Proceedings of The National Conference
On Undergraduate Research(NCUR) 200S6
The University of North Carolina at Asheville
April 6 – 8, 2006

Integration of Trilinos Into The Cactus Code Framework

Josh Abadie
Department of Computer Science
Louisiana State University
298 Coates Hall
Baton Rouge, Louisiana 70803

Faculty Advisor: Dr. Gabrielle Allen

Abstract

Many of today's pressing problems in science and engineering involve the solution of systems of partial differential equations on large scale computing systems. A number of frameworks and libraries have been designed and implemented to expedite such research requirements in these fields.

The Cactus Code Framework (Cactus) is an extensible, collaborative, parallel, computational application framework for large-scale high performance computing. Through its generalized and elegant design, Cactus' capabilities are provided by a plethora of plug-and-play modules, or 'thorns'. Although Cactus was written in ANSI C for maximum portability, thorns can be written in C, C++, Fortran 77, and Fortran 90. Currently, Cactus is being used in numerical relativity, bio-informatics, climate modeling, astrophysics, computational fluid dynamics, and grid computing among other fields.

Trilinos is a collection of self-contained interoperable 'packages', developed at Sandia National Laboratories. The primary purpose of Trilinos is to ease integration of different mathematical software libraries. Some of the packages in Trilinos implement distributed matrix and vector data types, general solvers, algebraic pre-conditioners, linear solvers and non-linear solvers. Interoperability between packages is achieved through a number of mechanisms designed to minimize inter-package dependence. Though mostly written in C++ with heavy use of objects, packages can also contain Fortran source.

While there are a great number of different libraries which Cactus supports, several of the libraries that have been assimilated into Trilinos, such as the solvers Aztec, Anasazi and Amesos, the pre-conditioner ML and many others are not accessible in Cactus. To provide a broader range of solvers to the Cactus user community, we have integrated Trilinos with Cactus. I will briefly discuss the design of Cactus and then describe the design and implementation of Trilinos integration into Cactus.

1. Introduction

1.1. motivation

Grand Challenge problems in physics require powerful solvers. A solver is an algorithm which attempts to find the vector **x** for an equation of the form **Ax=b** where **A** is an N by N matrix representing the coefficients of N linear equations with N unknowns and **b** is a

vector of length N which represents the constants of the linear equations. These solvers are very important because many complicated physical problems can be described as a set of linear equations. Such solvers help scientists model these physical systems, and by doing so, being to understand them better. The number of strategies that can be used to design solvers is infinite. Most of these strategies do not converge (Reach a solution) for all systems. Some only work under very specific circumstances. Therefore, it is good to have multiple solvers on hand to ensure that an answer can be found and that the resulting answer is a good approximation of the actual solution.

Cactus is a computational framework designed for large scale computation.[1] Some of the users of Cactus are researchers in climate modeling, bio-informatics and damage mechanics as is shown in Figure 1[2]. However, Cactus is only as good as the sum of its parts. That is to say, alone without any modules, Cactus is totally useless. It's real power lies in its ability to be extended, to reuse code, and facilitate collaboration. Everything Cactus "does" is contained in thorns (modules). Which thorns that are compiled into a build of Cactus are chosen before compile time.

