CHAPTER 2

Getting Started

While the ultimate goal in utilizing CFL3D will be to solve problems pertinent to the user's work, it is highly recommended for new users to begin by running some of the test cases described in Chapter 9. Experience gained while running these will help the user when running a "real" problem. The following sections provide general instructions for compiling and running CFL3D.

2.1 Acquiring the Code and Example Files

The files needed to run CFL3D are located on Vonneumann and Eagle, two Cray supercomputers located at NASA Ames Research Center. The files are in

```
~rumsey/Cfl3dv5/cfl3dv5.ascii
```

The files are tarred, compressed, encrypted, and uuencoded. A keyword (obtained from the Aerodynamic and Acoustic Methods Branch at NASA Langley – see below) must be used to decode the file `cfl3dv5.ascii`. The steps for this procedure are as follows:

Step 1

```
uudecode cfl3dv5.ascii
```

Step 2

```
crypt keyword < cfl3dv5.crypt > cfl3dv5.tar.Z
```

Step 3

```
uncompress cfl3dv5.tar.Z
```

Step 4

```
tar xvf cfl3dv5.tar
```

This step should create a directory named `Cfl3dv5` with the appropriate CFL3D files in it.

Step 5

```
rm cfl3dv5.ascii cfl3dv5.tar cfl3dv5.crypt
```

If the test cases are not required, a more convenient option might be to obtain the file `cfl3dv5.codeonly.ascii`. This file is also tarred, compressed, encrypted, and uuencoded, but it is much smaller than `cfl3dv5.ascii`.

If access to Vonneumann or Eagle is not possible, the encrypted codes are also available on the Langley CFD anonymous ftp site. To obtain the code in this manner, take the following steps:

Step 1

```
ftp tabdemo.larc.nasa.gov
```

or

```
ftp 128.155.24.42
```

Step 2

When prompted for the username, type

```
anonymous
```

Step 3

Type the user's e-mail address as the password.

Step 4

```
cd incoming
```

Step 5

```
cd Rumsey
```

Step 6

```
get cfl3dv5.ascii
```

or

```
get cfl3dv5.codeonly.ascii
```

To obtain the current keyword needed to decode the source files, send mail or e-mail to Dr. Christopher L. Rumsey (see page 4) requesting the code, along with an explanation describing the planned usage of the code. Include a FAX number with the request. All users will be asked to fill out and sign a CFL3D Usage Agreement form. After the completed form is received, the keyword will be provided.

After obtaining CFL3D, it is required that the user remember not to distribute any part of the code to others outside of his or her own working group without prior permission

from NASA Langley. The CFL3D code is currently restricted to use within the United States only.

2.1.1 The Code and Supplementary Files

The items listed below are made available to the user when CFL3D is obtained. The names in **bold** face indicate a directory.

2dtestcases	isrcheq_wkstn.f	makeprecfl3d_hp_sngl
3dtestcases	lbcx.f	makeprecfl3d_rs6000
Advice	makecfl3d_cray	makeprecfl3d_sgi
Maggie	makecfl3d_decalpha	makeprecfl3d_sgi_R10000
Multitask_cray	makecfl3d_hp_dbl	makeprecfl3d_sgi_R8000
README	makecfl3d_hp_sngl	makeprecfl3d_sun
Ronnie	makecfl3d_rs6000	plot3d_hp_dbl.f
Tools	makecfl3d_sgi	precfl.h
af3f.f	makecfl3d_sgi_R10000	precfl3d.f
bc.f	makecfl3d_sgi_R8000	rhs.f
cbsem.f	makecfl3d_sun	second_hp_dbl.f
cvmgt_wkstn.f	makeprecfl3d_cray	second_rs6000.f
dynptch.f	makeprecfl3d_decalpha	second_wkstn.f
input.doc	makeprecfl3d_hp_dbl	turbs.f

The CFL3D code consists of one main driver subroutine package, `cbsem.f`, and six sets of library subroutines, `bc.f`, `rhs.f`, `af3f.f`, `lbcx.f`, `turbs.f`, and `dynptch.f`. `Cbsem.f` contains the main subroutine (`mgblk`) of the program as well as many of the routines needed for the input and output of information. The file `bc.f` contains the boundary condition subroutines, including physical boundary conditions, 1-1 blocking, and grid embedding. It also contains the necessary routines for processing supplemental overlapped-grid and static patched-grid interpolation information. `Rhs.f` contains the subroutines needed for solving the right-hand side of the governing equations, while `af3f.f` contains the subroutines needed for calculating the left-hand side of the governing equations. (See Appendix A.) `Lbcx.f` contains a variety of subroutines, including those needed for multigrid and mesh sequencing, those providing the metrics and other grid information, those involved with computing the forces, and several others. The file `turbs.f` contains the turbulence models subroutines. Finally, `dynptch.f` contains the subroutines needed for dynamic grid patching.

