Cactus

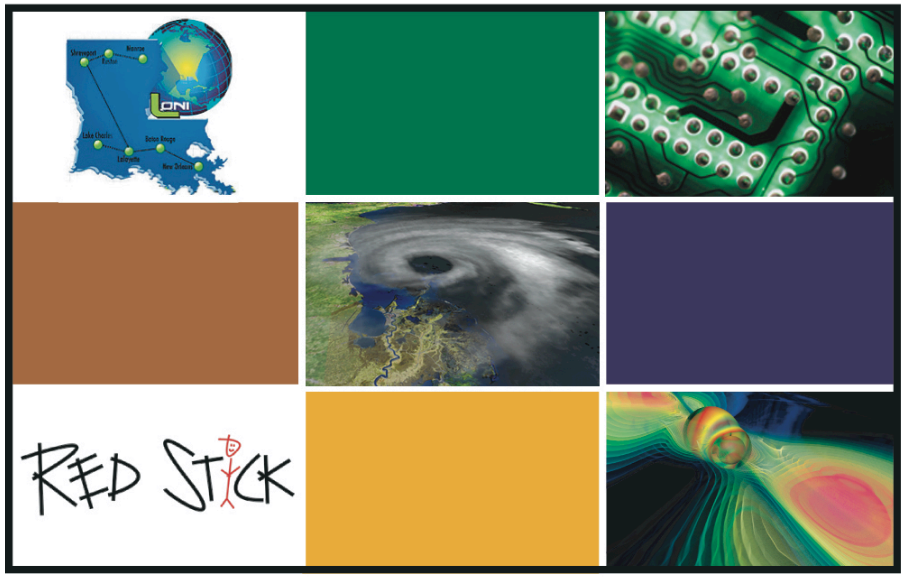
A Software Framework for High Performance Computing

Erik Schnetter

Baton Rouge, March 2008

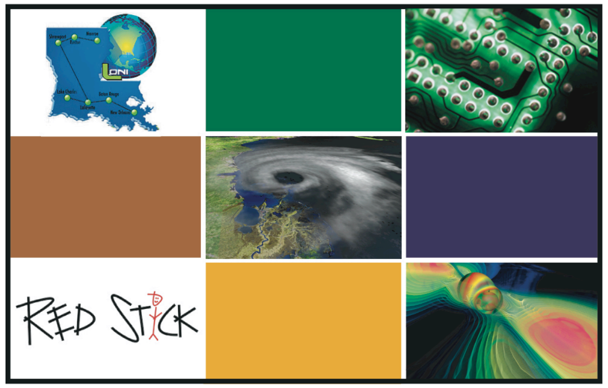
LSU

CENTER FOR COMPUTATION & TECHNOLOGY



Outline

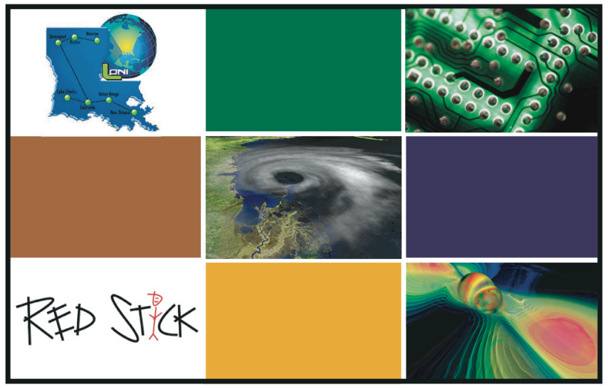
1. What is a software framework?
2. How to use Cactus
3. Parallel programming
4. Additional tools for Cactus users
5. Further documentation



CENTER FOR COMPUTATION
& TECHNOLOGY

Outline

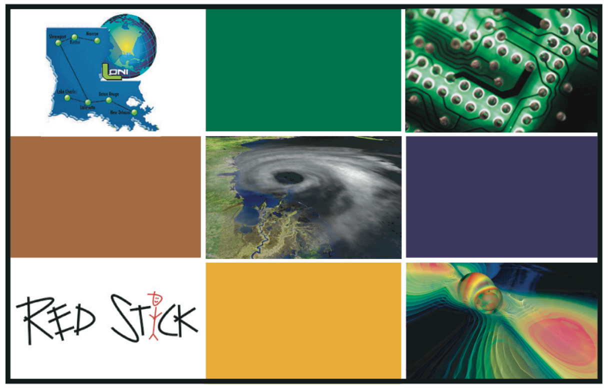
1. What is a software framework?
2. How to use Cactus
3. Parallel programming
4. Additional tools for Cactus users
5. Further documentation



LSU
CENTER FOR COMPUTATION
& TECHNOLOGY

Observations on Programming

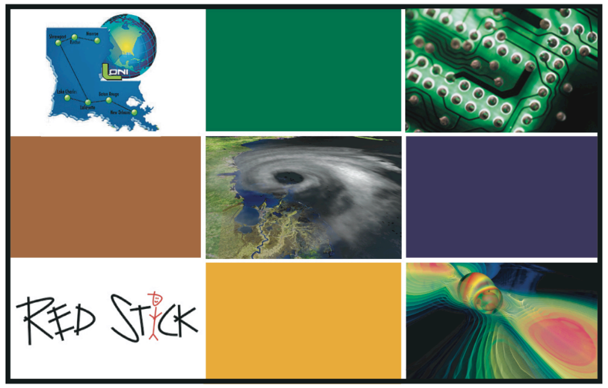
- Programming is a social activity
- Source code is read more often than written
- On interesting projects, many people are collaborating
- Programming means also:
  handling relations between people



LSU
CENTER FOR COMPUTATION
& TECHNOLOGY

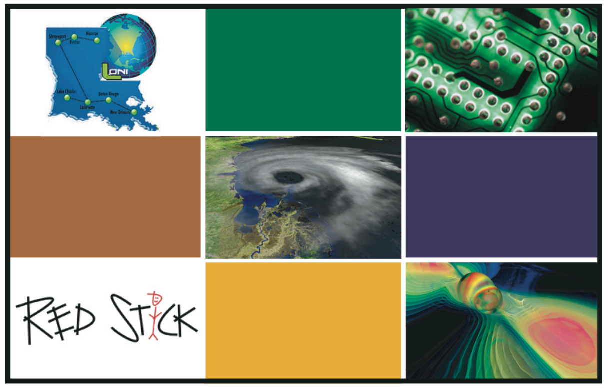
Tools for Collaborations

- Libraries can be re-used by many people
- cvs, svn, git repositories help handle source code conflicts and human mistakes
- Modular programming helps people keep an overview over programs
- Software Frameworks allows code to be developed separately, and later combined to form large applications



Software Frameworks

- Program is split into <u>components</u>
- Each component is a large piece of code (e.g. solver, I/O, parallelism, visualisation, ...)
- User develops new components, making use of existing components
- Simplifies component re-use, allows exchanging components, gives structure to overall program



CENTER FOR COMPUTATION
& TECHNOLOGY

Cactus

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

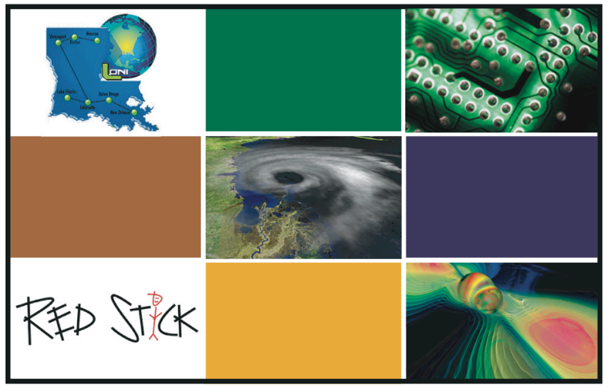
wave equation

Einstein equations

your physics

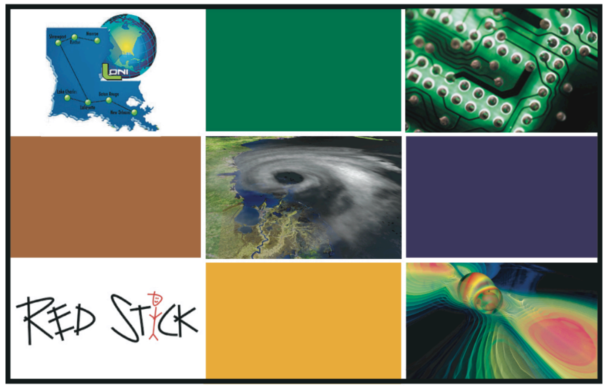
remote steering

Core <u>flesh</u> with plug-in <u>thorns</u>



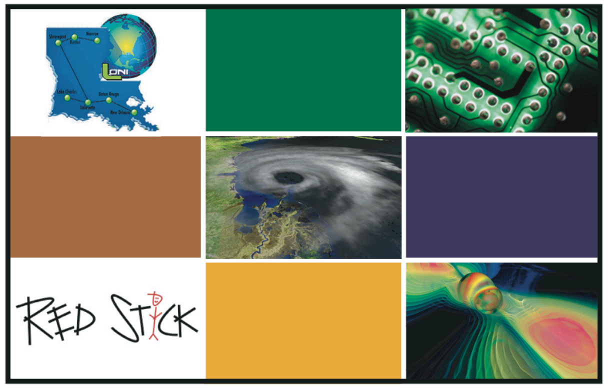
History

- Cactus 1.0 was released in April 1997 at NCSA by the numerical relativity group
- Cactus 4.0 is available since 1999
- Development mostly at the CCT, contributions from AEI (Germany)
- Used in several fields of science (numerical relativity, astrophysics, quantum gravity, CFD, geosciences, ...)



