What will you learn today? I

What will you learn today? II

- Formulation
- Discretization
- Standalone Code
- Thorn BadWave
- Running BadWave
- Method of Lines

10 Mesh Refinement
- Fixed mesh refinement
- Adaptive mesh refinement

11 Kranc

12 Simfactory

13 Einstein Toolkit
- HDF5 and VisIt

14 HTTPD

15 Computation and Live Rendering on GPGPUs

What will you learn today?

Converting a serial code to a parallel ARM code using the Cactus Computational Framework

You will learn

- what the Cactus Computational Framework is
- what AMR is (in basics)
- how Cactus can help for serial codes
- how Cactus can help to parallelize codes

What do we assume?

We assume you

- are able to write, or at least read, a serial code in C/C++/Fortran;
- are interested in writing parallel codes;
- might already have a serial problem to be parallelized.

It would be advantagous (but not necessary) if you

- are familiar with the Linux/Unix environment
- are familiar with Eclipse

Tutorial Outline

Introduction to Cactus	*seminar*
Introduction to tutorial environment	**handson**
Questions and Break	
Example implementations	*seminar*
The first thorn	**handson**
Questions and Break	
Parallelizing simple Example	**handson**
Running, Visualizing	**handson**
Frontiers in Cactus Development	*seminar*
Questions and Break	
Parallelizing advanced Examples	**handson**
Wrap Up	*seminar*

What is Cactus?

Cactus is

- a framework for developing portable, modular applications

What is Cactus?

Cactus is

- a framework for developing portable, modular applications
- focusing, although not exclusively, on high-performance simulation codes

What is Cactus?

Cactus is

- a framework for developing portable, modular applications
- focusing, although not exclusively, 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

Cactus Goals

- Portable
  - Different development machines
  - Different production machines

Cactus Goals

- Portable
  - Different development machines
  - Different production machines
- Modular
  - Standard interfaces for module interaction for easier code interaction, writing and debugging
  - Interchangeable modules with same functionality

Cactus Goals

- Portable
  - Different development machines
  - Different production machines
- Modular
  - Standard interfaces for module interaction for easier code interaction, writing and debugging
  - Interchangeable modules with same functionality
- Easy to use
  - Good documentation
  - Try to let users program the way they are used to
  - Support all major (HPC) programming languages

Philosophy

- Open code base to encourage community contributions

Philosophy

- Open code base to encourage community contributions
- Strict quality control for base framework

Philosophy

- Open code base to encourage community contributions
- Strict quality control for base framework
- Development always driven by real user requirements

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

What is Cactus for?

Assume:

- Computational problem
- Too large for single machine

- Distributed development

- Multiple programming languages

What is Cactus for?

Assume:

- Computational problem
- Too large for single machine
  - OpenMP
  - MPI
- Distributed development

- Multiple programming languages

What is Cactus for?

Assume:

- Computational problem
- Too large for single machine
  - OpenMP
  - MPI
- Distributed development
  - Modularize Problem
  - Versioning system(s)
- Multiple programming languages

What is Cactus for?

Assume:

- Computational problem
- Too large for single machine
  - OpenMP
  - MPI
- Distributed development
  - Modularize Problem
  - Versioning system(s)
- Multiple programming languages
  - Modularize Problem
  - Interfaces for inter-language communication

What is Cactus for?

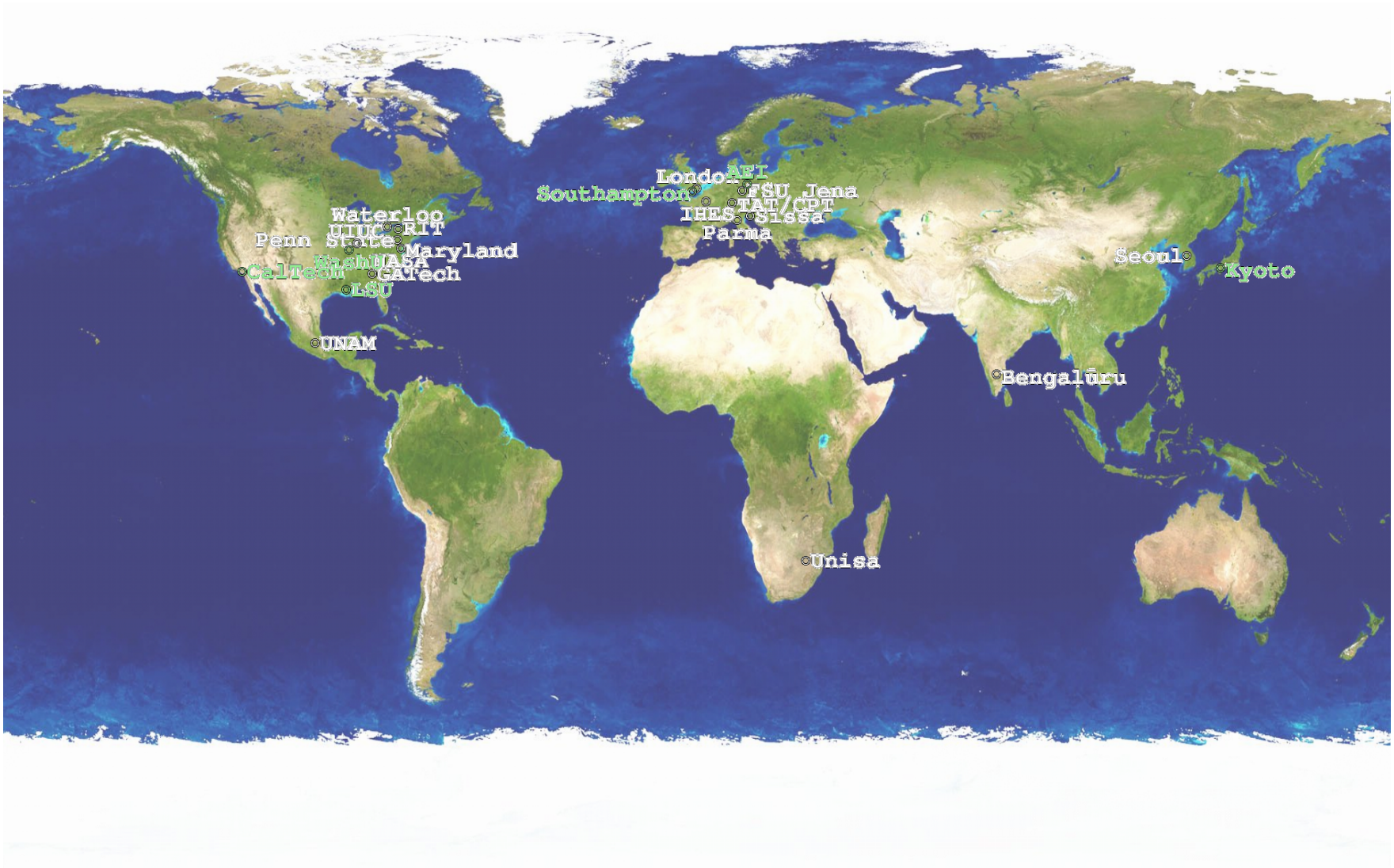
Assume:

- Computational problem
- Too large for single machine
  - OpenMP
  - MPI
- Distributed development
  - Modularize Problem
  - Versioning system(s)
- Multiple programming languages
  - Modularize Problem
  - Interfaces for inter-language communication

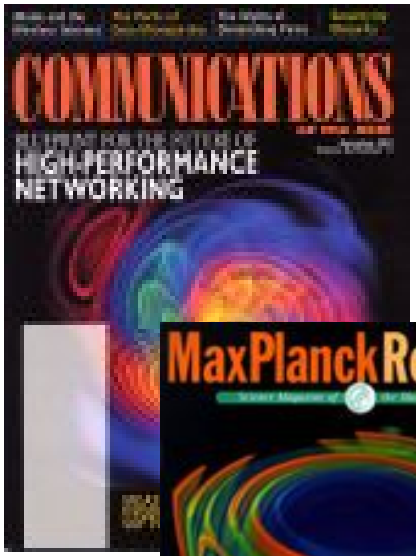
History

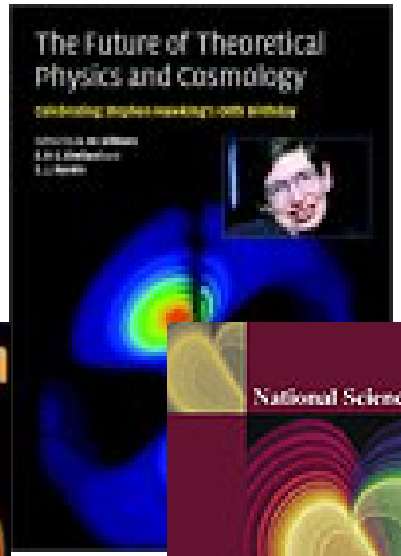
- Cactus is a direct descendant of many years of code development in Ed Seidel's group of researchers at the National Center for Supercomputing Applications (NCSA) as the group wrestled to numerically solve Einstein's Equations for General Relativity and thus model black holes, neutron stars, and boson 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.

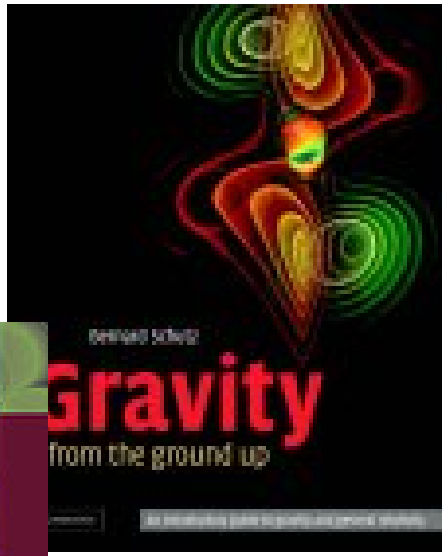
Current Users and Developers

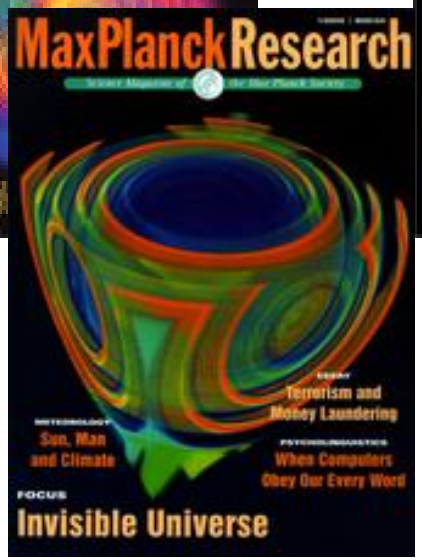


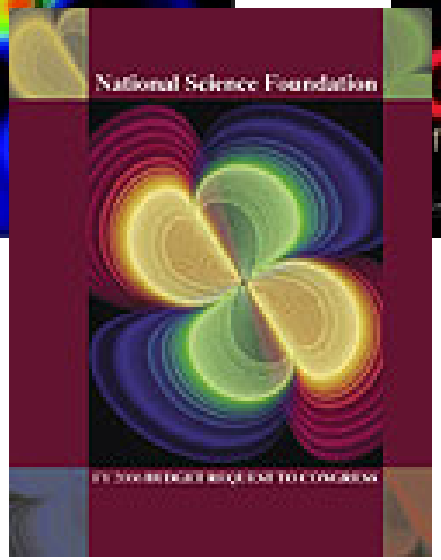
Covers

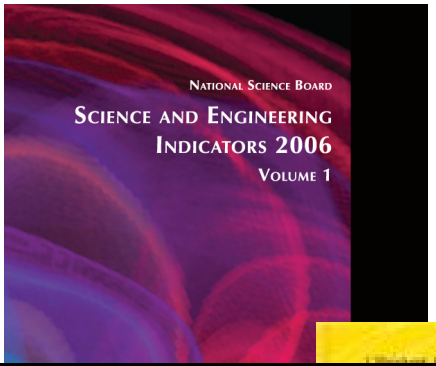

COMMUNICATIONS
HIGH-PERFORMANCE
NETWORKING


The Future of Theoretical
Physics and Cosmology


Gravity
from the ground up


MaxPlanckResearch
Terrorism and
Money Laundering
Sun, Man
and Climate
When Computers
Obey Our Every Word
FOCUS
Invisible Universe


National Science Foundation


National Science Board
Science and Engineering
Indicators 2006
Volume 1


Chapman & Hall/CRC

Cactus Funding

- Organizations:
  - Max-Planck-Gesellschaft
  - Center for Computation & Technology at LSU
  - National Center for Supercomputing Applications
  - Lawrence Berkeley National Laboratory
  - Washington University
  - University of Tübingen
- Grants:
  - NSF (PHY-9979985, 0540374, 0653303, 0701491, 0701566, PIF-0904015, 0903973, 0903782)
  - Europ. Commission (HPRN-CT-2000-00137, IST-2001-32133)
  - DFN-Verein (TK 6-2-AN 200)
  - DFG (TiKSL)
  - ONR (COMI)
  - DOE/BOR (OE DE-FG02-04ER46136, BOR DOE/LEQSF)

Cactus Awards

IEEE SCALE09 Challenge Winner	2009
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

The Flesh

- The **flesh** is the central component of Cactus. It interfaces with modular components called **thorns**. The flesh provides:

- Variables & Data Types

The Flesh

- The **flesh** is the central component of Cactus. It interfaces with modular components called **thorns**. The flesh provides:

- Variables & Data Types
- Parameters

The Flesh

- The **flesh** is the central component of Cactus. It interfaces with modular components called **thorns**. The flesh provides:

- Variables & Data Types
- Parameters
- Functions for:

The Flesh

- The **flesh** is the central component of Cactus. It interfaces with modular components called **thorns**. The flesh provides:

- Variables & Data Types
- Parameters
- Functions for:
  - Parallelisation

The Flesh

- The **flesh** is the central component of Cactus. It interfaces with modular components called **thorns**. The flesh provides:

- Variables & Data Types
- Parameters
- Functions for:
  - Parallelisation
  - Input/Output

The Flesh

- The **flesh** is the central component of Cactus. It interfaces with modular components called **thorns**. The flesh provides:

- Variables & Data Types
- Parameters
- Functions for:
  - Parallelisation
  - Input/Output
  - Coordinates

The Flesh

- The **flesh** is the central component of Cactus. It interfaces with modular components called **thorns**. The flesh provides:

- Variables & Data Types
- Parameters
- Functions for:
  - Parallelisation
  - Input/Output
  - Coordinates
  - Reduction

The Flesh

- The **flesh** is the central component of Cactus. It interfaces with modular components called **thorns**. The flesh provides:

- Variables & Data Types
- Parameters
- Functions for:
  - Parallelisation
  - Input/Output
  - Coordinates
  - Reduction
  - Interpolation

The Flesh

- The **flesh** is the central component of Cactus. It interfaces with modular components called **thorns**. The flesh provides:

- Variables & Data Types
- Parameters
- Functions for:
  - Parallelisation
  - Input/Output
  - Coordinates
  - Reduction
  - Interpolation
  - Information

The Flesh

- The **flesh** is the central component of Cactus. It interfaces with modular components called **thorns**. The flesh provides:

- Variables & Data Types
- Parameters
- Functions for:
  - Parallelisation
  - Input/Output
  - Coordinates
  - Reduction
  - Interpolation
  - Information
- Staggering

The Flesh

- The **flesh** is the central component of Cactus. It interfaces with modular components called **thorns**. The flesh provides:

- Variables & Data Types
- Parameters
- Functions for:
  - Parallelisation
  - Input/Output
  - Coordinates
  - Reduction
  - Interpolation
  - Information
- Staggering
  - Indexing

The Flesh

- The **flesh** is the central component of Cactus. It interfaces with modular components called **thorns**. The flesh provides:

- Variables & Data Types
- Parameters
- Functions for:
  - Parallelisation
  - Input/Output
  - Coordinates
  - Reduction
  - Interpolation
  - Information
- Staggering
  - Indexing
  - Ghostzones

Thorns

- Some thorns provide additional functionality, while others serve as applications.
- Thorns are grouped into **arrangements** which supply some common functionality.
- Example thorns:

CactusIO	input and output operations
CactusIOJpeg	JPEG image data compression and writing operations
CactusConnect	networking
HTTPD	starts the HTTP daemon for remote connections
PUGH	unigrid driver + tools; reductions and interpolations
PUGH	unigrid driver handles grid scalars, arrays and functions

Application Toolkits

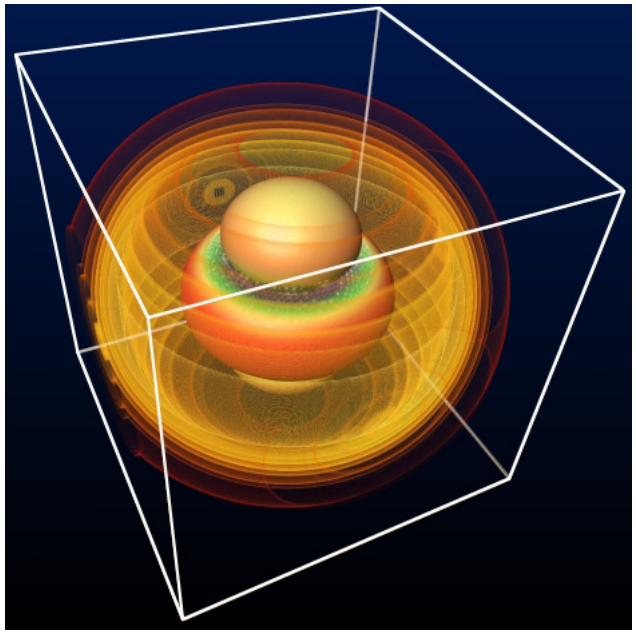
The **Einstein Toolkit** is a collection of arrangements for computational relativity. The toolkit includes a vacuum spacetime solver (McLachlan), a relativistic hydrodynamics solver, along with thorns for initial data, analysis and computational infrastructure.

The **Cactus Computational Toolkit** is a collection of arrangements that provides general computational infrastructure.

I/O Capabilities

Usual I/O and checkpointing in different formats:

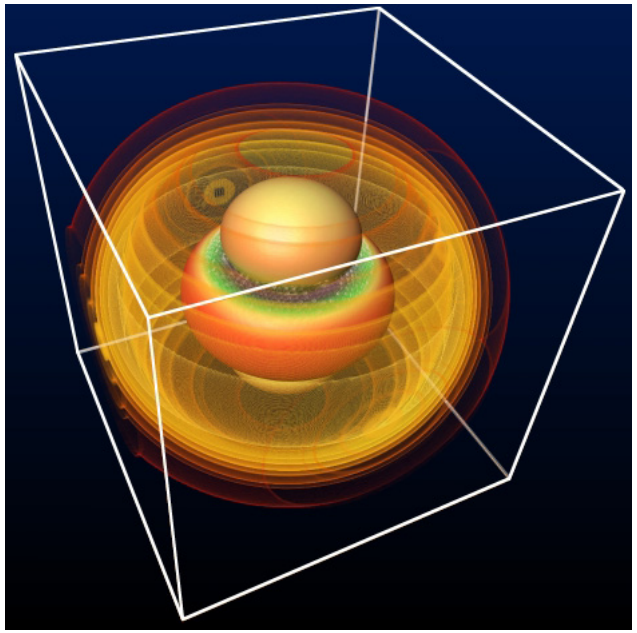
- Screen output



I/O Capabilities

Usual I/O and checkpointing in different formats:

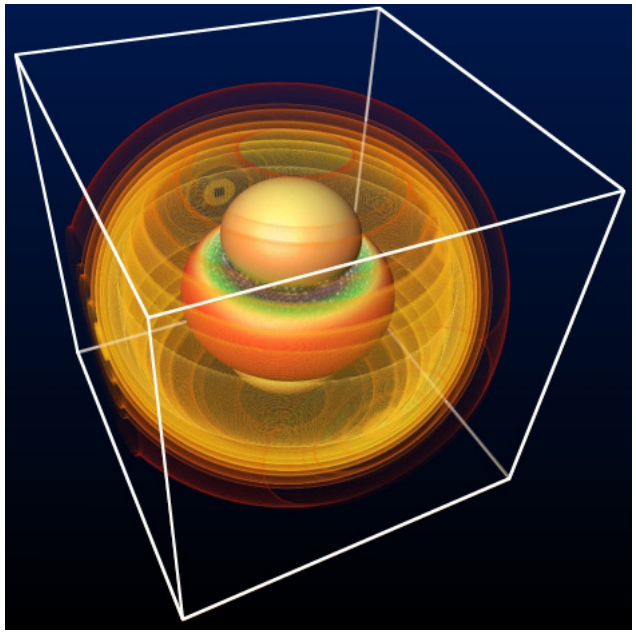
- Screen output
- ASCII file output



I/O Capabilities

Usual I/O and checkpointing in different formats:

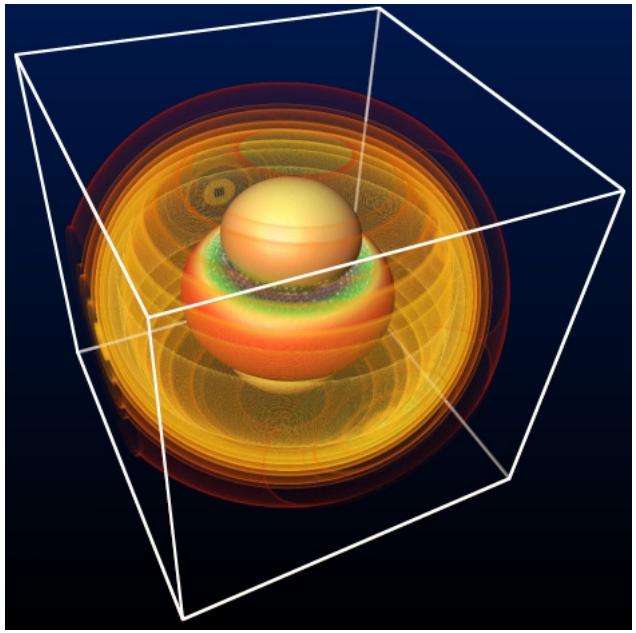
- Screen output
- ASCII file output
- HDF5 file in-/output



I/O Capabilities

Usual I/O and checkpointing in different formats:

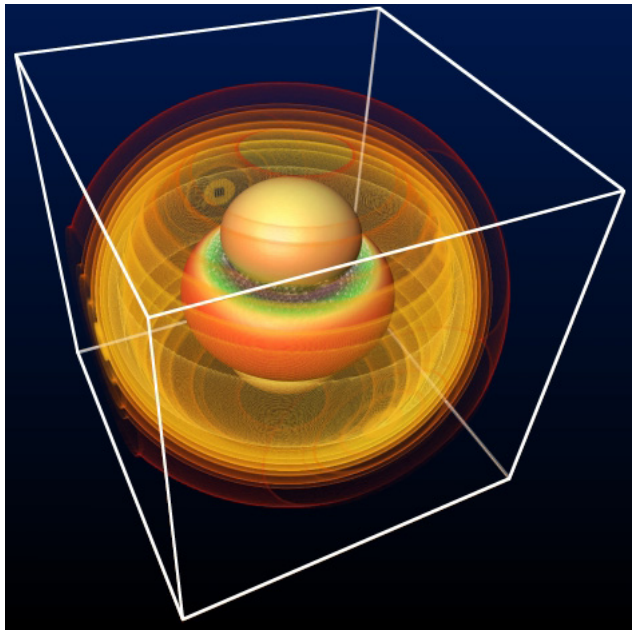
- Screen output
- ASCII file output
- HDF5 file in-/output
- Online Jpeg rendering



I/O Capabilities

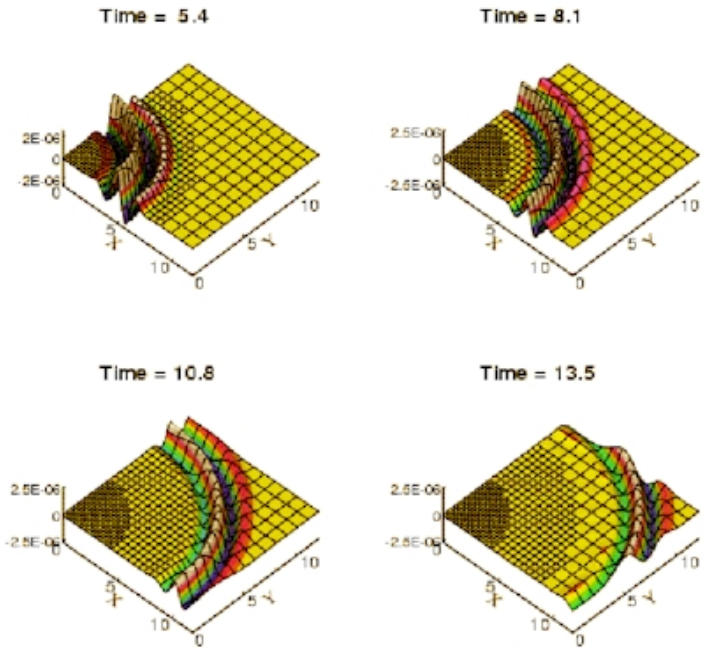
Usual I/O and checkpointing in different formats:

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



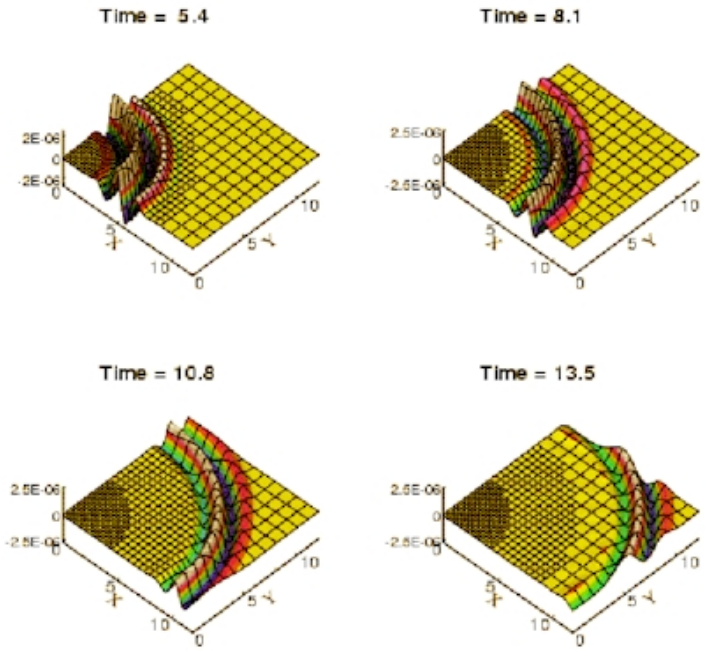
More Capabilities: Grids, Boundaries, Symmetries

