An introduction to the Einstein Toolkit

Miguel Zilhão

Center for Computational Relativity and Gravitation
Rochester Institute of Technology

March 2013, NRHEP2, Instituto Superior Técnico, Lisboa

Contents

Outline

Lecture goals

- introduction to the Einstein Toolkit (ET)
- its history and capabilities
- when you should consider using it
- where to get more information
- examples: consider installing VirtualBox, for Wednesday's tutorial session: `https://www.virtualbox.org/`

Why Cactus/ET

BSSN equations:

$$\partial_t \tilde{\gamma}_{ij} = \beta^k \partial_k \tilde{\gamma}_{ij} + 2\tilde{\gamma}_{k(i}\partial_{j)}\beta^k - \frac{2}{3}\tilde{\gamma}_{ij}\partial_k \beta^k - 2\alpha \tilde{A}_{ij},$$

$$\partial_t \chi = \beta^k \partial_k \chi + \frac{2}{3}\chi(\alpha K - \partial_k \beta^k),$$

$$\partial_t \tilde{A}_{ij} = \beta^k \partial_k \tilde{A}_{ij} + 2\tilde{A}_{k(i}\partial_{j)}\beta^k - \frac{2}{3}\tilde{A}_{ij}\partial_k \beta^k + \chi\left(\alpha R_{ij} - \nabla_i \partial_j \alpha\right)^{\mathrm{TF}}$$
$$+ \alpha\left(K\,\tilde{A}_{ij} - 2\tilde{A}_i{}^k \tilde{A}_{kj}\right) - 8\pi\alpha\left(\chi S_{ij} - \frac{S}{3}\tilde{\gamma}_{ij}\right),$$

$$\partial_t K = \beta^k \partial_k K - \nabla^k \partial_k \alpha + \alpha\left(\tilde{A}^{ij}\tilde{A}_{ij} + \frac{1}{3}K^2\right) + 4\pi\alpha(E + S),$$

$$\partial_t \tilde{\Gamma}^i = \beta^k \partial_k \tilde{\Gamma}^i - \tilde{\Gamma}^k \partial_k \beta^i + \frac{2}{3}\tilde{\Gamma}^i \partial_k \beta^k + 2\alpha \tilde{\Gamma}^i_{jk}\tilde{A}^{jk} + \frac{1}{3}\tilde{\gamma}^{ij}\partial_j \partial_k \beta^k$$
$$+ \tilde{\gamma}^{jk}\partial_j \partial_k \beta^i - \frac{4}{3}\alpha \tilde{\gamma}^{ij}\partial_j K - \tilde{A}^{ij}\left(3\alpha\chi^{-1}\partial_j \chi + 2\partial_j \alpha\right) - 16\pi\alpha\chi^{-1} j^i$$

Why Cactus/ET

Typical problem in Numerical Relativity...

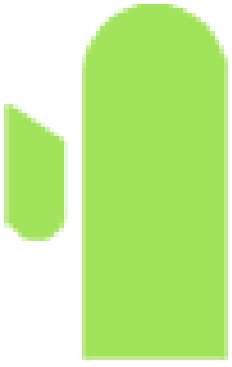
- mesh refinement
- efficiently parallelize
- large input/output
- somewhat complex tools for analysis

Typical workflow:

1. Compute initial data
2. Evolve equations
3. Analysis

What is Cactus

Cactus:



- general framework for the development of portable, modular applications
- programs are split into independent components (thorns)
- thorns are developed independently and should be interchangeable with others with same functionality
- thorns don't directly interact with each other
- Cactus framework (flesh) provides the "glue"
- supports C, C++, Fortran

Cactus history

- Direct descendant of many years of code development in Ed Seidel's group of researchers at NCSA
- 1995, Paul Walker, Joan Masso, Ed Seidel, and John Shalf: Cactus 1.0
- Originally for numerical relativity
- Over the years generalized for use by scientists in other domains

What is ET

ET $\approx$ Cactus $+$ collection of thorns and tools for numerical relativity:

- initial data
- vacuum spacetime solver
- hydrodynamics solver
- analysis tools (apparent and event horizon finder, wave extraction, ...)
- ...