Licensing

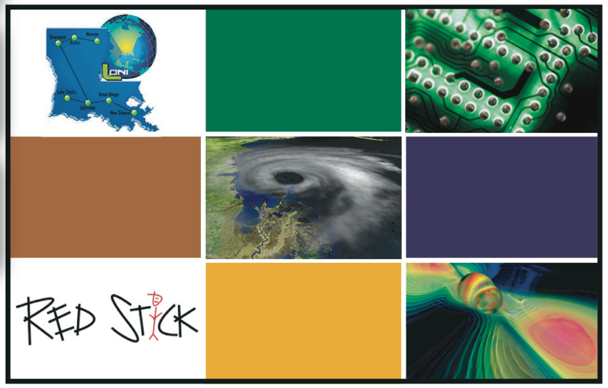
- Cactus is open source (LGPL), source code freely available, no conditions on use
- Framework developed at CCT, most thorns developed independently by application scientists
- Many thorns are public, many other thorns are private



Application Toolkits

- Common infrastructure for all codes in the same application area
- Defines common variables, common schedule events, etc.
- Contains public thorns for basic tasks (simple initial data, simple analysis methods)
- Numerical Relativity: Five production level codes based on Cactus, all but one private, all using the Einstein Toolkit
- Three-level structure:

Physics code
Einstein Toolkit
Computational Toolkit
Cactus



Outline

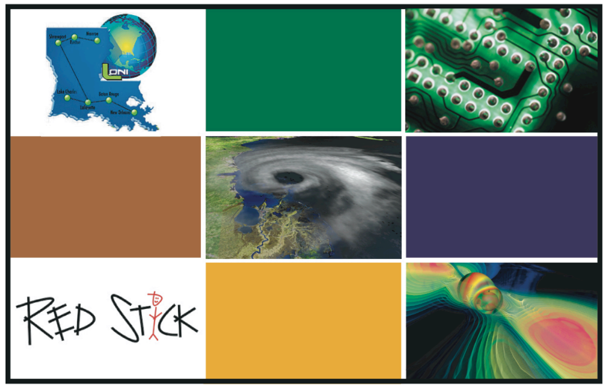
1. What is a software framework?

2. **How to use Cactus**

3. Parallel programming

4. Additional tools for Cactus users

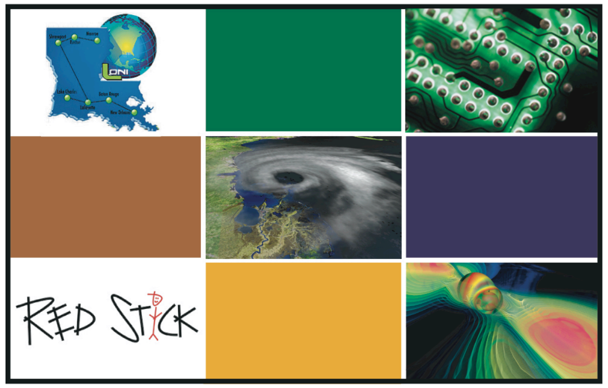
5. Further documentation



LSU
CENTER FOR COMPUTATION
& TECHNOLOGY

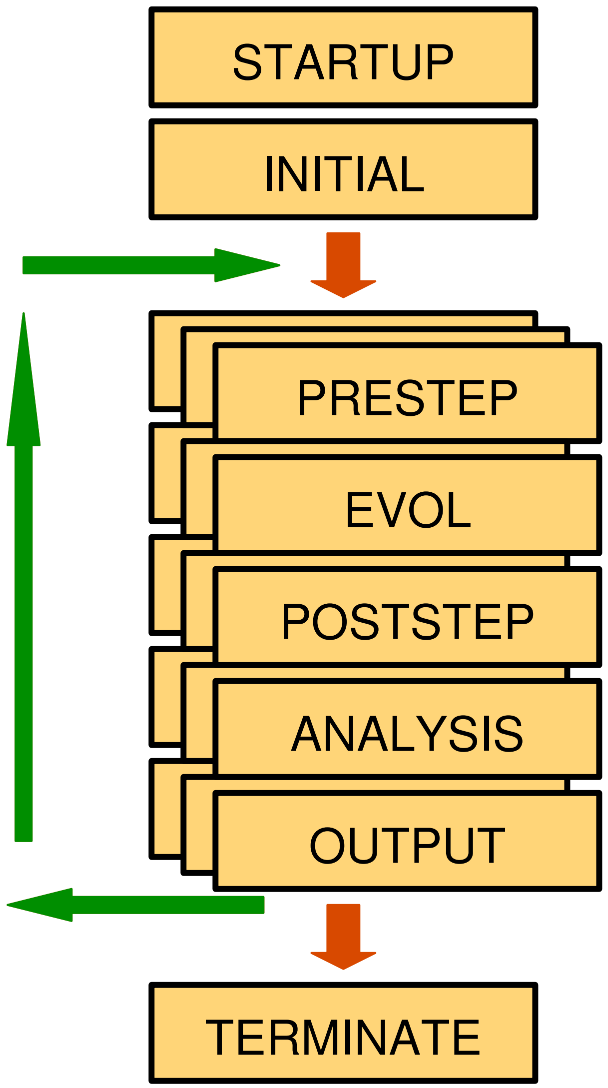
Anatomy of an Application which uses Cactus

- Application consists of a set of thorns (components)
- Each thorn has a specific task (coordinate system, Fourier transformation, initial data, equation to be solved, I/O, ...)
- Thorns are "connected" via their schedule
- Schedule is constructed at run time – no code needs to know all compiled thorns
- Thorns can be developed independently (important for large collaborations)



LSU
CENTER FOR COMPUTATION
& TECHNOLOGY

Run-Time Behaviour of an Application using Cactus

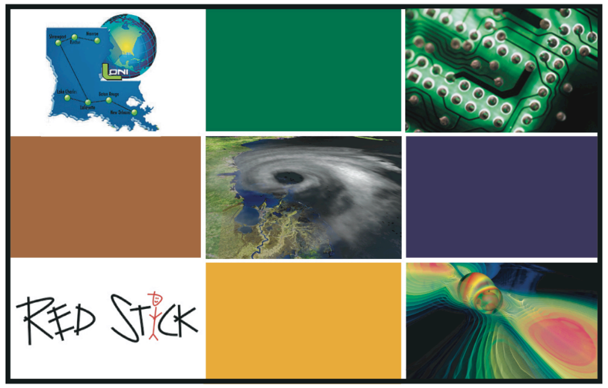


- Routines are scheduled to execute in predefined schedule bins
- Bins are executed in a specific order
- Can create groups of routines
- Can add conditions (*before*, *after*, *while*)

LSU
CENTER FOR COMPUTATION
& TECHNOLOGY

Anatomy of a Thorn

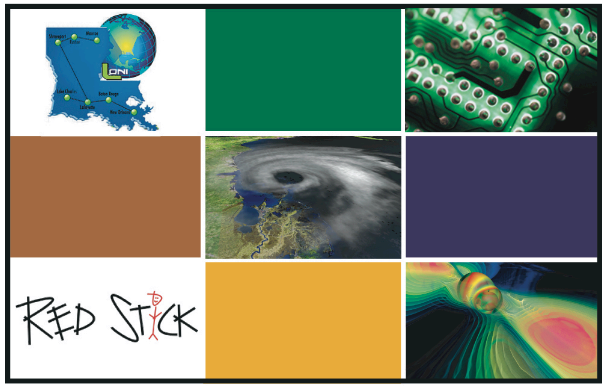
- A thorn in Cactus contains:
  - Cactus declarations (CCL language)
  - source code (C, C++, Fortran)
  - makefile fragments
  - documentation
  - test cases
  - example parameter files
- Thorns are the largest modular units giving structure to a program
- Each thorn has one specific, high-level task



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)