- Grids



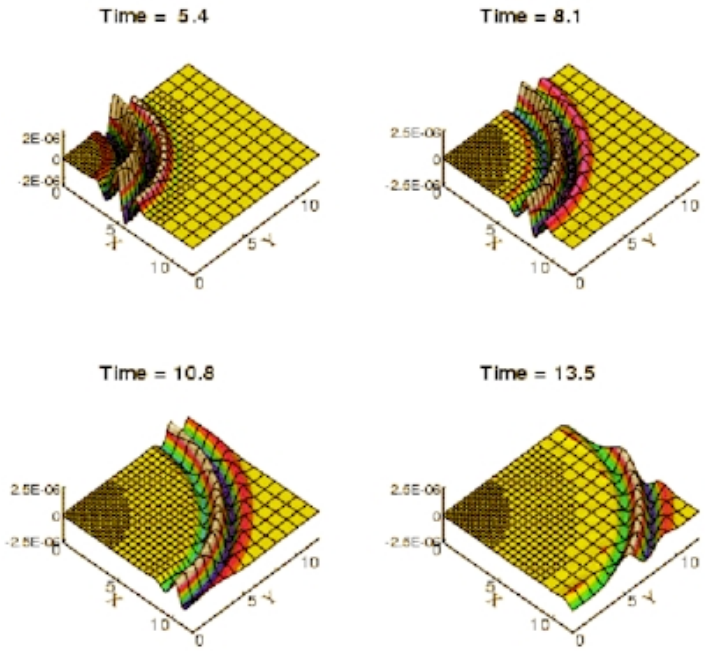
More Capabilities: Grids, Boundaries, Symmetries

- Grids
  - Only structured meshes (at the moment)



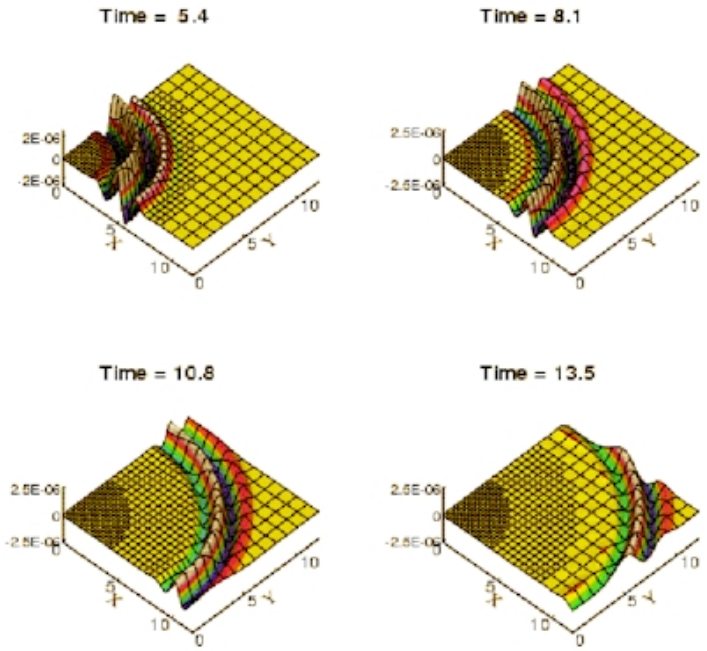
More Capabilities: Grids, Boundaries, Symmetries

- Grids
  - Only structured meshes (at the moment)
  - Unigrid (PUGH)



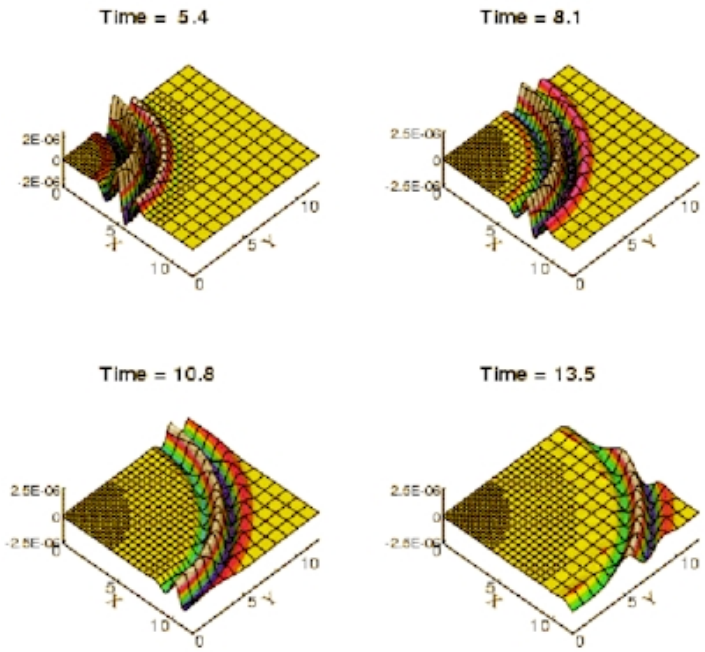
More Capabilities: Grids, Boundaries, Symmetries

- Grids
  - Only structured meshes (at the moment)
  - Unigrid (PUGH)
  - Adaptive Mesh Refinement (Carpet)



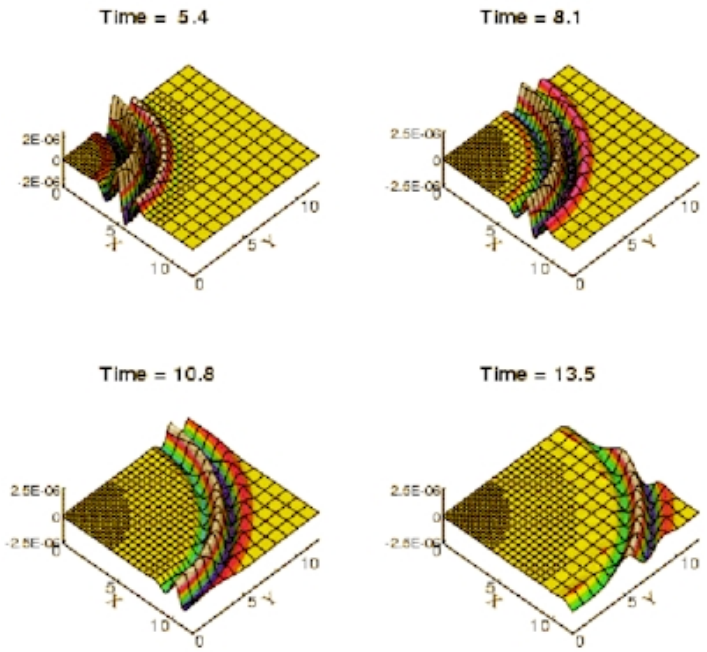
More Capabilities: Grids, Boundaries, Symmetries

- Grids
  - Only structured meshes (at the moment)
  - Unigrid (PUGH)
  - Adaptive Mesh Refinement (Carpet)
- Boundaries / Symmetries



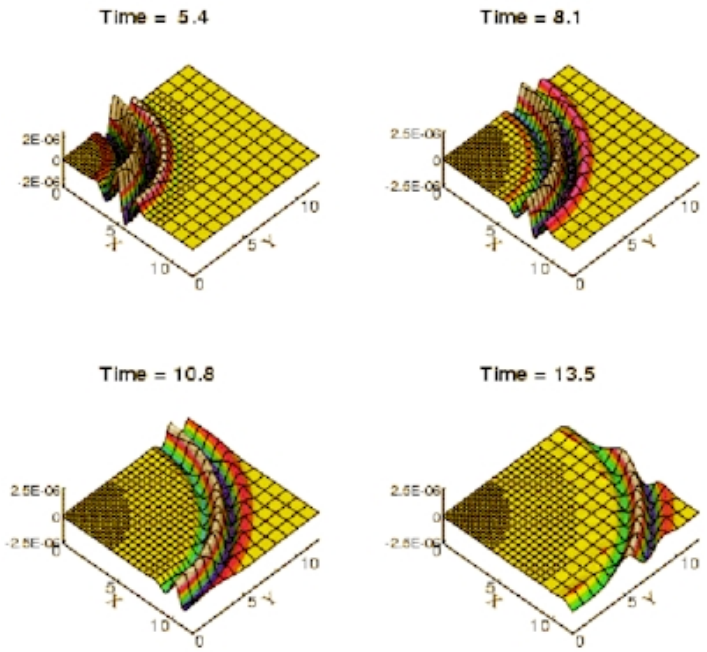
More Capabilities: Grids, Boundaries, Symmetries

- Grids
  - Only structured meshes (at the moment)
  - Unigrid (PUGH)
  - Adaptive Mesh Refinement (Carpet)
- Boundaries / Symmetries
  - Periodic



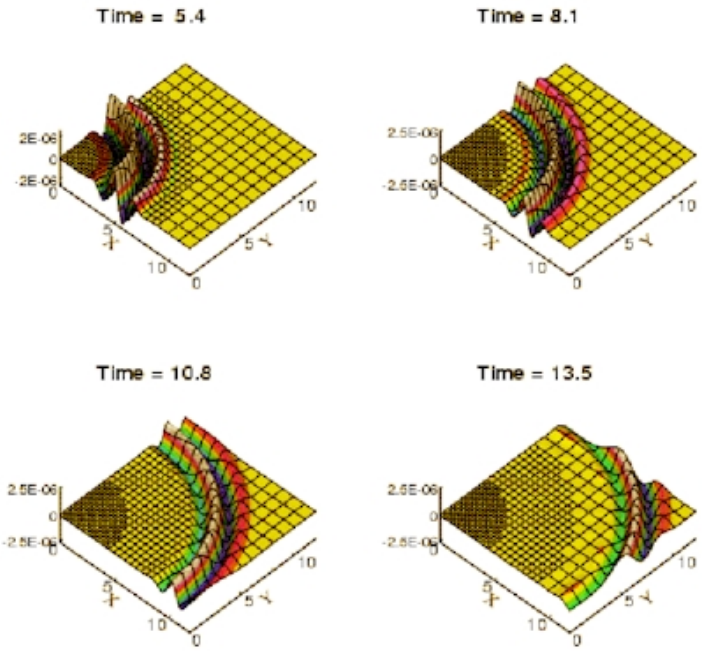
More Capabilities: Grids, Boundaries, Symmetries

- Grids
  - Only structured meshes (at the moment)
  - Unigrid (PUGH)
  - Adaptive Mesh Refinement (Carpet)
- Boundaries / Symmetries
  - Periodic
  - Static



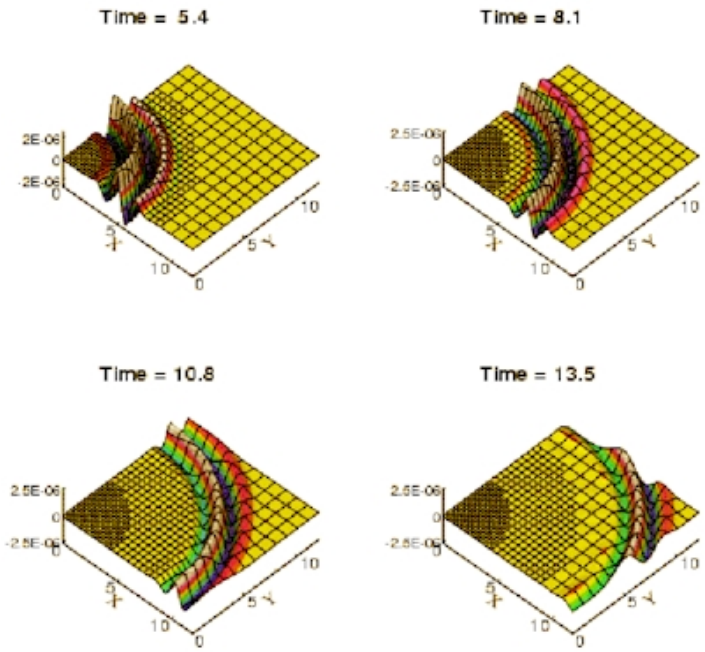
More Capabilities: Grids, Boundaries, Symmetries

- Grids
  - Only structured meshes (at the moment)
  - Unigrid (PUGH)
  - Adaptive Mesh Refinement (Carpet)
- Boundaries / Symmetries
  - Periodic
  - Static
  - Mirror symmetries



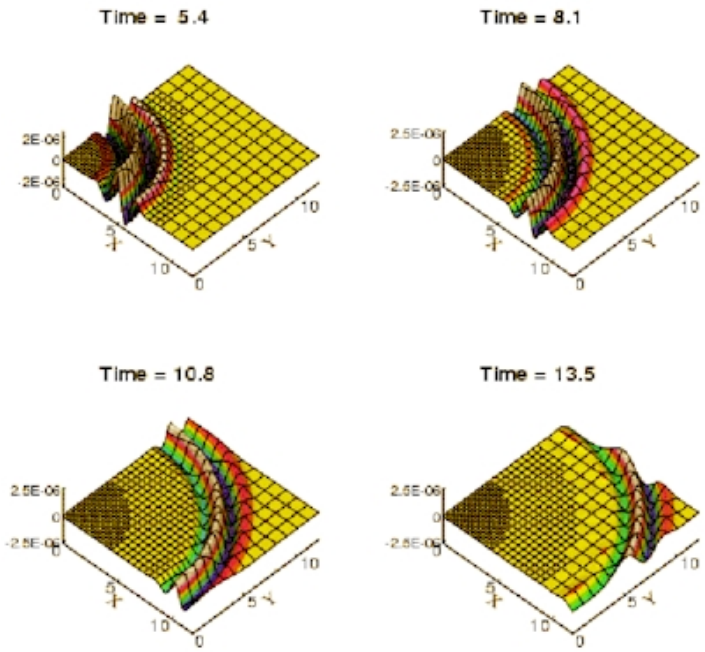
More Capabilities: Grids, Boundaries, Symmetries

- Grids
  - Only structured meshes (at the moment)
  - Unigrid (PUGH)
  - Adaptive Mesh Refinement (Carpet)
- Boundaries / Symmetries
  - Periodic
  - Static
  - Mirror symmetries
  - Rotational symmetries



More Capabilities: Grids, Boundaries, Symmetries

- Grids
  - Only structured meshes (at the moment)
  - Unigrid (PUGH)
  - Adaptive Mesh Refinement (Carpet)
- Boundaries / Symmetries
  - Periodic
  - Static
  - Mirror symmetries
  - Rotational symmetries
  - Problemspecific boundaries



The Cactus Computational Toolkit

Core modules (thorns) providing many basic utilities:

- I/O methods

The Cactus Computational Toolkit

Core modules (thorns) providing many basic utilities:

- I/O methods
- Boundary conditions

The Cactus Computational Toolkit

Core modules (thorns) providing many basic utilities:

- I/O methods
- Boundary conditions
- Parallel unigrid driver

The Cactus Computational Toolkit

Core modules (thorns) providing many basic utilities:

- I/O methods
- Boundary conditions
- Parallel unigrid driver
- Reduction and Interpolation operators

The Cactus Computational 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

The Cactus Computational 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

The Cactus Computational Toolkit

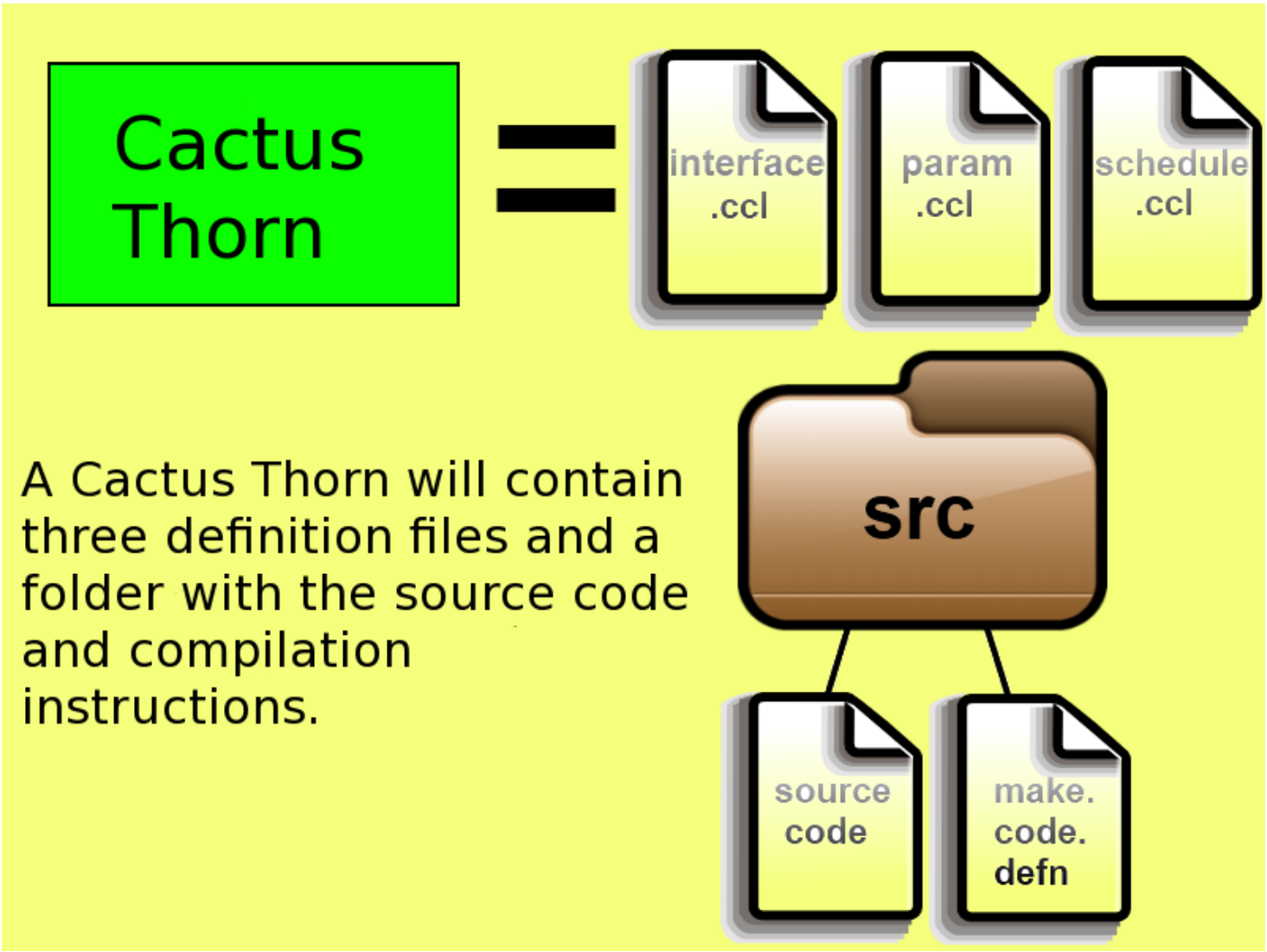
Core modules (thorns) providing many basic utilities:

- 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)

Many arrangements with many modules...

CactusBase	Basic utility and interface thorns
CactusBench	Benchmark utility thorns
CactusConnect	Network utility thorns
CactusElliptic	Elliptic solvers / interface thorns
CactusExamples	Example thorns
CactusExternal	External library interface thorns
CactusIO	General I/O thorns
CactusNumerical	General numerical methods
CactusPUGH	Cactus Unigrid Driver thorn
CactusPUGHIO	I/O thorns specifix for PUGH driver
CactusTest	Thorns for Cactus testing
CactusUtils	Misc. utility thorns
CactusWave	Wave example thorns

Cactus thorn structure



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

Cactus Configuration Language Files

The interaction of a thorn with the Flesh and other thorns is controlled by certain configuration files:

`interface.ccl`	defines the implementation (Section B2.3) the thorn provides, and the variables the thorn needs, along with their visibility to other implementations
`schedule.ccl`	defines which functions are called from the thorn and when they are called; it also handles memory and communication assignment for grid variables
`param.ccl`	defines the parameters that are used to control the thorn, along with their visibility to other implementations

These are called *Cactus Configuration Language* files.

param.ccl

param.ccl:

```
:
# Parameter definitions for thorn HTTPD
# $Header$

shares: Cactus

# cctk variables used
USES REAL cctk_final_time
USES REAL cctk_initial_time
USES INT  cctk_itlast        # time of last iteration

private:

# sets the port number
INT port "HTTP port number to use"
{
    # ports 1-65535 are valid
    1:65535 :: "Any valid port"
} 5555
# 5555 is the default port

# if true, starts the simulation paused
BOOLEAN pause "Pause ?" STEERABLE = ALWAYS
{
} "no"
# do not start simulation paused by default

# username and password for controlling cactus
STRING user "The username for Cactus Control "
{
  # any non-empty string is valid
```

interface.ccl

interface.ccl:

```
# Interface definition for thorn HTTPD
# $Header$

# state what thorn is implemented
Implements: HTTPD
# inherit all the functions of Socket
Inherits: Socket

# name of the include header
USES INCLUDE HEADER: SocketUtils.h

INCLUDE HEADER: Auth.h in http_Auth.h
INCLUDE HEADER: Cookies.h in http_Cookies.h
INCLUDE HEADER: Steer.h in http_Steer.h
INCLUDE HEADER: Content.h in http_Content.h

CCTK_INT FUNCTION Send_Twitter_Msg(CCTK_STRING IN msg)
USES FUNCTION Send_Twitter_Msg
```

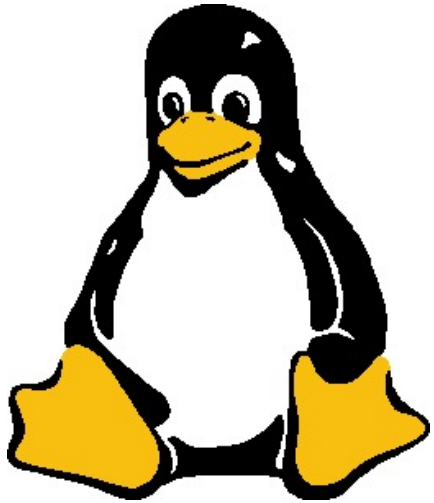
schedule.ccl

schedule.ccl:

```
# Schedule definitions for thorn HTTPD
# $Header$

# perform these functions at startup
SCHEDULE GROUP HTTP_Startup AT startup
{
  OPTIONS: GLOBAL
} "HTTP daemon startup group"

SCHEDULE HTTP_StartServer in HTTP_Startup
{
  LANG: C
  OPTIONS: GLOBAL
} "Start HTTP server"

# during startup but after HTTP_SetupPages, do HTTP_FirstServ
SCHEDULE HTTP_FirstServ in HTTP_Startup AFTER HTTP_SetupPages
{
  LANG: C
  OPTIONS: GLOBAL
} "Serve first pages at startup"

# poststep is after startup and after every evolution step
# HTTP_Work at poststep but before IOUtil_UpdateParFile
SCHEDULE HTTP_Work AT poststep BEFORE IOUtil_UpdateParFile
{
  LANG: C
  OPTIONS: GLOBAL
} "Working routine"

if(provide_pages)
```

Linux Overview

- Though Cactus is cross-platform and may be developed in the developer's operating system of choice, Linux is the primary development environment. We will use a Linux environment for this tutorial.



  - Linux was developed in 1991 by Linus Torvalds, after whom the operating system is named. The open-source philosophy underlying Linux inspires many software developers to license their software under the GPU (GNU Public License), allowing the software user access to the program source code as well as the right to modify and release forks of it.

Distributions

- For this reason, there are many varities of Linux called **distributions**. The major difference among distributions is often their package management system. The most popular distributions with original package management systems include:

Though Linux offers a feature-rich GUI with many options for desktop environments and window managers, it is nevertheless a command-driven system.

Linux has a mature command-line shell well-suited as a programming environment. One may write, compile, and debug programs from the CLI.

- Our OS is a minimalist Debian-based operating system that contains all the features required for Cactus/ETK development:

- Our OS uses a minimalist window manager called Fluxbox to ensure that it runs speedily inside of the virtual environment.

- We name our distribution ETK-OS.
- We offer ETK-OS as a virtual machine (VirtualBox) file and as a LiveCD.

- Our OS is a minimalist Debian-based operating system that contains all the features required for Cactus/ETK development:
  - Eclipse/Mojave

- Our OS uses a minimalist window manager called Fluxbox to ensure that it runs speedily inside of the virtual environment.

- We name our distribution ETK-OS.
- We offer ETK-OS as a virtual machine (VirtualBox) file and as a LiveCD.

- Our OS is a minimalist Debian-based operating system that contains all the features required for Cactus/ETK development:
  - Eclipse/Mojave
  - Cactus Computational Toolkit

- Our OS uses a minimalist window manager called Fluxbox to ensure that it runs speedily inside of the virtual environment.

- We name our distribution ETK-OS.
- We offer ETK-OS as a virtual machine (VirtualBox) file and as a LiveCD.

- Our OS is a minimalist Debian-based operating system that contains all the features required for Cactus/ETK development:
  - Eclipse/Mojave
  - Cactus Computational Toolkit
  - Einstein Toolkit

- Our OS uses a minimalist window manager called Fluxbox to ensure that it runs speedily inside of the virtual environment.

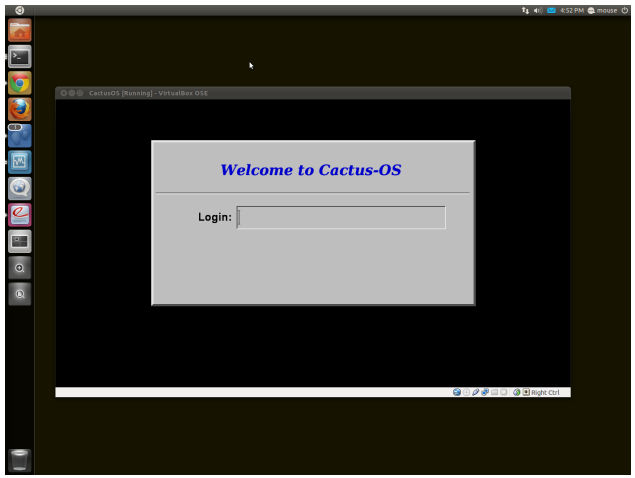
- We name our distribution ETK-OS.
- We offer ETK-OS as a virtual machine (VirtualBox) file and as a LiveCD.

- Our OS is a minimalist Debian-based operating system that contains all the features required for Cactus/ETK development:
  - Eclipse/Mojave
  - Cactus Computational Toolkit
  - Einstein Toolkit
  - Simfactory

- Our OS uses a minimalist window manager called Fluxbox to ensure that it runs speedily inside of the virtual environment.

- We name our distribution ETK-OS.
- We offer ETK-OS as a virtual machine (VirtualBox) file and as a LiveCD.

- To use the ETK-OS LiveCD, insert the CD into the DVD-ROM player and reboot the computer.
- Ensure that in your BIOS settings, the DVD-ROM takes priority over other bootable devices.



- ETK-OS will then boot from the DVD.
- The username is ‘cactus’.
- The password is not set in the LiveCD environment, nor is it necessary; the user already has administrative permission.

The DVD creates a special filesystem in RAM to which it performs all output operations. Thus all changes are written to RAM, not to the hard disk, and will be erased on reboot.