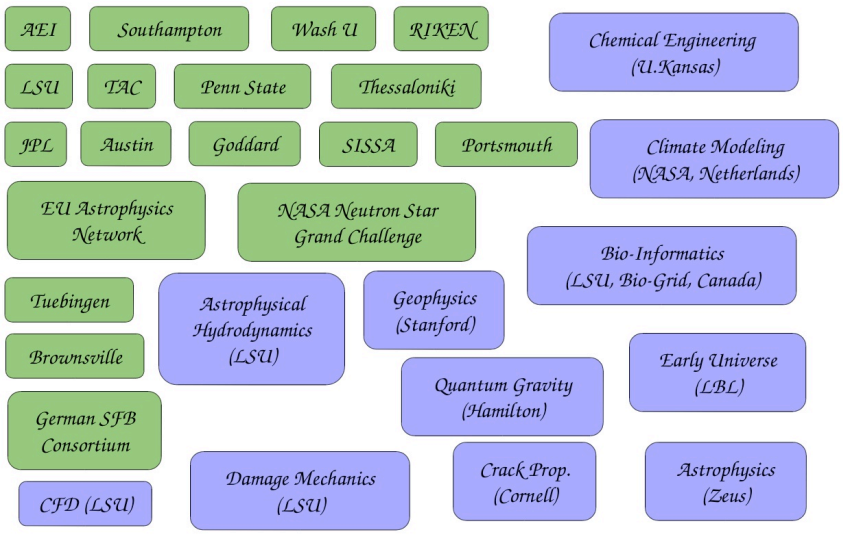

Figure 1 Selection of current users of Cactus

Cactus tries to provide an easy-to-use environment for collaborative, high-performance computing, from easy compilation on virtually any machine, to easy visualization of your output data. Since Cactus is only as good as the sum of it's parts, it is to the users and developers' advantage to add as many parts as humanly possible. The objective of this paper is to describe the efforts of the author to add a rather large and valuable part to Cactus, the Trilinos software library.

Trilinos is a set of libraries which contain classes, solvers and pre-conditioners linked together in such a way to make it easy to use and extend.[3] Some of the libraries that are supported in Trilinos include LAPACK, SuperLU, TAUCS, MUMPS, Aztec, ML, NOX, LOCA, Komplex, IFPACK, Meros, and Claps. In order to make using these libraries easier, most of the Libraries use a common set of data structures in the package Epetra. In addition, Trilinos includes as set of packages called TSF which implements a common interface for all of Trilinos' linear solvers. The ability to access and exploit these libraries would add a great deal of usefulness and value to Cactus.

1.2. a comparison of the design of cactus and trilinos

The designs of Cactus and Trilinos are very similar. These similarities as well as their differences make them very easy to combine and use. Both are designed to be easily extended, as portable as possible and contain Fortran and C/C++ code. While Trilinos is written in an object-oriented design, Cactus is based off of a more Structured (Structured is being used as a noun, referring to the programming style) way, with a core written in C. While at first this may reflect badly on Cactus, this design decision maximizes portability.

Trilinos and Cactus thankfully also differ on a very important design decision. Cactus acts a the "main" routine of your code, it takes care of e.g. parallelism, I/O, check-pointing, parameter file parsing for you (if you want), while Trilinos does not. This makes them easy to combine, as there won't be any conflicts on which does what.

2. Description of Work

There are some technical issues, which must be addressed in order to compile Trilinos code in Cactus. The most pressing of these is to be able to make an executable. The first step in the process is to compile Trilinos libraries we wish to use in order to produce object files that Cactus can link to.

2.1. compilation of trilinos

Though the compilation, or builds, of Trilinos can be placed anywhere relative to the source code, the Trilinos user's manual recommends placing all your compilations, of Trilinos directly beneath the source directory, as show in Figure 1.[4] In all, the author created four configurations of Trilinos, differing in the inclusion of the example module, Didasko, and enablement of MPI. Didasko is a tutorial module containing a set of examples on how to write Trilinos code, and is not usually included in a build to Trilinos.

The creation of a build of Trilinos is accomplished by two commands; configure and make. Most configuration information, including feature options, and the list of modules to compile is given to the build system via the configure command. The Trilinos user's manual lists most, if not all options which Trilinos will accept. Having run the configure command, simply typing build in the root of the build directory will compile Trilinos.

Though this two-command system is ubiquitous in the Unix/Linux world, the number of options, which must be sent to configure, can be rather annoying. Most of the configurations the author created were created with a configure command three lines long, or about 375-400 characters. This problem is lessened in the make system of Cactus through the use of a rather complex make system. Once Trilinos completes its work, Cactus now has something to link to. Now Cactus must be told how to link to Trilinos.

