The Cactus Framework

Erik Schnetter
September 2006
Outline

• History
• The Cactus User Community
• Cactus
• Usage Patterns
Bird’s eye view

- Cactus is a freely available, portable, and manageable environment for collaboratively developing parallel, scalable, high-performance multi-dimensional component-based simulations.

Saguaro
(Carnegiea gigantea)
History

- Cactus 1.0 was released in April 1997 at NCSA by the numerical relativity group
- Cactus 4.0 is available since 1999
- Since then incremental (i.e., mostly backwards-compatible) development
- Most users today still in numerical relativity
Overall Design

Core flesh with plug-in thorns

parallelism
memory management
I/O
SOR solver
your computational tools
multigrid
interpolation
reduction

extensible APIs
ANSI C
parameters
schedule
grid variables
make system
error handling
coordinates
boundary conditions
AMR
CFD
wave equation
Einstein equations
remote steering
your physics

interpolation
reduction
User Base

• Some groups base their whole code on Cactus

• Some groups use Cactus on the side

• In some places, individual students/postdocs use a Cactus-based public code

• Most Cactus users write thorns

• Few Cactus users contribute to the infrastructure

• Cactus and the core thorns are public (LGPL)

• Many thorns are private
Development Process

- Flesh and core thorns are developed by a small group
- Weekly video conferences
- Frequent bug reports/feature requests from users
- Trying to balance stability and new features
- Mostly steady development with ~10 releases; many users live off CVS, not stable versions
- (Physics thorns are developed by physicists)
Einstein Toolkit

- A common infrastructure for all relativity codes
- Defines common variables, common schedule events, etc.
- Comes with public thorns for basic tasks (simple initial data, simple analysis methods)
- There are at least five production level relativity codes based on Cactus, all but one private, all using the Einstein Toolkit
- Three-level structure:
Library vs. Framework

• A framework is like a library, except that it contains the main programme -- the user modules are libraries.

• Crucial for easy interoperability -- otherwise, two modules may “fight” over who may be the main programme.

• Cactus thorns are “connected” via their schedule.

• Schedule is constructed at run time -- no code needs to know all compiled thorns.

• Thorns can be developed completely independently.
Anatomy of a Thorn

- A thorn in Cactus contains:
  - Cactus declarations (CCL language)
  - source code (C, C++, Fortran)
  - makefile fragments
  - documentation

- test cases
- example parameter files

- Thorns are the basic modular units
- Usually, each thorn is in a separate CVS repository
interface.ccl

• Declares *thorn name* and *implementation name*

• Declares *grid functions*

• Can *inherit* public grid functions from other implementations

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

```c
IMPLEMENTS: ADMConstraints
INHERITS: ADMBase

CCTK_REAL Hamiltonian TYPE=gf{
    ham
} "Hamiltonian Constraint"

CCTK_REAL Momentum TYPE=gf{
    momx momy momz
} "Momentum Constraint"
```
• Calls routines at certain times, e.g. *initial* or *evol* or *analysis*

• *Schedule groups* introduce a hierarchical structure

• Rule-based: schedule *AFTER*, *BEFORE*, *WHILE*

• Allocates 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
- Five types: integer, real, boolean, keyword, string
- Allowed ranges need to be declared
- Can “inherit” public parameters from other implementations, possibly extending ranges

SHARES: ADMBase

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

PRIVATE:

CCTK_REAL gaussian_amplitude \
  "Amplitude"
{
  0.0:* :: "must be nonnegative"
} 1.0
Example Source Code

```c
#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”
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 closely tied to driver

• I/O (efficient and parallel) and checkpointing/recovery also somewhat driver specific
Application Base

• Current Cactus users are mostly in numerical relativity, including relativistic hydro

• We begin to use it for CFD

• Sporadic uses in many fields: astrophysics (Zeus), chemistry, oil field simulations, ...

• Cactus is mostly used for 3D time evolution with explicit time stepping

• Non-trivial initial data (elliptic equations) are mostly imported (this used to be different)

• We have a few public “Killer Thorns”
Visualisation

• gnuplot, xgraph, ygraph, etc. for 1D and 2D ASCII output

• Common HDF5 data format for Cactus simulations (because I/O is from a few thorns only)

• Amira, OpenDX for both debugging and production visualisation

• Built-in web server with jpeg slides

• www.cactuscode.org: 5555/
• 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
Discretisations

• Cactus supports block-structured regular grids best

• We use both Berger-Oliger AMR and multi-block discretisations

• We use (high order) finite differences

• Some experiments with pseudo-spectral discretisations

• Some experiments with particle codes (SPH)

• Plans for unstructured grids (finite elements, finite volumes)
Performance

- Performance must be measured
- Parallelisation performance depends on driver thorn
- I/O performance depends on I/O thorns
- Important: convenient pervasive performance measurements for application code
- Cactus offers timers
- Automatic timers for each scheduled routine
Random Details

- CCL files are parsed by perl code, creating C code and latex files
- Makefile fragments require GNU make
- Flesh written in ANSI C, thorns can be C, C++, or Fortran; other languages could be “easily” added
- Flesh helps with function calls between different languages (strings!)
- Fortran code is preprocessed with cpp (and sanitised with perl)
- Documentation uses latex
Building Cactus

- User can build several different configurations in the same Cactus tree
- User chooses list of thorns and set of options for each configuration
- Cactus is not “installed” in the way e.g. PETSc is; each user has the complete source tree

- Problem: User makes private modification → user forgets → results are not reproducible (solution: store source for each simulation)
- We keep a list of known good build options for each machine
Further Information

• Cactus: www.cactuscode.org
• Live simulation: www.cactuscode.org:5555
• Carpet (AMR driver): www.carpetcode.org