Chapter 2

ADCIRC Model - Overview, Compilation, and Execution

This chapter deals directly with the ADCIRC model, which is the underlying tidal hydraulics model used by the author. It gives a brief overview of the model's capabilities, which should be of interest and accessible to the generalist. Additionally, this chapter provides highly specific details on how to access, compile, and run ADCIRC. This information will be of interest only to individuals interested in running ADCIRC themselves. As will be discussed in detail later, ADCIRC can be run in a 'serial' fashion on a single computer or in a 'parallel' fashion on a cluster of computers. The details on parallel operation will necessarily be specific to the high-performance computing resources found at Penn State.

2.1 Model Overview

The source code for the ADCIRC model, along with a user's manual, theory report, an email listserv, and other resources are all available online at http://www.adcirc.org. The development of ADCIRC is generally attributed to Luettich & Westerink (1991). Since that time, many modifications and upgrades to the model have been made and ADCIRC presently enjoys very wide use among the academic community and federal agencies such as the Army Corps of Engineers, NOAA, and the Naval Research Laboratory.

2.1.1 Features and Capabilities

ADCIRC presently has the ability to operate in two-dimensional and three-dimensional barotropic (vertical density profile not resolved) mode. A three-dimensional baroclinic (vertical density profile resolved) is under development. In barotropic mode, the model solves for water elevation and water velocity.

The model uses an unstructured finite-element mesh to represent the domain. This approach is optimal for complex bathymetry and coastline boundaries as elements of varying size can be incorporated as needed (Fig. 2.1).

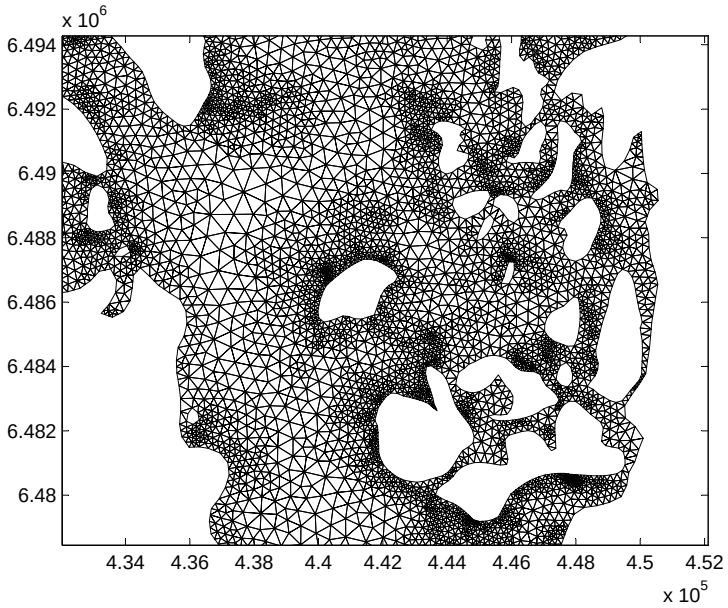

Figure 2.1: Finite element mesh of the region near the Beardslee Islands.

ADCIRC is a Fortran program which requires, at a bare minimum, two input files to run:

1. fort.14: this file describes the structure of the finite element mesh (see Chapter 3).

2. fort.15: this file is a parameter file that describes the particulars of the current ADCIRC run (see Chapter 4). This file is highly customizable

depending upon which features the user wishes to incorporate.

In addition to these files, there are many optional files that may or may not have to be present, depending upon which features the user wishes to incorporate into a particular run. For example, note that the ADCIRC model can be 'forced' by the following:

1. Gravity / tidal potential. These effects are dominant in most simulations. For a closed basin, such as one of the Great Lakes, tides will be forced due to the gravitational attraction on the water in the closed basin. For domains that have an 'open boundary,' tidal information must be specified on these open boundaries.

2. Meteorological conditions. One of the primary applications of ADCIRC is to storm surge predictions during large storm events. Users wishing to incorporate meteorological forcing will need to develop a fort.22 file (see Chapter 6).

3. Freshwater inflows. ADCIRC has the ability to incorporate freshwater inputs, i.e. rivers, into its simulations. Users wishing to models domains with river inflows will need to develop a fort.20 file (see Chapter 5).

One additional 'optional' input file is associated with the transport of passive scalars. Users wishing to study the dispersal of such tracers in the water column will need to provide a fort.10 file, which simply describes the initial concentration field in the domain. This feature may be of particular interest to biologists interested in the dispersal of the larvae of any number of species.

2.1.2 Model Output

Regarding the output of the model, ADCIRC provides for great flexibility. For example, the user can specify that water surface elevations and velocities be written only at selected points in the domain (fort.61 and fort.62 files, respectively). Alternatively, output at *all* points in the domain can be requested (fort.63 and for.64, respectively, for elevation and velocity output).

