The Whisky, Cactus, Carpet codes and numerical methods for relativistic hydrodynamics

Luca Baiotti

Yukawa Institute for Theoretical Physics
University of Kyoto
Plan of the lecture

• A bit of motivation
  - the equations we solve
  - the publicly available codes
  - results obtained so far

• Details of the codes
  - Cactus
  - Whisky
  - Carpet
  • all with practical examples
Introduction to Whisky
The fundamental equations

The fundamental equations standard numerical relativity aims at solving are the **Einstein equations**:

\[
G_{\mu\nu} \equiv R_{\mu\nu} - \frac{1}{2} g_{\mu\nu} R = 8\pi T_{\mu\nu}
\]

- **Spacetime**
- **Curvature scalar**
- **Matter and other fields**
- **Metric**: measure of spacetime distances

\[
T^\mu_\nu = \rho (1 + \varepsilon + p/\rho) u^\mu u^\nu + pg^{\mu\nu}
\]

- **Rest-mass density**
- **Internal energy density**
- **Pressure**

i.e.

- six, second-order-in-time, second-order-in-space, coupled, highly-nonlinear, quasi-hyperbolic partial differential equations (PDEs)
- four, second-order-in-space, coupled, highly-nonlinear, elliptic PDEs
When the spacetime is not vacuum

The additional set of equations to solve is:

$$\nabla_\mu T^{\mu\nu} = 0$$

i.e. 4 conservation equations for the energy and momentum,

$$\nabla_\mu (\rho u^\mu) = 0, \quad p = p(\rho, \varepsilon)$$

i.e. conservation of baryon number and equation of state (EOS) (microphysics input).

Standard hydrodynamical quantities ($p, \varepsilon, v^i$) do not lead to a set of first-order PDEs in a flux-conservative form.

To hard-wire the conservative nature of the hydrodynamical equations, the ”primitive variables” are replaced by the “conserved variables”:

$$\begin{bmatrix} \rho \\ \varepsilon \\ v^i \end{bmatrix} \rightarrow \begin{cases} D = \rho W \\
S_j = \rho h W^2 v_j \\
E = \rho h W^2 - p \end{cases}$$

where \( h = 1 + \varepsilon + \frac{p}{\rho}; \quad W = \frac{1}{\sqrt{1 - v^i v_i}} \)
When the spacetime is not vacuum

As a result, a first-order flux-conservative form can be obtained

\[ \nabla_\mu T^\mu_\nu = 0 \quad \iff \quad \frac{\partial \bar{U}}{\partial t} + \frac{\partial \bar{F}^i}{\partial x^i} = \bar{S} \]

Where \( \bar{U} = (D, S_j, \tau \equiv E - D) \) is the vector of conserved variables

\[ \bar{F}^i(\bar{w}) = \sqrt{-g} \left( D \left( v^i - \frac{\beta^i}{\alpha} \right), S_j \left( v^i - \frac{\beta^i}{\alpha} \right) + p\delta^i_j, \tau \left( v^i - \frac{\beta^i}{\alpha} \right) + pv^i \right) \text{“fluxes”} \]

\[ \bar{S}(\bar{w}) = \sqrt{-g} \left( 0, T^\mu_\nu \left( \frac{\partial g^\nu_j}{\partial x^\mu} - \Gamma^\delta_j_\mu g^\delta_j \right), \alpha \left( T^\mu_0 \frac{\partial \ln \alpha}{\partial x^\mu} - T^\mu_\nu \Gamma^0_\mu_\nu \right) \right) \text{“sources”} \]

This is called the Valencia formulation of GR hydrodynamics
(Banyuls Font Ibáñez Martí Miralles, Astrophys. J., 476, 221)
When the spacetime is not vacuum

\[ \nabla_\mu T^{\mu\nu} = 0 \iff \frac{\partial U}{\partial t} + \frac{\partial F^i}{\partial x^i} = \bar{S} \]

\[ \bar{F}^i(\bar{w}) = \sqrt{-g} \left( D \left( v^i - \frac{\beta^i}{\alpha} \right), S_j \left( v^i - \frac{\beta^i}{\alpha} \right) + p\delta^i_j, \tau \left( v^i - \frac{\beta^i}{\alpha} \right) + pv^i \right) \] “fluxes”

\[ \bar{S}(\bar{w}) = \sqrt{-g} \left( 0, T^{\mu\nu} \left( \frac{\partial g_{\nu j}}{\partial x^\mu} - \Gamma^j_{\mu\nu} g_{\delta j} \right), \alpha \left( T^{\mu 0} \frac{\partial \ln \alpha}{\partial x^\mu} - T^{\mu\nu} \Gamma^0_{\mu\nu} \right) \right) \] “sources”