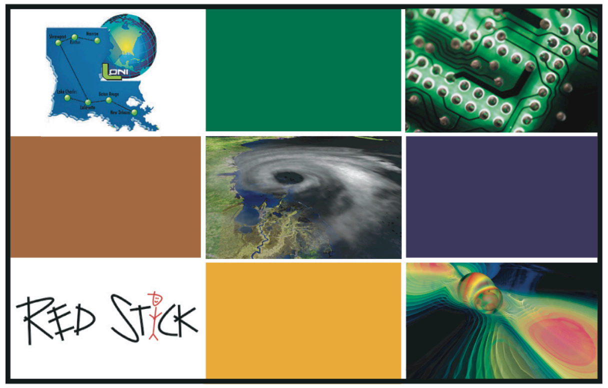
```
IMPLEMENTS: ADMConstraints
INHERITS: ADMBase

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

CCTK_REAL Momentum TYPE=gf
{
  momx momy momz
} "Momentum Constraint"
```



schedule.ccl

- 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

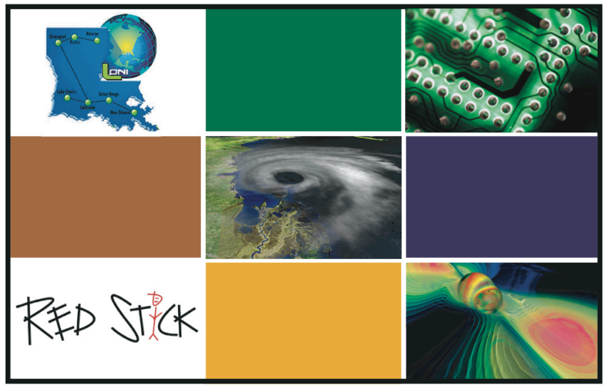
```
SCHEDULE ADMConstraints_Calculate AT analysis
{
  LANG: Fortran
  STORAGE: Hamiltonian Momentum
  SYNC: Hamiltonian Momentum
  TRIGGERS: Hamiltonian Momentum
} “Calculate the constraints”
```



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

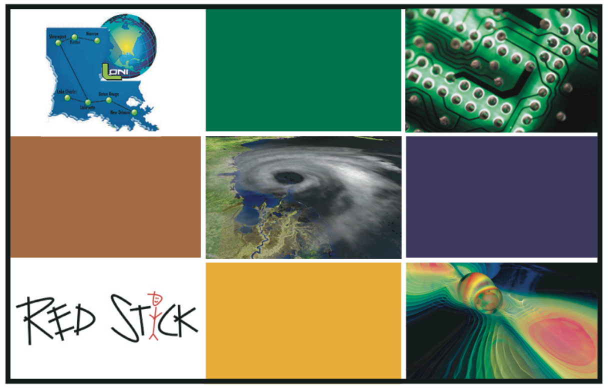
```
SHARES: ADMBase

EXTENTS KEYWORD initial_data
{
  “gaussian” :: “Gaussian pulse”
}

PRIVATE:

CCTK_REAL gaussian_amplitude \
  “Amplitude”
{
  0.0:* :: “must be nonnegative”
} 1.0
```



Example Source Code

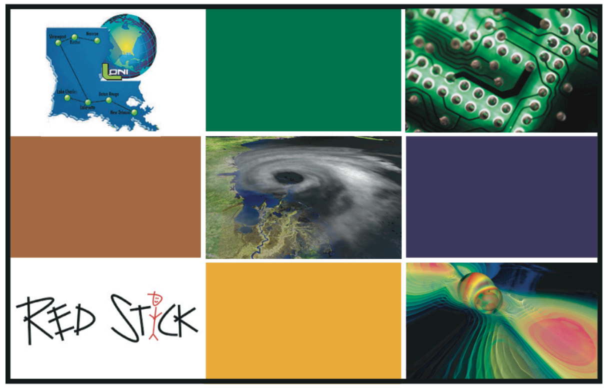
```
#include "cctk.h"
#include "cctk_Arguments.h"

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)
    ...
```

Source code can be written in
C, C++, Fortran 77, or Fortran 90



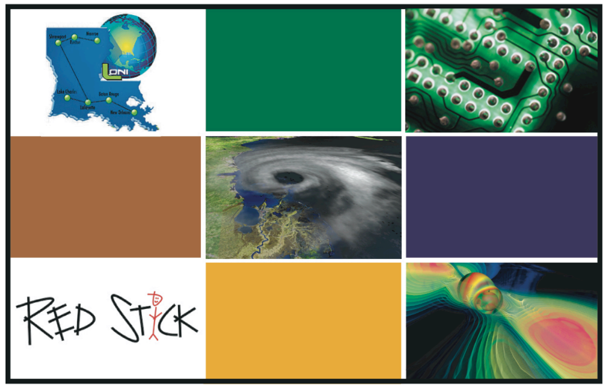
Driver

- 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



Building Cactus

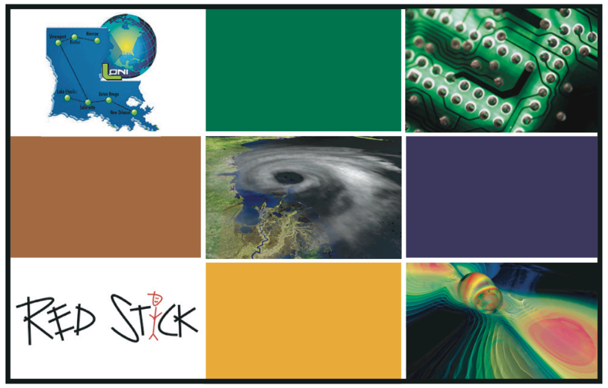
- User can build several different <u>configurations</u> in the same Cactus tree
- User chooses list of thorns and set of options for each configuration
- Cactus is not "installed" in the way e.g. PETSc is; each user has the complete source tree
- Problem: User makes private modification → user forgets → results are not reproducible (solution: store source for each simulation)
- We keep a list of known good build options for each machine



Parameter Files

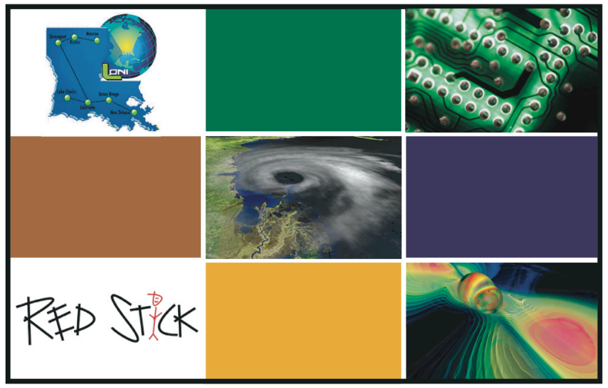
- At run time, parameter files activate thorns and specify parameter values
- Not all compiled thorns need to be active

```
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”
```



Running Cactus

- Serial: start executable in the normal way
  `./cactus_wavetoy wavetoyc.par`
- Parallel: Use mpirun as usual
  `mpirun -np 512 ./cactus_wavetoy wavetoyc.par`
- Batch queues:
  We keep a list of sample batch scripts for most LONI and TeraGrid machines
- Cactus is highly portable, runs on almost all available machines



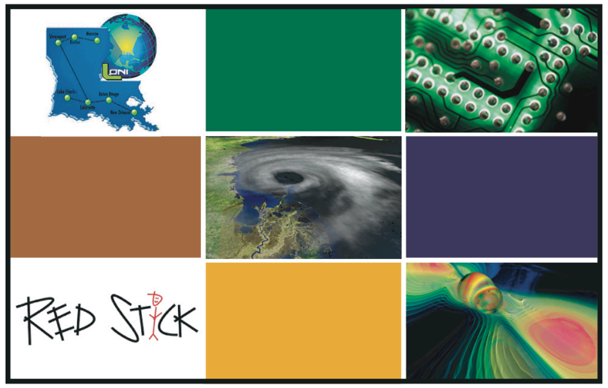
LSU
CENTER FOR COMPUTATION
& TECHNOLOGY

Checkpointing

- Supercomputers are unreliable, tend to have hardware problems
- Batch slots have too short time limits
- Necessary to checkpoint simulations regularly, recover after problems
- Framework knows all variables – can checkpoint without having to write special routines for each thorn!