Some supplementary files are also needed for CFL3D. A “makefile” is used to compile and link subroutines and to create an executable for the code. A preprocessing program called `precfl3d.f` is used to determine the memory and array dimensions needed by CFL3D for a particular case. For `precfl3d.f`, use `makeprecfl3d_`*`machinename`*. Here, and in the discussions that follow, the terms in italics pertain to user-specific items. For

instance, *machinename* might be cray, sgi, or sun for the Cray supercomputer, SGI, and Sun workstations, respectively. Precfl.h is the parameter file required by this makefile. The array dimensions set in precfl.h should be large enough for nearly any problem and should rarely need modification; if a larger value of a particular parameter is required, precfl3d will stop and request the user to increase the relevant parameter.

A makefile called makecfl3d_*machinename* is also needed to compile and link the CFL3D subroutines and create the executable, cfl3d. Several files are needed to compile the code on workstations. These files contain Fortran source code for the corresponding Cray intrinsic functions. For instance, when utilizing makecfl3d_rs6000, make sure that second_rs6000.f is available. Likewise, when using makecfl3d_hp_dbl, make sure that plot3d_hp_dbl.f and second_hp_dbl.f are in the working directory. Note that for double precision on the HP, the user must actually *remove* the subroutines plot3d, plot3c, and plot3t from cbsem.f prior to compiling with makecfl3d_hp_dbl. For all other makecfl3d makefiles, cvmgt_wkstn.f, isrcheq_wkstn.f, and second_wkstn.f are utilized and therefore must be available. (Note that wkstn is short for "workstation".)

The file input.doc contains the information found in Chapter 3. It is a listing and description of the input parameters used in CFL3D.

The files in directory **2dtestcases** are:

0012_patch	**Flatplate**	**Rae10**
0012_xmera	**Multielem**	**Rotorstator**
README.2d	**Multistream**	**Vibrate**

The files in directory **3dtestcases** are:

README.3d	**Delta**	**Oneram6**
Axibump	**F5wing**	

Directory **2dtestcases** contains various two-dimensional test cases. A patching example involving the NACA 0012 airfoil is located in **0012_patch**. In **0012_xmera**, a case utilizing grid overlapping for the same airfoil is found. A three-element airfoil case involving grid overlapping is set up in **Multielem**. Directory **Rae10** contains a single block case for the RAE 2822 airfoil. **Flatplate** contains the files needed to run the flat plate test case. A multistream nozzle case is in **Multistream**. In **Vibrate** are the files for a vibrating plate case. A rotor-stator case is set up in **Rotor**.

The three-dimensional test cases are located in the directory **3dtestcases**. A case solving the flow over an axisymmetric bump is in **Axibump**. Other cases currently available are for a delta wing (in **Delta**), an F-5 wing (in **F5wing**), and an Onera M-6 wing (in **Oneram6**). The steps for running these test cases are described in Chapter 9.

The files in directory **Advice** are:

cfl3dadvice.give	cfl3d.references	v4_to_v5.inputdif
cfl3dadvice.grid	cfl3d.turb.references	

The first four files contain the same information given in Chapter 10 and in "CFL3D Papers" on page 337. The v4_to_v5.inputdif file describes the differences between the Version 4.1 and Version 5.0 input files. (This information is also given in Appendix K.)

The files in directory **Maggie** are:

README	maggie.doc	makemaggie_sgi
ismax_wkstn.f	maggie.f	makemaggie_sun
ismin_wkstn.f	makemaggie_cray	second_wkstn.f
mag1.h	makemaggie_hp_sngl	

The files in the **Maggie** directory are needed for cases involving grid overlapping. The chimera scheme code is called maggie.f. Maggie.doc explains the input parameters for maggie.f. Makefiles for various machines are available and require mag1.h, which contains the appropriate dimensions for the problem at hand. Make sure ismax_wkstn.f, ismin_wkstn.f, and second_wkstn.f are available before running MaGGiE on a workstation.