An additional feature of ADCIRC is that it has the ability to perform harmonic analysis of the elevation and velocity fields. In other words, ADCIRC, following a sufficiently long run, is able to determine the tidal constituents (amplitudes and phases) at either selected points in the domain (fort.51 and

fort.52, respectively, for elevation and velocity) or at all points in the domain (fort.53 and fort.54, respectively, for elevation and velocity). These files provide valuable information about the tides and currents at all points in a model domain. Presently, knowledge of this information is limited to very sparse NOAA tidal stations or tidal databases such as the Eastern Pacific Tidal Database (http://www.unc.edu/ims/ccats/tides/tides.htm), which does not cover Glacier Bay proper.

2.2 Model Compilation

In this section, specific instructions on compiling the ADCIRC source code (Fortran) into an executable are provided. When the source code is downloaded and unzipped, a file folder structure as shown in Fig. 2.2 is obtained.

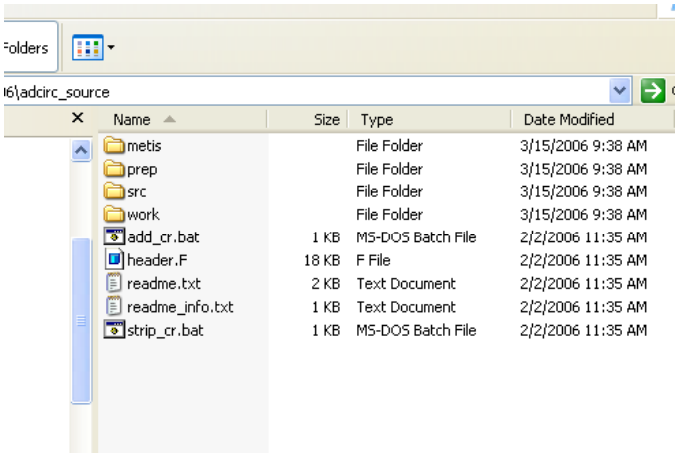

Figure 2.2: File folder structure obtained after unzipping ADCIRC source code.

2.2.1 Serial Model

Compilation of the source code is a platform dependent option. The following describes the steps required to compile a serial version of ADCIRC on the Linux clusters available at Penn State. These resources are described in detail at http://gears.aset.psu.edu/hpc/index.shtml. The web pages therein de-

scribe the different clusters, how to obtain an account, and how to access this account using the secure shell client (http://css.its.psu.edu/internet/ssh/).

With an active and connected account on one of the clusters (e.g. lionxo.aset.psu.edu), it is a simple matter to use the ssh client (Fig. 2.3) to upload the ADCIRC source files to the account.

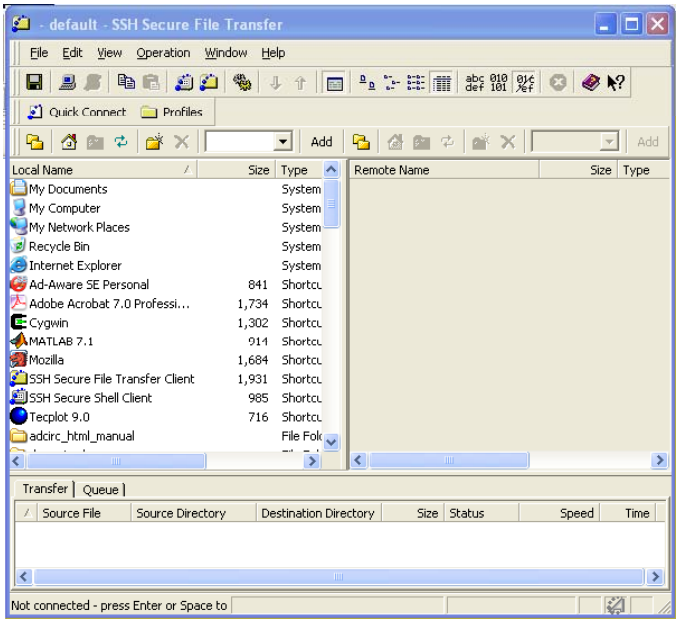

Figure 2.3: Graphical user interface of the ssh client.

Next, using a terminal window, change directories to the *work* folder. Third, before compiling, the permissions on the file *config.guess* must be changed with the command

```
>> chmod +x config.guess
```

Finally, issuing the command

```
>> make adcirc
```

will result in the creation of the executable file *adcirc*. At this point, the model is ready to run and can be executed with the simple command

```
./adcirc
```

2.2.2 Parallel Model

The ADCIRC model source code has been parallelized to run on interconnected processors. The obvious advantage of this is decreased run times. If a run can be spread out over 8 processors, it should run approximately 8 times faster. Given the large computational grid (∼80,000 elements) and the small time step (∼1 second) for the present application to Glacier Bay, this is a very real consideration.

