Tutorial: introduction to the Einstein Toolkit

Oleg Korobkin

AlbaNova / Oscar Klein Center
Stockholm University
Einstein Toolkit 2015 Workshop

August 11, 2015

What is Cactus?

Cactus is:

- a framework for developing portable, modular applications
- focusing on high-performance simulation codes
- designed to allow experts in different fields to develop modules based upon their experience and to use modules developed by experts in other fields with minimal knowledge of the internals or operation of the other modules

Framework - no prebuild executable

Cactus:

- does not provide executable files
- provides infrastructure to create executables

Why?

- Problemspecific code not part of Cactus
- System libraries different on different systems
- Cactus is free software, but often problemspecific codes are not $\rightarrow$ non-distributable binary

What is Cactus for?

Solving computational problems which are:

- Too large for single machine
- Require parallelization (MPI, OpenMP, GPU?)
- Involve multi-physics
- Use eclectic/legacy code
- Use code written in different programming languages
- Take advantage of distributed development

Cactus Goals

- Portability:
  - can be installed on a variety of machines (with UNIX-family OS);
  - configurable for maximum performance.
- Modularity:
  - functionality is split up between modules ("thorns") wrapped in meaningful high-level interfaces for inter-module communication;
  - *encapsulation*: old/messy internal code of the thorn is not interfering with other thorns;
  - *multiple inheritance*: thorns can be specified as extensions of the functionality of multiple other thorns;
  - *polymorphism*: interchangeable thorns can implement the same functionality.
- Ease of use:
  - good documentation;
  - allowing users program the way they like;
  - support major (HPC) programming languages: C/C++ and Fortran.

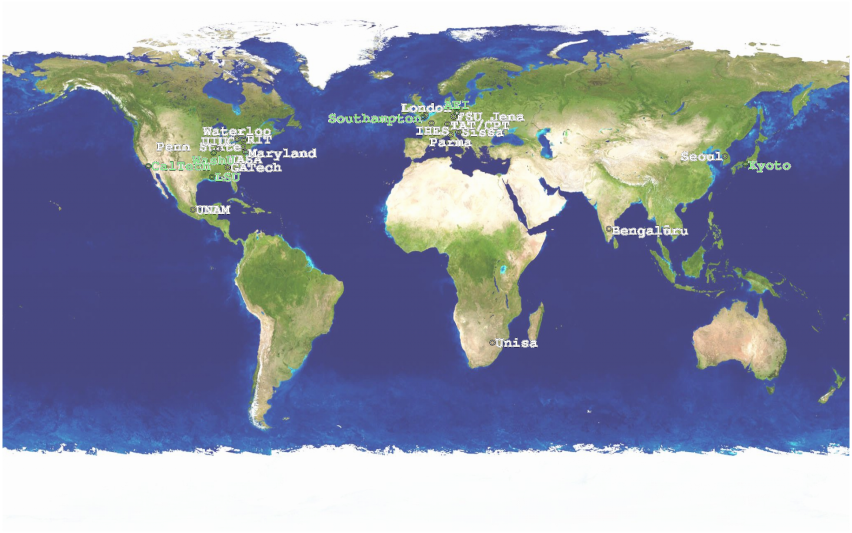
Philosophy

- Open code base to encourage community contributions
- Strict quality control for base framework
- Development always driven by real user requirements
- Support and develop for a wide range of application domains

History

- Cactus is a direct descendant of many years of code development in Ed Seidel's group of researchers at the National Center for Supercomputing Applications (NCSA) as the group wrestled to numerically solve Einstein's Equations for General Relativity and thus model black holes, neutron stars, and boson stars.
- In 1995, Ed Seidel and many of his group moved to the Max Planck Institute for Gravitational Physics (Albert Einstein Institute or AEI) in Potsdam, Germany. Frustrated by the difficulties of coordinating the development and use of several different codes across a large collaborative group, Paul Walker, Joan Massó, Ed Seidel, and John Shalf, designed and wrote the original version of Cactus, Cactus 1.0, which provided a collaborative parallel toolkit for numerical relativity.

Current Users and Developers



Covers

COMMUNICATIONS
HIGH-PERFORMANCE NETWORKING
The Future of Theoretical Physics and Cosmology
MaxPlanckResearch
Sun, Man and Climate
Invisible U
National Science Foundation
Grav
from the grou
NATIONAL SCIENCE BOARD
SCIENCE AND ENGINEERING INDICATORS 2006
VOLUME 1
Visualization and Processing of Tensor Fields
Springer
Chapman & Hall/CRC Computational Science Series
PETASCALE COMPUTING
ALGORITHMS AND APPLICATIONS
EDITED BY DAVID A. BADER
Chapman & Hall/CRC
Naturwissenschaftliche Rundschau
NR 689
11

Cactus Awards

IEEE Sidney Fernback Award	2006
High-Performance Bandwidth Challenge	SC2002
High-Performance Computing Challenge	SC2002
Gordon Bell Prize for Supercomputing	SC2001
HPC “Most Stellar” Challenge Award	SC1998
Heinz Billing Prize for Scientific Computing	1998

Cactus Structure

Structure Overview

Two fundamental parts:

- The Flesh
  - The core part of Cactus
  - Independent of other parts of Cactus
  - Acts as utility and service library

Structure Overview

Two fundamental parts:

- The Flesh
  - The core part of Cactus
  - Independent of other parts of Cactus
  - Acts as utility and service library
- The Thorns
  - Separate libraries (modules) which encapsulate the implementation of some functionality
  - Can specify dependencies on other implementations

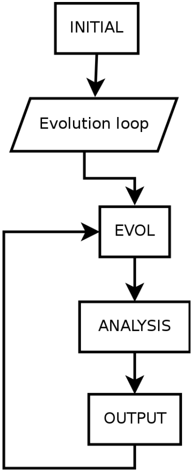
The Flesh

- The **flesh** is the core part of Cactus. It interfaces with modular components called **thorns**. The flesh provides:
  - Make system
  - Language-independent data types
  - Definition of variables
  - Access to simulation parameters
  - Rule-based schedule
  - Basic utility functions
  - Error handling
  - Basic info output

The Flesh

- The **flesh** is the core part of Cactus. It interfaces with modular components called **thorns**. The flesh provides:
  - Make system
  - Language-independent data types
  - Definition of variables
  - Access to simulation parameters
  - **Rule-based schedule**
  - Basic utility functions
  - Error handling
  - Basic info output

Rule-based Schedule



Basic schedule bins:

- STARTUP
- PARAMCHECK: check parameters consistency;
- INITIAL: set up initial data;
- CHECKPOINT: write simultaion checkpoint;
- RECOVER: recover from a checkpoint;
- PRESTEP, EVOL, POSTSTEP: evolution steps;
- ANALYSIS: periodic analysis and output;
- TERMINATE: clean-up phase.