ET history

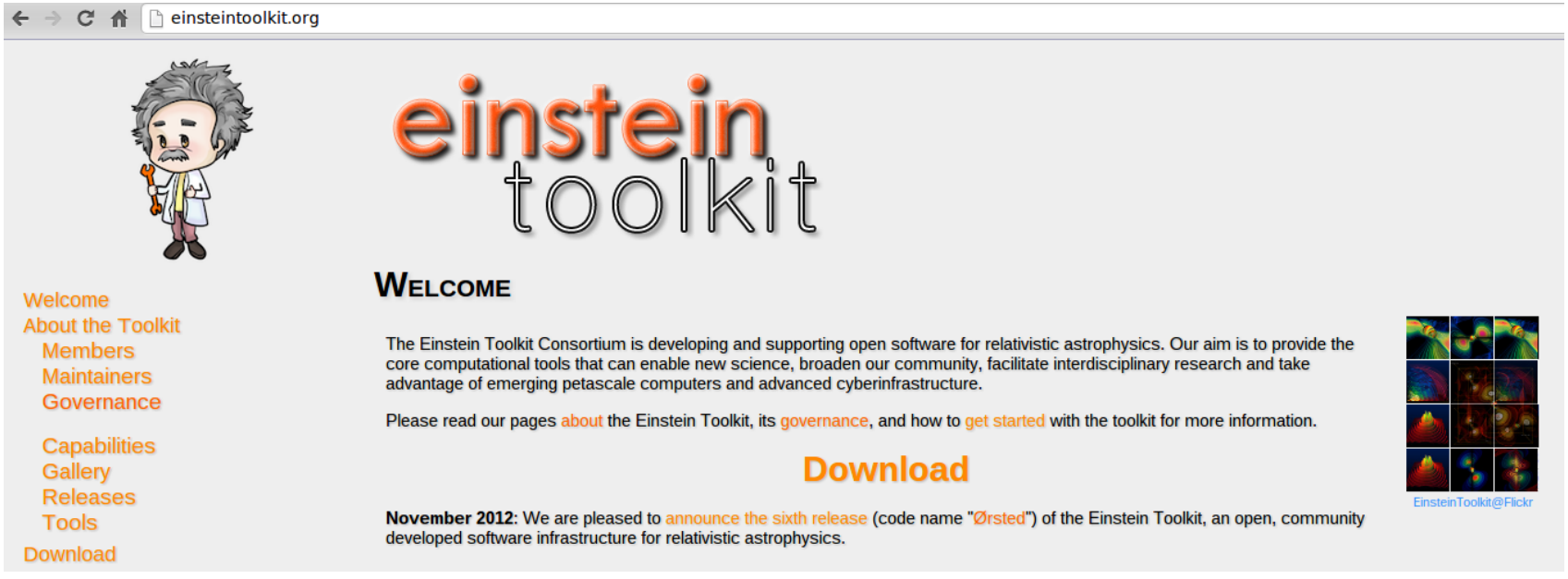
- First version (Bohr) released 2010-06-17
- New version released $\approx$ every 6 months
- Last stable version (Ørsted) released 2012-11-08
- About 50 contributors over the past decade, currently 9 maintainers from 6 different institutions

ET goals

- Separating **Physics** from **Computational Science**
- Providing computational tools which are
  - maintained
  - of high quality
  - easy to use
  - open source
- User can focus on science

Outline

Obtaining ET



Obtaining ET

einstein toolkit

Download

The Einstein Toolkit is hosted on many different machines around the world. We provide a script called GetComponents to simplify downloading the toolkit. This page just describes how to download the toolkit - you may also be interested in the Tutorial for New Users which leads you through these steps and more on the Queen Bee supercomputer.

Enter the directory on your machine in which you would like to download the ET (for example, your home directory), and type the commands listed below. This will create a directory called Cactus in which the components of the Einstein Toolkit are downloaded.

Current release: Ørsted (released on November 8th, 2012)

This is the recommended version of the toolkit for most users. See the release notes for more information.

```
curl -O https://raw.github.com/gridaphobe/CRL/master/GetComponents

chmod a+x GetComponents

./GetComponents --parallel https://svn.einsteintoolkit.org/manifest/branches/ET_2012_11/einsteintoolkit.th
```

ET contents: arrangements

Provided arrangements ($\approx$ collection of thorns):

- Several Cactus thorns (I/O, Method of Lines, ...)
- Carpet (Adaptive Mesh Refinement driver)
- EinsteinBase
- EinsteinInitial
- EinsteinEvolve
- EinsteinAnalysis
- McLachlan (BSSN implementation)
- ...

