Cactus Tutorial

Thorn Writing 101

Yaakoub Y El Khamra

Frameworks Group, CCT

17 Feb, 2006

Agenda

- Downloading Cactus
  - The “GetCactus” Script
  - CVS
- Thorn writing
  - Making a new thorn
  - Editing the .ccl files
  - Adding source code
- Running your code
  - Building a configuration
  - Viewing results

Downloading Cactus

- 2 convenient methods to download Cactus
- Always work with the development version
- Do NOT mix stable and development thorns and flesh
- Make sure you download all the thorns you need. If you need more thorns you do not have, download them
- Keep an eye out for warnings, use the 'ls' command to list the contents of your cactus directory. Make sure everything is there

GetCactus Script

- Obtained from: http://www.cactuscode.org/toolkit/getcactus#getcactus
- You will need a ThornList, a list of thorns for the script to download.
- If you do not supply a ThornList the script will download the flesh only.
- ThornLists can be obtained from: http://www.cactuscode.org/toolkit/thornlists

GetCactus Script

- Check that the script has the right permissions:
  [yye00@fmws01 Tutorial2]$ ls -alF GetCactus
  -rw-rw-r-- 1 yye00 yye00 44964 Feb 16 12:07 GetCactus
- Change the permissions to include execution:
  [yye00@fmws01 Tutorial2]$ chmod +x GetCactus
  [yye00@fmws01 Tutorial2]$ ls -alF GetCactus
  -rwxrwxr-x 1 yye00 yye00 44964 Feb 16 12:07 GetCactus*
- Notice the 'x' in the file permissions, this means we can now run the script
- Run the script: ./GetCactus Tutorial2.th

CVS

- Please attend the CVS tutorials given at CCT
- For those who know how to use CVS, perform the following:
  cvs -d :pserver:cvs_anon@cvs.cactuscode.org:/cactusdevcvs login
  the password is 'anon'.
- Once you are logged in:
  cvs -d :pserver:cvs_anon@cvs.cactuscode.org:/cactusdevcvs co Cactus
  to checkout Cactus
- cd arrangements
  to enter the arrangements directory. Once inside arrangements, checkout the arrangements you need using:
  cvs -d :pserver:cvs_anon@cvs.cactuscode.org:/cactusdevcvs co <arrangement_name>

Let's download Cactus

- Journey to the source of all evil: www.cct.lsu.edu/~yye00
- Download the GetCactus script and the MyThorns.th files.
- Run ./GetCactus MyThorns.th
- You now have a cactus checkout.

Thorn Writing

- First things first, cd into the Cactus directory:
  cd Cactus
- Always when starting up, do a make help to check for options
- make newthorn
- If you are not inspired, specify RungeKutta as a thorn name andTutorial2 as an arrangement.

Looking at your thorn...

- Your thorn has been created in:
  arrangements/Tutorial2/RungeKutta
- There you will the following files:
  - README
  - interface.ccl
  - param.ccl
  - schedule.ccl
- and the following directories:
  src/, doc/, par/, test/
- doc/ has documentation.latex (Attend the latex tutorials at CCT)
- src/ has make.code.defn

VIM TIME

- Keep your VIM cheat sheet handy
- cd arrangements/Tutorial2/RungeKutta
- vim interface.ccl
- Hit 'i' to enter insert mode.
- Next slide: this is what you have to have in your interface.ccl

interface.ccl

- Input the following:

```
implements: RK      #what the thorn implements
inherits:           #what implementation the thorn inherits
public:
CCTK_REAL group1 type=Array DIM=1 size=group_size
{
  x, y
} "variables I will use"
```

- There is more on cactus variables later...

param.ccl

- This param.ccl file contains examples of the 3 basic parameter types:

```
# Parameter definitions for thorn RungeKutta
# $Header:$
INT group_size "size of our arrays in 1D"
{
  10:1000 ::"this is a dummy range just to demonstrate ranges"
} 1000
REAL x_spacing "size of the spacing in the x-direction"
{
  0:* ::"this means it can be anything greater than zero"
} 0.01
REAL x_zero "Part of the initial condition"
{
  *:* ::"this means the range could be anything"
} 0.0
```

-

```
REAL y_zero "Rest of the initial condition"
{
  *:* ::"this means the range could be anything"
} 1.0

KEYWORD method_order "which method to use"
{
  "second_order"  :: "use the second order runge kutta method"
  "fourth_order" :: "use the fourth order runge kutta method"
} "fourth_order"
```

schedule.ccl

- You ALWAYS declare STORAGE for your VARIABLES, and schedule your SUBROUTINE

```
# Schedule definitions for thorn RungeKutta
# $Header:$
STORAGE: group1
schedule RungeKutta at EVOL
{
  LANG:F77
} "Runge Kutta ODE Solver"
```

Checklist

- Interface.ccl: what you implement, what you inherit, the variables that you use
- param.ccl: the external parameters you control your thorn through
- schedule.ccl: the storage you need for your variables (and when) and when you want to call your subroutines/functions
- README: exactly that, documentation. Make sure you have something in here.