Note that

• the source terms do not contain differential operators acting on the stress-energy tensor (definition of flux-conservative system);

• because of the presence of a source term, the conservative system is not strictly homogeneous (this is sometimes called balance equation). Strict homogeneity is recovered in vacuum.
For the evolution of the Einstein equations we use the CCATIE code, which functions in Cactus and is used for black-hole simulations at the AEI (Albert-Einstein-Institut, Potsdam, Germany) and at CCT (Center for Computation and Technology, Louisiana State University, USA).

The Einstein equations are solved in the BSSN-NOK formalism.
The Einstein equations are solved in the BSSN-NOK formalism, in which the evolved variables are:

\[ \phi = \frac{1}{12} \ln \text{det} \gamma_{ij}, \quad \tilde{\gamma}_{ij} = e^{-4\phi} \gamma_{ij} \]

\[ K \equiv \text{tr} K_{ij} = g^{ij} K_{ij} \]

\[ \tilde{A}_{ij} = e^{-4\phi} (K_{ij} - \frac{1}{3} \gamma_{ij} K) \]

\[ \tilde{\Gamma}^i = \tilde{\gamma}^{jk} \tilde{\Gamma}^i_{jk} \]

and the evolved equations are:

\[(\partial_t - \mathcal{L}_\beta) \tilde{\gamma}_{ij} = -2\alpha \tilde{A}_{ij},\]

\[(\partial_t - \mathcal{L}_\beta) \phi = -\frac{1}{6} \alpha K ,\]

\[(\partial_t - \mathcal{L}_\beta) \tilde{A}_{ij} = e^{-4\phi} [-D_i D_j \alpha + \alpha (R_{ij} - 8\pi S_{ij})]^{TP} + \alpha (K \tilde{A}_{ij} - 2 \tilde{A}_{ik} \tilde{A}^{k}\),\]

\[(\partial_t - \mathcal{L}_\beta) K = -D^i D_i \alpha + \alpha \left[ \tilde{A}_{ij} \tilde{A}^{ij} + \frac{1}{3} K^2 + 4\pi (\rho_{\text{ADM}} + S) \right],\]

\[ \partial_t \tilde{\Gamma}^i = \tilde{\gamma}^{jk} \partial_j \partial_k \tilde{\Gamma}^i + \frac{1}{3} \tilde{\gamma}^{ij} \partial_k \partial_j \tilde{\Gamma}^i + \tilde{\Gamma}^i \partial_j \tilde{\Gamma}^j - \tilde{\Gamma}^j \partial_j \tilde{\Gamma}^i + 2 \tilde{A}^i \partial_j \tilde{\Gamma}^j - 2 \tilde{A}^{ij} \partial_j K + 2 \alpha (\tilde{\Gamma}^i \tilde{A}^{jk} + 6 \tilde{A}^{ij} \partial_j \phi \]

\[ - \frac{2}{3} \tilde{\gamma}^{ij} \partial_j K - 8\pi \tilde{\gamma}^{ij} S_j ,\]
Numerical Relativity with Cactus/Whisky

For the evolution of the Einstein equations we use the CCATIE code, which functions in Cactus and is used for black-hole simulations at the AEI (Albert-Einstein-Institut, Potsdam, Germany) and at CCT (Center for Computation and Technology, Louisiana State University, USA).

The Einstein equations are solved in the BSSN-NOK formalism.

There is also an alternative, public implementation: McLachlan
XiRel is a ambitious project, aiming at developing a highly scalable, efficient and accurate adaptive mesh refinement layer for the Cactus Framework, based on the Carpet driver, and optimized and supported for numerical relativists studying the physics of black holes, neutron stars and gravitational waves.

New tools will be distributed as part of a freely available Einstein Toolkit providing an open code for black hole simulations and so on and incorporating mechanisms to encourage code sharing, verification, and validation.

