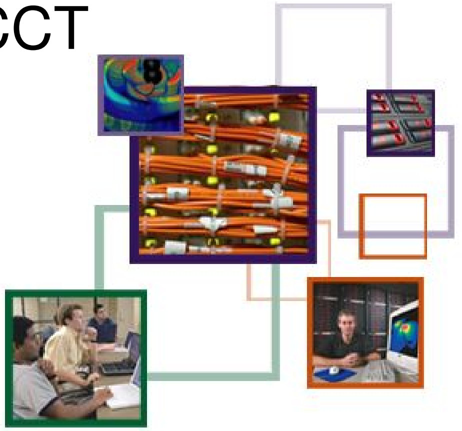
Cactus Tutorial

Building and Running Cactus

Yaakoub El Khamra

Cactus Developer, Frameworks Group CCT

23 May, 2006



Center for Computation & Technology

AT LOUISIANA STATE UNIVERSITY

Introduction

- Now that you have successfully obtained a fresh cactus checkout, it is time to actually compile and run cactus
- If you check the cactus checkout directory, this is what you see:
  - arrangements/: directory where the arrangements you checked out reside
  - doc/: directory where the documentation resides
  - lib/: perl scripts and make files that cactus needs to function
  - src/: the cactus flesh resides in this directory
  - Makefile: this is the master make file for cactus, and this is your primary point of interaction with your cactus checkout

Cactus Configuration

- A Cactus configuration is a collection of thorns and their compilation/build options
- Cactus can be built in different configurations from the same copy of the source files, and these different configurations coexist in the `Cactus/configs` directory
- Reasons for having multiple configurations:
  - You can have different configurations for different thorn collections compiled into your executable.
  - Different configurations can be for different architectures. You can keep executables for multiple architectures based on a single copy of source code, shared on a common file system.
  - You can compare different compiler options, debug-modes. You might want to compile different communication protocols (e.g. MPI or Globus) or leave them out all together.

Creating a new configuration

- The simplest way to build a Cactus configuration is to use the cactus make system
- A `make <configuration_name>` will prompt the user if they want to create a configuration with the name: configuration_name
- This generates a configuration with the name `configuration_name`, doing its best to automatically determine the default compilers and compilation flags suitable for the current architecture.
- You can supply a specific list of thorns (henceforth referred to as a thornlist) for your configuration
- You can also supply a specific list of build options you want Cactus to use for building your configuration
- You can supply a list of external packages you want Cactus to make use of during the build process

ThornLists

- Thornlists are exactly that, a list of thorns (and the arrangements they belong to) that you want compiled into your configuration.
- If you do not specify a thornlist, Cactus will attempt to compile all the thorns you currently have in your Cactus checkout. This might not always be a good thing as you probably will not need all the thorns. For this reason, Cactus will prompt the user if they want to make changes to the default (All Thorns) thornlist of the configuration if no thornlist is specified
- To specify a thornlist, you can use the `THORNLIST=<ThornList_File>` option while building the configuration where `ThornList_File` is the name of file containing a list of thorns with the syntax <arrangement name>/<thorn name>, lines beginning with # or ! are ignored.
- You can also use the `THORNLIST_DIR` option which specifies the location of directory containing `THORNLIST`. This defaults to the current working directory.

ThornList Example

- An example of a thornlist file could be:

```
# arrangement/thorn                   # implements (inherits)
   [friend] {shares}
#
CactusBase/Boundary                   # boundary ( ) [ ] { }
CactusBase/CartGrid3D                 # grid (coordbase) [ ]
   {driver}
CactusBase/CoordBase                  # CoordBase ( ) [ ] { }
CactusBase/IOASCII                    # IOASCII ( ) [ ] {IO}
CactusCFD/CFDBase                     # cfdbase (cfdmesh) [ ] {
   }
CactusCFD/CFDEvolvers                 # cfdevolver
   (cfdbase,cfdmesh) [ ] { }
```

Build Options

- There are several methods to specify build options for Cactus configurations
  - Create a file `~/.cactus/config`: All available configuration options may be set in the file `~/.cactus/config`, any which are not set will take a default value. The file should contain lines of the form: `<option> [=] ...,` The equals sign is optional.
  - Add the options to a configuration file and use, `gmake <config>-config options=<filename>,` the options file has the same format as ~ /.cactus/config.
  - Pass the options individually on the command line, `gmake <configuration name>-config <option name>=<chosen value>, ...`

Compilers and Tools

- CC: The C compiler.
- CXX: The C++ compiler.
- F90: The Fortran 90 compiler.
- F77: The Fortran 77 compiler.
- CPP: The preprocessor used to generate dependencies for and to preprocess C and C++ code.
- FPP: The preprocessor used to generate dependencies for and to preprocess Fortran code.
- LD: The linker.