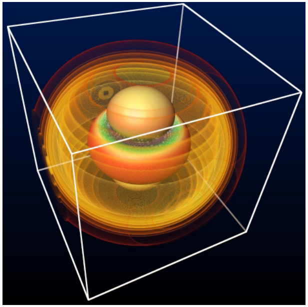
Basic utilities toolkit

Core modules (thorns) providing many basic utilities:

- I/O methods
- Boundary conditions
- Parallel unigrid driver
- Reduction and interpolation operators
- Interface to external elliptic solvers
- Web-based interaction and monitoring interface
- Simple example thorns (wavetoy)

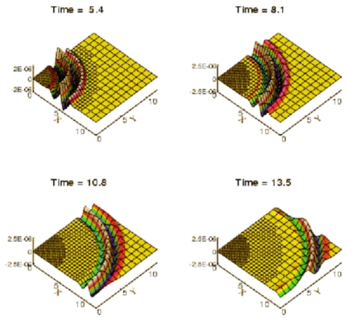
I/O Capabilities

Usual I/O and checkpointing in different formats:



- Screen output
- ASCII file output
- HDF5 file in-/output
- Online Jpeg rendering
- Runtime visualization with VisIt

Grids, boundaries, symmetries



Grids:

- Only structured meshes (for now)
- Basic: unigrid
- Advanced: FMR, AMR, multiblock

Boundary conditions / Symmetries:

- BCs: static, periodic, penalty
- Symmetries: mirror, rotational

The Driver

- Special thorn
- Only one active for a run, choose at starttime
- The only code (almost) dealing with parallelism
- Other thorns access parallelism via API
- Underlying parallel layer transparent to application thorns
- Examples
  - PUGH: unigrid, part of the Cactus computational toolkit
  - Carpet: mesh refinement, `http://www.carpetcode.org`

Grid functions

Cactus provides methods to

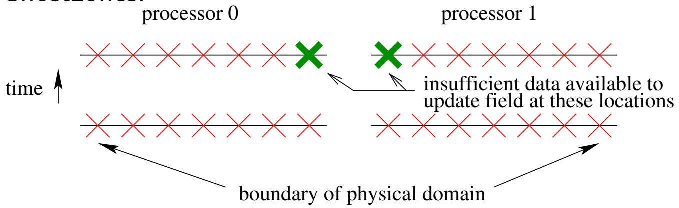
- Distribute variables across processes (grid function)
- Synchronize processor domain boundaries between processes
- Compute reductions across grid functions
- Actual implementation in driver thorn

Ghost zones

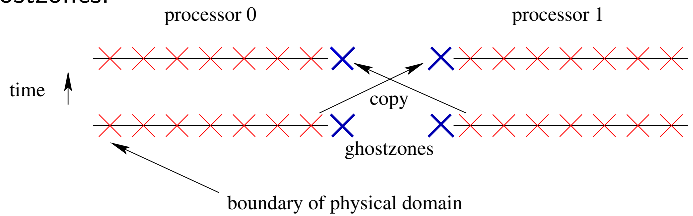
- Grid variables: distributed across processes
- Assumption: Most work done (quasi-) locally:
  True for hyperbolic and parabolic problems
- Split of computational domain into blocks
- Only communication at the borders (faces)
- At least stencil size many ghostzones

Ghost zone example

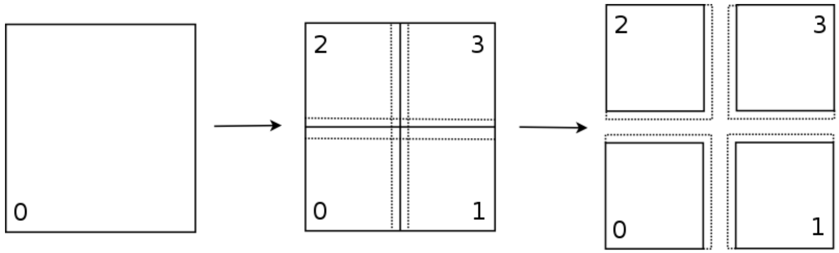
Without Ghostzones:


processor 0
processor 1
time
insufficient data available to update field at these locations
boundary of physical domain

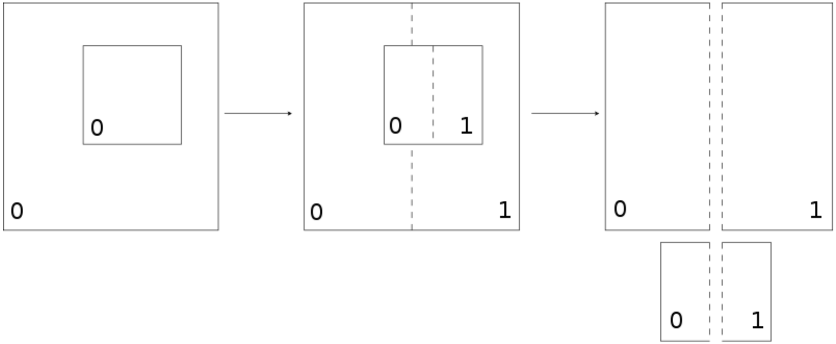
With Ghostzones:



Domain decomposition

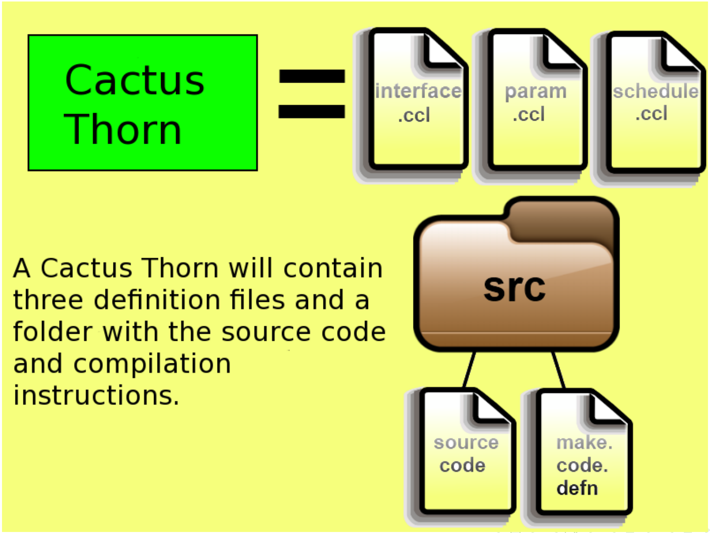


Mesh refinement decomposition



Cactus Thorn

Inside view of a plug-in module, or thorn for Cactus



A Cactus Thorn will contain three definition files and a folder with the source code and compilation instructions.

Thorn Specification

Three configuration files per thorn:

- interface.ccl declares:
  - name of the interface that the thorn implements
  - inheritance relations to other interfaces
  - thorn variables: grid functions and global scalars
  - custom functions, used or provided by the thorn

Thorn Specification

Three configuration files per thorn:

- interface.ccl declares:
  - name of the interface that the thorn implements
  - inheritance relations to other interfaces
  - thorn variables: grid functions and global scalars
  - custom functions, used or provided by the thorn
- schedule.ccl declares:
  - when each subroutine in the thorn has to be scheduled
  - when variables should be allocated/freed
  - when variables should be syncronized
  - new schedule bins in the *schedule tree*

Thorn Specification

Three configuration files per thorn:

- interface.ccl declares:
  - name of the interface that the thorn implements
  - inheritance relations to other interfaces
  - thorn variables: grid functions and global scalars
  - custom functions, used or provided by the thorn
- schedule.ccl declares:
  - when each subroutine in the thorn has to be scheduled
  - when variables should be allocated/freed
  - when variables should be syncronized
  - new schedule bins in the *schedule tree*
- param.ccl declares:
  - runtime parameters for the thorn
  - use/extension of parameters of other thorns

Example interface.ccl

```
IMPLEMENTS: wavetoy

INHERITS: grid

PUBLIC:

REAL scalarevolve TYPE=gf TIMELEVELS=3
{
  phi
} "The evolved scalar field"
```

Example schedule.ccl

```
SCHEDULE WaveToyC_Evolution AT evol
{
  LANG: C
} "Evolution of 3D wave equation"

SCHEDULE GROUP WaveToy_Boundaries AT evol \
               AFTER WaveToyC_Evolution
{
} "Boundaries of 3D wave equation"

STORAGE: scalarevolve[3]
```

Example param.ccl

```
SHARES: grid
USES KEYWORD type

PRIVATE:
KEYWORD initial_data "Type of initial data"
{
    "plane"    :: "Plane wave"
    "gaussian" :: "Gaussian wave"
} "gaussian"

REAL radius "The radius of the gaussian wave"
{
    0:* :: "Positive"
} 0.0
```

Examples

Hello World, Standalone

Standalone in C:

```
#include <stdio.h>
int main(void)
{
  printf("Hello World!\n");
  return 0;
}
```

Hello World Thorn

- `interface.ccl`:

```
implements: HelloWorld
```

- `schedule.ccl`:

```
schedule HelloWorld at CCTK_EVOL
{
  LANG: C
} "Print Hello World message"
```

- `param.ccl`: empty

Hello World Thorn cont.

- `src/HelloWorld.c`:

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

void HelloWorld(CCTK_ARGUMENTS)
{
  DECLARE_CCTK_ARGUMENTS
  CCTK_INFO("Hello World!");
  return;
}
```

- `make.code.defn`:

```
SRCS = HelloWorld.c
```

Hello World Thorn

- parameter file:

```
ActiveThorns = "HelloWorld"
Cactus::cctk_itlast = 10
```

- run: `[mpirun] <cactus executable> <parameter file>`

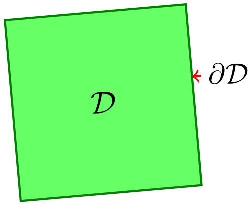
Hello World Thorn

- Screen output:

```
         10
  1    0101       ************************
  01   1010 10      The Cactus Code V4.0
 1010 1101 011      www.cactuscode.org
  1001 100101     ************************
    00010101
     100011      (c) Copyright The Authors
      0100       GNU Licensed. No Warranty
      0101

Cactus version:    4.0.b17
Compile date:      May 06 2009 (13:15:01)
Run date:          May 06 2009 (13:15:54-0500)
[...]