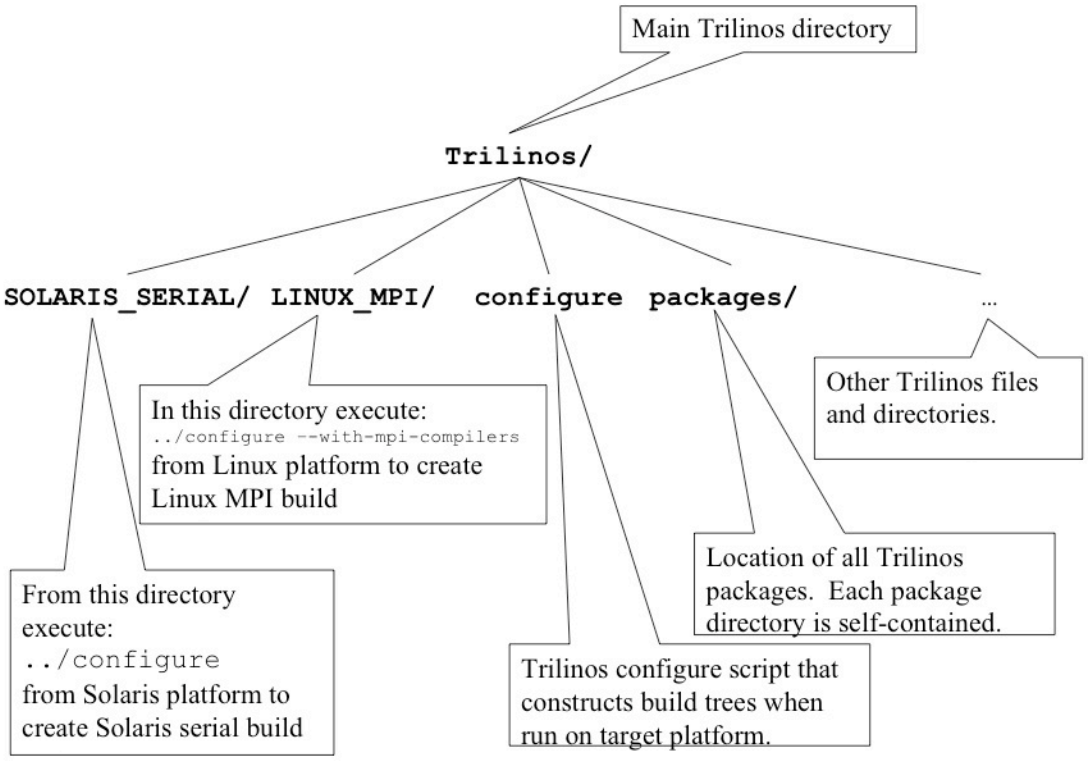

Figure 2 Recommended Layout for Trilinos Build Directories

2.2. inclusion of external libraries into cactus

As has been mentioned before, it is easy to extend the Cactus Code. The creation of a new thorn is accomplished via a single command: "make newthorn". This command should be executed from the root directory of the Cactus checkout. The command starts the execution of a Perl script, which prompts the user on information about the new thorn. The new thorn is then created in the chosen arrangement, in the arrangements directory. The choice of arrangement is chosen during the newthorn configuration by the developer. The basic Cactus checkout includes the arrangement CactusExternal. This arrangement is comprised of very simple thorns that tell Cactus how to link to external code. In this arrangement thorn named Trilinos was created.
A new thorn in Cactus contains three files with the .ccl (Cactus Configuration Language.) file extension, directories for source code, documentation and testing, and a README file. For this thorn all we need to touch is one of the auto-generated ccl files: interface.ccl. All thorns require at least one line in this file to be changed. This line tells the build system what this thorn does, by specifying a single string, which will represent what sort of feature this thorn implements. More than one thorn can implement the same thing. In fact, more than one thorn that implements the same thing can be compiled in the same configuration of Cactus. The thorn that is actually used is specified at runtime. The Trilinos thorn's interface.ccl file, surprisingly, is set to implement "Trilinos." Of the files, which the "make newthorn" system creates, this is the only one we have to edit. However, two more files must be created in order to setup Cactus properly with Trilinos. The first is another ccl file: configuration.ccl. In this file, you may specify a set of scripts