The files in **Multitask_cray** are:

README	makecfl3d_cray_multi	xlim_multi.f
fhat_multi.f	tinvr_multi.f	

These files should be used in conjunction with the standard CFL3D files when Cray multitasking is desired.

The files in directory **Ronnie** are:

README	makeronnie_sgi	ronnie.f
makeronnie_cray	makeronnie_sun	second_wkstn.f
makeronnie_decalpha	ron1.h	
makeronnie_hp_sngl	ronnie.doc	

When utilizing grid patching, the files in the **Ronnie** directory are needed. The code that sets up the patching interpolation stencils is called ronnie.f. Ronnie.doc explains the input parameters for ronnie.f. Makefiles for various machines are available and utilize ron1.h in the compilation. The user must set the parameters in ron1.h for each particular case. Also, make sure second_wkstn.f is available before running ronnie on a workstation.

The files in the **`Tools`** directory are:

```
README                  make1to1.f              reverseijk.f
everyother.f            make_p3dtotec3d_cray    v4tov5_input.f
everyotherp3d.f         p3dtotec3d.f            v4tov5_restart.f
historytec3d.f          p3dtotec3d_pre.c        v5inpdoubhalf.f
```

These programs are useful for converting files from one format to another. The code `everyother.f` takes an existing CFL3D-type grid and coarsens it by taking every other grid point. The `everyotherp3d.f` program performs the same task for PLOT3D-type grids. The file `historytec3d.f` reads in the residual history file and creates a corresponding Tecplot[4] file. It will also convert iterations to work units if desired. The `make1to1.f` program assists in the creation of the boundary condition and one-to-one blocking sections of the input file. The `p3dtotec3d.f` tool (along with `p3dtotec3d_pre.c` and `make_p3dtotec3d_cray`) creates a Tecplot file from PLOT3D grid and solution files. The `reverseijk.f` code reads a CFL3D-type grid file, swaps indices (as desired), and writes out a new CFL3D-type grid file. The `v4tov5_input.f` tool takes a Version 4.1 input file and converts it to Version 5.0 form. The `v4tov5_restart.f` tool converts a Version 4.1 restart file to Version 5.0 form. Finally, `v5inpdoubhalf.f` takes an existing input file and creates a new input file with either double or half the grid points of the original.

2.2 Obtaining a Grid File

The first step in solving a CFD problem is obtaining a grid. A pre-packaged tool such as GRIDGEN[36] could be used to create a grid, or perhaps a user-written program could be used for relatively simple configurations. How ever the grid is obtained, an essential step toward success with CFL3D will be having the grid file written in CFL3D or PLOT3D/TLNS3D format. See “Grid File” on page 65 for the appropriate formats for the file. Time can also be saved at this point by considering what options will be desirable later. For example, since the use of multigrid is highly recommended, choose grid dimensions wisely. See “A Word About Grid Dimensions” on page 127. If viscous cases will be run, make sure the grid spacing normal to the wall in the boundary layer is fine enough. At least 20 grid points are recommended in a *laminar* boundary layer. At least 3 points are recommended in the laminar sub-layer of a *turbulent* boundary layer (y^+ of first grid point off the wall should be $O(1)$). Basically, consider how the problem will be solved and choose dimensions, blocking strategies, grid-line stretching, etc., accordingly.

2.3 Creating the Input File

Chapter 3 lists and describes all the parameters in the input file. The easiest way to create an input file for a particular case is to copy the input file from the test case (see

Chapter 9) most resembling the case at hand. Then change those parameters that pertain to the current case. The other parameters should already be set to the recommended values. Besides the test case sample inputs, some input examples for 1-1 blocking and grid embedding are provided in Chapter 6 and several input examples for multigrid, mesh sequencing, and grid embedding are provided in Chapter 7. CFL3D provides additional help in setting up the input file with an extensive set of diagnostics which halt execution at detectable errors and provide the user with a message indicating the problem.