ET contents: tools

Provided tools:

- GetComponents
- Simfactory
- Formaline

Cactus arrangements

Main core Cactus arrangements:

CactusBase
: Infrastructure thorns for boundary conditions, coordinates, IO, symmetries and time

CactusNumerical
: Numerical infrastructure thorns: time integration, dissipation, symmetry boundary conditions, spherical surfaces, local interpolation, Method of Lines (MoL), ...

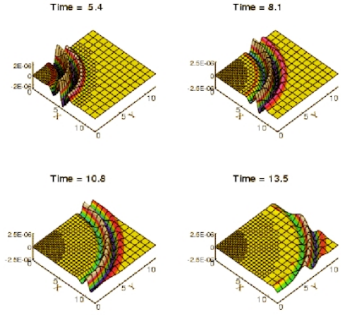
CactusUtils
: Utility thorns: formaline, nan-checking, termination triggering and timer reports

ExternalLibraries
: Provides external libraries: Lapack, GSL, HDF5, FFTW, Lorene (for initial data) and others

Carpet

Berger-Oliger Adaptive Mesh Refinement (AMR) driver:

- Splits grid functions and arrays among the MPI processes
- Setups mesh refinement grid hierarchy
- Communicates ghost cell information between MPI processes
- Communicates between refinement levels by prolongation and restriction
- Modifies grid hierarchy (regridding) when requested
- Performs parallel IO



ET Arrangements: EinsteinBase

Main thorns:

- ADMBase (defines spacetime variables)
- HydroBase (defines hydro variables)
- TmunuBase (defines “right-hand-side”)
- EOS_Base (EOS registration)
- EOSG_Base (interface to generic EOS’)

EinsteinBase: ADMBase

Holds everything else together:

1. Defines groups of grid functions for basic variables:
   - metric (`gxx, gxy, gxz, gyy, gyz, gzz`)
   - extrinsic curvature (`kxx, kxy, kxz, kyy, kyz, kzz`)
   - lapse (`alp`), shift (`betax, betay, betaz`)
   - time derivative of lapse (`dtalp`) and time derivative of shift (`dtbetax, dtbetay, dtbetaz`)
2. Defines basic parameters to choose the initial data, evolution method and number of active timelevels that other thorns can use or extend where appropriate.
3. Defines a schedule group for other thorns to schedule their routines modifying the ADMBase variables.

EinsteinBase: HydroBase

Basic variables for hydro evolutions:

- `rho`: rest mass density ρ
- `press`: pressure P
- `eps`: Specific internal energy ϵ
- `vel[3]`: contravariant fluid three velocity v^i
- `w_lorentz`: Lorentz factor γ
- `Y_e`: electron fraction Y_e
- `temperature`: temperature T
- `entropy`: specific entropy per particle s
- `Bvec[3]`: contravariant magnetic field vector B^i

Also sets up scheduling groups for recommended interaction with the fluid variables

EinsteinBase: TmunuBase

Defines grid functions for stress-energy tensor:

- The time component T_{00}: `eTtt`
- The mixed components T_{0i}: `eTtx`, `eTty`, `eTtz`
- The spatial components T_{ij}: `eTxx`, `eTxy`, `eTxz`, `eTyy`, `eTyz`, `eTzz`

Also sets up scheduling groups for other thorns to schedule routines that adds to the stress-energy tensor

EinsteinInitial

TwoPunctures
: Puncture binary black hole initial data

TOVSolver
: Single TOV star

GRHydro_InitData
: Test initial data for hydro evolutions

...