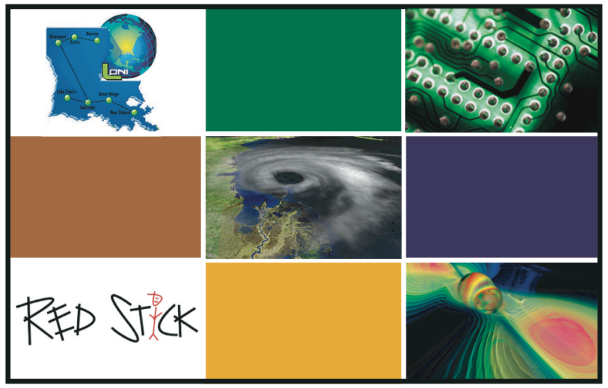
Output and Visualisation

- Simple screen output indicates progress of simulation
- There are standard thorns for ASCII and binary I/O
- Output can be visualised e.g. using gnuplot (lines, surfaces) or VisIt (real 3D visualisation)



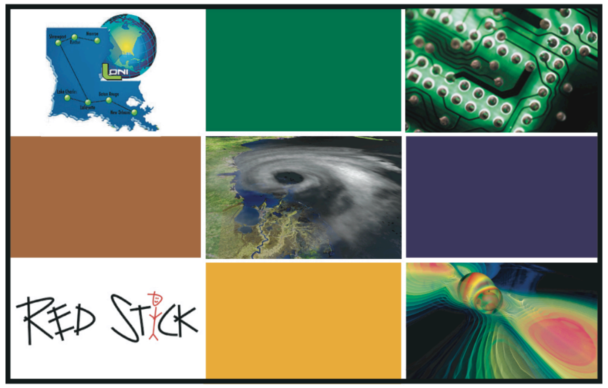
Outline

1. What is a software framework?
2. How to use Cactus
3. **Parallel programming**
4. Additional tools for Cactus users
5. Further documentation



Parallel Programming Models in Cactus

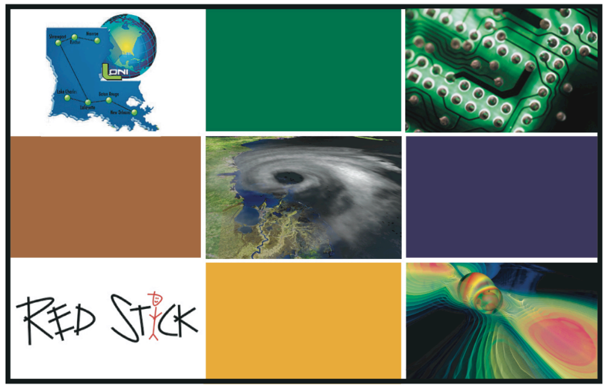
- We use Cactus on laptops as well as on the planet’s largest supercomputers
- Cactus is intended to be used on both distributed and shared memory machines
- The <u>driver</u> uses MPI to distribute the data onto a set of nodes
- Recently, OpenMP has become useful (Abe, Queen Bee, Ranger)



LSU
CENTER FOR COMPUTATION
& TECHNOLOGY

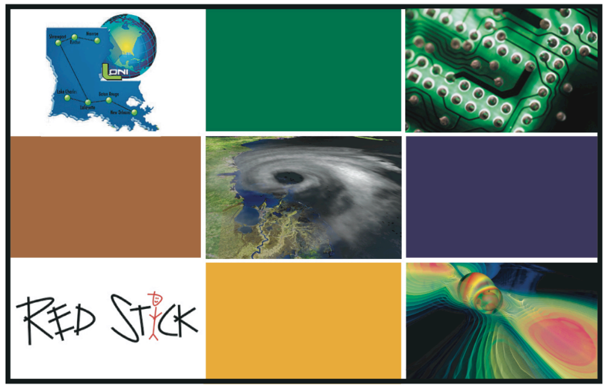
Parallelism through the Cactus Driver

- Parallelism (MPI) is handled by the driver
- Ideally, user code should not care how the grid/mesh is distributed
- Driver can be optimised / exchanged without changing user code
- User code does not have to care about details of parallelisation

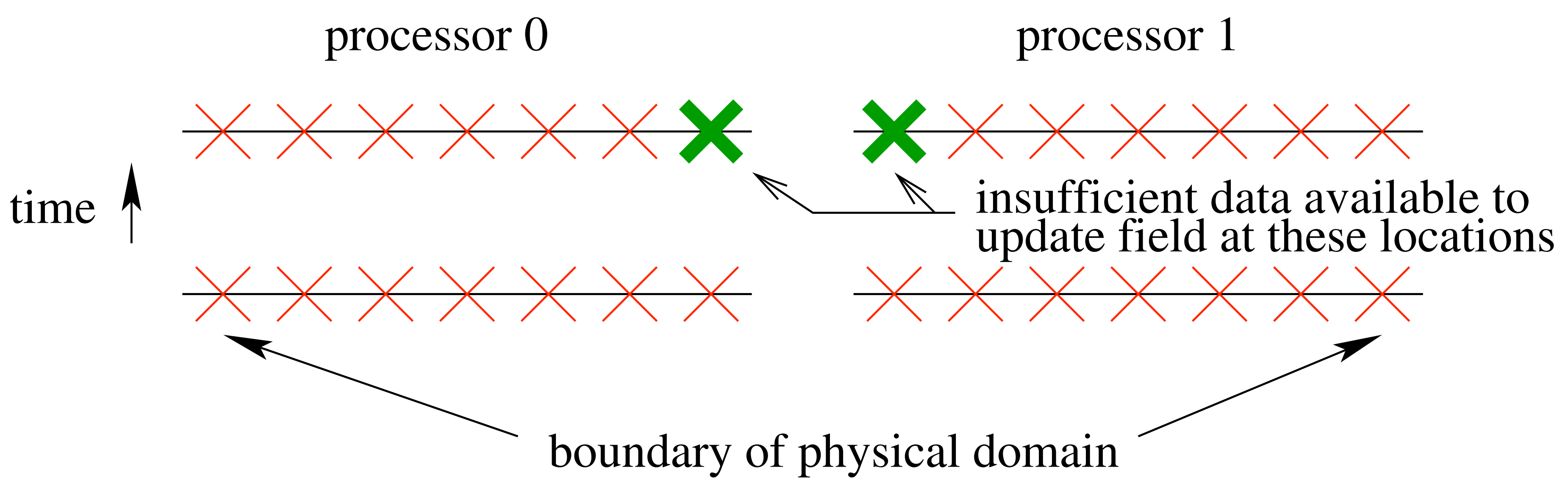


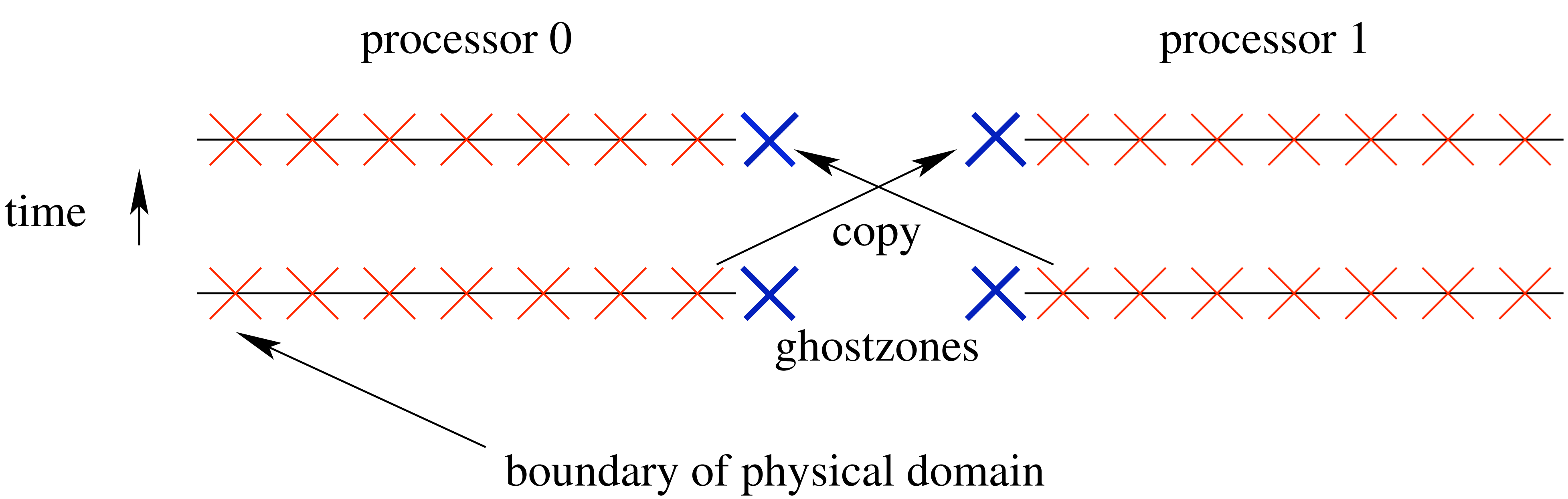
Parallelism Details

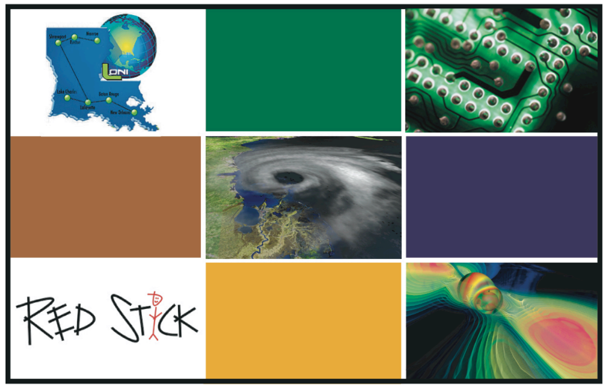
- User code sees a subset of grid points only (those on the local node)
- Driver introduces <u>ghost zones</u> for nearest neighbour communication
- Ghost zone exchange is performed by driver, as specified in schedule.ccl
- OpenMP can be used as well (hybrid communication model)