To get ADCIRC to compile, in parallel form, on the PSU Linux cluster, change directories to the *work* folder and open the file *cmplrflags.mk*. Line 9 must be changed from the existing

```
PFC       :=  mpif90
```

to the following

```
PFC       :=  mpif90 -f90=pgf90 -config=pgf90
```

Once this is done, issuing the command

```
>> make all
```

will result in the creation of five executable files: *adcprep*, *adcprep2*, *adcirc*, *padcirc*, and *adcpost*. For a parallel run, described in the next section, the 1st, 4th, and 5th of these executables will be used.

2.3 Model Execution

Once the model has been compiled, there a variety of ways in which it can be executed. The steps for both serial and parallel runs are described below.

2.3.1 Serial Model

The simplest way to run the model is to issue the command

```
>> ./adcirc
```

from the directory in which the executable file resides. The input files fort.14 and fort.15 (and others, depending upon the run) must also be present. This approach is slightly inconvenient if one has many directories containing many different input files for simulation. In this case, the executable would have to be copied into each of these folders.

A slightly different approach is to store the executable file *adcirc* in a single location, say in the directory */adcirc_files/* created in your home directory. Then, if this location is added to the *path* variable, the executable can be accessed from any directory.

In Linux, the present path can be checked by issuing the command:

```
>> echo $PATH
```

A permanent change to the path is made by opening up the file .bash_profile, found in your home directory. There, you will find a line something like

```
PATH=$PATH:$HOME/bin
```

This line should be changed to

```
PATH=$PATH:$HOME/bin:/home2/dfh4/adcirc_files
```

where /home2/dfh4/ represents (in this case mine) the user's home directory. The next time you log in, reissue the echo $PATH command and you should see the updated path.

You can now run the *adcirc* executable from any folder simply by issuing the command

```
>> adcirc
```

As before, the necessary input files must be in the directory from which the command is issued.

Batch Runs

While ADCIRC will run successfully as described above, this 'command line' operation is only of limited use on the PSU Linux clusters. This is because jobs started in this way run on the 'login' node and are limited to only one hour of run time. As alluded to above, realistic (i.e. days to weeks long) simulations with ADCIRC will require much more than one hour of computer time.

To execute longer runs, jobs must be submitted in 'batch mode.' This is done by preparing a small script file which is then sent in to start the job. For example, the following:

```
#PBS -l nodes=1:ppn=1
#PBS -l walltime=23:00:00
#PBS -j oe
cd $PBS_O_WORKDIR
echo " "
echo "started on `hostname` at `date`"
adcirc
echo " "
echo "ended at `date`"
echo " "
```

could be saved as a file called *myjob*. The command

```
>> qsub myjob
```

submits this job to be placed in the queue. A few notes:

1. The walltime places an upper bound on the computer time that will be allocated to the job. It is therefore important that one estimate the time required in order to prevent an early program termination.

2. The nodes / ppn line, as written, requests one processor. In serial form, this is all that can be utilized.

3. Finally, the *adcirc* syntax (no preceding ./) is assuming that the path variable has been changed, as described in the previous section.

The status of your request can be reviewed at any time with the command

```
>> qstat -u dfh4
```

where, in this case, dfh4 is my account user id.

2.3.2 Parallel Model

As mentioned above, ADCIRC can run in parallel fashion, thereby greatly reducing computational times. A parallel run has three main steps. The discussion below will assume that all of the relevant executables are in a location that is in the path.

1. *adcprep* - issuing this command at the prompt the user to specify the number of processors to be utilized (say 4, 8, 16, etc.). When this program is completed, the user will note that a number (the same number entered by the user) of folders will have been created.

2. *padcirc* - while this parallel version of the ADCIRC executable can be run from the command line, it is not recommended, due to the time limits discussed above. As a result, a script file should be prepared and submitted with the qsub command. The following is an example script file:

   ```
   #PBS -l nodes=4:ppn=2
   #PBS -l walltime=8:00:00
   #PBS -j oe

   cd $PBS_O_WORKDIR
   echo "Job started at ‘date‘"
   /usr/global/bin/icmpirun padcirc
   echo "Job ended at ‘date‘"
   ```

   In this script, the first line requests 4 nodes and 2 processors per node, for a total of 8 processors. This total number *must* match with the number that was specified during *adcprep*.

3. *adcpost* - when the run is complete, the user will find 'partial' input and output files in the folders that were created during the *adcprep* run. Typing *adcpost* at the command prompt will step the user through the process of recombining these individual files into global output files.

   As an aside, the author has found that the combined file is much greater in size than the sum of the individual files. Some investigation revealed that this is due to the fact that, during recombination, 'trailing zeros' are added to each line. This does not pose any problems, in terms of post-processing / visualization of the output, but it can lead to unreasonably large file sizes.

   Discussions with other researchers have led to the following 'fix' for this problem. In the file post.f, which is found in the folder *prep* in the source code distribution, one will find lines of code like:

```
 WRITE(xx,80) OUTMSG
```

where the xx is 63, 64, or some other output file designation. If this line of code is changed to

```
WRITE(xx,*) TRIM(OUTMSG)
```

and the *adcpost* code is recompiled, the trailing zeros will be eliminated.