
  - Code design and language choices must minimize pipeline for new developers (particularly students)

Goal: Modularize IllinoisGRMHD

- We aim to modularize IllinoisGRMHD following four key tenets:
  - Code design and language choices must minimize pipeline for new developers (particularly students)
  - Compartmentalized code is simpler and aids in debugging/extensibility (Monoliths are bad!)

Goal: Modularize IllinoisGRMHD

- We aim to modularize IllinoisGRMHD following four key tenets:
  - Code design and language choices must minimize pipeline for new developers (particularly students)
  - Compartmentalized code is simpler and aids in debugging/extensibility (Monoliths are bad!)
  - Infrastructure agnosticism–IllinoisGRMHD is already used in two different infrastructures, and the new form should have no references to any infrastructure-specific features

Goal: Modularize IllinoisGRMHD

- We aim to modularize IllinoisGRMHD following four key tenets:
    - Code design and language choices must minimize pipeline for new developers (particularly students)
    - Compartmentalized code is simpler and aids in debugging/extensibility (Monoliths are bad!)
    - Infrastructure agnosticism–IllinoisGRMHD is already used in two different infrastructures, and the new form should have no references to any infrastructure-specific features
    - Proper documentation, code tests, and examples not only help internal work but encourage wider community adoption and contributions

Goal: Modularize IllinoisGRMHD

- We aim to modularize IllinoisGRMHD following four key tenets:
  - Code design and language choices must minimize pipeline for new developers (particularly students)
  - Compartmentalized code is simpler and aids in debugging/extensibility (Monoliths are bad!)
  - Infrastructure agnosticism–IllinoisGRMHD is already used in two different infrastructures, and the new form should have no references to any infrastructure-specific features
  - Proper documentation, code tests, and examples not only help internal work but encourage wider community adoption and contributions
- In addition, thorns will be created to provide these features within the Einstein Toolkit

Goal: Modularize IllinoisGRMHD

- We aim to modularize IllinoisGRMHD following four key tenets:
    - Code design and language choices must minimize pipeline for new developers (particularly students)
    - Compartmentalized code is simpler and aids in debugging/extensibility (Monoliths are bad!)
    - Infrastructure agnosticism–IllinoisGRMHD is already used in two different infrastructures, and the new form should have no references to any infrastructure-specific features
    - Proper documentation, code tests, and examples not only help internal work but encourage wider community adoption and contributions
- In addition, thorns will be created to provide these features within the Einstein Toolkit
- Work has begun on conservative-to-primitive routines, equation of state code, and reconstruction code

Philosophy: Contributor Pipeline

- Students of NR must become experts in
  - Physics
  - Mathematics
  - Computer Science
  - Astronomy

Philosophy: Contributor Pipeline

- Students of NR must become experts in
    - Physics
    - Mathematics
    - Computer Science
    - Astronomy
- Lowering barrier for entry is important to encourage continuing development and contribution, as well as higher adoption rates for the code
- Largest barrier for many physicists is often computational in nature

Philosophy: Contributor Pipeline

- Students of NR must become experts in
  - Physics
  - Mathematics
  - Computer Science
  - Astronomy
- Lowering barrier for entry is important to encourage continuing development and contribution, as well as higher adoption rates for the code
- Largest barrier for many physicists is often computational in nature
- Most common background for current physicists is Python and basic C++
- We aim to minimize barrier for entry by using
  - Python for codegen (NRPy+)
  - C with minimal data structures, minimal abstraction

Philosophy: Infrastructure Agnosticism

- IllinoisGRMHD, like many codes, is a monolithic code, meaning
  - it is difficult/messy to extend or improve
  - all code exists in the same directory–functions/routines not easily discoverable to developers
  - the code hierarchy is opaque and must be figured out by each new developer

