Cactus 4.0 : An Introduction and Perspectives On Future Plans

Tom Goodale
goodale@cct.lsu.edu

www.CactusCode.org

What Is Cactus

- Cactus is a framework for developing portable**,** modular applications, in particular, although not exclusively, high-performance simulation codes.
- Cactus is designed to allow experts in different fields to develop modules based upon their expertise and to leverage off modules developed by experts in other fields to perform their work, with minimal knowledge of the internals or operation of the other modules.
- This enables it to be used in large, geographically dispersed, collaborations.
- Cactus and the Cactus Computational Toolkit are Open Source and freely available.

(*) Any Unix-like machine plus Windows

Cactus isn't the only framework

- Many framework-like tools developed over the last few years
- POOMA
- Overture
- SAMRAI
- KeLP
- PETSc
- Common Component Architecture
- ...

POOMA

- Parallel Object-Oriented Methods and Applications
- www.acl.lanl.gov/PoomaFramework
- Object oriented framework for applications in computational science
- Library of c++ classes
- Provides 'data parallel programming at the highest abstraction layer'

Overture

- www.llnl.gov/CASC/Overture
- Object-oriented tools for solving CFD and combustion problems in complex moving geometries
- C++ library
- Structured finite difference or finite volume methods
- Multiblock and AMR
- Has OpenGL visualisation tools
- Suns, SGI, Linux

SAMRAI

- Structured Adaptive Mesh Refinement Applications Infrastructure
- www.llnl.gov/CASC/SAMRAI
- Parallel structured AMR
- Supports Fortran
- C++ class libraries

KeLP

- Kernel Lattice Parallelism
- www-cse.ucsd.edu/groups/hpcl/scg/kelp
- Structured irregular applications - e.g AMR, multigrid
- C++ class library
- Needs MPI

Cactus Goals

- Portable
  - Must be able to compile and run on any platform we want to run on
- Modular
  - People should be able to write modules which interact through standard interfaces with other modules without having to know the internals of the other modules
  - Modules with same functionality should be interchangeable

Cactus Goals

- Support legacy codes
  - Old codes must be able to become modules without significant changes
  - Must support Fortran 77 !
  - Should have data structures compatible with old codes
- Make use of existing technologies and tools where appropriate

Cactus Goals

- Future proof
  - Must not be tied to any particular paradigm
  - Parallelism should be independent but compatible with currently available systems such as MPI or PVM
  - IO system should be independent of but compatible with currently available systems such as HDF
  - Maintainable

Cactus Goals

- Easy to use
  - If not, it won't be used
  - Must be well documented for users and developers
  - Should allow people to program as before.
- Maintainable
  - Internals should be cleanly written
  - Internals should be documented

History

- Cactus was first developed in 1997 by Paul Walker, Joan Masso and others as a continuation of a long line of numerical relativity codes, such as the NCSA G-code and Paul's Framework.
- Through the first couple of years Cactus became progressively more modular, allowing modules for different formulations of Einstein's equations and different physical systems.
- Although in principle Cactus was modular, its history and evolution had left many dependencies between modules and between the core and the modules. Cactus 4.0 is a complete redesign of the core, which moved everything possible out into modules, and put structures in place to enable modules to enable modules to be far more independent.
- Cactus 4.0 builds upon the lessons learned in earlier versions, and incorporates our vision of a modular and hopefully future proof framework.

Current Users

- Numerical Relativity
  - Used by many groups including:
    - AEI(Germany),UNAM (Mexico), Tuebingen(Germany), Southampton (UK), Sissa(Italy), Valencia (Spain), University of Thessaloniki (Greece), MPA (Germany), RIKEN (Japan), TAT(Denmark), Penn State (USA), University of Texas at Austin (USA), University of Texas at Brwosville (USA), LSU (USA), WashU (USA), University of Pittsburg (USA), University of Arizona (USA), Washburn (USA), UIB (Spain), University of Maryland (USA), Monash (Australia)
  - Over fifty users on mailing list
  - See http://www.appleswithapples.org
- Climate Modelling
- CFD
  - KISTI
  - DLR looking at flow in turbines