If a hard disk is attached to the machine, it may be mounted (made available to ETK-OS for I/O). Mounting a filesystem makes the root directory of that filesystem available as a subfolder from the root directory of the working filesystem.

Any files on the DVD, including its binaries, are read-only and cannot be changed.

Virtualization

- If you prefer to run ETK-OS within your working OS, you may use virtualization.
- **Virtualization** is the process of emulating one operating system within another. The emulated OS is called the **guest OS**; the OS in which the emulation is done is the **host OS**.

Virtualization

- If you prefer to run ETK-OS within your working OS, you may use virtualization.
- **Virtualization** is the process of emulating one operating system within another. The emulated OS is called the **guest OS**; the OS in which the emulation is done is the **host OS**.
- **Virtualization software** is required to achieve this end.

Virtualization

- If you prefer to run ETK-OS within your working OS, you may use virtualization.
- **Virtualization** is the process of emulating one operating system within another. The emulated OS is called the **guest OS**; the OS in which the emulation is done is the **host OS**.
- **Virtualization software** is required to achieve this end.
- Examples of virtualization software packages include:

Parallels™

vmware®

VirtualBox

VirtualBox is:

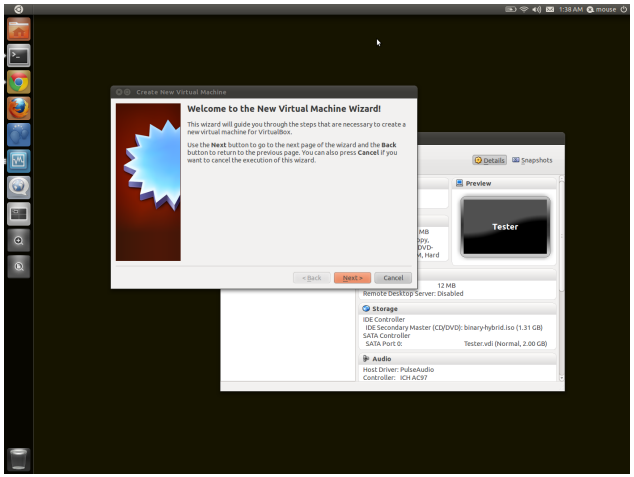
- cost-free
- open-source
- cross-platform
- intuitive
- easily configurable

VirtualBox

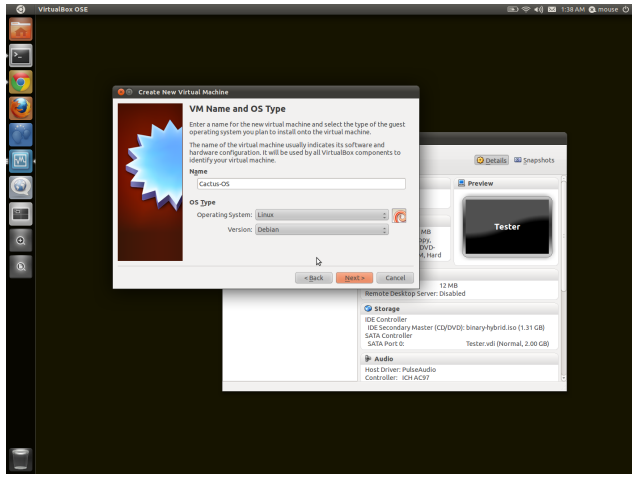
VirtualBox is:

- cost-free
- open-source
- cross-platform
- intuitive
- easily configurable

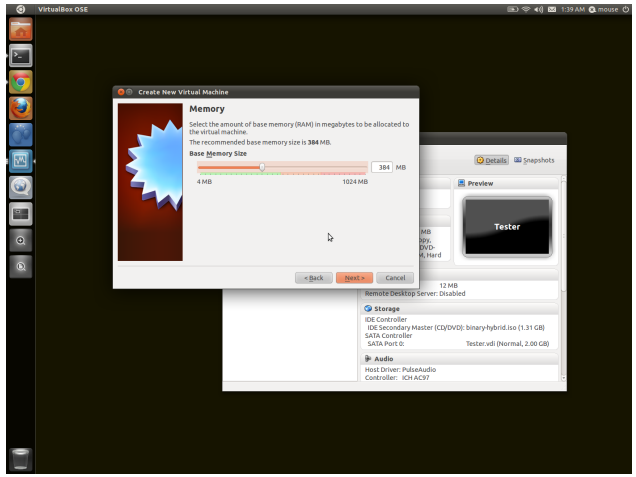
Thus we choose it for our virtualization needs. In VirtualBox, the guest OS resides in a VBox machine file which is transferrable from one host OS to another.


Create New Virtual Machine
Welcome to the New Virtual Machine Wizard!
This wizard will guide you through the steps that are necessary to create a new virtual machine for VirtualBox.
Use the Next button to go to the next page of the wizard and the Back button to return to the previous page. You can also press Cancel if you want to cancel the execution of this wizard.
< Back
Next >
Cancel
Details
Snapshots
Preview
Tester
Storage
IDE Controller
IDE Secondary Master (CD/DVD): binary-hybrid.iso (1.31 GB)
SATA Controller
SATA Port 0:
Tester.vdi (Normal, 2.00 GB)
Audio
Host Driver: PulseAudio
Controller: ICH AC97

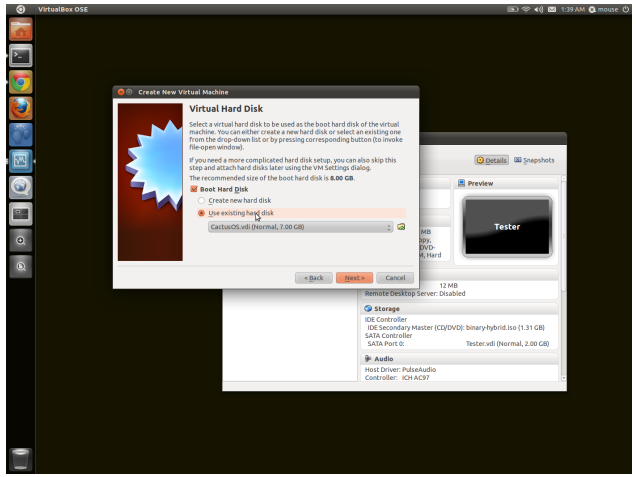
- Call the system ‘ETK-OS’.
- It is a Debian-based Linux distribution.

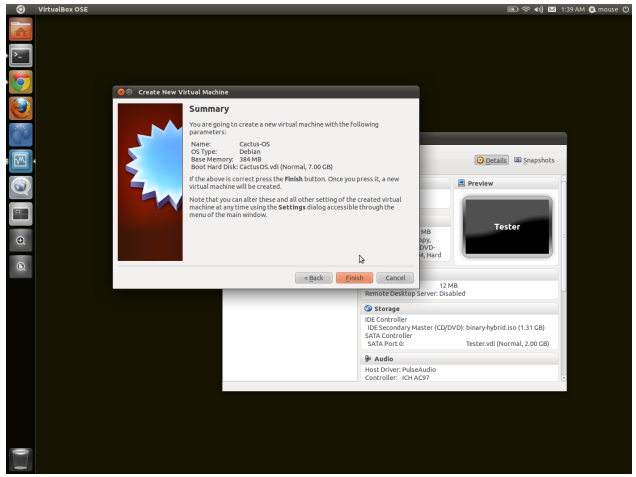


- Set the RAM equal to one-third to one-fourth of your total physical RAM.
- Setting the VM RAM too low may cause it to run out of available memory during the tutorial.
- However, setting it to high may cause it to slow down. VirtualBox requires RAM available to the host OS to support low-level processes which allow it to run.

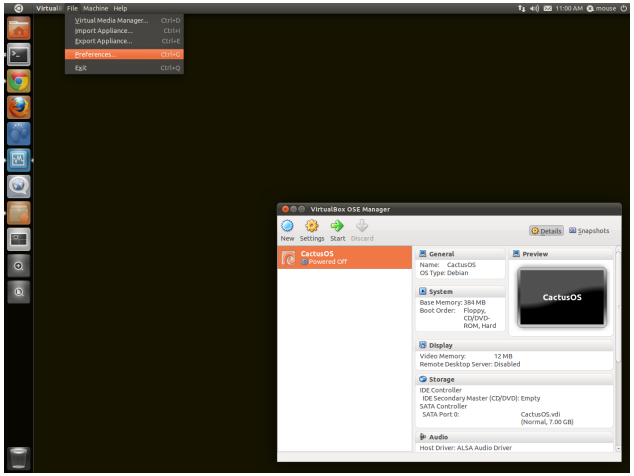


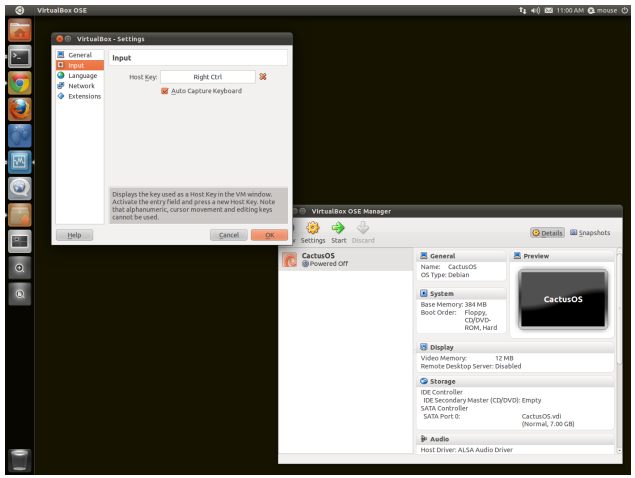
- Use the .vdi file from the DVD-ROM. You'll want to copy the .vdi file onto your hard drive before selecting.




VirtualBox OSE
Create New Virtual Machine
Summary
You are going to create a new virtual machine with the following parameters:
Name: Cactus-OS
OS Type: Debian
Base Memory: 384 MB
Boot Hard Disk: CactusOS.vdi (Normal, 7.00 GB)
If the above is correct press the Finish button. Once you press it, a new virtual machine will be created.
Note that you can alter these and all other setting of the created virtual machine at any time using the Settings dialog accessible through the menu of the main window.
< Back
Finish
Cancel
Details
Snapshots
Preview
Tester
Storage
IDE Controller
IDE Secondary Master (CD/DVD): binary-hybrid.iso (1.31 GB)
SATA Controller
SATA Port 0:
Tester.vdi (Normal, 2.00 GB)
Audio
Host Driver: PulseAudio
Controller: ICH AC97

In order to escape the VirtualBox environment, you may press the VirtualBox Host Key. By default it is Right Ctrl. Since you may have to use the Ctrl key within the Linux enviroment, you may wish to change the Host Key. Under the VirtualBox menu, click 'Machine'.





In the left-hand pane of the dialogue window that appears, click on 'Input'. On the right-hand side, you may change the Host Key; simply click within the text window and hit the desired Host Key.

The Terminal

- During the tutorial, we will use a command-line interface (terminal).

The Terminal

- During the tutorial, we will use a command-line interface (terminal).
- On the desktop, click on the icon labelled ‘gterm’.

The Terminal

- During the tutorial, we will use a command-line interface (terminal).
- On the desktop, click on the icon labelled ‘gterm’.
- A terminal window will appear.

Essential Terminal Commands: Navigation and Files

pwd	present working directory	pwd
ls	list directory contents	ls [dir]
cd	change directory ('..' means up)	cd [dir]
cp	copy file/directory (-r)	cp [file/dir1] [file/dir2]
mv	move/rename file/directory	mv [file/dir1] [file/dir2]
rm	delete file/directory (-r)	rm [file/dir(s)]

Text File I/O

cat	print text file to screen	cat [file(s)]
head	print first n lines of file (default 10)	head -n [num] [file
tail	print last n lines of file (default 10)	tail -n [num] [file(s
nl	print text file to screen with line numbering	nl [file(s)]
less	print text file to screen, scrollable	less [file(s)]

Text File I/O

nano	simple, intuitive text editor	nano [file(s)]
vim	full-featured programming text editor	vim [file(s)]

- To navigate vim/less: h, j, k, l.
- Use i to insert in vim, ESC to quit insert mode.
- Use Ctrl+X to save/quit nano, ZZ to save/quit vim, and q to quit less.

Searching and Sorting

grep	find regex in a file	grep [regex] [file]
sort	sort the lines of a file	sort [file]
uniq	isolate unique lines in a file	uniq [file]
tac	print lines of file backwards	tac [file]
rev	print characters in lines backwards	rev [file]

Documentation

man	manual pages	man [command]
apropos	find commands	apropos [keyword]

An Overview of Application Thorns

Basic Thorn Anatomy

- Each thorn consists of a subdirectory containing four files:

Basic Thorn Anatomy

- Each thorn consists of a subdirectory containing four files:
- interface.ccl - This file defines the Cactus interface. It will include a header block which gives details about the thorns interactions with other thorns, if any. It will also include a block defining the thorn's global variables, if any.

Basic Thorn Anatomy

- Each thorn consists of a subdirectory containing four files:
- interface.ccl - This file defines the Cactus interface. It will include a header block which gives details about the thorns interactions with other thorns, if any. It will also include a block defining the thorn's global variables, if any.
- param.ccl - This file includes the parameters introduced by this thorn, as well as the parameters needed from other thorn interactions, as defined in the interface.ccl file. This file can also detail the scope of the given parameters (i.e. global, private, etc.)

Basic Thorn Anatomy

- Each thorn consists of a subdirectory containing four files:
- interface.ccl - This file defines the Cactus interface. It will include a header block which gives details about the thorns interactions with other thorns, if any. It will also include a block defining the thorn's global variables, if any.
- param.ccl - This file includes the parameters introduced by this thorn, as well as the parameters needed from other thorn interactions, as defined in the interface.ccl file. This file can also detail the scope of the given parameters (i.e. global, private, etc.)
- configuration.ccl - This file is optional. It contains capabilities which a thorn provides, requires, or may use if available. For example, this file can provide access to external libraries, provide access to functions for other thorns, or split a thorn into several thorns, all of which require some commone (not aliased) functions.

Continued

- schedule.ccl - This file contains scheduling information for routines called by the flesh. This will determine which routines will be run in which order; via:

Continued

- schedule.ccl - This file contains scheduling information for routines called by the flesh. This will determine which routines will be run in which order; via:
  - Assignment Statements - switch on grid variable storage for the duration of the program's execution.

Continued

- schedule.ccl - This file contains scheduling information for routines called by the flesh. This will determine which routines will be run in which order; via:
  - Assignment Statements - switch on grid variable storage for the duration of the program's execution.
  - Schedule Blocks - schedule a given subroutine from a thorn to be called at given time's in the program's execution.

Continued

- schedule.ccl - This file contains scheduling information for routines called by the flesh. This will determine which routines will be run in which order; via:
  - Assignment Statements - switch on grid variable storage for the duration of the program's execution.
  - Schedule Blocks - schedule a given subroutine from a thorn to be called at given time's in the program's execution.
  - Conditional Statements - Allow both assignment statements and schedule blocks to be processed given conditional parameter values.

Continued

- The thorn will also include a README file with a brief description of the thorn, as well as several subdirectories, including:

Continued

- The thorn will also include a README file with a brief description of the thorn, as well as several subdirectories, including:
  - /src - For holding source files, as well as compilation instructions.

Continued

- The thorn will also include a README file with a brief description of the thorn, as well as several subdirectories, including:
  - /src - For holding source files, as well as compilation instructions.
  - /src/include - For include files.

Continued

- The thorn will also include a README file with a brief description of the thorn, as well as several subdirectories, including:
  - /src - For holding source files, as well as compilation instructions.
  - /src/include - For include files.
  - /doc - For documentation.

Continued

- The thorn will also include a README file with a brief description of the thorn, as well as several subdirectories, including:
  - /src - For holding source files, as well as compilation instructions.
  - /src/include - For include files.
  - /doc - For documentation.
  - /par - For example parameter files.

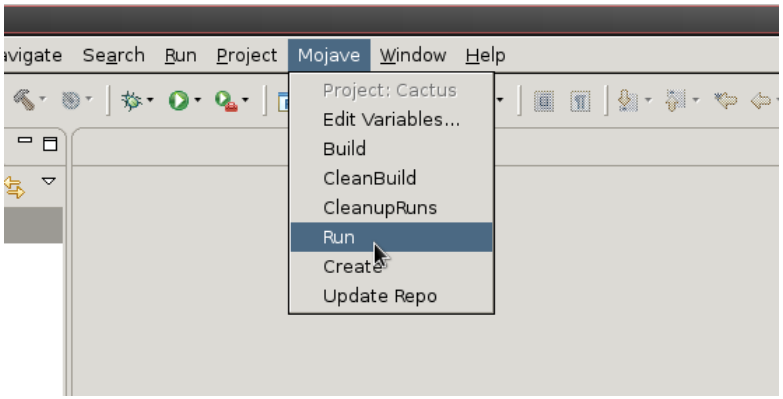
Continued

- The thorn will also include a README file with a brief description of the thorn, as well as several subdirectories, including:
  - /src - For holding source files, as well as compilation instructions.
  - /src/include - For include files.
  - /doc - For documentation.
  - /par - For example parameter files.
  - /test - For holding the thorn's test suite.

Running Cactus

To run HelloWorld through Mojave:

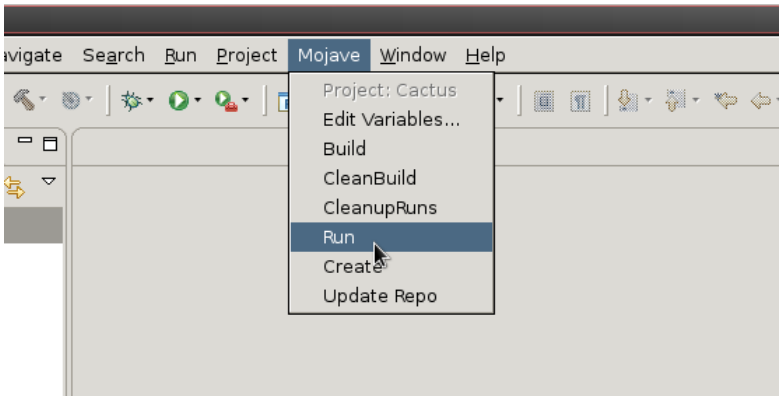
- Open virtualbox and start CactusOS



Running Cactus

To run HelloWorld through Mojave:

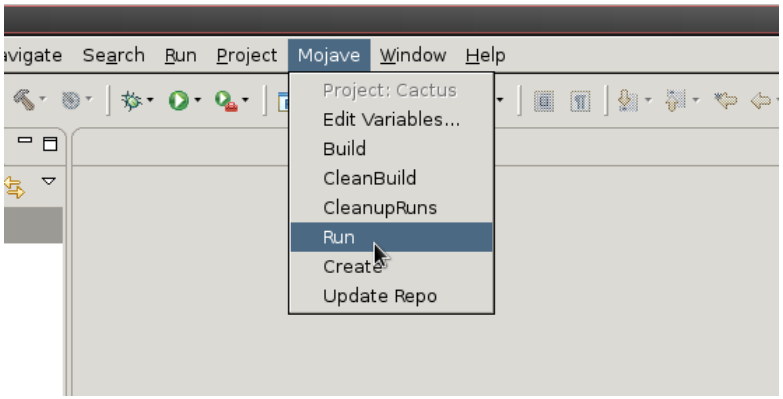
- Open virtualbox and start CactusOS
- Launch Eclipse



Running Cactus

To run HelloWorld through Mojave:

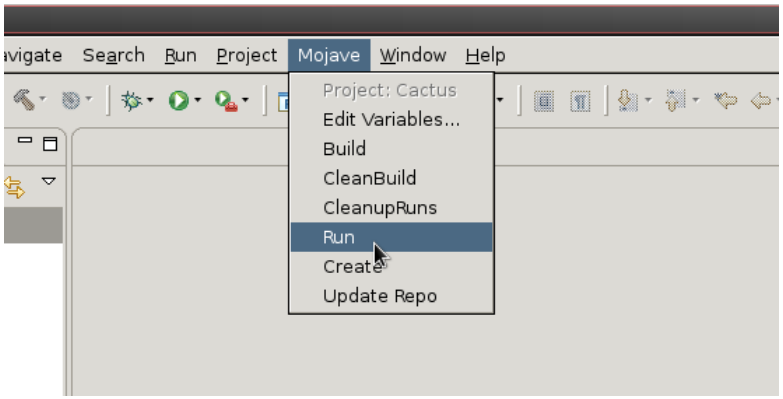
- Open virtualbox and start CactusOS
- Launch Eclipse
- Select Mojave from the drop down tab



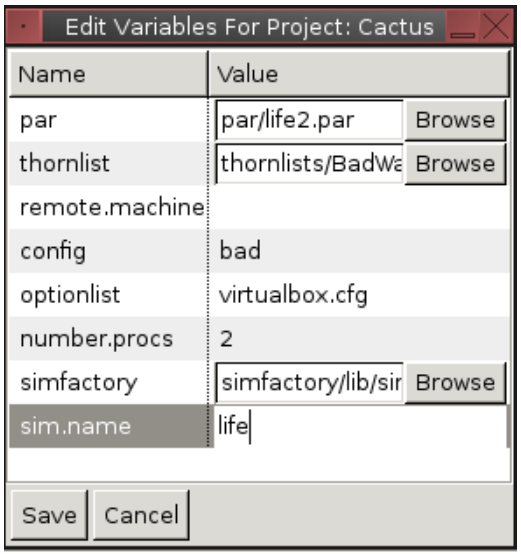
Running Cactus

To run HelloWorld through Mojave:

- Open virtualbox and start CactusOS
- Launch Eclipse
- Select Mojave from the drop down tab
- Select Run



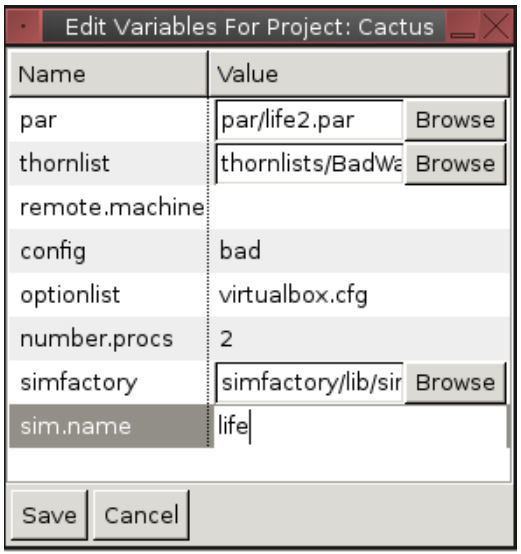
Running Cactus



To run Conway's Life through Mojave:

- Select Mojave from the drop down tab

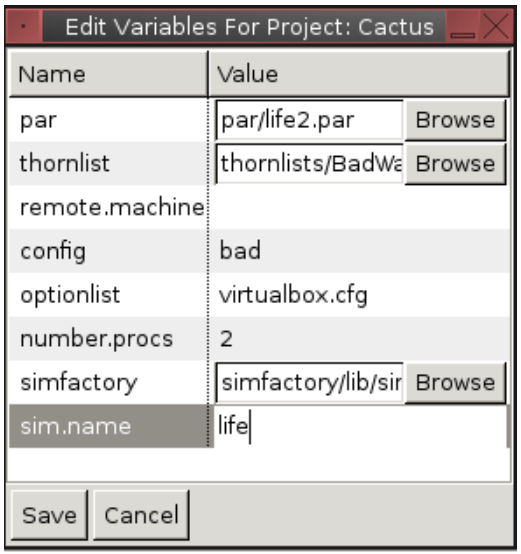
Running Cactus



To run Conway's Life through Mojave:

- Select Mojave from the drop down tab
- Edit Variables

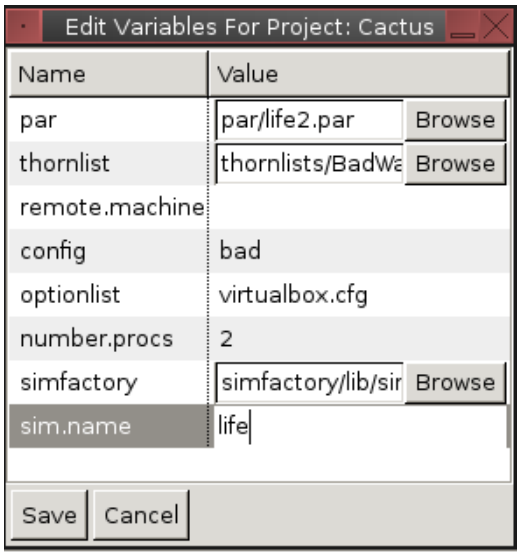
Running Cactus



To run Conway's Life through Mojave:

- Select Mojave from the drop down tab
- Edit Variables
- For the par file:

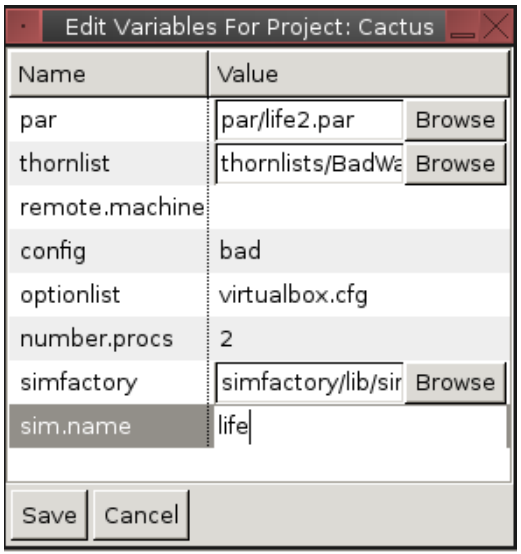
Running Cactus



To run Conway's Life through Mojave:

- Select Mojave from the drop down tab
- Edit Variables
- For the par file:
  - Select Browse – Cactus – par – life2.par

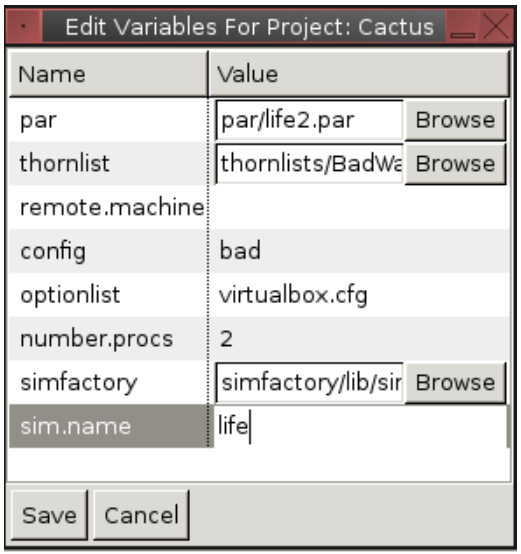
Running Cactus



To run Conway's Life through Mojave:

- Select Mojave from the drop down tab
- Edit Variables
- For the par file:
  - Select Browse – Cactus – par – life2.par
- In the sim.name space type: life

Running Cactus



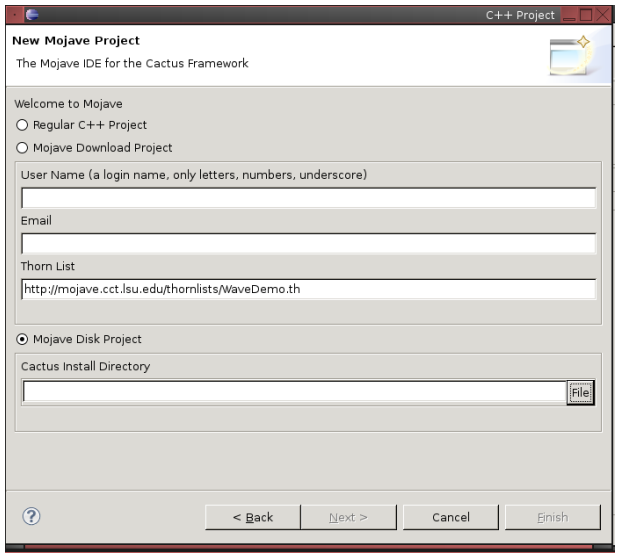
To run Conway's Life through Mojave:

- Select Mojave from the drop down tab
- Edit Variables
- For the par file:
  - Select Browse – Cactus – par – life2.par
- In the sim.name space type: life
- Select Save – Mojave – Run

Running Cactus

To run Conway's Life through Mojave:

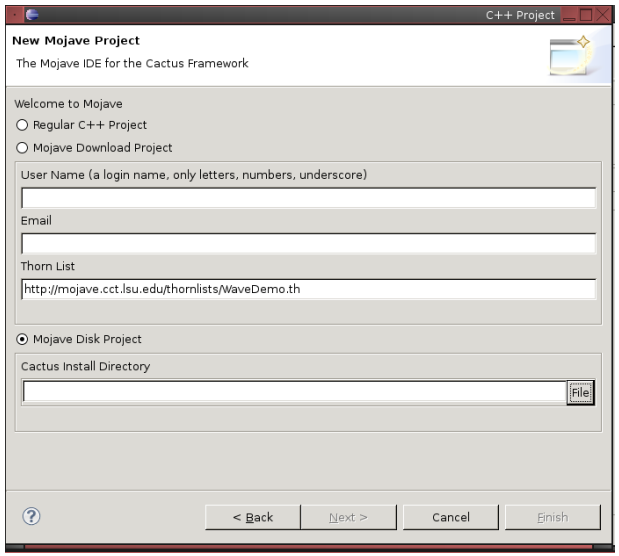
- Select Mojave Disk Project



Running Cactus

To run Conway's Life through Mojave:

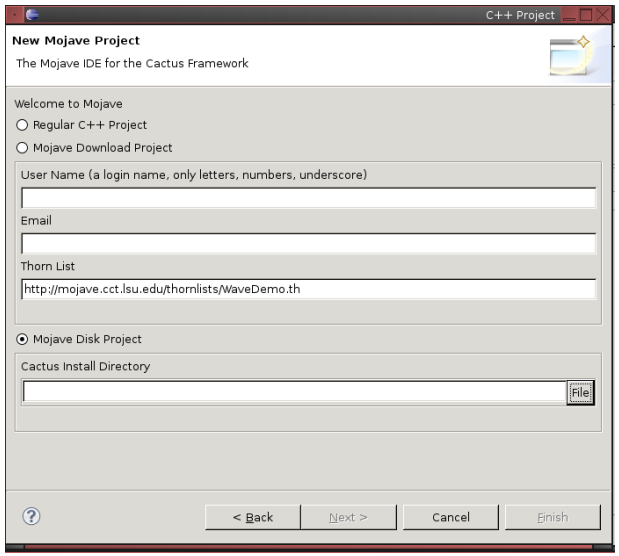
- Select Mojave Disk Project
- Select the File button



Running Cactus

To run Conway's Life through Mojave:

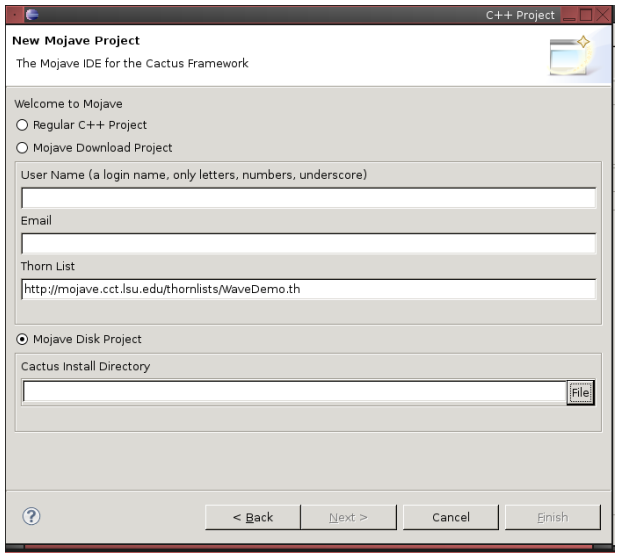
- Select Mojave Disk Project
- Select the File button
- Open your Cactus directory and click OK – Finish



Running Cactus

To run Conway's Life through Mojave:

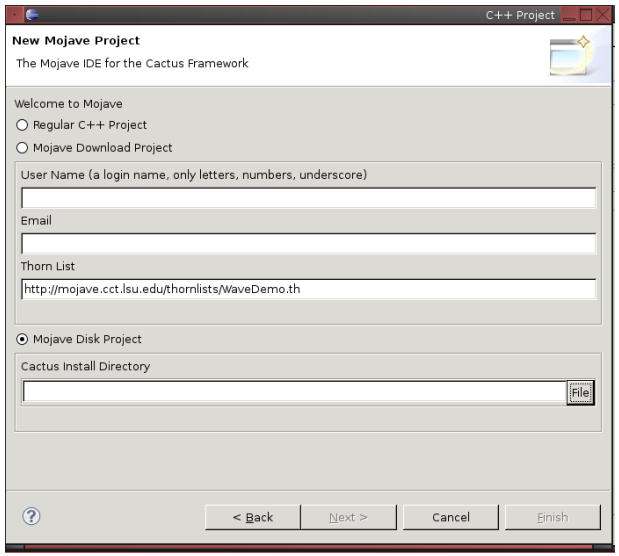
- Select Mojave Disk Project
- Select the File button
- Open your Cactus directory and click OK – Finish
- Select Mojave from the pulldown tab



Running Cactus

To run Conway's Life through Mojave:

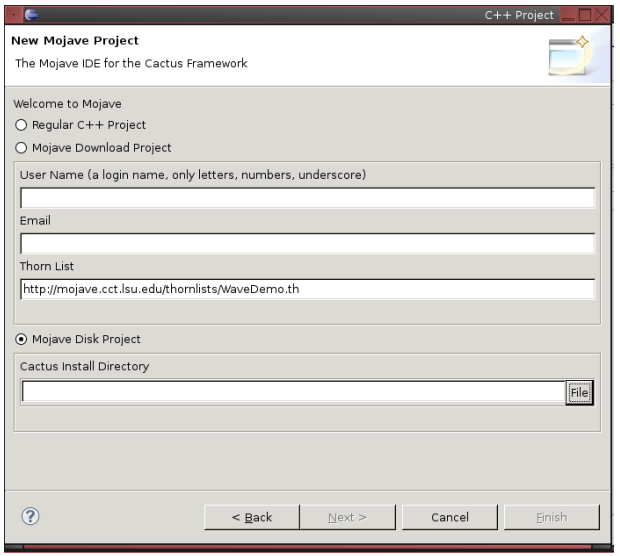
- Select Mojave Disk Project
- Select the File button
- Open your Cactus directory and click OK – Finish
- Select Mojave from the pulldown tab
- Select Edit Variables



Running Cactus

To run Conway's Life through Mojave:

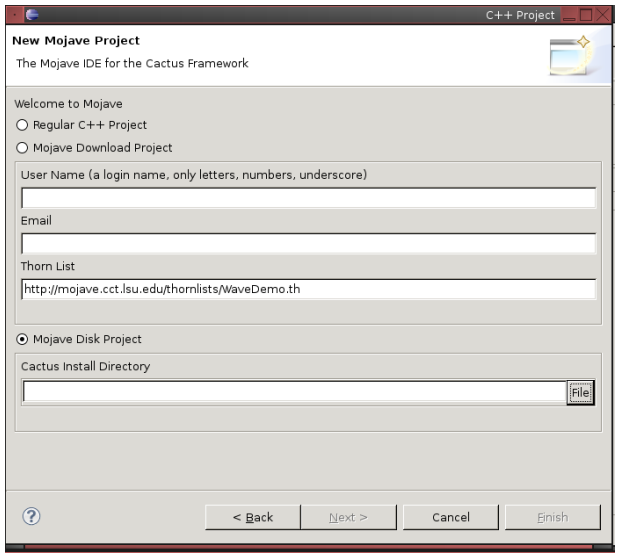
- For the par option:



Running Cactus

To run Conway's Life through Mojave:

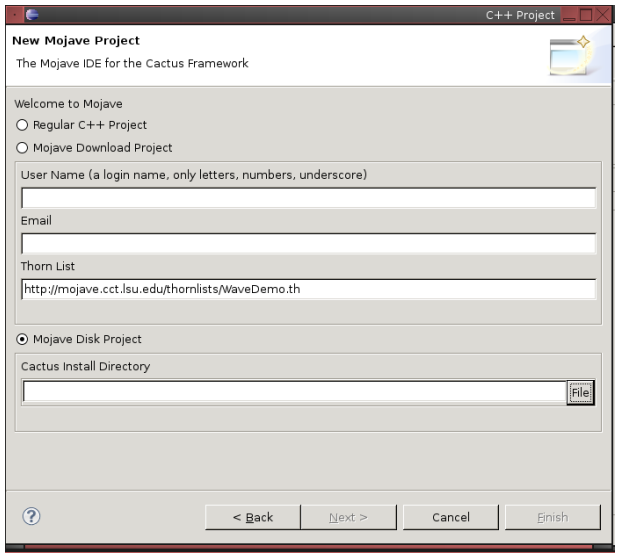
- For the par option:
  - Select Browse – Cactus – par – life.par



Running Cactus

To run Conway's Life through Mojave:

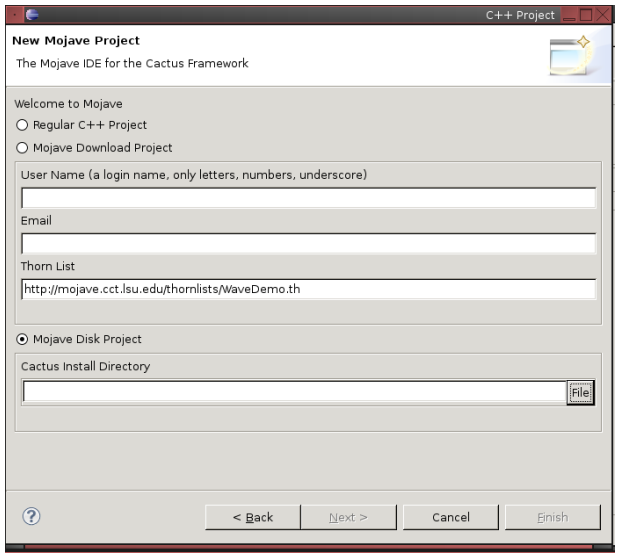
- For the par option:
  - Select Browse – Cactus – par – life.par
- For the thornlist option:



Running Cactus

To run Conway's Life through Mojave:

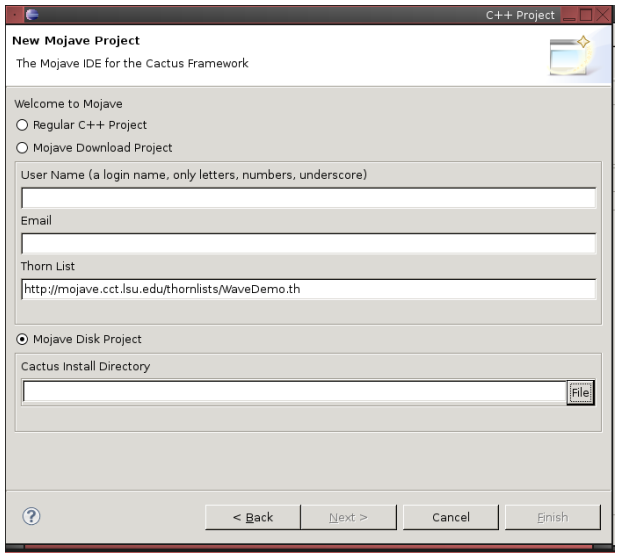
- For the par option:
  - Select Browse – Cactus – par – life.par
- For the thornlist option:
  - Select Browse – Cactus – thornlists – BadWaves.th

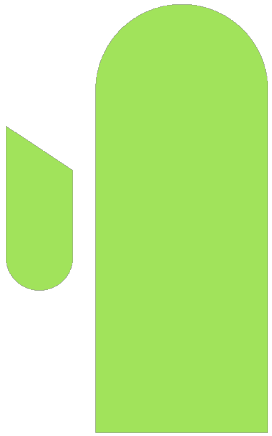


Running Cactus

To run Conway's Life through Mojave:

- For the par option:
  - Select Browse – Cactus – par – life.par
- For the thornlist option:
  - Select Browse – Cactus – thornlists – BadWaves.th
- Leave the other variables as they are





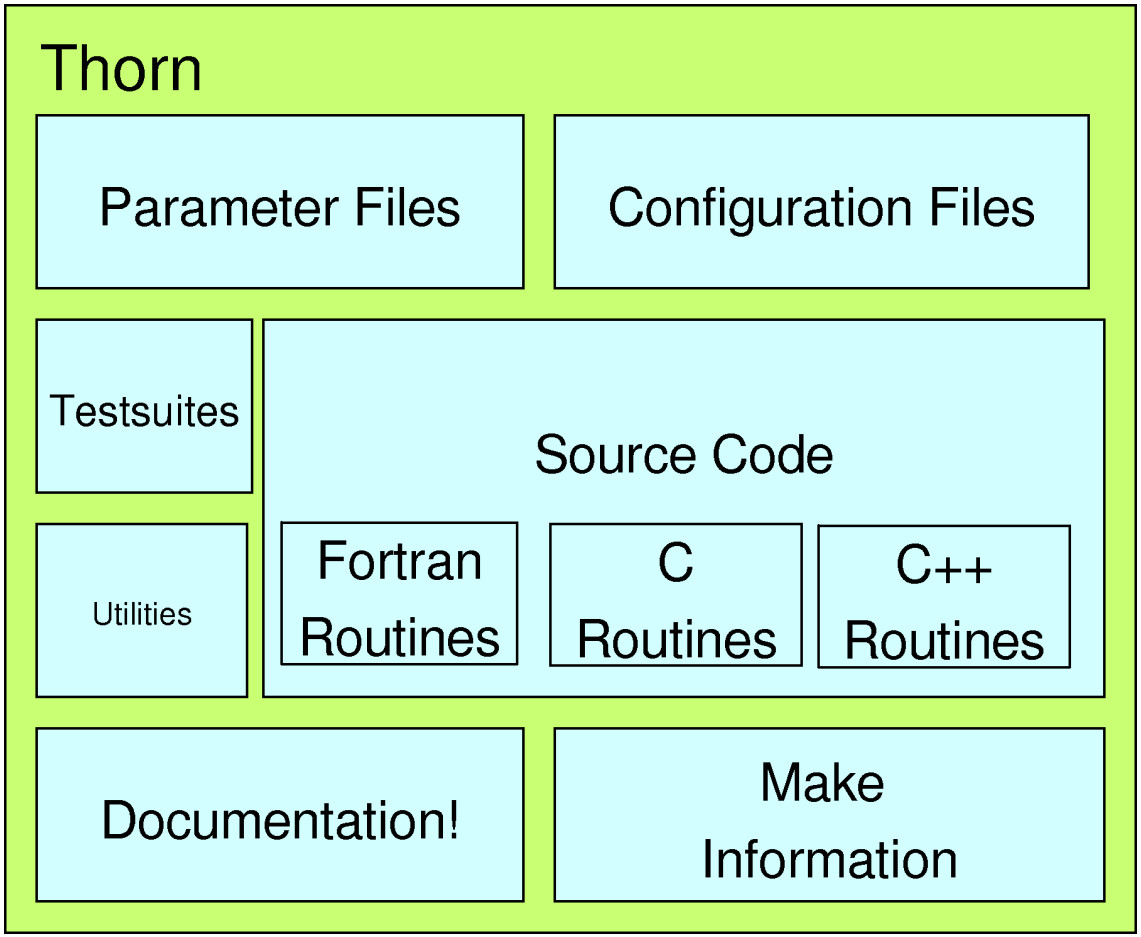
Writing Cactus Thorns

Plan:

- Thorn structure
- HelloWorld thorn
- Conway's Game of Life
  - Standalone code
  - The `Life thorn`
- Solving the Wave Equation
  - Standalone code
  - The BadWave thorn
  - Advanced thorns: PUGH, MoL, AMR

Thorn Structure

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



Thorn Specification

Thorn configuration files:

Thorn Specification

Thorn configuration files:

- interface.ccl declares:
  - an 'implementation' name
  - inheritance relationships between thorns
  - Thorn variables
  - Global functions, both provided and used

Thorn Specification

Thorn configuration files:

- interface.ccl declares:
  - an 'implementation' name
  - inheritance relationships between thorns
  - Thorn variables
  - Global functions, both provided and used
- schedule.ccl declares:
  - When the flesh should schedule which functions
  - When which variables should be allocated/freed
  - Which variables should be syncronized when

Thorn Specification

Thorn configuration files:

- interface.ccl declares:
  - an 'implementation' name
  - inheritance relationships between thorns
  - Thorn variables
  - Global functions, both provided and used
- schedule.ccl declares:
  - When the flesh should schedule which functions
  - When which variables should be allocated/freed
  - Which variables should be syncronized when
- param.ccl declares:
  - Runtime parameters for the thorn
  - Use/extension of parameters of other thorns

Writing a Hello World thorn

We now demonstrate the process of writing a thorn using a simple Hello World example.

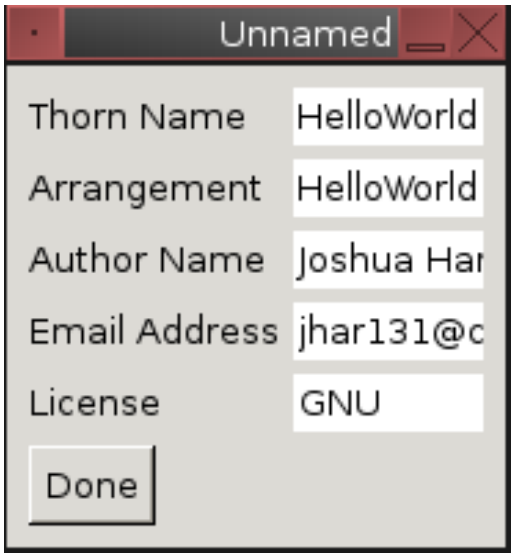
Here is the standalone C code for a Hello World program:

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

Creating a New Thorn

To Create A New Thorn:

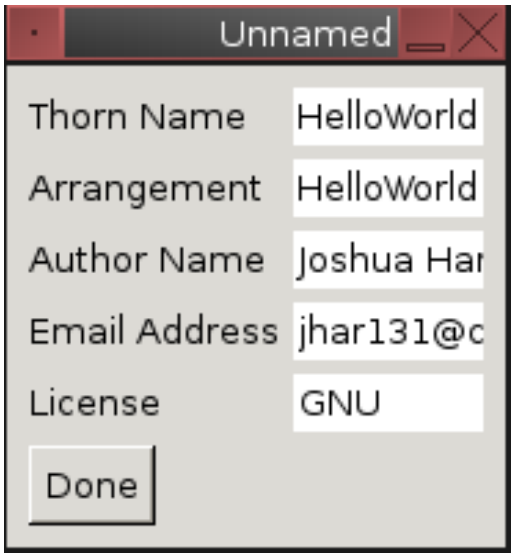
- Select Mojave – NewThorn from the drop down tab



Creating a New Thorn

To Create A New Thorn:

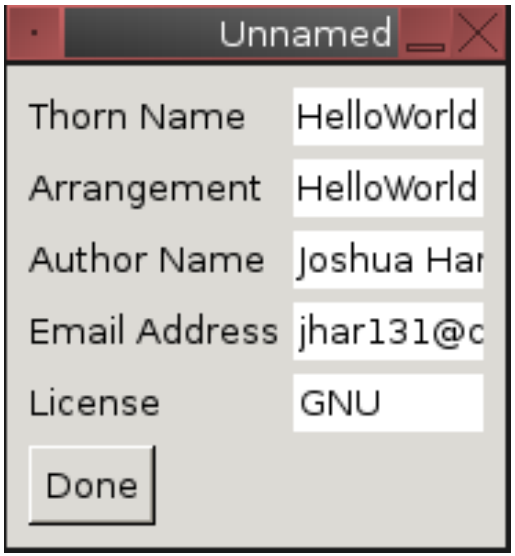
- Select Mojave – NewThorn from the drop down tab
- Title NewThorn and Arrangement as HelloWorld



Creating a New Thorn

To Create A New Thorn:

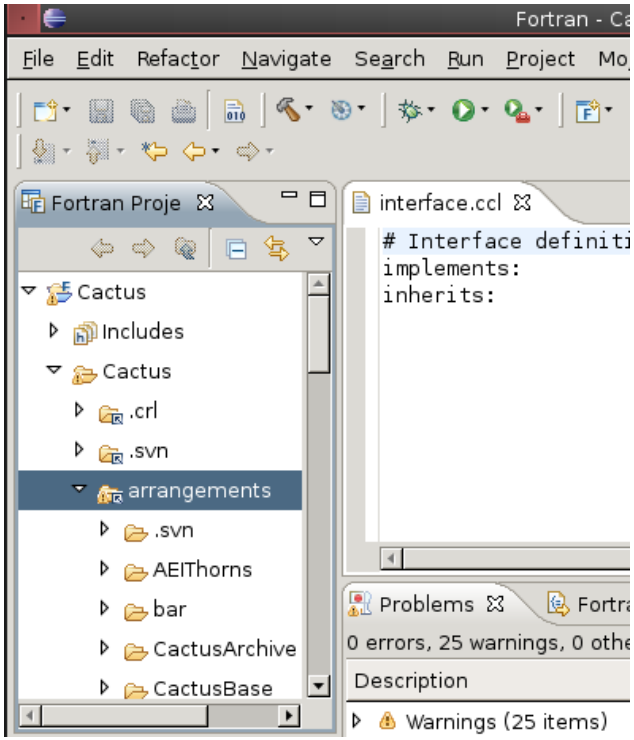
- Select Mojave – NewThorn from the drop down tab
- Title NewThorn and Arrangement as HelloWorld
- Fill the other fields as you see fit



Creating a New Thorn

From the Fortran Projects column:

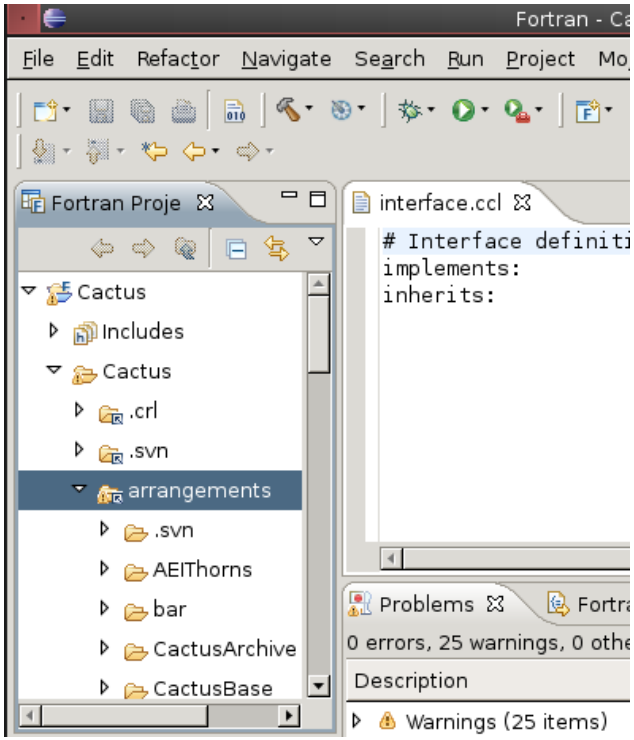
- Expand arrangements



Creating a New Thorn

From the Fortran Projects column:

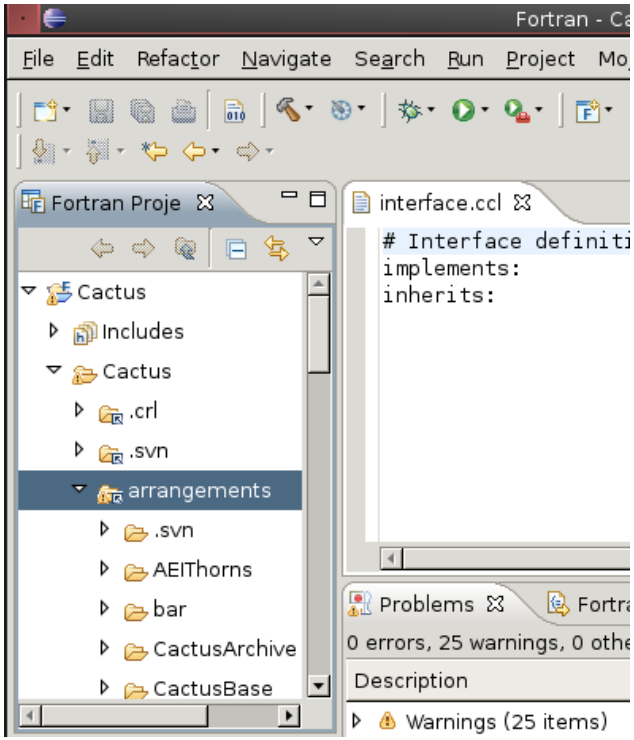
- Expand arrangements
- Expand the HelloWorld arrangement and thorn



Creating a New Thorn

From the Fortran Projects column:

- Expand arrangements
- Expand the HelloWorld arrangement and thorn
- Open interface.ccl and schedule.ccl



Hello World Thorn

Copy the following into each CCL file:

- `interface.ccl`:

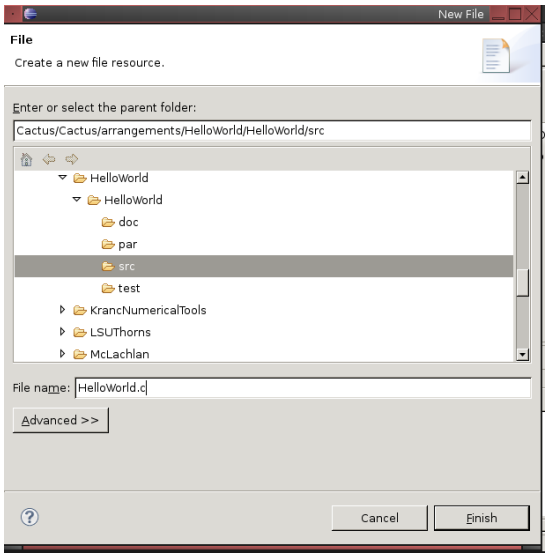
```
implements: HelloWorld
```

- `schedule.ccl`:

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

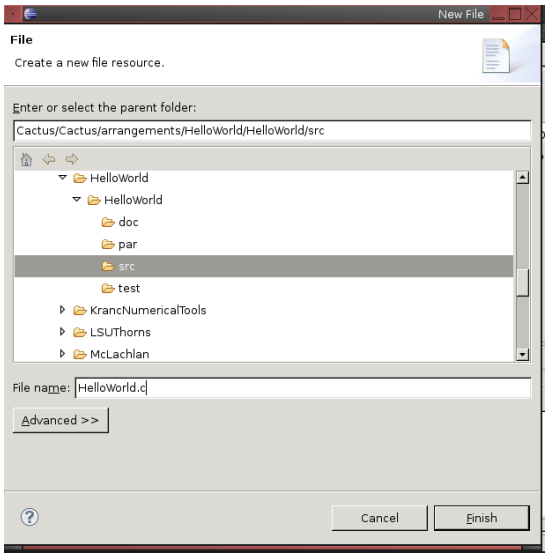
- `param.ccl`: empty

Running Cactus



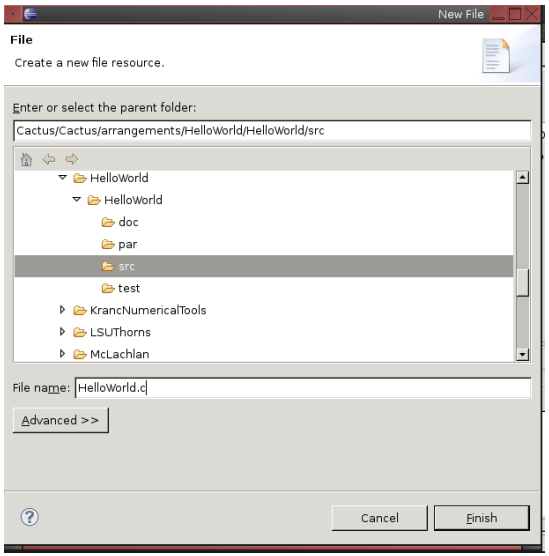
- Expand the src folder

Running Cactus



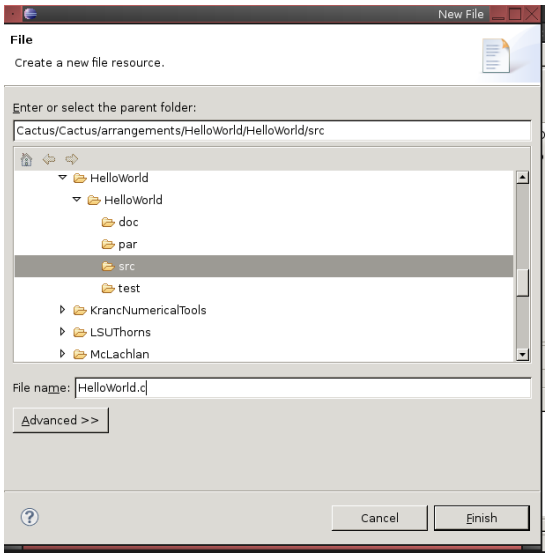
- Expand the src folder
- Right-Click src and select New – File

Running Cactus



- Expand the src folder
- Right-Click src and select New – File
- Make sure the parent folder is correct

Running Cactus



- Expand the src folder
- Right-Click src and select New – File
- Make sure the parent folder is correct
- Title the file: HelloWorld.c

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
```

Running Cactus

- Move to the Cactus/Cactus/par subdirectory
- Create a new file: hello.par
- `hello.par`:

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

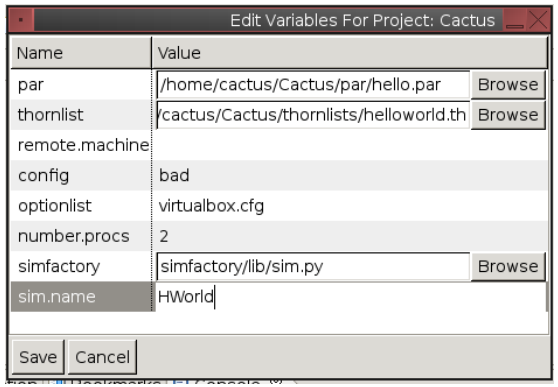
Running Cactus

- Move to the Cactus/Cactus/thornlists subdirectory
- Create a new file: hello.th
- `hello.th`:

```
HelloWorld/HelloWorld
```

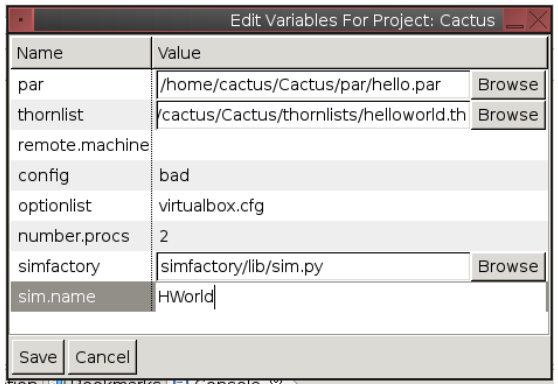
Running Cactus

- Select Mojave from the drop-down tab.



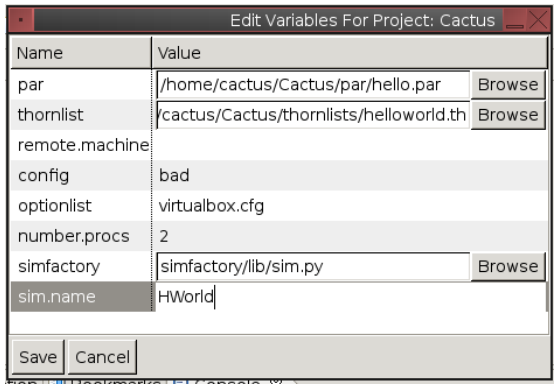
Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables



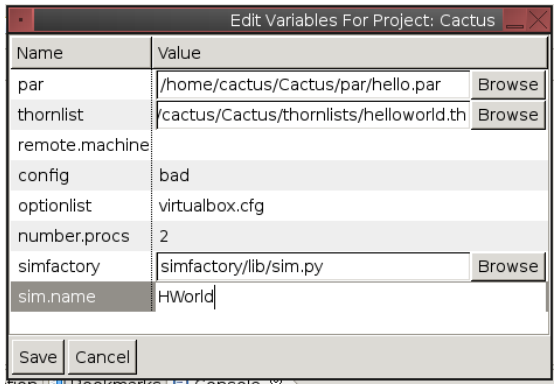
Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables
- Using the Browse button input:



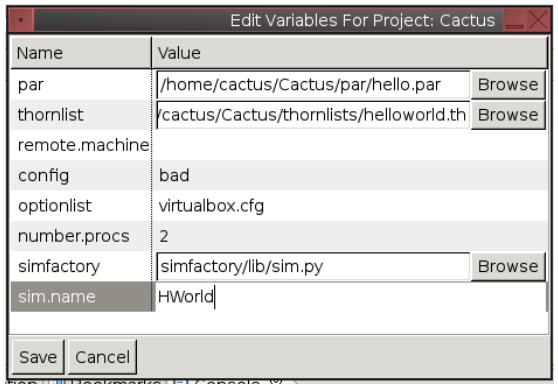
Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables
- Using the Browse button input:
  - hello.par in the par field



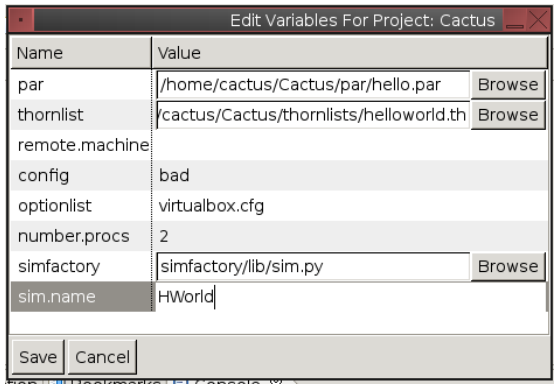
Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables
- Using the Browse button input:
  - hello.par in the par field
  - helloworld.th in the thornlist field



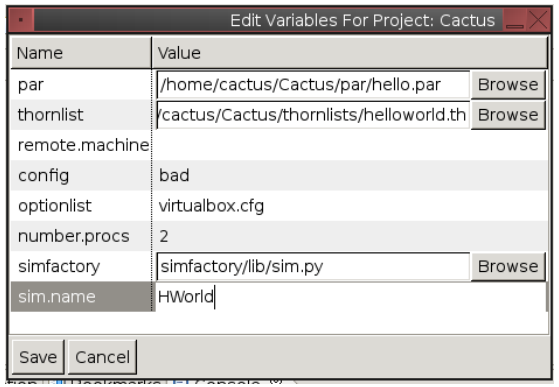
Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables
- Using the Browse button input:
  - hello.par in the par field
  - helloworld.th in the thornlist field
- Input bad2 in the config field



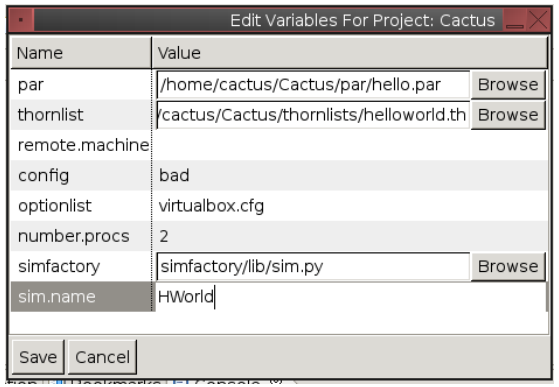
Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables
- Using the Browse button input:
  - hello.par in the par field
  - helloworld.th in the thornlist field
- Input bad2 in the config field
- Input HWorld in the sim.name field

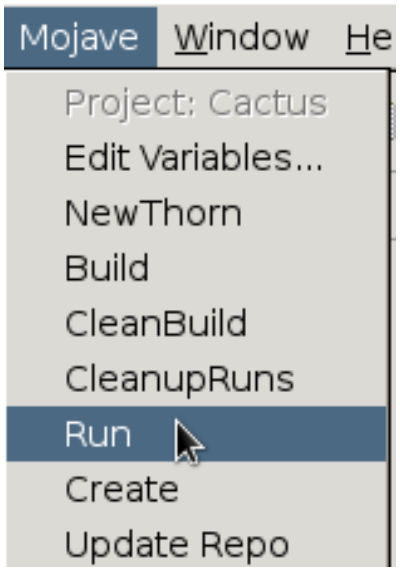


Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables
- Using the Browse button input:
  - hello.par in the par field
  - helloworld.th in the thornlist field
- Input bad2 in the config field
- Input HWorld in the sim.name field
- Select Save

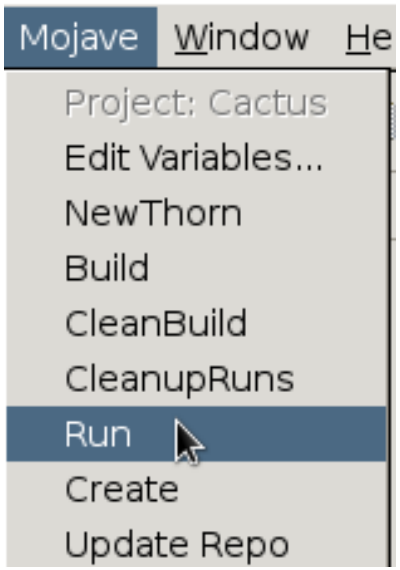


Running HelloWorld



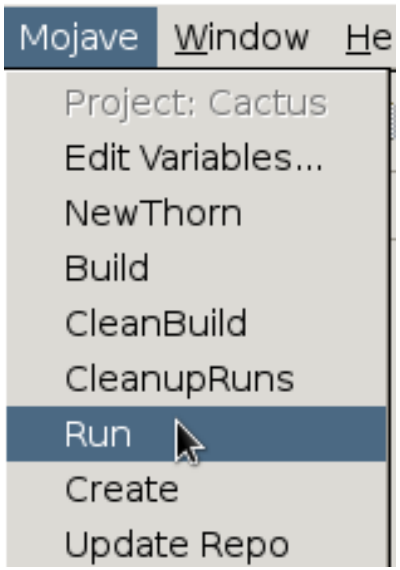
- To run HelloWorld:
  - From the Mojave drop down tab:

Running HelloWorld



- To run HelloWorld:
  - From the Mojave drop down tab:
  - Select Create

Running HelloWorld



- To run HelloWorld:
  - From the Mojave drop down tab:
  - Select Create
  - Select Run

Hello World Thorn

- Screen output:

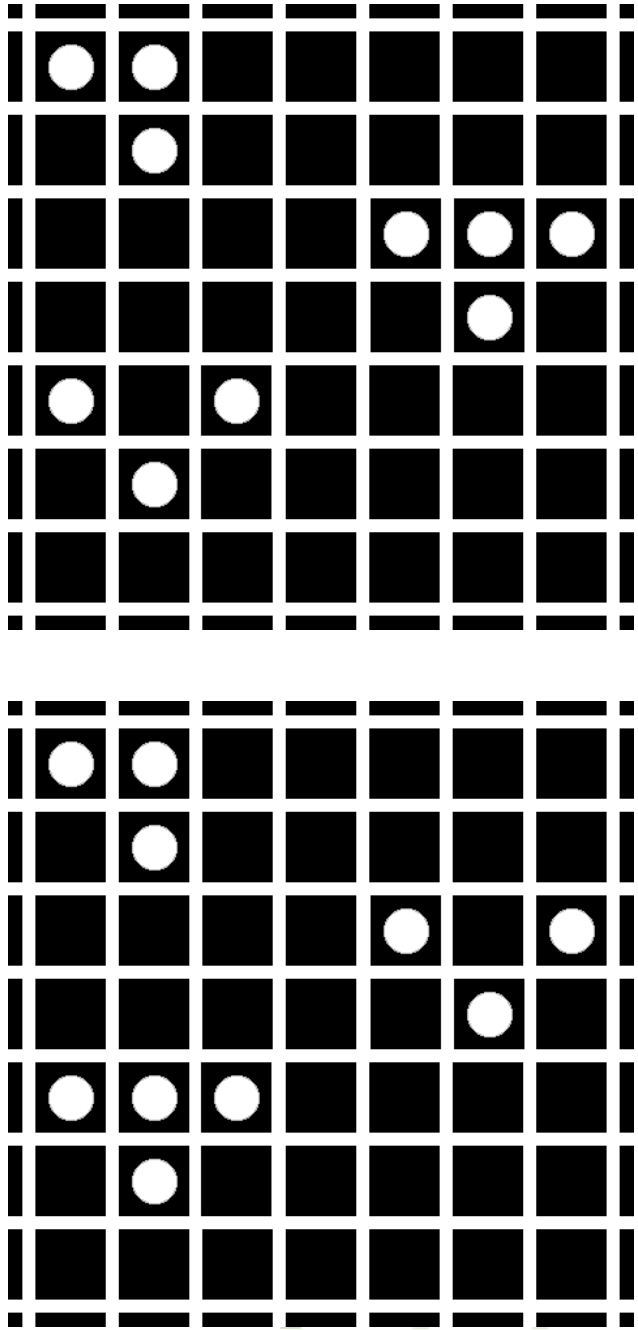
```
      10
  1   0101       ************************
  01  1010 10      The Cactus Code V4.0
 1010 1101 011      www.cactuscode.org
  1001 100101    ************************
    00010101
     100011     (c) Copyright The Authors
      0100      GNU Licensed. No Warranty
      0101

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.
```

Conway's Game of Life

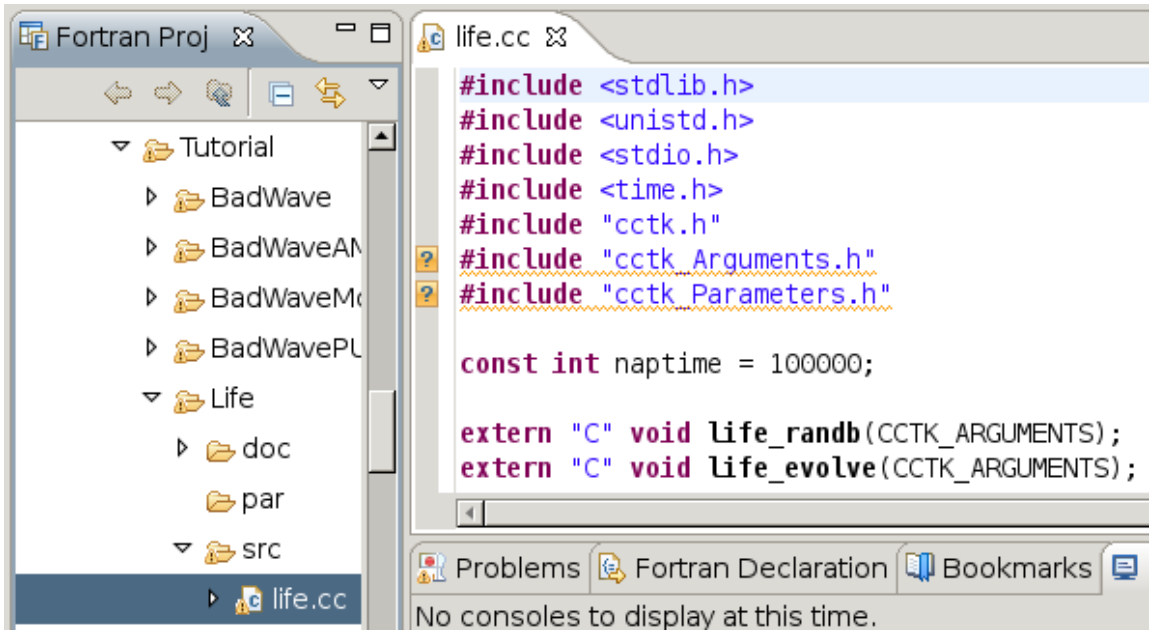
Conway's Game of Life

- Conway's Game of Life begins with a board of single-celled organisms. For each step of the game, a given cell lives if it is surrounded by two or three other cells. If a pixel on the board is surrounded by three cells, a cell is born on that pixel. If a living cell is surrounded by more than three cells, it dies.
- **Algorithm Conway**:
  1. Each element in an NxM matrix is initialized to 0 or 1.
  2. For each step of the game, generate a new board as follows: for each pixel on the current board, the value for the pixel for the new board is set as follows:
     1. If this pixel contains a 1 and if two or three 1's occupy any of its eight neighboring pixels, set the new value to 1.
     2. If this pixel contains a 0 and if three 1's occupy any of its eight neighboring pixels, set the new value to 1.
     3. If this pixel contains a 1 and the number of neighboring 1's is less than two or more than three, set the new value to 0.
     4. Otherwise set the new value to 0.

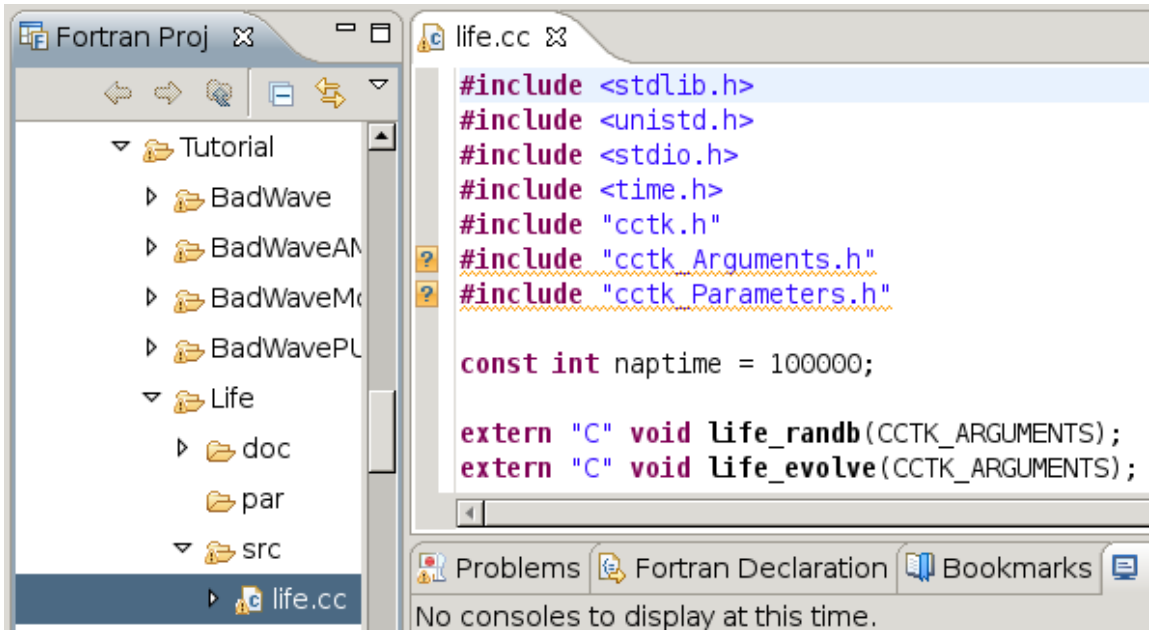
Consider the grid to the right. The cells in the upper left-hand corner each live because they each have two neighboring cells. In the upper-right hand corner, the cell surrounded by the three others dies from overcrowding on this iteration. In the lower left-hand corner, the space surrounded by three others bears a living cell.



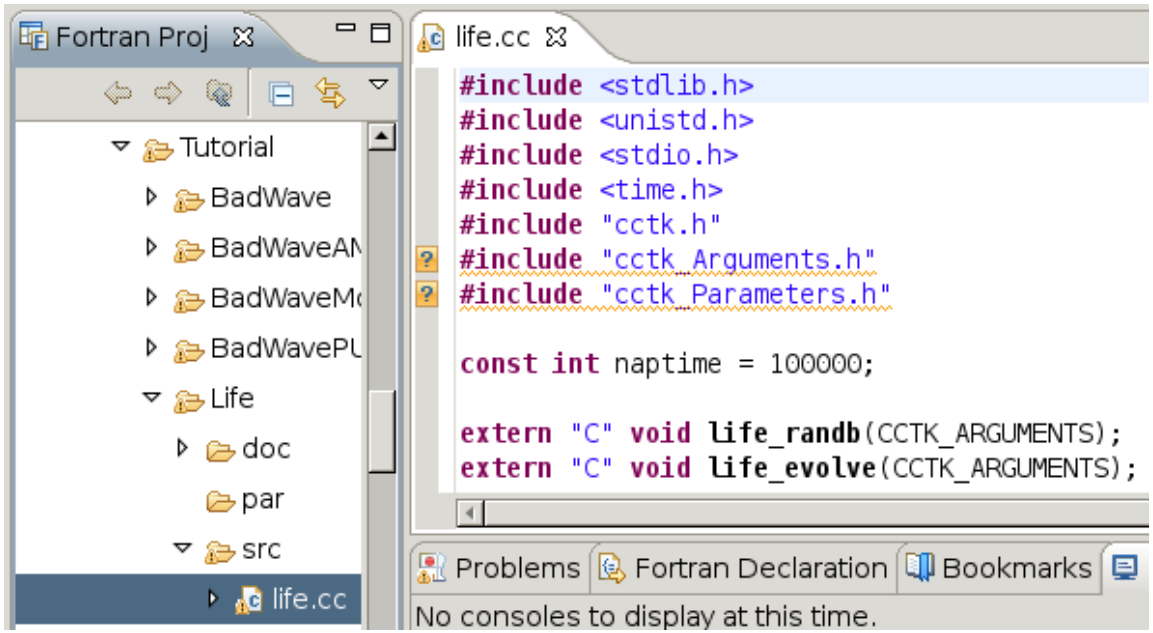
- The Cactus code for Conway's Game of Life can be found:



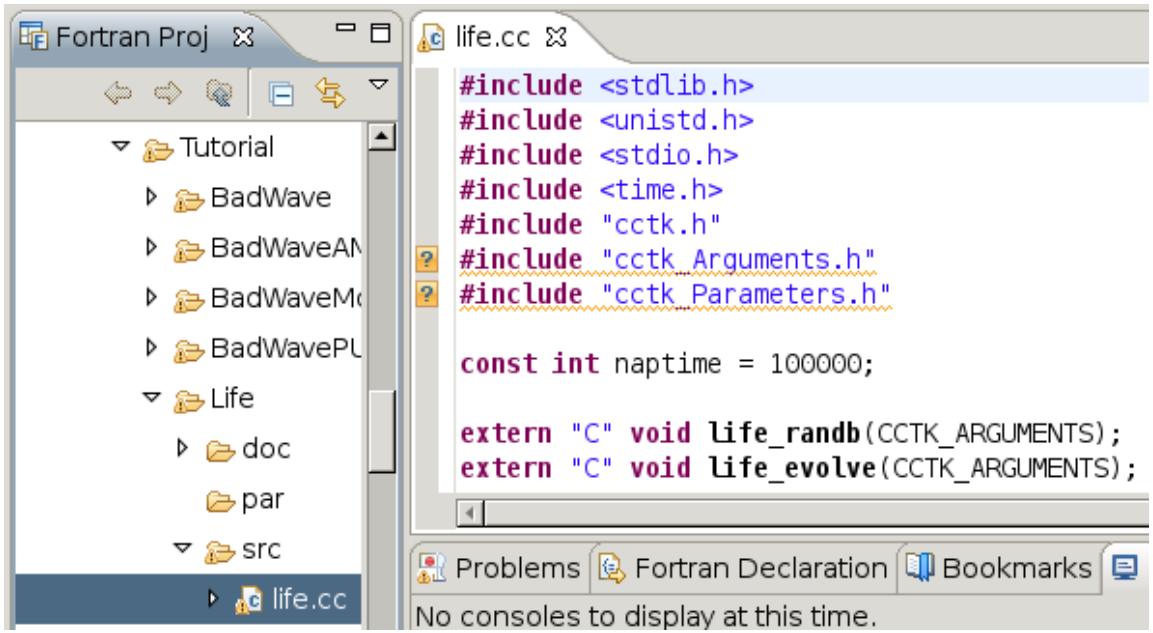
- The Cactus code for Conway's Game of Life can be found:
- Cactus/arrangements/Tutorial/Life/src/life.cc



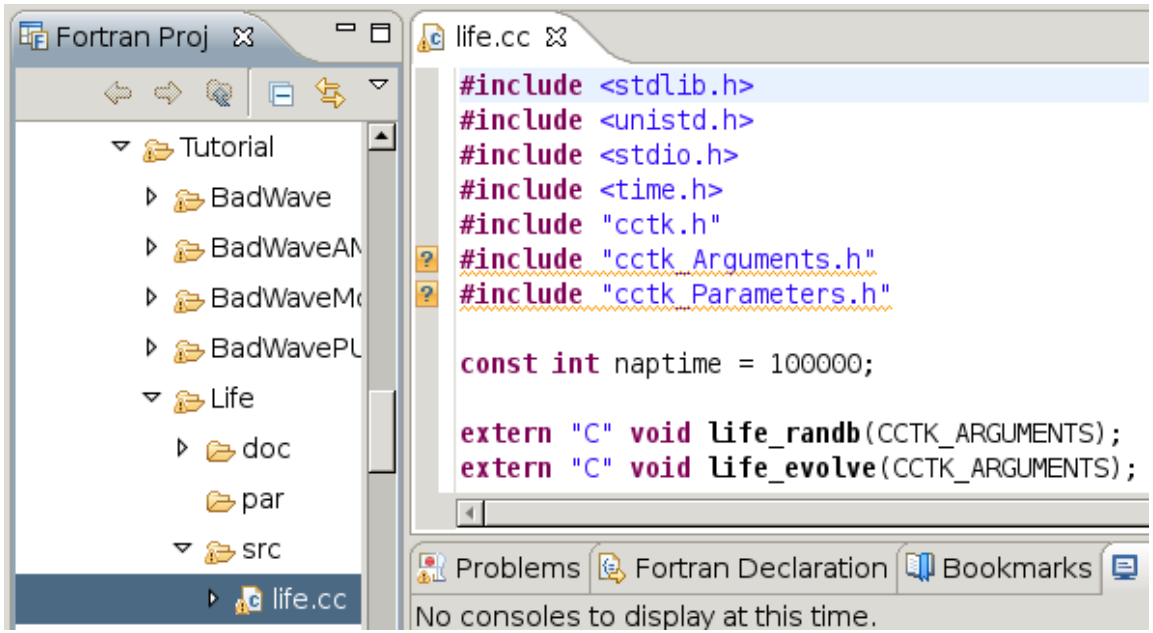
- The Cactus code for Conway's Game of Life can be found:
- Cactus/arrangements/Tutorial/Life/src/life.cc
- Notable alterations:



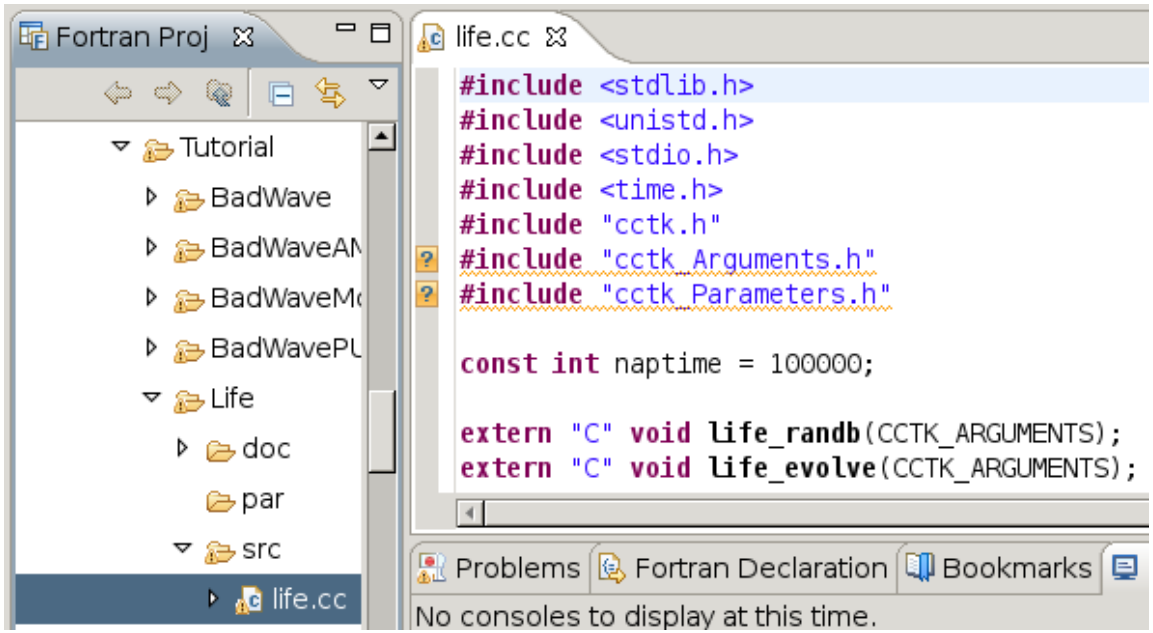
- The Cactus code for Conway's Game of Life can be found:
- Cactus/arrangements/Tutorial/Life/src/life.cc
- Notable alterations:
  - The CCTK arguements and parameters declarations.



- The Cactus code for Conway's Game of Life can be found:
- Cactus/arrangements/Tutorial/Life/src/life.cc
- Notable alterations:
  - The CCTK arguements and parameters declarations.
  - The removal of the print function. (void pboard)



- The Cactus code for Conway's Game of Life can be found:
- Cactus/arrangements/Tutorial/Life/src/life.cc
- Notable alterations:
  - The CCTK arguements and parameters declarations.
  - The removal of the print function. (void pboard)
  - The thorn IOJpeg will be used for output.



Notable Alterations:

Additional Header files for communication with Cactus:

- C code:

```
1 #include <stdlib.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <time.h>
```

- Cactus code:

```
1  #include <stdlib.h>
2  #include <unistd.h>
3  #include <stdio.h>
4  #include <time.h>
5  #include "cctk.h"                    //three additional headers
6  #include "cctk_Arguments.h"
7  #include "cctk_Parameters.h"
```

Notable Alterations:

Using Cactus' GFINDEX2D and the removal of INDEX2D:

- C code:

```
32    int cc = INDEX2D(i,j);
```

```
59    int cc = INDEX2D(i,j);
```

- Cactus code:

```
22   int cc = CCTK_GFINDEX2D(cctkGH, i, j);
```

```
36   int cc = CCTK_GFINDEX2D(cctkGH, i, j);
```

Definition Files

- `interface.ccl`:

```
# Interface definition for thorn Life
implements: life
inherits:

CCTK_INT boards TYPE=gf
{
        b, nb
} "boards"
```

- `param.ccl`:

```
# Parameter definitions for thorn Life

CCTK_INT random_seed "Initializer for the random number generator"
{
        *:* :: "Determines the starting position for the game of life"
} 0
```

Conways Game of Life: Cactus Implementation

- make.code.defn:

```
# Main make.code.defn file for thorn Life

# Source files in this directory
SRCS = life.cc

# Subdirectories containing source files
SUBDIRS =
```

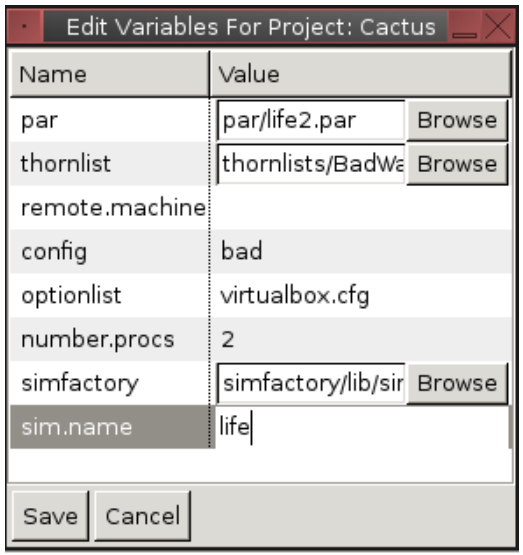
- schedule.ccl:

```
# Schedule definitions for thorn Life

storage: boards

schedule life_randb at INITIAL
{
        LANG: C
        SYNC: boards
} "setup board"

schedule life_evolve at EVOL
{
        LANG: C
        SYNC: boards
} "another generation lives or dies"
```

Running Cactus

To run Conway's Life through Mojave:

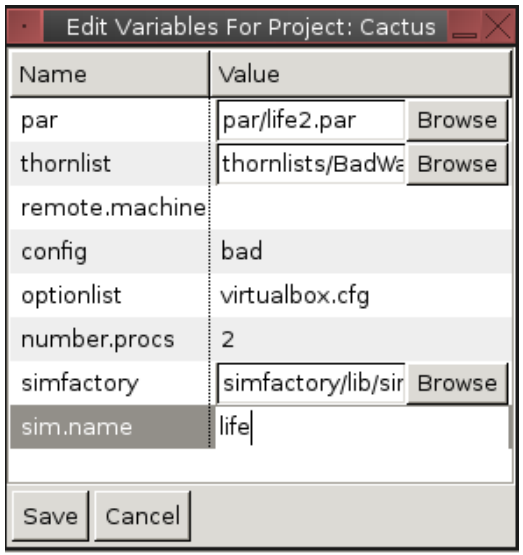
- Select Edit Variables from the Mojave tab



Running Cactus

To run Conway's Life through Mojave:

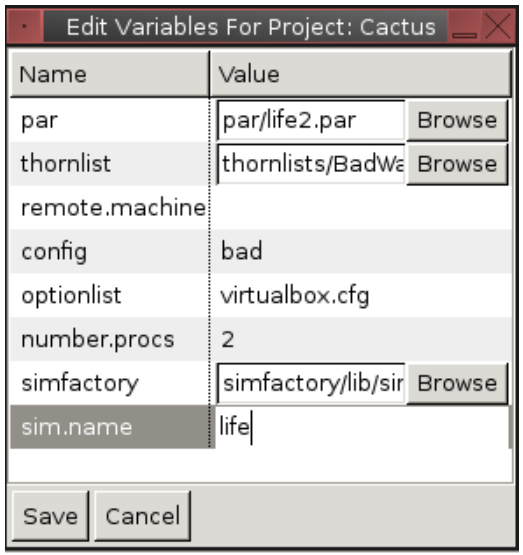
- Select Edit Variables from the Mojave tab
- For the par option:



Running Cactus

To run Conway's Life through Mojave:

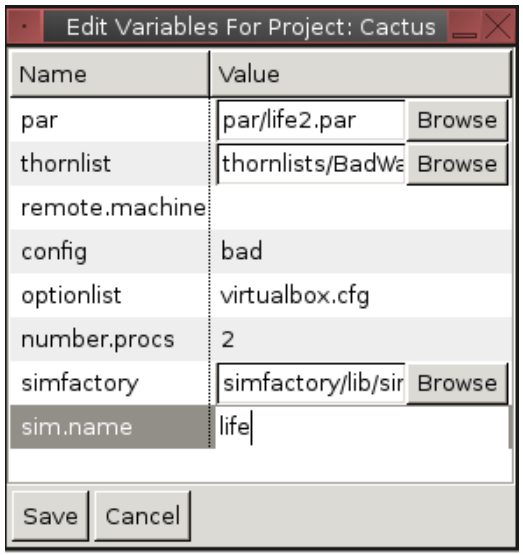
- Select Edit Variables from the Mojave tab
- For the par option:
  - Select Browse – Cactus – par – life2.par



Running Cactus

To run Conway's Life through Mojave:

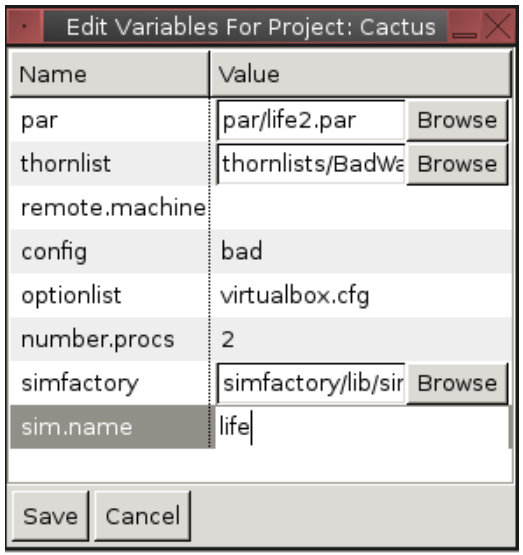
- Select Edit Variables from the Mojave tab
- For the par option:
  - Select Browse – Cactus – par – life2.par
- For the config option:



Running Cactus

To run Conway's Life through Mojave:

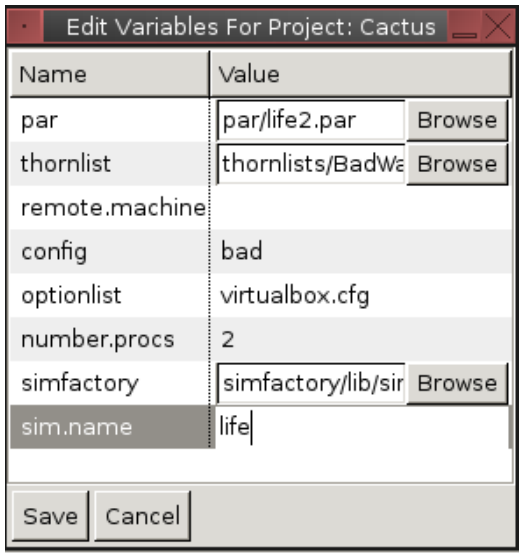
- Select Edit Variables from the Mojave tab
- For the par option:
  - Select Browse – Cactus – par – life2.par
- For the config option:
  - Input: bad



Running Cactus

To run Conway's Life through Mojave:

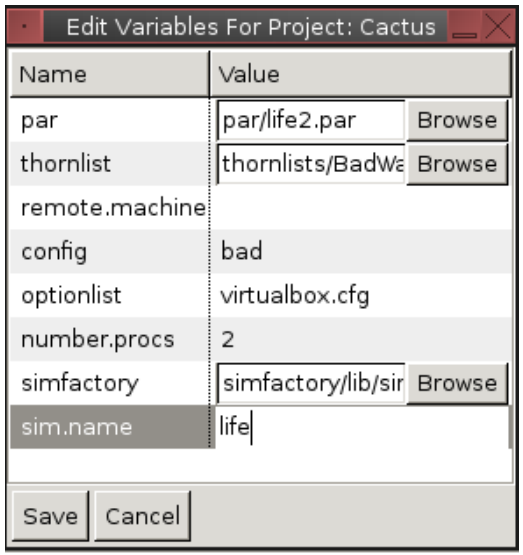
- Select Edit Variables from the Mojave tab
- For the par option:
  - Select Browse – Cactus – par – life2.par
- For the config option:
  - Input: bad
- Name the simulation life2



Running Cactus

To run Conway's Life through Mojave:

- Select Edit Variables from the Mojave tab
- For the par option:
  - Select Browse – Cactus – par – life2.par
- For the config option:
  - Input: bad
- Name the simulation life2
- Mojave-Create and Mojave-Run



Visualizing Conway's Life Results

- After running Conway's Life, you may navigate to the `simulations/life2/output-*` directory to see the JPEG image output.
- To animate the images, use the ImageMagick command `animate`. Since your VM uses minimal RAM, you may wish to animate only a fraction of the images:
- `animate "_[0-9][0-9]\."`
- This will animate images 11-99.

Parallelization Framework in Cactus

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:

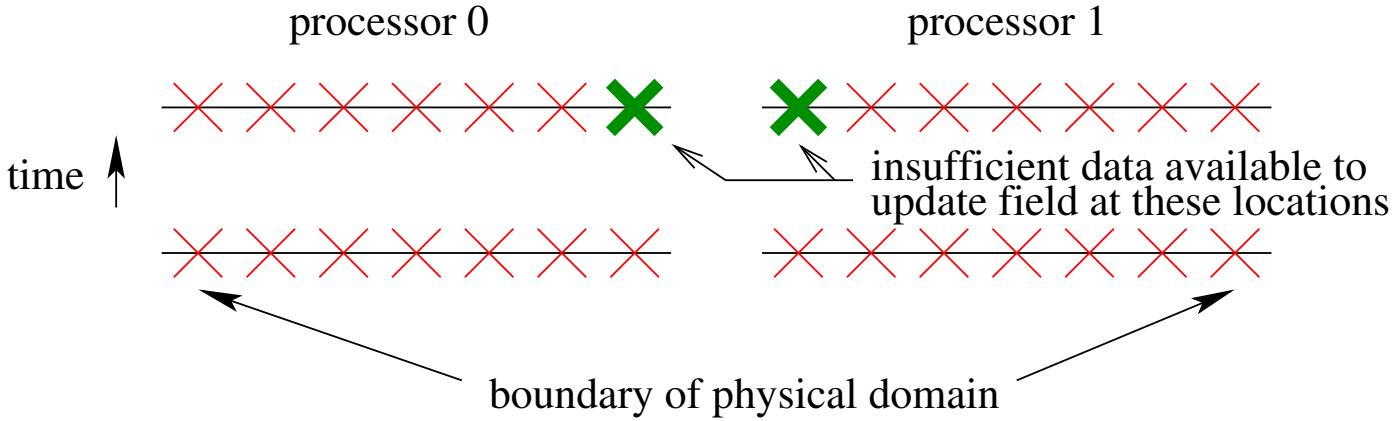
- Distribute variables across processes (grid function)
- Synchronize processor domain boundaries between processes
- Compute reductions across grid functions
- Actual implementation in driver thorn

Ghost Zones

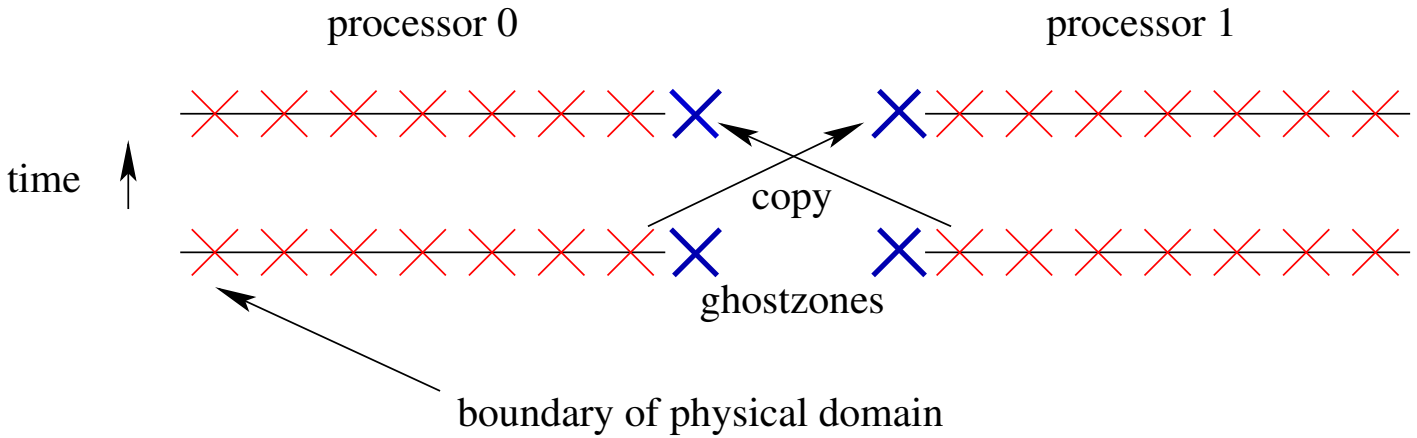
- Grid variables: distributed across processes
- Assumption: Most work done (quasi-) locally:
  True for hyperbolic and parabolic problems
- Split of computational domain into blocks
- Only communication at the borders (faces)
- At least stencil size many ghostzones

Ghost Zone Example

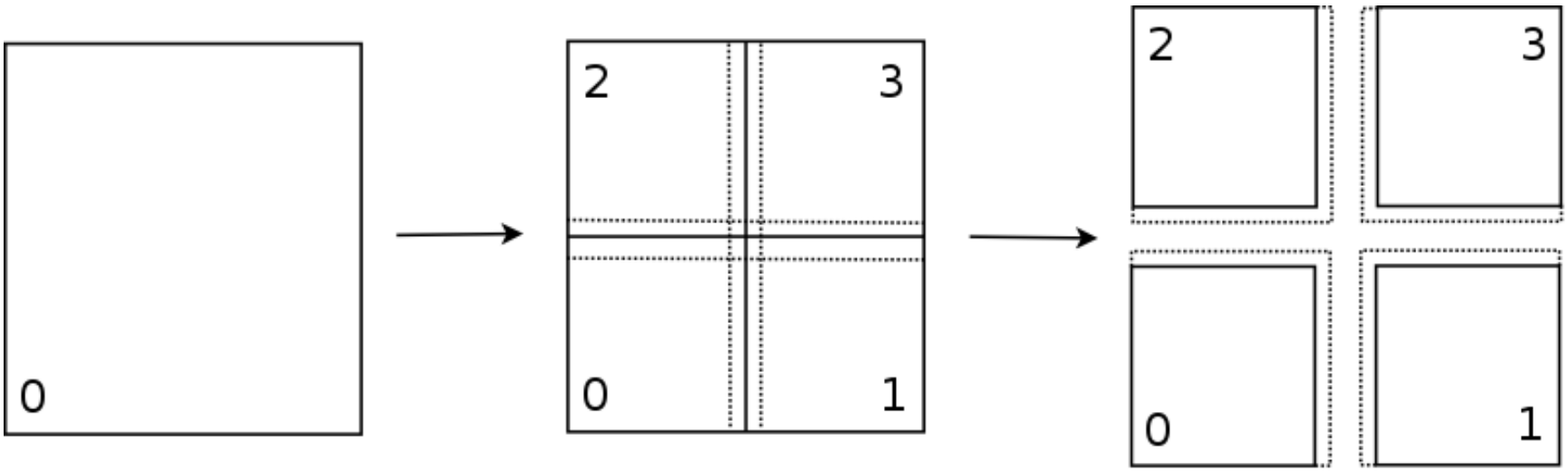
Without Ghostzones:



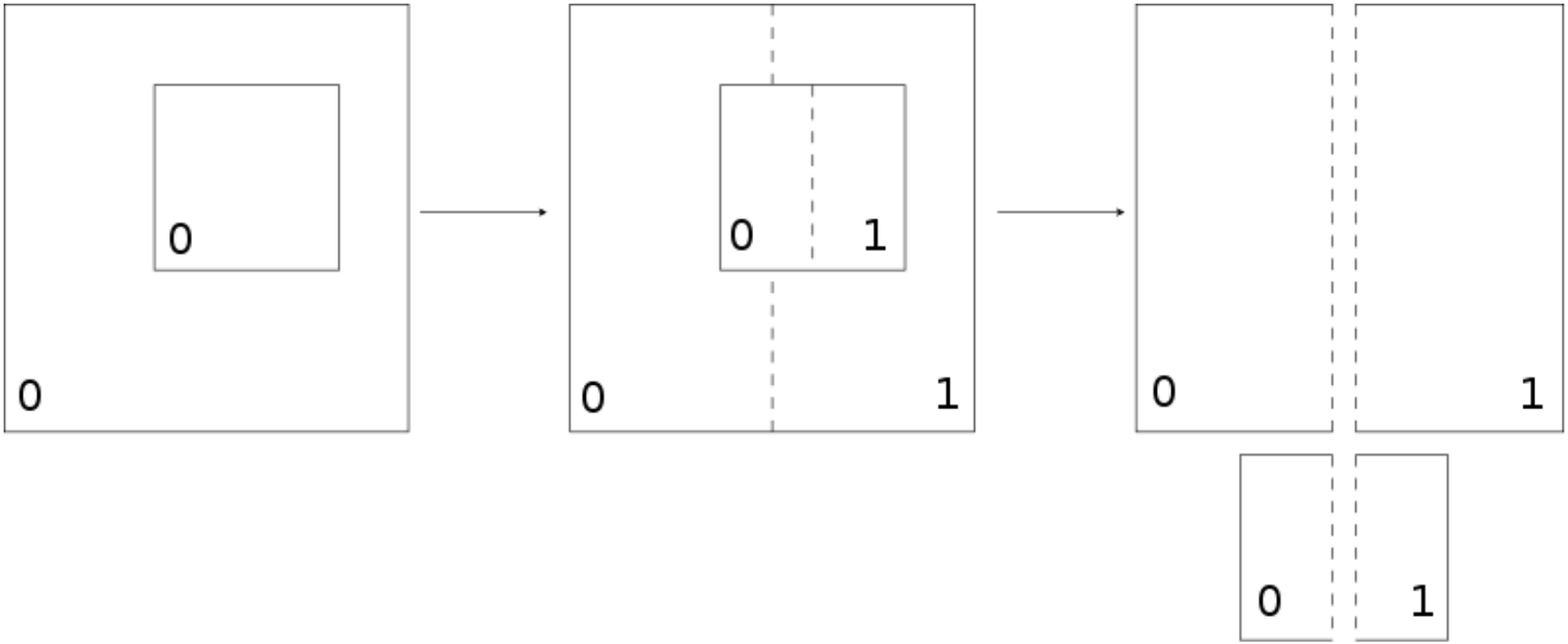
With Ghostzones:



Domain Decomposition



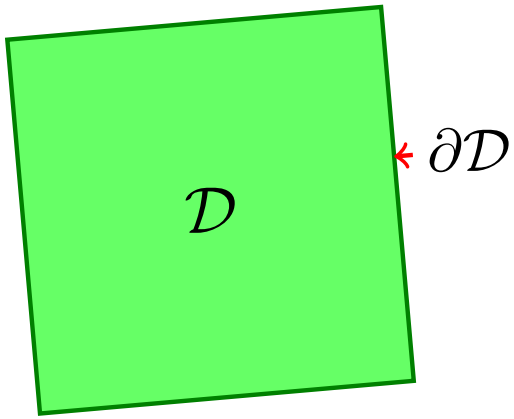
Mesh Refinement Decomposition



Solving the Wave Equation

The Wave Equation

- PDE which describes wave propagation in a medium
- one of the fundamental equations of mathematical physics
- *For a spatial domain $\mathcal{D}$, find a scalar field $\psi(x, y, z, t)$ which satisfies:*



$$\frac{\partial^2\psi}{\partial t^2} = c^2\nabla^2\psi \qquad \textit{inside } \mathcal{D}, \textit{ and}$$

$$B(\psi, \partial_i\psi)|_{\partial\mathcal{D}} = 0 \qquad \textit{at } \partial\mathcal{D}.$$

where c is the speed of the wave, $B(\psi, \partial_i\psi)$ specifies the boundary conditions, and ∇^2 is the Laplacian of ψ:

$$\nabla^2\psi = \left[\frac{\partial^2}{\quad} + \frac{\partial^2}{\quad} + \frac{\partial^2}{\quad}\right]\psi$$

First-order form

The wave equation is a second-order PDE for a single scalar function. To improve its numerical stability properties, we can rewrite it as a system of first-order PDEs:

$$\frac{\partial \psi}{\partial t} = c\vec{\nabla} \cdot \vec{p} \qquad \frac{\partial \vec{p}}{\partial t} = c\nabla\psi$$

with a new unknown vector variable $\vec{p}$. If $\vec{p} = \nabla\psi$, we recover the original scalar wave equation.

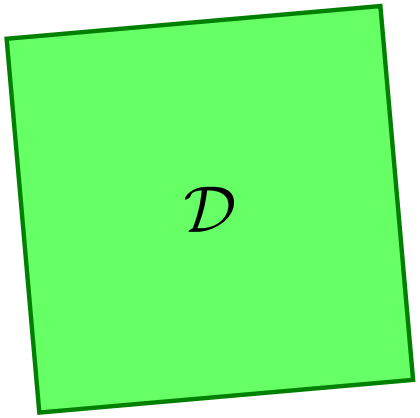
We are going to solve this system of PDEs inside a *cube* $\mathcal{D} = [-a, a]^3$ with the following boundary conditions:

$$\psi|_{\partial\mathcal{D}} = 0 \qquad \vec{p}_{\parallel}|_{\partial\mathcal{D}} = 0 \qquad \vec{p}_{\perp}|_{\partial\mathcal{D}_{i+}} = \vec{p}_{\perp}|_{\partial\mathcal{D}_{i-}}$$

where $\vec{p}_{\perp}$, $\vec{p}_{\parallel}$ are the (outward) normal and tangential components of $\vec{p}$, and $\partial\mathcal{D}_{i+}/\partial\mathcal{D}_{i-}$ are the opposite faces normal to the i-th axis.

Discretization

We will be solving the wave equation using finite difference (FD) methods on rectangular grids (adding mesh refinement later).



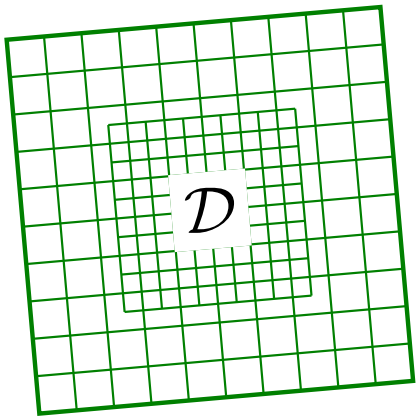
- spatial domain $\rightarrow$ rectangular grid
- continuous function $\rightarrow$ grid function
- partial derivative $\rightarrow$ FD operator
- PDE $\rightarrow$ system of algebraic equations

$$\left.\begin{aligned} \frac{\partial \psi}{\partial t} &= c\vec{\nabla}\cdot\vec{p} \\ \frac{\partial \vec{p}}{\partial t} &= c\nabla\psi \end{aligned}\right| \rightarrow \left|\begin{aligned} D_t\psi &= K_\psi := c(D_x p_x + D_y p_y + D_z p_z) \\ D_t p_x &= K_x := cD_x\psi \\ D_t p_y &= K_y := cD_y\psi \\ D_t p_z &= K_z := cD_z\psi \end{aligned}\right.$$

where D_t, D_x, D_y and D_z are the *finite differencing* operators.

Discretization

We will be solving the wave equation using finite difference (FD) methods on rectangular grids (adding mesh refinement later).



- spatial domain $\longrightarrow$ rectangular grid
- continuous function $\longrightarrow$ grid function
- partial derivative $\longrightarrow$ FD operator
- PDE $\longrightarrow$ system of algebraic equations

$$\left.\begin{aligned} \frac{\partial \psi}{\partial t} &= c\vec{\nabla}\cdot\vec{p} \\ \frac{\partial \vec{p}}{\partial t} &= c\nabla\psi \end{aligned}\right| \longrightarrow \left| \begin{aligned} D_t\psi &= K_\psi := c(D_x p_x + D_y p_y + D_z p_z) \\ D_t p_x &= K_x := cD_x\psi \\ D_t p_y &= K_y := cD_y\psi \\ D_t p_z &= K_z := cD_z\psi \end{aligned}\right.$$

where D_t, D_x, D_y and D_z are the *finite differencing* operators.

Spatial derivatives

We approximate spatial derivatives by finite differences of the grid function values at neighboring points. We will use a centered scheme, which is second-order accurate in grid spacing Δ:

$$(D_x\psi)_{i,j,k} = \frac{\psi_{i+1,j,k} - \psi_{i-1,j,k}}{2\Delta}$$

$$(D_y\psi)_{i,j,k} = \frac{\psi_{i,j+1,k} - \psi_{i,j-1,k}}{2\Delta}$$

$$(D_z\psi)_{i,j,k} = \frac{\psi_{i,j,k+1} - \psi_{i,j,k-1}}{2\Delta}$$

Runge-Kutta time integration

Simplest Runge-Kutta method: Newton integration (1st order accurate)

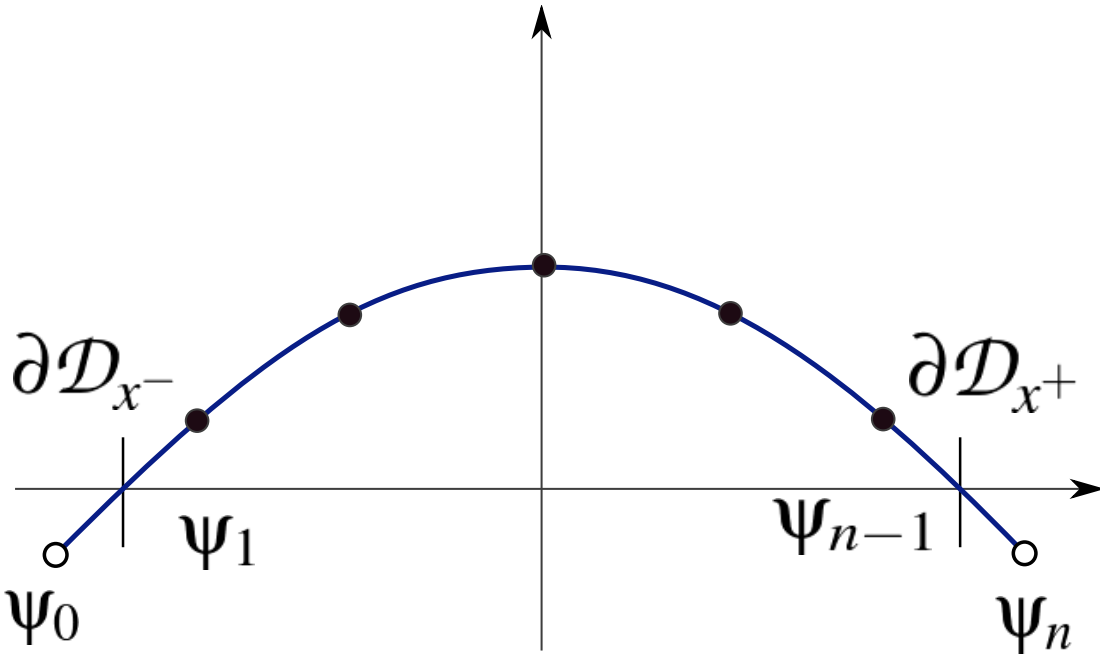
$$\frac{y_{n+1} - y_n}{\Delta t} = K_y(t_n, y_n) \quad \rightarrow \quad y_{n+1} = y_n + K_y(t_n, y_n)\Delta t$$

We will use a 2nd order accurate Runge-Kutta integration scheme (also known as a *Heun method*) with two intermediate steps:

$$\begin{aligned} k_1 &= K_y(t_n, y_n) \\ \tilde{y}_n &= y_n + k_1 \Delta t, \\ k_2 &= K_y(t_n + \Delta t, \tilde{y}_n) \\ y_{n+1} &= y_n + \left(\frac{1}{2} k_1 + \frac{1}{2} k_2 \right) \Delta t \end{aligned}$$

Discrete boundary conditions

We place the grid points in such a way that the boundary of the domain $\partial\mathcal{D}$ lies in between the two last grid points:

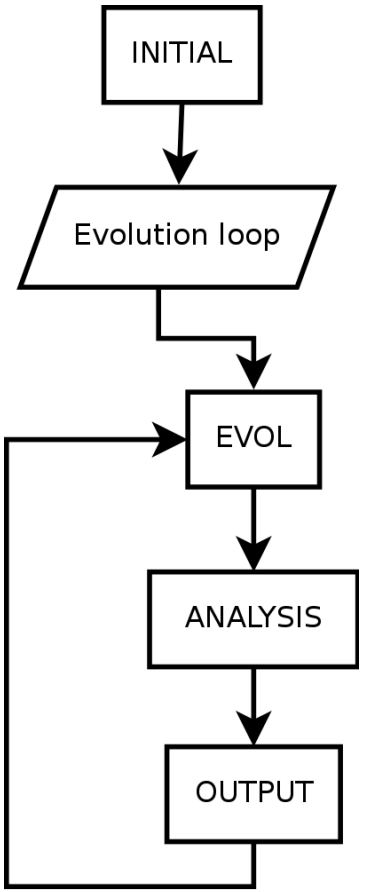


$$
\begin{aligned}
\psi|_{\partial\mathcal{D}} &= 0 \\
\vec{p}_{\parallel}|_{\partial\mathcal{D}} &= 0 \\
\vec{p}_{\perp}|_{\partial\mathcal{D}_{i+}} &= \vec{p}_{\perp}|_{\partial\mathcal{D}_{i-}} \\
&\Downarrow \\
\psi_{0,j,k} &= -\psi_{1,j,k}, \\
\vec{p}_{y,0,j,k} &= -\vec{p}_{y,1,j,k}, \\
\vec{p}_{z,0,j,k} &= -\vec{p}_{z,1,j,k}, \\
\vec{p}_{x,0,j,k} &= \vec{p}_{x,1,j,k},
\end{aligned}
$$

... etc.

Standalone code