Ghost Zone Mechanism







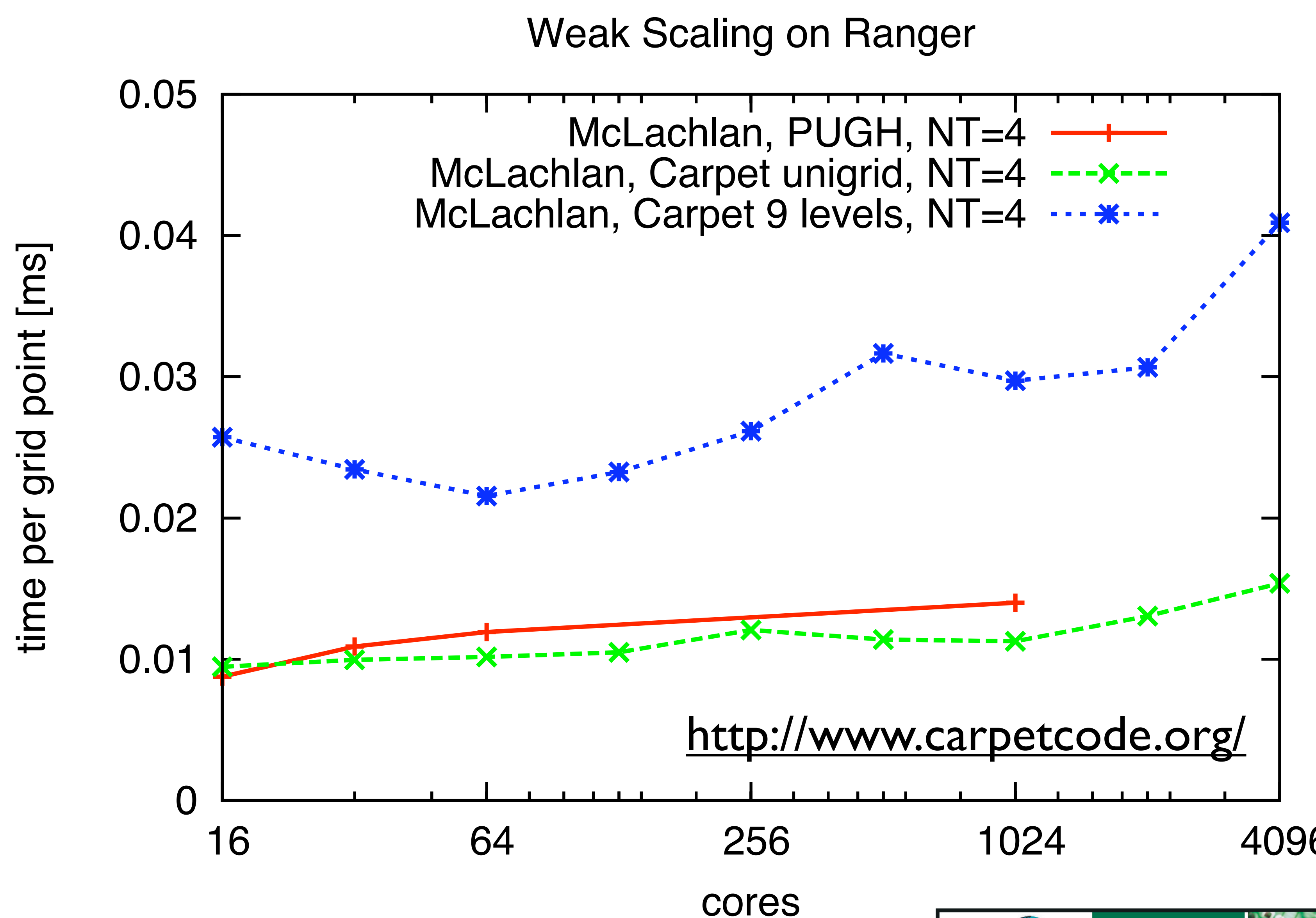
Parallelism Details

- Assume you have a large 3D array `rho(i,j,k)` that should be distributed over all processors:
- Cactus variable `cctk_gsh` ("global shape") contains total number of grid points
- `cctk_lsh` ("local shape") contains processor-local number of grid points
- `cctk_nghostzones` contains size of overlap region

Thorn LoopControl

- Can use a special thorn <u>LoopControl</u> to iterate over grid points (via C macros)
- Adds OpenMP parallelisation
- Adds loop tiling, a cache optimisation
- Parallelisation and tiling parameters are optimised and adapted at run time to improve performance

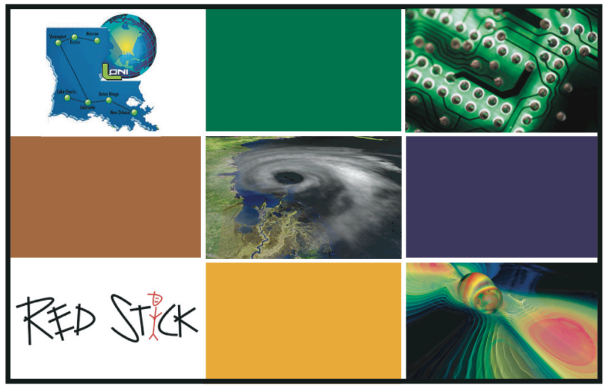
Scaling Efficiency



Solving Einstein equations with mesh refinement, combining MPI and OpenMP

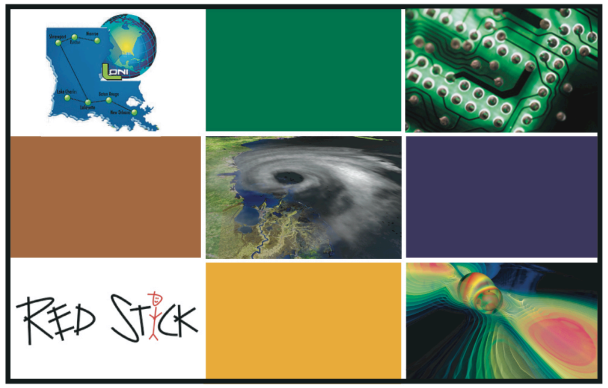
Parallel I/O

- I/O can become bottleneck when running on many processors
- Cactus I/O thorns use HDF5, a portable, standardised, efficient binary file format
- I/O routines have been heavily optimised
- Different machines and file systems require different I/O strategies for efficiency



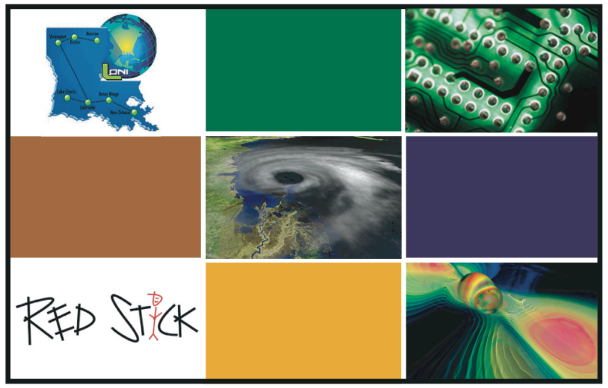
Outline

1. What is a software framework?
2. How to use Cactus
3. Parallel programming
4. **Additional tools for Cactus users**
5. Further documentation



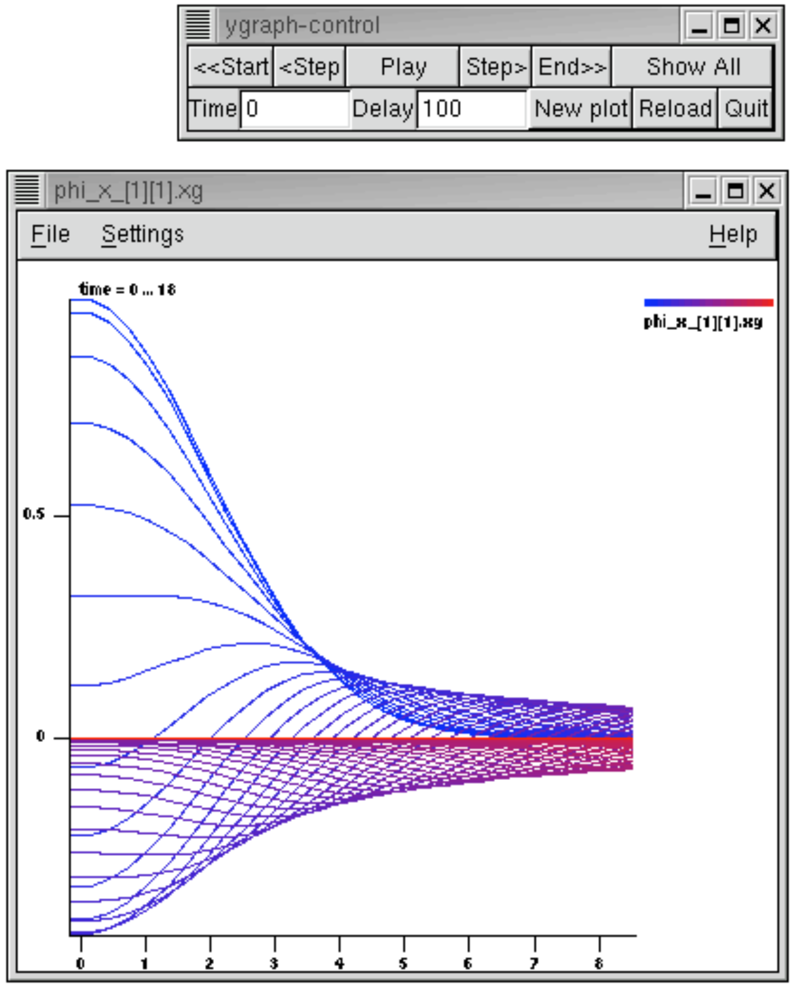
BBH Factory (SimFactory)