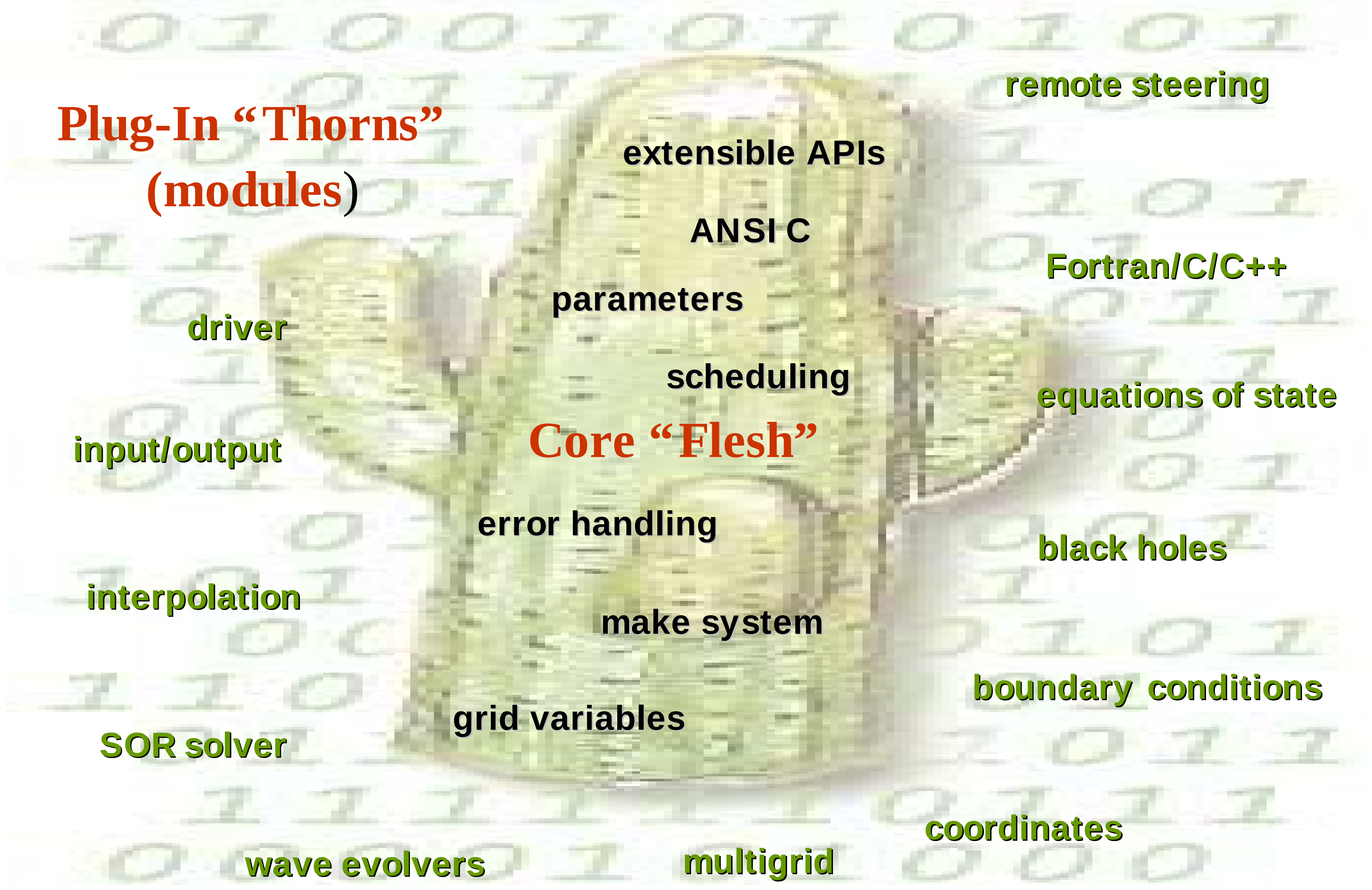
Current Users...

- Astrophysical Hydrodynamics
  - Mike Norman's Zeus code is now in Cactus
- Chemical Reaction
- Bioinformatics
- Plasma Physics
- Quantum Gravity
- ...
- Used by many CS projects ...
  - GridLab
  - GrADS
  - TeraGrid

Structure

- The source code of Cactus consists of a core part – the “ Flesh” and a set of modules called “ thorns” .
- The Flesh is independent of all thorns and after Cactus is initialised, it generally acts as a utility and service library which the thorns call to get information or ask for some action to happen.
- Thorns are separate libraries which encapsulate some functionality. In order to keep a distinction between functionality and implementation of the functionality, each thorn declares that it provides a certain “ implementation” . Different thorns can provide the same “ implementation” , and thorn dependencies are expressed in terms of “ implementations” rather than explicit references to thorns, thus allowing the different thorns providing the same “ implementation” to be interchangeable.