Activating thorn Cactus...Success -> active implementation Cactus
Activation requested for
--->HelloWorld<---
Activating thorn HelloWorld...Success -> active implementation HelloWorld
--------------------------------------------------------------------------------
INFO (HelloWorld): Hello World!
INFO (HelloWorld): Hello World!
[...] 8x
--------------------------------------------------------------------------------
Done.
```

WaveToy Thorn: Wave Equation

For a given source function $S(x, y, z, t)$ find a scalar wave field $\varphi(x, y, z, t)$ inside the domain $\mathcal{D}$ with a boundary condition:



- inside $\mathcal{D}$:

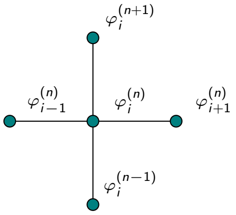
$$\frac{\partial^2\varphi}{\partial t^2} = c^2\Delta\varphi + S$$

- on the boundary $\partial\mathcal{D}$:

$$\varphi|_{\partial\mathcal{D}} = \varphi(t = 0)$$

WaveToy Thorn: Discretization

Discretization:
approximating continuous function $\varphi(x, t)$ with a grid function $\varphi_i^{(n)}$:



$$\frac{\partial^2\varphi}{\partial t^2} = c^2(\partial_x^2\varphi) + S$$
$$\Downarrow (c \equiv 1)$$
$$\frac{\varphi_i^{(n+1)} - 2\varphi_i^{(n)} + \varphi_i^{(n-1)}}{2\Delta t^2} = \frac{\varphi_{i+1}^{(n)} - 2\varphi_i^{(n)} + \varphi_{i-1}^{(n)}}{2\Delta x^2} + S_i^{(n)}$$

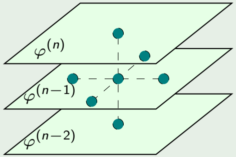
WaveToy Thorn

Thorn structure:

interface.ccl

- grid function phi[3]:

phi
phi_p
phi_p_p



- Boundary_SelectVarForBC

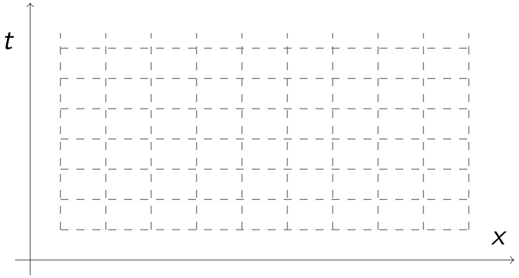
param.ccl

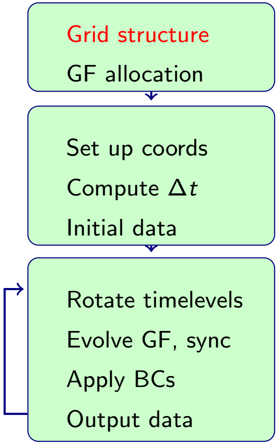
- Parameters of initial Gaussian pulse: amplitude A, radius R, width σ

schedule.ccl

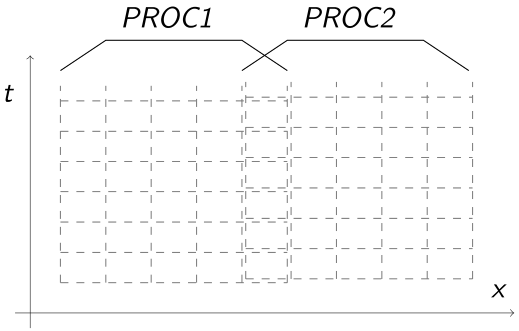
- WaveToy_InitialData
- WaveToy_Evolution
- WaveToy_Boundaries

WaveToy Thorn: Algorithm Illustration