EinsteinEvolve

McLachlan
: Spacetime evolution

GRHydro
: GRMHD matter evolution

- both designed to interoperate through the ADMBase and TmunuBase interface

PunctureTracker
: Black hole tracking (allowing Carpet to accordingly move and refine grids)

NewRad
: Implements radiative boundary conditions

EinsteinAnalysis

AHFinderDirect
: Find black hole apparent horizons (quickly)

AHfinder
: Find black hole apparent horizons (slowly)

WeylScal4
: Calculate the Newman-Penrose scalar Ψ_4

Multipole
: Decompose arbitrary grid functions into spin-weighted spherical harmonics

ADMAnalysis
: Calculate several quantitites from the ADM variables

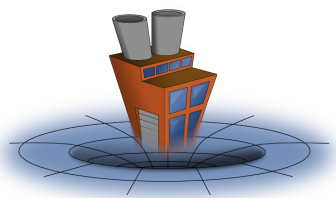
EHFinder
: Find event horizons in numerical spacetimes

ADMConstraints
: Calculate the ADM constraints from the ADM variables

Tools: GetComponents

- Different thorns are hosted at different repositories (with different version control systems)
- GetComponents script provides unified way of assembling all of these

Tools: Simfactory



- Access remote systems, synchronise source code trees
- Configure and build on different systems semi-automatically
- Provide maintained list of supercomputer configurations
- Manage simulations (follow “best practices”, avoid human errors)

Tools: Formaline

- Ensures that simulations are and remain repeatable, remember exactly how they were performed
- Take snapshots of source code, system configuration; store it in executable and/or git repository
- Tag all output files

More information

http://einsteintoolkit.org/
http://einsteintoolkit.org/documentation/
http://einsteintoolkit.org/community/support/
https://docs.einsteintoolkit.org/et-docs/Tutorial_for_New_Users

Outline

Outline

WaveMoL Thorn: wave equation

$$\partial_t^2 \phi = \partial_{x^i}^2 \phi^i$$

To illustrate usage of Cactus Method of Lines (MoL) thorn, we rewrite the equations in first order form:

$$\begin{aligned} \partial_t \Phi &= \partial_{x^i} \Pi^i, \\ \partial_t \Pi^j &= \partial_{x^j} \Phi, \\ \partial_t \phi &= \Phi, \\ \partial_{x^j} \phi &= \Pi^j. \end{aligned}$$

The first three equations (five separate PDEs) will be evolved. The final equation is used to set the initial data and can be thought of as a constraint.

WaveMoL Thorn

Directory structure:

```
WaveMoL/
|-- COPYRIGHT
|-- README
|-- doc
|-- configuration.ccl
|-- interface.ccl
|-- schedule.ccl
|-- param.ccl
`-- src
    |-- InitSymBound.c
    |-- Startup.c
    |-- WaveMoLRegister.c
    |-- WaveMoL.c
    `-- make.code.defn
```

WaveMoL Thorn

Directory structure:

```
WaveMoL/
|-- COPYRIGHT
|-- README
|-- doc
|-- configuration.ccl
|-- interface.ccl
|-- schedule.ccl
|-- param.ccl
`-- src
    |-- InitSymBound.c
    |-- Startup.c
    |-- WaveMoLRegister.c
    |-- WaveMoL.c
    `-- make.code.defn
```

WaveMoL Thorn

- `interface.ccl`:

Name of thorn as seen by other thorns

```
implements: wavemol

USES INCLUDE: Symmetry.h
```

Functions used from other thorns:

```
CCTK_INT FUNCTION MoLRegisterEvolvedGroup(CCTK_INT IN EvolvedIndex, \
                                          CCTK_INT IN RHSIndex)
CCTK_INT FUNCTION MoLRegisterConstrained(CCTK_INT IN ConstrainedIndex)

REQUIRES FUNCTION MoLRegisterEvolvedGroup
REQUIRES FUNCTION MoLRegisterConstrained

CCTK_INT FUNCTION Boundary_SelectGroupForBC(CCTK_POINTER_TO_CONST IN GH, \
  CCTK_INT IN faces, CCTK_INT IN boundary_width, CCTK_INT IN table_handle, \
  CCTK_STRING IN var_name, CCTK_STRING IN bc_name)
REQUIRES FUNCTION Boundary_SelectGroupForBC

public

cctk_real scalarevolvemol_scalar type = GF Timelevels = 3 tags='tensortypealias="Scalar
{
  phi
  phit
} "The scalar field and time derivative"

cctk_real scalarevolvemol_vector type = GF Timelevels = 3 tags='tensortypealias="U"'
{
  phix
  phiy
  phiz
} "The scalar field spatial derivatives"
```