Structure


Plug-In “Thorns” (modules)
remote steering
extensible APIs
ANSI C
Fortran/C/C++
parameters
driver
scheduling
equations of state
Core “Flesh”
input/output
error handling
black holes
interpolation
make system
boundary conditions
grid variables
SOR solver
coordinates
wave evolvers
multigrid

Cactus Flesh

- Make System
  - Organises builds as *configurations* which hold everything needed to build with a particular set of options on a particular architecture.
- API
  - Functions which must be there for thorns to operate.
- Scheduling
  - Sophisticated scheduler which calls thorn-provided functions as and when needed.
- CCL
  - Configuration language which tells the flesh all it needs to know about the thorns.

Make System

- Designed to allow same source checkout to be used to build on various architectures and for various compilation options on one machine.
- Compilation options grouped into **configuration time** options and **compile time** options.
    - Configuration time - compilers, optimisation and debugging flags, etc.
    - Compilation time - warnings, verbosity of compilation, etc.
- Each configuration has a **ThornList** which lists the thorns to be compiled in. When this list changes, only those thorns directly affected by the change are recompiled.

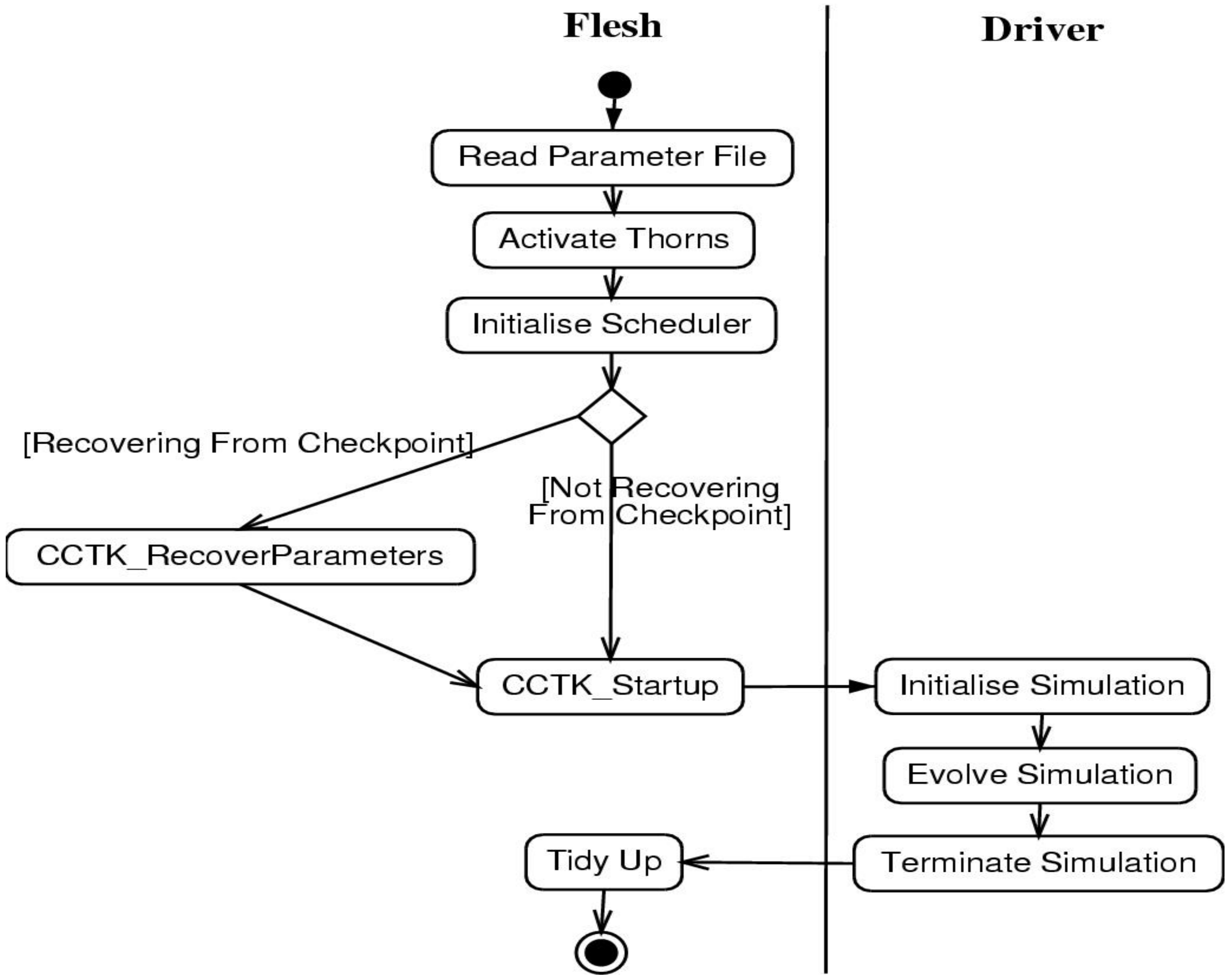
Thorn Specification

- The Flesh finds out about thorns by 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
  - Scheduling directives
    - The flesh incorporates a scheduler which is used to call defined routines from different thorns in a particular order.
  - Interface definitions
    - All variables which are passed between scheduled routines need to be declared.
    - Any thorn-provided functions which other thorns call should be declared.
  - Parameter definitions
    - The flesh and thorns are controlled by a parameter file; parameters must be declared along with their allowed values.