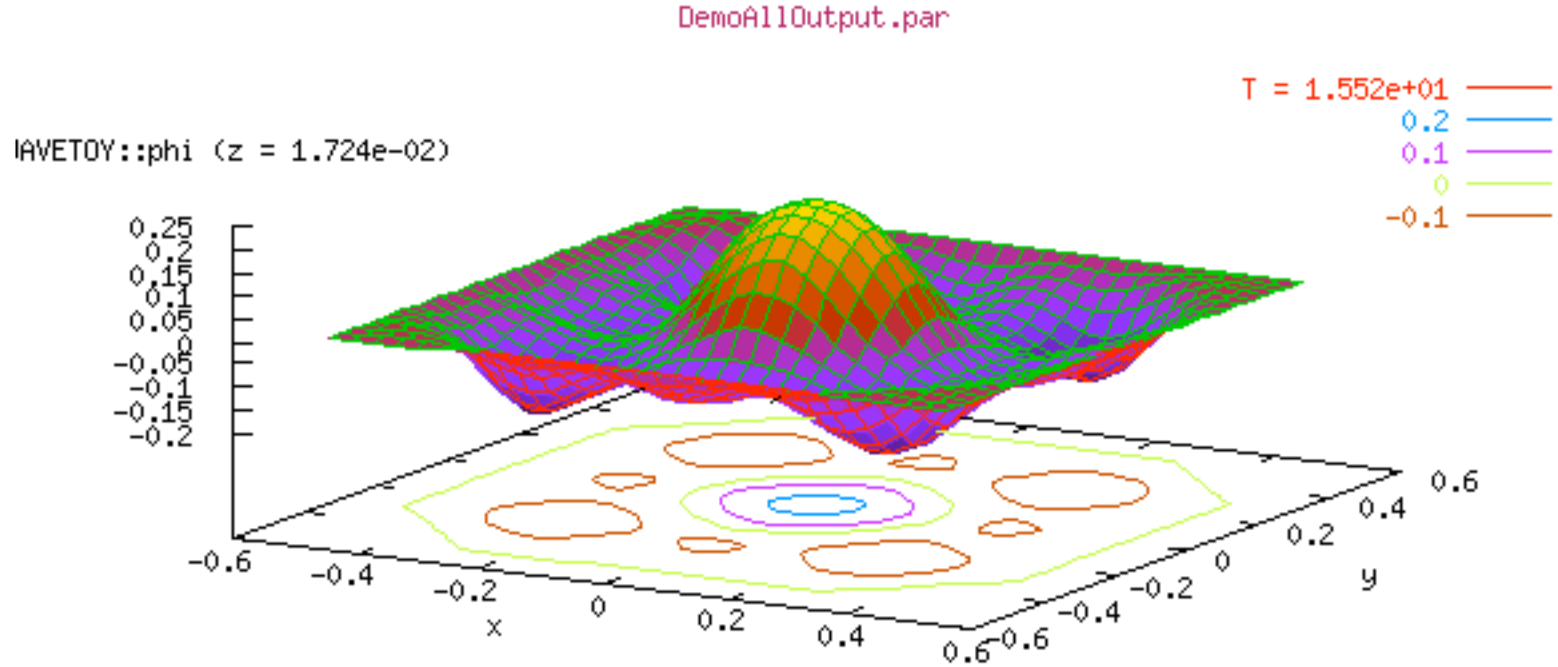
- When running many simulations, it is tedious to keep track of their output
- When using different machines (laptop, workstation, Tezpur, Queen Bee, ...), it is tedious to keep source code consistent
- Submitting simulations is different on every machine
- BBH Factory offers commands to simplify these tasks