WaveMoL Thorn

- `interface.ccl` (cont.):

```
cctk_real scalarrhsmol_scalar type = GF Timelevels = 1
{
  phirhs
  phitrhs
} "The right hand side for the scalar field"

cctk_real scalarrhsmol_vector type = GF Timelevels = 1
{
  phixrhs
  phiyrhs
  phizrhs
} "The right hand side for the scalar field derivatives"
```

WaveMoL Thorn

- `schedule.ccl`:

```
STORAGE: scalarevolvemol_scalar[3], scalarevolvemol_vector[3]
STORAGE: scalarrhsmol_scalar, scalarrhsmol_vector
STORAGE: energy

schedule WaveMoL_Startup at STARTUP
{
  LANG: C
} "Register Banner"

schedule WaveMoL_InitSymBound at BASEGRID
{
  LANG: C
  OPTIONS: META
} "Schedule symmetries"

schedule WaveMoL_RegisterVars in MoL_Register
{
  LANG: C
  OPTIONS: META
} "Register variables for MoL"

schedule WaveMoL_CalcRHS in MoL_CalcRHS
{
  LANG: C
} "Register RHS calculation for MoL"
```

WaveMoL Thorn

- `schedule.ccl` (cont.):

```
schedule WaveMoL_Energy in MoL_PostStep
{
  LANG: C
} "Calculate the energy"

schedule WaveMoL_Energy at POSTINITIAL
{
  LANG: C
} "Calculate the energy"

schedule WaveMoL_Boundaries in MoL_PostStep
{
  LANG: C
  OPTIONS: LEVEL
  SYNC: scalarevolvemol_scalar
  SYNC: scalarevolvemol_vector
} "Register boundary enforcement in MoL"

schedule GROUP ApplyBCs as WaveMoL_ApplyBCs in MoL_PostStep after WaveMoL_Boundaries
{
} "Apply boundary conditions for WaveMoL"
```

WaveMoL Thorn

- param.ccl:

```
shares: MethodOfLines

USES CCTK_INT MoL_Num_Evolved_Vars
USES CCTK_INT MoL_Num_Constrained_Vars
USES CCTK_INT MoL_Num_SaveAndRestore_Vars

restricted:

CCTK_INT WaveMoL_MaxNumEvolvedVars "The maximum number of evolved variables used by Wave
{
  5:5 :: "Just 5: phi and the four derivatives"
} 5

CCTK_INT WaveMoL_MaxNumConstrainedVars "The maximum number of constrained variables used
{
  1:1 :: "The energy"
} 1

private:

KEYWORD bound "Type of boundary condition to use"
{
  "none"      :: "No boundary condition"
  "flat"      :: "Flat boundary condition"
  "radiation" :: "Radiation boundary condition"
} "none"
```

WaveMoL Thorn

Directory structure:

```
WaveMoL/
|-- COPYRIGHT
|-- README
|-- doc
|-- configuration.ccl
|-- interface.ccl
|-- schedule.ccl
|-- param.ccl
`-- src
    |-- InitSymBound.c
    |-- Startup.c
    |-- WaveMoLRegister.c
    |-- WaveMoL.c
    `-- make.code.defn
```

WaveMoL Thorn

- `WaveMoL_RegisterVars`:

```
void WaveMoL_RegisterVars(CCTK_ARGUMENTS) {

  DECLARE_CCTK_ARGUMENTS;
  DECLARE_CCTK_PARAMETERS;

  CCTK_INT ierr = 0, group, rhs, var;

  group = CCTK_GroupIndex("wavemol::scalarevolvemol_scalar");
  rhs = CCTK_GroupIndex("wavemol::scalarrhsmol_scalar");

  ierr += MoLRegisterEvolvedGroup(group, rhs);

  group = CCTK_GroupIndex("wavemol::scalarevolvemol_vector");
  rhs = CCTK_GroupIndex("wavemol::scalarrhsmol_vector");

  ierr += MoLRegisterEvolvedGroup(group, rhs);

  var = CCTK_VarIndex("wavemol::energy");

  ierr += MoLRegisterConstrained(var);
}
```