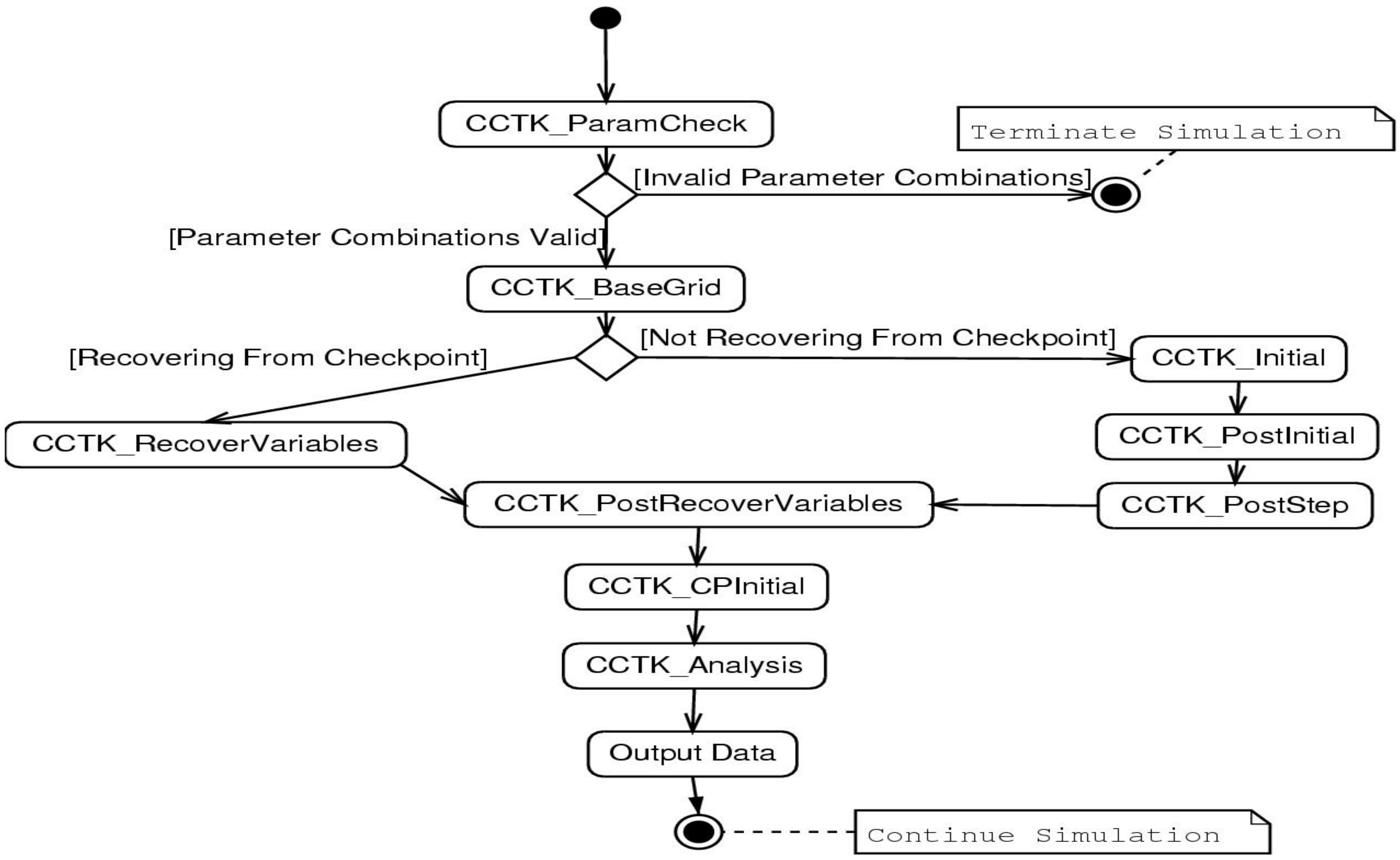
Scheduling

- Thorns specify which functions are to be called at which time, and in which order.
- Rule based scheduling system
- Routines are either **before** or **after** other routines (or don't care).
- Routines can be grouped, and whole group scheduled.
- Functions or groups can be scheduled **while** some condition is true.
- Flesh sorts all rules and flags an error for inconsistent schedule requests.