Philosophy: Infrastructure Agnosticism

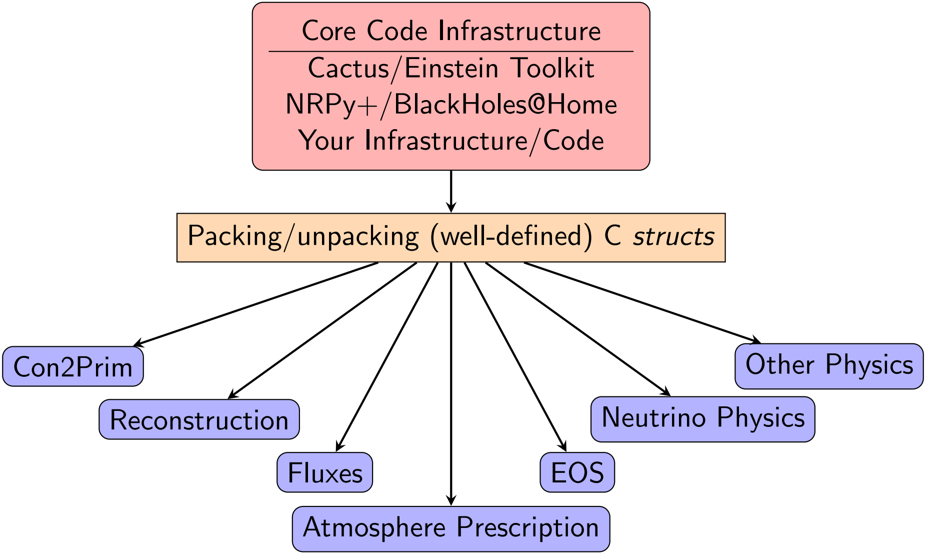
- IllinoisGRMHD, like many codes, is a monolithic code, meaning
  - it is difficult/messy to extend or improve
  - all code exists in the same directory–functions/routines not easily discoverable to developers
  - the code hierarchy is opaque and must be figured out by each new developer
- IllinoisGRMHD must also work in 2 different infrastructures:
  - Einstein Toolkit
  - BlackHoles@Home (NRPy+-based)

Philosophy: Infrastructure Agnosticism

- IllinoisGRMHD, like many codes, is a monolithic code, meaning
  - it is difficult/messy to extend or improve
  - all code exists in the same directory–functions/routines not easily discoverable to developers
  - the code hierarchy is opaque and must be figured out by each new developer
- IllinoisGRMHD must also work in 2 different infrastructures:
  - Einstein Toolkit
  - BlackHoles@Home (NRPy+-based)
- Ultimate goals of modularization are to
  - "de-EinsteinToolkitify" IllinoisGRMHD
  - aim for zero dependencies on other codes

  while adopting best practices in code development, such as
  - Extensive documentation (via Jupyter notebooks)
  - Automatic codegen whenever useful (via NRPy+)
  - Self-contained unit tests for each module (with their own main() functions)
  - "Perfect" examples of how to implement the module

Project Vision



Core Code Infrastructure

Cactus/Einstein Toolkit

NRPy+/BlackHoles@Home

Your Infrastructure/Code

Packing/unpacking (well-defined) C *structs*

Con2Prim

Reconstruction

Fluxes

Atmosphere Prescription

EOS

Neutrino Physics

Other Physics

Conservative-to-Primitive Routines

- Many Con2Prim routines exist which have different failure conditions, robustness, etc.

Conservative-to-Primitive Routines

- Many Con2Prim routines exist which have different failure conditions, robustness, etc.

- Several groups use the same Con2Prim routines, but they are included separately within each group's code

Conservative-to-Primitive Routines

- Many Con2Prim routines exist which have different failure conditions, robustness, etc.
- Several groups use the same Con2Prim routines, but they are included separately within each group's code
- Creating a shared library of Con2Prim routines with a common interface would simplify the process of experimenting with new routines for all groups

Conservative-to-Primitive Routines

- Many Con2Prim routines exist which have different failure conditions, robustness, etc.
- Several groups use the same Con2Prim routines, but they are included separately within each group's code
- Creating a shared library of Con2Prim routines with a common interface would simplify the process of experimenting with new routines for all groups
- Implementation of a new routine can be immediately tested and used by others once it is made public.