Flags for Compilers and Tools

- CFLAGS: Flags for the C compiler.
- CXXFLAGS: Flags for the C++ compiler.
- F90FLAGS: Flags for the Fortran 90 compiler.
- F77FLAGS: Flags for the Fortran 77 compiler.
- CPPFLAGS: Flags for the preprocessor (used to generate compilation dependencies for and preprocess C and C++ code).
- FPPFLAGS: Flags for the preprocessor (used to generate compilation dependencies for and preprocess Fortran code).
- LDFLAGS: Flags for the linker. This variable is ignored while the compilers and linkers are autodetected.

Useful Flags

- DEBUG: Specifies what type of debug mode should be used, the default is no debugging. Current options are yes, no, or memory. The option yes switches on all debugging features, whereas memory just employs memory tracing (B10.3).
- OPTIMISE, OPTIMIZE: Specifies what type of optimisation should be used. The only options currently available are yes and no. The default is to use optimisation.
- C_OPTIMISE_FLAGS: Optimisation flags for the C compiler, their use depends on the type of optimisation being used.
- CXX_OPTIMISE_FLAGS: Optimisation flags for the C++ compiler, their use depends on the type of optimisation being used.
- F90_OPTIMISE_FLAGS: Optimisation flags for the Fortran 90 compiler, their use depends on the type of optimisation being used.
- F77_OPTIMISE_FLAGS: Optimisation flags for the Fortran 77 compiler, their use depends on the type of optimisation being used.

More Useful Flags

- C_WARN_FLAGS:Warning flags for the C compiler, their use depends on the type of warnings used during compilation
- CXX_WARN_FLAGS: Warning flags for the C++ compiler, their use depends on the type of warnings used during compilation
- F90_WARN_FLAGS: Warning flags for the Fortran 90 compiler, their use depends on the type of warnings used during compilation
- F77_WARN_FLAGS: Warning flags for the Fortran 77 compiler, their use depends on the type of warnings used during compilation
- C_LINE_DIRECTIVES: Whether error messages and debug information in the compiled C and C++ files should point to the original source file or to an internal file created by Cactus. The only options available are yes and no, the default is no. Set this to no if your compiler reports error messages about unrecognised # directives.
- F_LINE_DIRECTIVES: Whether error messages and debug information in the compiled Fortran files should point to the original source file or to an internal file created by Cactus. The only options available are yes and no, the default is no. Set this to no if your compiler reports error messages about unrecognized # directives.

External Packages

- Cactus can be compiled with support for external packages.
- Some thorns cannot be compiled without extra packages (EllPETSc for example requires PETSc)
- Common external libraries and their build options:
  - MPI: The MPI package to use, if required. Supported values are CUSTOM, NATIVE, MPICH or LAM.
  - HDF5: Supported values are yes, no. A blank value is taken as no.
  - LAPACK: Supported values are yes, no. A blank value is taken as no.
  - PETSC: Supported values are yes, no. A blank value is taken as no.
  - PTHREADS: Supported values are yes.

MPI

- MPI stands for Message Passing Interface
- MPI provides inter-processor communication. It can either be implemented natively on a machine (this is usual on most supercomputers), or through a standard package such as MPICH, LAM, WMPI, or PACX.
- To compile with MPI, the configure option is: `MPI = <MPI_TYPE>`, where `MPI = <MPI_TYPE>` can take the values: CUSTOM, NATIVE, MPICH, LAM, WMPI, HPVM, MPIPro, PACX
- For more details please consult the Cactus Users Guide section A 2.1.3

MPICH

- You will be using MPICH in the tutorials later on
- Options for MPICH are as follows:
  - `MPICH_ARCH`: machine architecture.
  - `MPICH_DIR`: directory in which MPICH is installed. If this option is not defined it will be searched for.
  - `MPICH_DEVICE`: the device used by MPICH. If not defined, the configuration process will search for this in a few defined places. Supported devices are currently `ch_p4`, `ch_shmem`, `globus` and `myrinet(ch_gm)`.
- If `MPICH_DEVICE` is chosen to be `ch_gm`, (http://www.myri.com), an additional variable must be set: `MYRINET_DIR`: directory in which Myrinet libraries are installed.