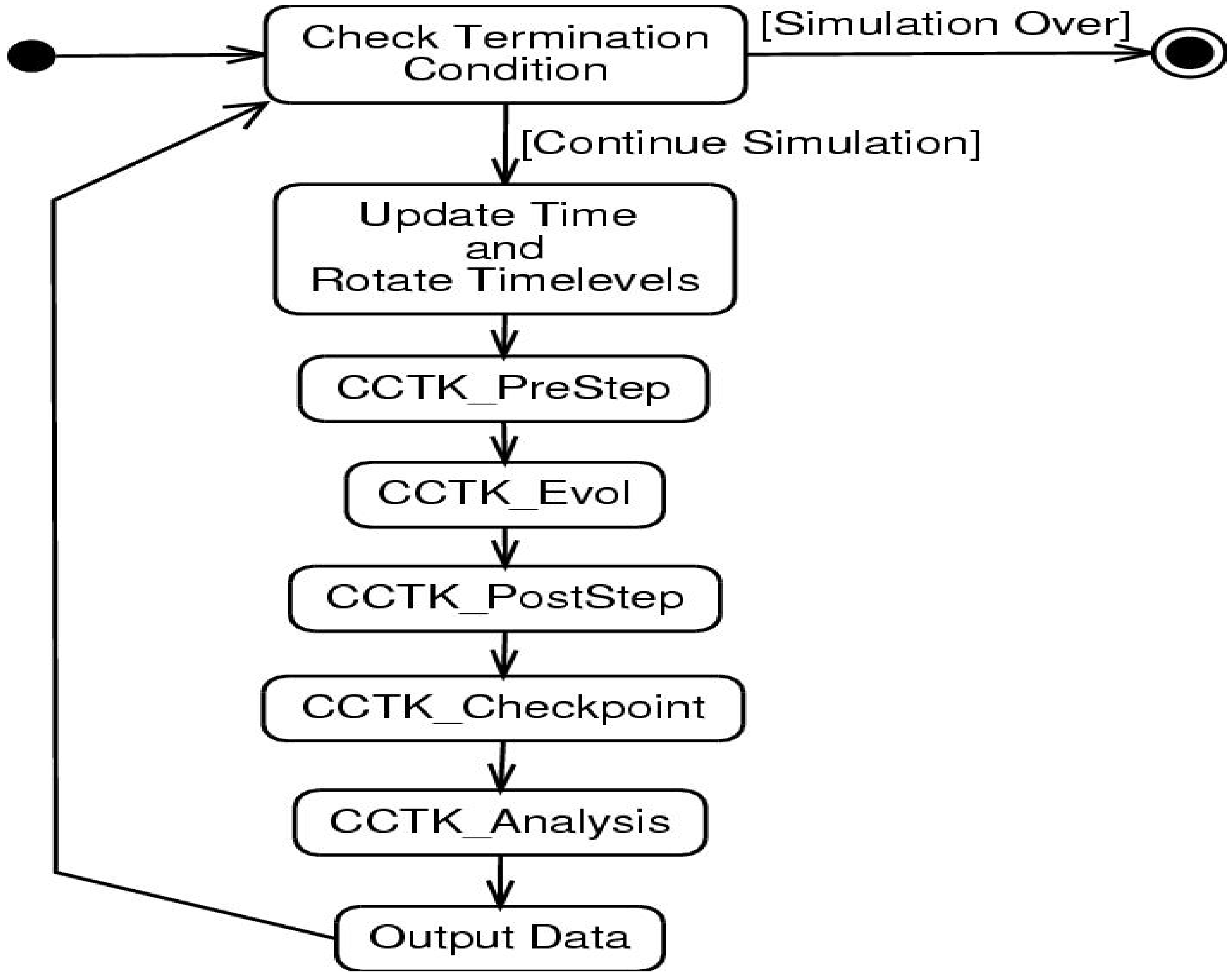
Program Flow



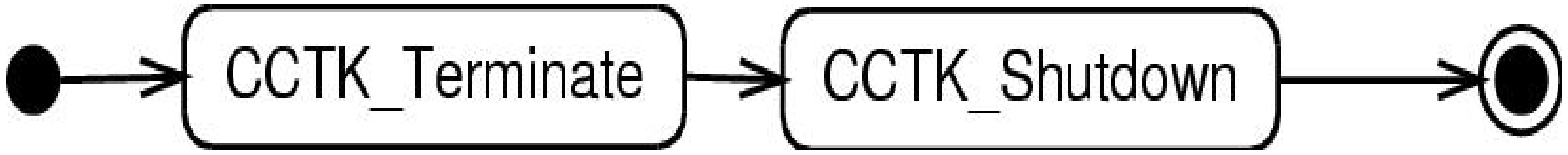
Simulation Initialisation



Evolution


Check Termination Condition
[Simulation Over]
[Continue Simulation]
Update Time and Rotate Timelevels
CCTK_PreStep
CCTK_Evol
CCTK_PostStep
CCTK_Checkpoint
CCTK_Analysis
Output Data

Termination



The Driver Layer

- In principle drivers are the only thorns which know anything about parallelism
- Other thorns access parallelism via an API provided by the flesh
- Underlying parallel layer could be anything from a TCP-socket to Java RMI. It should be transparent to application thorns.
- Could even be a combination of things.
- Can even run with no parallel layer at all.

- Can pick actual driver to use at runtime - no need to recompile code to test differences between parallel layers. Can take one executable and use whatever the best layer for any particular environment happens to be.