WaveMoL Thorn

- `WaveMoL_CalcRHS`:

```
void  WaveMoL_CalcRHS(CCTK_ARGUMENTS) {

  DECLARE_CCTK_ARGUMENTS;

  int i,j,k, index, istart, jstart, kstart, iend, jend, kend;
  CCTK_REAL dx,dy,dz, hdxi, hdyi, hdzi;

  /* Set up shorthands */
  dx = CCTK_DELTA_SPACE(0); dy = CCTK_DELTA_SPACE(1); dz = CCTK_DELTA_SPACE(2);

  hdxi = 0.5 / dx; hdyi = 0.5 / dy; hdzi = 0.5 / dz;

  istart = 1; jstart = 1; kstart = 1;

  iend = cctk_lsh[0]-1; jend = cctk_lsh[1]-1; kend = cctk_lsh[2]-1;

  /* Calculate the right hand sides. */
  for (k=0; k<cctk_lsh[2]; k++) {
    for (j=0; j<cctk_lsh[1]; j++) {
      for (i=0; i<cctk_lsh[0]; i++) {
        index = CCTK_GFINDEX3D(cctkGH,i,j,k);
        phirhs[index] = phit[index];
        phitrhs[index] = 0;
        phixrhs[index] = 0;
        phiyrhs[index] = 0;
        phizrhs[index] = 0;
      }
    }
  }
```

WaveMoL Thorn

- `WaveMoL_CalcRHS` (cont.) :

```
  for (k=kstart; k<kend; k++) {
    for (j=jstart; j<jend; j++) {
      for (i=istart; i<iend; i++) {
        index = CCTK_GFINDEX3D(cctkGH,i,j,k);

        phitrhs[index] =
            (phix[CCTK_GFINDEX3D(cctkGH, i+1, j, k)] -
             phix[CCTK_GFINDEX3D(cctkGH, i-1, j, k)]) *hdxi
          + (phiy[CCTK_GFINDEX3D(cctkGH, i, j+1, k)] -
             phiy[CCTK_GFINDEX3D(cctkGH, i, j-1, k)]) *hdyi
          + (phiz[CCTK_GFINDEX3D(cctkGH, i, j, k+1)] -
             phiz[CCTK_GFINDEX3D(cctkGH, i, j, k-1)]) *hdzi;
        phixrhs[index] = (phit[CCTK_GFINDEX3D(cctkGH, i+1, j, k)] -
                          phit[CCTK_GFINDEX3D(cctkGH, i-1, j, k)]) *hdxi;
        phiyrhs[index] = (phit[CCTK_GFINDEX3D(cctkGH, i, j+1, k)] -
                          phit[CCTK_GFINDEX3D(cctkGH, i, j-1, k)]) *hdyi;
        phizrhs[index] = (phit[CCTK_GFINDEX3D(cctkGH, i, j, k+1)] -
                          phit[CCTK_GFINDEX3D(cctkGH, i, j, k-1)]) *hdzi;

      }
    }
  }

}
```

WaveMoL Thorn

- Example parameter file:

```
ActiveThorns = "CoordBase localreduce SymBase NaNChecker PUGHReduce CartGrid3D PUGH"
ActiveThorns = "Boundary IOBasic IOUtil IOASCII IDWaveMoL PUGHSlab WaveMoL Time MoL"

idwavemol::initial_data = "gaussian"
wavemol::bound = "radiation"

grid::domain = "full"
grid::type = "byspacing"
grid::avoid_origin = "no"
driver::global_nx = 51
driver::global_ny = 51
driver::global_nz = 51
grid::dxyz = 0.02
driver::ghost_size = 1
time::dtfac = 0.5

cactus::cctk_itlast = 100

iobasic::outScalar_every = 1
iobasic::outScalar_vars = "wavemol::phi"

iobasic::outInfo_every = 1
iobasic::outInfo_vars = "wavemol::phi"

ioascii::out1D_every = 2
ioascii::out1D_vars = "wavemol::scalarevolvemol_scalar wavemol::energy"

# Method of Lines (MoL)
methodoflines::ode_method = "icn"
```

Outline

Outline

Final tips and remarks

- Check available thorns; one may already be doing what you are looking for (or may be easily adapted/extended)
- Copy provided parameter files and adapt from there
- Drawbacks: relying on 3rd party tools...
- Use bug tracking system (`https://trac.einsteintoolkit.org/wiki`) for bugs, issues or wish list
- Email questions to users mailing list (`users@einsteintoolkit.org`)