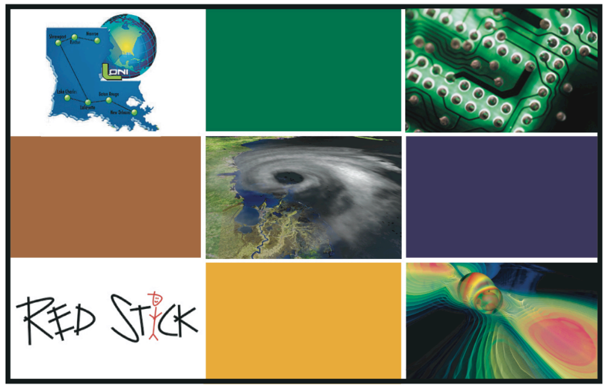


2D Visualisation

- See http://www.cactuscode.org/Visualization/
- ygraph, gnuplot



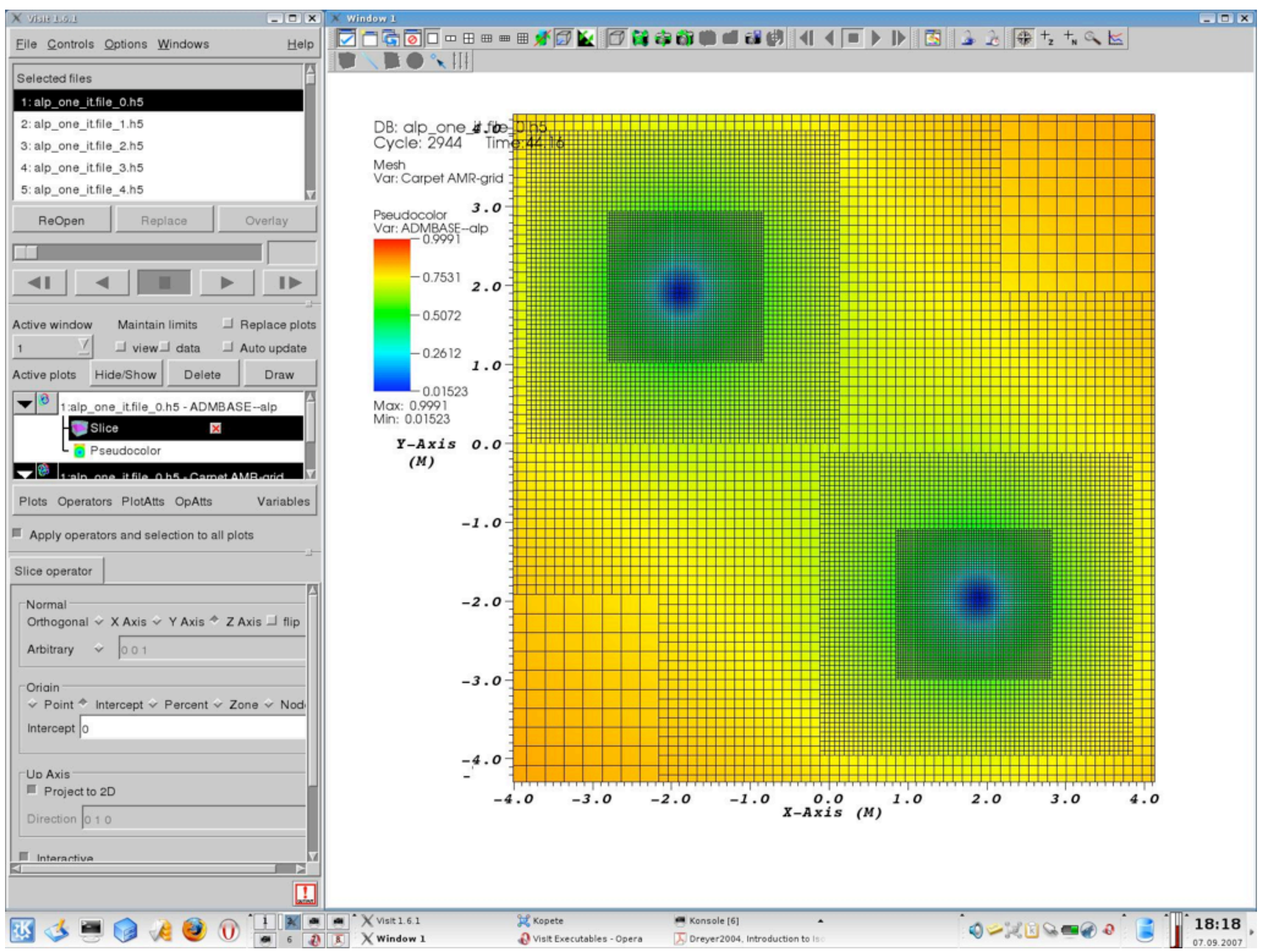


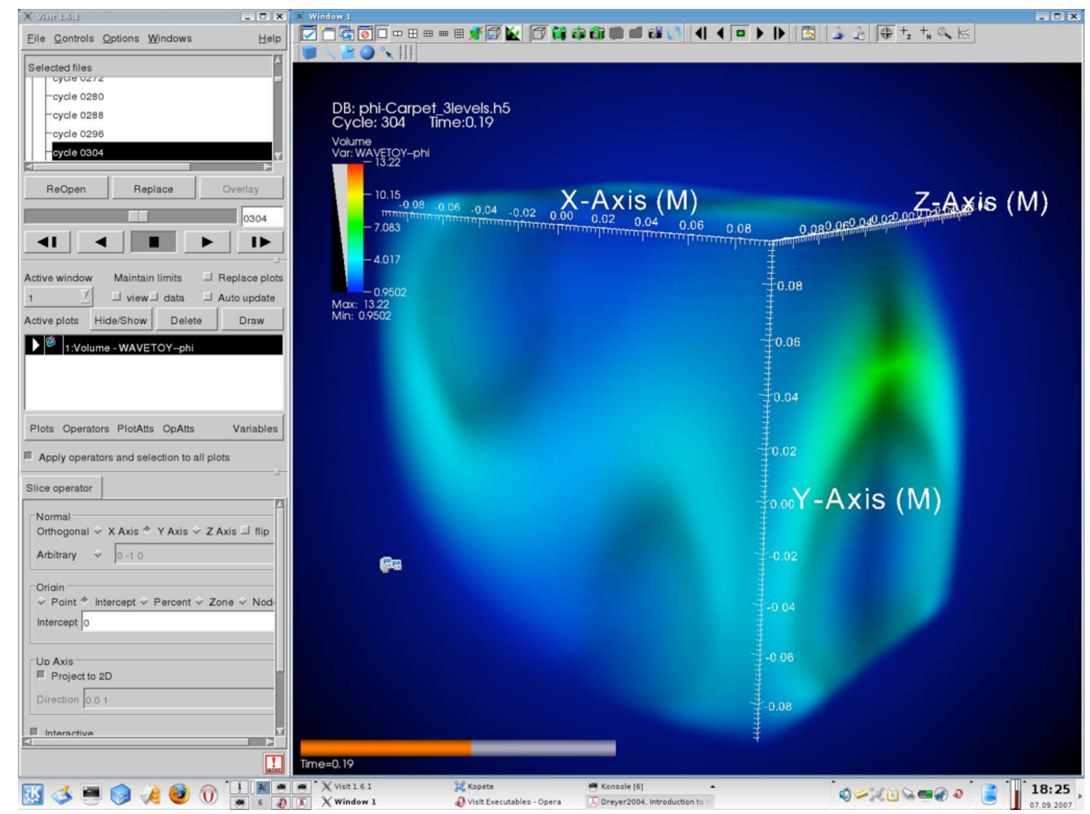


CENTER FOR COMPUTATION
& TECHNOLOGY

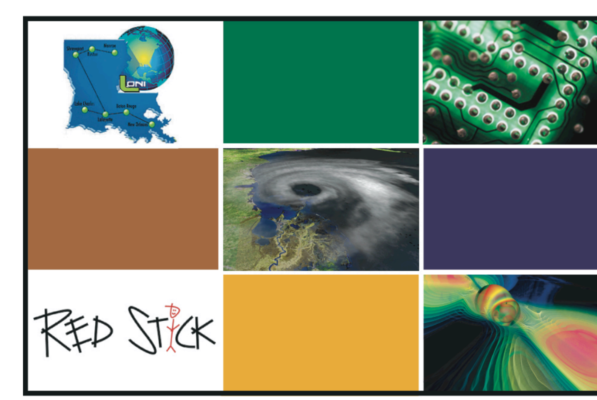
3D Visualisation

- Amira (commercial), OpenDX (difficult), VisIt



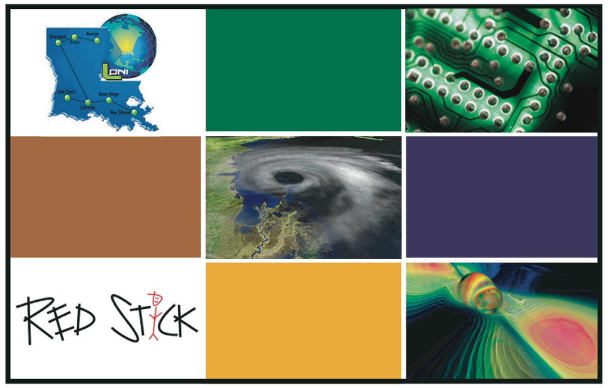


VisIt



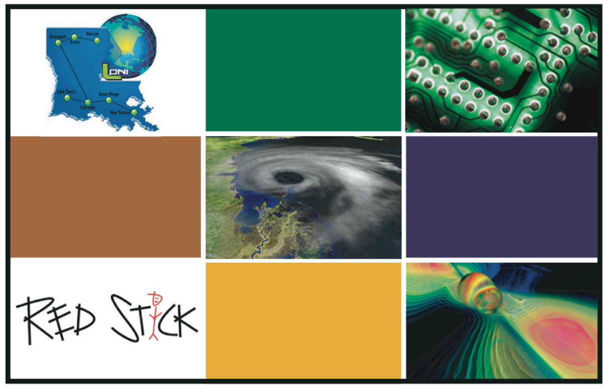
Remote Visualisation

- Usually, data reside on a supercomputer, and they need to be displayed locally
- Can copy output files, but that is slow, cumbersome, and requires local disk space
- Can use remote visualisation instead: VisIt or other tools can fetch data from remote machine automatically



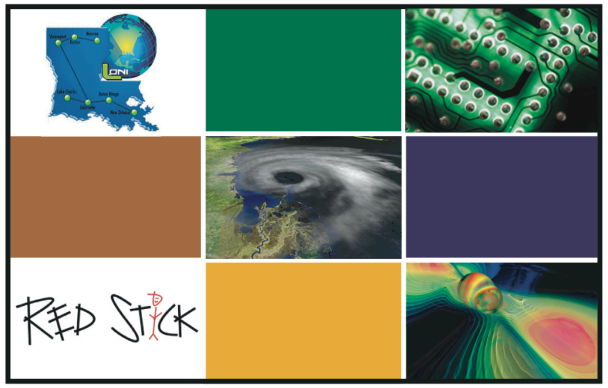
Built-in Web Server

- Thorn HTTPS implements a simple web server
- Can be used to monitor simulation as it is running, look at variables
- Can be used to change parameters (e.g. output frequency), abort simulation
- Live demo: http://www.cactuscode.org:5555/



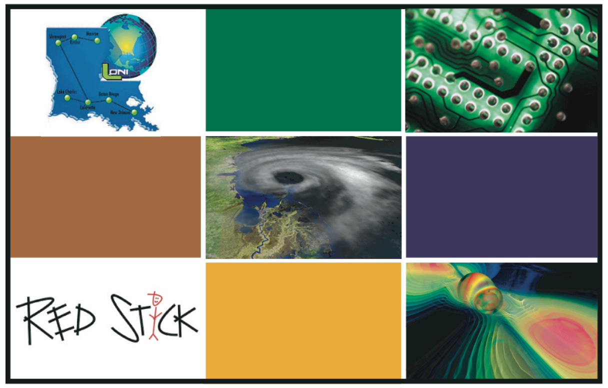
Correctness, Reproducibility, Provenance

- Thorn Formaline can conserve the source code for each simulation (compressed)
- Works automatically, never again lose the source code for a certain result
- Keep the source code for all publications for months or years
- Tags all output files with the source code version



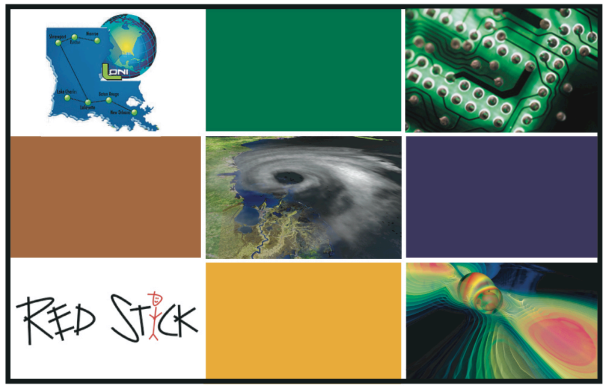
Automated "Lab Books"

- Thorn Formaline can keep a log of important "events" during a simulation
- Can announce progress to an information service (database)
- Can monitor simulations on automatically generated web pages (portal)
- http://portal.aei.mpg.de/