WaveToy Thorn: Algorithm Illustration



Grid structure

GF allocation

Set up coords

Compute Δt

Initial data

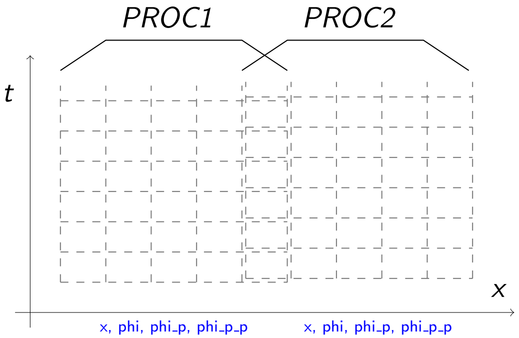
Rotate timelevels

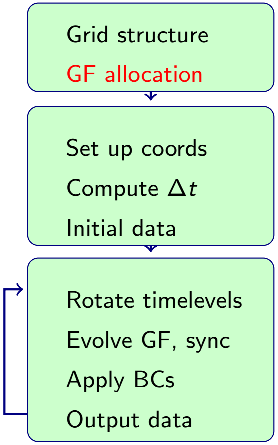
Evolve GF, sync

Apply BCs

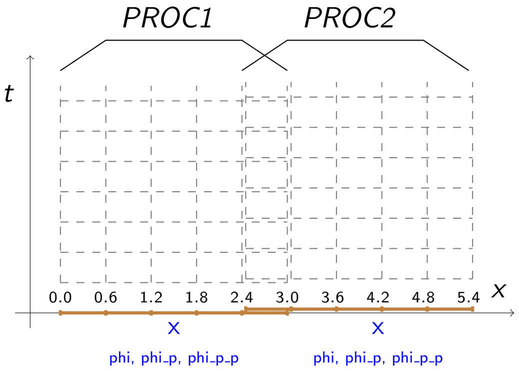
Output data

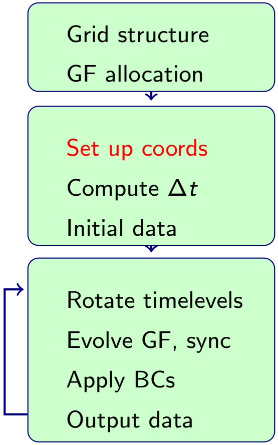
WaveToy Thorn: Algorithm Illustration



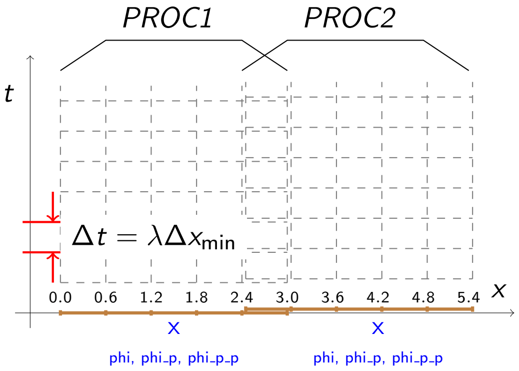


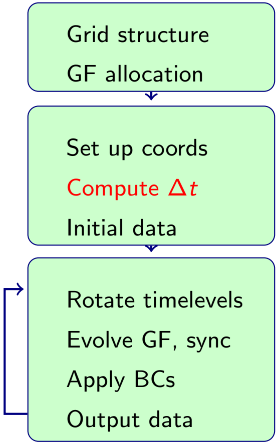
WaveToy Thorn: Algorithm Illustration



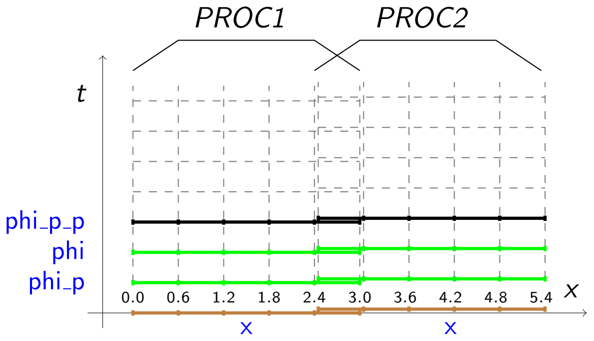


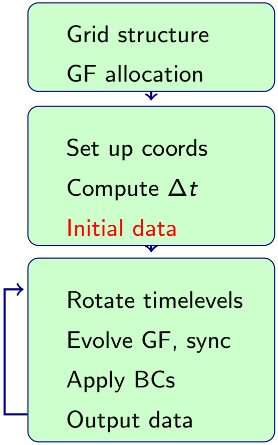
WaveToy Thorn: Algorithm Illustration



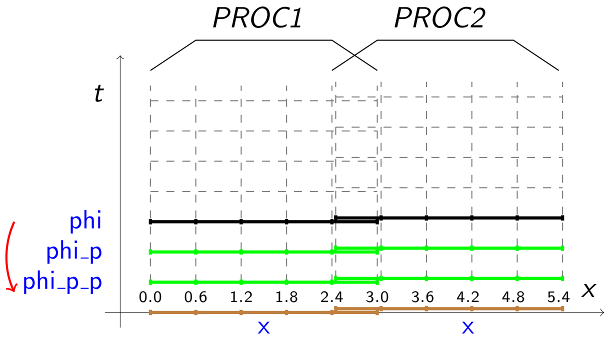


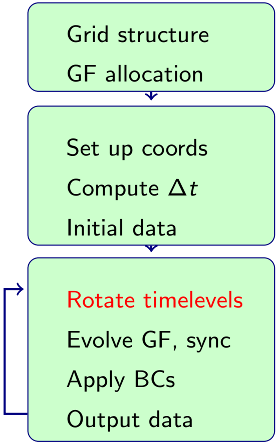
WaveToy Thorn: Algorithm Illustration





WaveToy Thorn: Algorithm Illustration




Grid structure

GF allocation

Set up coords

Compute Δt

Initial data

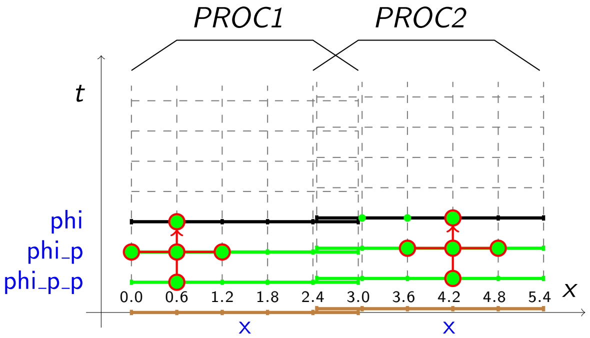
Rotate timelevels

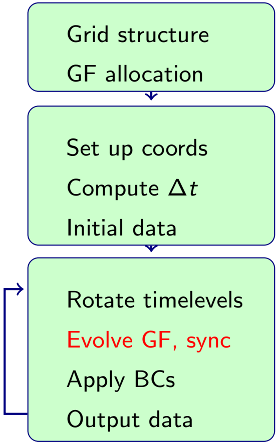
Evolve GF, sync

Apply BCs

Output data

WaveToy Thorn: Algorithm Illustration




Grid structure

GF allocation

Set up coords

Compute Δt

Initial data

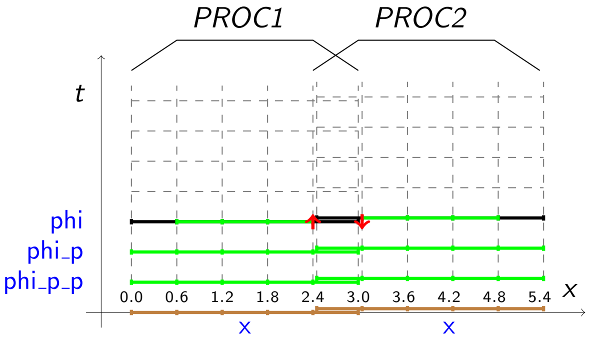
Rotate timelevels

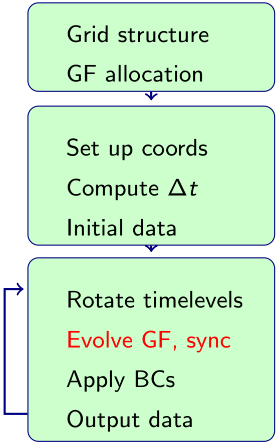
Evolve GF, sync

Apply BCs

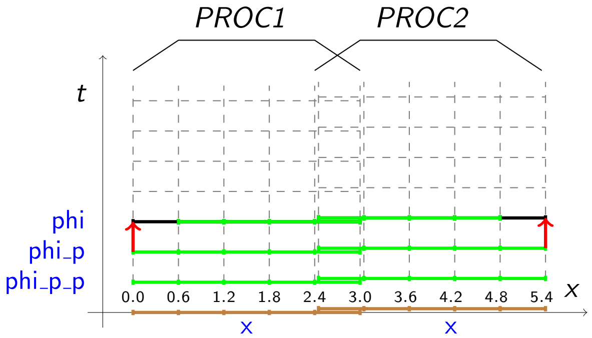
Output data

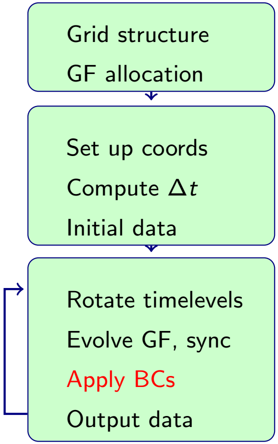
WaveToy Thorn: Algorithm Illustration



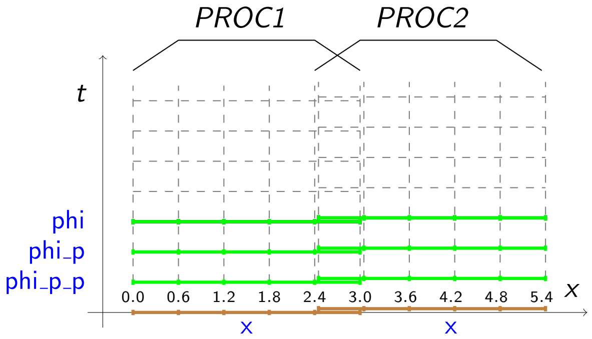


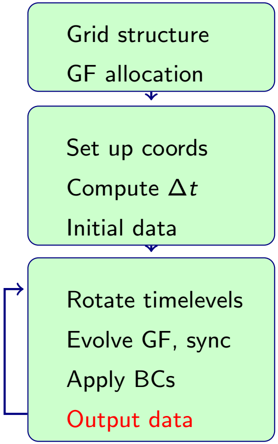
WaveToy Thorn: Algorithm Illustration



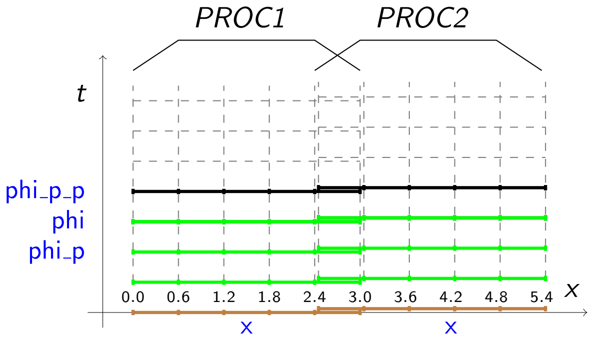


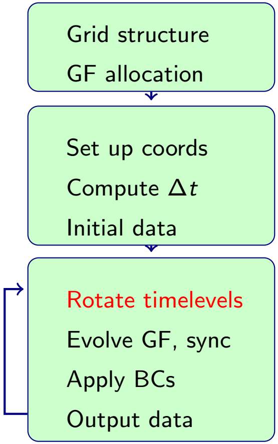
WaveToy Thorn: Algorithm Illustration





WaveToy Thorn: Algorithm Illustration





WaveToy Thorn

Directory structure:

```
WaveToy/
|-- COPYRIGHT
|-- README
|-- configuration.ccl
|-- doc
|   `-- documentation.tex
|-- interface.ccl
|-- schedule.ccl
|-- param.ccl
`-- src
    |-- WaveToy.c
    `-- make.code.defn
```

WaveToy Thorn

Directory structure:

```
WaveToy/
|-- COPYRIGHT
|-- README
|-- configuration.ccl
|-- doc
|   `-- documentation.tex
|-- interface.ccl
|-- schedule.ccl
|-- param.ccl
`-- src
    |-- WaveToy.c
    `-- make.code.defn
```

WaveToy Thorn

- `interface.ccl`:

```
IMPLEMENTS: wavetoy_simple
INHERITS: grid

PUBLIC:

CCTK_REAL scalarevolve TYPE=gf TIMELEVELS=3
{
  phi
} "The evolved scalar field"

CCTK_INT FUNCTION Boundary_SelectVarForBC( \
  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_SelectVarForBC
```

WaveToy Thorn cont.

- `schedule.ccl`:

```
STORAGE: scalarevolve[3]

SCHEDULE WaveToy_InitialData AT CCTK_INITIAL
{
  LANG: C
} "Initial data for 3D wave equation"

SCHEDULE WaveToy_Evolution AT CCTK_EVOL
{
  LANG: C
  SYNC: scalarevolve
} "Evolution of 3D wave equation"

SCHEDULE WaveToy_Boundaries AT CCTK_EVOL AFTER WaveToy_Evolution
{
  LANG: C
} "Select boundary conditions for the evolved scalar"

SCHEDULE GROUP ApplyBCs as WaveToy_ApplyBCs AT CCTK_EVOL AFTER WaveToy_Boundaries
{
} "Apply boundary conditions"
```

WaveToy Thorn cont.

- `param.ccl`:

```
CCTK_REAL amplitude "The amplitude of the waves"
{
 *:* :: "Anything"
} 1.0

CCTK_REAL radius "The radius of the gaussian wave"
{
 0:* :: "Positive"
} 0.0

CCTK_REAL sigma "The sigma for the gaussian wave"
{
 0:* :: "Positive"
} 0.1
```

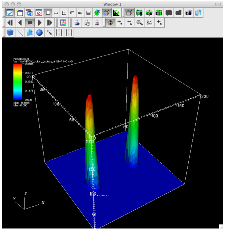
WaveToy Thorn cont.

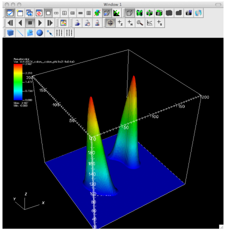
- Example parameter file:

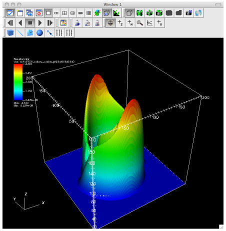
```
Cactus::cctk_run_title = "Simple WaveToy"