XiRel will create the basis for a modern, scalable cyberinfrastructure for the worldwide numerical relativity community, providing enabling software for highly scalable codes able to leverage today's petascale machines, to prepare for next generation architectures and computing paradigms, and to support effective collaboration.
**Numerical Relativity with McLachlan**

**McLachlan** is a publicly available, state-of-the-art BSSN-NOK code for solving the Einstein equations. McLachlan was created in the XiRel project, with the intention of building a public code for the numerical relativity community.

McLachlan should be "good enough" for "standard" calculations of black holes, neutron stars, binary systems of these, and many other cases. McLachlan uses Kranc for code generation and is based on the Cactus framework, the Carpet adaptive mesh refinement driver, and the Einstein Toolkit. It can be combined with Whisky.

McLachlan's main authors are Peter Diener, Erik Schnetter, and Jian Tao. McLachlan is freely available under the GPL license. A public code repository is planned but for the moment, please email Erik Schnetter <schnetter@cct.lsu.edu> to obtain a copy of the code.
Formulation, Gauge Conditions, Discretisation

McLachlan implements the so-called $\phi$ variant of the BSSN-NOK formulation with 1+log slicing

$$\partial_t \alpha - \beta^i \partial_i \alpha = -2\alpha(K - K_0)$$

and a $\Gamma$-driver shift condition. It supports both the first and second order (in time) variant of the 1+log slicing.

$$\partial_t \beta^i - \beta^j \partial_j \beta^i = \frac{3}{4} \alpha B^i,$$
$$\partial_t B^i - \beta^j \partial_j B^i = \partial_t \tilde{\Gamma}^i - \beta^j \partial_j \tilde{\Gamma}^i - \eta B^i$$

McLachlan uses $n$-th order accurate centred finite differencing stencils (for even $n$) with $(n+1)$-th order accurate Kreiss-Oliger dissipation.

$$\mathcal{L}_{\text{diss}} = -\varepsilon h^{n-1} \partial^n_i u$$

McLachlan can optionally be used with multi-patch systems, where finite differences need to take the Jacobian of the transformation between patch-local and global coordinates into account. A variety of time integrators are provided by the Cactus thorn CactusBase/MoL. The matter interface is provided by AEIThorns/TmunuBase.
Based on Cactus and exploiting the CCATIE code, Whisky (www.whiskycode.org) was developed at the AEI and SISSA starting in 2002; the code solves the general relativistic hydrodynamics equations in arbitrarily curved and time-varying spacetimes.

There exist a public version of Whisky, but most of the results were obtained with what at that time was a development “unstable” version with more features and small (though sometimes essential ...) improvements.
Gravitational-wave--source modelling
with Whisky
GW-source modelling with Whisky

- Neutron star oscillations: linear/nonlinear; magnetised/not
  Baiotti, Giacomazzo, Rezzolla

- Dynamical (barmode) instability
  Baiotti, De Pietri, Manca, Rezzolla

- Binary neutron stars
  Baiotti, Giacomazzo, Rezzolla

- Mixed binary systems
  Tonita, Löffler, Rezzolla

- Core collapse
  Ott

- Rotating neutron-star collapse to black holes
  Baiotti, Giacomazzo, Hawke, Rezzolla, Sarjo, Schnetter, Stergioulas
• **Whisky:**
  - full GR Hydrodynamical Code
  - uses **HRSC** (High Resolution Shock Capturing) methods
  - adopts **Adaptive Mesh Refinement** techniques through the **Carpet** code

• It’s meant as an “**astrophysical laboratory**” to study various sources of gravitational waves
The Whisky code

Some details (in the following slides)

- Carpet: a mesh-refinement driver
- Equations of state
- The general algorithm (HRSC)
- The Method of Lines for the time update
- The conversion from conservative to primitive variables
- The artificial atmosphere
- Other details
Carpet (www.carpetcode.org) is a mesh-refinement driver mainly developed by E. Schnetter [CQG 21, 1465 (2004)].