to run during compilation that can output to the make system information that it needs to compile. This script is the second file created. The second file can be any sort of executable script, in this case Python. This file, trilinos.py will be located in the root directory of the thorn. It is a rather short script that gives the Cactus build system two lists of directories and a list of libraries. One list contains all the directories with .h files in Trilinos, and the other includes all directories with library files. Finally, the third list tells Cactus the Trilinos libraries that it should link to at compile time.

Input to this file is given via environment variables. Two variables must be set. The first, TRILINOS_DIR, is the source code directory of Trilinos, and the second, TRILINOS_LIB_DIR, is the build directory you wish to use.

2.3. compilation of cactus

As described in the Cactus documentation, once the thorn is correctly configured and the environment variables are set, a new configuration of Cactus can be made. A compilation of Cactus is accomplished via running the same command twice.[5] "Make <configuration name>" will create a new configuration of Cactus. Options, such as a list of thorns to compile, and a file containing compile options can be included in the first command, but this is not optional. If a thorn list is not specified on the first command, when the command is run again, a list of all thorns in the checkout of cactus is created, and the user is asked to edit this list, in order to choose which thorns to compile. After the second command is executed, in the root directory of Cactus, the directory exe in the root of Cactus will hold an executable called cactus_<configuration name>. This is your actual executable.

Running this command requires a parameter file. This file contains any options that the thorns need, and most importantly, which thorns should be "active" during execution. Remember, more than one thorn, which implements the same functionality, can be compiled into Cactus. This line, in the parameter file, activethorns=<thorns> relieves the ambiguity. This file is given as a command line option to the Cactus executable. And with that, Cactus and Trilinos are combined!

3. Testing

In order to test that Cactus is working correctly with Trilinos, it is necessary to create yet another thorn, which uses functions and classes, found in Trilinos. In order to save time, an example program that was included in Trilinos was used as the base for the thorn. In order to insure that the thorn compiles correctly, it is necessary in the thorn's interface.ccl file to specify the necessity of including our Trilinos thorn. This is done with a single line: 'inherits: trilinos'. Also in this file goes a list of all header files from Trilinos required by the thorns code. The inherits line will insure that the Trilinos thorn must be included in order to compile the test thorn. A new configuration can then be made including both the Trilinos test thorn the Trilinos thorn, and the test thorn. Since performing these steps with the test thorn did not produce any errors, and ran perfectly, it can be safely assumed the integration is a success.

4. Further work

While this deceptively easy procedure is all that is required to make Trilinos and Cactus work together, this is not enough to make them work together well. At this point, "application thorns," to actually solve problems will be written which transfer data between the data structures of Cactus and Trilinos. A set of data transfer functions should be written to make this task less tedious. This set of functions, have been discussed and outlined; however development has just started on this functionality.

5. Acknowledgements

The author would like to express his appreciation to Yaakoub El Khamra, who without which this paper would have been impossible to complete. The author would also like to thank Kathy Traxler for her constant support, proofreading, and coordinating. Finally, the author is very grateful to Louisiana State University as well as the Computer Science department at LSU for the financial support.

6. References

1. The Cactus Code, "About Cactus," cactuscode.org, http://www.cactuscode.org/aboutCactus/

2. Yaakoub El Khamra, "Cactus Tutorial" cct.lsu.edu, http://cct.lsu.edu/~yye00/cactus_tutorial.pdf

3. Sandia National Laboratories, "About Trilinos", sandia.gov, http://software.sandia.gov/trilinos/about.html

4. Michael Heroux and James Willenbring, *Trilinos Users Guide* (New Mexico: Sandia National Laboratories, 2003), 12

5. "Cactus 4.0 Users' Guide," cactuscode.org, http://www.cactuscode.org/old/guides/stable/usersguide/usersguidestable.pdf