2.4 Utilizing Patched and/or Overlapped Grids

When using patched and/or overlapped grids, the files containing the appropriate interpolation coefficients must be obtained prior to running CFL3D. The preprocessors ronnie and MaGGiE[11] will generate the needed coefficients for patched grids and overlapped grids, respectively, and will output the interpolation information in the format needed by CFL3D. Information on using ronnie and MaGGiE can be found in the `ronnie.doc` and `maggie.doc` files, respectively. The preprocessor `precfl3d`, which sets all the array size parameters needed by CFL3D, will need the appropriate patch and/or overlap data files in order to properly size the arrays needed for these options. Therefore, be sure to run ronnie and/or MaGGiE *before* running `precfl3d`.

2.4.1 Running ronnie

When utilizing the grid patching option of CFL3D, the preprocessor ronnie must be run first. The basic steps for running this code are as follows.

Step 1

Prepare an input deck, typically called `ronnie.inp`, for the case. Also, modify the header file `ron1.h` as needed.

Step 2

Use the makefile to compile, link, and create the executable for the ronnie code (be sure `ron1.h` is in the current directory):

```
make -f makeronnie_machinename
```

Step 3

Run the ronnie code:

```
ronnie < ronnie.inp
```

If successful, ronnie will create a file containing patch interpolation data. The name of the patch file is specified by the user in the input file; it is typically called `patch.bin`.

2.4.2 Running MaGGiE

If grid overlapping is used in some portion of the grid configuration, then the preprocessor MaGGiE must be utilized. The basic steps for running this code are as follows.

Step 1

Prepare an input deck, typically called `maggie.inp`, for the case. Also, modify the header file `mag1.h` as needed.

Step 2

Use the makefile to compile, link, and create the executable for the MaGGiE code (be sure `mag1.h` is in the current directory):

```
make -f makemaggie_machinename
```

Step 3

Run the MaGGiE code:

```
maggie < maggie.inp
```

If successful, MaGGiE will create a file containing overlap interpolation data. The name of the overlap file is specified by the user in the input file; it is typically called `ovrlp.bin`.

2.5 Running CFL3D

The basic steps for running CFL3D are as follows. Remember, if using grid patching and/or grid overlapping, run ronnie and/or MaGGiE before proceeding with the following steps.

Step 1

Compile, link, and create the executable for the `precfl3d.f` code. Make sure the parameter file `precfl.h` is available.

```
make -f makeprecfl3d_machinename
```

The resulting executable will be called `precfl3d`.

Step 2

Run `precfl3d`.

```
precfl3d < inputfilename
```

The *`inputfilename`* here is the CFL3D input file name. Detailed diagnostic information is printed out in the file `precfl3d.out`. Most items in the input deck will be checked for

consistency. However, since `precfl3d` does not read in the grid file, it cannot detect grid problems and, most importantly, it cannot detect ordering errors in the 1-1 block interfaces. Check the output from CFL3D itself to verify that 1-1 block interfaces are set correctly. If `precfl3d` has not run successfully, then the most likely cause is an error in the input file; `precfl3d` echoes the input file as it goes along, so check the bottom of `precfl3d.out` to determine the approximate location of the input error. A successful execution of `precfl3d` will have one of the following lines at the end of the output file and print to the screen:

```
you *MUST* recompile cfl3d
```

or

```
you do not need to recompile cfl3d
```

When the code has run successfully, there will be no further need for this program until a new input file is developed.

Running `precfl3d` will either create or modify the parameter files, `cfl1.h`, `cfl2.h`, `cfl3.h`, `cfl4.h`, and `cfl5.h`, which are used when compiling CFL3D. Remember that these files *are* case dependent. Therefore, another case will generally require that `precfl3d` be run again and CFL3D recompiled.

Step 3

Compile, link, and create the executable for CFL3D.

```
make -f makecfl3d_machinename
```

The executable will be called `cfl3d`.

Step 4

Run `cfl3d`. This step can be performed interactively or a submittal file can be utilized to send the case to a particular queue of the machine being used.

```
cfl3d < inputfilename
```

Users have been known to submit cases for large numbers of iterations on the first run only to discover that an input parameter was set incorrectly. It is wise to begin running CFL3D with only a few iterations and then check the output. Look over the main output file. Make sure there are no warning messages and that, in general, it “looks right” for the particular case being run. Take the PLOT3D files to a Silicon Graphics (or compatible) workstation and check any block boundaries to see if the flow is passing from block to block as expected. *Then* submit the case for a more extensive computation.