Carpet follows a (simplified) Berger-Oliger [J. Comput. Phys. 53, 484 (1984)] approach to mesh refinement, that is:

- refined sub-domains consist of a set of cuboid (=rectangular parallelepiped) grids
- refined sub-domains have boundaries aligned with the grid lines
- the refinement ratio between refinement levels is constant

While the refined meshes are not automatically moving on the grid, they can be activated and deactivated during the evolution, obtaining a progressive fixed mesh refinement or even a moving-grid mesh refinement.
Equations of state

The evolution equations of the matter are given by the conservation of the baryon number and energy-momentum:

\[ \nabla_\mu T^{\mu\nu} = 0 \]
\[ \nabla_\mu J^{\mu} = 0 \]
\[ J^{\mu} \equiv \rho u^{\mu} \]

plus an Equation of State \( P = P(\rho, \varepsilon) \).

The basic EoS’s implemented in the public version of Whisky are the ideal-fluid EoS:
\[ P = \rho \varepsilon (\Gamma - 1) \]

or the polytropic EoS:
\[ P = k \rho^\gamma \]

or a hybrid EoS:
\[ P = P_p + P_{th} \]
\[ P_{th} = \rho (\varepsilon - \varepsilon_p) (\gamma_{th} - 1) \]

See, e.g., Dimmelmeier et al PRD71 (2005)
The Whisky code

Notes on HRSC (High Resolution Shock Capturing) methods.

The real lecture will be given on Thursday by Prof. Font.
The method of lines

The MoL is a procedure to separate the space and time discretisation processes. First, the continuum equations are considered to be discretised in space only, while leaving the problem continuous in time. This is equivalent to transforming a set of PDEs into a set of ODEs. Then, the resulting system of ODEs can be solved numerically with any stable solver.

Given a grid and a spatial differencing L of the vector of variables U, the equations are written as ODEs in the form

$$\frac{d(\mathcal{U})}{dt} = L(\mathcal{U}) = \frac{1}{\Delta V} \left[ \int \int \int s \, d^3x + \int_{y_{q-1/2}}^{y_{q+1/2}} \int_{z_{r-1/2}}^{z_{r+1/2}} F^a_x(U(x_{p-1/2}, y, z)) \, dy \, dz - \int_{y_{q-1/2}}^{y_{q+1/2}} \int_{z_{r-1/2}}^{z_{r+1/2}} F^a_x(U(x_{p+1/2}, y, z)) \, dy \, dz \right]$$

**ADVANTAGES**

- Any stable ODE method can be used (RK2, RK3, RK4, ICN,...) and the control of the truncation error is transparent
- The coupling between different treatments in the hydro or spacetime is minimised: improvements in one code are instantly effective everywhere
- Different numerical methods are easily implemented (just a change in the way the L(q) is computed)
The evolved variables are the conserved variables, but in order to compute the fluxes, the source terms (and to have physical insight) it is necessary to transform back the updated result to the primitive variables.

The conversion from primitive to conservative is given analytically, but converting in the other direction is not possible in closed form. A standard procedure for the conversion consists in solving numerically the equation:

\[ p - \bar{p}[\rho(U, p), \varepsilon(U, p)] \]

where \( p \) is the pressure to be found and \( \bar{p} \) is the pressure expressed through the EoS, in terms of the updated conserved variables and the pressure \( p \) itself.
Conversion
from conservative to primitive variables

This is done by inverting the equations

\[ D = \rho W \]
\[ S_j = \rho h W^2 v_j \]
\[ \tau = \rho h W^2 - \rho W - p \]

in order to express \( \rho \) and \( \varepsilon \) in terms of the conserved variables and of the pressure \( p \):

\[ \rho = \frac{D}{\tau + p + D} \sqrt{(\tau + p + D)^2 - S^2} \]
\[ \varepsilon = D^{-1} \left( \sqrt{(\tau + p + D)^2 - S^2} - p \frac{\tau + p + D}{\sqrt{(\tau + p + D)^2 - S^2}} - D \right) \]

where \( S \equiv \gamma^{ij} S_i S_j \)
In *Whisky*, a Newton-Raphson root finder is used. So we need the derivative of the function with respect to the dependent variable, the pressure. Such a derivative is expressed in terms of the EoS and of the conserved variables:

$$\frac{d}{dp}\{p - \bar{p}[\rho(U, p), \varepsilon(U, p)]\} = 1 - \frac{\partial \bar{p}(\rho, \varepsilon)}{\partial \rho} \frac{\partial \rho}{\partial p} - \frac{\partial \bar{p}(\rho, \varepsilon)}{\partial \varepsilon} \frac{\partial \varepsilon}{\partial p}$$

where

$$\frac{\partial \rho}{\partial p} = \frac{DS^2}{\sqrt{(\tau + p + D)^2 - S^2(\tau + p + D)^2}}$$

$$\frac{\partial \varepsilon}{\partial p} = \frac{pS^2}{\rho[(\tau + p + D)^2 - S^2](\tau + p + D)}$$