```
int main(int argc,char **argv) {
    FILE *fp = fopen("psi.d.asc","w+");
    BadWave_Init();
    BadWave_ApplyBounds();
    BadWave_print(fp);
    // evolve 100 steps
    for(int i=0;i<100;i++) {
        iter = i;
        BadWave_Evolve();
        BadWave_ApplyBounds();
        BadWave_Evolve2();
        BadWave_TmpApplyBounds();
        BadWave_print(fp);
    }
    fclose(fp);
    return 0;
}
```



Standalone code: parameters and definitions

```
// Parameters:
bool verbose = true;      // enable verbose output
double wave_speed = 1.0; // wave speed factor
const int isiz = 32, jsiz = 32, ksiz = 32; //grid size

// 1D array to hold the grid and a function to find 3D indices
typedef double gridfunc[isiz*jsiz*ksiz];
inline int INDEX3D(int i,int j,int k) { return (i+j*isiz)*ksiz+k

// Grid functions:
gridfunc psi,px,py,pz;              // psi and p
gridfunc tpsi,tpx,tpy,tpz;          // temporary variables
gridfunc kpsi,kpx,kpy,kpz;          // the kernels
gridfunc k2psi,k2px,k2py,k2pz;

// Other global variables: iteration, grid spacing and time step
int iter=0;    double dx, dy, dz, dt;
```

Writing thorn BadWave: param.ccl

```
# Parameter definitions for thorn BadWave

private:
CCTK_REAL wave_speed "characteristic speed at boundary"
{
  "0.1:*" :: "wave speed"
} 1.

private:
CCTK_BOOLEAN verbose "whether to print stuff"
{
    :: "verbose flag"
} 0
```

Writing thorn BadWave: interface.ccl

```
# Interface definition for thorn BadWave
implements: BadWaveTutorial
inherits: grid

CCTK_REAL wave_vars TYPE=gf
{
    psi
    px, py, pz
} "Basic components of wave equation code"

CCTK_REAL tmp_wave_vars TYPE=gf { tpsi, tpx, tpy, tpz } <.
CCTK_REAL kernels TYPE=gf {
    kpsi kpx, kpy, kpz, k2psi, k2px, k2py, k2pz
} <...>
```

Writing thorn BadWave: schedule.ccl

```
# Schedule definitions for thorn BadWave
storage: wave_vars, tmp_wave_vars, kernels

schedule BadWave_Init at INITIAL
{ LANG: C
} "Startup and initialize"

schedule BadWave_ApplyBounds at POSTSTEP
{   LANG: C
    SYNC: wave_vars
} "Apply boundary conditions"

schedule BadWave_Evolve at EVOL <...>
schedule BadWave_TmpApplyBounds at EVOL after BadWave_Evolve <..
schedule BadWave_Evolve2 at EVOL after BadWave_TmpApplyBounds <.
```

Adapting the source code for Cactus

Modifications to standalone code:

- include Cactus headers:

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

- declare all scheduled functions with external C linkage:

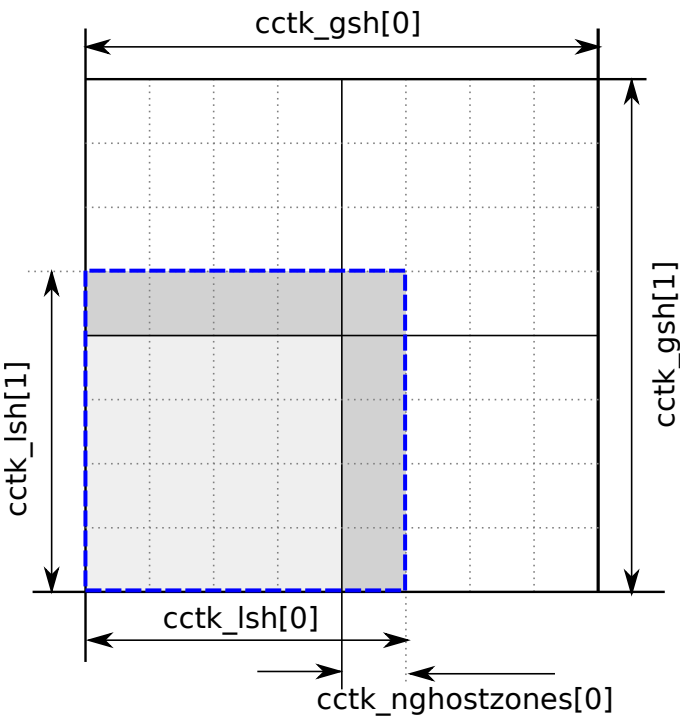
```
extern "C" void BadWave_Init(CCTK_ARGUMENTS);
extern "C" void BadWave_ApplyBounds(CCTK_ARGUMENTS);
extern "C" void BadWave_TmpApplyBounds(CCTK_ARGUMENTS);
extern "C" void BadWave_Evolve(CCTK_ARGUMENTS);
extern "C" void BadWave_Evolve2(CCTK_ARGUMENTS);
```

- add the CCTK_ARGUMENTS macro to scheduled functions:

```
void BadWave_Init(CCTK_ARGUMENTS)
{
    DECLARE_CCTK_ARGUMENTS;
    ...
}
```

this tells functions which part of the grid they will be working

Grid definitions in Cactus



- `cctk_gsh`: global grid dimensions
- `cctk_lsh`: local grid dimensions
- `cctk_delta_space`: grid spacing
- `cctk_nghostzones`: number of ghostzones
- `cctk_bbox`: whether a boundary is an outer one
- `CCTK_GFINDEX3D(i,j,k)`: computes local 1D grid index

Simple parameter file

Parfile: `par/BadWavePUGH.par`

- include the driver thorn (PUGH)
- specify grid size
- specify BadWave parameters

```
ActiveThorns = "BadWave PUGH"

cactus::cctk_itlast   = 100
pugh::global_nsize    = 32
pugh::ghost_size      = 1

BadWavePUGH::wave_speed = 2.0
```

More advanced parameter file

Parfile: `par/BadWavePUGHv2.par`
Includes new thorns:

- `CoordBase`: coordinate extents
- `CardGrid3D`: provides variety of grid configurations
- `SymBase`: basic thorn for specifying symmetries of the domain
- `IOBasic`, `IOUtil`, `IOScalar`, `IOASCII`: basic and advanced I/O
- `Time`: provides global time and iteration variables
- `PUGHSlab`, `PUGHReduce`, `LocalReduce`: required by I/O thorns

Method of Lines

- Method of lines is a general name for an approach of solving PDEs with time variable, in which the PDE is approximated by a system of interdependent ODEs for each grid point.
- Method of lines allows to extend time integration methods developed for ODEs to PDEs.
- Cactus thorn MoL implements several different time integration methods.
- We can use these methods if we modify our thorn: `BadWaveMoL`

Registering evolved variables with MoL

- in Wave.cc:

```
void BadWaveMoL_Register(CCTK_ARGUMENTS)
{
    DECLARE_CCTK_ARGUMENTS;
    MoLRegisterEvolved(CCTK_VarIndex("BadWaveMoL::psi"),
                       CCTK_VarIndex("BadWaveMoL::kpsi"));
    ...
}
```

- in schedule.ccl:

```
storage: wave_vars[3], kernels[3]
schedule BadWaveMoL_Register in MoL_Register
{
    LANG: C
} "Register for MoL"
```

Registering evolved variables with MoL

- in `interface.ccl`:

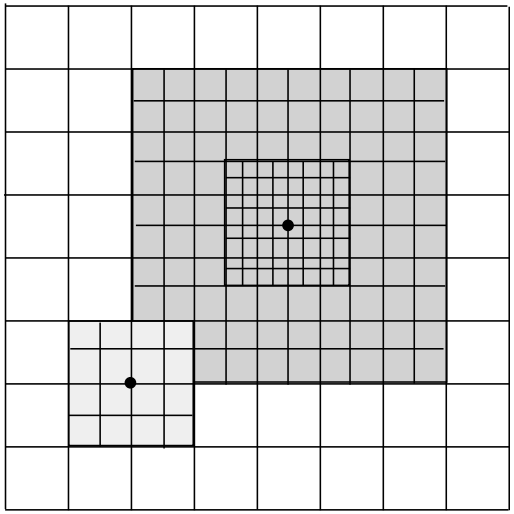
```
CCTK_INT FUNCTION MoLRegisterEvolved \
  (CCTK_INT IN EvolvedIndex, CCTK_INT IN RHSIndex)
USES FUNCTION MoLRegisterEvolved
```

- in the parfile (`par/BadWaveMoL.par`):

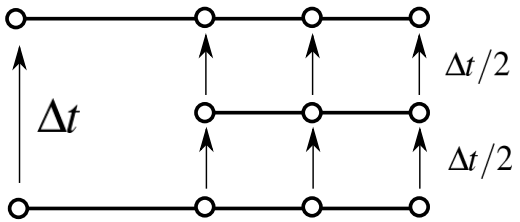
```
ActiveThorns = "BadWaveMoL carpet
CarpetIOBasic CarpetIOASCII CarpetIOScalar CarpetLib Lo
IOUtil SymBase Time CarpetReduce CartGrid3D MoL"

MoL::ODE_Method = "RK4"
MoL::MoL_Intermediate_Steps = 4
MoL::MoL_Num_Scratch_Levels = 1
```

Berger-Oliger mesh refinement

- "Box-in-box" refinement:



- Time update:



STARTUP
PARAMCHECK

↑ BASEGRID
INITIAL, POSTINITIAL

↓ RESTRICT

↑ ANALYSIS

Main loop

- ↑ REGRID
- ↑ Advance time
  EVOL (incl. MoL)
- ↓ RESTRICT
- ↑ CHECKPOINT
  ANALYSIS

Fixed mesh refinement

Parfile: par/BadWaveFMR.par

```
ActiveThorns = "CarpetRegrid2 Dissipation
SpaceMask SphericalSurface"

CarpetRegrid2::num_levels_1 = 2
Carpet::max_refinement_levels = 2
CarpetRegrid2::num_centres = 1
CarpetRegrid2::Position_x_1 = 0.5
CarpetRegrid2::Position_y_1 = 0.5
CarpetRegrid2::Position_z_1 = 0.5
CarpetRegrid2::radius_1[1] = 0.1
#CarpetRegrid2::radius_1[2] = 0.1
CarpetRegrid2::verbose = "yes"
Carpet::init_fill_timelevels="yes"
Dissipation::vars = "BadWaveMoL::psi"
```

Adaptive mesh refinement

Modifications to the code:

```
void BadWaveAMR_BoxMover(CCTK_ARGUMENTS)
{
    DECLARE_CCTK_ARGUMENTS;
    int max_refinement_levels = 30;

    radius[max_refinement_levels*0+1]=.05;

    // Make the box wiggle
    position_x[0] = position_y[0] = position_z[0] = 0.7+0.05*sin(cctk_t
    active[0]=1;
    num_levels[0]=2; // two levels: 0=base grid, 1=first refined grid

    // Turn on the next box
    active[1] = 1;
    radius[max_refinement_levels*1+1]=.05;
    position_x[1]=position_y[1]=position_z[1]=.3;
    num_levels[1]=2;
}
```

Adaptive mesh refinement

- Modifications to the `schedule.ccl`:

```
schedule BadWaveAMR_BoxMover at preregrid
{
    LANG: C
} "Jiggle the box"
```

- Modifications to the `interface.ccl`:

```
inherits: grid, CarpetRegrid2
```

Adaptive mesh refinement

Modifications in the parfile (`pars/BadWaveAMR.par`):

```
CarpetRegrid2::regrid_every = 2
Carpet::max_refinement_levels = 3
CarpetRegrid2::num_centres = 2

CarpetRegrid2::num_levels_1 = 3
CarpetRegrid2::Position_x_1 = 0.3
CarpetRegrid2::Position_y_1 = 0.3
CarpetRegrid2::Position_z_1 = 0.3
CarpetRegrid2::radius_1[1] = 0.2
CarpetRegrid2::radius_1[2] = 0.1

CarpetRegrid2::num_levels_2 = 2
CarpetRegrid2::Position_x_2 = 0.7
CarpetRegrid2::Position_y_2 = 0.7
CarpetRegrid2::Position_z_2 = 0.7
CarpetRegrid2::radius_2[1] = 0.05
```

Adaptive mesh refinement

Modifications in the parfile (pars/BadWaveAMR.par):

```
ActiveThorns = "IOJPeg CarpetSlab CarpetInterp
         CarpetInterp2 AEILocalInterp"
IOJpeg::out_every                = 1
IOJpeg::out_vars                 = "BadWaveAMR::psi"

IOJPeg::gridpoints = interpolate
IOJpeg::array2d_x0           = 0.0
IOJpeg::array2d_y0           = 0.0
IOJpeg::array2d_z0           =  0.5
IOJpeg::array2d_npoints_i    = 101
IOJpeg::array2d_dx_i         =  0.01
IOJpeg::array2d_dy_i         =  0
IOJpeg::array2d_dz_i         =  0
IOJpeg::array2d_npoints_j    = 101
IOJpeg::array2d_dx_j         =  0
IOJpeg::array2d_dy_j         =  0.01
IOJpeg::array2d_dz_j         =  0
```

Frontiers in Cactus Development

Three Goals:

- Extend/Improve Cactus to
  - new methods required for "old" community
  - new methods required for new users
- Make Cactus easier to use
  - for experts
  - for new users
- Make use of Cactus as educational tool
  - Use of Cactus in courses
  - Provide tutorials
  - Cactus as CS research object

Cactus Improvements

Examples of improvements of existing capabilities

- Improve AMR scaling
  - Better/dynamic load-balancing
  - Improving AMR regrid-scaling
- Improve Cactus scheduling
  - Make flesh data-dependency aware
  - Automatic dependency-resolution
- Improve and extend parallel IO capabilities
  - Better support parallel HDF5
  - Include support for F5, other formats, e.g., SDF, Silo, NetCDF

Making Cactus easier to use

For Experts

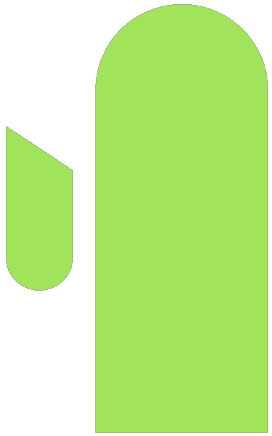
- Improve Utilities
  - GetComponents: improve updating support
  - Simfactory: testsuite support
- Support thorn versioning and dependencies

For new Users

- Provide Eclipse environment
- Provide suite of simple examples
- Support major desktop platforms
- Improve library handling
- Improve documentation

Cactus in education

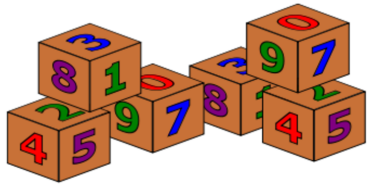
- Use of Cactus in courses
  - "Scientific Computing" at LSU
  - Part of summer schools, e.g., on numerical relativity in South Korea
- Provide tutorials
  - TeraGrid conference
  - LONI - Louisiana-wide HPC tutorial every 6 months
- Cactus as CS research object
  - active CS research projects
  - graduate projects
  - undergraduate training



Cactus: Advanced Topics

Kranc

Kranc

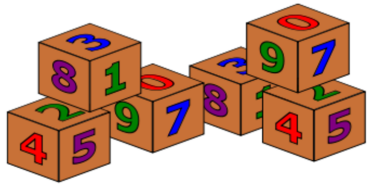


Kranc Assembles Numerical Code

- A tool for generating thorns
- Original version by S. Husa, I. Hinder, C. Lechner
- Used for many scientific papers
- Allows user to specify continuum problem without regard for implementation

Kranc

Kranc

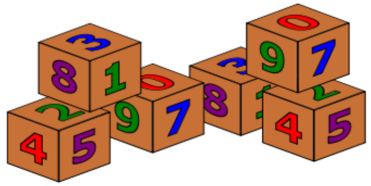


Kranc Assembles Numerical Code

- A tool for generating thorns
- Original version by S. Husa, I. Hinder, C. Lechner
- Used for many scientific papers
- Allows user to specify continuum problem without regard for implementation

Kranc

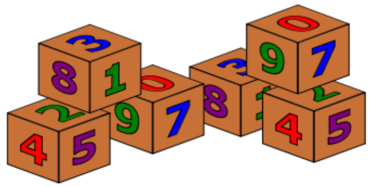
Kranc



Kranc Assembles Numerical Code

- A tool for generating thorns
- Original version by S. Husa, I. Hinder, C. Lechner
- Used for many scientific papers
- Allows user to specify continuum problem without regard for implementation

Kranc

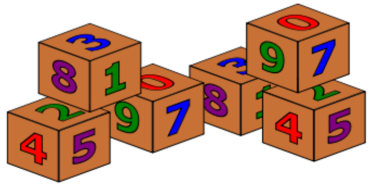
Kranc 

Kranc Assembles Numerical Code

- Minimally, a Kranc script contains
  - Cactus Group Definitions
  - A set of Calculations
- A calculation in Kranc is written using Mathematica (a symbolic programming language)
  - Example: dot[h[la,lb]] $->$ -2 alpha K[la,lb]
  - dot means "time derivative"
  - h[la,lb] and K[la,lb] are matrices

Kranc

Kranc

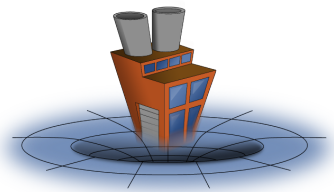


Kranc Assembles Numerical Code

- Included in Cactus-OS distribution: `Cactus/repos/Kranc`.
- The state-of-the art numerical relativity code, produced by Kranc: `Cactus/arrangements/McLachlan`.
- For more, see: `http://kranccode.org`

Simfactory

The Simulation Factory:



- Command-line tools for setting up Cactus distribution and managing simulations on a variety of supercomputers, including most of the TeraGrid machines.
- Captures the best practices of experienced users, ensuring repeatable and well-documented scientific results.
- Developed by Erik Schnetter and Michael Thomas.
- Included in Cactus-OS distro: `Cactus/simfactory`.
- For more, see: `http://simfactory.org`

Einstein Toolkit

Open-source code for the numerial relativity and relativistic astrophysics.

Capabilities:

- Accurate evolutions of vacuum spacetimes using Einstein equations.
- Relativistic hydrodynamics, based on the public version of Whisky code.

- Initial data solvers: single and bindary BHs and relativistic stars.
- Analysis: finds BH horizon, calculates spacetime curvature, extracts gravitational waves.
- More information: `http://einsteintoolkit.org`

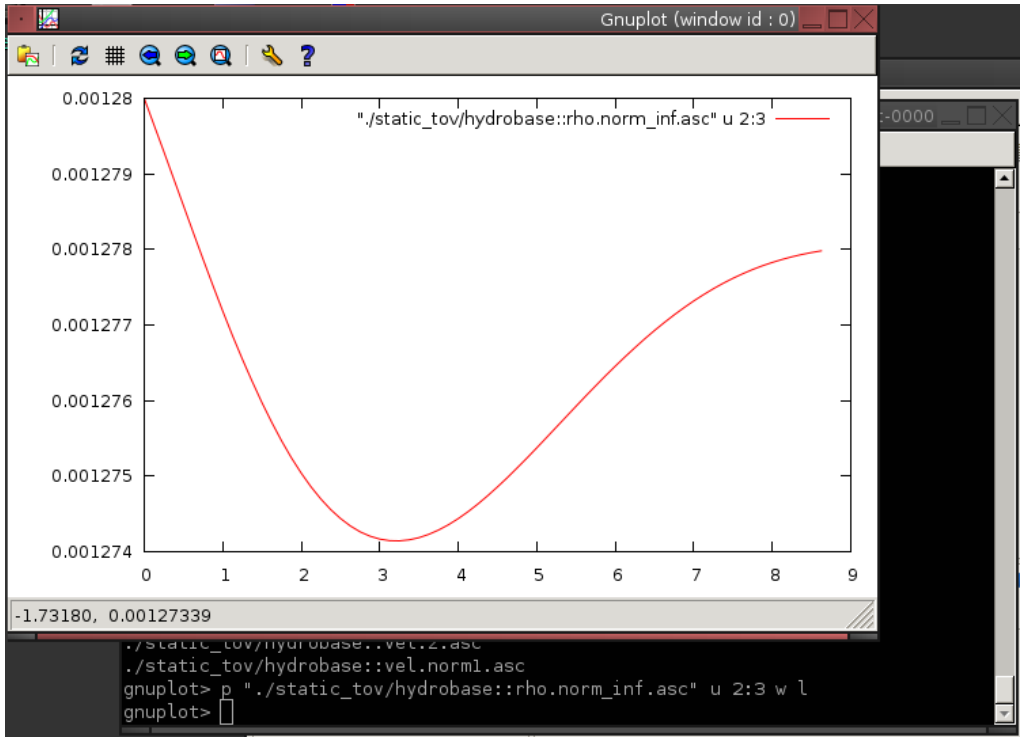
Einstein Toolkit on your virtual box

In Eclipse, open "Mojave > Edit variables..." and set parameters for precompiled version of the Einstein toolkit and a parameter file:

Name	**Value**
par	`par/static_tov.par`
thornlist	`thornlists/einsteintoolkit.th`
config	`et`
sim.name	`tov_star1`

Select "Mojave > Create", then "Mojave > Run" to create and run an evolution of a relativistic TOV star!

Simple visualization with gnuplot