ActiveThorns = "time boundary Carpet CarpetLib CartGrid3D"
ActiveThorns = "CoordBase ioutil CarpetIOBasic CarpetIOASCII"
ActiveThorns = "CarpetIOHDF5 SymBase wavetoy"

cactus::cctk_itlast = 10000
time::dtfac = 0.5

IO::out_dir             = $parfile
IOBasic::outInfo_every  = 1
IOASCII::out1D_every    = 1
IOASCII::out1D_vars     = "wavetoy_simple::phi"

iohdf5::out_every = 10
iohdf5::out_vars        = "grid::coordinates{out_every=10000000} wavetoy_simple::phi"
```

WaveToy Thorn

Visualizing the results with VisIt:







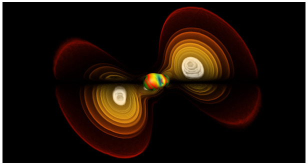
Many arrangements with many modules...

CactusBase	Basic utility and interface thorns
CactusConnect	Network utility thorns
CactusDoc	New, revised user-friendly documentation
CactusElliptic	Elliptic solvers / interface thorns
CactusIO	General I/O thorns
CactusNumerical	General numerical methods
CactusPUGH	Cactus Unigrid Driver thorn
CactusTest	Thorns for Cactus testing
CactusUtils	Misc. utility thorns
CactusWave	Wave example thorns
Carpet	Advanced multiblock/AMR driver (www.carpetcode.org)
ExternalLibraries	Interfaces to LAPACK, GSL, HDF5, LORENE etc.

The Einstein Toolkit

The Einstein Toolkit

The **Einstein Toolkit** is a collection of codes for numerical relativity. The toolkit includes a vacuum spacetime solver (McLachlan), a relativistic hydrodynamics solver, along with thorns for initial data, analysis and computational infrastructure.



The Einstein Toolkit currently consists of an open set of over 100 Cactus thorns for computational relativity along with associated tools for simulation management and visualization. The toolkit includes a vacuum spacetime solver (McLachlan), a relativistic MHD solver GRHydro (formerly the public version of the Whisky code), along with thorns for initial data, analysis and computational infrastructure.

Website: http://einsteintoolkit.org

McLachlan: Metric solver

The Einstein's equations in covariant 4-dimensional tensor form:

$$R_{\mu\nu} - \frac{1}{2} g_{\mu\nu} R = \frac{8\pi G}{c^4} T_{\mu\nu}$$

McLachlan: Metric solver

ADM formulation: the Einstein's equations in 3+1 split:

$$\mathrm{d}s^2 = g_{\mu\nu}\mathrm{d}x^\mu \mathrm{d}x^\nu \equiv (-\alpha^2 + \beta_i\beta^i)\mathrm{d}t^2 + 2\beta_i \mathrm{d}t\ \mathrm{d}x^i + \gamma_{ij}\mathrm{d}x^i\mathrm{d}x^j\ ,$$

Evolution equations:

$$\begin{aligned}(\partial_t - \mathcal{L}_\beta)\,\gamma_{ij} &= -2\alpha K_{ij},\\ (\partial_t - \mathcal{L}_\beta)\,K_{ij} &= -D_iD_j\alpha + \alpha(R_{ij} + KK_{ij} - 2K_{ik}K^k{}_j),\end{aligned}$$

Constraints:

$$\begin{aligned}\mathcal{H} &\equiv R + K^2 - K_{ij}K^{ij} = 0,\\ \mathcal{D}^i &\equiv D_j(K^{ij} - \gamma^{ij}K) = 0.\end{aligned}$$

McLachlan: Metric solver

Introduce BSSN variables, ϕ, K, $\tilde{\gamma}_{ij}$, $\tilde{A}_{ij}$, and $\tilde{\Gamma}^i$:

$$\begin{aligned}
\phi &= \ln\psi = \frac{1}{12}\ln\gamma, \\
K &= \gamma_{ij}K^{ij}, \\
\tilde{\gamma}_{ij} &= e^{-4\phi}\gamma_{ij}, \\
\tilde{A}_{ij} &= e^{-4\phi}A_{ij} = e^{-4\phi}(K_{ij} - \frac{1}{3}\gamma_{ij}K),
\end{aligned}$$

where ψ is the conformal factor: $\gamma_{ij} = \psi^4\tilde{\gamma}_{ij}$.

McLachlan: Metric solver

BSSN formulation (better numerical stability than ADM):

$$
\begin{aligned}
(\partial_t - \mathcal{L}_\beta)\, \tilde{\gamma}_{ij} &= -2\alpha \tilde{A}_{ij}, \\
(\partial_t - \mathcal{L}_\beta)\, \phi &= -\frac{1}{6}\alpha K, \\
(\partial_t - \mathcal{L}_\beta)\, \tilde{A}_{ij} &= e^{-4\phi}[-D_i D_j \alpha + \alpha R_{ij}]^{TF} + \alpha(K\tilde{A}_{ij} - 2\tilde{A}_{ik}\tilde{A}^k{}_j), \\
(\partial_t - \mathcal{L}_\beta)\, K &= -D^i D_i \alpha + \alpha(\tilde{A}_{ij}\tilde{A}^{ij} + \frac{1}{3}K^2), \\
\partial_t \tilde{\Gamma}^i &= \tilde{\gamma}^{jk}\partial_j\partial_k\beta^i + \frac{1}{3}\tilde{\gamma}^{ij}\partial_j\partial_k\beta^k + \beta^j\partial_j\tilde{\Gamma}^i - \tilde{\Gamma}^j\partial_j\beta^i + \frac{2}{3}\tilde{\Gamma}^i\partial_j\beta^j \\
&\quad - 2\tilde{A}^{ij}\partial_j\alpha + 2\alpha(\tilde{\Gamma}^i{}_{jk}\tilde{A}^{jk} + 6\tilde{A}^{ij}\partial_j\phi - \frac{2}{3}\tilde{\gamma}^{ij}\partial_j K).
\end{aligned}
$$

for more details, see [Alcubierre et al. (2003), arXiv:gr-qc/0206072]

Kranc

Kranc

Kranc Assembles Numerical Code

- High-level thorn generator
- Original version by S. Husa, I. Hinder, C. Lechner
- Used for many scientific papers (e.g., [arXiv:gr-qc/040423]
- A calculation in Kranc is written using Mathematica (a symbolic programming language)
- Minimal Kranc script contains:
  - variable groups definitions;
  - a set of calculations.
- For more, see: `http://kranccode.org`

Kranc code example

Original formula (evolution of K, the trace of extrinsic curvature):

$$(\partial_t - \mathcal{L}_\beta)\, K = -e^{-4\phi}(\tilde{\gamma}^{ij}\partial_i\partial_j\alpha - \tilde{\Gamma}^k\partial_k\alpha + 2\tilde{\gamma}^{ij}\partial_i\phi\partial_j\alpha)$$
$$+ \ \alpha(\tilde{A}_{ij}\tilde{A}^{ij} + \frac{1}{3}K^2),$$

Corresponding expression from the Kranc script is easy to read:

```
dottrK  -> - em4phi ( gtu[ua,ub] ( PD[alpha,la,lb]
                   + 2 cdphi[la] PD[alpha,lb] )
                   - Xtn[ua] PD[alpha,la] )
             + alpha (Atm[ua,lb] Atm[ub,la] + (1/3) trK^2)
```

Kranc code example

The autogenerated code in C is not easy to read, but instead it is optimized in terms of performance and can be tuned for a particular architecture:

```
CCTK_REAL_VEC dottrK =
  kmadd(em4phi,kmul(ToReal(-1.),kmadd(kmadd(JacPDstandardNth1alpha,Xtn1,
  kmadd(JacPDstandardNth2alpha,Xtn2,kmul(JacPDstandardNth3alpha,Xtn3))),
  ToReal(-1.),kmadd(gtu11,kmadd(cdphi1,kmul(JacPDstandardNth1alpha,ToRea
  l(2.)),JacPDstandardNth11alpha),kmadd(gtu22,kmadd(cdphi2,kmul(JacPDsta
  ndardNth2alpha,ToReal(2.)),JacPDstandardNth22alpha),kmadd(gtu12,kadd(J
  acPDstandardNth12alpha,kmadd(kmadd(cdphi2,JacPDstandardNth1alpha,kmul(
  cdphi1,JacPDstandardNth2alpha)),ToReal(2.),JacPDstandardNth21alpha)),k
  madd(gtu33,kmadd(cdphi3,kmul(JacPDstandardNth3alpha,ToReal(2.)),JacPDs
  tandardNth33alpha),kmadd(gtu13,kadd(JacPDstandardNth13alpha,kmadd(kmad
  d(cdphi3,JacPDstandardNth1alpha,kmul(cdphi1,JacPDstandardNth3alpha)),T
  oReal(2.),JacPDstandardNth31alpha)),kmul(gtu23,kadd(JacPDstandardNth23
  alpha,kmadd(kmadd(cdphi3,JacPDstandardNth2alpha,kmul(cdphi2,JacPDstand
  ardNth3alpha)),ToReal(2.),JacPDstandardNth32alpha))))))))),kmul(alpha
  L,kmadd(Atm11,Atm11,kmadd(Atm22,Atm22,kmadd(Atm33,Atm33,kmadd(kmul(trK
  L,trKL),ToReal(0.333333333333333333333333333333),kmadd(kmadd(Atm12,Atm
  21,kmadd(Atm13,Atm31,kmul(Atm23,Atm32))),ToReal(2.),kmul(kadd(rho,trS)
  ,ToReal(12.566370614359172)))))))));
```

Magnetohydrodynamical part

- The Valencia formulation of ideal MHD ([J. Font, LRR (2008)]);
- ideal MHD means E_ν vanishes in the comoving frame, $u_\mu F^{\mu\nu} = 0$;
- hydrodynamic and electromagnetic contributions to stress-energy tensor:

$$T^{\mu\nu}_{\mathrm{H}} = \rho h u^\mu u^\nu + P g^{\mu\nu} = (\rho + \rho\epsilon + P)\, u^\mu u^\nu + P g^{\mu\nu} \quad ,$$

$$T^{\mu\nu}_{\mathrm{EM}} = F^{\mu\lambda} F^{\nu}{}_{\lambda} - \frac{1}{4} g^{\mu\nu} F^{\lambda\kappa} F_{\lambda\kappa} = b^2 u^\mu u^\nu - b^\mu b^\nu + \frac{b^2}{2} g^{\mu\nu}$$

- evolved (primitive) variables: ρ, ϵ, P, u^μ and $b^\mu = u_\nu {}^{*}F^{\mu\nu}$
- evolution equations:

$$\nabla_\mu J^\mu = 0 \; , \qquad \nabla_\mu T^{\mu\nu} = 0 \; , \qquad \nabla_\nu {}^{*}F^{\mu\nu} = 0 \; .$$

Flux-conservative evolution system

Evolution is written in first-order form for the following conserved variables:

$$
\begin{aligned}
D &= \sqrt{\gamma}\rho W \ , && (1)\\
S_j &= \sqrt{\gamma}\left(\rho h^* W^2 v_j - \alpha b^0 b_j\right) \ , && (2)\\
\tau &= \sqrt{\gamma}\left(\rho h^* W^2 - P^* - (\alpha b^0)^2\right) - D \ , && (3)\\
\mathcal{B}^k &= \sqrt{\gamma} B^k \ , && (4)
\end{aligned}
$$

where $\gamma = \mathrm{det}\gamma_{ij}$, $v^i = \frac{u^i}{W} + \frac{\beta^i}{\alpha}$ is the velocity of the fluid as seen by an Eulerian observer at rest, and $W \equiv (1 - v^i v_i)^{-1/2}$ is the Lorentz factor. The evolution system for the conserved variables can be written as:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}^i}{\partial x^i} = \mathbf{S} \ , \qquad (5)$$

Further details in [P. Mösta et al. (2013), arXiv:1304.5544]

Numerical methods

Numerical discretization of the MHD part:

- cell-centered finite volume scheme;
- reconstruction of primitive variables on cell interfaces: TVD, PPM, ENO, WENO5, ePPM, MP5;
- conversion from conserved to primitive variables for wide range of EoS: polytropic, Γ-law, hybrid, and microphysical finite-temperature EoS;
- preserving the $\nabla \cdot B = 0$ constraint: hyperbolic divergence cleaning, and simplified constained transport;
- Harten-Lax-van Leer-Einfeldt (HLLE) approximate Riemann solver.

GRHydro-MHD materials

The GRHydro-MHD code comes with a set of testcases, which can be compared to the results presented in the paper [arXiv:1304.5544].
Test cases include:

- monopoles test: response of the divergence cleaning scheme to numerically generated monopoles;
- cylindrical explosions, magnetic rotors, Alfvén waves and advected loops;
- spherical accretion onto a black hole;
- oscillations of a magnetised polytropic neutron star;
- collapse of a magnetised stellar core.

Web link: `http://einsteintoolkit.org/publications/2013_MHD/`