The other derivatives are obtained from the EoS.
Artificial atmosphere

In Eulerian simulations there are usually parts of the numerical domain where vacuum should be present (e.g. outside the star). This is exacerbated in the case of simulations of compact objects (especially if gravitational waves must be computed), in which one has to set the outer boundaries rather far from the central objects.

However, in the vacuum limit the hydrodynamics equations break down, because the speed of sound tends to the speed of light and the Riemann solvers fail. In addition the conversion from conserved variables to primitive variables would have to be considered as a special case anyway.

To avoid this problem, it is customary to introduce a tenuous atmosphere, i.e. a low-density, low-pressure region surrounding the compact objects. The atmosphere is treated in Whisky as a perfect fluid with zero coordinate velocity and constant rest mass and pressure. The density of the atmosphere is set to be several orders of magnitude smaller than the maximum density of the stars. Alternative prescriptions are possible, but not implemented yet.
Artificial atmosphere

The atmosphere is set up by the initial-data routines. During each evolution step each point is checked twice, to see whether its density is below the given threshold for the atmosphere.

First, if an evolution step would bring the point in the atmosphere, the point is not evolved and flagged to be an atmosphere point.

Then, in the conversion from conservative to primitive variables, if a point is flagged or if its density is below the threshold, the values of the hydrodynamical variables at that point are set to the values of the atmosphere.

The treatment of the atmosphere is one of the most delicate parts of codes implementing HRSC methods. However, if the control of the atmosphere and the setting of its parameters are carried out with care, the atmosphere has negligible influence on the evolution of the compact objects.
Obtaining the public version of Whisky

If you want to start to have a look:

Whisky website:
http://www.whiskycode.org
Numerical Relativity with Cactus

Cactus (www.cactuscode.org) is an open-source problem-solving environment designed for scientists and engineers. Its modular structure easily enables parallel computation across different architectures. Cactus originated in the academic research community more than 10 years ago and it is now developed mainly at CCT/LSU [Center for Computation & Technology at the Louisiana State University (Baton Rouge)].

Cactus runs on many architectures. Applications, developed on standard workstations or laptops, can be seamlessly run on clusters or supercomputers (even GPU support is being thought). Cactus provides easy access to many cutting edge software technologies being developed in the academic research community, including the Globus Metacomputing Toolkit, HDF5 parallel file I/O, the PETSc scientific library, adaptive mesh refinement, web interfaces, and advanced visualisation tools.
Cactus acts as the “main” routine of your code, and can take care of e.g. parallelism, IO, checkpointing, parameter file parsing, and provides different computational infrastructure such as reduction operators, interpolators, coordinate, elliptic solvers, …

STRUCTURE

The source code of Cactus consists of:

• a core part – the “Flesh”:

  independent of all other code; it acts as a utility and service library which thorns call to get information or ask for some action to happen. The flesh also provides the scheduling.

• a set of basic modular units, called “thorns”.

  Thorns are separate libraries which encapsulate some functionality.

There are many public Cactus thorns, especially for numerical relativity.
Numerical Relativity with Cactus

Anatomy of a Thorn

- A thorn in Cactus contains:
  - Cactus declarations [in CCL (Cactus Configuration Language)]
  - source code (C, C++, Fortran)

- makefile fragments (the list of source files and other thorn-specific configurations)
- documentation
- test cases
- example parameter files
The Flesh finds out about thorns through configuration files in each thorn. These files are converted at compile time into a set of routines the Flesh can call to find out about thorns.

There are three such files:

1. **Interface definitions**

All variables which are passed between scheduled routines need to be declared.

2. **Scheduling directives**

The flesh incorporates a scheduler which is used to call defined routines from different thorns in a particular order.