```
cd ~/simulations/tov_star/output-0000
gnuplot> p "static_tov/hydrobase::rho.maximum.asc" u 2:3 w l
gnuplot> p "static_tov/admbase::alpha.x.asc" u 10:13 i 0 w l
```

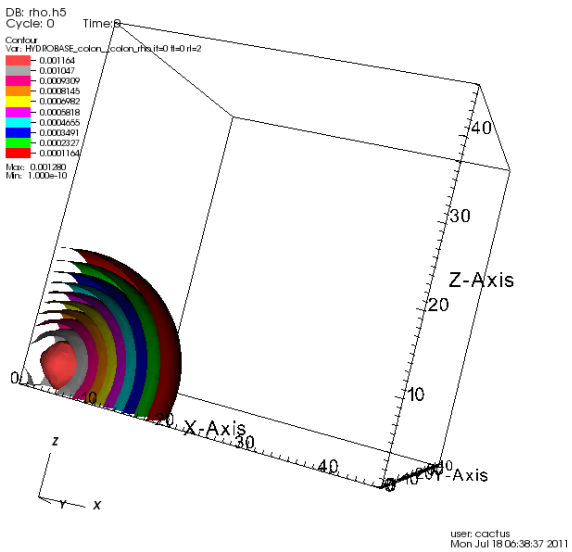
3D output and visualization

To enable 3D output, copy `static_tov.par` to a new parameter file (say, `static_tov_h5.par`) and append the following lines, then use it to create and run a new simulation:

```
IOHDF5::out_criterion = time
IOHDF5::out_dt        = 100.0
IOHDF5::out_vars = "
  grid::coordinates{out_every=10000000}
  HydroBase::rho
"
```

Using VisIt

VisIt is an interactive 3D visualization tool.



- Open VisIt
- Open "File > Open File...", navigate to the output of your simulation, or use:

  `~/simulations/tov_star/output-0000/`
- Select filter: `*.h5`
- Select the star density: `rho.h5`

- Add contour plot for the second refinement level (`HYDROBASE::rho it=0 tl=0 rl=2`)
- Press "Draw" button.

VisIt website: `https://wci.llnl.gov/codes/visit/`

Thorn HTTPD

Remote steering/monitoring tool

- Mini web server implemented as a thorn
- Connect to a running simulation with a web browser
- View and steer simulation parameters
- Basic remote visualization

Adding HTTPD to your configuration

Using HTTPD is simple:

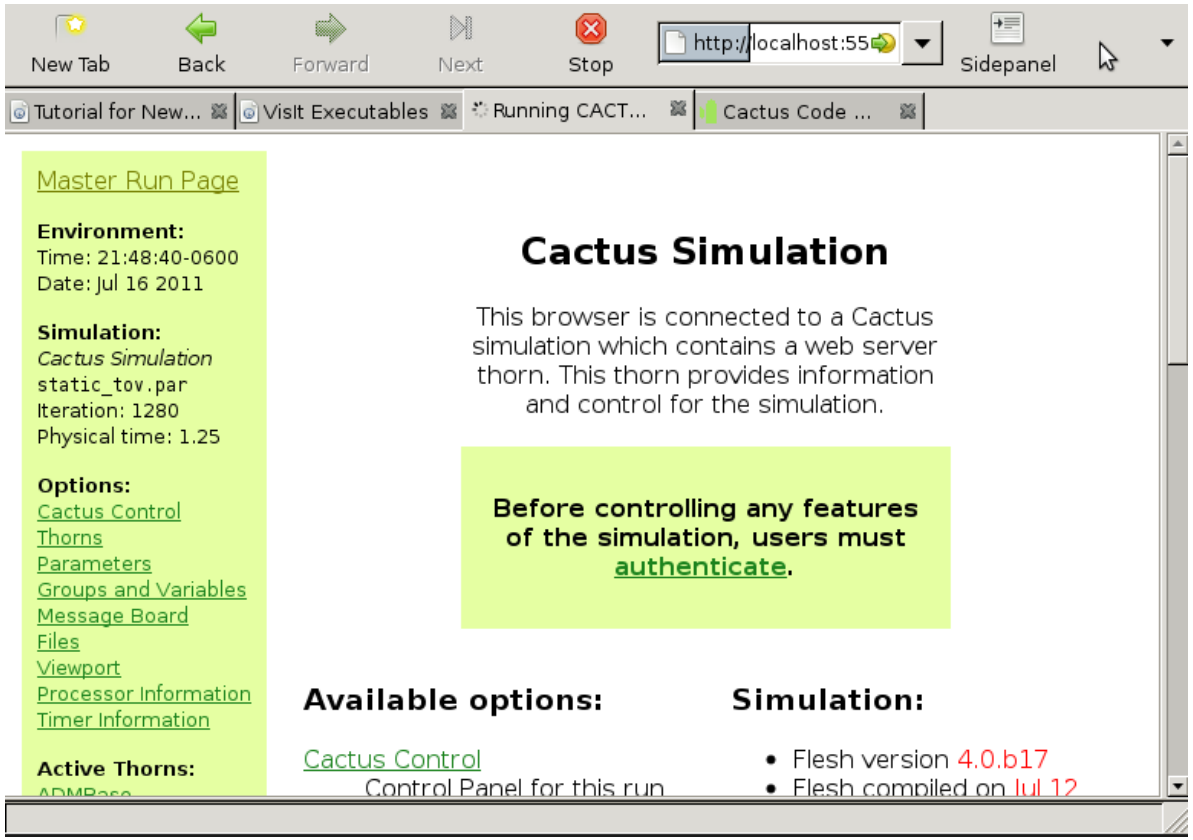
- Add the following thorns to your thornlist (file `~/Cactus/thornlists/BadWaves.th`):

```
CactusConnect/HTTPD
CactusConnect/HTTPDExtra
CactusConnect/Socket
```

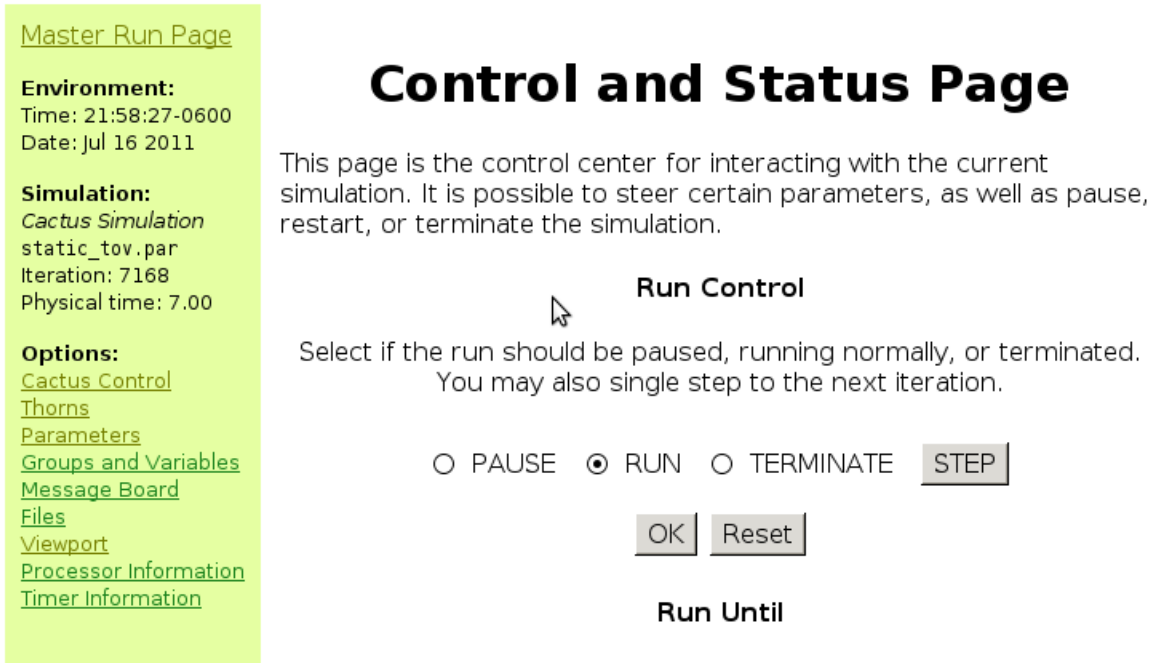
- Enable these thorns in your parfile (for example, in `~/Cactus/pars/BadWaveAMR.par`):

```
ActiveThorns = "HTTPD HTTPDExtra Socket"
```

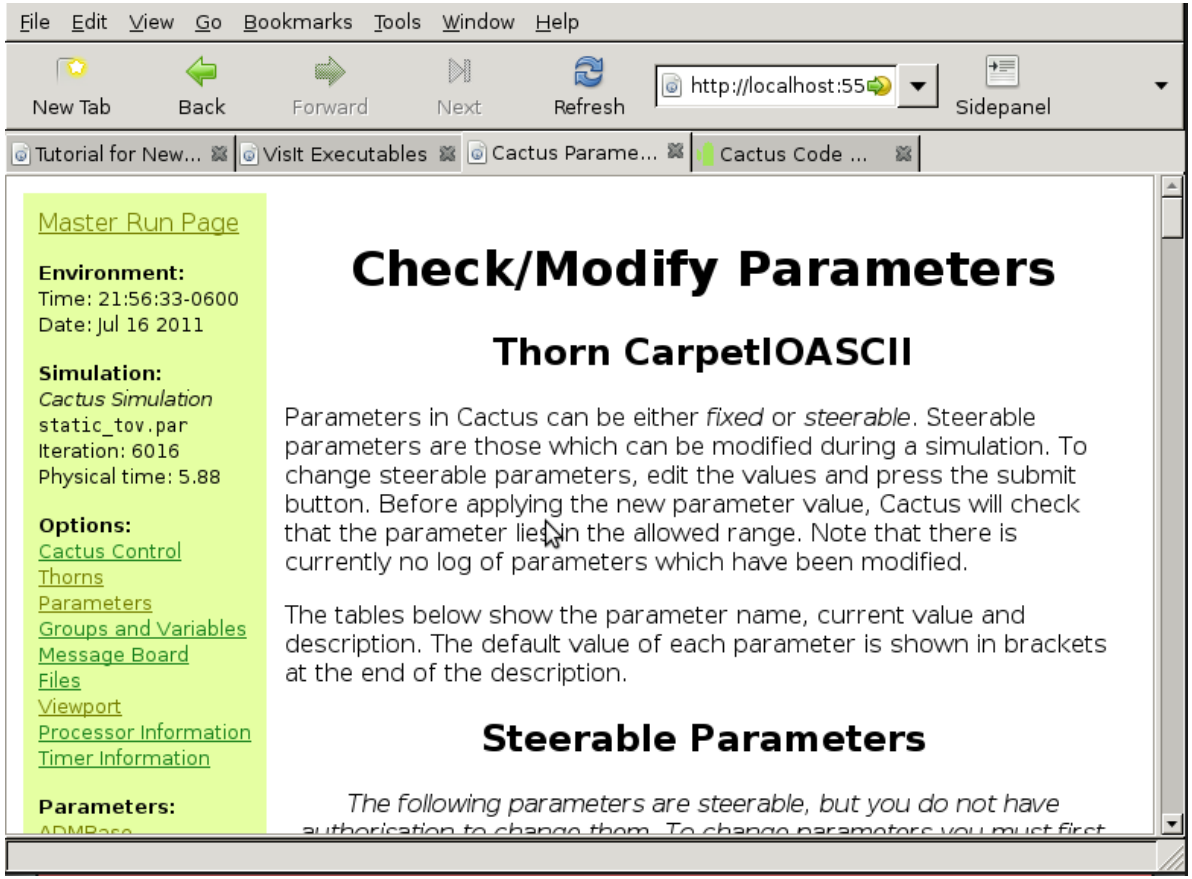
Sample HTTPD session



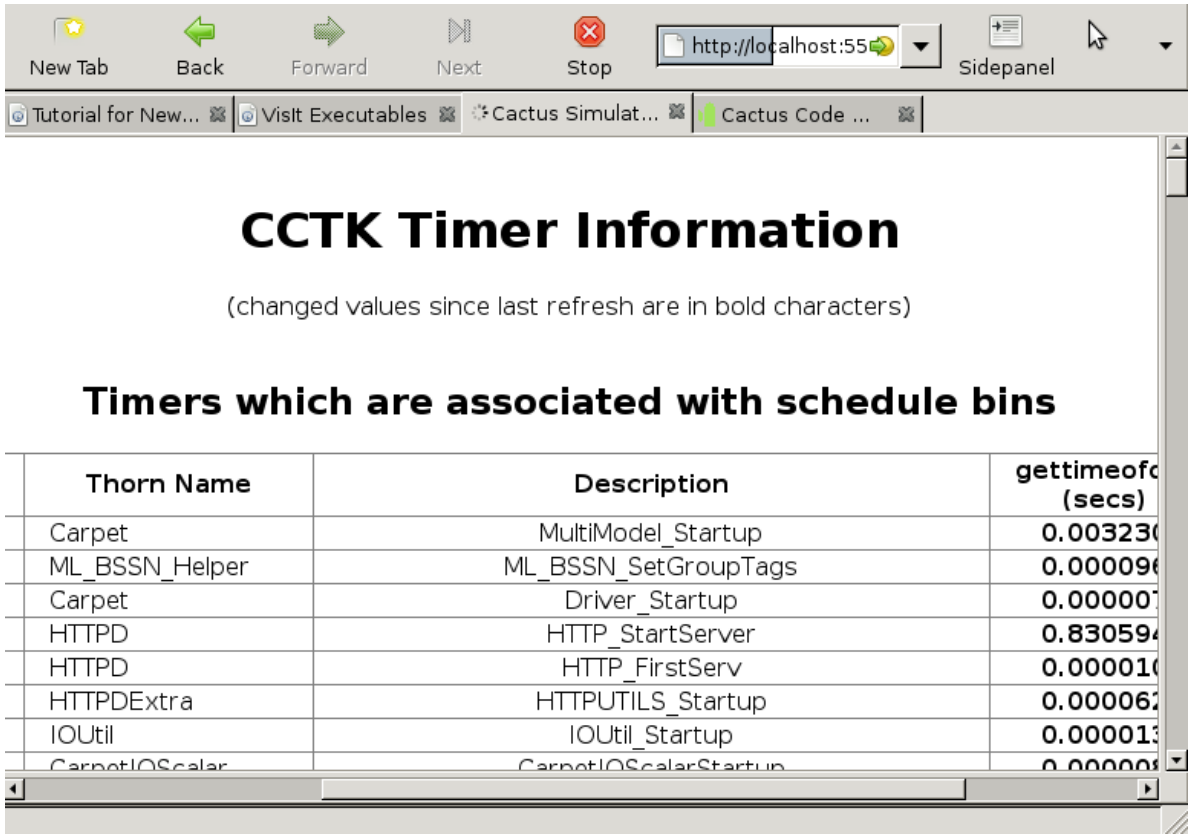
Sample HTTPD session



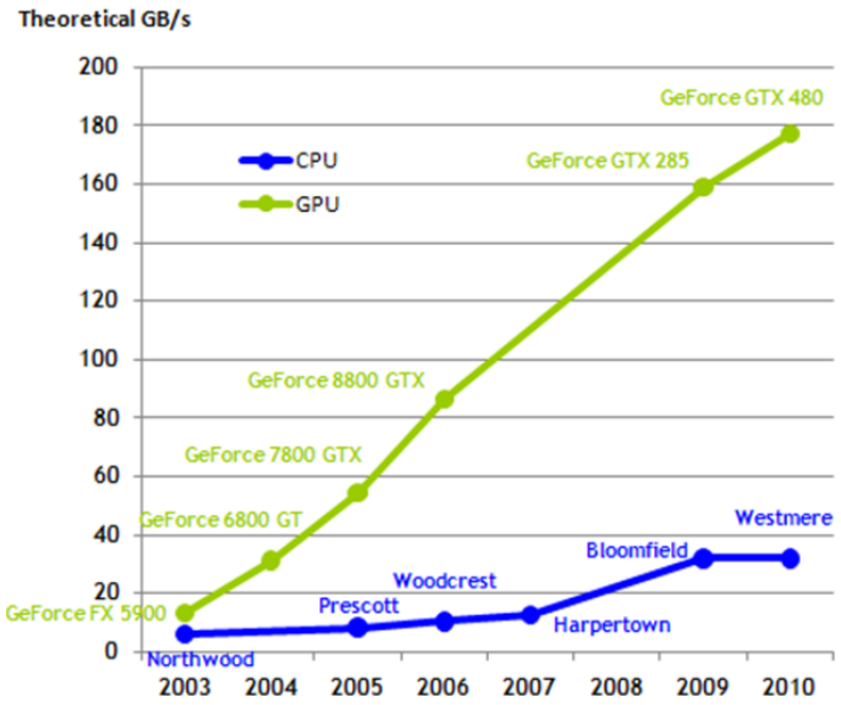
Sample HTTPD session



Sample HTTPD session

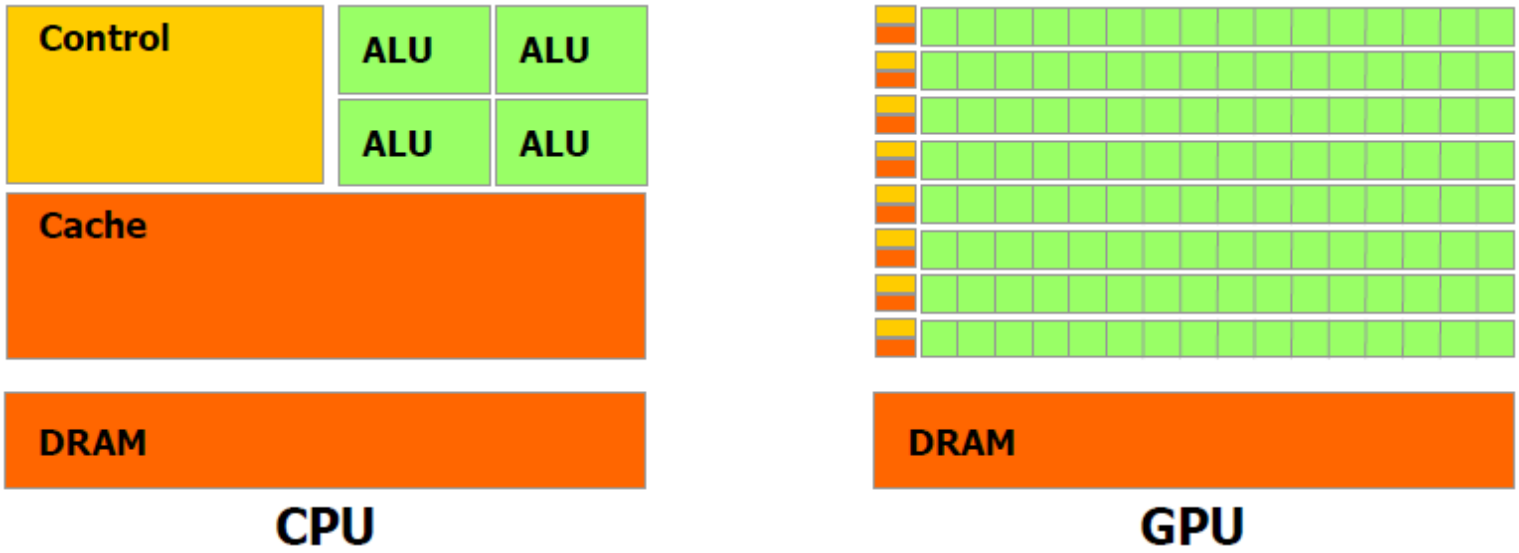


Motivations



- General Purpose Graphics Processing Units (GPGPUs) are high-performance graphical processors that can be used to accelerate a wide range of applications. (CUDA C Programming Guide Version 3.2)

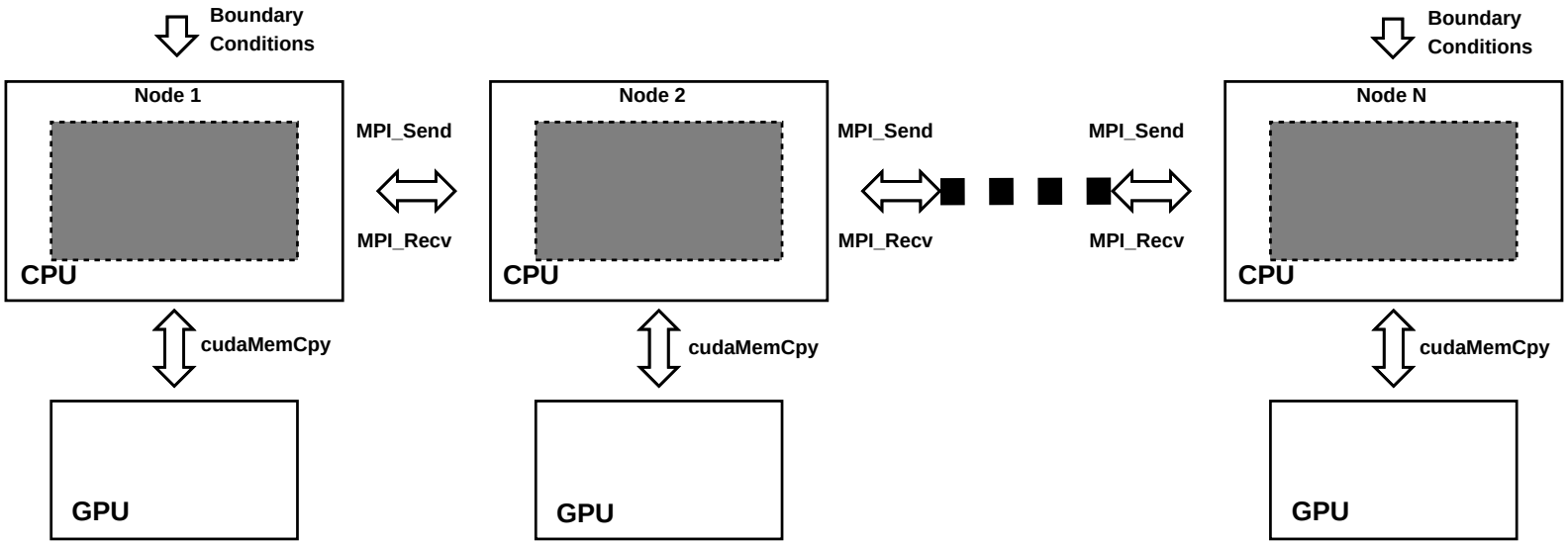
GPU vs. CPU



- Compared to CPU, GPU devotes more transistors to data processing rather than data caching and flow control (CUDA C Programming Guide Version 3.2)

Cactus CUDA Toolkit(under development)

- MPI + CUDA programming in Cactus is made simple.



CaCUDAViz

- The volume rendering of a scalar wave spreading from the center. using CUDA on a GPU, while the rendering parameters are steerable at runtime via the Cactus web server.