CoordGauge

Tom Goodale

Date: 2004/06/11 13:13:24

Abstract

This thorn provides an infrastructure for dynamic gauge selection. However, at present (summer 2004) many thorns don't use this infrastructure.

1 The Infrastructure

The **CoordGauge** implementation schedules five groups:

```
CoordGauge
LapseSelect IN CoordGauge BEFORE PickCoordGauge
LapseApply  IN CoordGauge AFTER  PickCoordGauge
ShiftSelect IN CoordGauge BEFORE PickCoordGauge
ShiftApply  IN CoordGauge AFTER  PickCoordGauge
```

and one function

```
PickCoordGauge IN CoordGauge
```

and has two public grid scalars

```
selected_lapse
selected_shift
```

and two string parameters

```
lapse_list
shift_list
```

It also provides four aliased functions

```
int CoordGauge_RegisterLapse("lapse-name")
int CoordGauge_RegisterShift("shift-name")
CoordGauge_Lapse("lapse-name")
CoordGauge_Shift("shift-name")
```

(If someone can think of better names, please say so 8-)

Then each thorn which wants to apply a coordinate gauge condition registers itself, receiving a unique integer as an id, and schedules a selection routine and an application routine in the appropriate schedule groups.

The selection routine decides if this gauge condition should be applied at this time, and calls the `CoordGauge_Lapse/Shift` aliased function. (It should check that it is actually in the appropriate parameter as a minimum.)

The `PickCoordGauge` function traverses the list of lapses/shifts and selects the first one in the list which has called the `CoordGauge_Lapse/Shift` aliased function and sets the appropriate grid scalar to the id of this one.

The application routine checks to see if the grid scalar is set to its id, and if so, applies the gauge condition.

Evolution thorns could schedule **CoordGauge** at the appropriate point or points in their schedule.

An advantage of this scheme over the current one is that it provides the selection routines with a full set of variables from which to decide whether they should apply a guage or not. So it becomes very easy to choose to switch off maximal if the lapse has collapsed within a certain volume, etc.

This is simpler than the previous scheme as there is no arbitrary 'bid' floating around. It also allows us to keep the logic of the final selection in one place, thus allowing people to override this logic if they need to.

2 Current Status

As of summer 2004, many thorns don't use the above mechanism, instead they directly extend `ADMBase::lapse_evolution_method` and/or `ADMBase::shift_evolution_method`. That is, the thorn implementing a coordinate condition thorn says in its `param.ccl`:

```
EXTENDS KEYWORD lapse_evolution_method "Which lapse condition to use"
{
"super-duper" :: "my super-duper new lapse condition"
} ""
```

(and/or the equivalent for `shift_evolution_method`).

The thorn then schedules a routine in some suitable schedule bin/group (probably `CCTK_PRESTEP` or `MOL_PRESTEP`) to check `ADMBase::lapse_evolution_method` and/or `ADMBase::shift_evolution_method`, and if they're equal to the appropriate string, it does the coordinate condition.