3. **Parameter definitions**

The flesh and thorns are controlled by a parameter file; parameters must be declared along with their allowed values and default value.
Numerical Relativity with Cactus

interface.ccl

• Declares the
  implementation name

• Declares grid functions
  [namely the 3D variables which your thorn uses
   and wants to communicate to Cactus in order
   to get parallelisation, IO, interpolation,
   communication, checkpointing etc.]

  and other variables
  (INTEGER, REAL, COMPLEX)

• Can inherit public grid
  functions from other
  implementations

• Declares routines (APIs
  provided/used by the
  thorn)

  IMPLEMENTS: ADMConstraints
  INHERITS: ADMBase

  CCTK_REAL Hamiltonian TYPE=gf
  {
    ham
  } “Hamiltonian Constraint”

  CCTK_REAL Momentum TYPE=gf
  {
    momx momy momz
  } “Momentum Constraint”
Numerical Relativity with Cactus

**schedule.ccl**

- Tells Cactus how to call routines at certain times, e.g. *initial* or *evol* or *analysis*
- The flexible scheduler rule-based which controls the program flow: `schedule AFTER, BEFORE, WHILE`
- *Schedule groups* introduce a hierarchical structure
- Allocates memory storage for grid variables
- Synchronises variables

```fortran
SCHEDULE ADMConstraints_Calculate AT analysis {
    LANG: Fortran
    STORAGE: Hamiltonian Momentum
    SYNC: Hamiltonian Momentum
    TRIGGERS: Hamiltonian Momentum
}  “Calculate the constraints”
```
param.ccl

- Declares parameters, of five types: integer, real, boolean, keyword, string
- Allowed ranges and default value need to be declared
- Can “inherit” public parameters from other implementations, possibly extending ranges
- Parameters can be changed during the simulation for remote steering

SHARES: ADMBase

EXTENTS KEYWORD initial_data
{
   "gaussian" :: "Gaussian pulse"
}

PRIVATE:

CCTK_REAL gaussian_amplitude \n   "Amplitude"
{
   0.0::* :: "must be nonnegative"
} 1.0
Numerical Relativity with Cactus

Example Source Code

```
#include "cctk.h"
#include "cctk_Arguments.h"

subroutine ADMConstraints_calculate (CCTK_ARGUMENTS)
  implicit none
  DECLARE_CCTK_ARGUMENTS

  CCTK_REAL :: dx, dy, dz
  integer   :: i, j, k

  dx = CCTK_DELTA_SPACE(1)
  ...

  do i = 2, cctk_lsh(1)-1
    ...
    ham(i,j,k) = (gxx(i+1,j,k) - gxx(i-1,j,k)) / (2*dx)
    ...
```
Parameter Files

- At run time, parameter files activate thorns and specify parameter values
- Not all compiled thorns need to be active

ActiveThorns = “PUGH CartGrid3D ADMBase IDSimple ADMConstraints”

driver::global_nx = 101
...
grid::xmin = 0.0
grid::xmax = 30.0
...
grid::type = “octant”

ADMBase::initial_data = “Minkowski”
Numerical Relativity with Cactus

Driver

- A *driver* is a special thorn that handles memory management and parallelisation

- Two drivers exist: *PUGH* (uniform grid) and *Carpet* (AMR, multi-block)

- Two more AMR drivers in development, based on *SAMRAI* and *Paramesh*

- Interpolation, reduction, and hyperslabbing operations are closely tied to the driver

- drivers provide I/O (efficient and parallel) and checkpointing/recovery
Numerical Relativity with Cactus

Metadata and Data Preservation

- Thorn Formaline collects meta-data about a simulation (and sends them to a server)
- Collects machine name, user name, parameters, current simulation time, special events, etc.
- Allows real-time overview about currently running simulations by all people on all machines
- Some simulation results are later semi-automatically staged to be permanently stored in an archive
The Cactus Code is (C) Copyrighted by the individual Authors and distributed under the GNU General Public License.

Most of the computational packages that come with it follow this license, but the authors of any thorn are free to follow any licensing they deem appropriate as long as they state it explicitly in their thorn distribution.
Installing Cactus

Cactus is maintained using CVS (Concurrent Versions System). The preferred method is to use the GetCactus script along with a thornlist as described below. There is a page of basic thornlists on the website, or you can use your own customised thornlist.