And now for something completely different

- Code: add all the source files to your make.code.defn file. In our case, 1 file only (for now). RungeKutta.F77
- make.code.defn should have:

```
# Main make.code.defn file for thorn RungeKutta
# $Header:$

# Source files in this directory
SRCS = RungeKutta.F77

# Subdirectories containing source files
SUBDIRS =
```

In your source

- You cannot go wrong with documentation for your file.
- Always have the following to the beginning of your file:

```
/*@@
  @file      RungeKutta.F77
  @date
  @author    Yaakoub Y El Khamra
  @desc
             Solve an ODE using
RungeKutta 1st or 4th order
  @enddesc
 @@*/
```

- Also add the following:

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

Also in your source code....

- For our subroutine, let's add the following:

```
subroutine RungeKutta(CCTK_ARGUMENTS)
implicit none
DECLARE_CCTK_ARGUMENTS
DECLARE_CCTK_PARAMETERS
.... code.... code... more code...
end subroutine RungeKutta
```

More in your source code

- Let's add some local variables:

```
CCTK_INT i, CCTK_Equals
CCTK_REAL k1,k2,k3,k4
CCTK_REAL x1,y1,x2,y2,x3,y3
CCTK_REAL my_function
```

- And let's setup the initial condition:

```
c     Setup the initial condition
      x(1) = x_zero
      y(1) = y_zero
```

Control Statements using parameters

- Remember our parameters? Let's use them:

```
c       Check the order of RK to use
        if (CCTK_Equals(method_order, "second_order").eq.1) then
          call CCTK_INFO("Using second order Runge Kutta")

c       iterate over the x and y arrays

        elseif (CCTK_Equals(method_order, "fourth_order").eq.1) then
          call CCTK_INFO("Using fourth order Runge Kutta")

c       iterate over the x and y arrays

        endif
```

We also need an ODE function:

- Add this way at the bottom of your source file:

```
c     This is the ODE we want to solve: y'=3e(-4x)-2y
      function my_function(x,y)

      implicit none

      CCTK_REAL x, y, my_function
      my_function = 3.0*EXP(-4.0*x)-2.0*y
      return
      end function my_function
```

Let's write the code

- Check your cheat sheets, you have 2 algorithms that you need to implement.
- Remember your F77 fortran tutorials and references, this code is simple, 2 do-end do loops.
- Ask for help if you are stuck anywhere. We can and will help.
- Once you are done, we need to compile the code, i.e. Create a configuration

Configuration:

- Let's make a new configuration, called rungekutta
- Pass in the option DEBUG=yes to enable debugging, this is very handy.
- Attend the debugging tutorials given at CCT
- In short, the command you need to run is the following:

  make rungekutta DEBUG=yes

Building your configuration

- Now that you have configured your configuration, let's build it
- We will use the thorn in the next slide
- Keep an eye out for warnings and errors. This is a good time to learn about ccl syntax
- If you get errors or warnings, please ask for help.

ThornList:

- Tutorial2/RungeKutta
-
- CactusBase/LocalReduce
- CactusBase/IOBasic
- CactusBase/IOASCII
- CactusBase/IOUtil
- CactusBase/CoordBase
-
- CactusPUGH/PUGH
- CactusPUGH/PUGHReduce
- CactusPUGH/PUGHSlab

Congratulations you are done

- Write the following to an RK.par file:

```
ActiveThorns=" IOASCII PUGHSLAB PUGH PUGHREDUCE
LOCALREDUCE IOBASIC IOUTIL COORDBASE RUNGEKUTTA"
cactus::cctk_itlast=1
ioascii::out1D_every=1
ioascii::out1D_vars="RK::x  RK::y "
ioascii::out1D_dir="RungeKutta_out"
ioascii::out1D_style="xgraph"
RK::method_order="fourth_order"
RK::x_spacing = 0.001
```

Run Forest Run...

- To run: ./exe/cactus_RungeKutta RK.par
- To look at output:
  vim RungeKutta_out/y_1D.xg
- To plot the output, please attend the gnuplot tutorial at CCT. It is yet to be announced, if you have preferred times please let us know.