Current Drivers

- There are several drivers available at the moment, both developed by the cactus team and by the community.
- **PUGH**
  - a parallel uni-grid driver, which comes as part of the the computational toolkit
- **PAGH**
  - a parallel AMR driver which uses the GrACE library for grid hierarchy management
- **Carpet**

  a parallel fixed mesh refinement driver
- **SimpleDriver**
  - a simple demonstration driver which illustrates driver development
- Discussions and plans to add others, e.g. Paramesh and Chombo as other drivers.
  - Clear interfaces to follow

Cactus Computational Toolkit

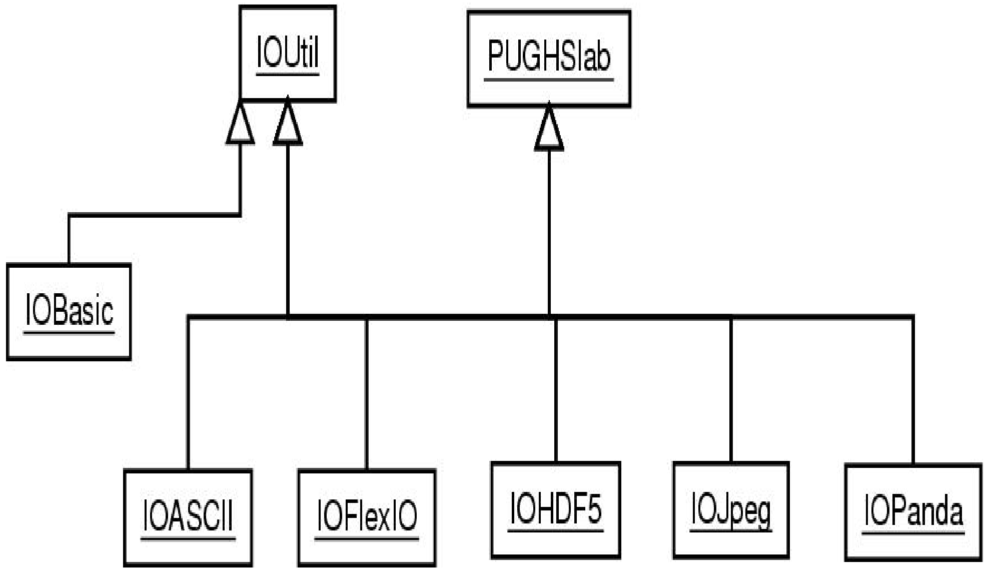
- Core thorns which provide many basic utilities, such as:
  - Boundary conditions
  - I/O methods
  - Reduction and Interpolation operations
  - Coordinate Symmetries
  - Parallel drivers
  - Elliptic solvers
  - Web-based interaction and monitoring interface
  - ...