1) Save the checkout script with the name GetCactus from http://www.cactuscode.org/download/GetCactus

2) To checkout the flesh type the below and follow the instructions:

   chmod a+x GetCactus
   ./GetCactus

3) To checkout some thorns, choose and save a ThornList, or provide one to your own needs, and issue the command:

   ./GetCactus <thornlist-name>

To add more Thorns to your Cactus, you can use the GetCactus script with a new ThornList.
Installing Cactus

Alternatively, you can use the CVS commands yourself.

- For those who know how to use CVS, perform the following:
  
  ```cvs
  cvs -d :pserver:cvs_anon@cvs.cactuscode.org:/cactusdevcvs login
  
  the password is 'anon'.
  
  Once you are logged in:
  
  ```cvs
  cvs -d :pserver:cvs_anon@cvs.cactuscode.org:/cactusdevcvs co Cactus
  
  to checkout Cactus
  
  - ‘cd arrangements’ to enter the arrangements directory. Once inside arrangements, checkout the arrangements you need using:
    
    ```cvs
    cvs -d :pserver:cvs_anon@cvs.cactuscode.org:/cactusdevcvs co <arrangement_name>
    ```
Getting help with Cactus

```sh

gmake UsersGuide

gmake ThornGuide

gmake ReferenceManual
```

runs LaTeX to produce a copy of the Users' Guide.

runs LaTeX to produce a copy of the Thorn Guide, for all the thorns in the arrangements directory.

runs LaTeX to produce a copy of the reference manual

All of this information and much more can be retrieved using:

gmake help
compiling cactus

cd Cactus
gmake <config-name>-config e.g. gmake WaveDemo-config
gmake <config-name> e.g. gmake WaveDemo

Hopefully that went OK, and you now have an executable, exe/cactus_WaveDemo.

If it has not worked, you may need to set some of the compilation parameters:

CC: The C compiler.
CXX: The C++ compiler.
F90: The Fortran 90 compiler.
F77: The Fortran 77 compiler.
CPP: The preprocessor used to generate dependencies for and to preprocess C and C++ code.
FPP: The preprocessor used to generate dependencies for and to preprocess Fortran code.
LD: The linker.
CFLAGS: Flags for the C compiler.
CXXFLAGS: Flags for the C++ compiler.
F90FLAGS: Flags for the Fortran 90 compiler.
F77FLAGS: Flags for the Fortran 77 compiler.
LDFLAGS: Flags for the linker.
DEBUG: Activates debugging.
OPTIMIZE, OPTIMISE: Specifies what type of optimisation should be used.
C_OPTIMISE_FLAGS: Optimisation flags for the C compiler.
CXX_OPTIMISE_FLAGS: Optimisation flags for the C++ compiler.
F90_OPTIMISE_FLAGS: Optimisation flags for the Fortran 90 compiler.
F77_OPTIMISE_FLAGS: Optimisation flags for the Fortran 77 compiler.

and many others, also for including external libraries as MPI, HDF5, PETSC, LAPACK, GSL, BLAS...
Testing that a code gives the expected results on any new machine is very important, as it is important to check that changes that should not affect some results, really don’t affect them … **This is especially important in codes developed collaboratively.**

So, check that the installation on your machine is OK, by running the **test-suites**, just type

```
gmake WaveDemo-testsuite
```

and use the default answers to each question.
WaveToy Demonstration

The WaveToy set of thorns perform evolutions of a 3D scalar field with the linear wave equation

\[
\frac{\partial^2 w}{\partial t^2} - a^2 \frac{\partial^2 w}{\partial x^2} = 0
\]

This application is one of the simplest examples of a solving a hyperbolic partial differential equation using finite differences, and so provides a very good learning example of how a PDE can be solved within the Cactus framework. Despite it's simplicity, the WaveToy example is prototypical of much more complicated systems of equations.

The WaveToy application is provided in each of the programming languages supported by Cactus.
Run the demo

Move the downloaded demo parameter file into the Cactus directory. To start the simulation, run your executable with the demo parameter file, if you have a single processor executable

```
./exe/cactus_WaveDemo ./par/WaveDemo.par
```

If you compiled with MPI and have a multiprocessor version, you will need to use the appropriate mpi command for running.