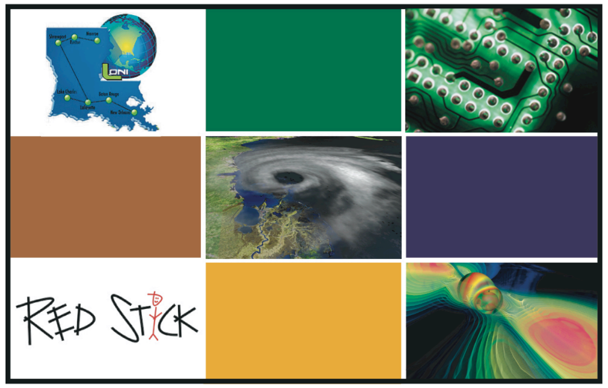
Automated Code Testing

- Cactus thorns can contain test cases
- Can be run on demand after changing code
- We run all test cases every night to ensure problems are detected
- Results displayed on a web site (portal)
  http://portal.aei.mpg.de/



XiRel, Cyberinfrastructure for Numerical Relativity

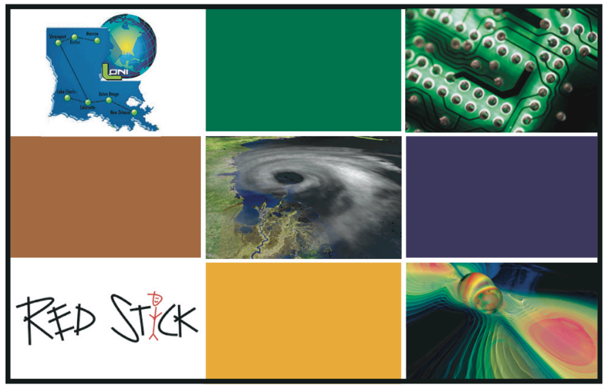
- XiRel is an international collaboration of four numerical relativity groups
- Main goals: improve performance on large machines, improve Einstein Toolkit, improve reliability and reproducibility of scientific results
- NSF funded (3 years), PI G. Allen



LSU
CENTER FOR COMPUTATION
& TECHNOLOGY

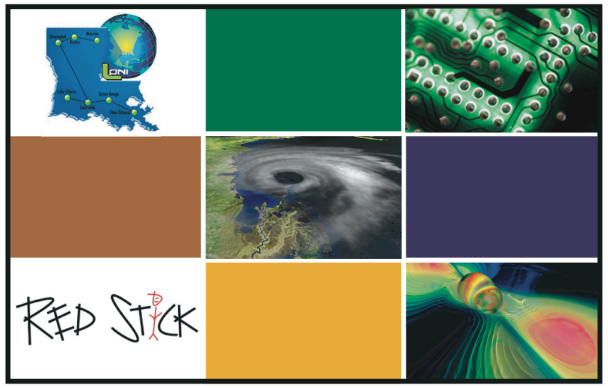
Alpaca, Application level profiling and correctness analysis tools

- Alpaca is a project to research tools for high-level (“application level”) debugging and performance optimisation
- Main goals: make debugger or profiler aware of the information that the framework has about the application
- NSF funded (3 years),
  PI E. Schnetter



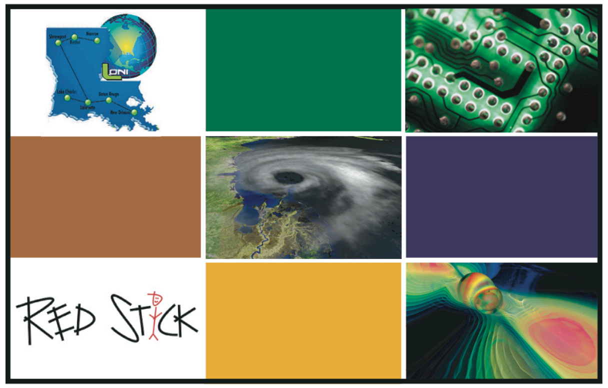
Outline

1. What is a software framework?
2. How to use Cactus
3. Parallel programming
4. Additional tools for Cactus users
5. **Further documentation**



Web Pages etc.

- Cactus: http://www.cactuscode.org/
- Carpet mesh refinement driver:
  http://www.carpetcode.org/
- Mailing lists: see web pages, especially
  <users@cactuscode.org>
- People:
  Gabrielle Allen, Erik Schnetter



Documentation

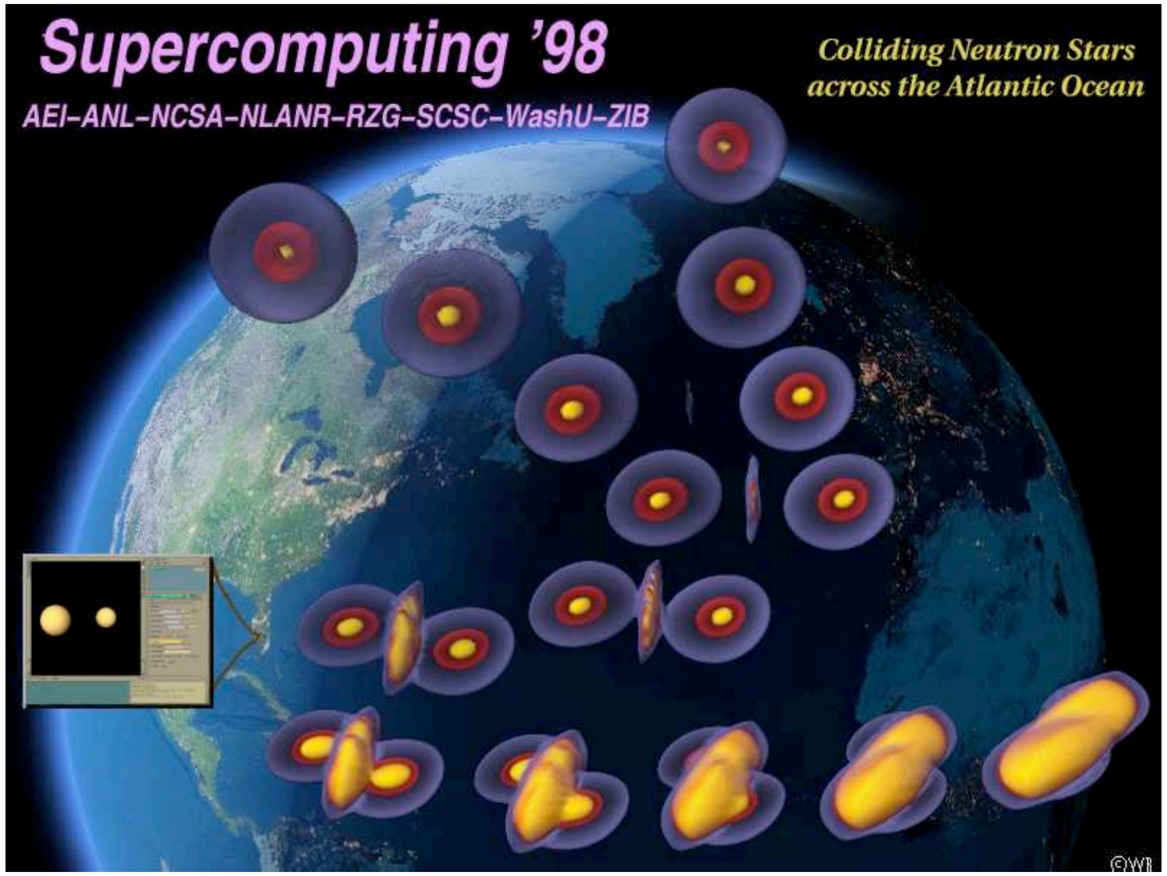
- Documentation is included in checkout
- Also on the web: http://www.cactuscode.org/Documentation/

Tutorials

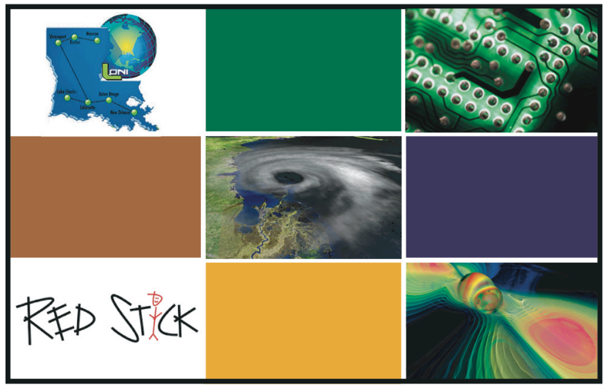
- Scalar wave equation example: http://www.cactuscode.org/WaveToyDemo/
- CactusEinstein workshop (2006): http://wiki.cct.lsu.edu/numrel/space/Events/CactusEinstein_2006

Prizes

Cactus team won many prizes, e.g. prestigious IEEE Sidney Fernbach award for CCT director Ed Seidel (2006)



http://www.cactuscode.org/About/Prizes



Why Cactus?

- Checkpointing
- Efficient parallelisation
- Efficient I/O in standard file formats
- Visualisation tools
- Goodies (Formaline, LoopControl, BBH Factory)

- Biggest payoff does not come from framework itself, but from what becomes possible once a framework is used