Development Progress

- several structs are defined to uniformly provide data about the simulation

Development Progress

- several structs are defined to uniformly provide data about the simulation
- Point-wise data structs provide convenient method for communicating data about conservatives, primitives, and metric quantities to Con2Prim methods
- Comments detail the quantities contained in each structure, as well as any assumptions (e.g. conservatives are densitized)

Development Progress

- several structs are defined to uniformly provide data about the simulation
- Point-wise data structs provide convenient method for communicating data about conservatives, primitives, and metric quantities to Con2Prim methods
- Comments detail the quantities contained in each structure, as well as any assumptions (e.g. conservatives are densitized)
- Code has been condensed into a small number of functions, and some logic has been reduced or pulled into the associated functions

Development Progress

- several structs are defined to uniformly provide data about the simulation
- Point-wise data structs provide convenient method for communicating data about conservatives, primitives, and metric quantities to Con2Prim methods
- Comments detail the quantities contained in each structure, as well as any assumptions (e.g. conservatives are densitized)
- Code has been condensed into a small number of functions, and some logic has been reduced or pulled into the associated functions
- Functions for packing/unpacking the structs are provided

Con2Prim Pseudocode

```
Declare/initialize GRMHD_parameters params;
Declare/initialize eos_parameters eos;
Declare/initialize con2prim_diagnostics diagnostics;
OMP for (loop over grid)
{
  Declare/initialize metric_quantities metric;
  Declare/initialize primitive_quantities prims, prims_guess;
  Declare/initialize conservative_quantities cons, cons_undens;

  apply_inequality_fixes( &params, &eos, &metric, &prims, &cons, &diagnostics);

  undensitize_conservatives( &eos, &metric, &prims, &cons, &cons_undens );

  guess_primitives( &eos, &metric, &prims, &cons, &prims_guess );

  con2prim_method( &eos, &metric, &cons_undens, &prims_guess, &diagnostics );

  if(con2prim fails)
    font_fix( &eos, &metric, &cons_undens, &prims, &prims_guess, &diagnostics);

  return_primitives(&prims, primitive variable pointers);

  return_conservatives(&cons, conservative variable pointers);
}

diagnostic_report(&diagnostics);
```

Current Status and Remaining Tasks

- significant progress towards condensing Con2Prim code into a small number of self-contained function calls
- Con2Prim thorn now compiles, can be called by IllinoisGRMHD's Con2Prim driver function
- Next steps are debugging and validation

Current Status and Remaining Tasks

- significant progress towards condensing Con2Prim code into a small number of self-contained function calls
- Con2Prim thorn now compiles, can be called by IllinoisGRMHD's Con2Prim driver function
- Next steps are debugging and validation
- Still has a "kernel" function containing much of the logic which needs to be popped open and condensed after debugging changes from IllinoisGRMHD's arrays to the new structs

Current Status and Remaining Tasks

- significant progress towards condensing Con2Prim code into a small number of self-contained function calls
- Con2Prim thorn now compiles, can be called by IllinoisGRMHD's Con2Prim driver function
- Next steps are debugging and validation
- Still has a "kernel" function containing much of the logic which needs to be popped open and condensed after debugging changes from IllinoisGRMHD's arrays to the new structs
- Current thorn implementation is tentative and will change significantly before release

Current Status and Remaining Tasks

- significant progress towards condensing Con2Prim code into a small number of self-contained function calls
- Con2Prim thorn now compiles, can be called by IllinoisGRMHD's Con2Prim driver function
- Next steps are debugging and validation
- Still has a "kernel" function containing much of the logic which needs to be popped open and condensed after debugging changes from IllinoisGRMHD's arrays to the new structs
- Current thorn implementation is tentative and will change significantly before release
- Code will move to a different location eventually, but my working version is available at `https://github.com/SamuelCupp/Con2Prim_beta`