Cactus Computational Toolkit

- CactusBase
  - Boundary, IOUtil, IOBasic, CartGrid3D, IOASCII, Time, LocalInterp
- CactusBench
  - BenchADM, BenchIO
- CactusConnect
  - HTTPD HTTPDExtra, Socket
- CactusExamples
- CactusElliptic
  - EllBase, EllPETSc, EllSOR, EllTest
- CactusPUGH
  - PUGHInterp, PUGH, PUGHReduce, PUGHSlab
- CactusPUGHIO
  - IOFlexIO, IOHDF5, IOPanda, IOStreamedHDF5, IsoSurfacer
- CactusIO
  - IOJpeg
- CactusUtils
  - NANChecker
- CactusWave
  - IDScalarWave, IDScalarWaveC, IDScalarWaveCXX, WaveBinarySource, WaveToyC, WaveToyCXX, WaveToyF77, WaveToyF90, WaveToyFreeF90
- CactusExternal
  - FlexIO, jpeg6b

Current Capabilities: IO

- Support for IO and checkpointing in many different formats
  - Basic screen output of norms
  - 2-d slices as jpegs.
  - n-d ASCII data suitable for x/y-graph or gnuplot.
  - n-d data in John Shalf's IEEEIO format.
  - n-d data in HDF5 format, which may be written to disk or streamed to visualisation clients or other simulations.
  - Output using the Panda software from UIUC.



Current Capabilities: Grids, Boundaries, Symmetries, etc

- Cactus currently supports data on structured meshes. These meshes can either be unigrid, or can be adapted in either fixed mesh refinement or adaptively.
- The grid can be restricted to octants, quadrants or “ bitants” .
- The thorns provided with Cactus support many boundary conditions, e.g. copy, radiative, fixed, etc.
- Periodic boundary conditions have been supported since version 1 (contrary to rumour). There are thorns to support Cartoon boundaries and Rotational symmetries (in development), and we are working on making the symmetries completely transparent to simulation codes.

Current Capabilities: Methods

- Almost all codes in Cactus at the moment are explicit finite difference codes on structured meshes.
- In principle finite volume or finite element on structured meshes is possible.
- There is now a generic method-of-lines thorn which makes developing thorns using such methods very quick and easy.
- We have an interface for elliptic solvers and support for generic elliptic solver packages such as PETSc as well as a numerical-relativity-specific multigrid solver written by Bernd Bruegmann. However our interface is not as generic as it could be, and it may not be too useful as it stands for solving general implicit problems.
  - A new interface has been proposed in the last couple of weeks which would apply a similar methodology as is used for boundary conditions for elliptic equations.

Current Capabilities: Interaction

- The HTTPD thorn provides an interface which allows a web-browser to connect to a running simulation
- This allows a user to examine the state of the running simulation and change certain parameters, such as the frequency of Io or the variables to be output, or in fact any parameter which some thorn author has declared may be changed during the simulation.
- These capabilities may be extended by any other thorn. E.g. the HTTPDExtra thorn allows the user to download any file output by the IO thorns in the Computational toolkit, and even to view two-dimensional slices as jpegs.
  - There is also a helper-script for web-browsers which allows the appropriate visualisation tool to be launched when a user requests a file.

Current Capabilities: Visualisation

- The output from the Computational Toolkit IO thorns can be visualised by many clients, such as: Amira, OpenDX, GnuPlot, Xgraph, Ygraph, ...
- There is currently work from the climate modelling community to add IO thorns for the NetCDF format and this will bring in a new set of possible visualisation clients.
- The IsoSurfacer thorn calculates isosurfaces of a variable in parallel and may stream the data out to a suitable client.
- We provide one, **IsoView**, but it is an open format which can be used by other clients, e.g. **Amira**.
- Lots of information on web pages, including binaries for various visualisation tools and HOWTOs for setting these tools up and using them.

Ongoing development

Much ongoing development work:

- Generic optimisation
  - timing and profiling information
  - Working with NCSA, Intel, LBL, ...
- Associated scripts
  - checkout
  - remote testing
  - ...
- Task farming and parameter searches
- Grid applications
- Describing tensors or other geometric quantities in a generic manner for grid variables.

Case Study: Framework for Einstein's Equations

- In the last year we had intensive discussions between various groups about how best to support various different approaches to solving Einstein's equations within the Cactus Framework.
- Result was new revised CactusEinstein arrangement which is not tied to any particular approach.
- The core of this new arrangement is one thorn **ADMBase** which defines the basic ADM variables which all thorns operate on or with, or at least use to communicate with other thorns.
- The ADMBase thorn is the only one required for inter-operability between different formulations, however there are another four auxiliary thorns which may be used to enhance the functionality:
  - **CoordGuage, SpaceMask, StaticConfomal and ADMCoupling**

Einstein: The Auxiliary Thorns

- All the auxiliary thorns are independent of each other and are optional to the use of the arrangement
- CoordGauge
  - This is used to schedule combinations of gauge conditions for the ADM lapse and shift.
- SpaceMask
  - This may be used to store and communicate information about any masks which affect areas of the computation
- StaticConformal
  - Many simulations involve a particular static conformal factor. This thorn provides this conformal factor and (optionally) its derivatives.
- ADMCoupling
  - This may be used to share variables between metric evolvers and stress-energy evolvers (e.g. hydro, scalar-field, e-m field, ...)