When the simulation starts, you will see output describing for example the activated thorns and the scheduling tree.

(see screen)
Other thorns

Thorns you may need are in the following public arrangements:

CactusBase
CactusEinstein
AEIThorns (BSSN_MoL, AHFinderDirect, EHFinder, Dissipation, Formaline, ManualTermination, ReflectionSymmetry, RotatingSymmetry90-180, ...)
CactusEOS
Whisky
CactusPUGH, CactusPUGHIO
CactusUtils
there are also others...
Numerical relativity with Carpet

Carpet (www.carpetcode.org) is a mesh-refinement driver mainly developed by E. Schnetter [CQG 21, 1465 (2004)].

Carpet follows a (simplified) Berger-Oliger [J. Comput. Phys. 53, 484 (1984)] approach to mesh refinement, that is:

- refined sub-domains consist of a set of cuboid (=rectangular parallelepiped) grids
- refined sub-domains have boundaries aligned with the grid lines
- the refinement ratio between refinement levels is constant

While the refined meshes are not automatically moving on the grid, they can be activated and deactivated during the evolution, obtaining a progressive fixed mesh refinement or even a moving-grid mesh refinement.
Consistency

Look at the interface between coarse and fine grids:

(redis: coarse grid, green: fine grid; red grid points in overlap region are not shown)

• Since coarser and finer grids overlap, they have to stay consistent
• Since finer grids are more accurate, we copy/interpolate them to the coarser grids: Restriction
• This needs to happen after every time step
**Time stepping**

Look again at the interface between coarse and fine grids:

![Diagram of time stepping]

Berger-Oliger time stepping:
1. Coarse grid time step
2. First fine grid time step
3. Second fine grid time step
4. Restrict from fine to coarse grid

(red: coarse grid, green: fine grid; red grid points in overlap region are not shown)

- The CFL criterion requires smaller time steps on finer grids
- Reducing the overall step size is possible, but expensive, so *Subcycling in Time* (i.e. perform smaller time steps on the finer grids) is usually convenient
- This happens recursively on all levels
Fine-grid boundaries

Look once more at the interface between coarse and fine grids:

![Diagram showing fine-grid boundaries](image)

(red: coarse grid, green: fine grid; red grid points in overlap region are not shown)

• The finer grids need boundary conditions
• Their boundary points are copied/interpolated from coarser grids: *Prolongation*; Lagrangian or ENO interpolations of different orders are available
• This needs to happen for every time step
Numerical relativity with **Carpet**

**Notes**

- *Proper nesting*: Each grid must be contained in the set of next coarser grids, so that the level $L$ boundaries can be prolonged from the level $L-1$ interiors

- Always check that nothing is too wrong at refinement boundaries (e.g. propagating waves may be partially reflected, the divergence of the magnetic field may not be zero,...)

- It is desirable to automate the movements of the refined grids
- Carpet is used in production, but also actively developed. Be cautious, and interact with the developers
- Replacing PUGH by Carpet should give identical results
Numerical relativity with Carpet

Scheduling modes

Each scheduled routine can operate in one of three modes, selected by an OPTIONS statement in the schedule:

*local*: The default. The routine has access to a single grid block, e.g. it evaluates the right hand side

*level*: Access to a whole refinement level at once, but not to individual blocks (e.g. for applying a boundary condition)

*global*: Access to the whole grid hierarchy, but not to individual level (e.g. for finding apparent horizons)
Numerical relativity with Carpet

Getting carpet

- Carpet’s home page is www.carpetcode.org
- There is a mailing list developers@lists.carpetcode.org with public archives
- There is an (old) tutorial and documentation available from the web site
- You can get from the webpage the last stable release with ‘darcs’
- Carpet requires an MPI installation
- There is a WaveToy for Carpet, for practicing, in carpet-stable-3/CarpetExtra/
1. In a few days (or at most weeks), simulate your favourite compact object, vanilla flavour...

2. In a month or so, implement the additional physics you want to include...

3. In 3-6 months, perform complete simulations with various parameters, both numerical (check the consistency/convergence) and physical (masses, spins, ...)

4. Publish the results in one or more articles...

5. If possible, share your additional implementations with the community!