HDF5

- You will also be using HDF5 in the tutorials later on
- The build options for HDF5 are as follows:
  - `HDF5 = yes/no`
  - `HDF5_DIR = <dir>`
- Depending on your architecture, you might need to set the path for libz or libsz. This can be done using the following options:
  - `LIBZ_DIR = <dir>`
  - `LIBSZ_DIR = <dir>`

Building a Configuration

- Once you have created a new configuration, the command `gmake <configuration name>`, will build an executable, prompting you along the way for the thorns which should be included if no thornlist were specified in the creation stage
- To revisit the thornlist Cactus is compiling you can type: `gmake <config>-thornlist`
- Or you can edit it using `gmake <config>-editthorns`
- Instead of using the editor to specify the thorns you want to have compiled, you can edit the ThornList outside the make process. It is located in `configs/<config>/ThornList`, where `<config>` refers to the name of your configuration.

Make options for building

- `gmake <target> PROMPT=no`: turns off all prompts from the make system.
- `gmake <target> SILENT=no`: print the commands that gmake is executing.
- `gmake <target> WARN=yes`: show compiler warnings during compilation.
- `gmake <target> FJOBS=<number>`: compile in parallel, across files within each thorn.
- `gmake <target> TJOBS=<number>`: compile in parallel, across thorns.
- Note that with more modern versions of `gmake`, it is sufficient to pass the normal `-j <number>` flag to `gmake` to get parallel compilation.

If all goes well....

- If all goes well you will now have a cactus executable residing in your newly created `exe/` directory
- Cactus executables always run from a parameter file (which may be a specified as a command line argument taken from standard input), which specifies which thorns to use and sets the values of any parameters which are different from the default values.
- There is no restriction on the name of the parameter file, although it is conventional to use the file extension .par
- Optional command line arguments can be used to customize runtime behavior, and to provide information about the thorns used in the executable.

To run your cactus executable

```
./cactus_<config> <parameter file> [command line
  options]
```

or if the parameter file should be taken from standard input:

```
./cactus_<config> [command line options] -
```

Remember that cactus executables are created in the

exe/ directory. For MPI enabled Cactus executables,

you can run Cactus as follows:

```
mpirun -np <number> ./cactus_<config> <parameter file>
  [command line options]
```

Command line options

-O or -describe-all-parameters: Produces a full list of all parameters from all thorns which were compiled, along with descriptions and allowed values. This can take an optional extra parameter v (i.e. -Ov to give verbose information about all parameters).

-o<param> or -describe-parameter=<param>: Produces the description and allowed values for a given parameter -- takes one argument.

-T or -list-thorns: Produces a list of all the thorns which were compiled in.

-t<arrangement or thorn> or -test-thorn-compiled=<arrangement or thorn>: Checks if a given thorn was compiled in -- takes one argument.

-h, -? or -help: Produces a help message.

-v or -version: Produces version information of the code.

Command line options

-W<level> or -warning-level=<level>: Sets the warning level of the code. All warning messages are given a level -- the lower the level the greater the severity. This parameter controls the level of messages to be seen, with all warnings of level 4#4 <level> printed to standard output (warnings with level <= <level> are silently discarded). The default is a warning level of 1, meaning that only level 0 and level 1 messages should be printed.

-E<level or -error-level=<level>: This is similar to -W, but for fatal errors: Cactus treats all warnings with level <= <level> as fatal errors, and aborts the Cactus run immediately (after printing the warning message). The default value is zero, only level 0 warnings will abort the Cactus run.

Command line options

-r[o|e|oe|eo] or -redirect=[o|e|oe|eo]: This redirects the standard output (`o') and/or standard error (`e') of each processor to a file. By default the standard outputs from processors other than processor 0 are discarded.

-i or -ignore-next: Ignore the next argument on the command line.