Einstein: Other Thorns

- Apart from ADMBase and the auxiliary thorns, the CactusEinstein arrangement contains a set of thorns which illustrate its use. These thorns are also used in real production runs.
  - An Einstein metric evolver
  - Initial data for black holes and gravitational waves
  - An apparent horizon finder
  - Gravitational wave extraction
  - Analysis tools to find the Newman-Penrose quantities of a space-time.
  - Various other analysis tools to determine such things as the hamiltonian and momentum constraints, the Ricci and Riemann tensor components, ...

Future Plans

- Release 4.0 final !
  - Finish the remaining outstanding features such as function aliasing, better coordinate and symmetry handling, driver interface enhancements, ...
  - Squash bugs.
  - Finish Documentation.
- Then start work on 4.1 (we will use a linux-like release numbering system from this point).
- We have a fair number of features we'd like to put in in 4.1:
  - More numerical methods
  - Infrastructure enhancements
  - Connection to other frameworks
- Please send us ideas/suggestions too !

More Numerical Methods

- Support unstructured meshes
  - Allow much greater ability to use FE and FVM.
- Better support for “ multi-model” - i.e. multi-block, multi-patch, and multi-domain – simulations.
  - Can be done now, but requires a lot of detailed knowledge of the infrastructure and for the code to be structured differently.
  - We'd like it to be easy to develop each model separately and then plug them together.
- Support particle methods such as particle in cell (PIC) and smoothed particle hydrodynamics (SPH).
- Support for spectral methods.

More Dynamic Infrastructure

- Dynamic loading and unloading of thorns
  - Currently thorns can be either inactive or active, and thorns may only be activated while the parameter file is being parsed. We'd like to add support to activate or de-activate at any time, and support for thorns to be dynamically loadable libraries which would thus enable new capabilities to be loaded into long-running simulations, or bugs in existing capabilities fixed and then reloaded without terminating the simulation.
- Scripting as an alternative or in co-operation with the scheduler
  - Currently the scheduler is rule-based, much like a make system.
  - For some applications and users it may be more natural to explicitly script which actions happen when, although this then requires more knowledge of the thorns than is currently needed to run Cactus.

Other Infrastructure Enhancements

- Support for more programming languages
  - Currently support C, C++, FORTRAN 77 and Fortran 90
  - Would like to add infrastructure for generic language support and in particular for thorns written in Java, Perl, Python, ...
- Cactus Communication Infrastructure
  - Cactus supports a small number of parallel operations at the moment, and these are provided by the “ driver” thorn or its helpers.
  - Would like to support more parallel operations and perhaps allow drivers to be built on top of such operations; thus allowing drivers to be independent of underlying parallel infrastructure.
- Sub-grid-variables
  - Variables which are only distributed across faces of the computational grid.
  - These could be used, for example, for Cauchy-Characteristic Matching.

Other Infrastructure...

- Scheduling of aliased functions
  - Aliased functions are currently provided for inter-thorns calling, but in 4.1 we want to make it possible to call such functions from the scheduler
- Arrangement paths
  - Currently all arrangements have to be in the arrangements directory of the Cactus checkout (or be symlinked there). In 4.1 it will be possible to have arrangements in many places.
  - This will also allow the central 'installation' of a Cactus checkout/tarball.

Connection To Other Frameworks

- There are many other frameworks for simulations
  - IEEE1516 High level Architecture
  - CCA
  - SciRun
  - PALM
  - OASIS
  - MpCCI
  - Overture
  - ...
- Want to investigate and work on interoperability between the Cactus Framework and thorns and these frameworks.

Other Things

- Central Thorn Information Page
  - Would like to have a page listing all publicly available thorns with brief descriptions and a URL to more information.
- Newsletter and Community Pages
  - Currently it is not obvious what different disciplines use Cactus, we'd like to produce a newsletter with news from communities and have more comprehensive pages describing the communities.
- Increase community involvement in use and development of code, and particularly of problem domain specific thorns or arrangements, such as the Einstein arrangement.

Cactus Summary

- Cactus is a powerful framework for developing portable applications, particularly suited to large collaborations. I've barely scratched the surface in this talk.
  - See http://www.cactuscode.org for more information.
- Cactus is currently used by many groups around the world, in several fields, and the number of users is growing.
- Cactus 4.0 was a major revamp of the infrastructure to make it cleaner and more modular.
- Future versions of Cactus will add features which increase the range of methods and problem domains which it is suited for.