-parameter-level=<level>: Set the level of parameter checking to be used, either strict, normal (the default), or relaxed

Parameter file

- A parameter file (or par file) is used to control the behavior of a Cactus executable. It specifies initial values for parameters as defined in the various thorns' param.ccl files
- A parameter file is a text file whose lines are either comments or parameter statements. Comments are blank lines or lines that begin with either `#' or `!'. A parameter statement consists of one or more parameter names, followed by an `=', followed by the value(s) for this (these) parameter(s). All string parameters are case insensitive.
- The first parameter statement in any parameter file should set ActiveThorns, which is a special parameter that tells the program which thorns are to be activated. Only parameters from active thorns can be set (and only those routines scheduled by active thorns are run). By default all thorns are inactive

Parameter file syntax

- Parameters following the ActiveThorns parameter all have names whose syntax depends on the scope of the parameter
- Global parameters: Just the name of the parameter itself. Global parameters are to be avoided; there are none in the Flesh and Cactus Toolkits.
- Restricted parameters: The name of the *implementation* which defined the parameter, followed by two colons, then the name of the parameter -- e.g. `driver::global_nx`.
- Private parameters: The name of the *thorn* which defined the parameter, two colons, and the name of the parameter -- e.g. `wavetoyF77::amplitude`.

Parameter file example

```
ActiveThorns = "CoordBase localreduce SymBase NaNChecker PUGHReduce CartGrid3D PUGH
    Boundary IOBasic IOUtil IOASCII IDWaveMoL PUGHSlab WaveMoL Time MoL"

#Cactus Flesh parameters

cactus::cctk_itlast = 100

#Driver restricted parameters

driver::global_nx = 51

driver::global_ny = 51

driver::global_nz = 51

#Sample output parameters

iobasic::outScalar_every = 1

iobasic::outScalar_vars = "wavemol::phi"

ioascii::out1D_every = 2

ioascii::out1D_vars = "wavemol::scalarevolvemol_scalar wavemol::energy"

IO::out_dir = “Sample_output”
```

Configuration Administration

- As described earlier, the Cactus Makefile is useful for more than just creating and building configurations
- The makefile allows you to administer your cactus configurations
- Common configuration administration targets:
- gmake <config>-clean: removes all object and dependency files from a configuration.
- gmake <config>-cleandeps: removes all dependency files from a configuration.
- gmake <config>-cleanobjs: removes all object files from a configuration.
- gmake <config>-config: creates a new configuration or reconfigures an old one.

Configuration Administration

- gmake <config>-cvsupdate: updates the Flesh and thorns for a configuration using CVS
- gmake <config>-delete: deletes a configuration (rm -r configs/<config>).
- gmake <config>-editthorns: edits the ThornList.
- gmake <config>-examples: copies all the example parameter files relevant for this configuration to the directory examples in the Cactus home directory. If a file of the same name is already there, it will not overwrite it.
- gmake <config>-realclean: removes from a configuration all object and dependency files, as well as files generated from the CST. Only the files generated by configure and the ThornList file remain.

Configuration Administration

- gmake <config>-rebuild: rebuilds a configuration (reruns the CST).
- gmake <config>-rebuild: reconfigures an existing configuration using its previous configuration options
- gmake <config>-testsuite: runs the test programs associated with each thorn in the configuration.
- gmake <config>-thornlist: regenerates the ThornList for a configuration
- gmake <config>-ThornGuide: builds documentation for the thorns in this configuration.
- gmake <thorn>-ThornDoc: builds the documentation for the thorn.
- gmake ThornDoc: builds the documentation for all thorns.
- gmake ArrangementDoc: builds the documentation for all arrangements.
- gmake <config>-configinfo: displays the options used to build the configuration.
- gmake <config>-cvsupdate: updates the Flesh and this configuration's thorns from the CVS repositories.

Getting Help

- gmake UsersGuide: runs LaTeX to produce a copy of the Users' Guide.
- gmake ThornGuide: runs LaTeX to produce a copy of the Thorn Guide, for all the thorns in the arrangements directory.
- gmake MaintGuide: runs LaTeX to produce a copy of the Maintainers' Guide.
- gmake ReferenceManual: runs LaTeX to produce a copy of the reference manual
- All of this information can be retrieved using the ultimate make target: gmake help

Questions